Résolutions numériques

Les méthodes de résolutions numériques sont extrêmement présentes dans la recherche comme dans l'industrie. Elles permettent parfois un gain de temps, une automatisation, ou une approximation d'une solution introuvable analytiquement.

1 Résolution de problèmes du type $f(x) = 0$

1.1 Problématique

Soit f une fonction continue sur un intervalle $[a, b]$ de $\mathbb{R}$.

Nous cherchons $\alpha \in [a, b]$ tel que $f(\alpha) = 0$. Les méthodes sont en générales itératives. On cherche à construire une suite $(x_n)_{n\geq 0}$ telle que $\lim_{n\to\infty} x_n = \alpha$.

Remarque : On considère que l'intervalle est suffisamment petit pour que la racine de α soit unique.

1.2 Critères d'arrêt

Soit ϵ une tolérance fixée. On utilise :

- Soit un critère basé sur l'incrément :
  - Absolu $|x_{n+1} - x_n| < \epsilon$
  - Relatif $\left|\frac{x_{n+1}-x_n}{x_{n+1}}\right| < \epsilon$
- Soit un critère basé sur le résidu :
  - $|f(x_n)| < \epsilon$

Remarque : Il est préférable de limiter aussi le nombre d'itération au cas où aucun des critères d'arrêt ne se vérifie.

1.3 Les méthodes

1.3.1 La méthode de la dichotomie

Soit la fonction continue f: $[a, b] \to \mathbb{R}$.

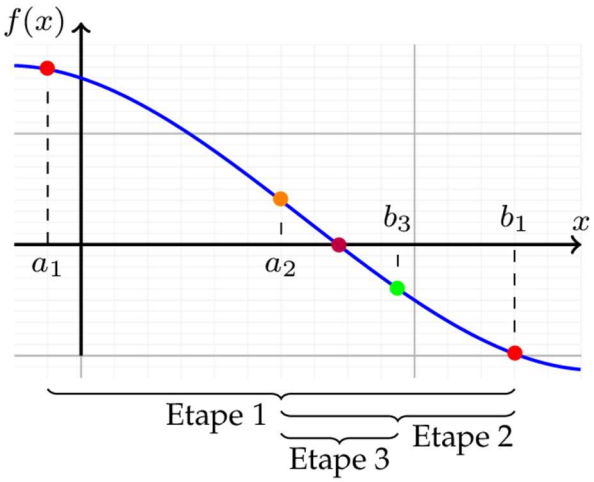
f prend toutes les valeurs intermédiaires entre $f(a)$ et $f(b)$. En particulier, si f est telle que

$f(a)f(b) < 0$ (c'est-à-dire $f(a)$ et $f(b)$ ne sont pas de même signe), alors il existe $\alpha \in]a, b[$ tel que $f(\alpha) = 0$.

Pour résoudre l'équation $f(x) = 0$, la méthode de la dichotomie consiste à :

- Calculer le point milieu m de l'intervalle $[a, b]$;
- Évaluer le signe de $f(a)f(m)$;
- Choisir le sous-intervalle $[a, m]$ ou $[m, b]$ dans lequel se trouve le zéro ;
- On réitère le procédé tant que le nouvel intervalle est plus grand que la tolérance.

Remarque : La méthode de dichotomie fait partie des méthodes « diviser pour régner » dont le principe est de résoudre un problème de taille importante en diminuant sa taille/complexité jusqu'à vérifier le critère d'arrêt.



Q.1. Compléter le code Python suivant afin de réaliser la méthode de dichotomie.

```
def dichotomie(f,debut,fin,tolerance):
    delta = abs(debut - fin)
    while delta > tolerance:
        milieu = (debut + fin) / 2
        delta = abs(debut - fin)
        if f(milieu) == 0:
            return milieu
        elif f(debut) * f(milieu) > 0:
            debut = milieu
        else:
            fin = milieu
    return (debut+fin)/2
```

Q.2. Quel est son critère d'arrêt ?

L'arrêt se fait quand l'intervalle est plus petit que la tolérance, il s'agit d'un critère d'arrêt basé sur l'incrément absolu.

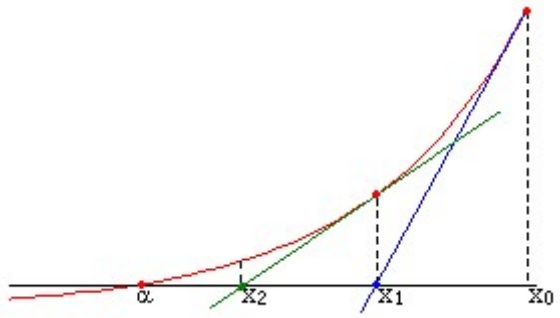
1.3.2 La méthode de Newton

La méthode de dichotomie est robuste. Elle est souvent utilisée pour localiser une racine de l'équation $f(x) = 0$. Cependant sa vitesse de convergence est plus lente que la méthode de Newton.

Supposons que f soit dérivable sur l'intervalle $[\boldsymbol{a}, \boldsymbol{b}]$ et f' connue.

Pour résoudre l'équation $f(x) = 0$, la méthode de Newton consiste à :

- Choisir une valeur $x_0 \in [\boldsymbol{a}, \boldsymbol{b}]$, plus cette valeur est proche de la solution recherchée, plus la méthode sera efficace ;
- Construire le nouvel itéré x_1 à l'intersection entre l'axe des abscisses et la tangente à la courbe décrite par $f(x)$ passant par le point $(x_0, f(x_0))$;
- Réitérer ce processus tant que le critère d'arrêt n'est pas atteint : $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ avec $n \geq 0$.



Q.3. Compléter le code Python suivant pour réaliser la méthode de Newton.

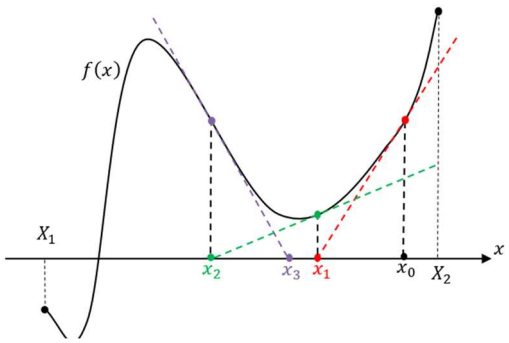
```
def f(x):
    return x**2 - 4 # l'équation est x^2 - 4 = 0, donc x = 2 ou x = -2

def df(x):
    return 2 * x

def methode_newton(f, df, x0, tol=1e-6, max_iter=100):
    for i in range(max_iter):
        x0 = x0 - f(x0) / df(x0) # Formule d'itération de Newton
        if abs(f(x0)) < tol:
            return x0
    print('Max itération atteint, la solution actuelle est :', x0)
```

Remarques :

- La valeur de la dérivée $f'(x)$ ne doit pas être nulle. En pratique, elle ne doit pas non plus être proche de 0 car l'intersection entre la tangente et l'axe des abscisses peut alors sortir de l'intervalle $[a, b]$.
- Selon le choix du point initial x_0, la méthode peut échouer.
- La méthode de Newton peut souffrir d'instabilité et de non-convergence lorsque la fonction f présente des oscillations, voire des valeurs bruitées (exemple de données physiques acquises sur un système réel).



- Comme la dichotomie, la méthode de Newton étant itérative, elle est souvent implémentée par des fonctions récursives.

2 Résolution numérique d'un système d'équations linéaires

2.1 Problématique

On considère un système d'équations linéaires de n équations à n inconnues. On peut écrire ce système par son écriture matricielle $AX = B$, où :

- A est une matrice carrée de taille n de coefficients réels ;
- X un vecteur colonne de taille n des inconnues réelles ;
- B un vecteur colonne de taille n du second membre de l'équation à valeur réelles.

Nous supposons que la matrice A est inversible, le système admet donc une unique solution.

2.1.1 La méthode du pivot de Gauss ou Gauss-Jordan

Il s'agit de la méthode vue pendant les cours de mathématique de première année qui vise à obtenir une matrice triangulaire en effectuant des opérations élémentaires sur les lignes (et colonnes) des matrices.

Cette méthode peut tout à fait se réaliser numériquement étape par étape. Cependant il est beaucoup plus simple (et conseillé par le programme) d'utiliser une méthode d'une bibliothèque comme numpy :

```
import numpy as np
x,y,z = np.linalg.solve(K,B)
```

Il faut bien sûr avoir défini `K` et `B` telles que $K\begin{bmatrix}x\\y\\z\end{bmatrix} = B$.

Q.4. Mettre le système suivant sous la forme d'une équation matricielle.

$$\begin{cases}-2x+4y+z=-18\\8x+2y-z=6\\2x-y+2z=27\end{cases} \Leftrightarrow \begin{bmatrix}-2 & 4 & 1\\8 & 2 & -1\\2 & -1 & 2\end{bmatrix}\begin{bmatrix}x\\y\\z\end{bmatrix} = \begin{bmatrix}-18\\6\\27\end{bmatrix}$$

Q.5. Compléter le code Python suivant pour résoudre l'équation matricielle.

```
import numpy as np

K = np.array([[-2, 4, 1], [8, 2, -1], [2, -1, 2]])
B = np.array([-18, 6, 27])
# Résolution de l'équation linéaire KX = B
x,y,z=np.linalg.solve(K,B)
```

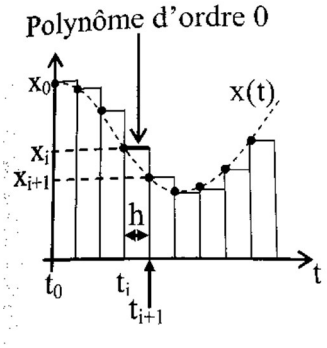
3 Intégration et dérivation numérique

3.1 Échantillonnage

Les grandeurs physiques sont échantillonnées avec un pas de temps h en général régulier. Pour une étude compris entre les temps t_0 et t_n, l'intervalle de temps est donc divisé en n segments de longueur $h = \frac{t_n - t_0}{n}$. Le temps s'exprime alors par $t_k = t_0 + k\,.h$ avec $k \in [0, n]$. On note $x_k = x(t_k)$ la valeur échantillonnée de la grandeur physique x au temps t_k.

3.2 Approximation de fonction

Pour effectuer une intégration ou une dérivation numérique, nous ne disposons que d'un nombre fini de valeur. Plutôt que de rechercher une unique fonction d'interpolation qui passe au mieux par toutes les valeurs, il est préférable d'utiliser des fonctions d'interpolation continues par morceaux et simples (polynômes d'ordre 0, 1 ou 2).



3.3 Dérivation numérique – différences finies

3.3.1 Différences finies progressives (avant)

La dérivation numérique est une approximation liée au développement de Taylor à l'ordre 1 :

$$f(x+h) = f(x) + hf'(x) + o(h)$$

Soit

$$f(x+h) \approx f(x) + hf'(x) \Leftrightarrow f'(x) = \frac{f(x+h) - f(x)}{h}$$

Soit en discret

$$\frac{dx(t)}{dt} \approx \frac{x_{k+1} - x_k}{h}$$

Il est nécessaire d'utiliser la méthode des différences finies avants pour déterminer la dérivée au premier point ; ce qui est utile par exemple pour déterminer si la pente à l'origine d'un signal est nulle ou non nulle, caractérisant un système d'ordre 1 ou 2.

3.3.2 Différences finies rétrogrades (arrière)

Il existe aussi la méthode des différences finies rétrogrades en utilisant le développement de Taylor avec $f(x-h)$, on trouve :

$$\frac{dx(t)}{dt} \approx \frac{x_k - x_{k-1}}{h}$$

3.3.3 Différences finies centrées

En soustrayant les deux développements de Taylor précédents on peut définir :

$$\frac{dx(t)}{dt} \approx \frac{x_{k+1} - x_{k-1}}{2h}$$

Cette méthode est plus précise mais ne permet pas de déterminer la dérivée ni au premier ni au dernier point.

Remarque : Il est possible de déterminer les dérivées d'ordres supérieurs sur le même principe en réalisant des développements de Taylor aux ordres supérieurs.

3.3.4 Implémentations Python

Q.6. Compléter le code Python suivant implémentant les méthodes des différences finies avant et arrière.

```
def f(x):
    """
    Fonction d'exemple pour laquelle nous cherchons sa dérivée première.
    """
    return x**2 - 4 # La dérivée exacte est 2*x


def approx_derivative_avant(f, x, h=1e-5):
    """
    Approximation de la dérivée de f en x en utilisant la méthode des
    différences finies progressives.
    """
    return (f(x + h) - f(x )) / h


def approx_derivative_arriere(f, x, h=1e-5):
    """
    Approximation de la dérivée de f en x en utilisant la méthode des
    différences finies rétrogrades.
    """
    return (f(x) - f(x - h)) / h
```

Q.7. Déterminer une approximation de la sortie du code suivant.

```
def f(x):
    return np.sin(x) # La dérivée exacte est cos(x)


def approx_derivative_centree(f, x, h=1e-5):
    return (f(x + h) - f(x - h)) / (2 * h)


print(approx_derivative_centree(f,np.pi))
```

La dérivée de sin est cos, cos(pi) donne -1, aux erreurs d'approximation près.

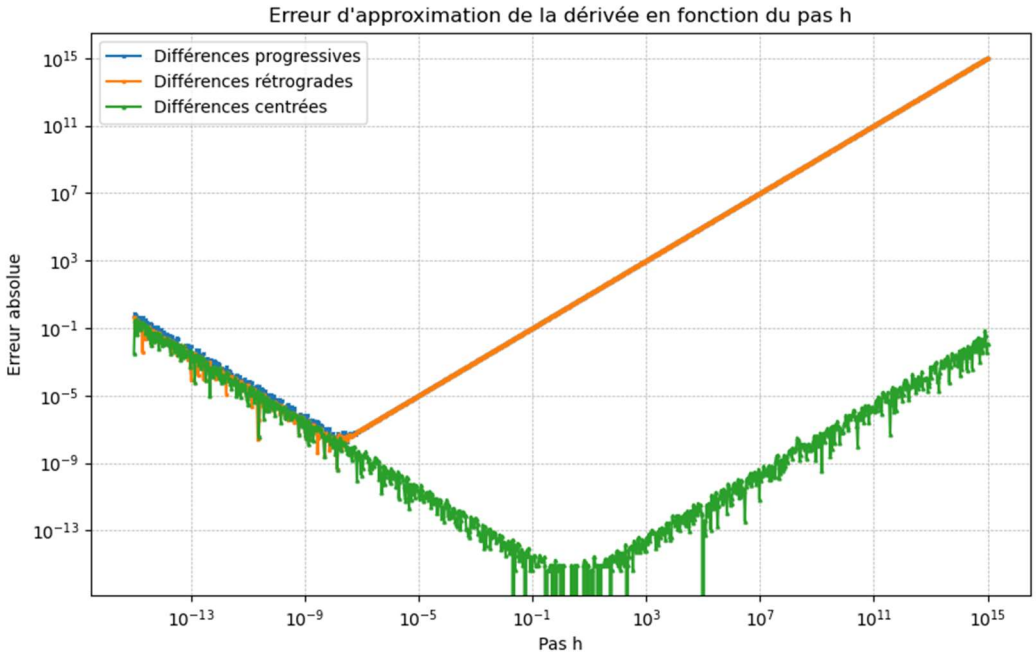
3.3.5 Erreurs

On remarque sur le graphique suivant que les erreurs pour les méthodes rétrogrades et progressives sont similaires et admettent un minimum.

On distingue deux types d'erreurs : celles liées à l'approximation du développement de Taylor, les erreurs de troncature ; et celles liées à la précision de la machine, les erreurs d'arrondi.

On remarque qu'après le minimum, les erreurs d'arrondi diminuent ce qui rend l'erreur directement linéaire avec le pas h (erreur de troncature).

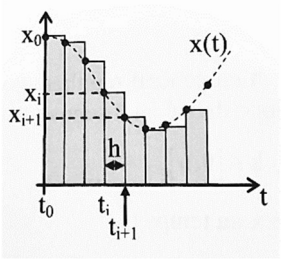
Le minima est atteint pour un pas h plus grand pour la méthode centrée.



3.4 Intégration numérique

L'intégration numérique cherche à approximer l'intégrale $I = \int_0^{t_n} x(t)dt$, pour cela on calcule l'aire sous la courbe.

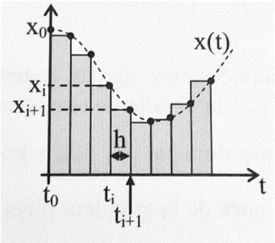
3.4.1 Méthode des rectangles à gauche



L'aire sous la courbe peut être approximée par :

$$I = \sum_{i=0}^{n-1} h.x_i = h\sum_{i=0}^{n-1} x_i$$

3.4.2 Méthode des rectangles à droite



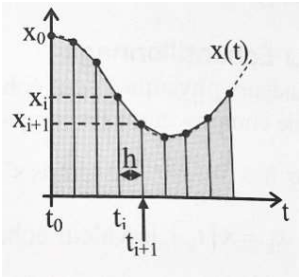
Cela revient à commencer par la fin, on trouve :

$$I = \sum_{i=1}^{n} h.x_i = h\sum_{i=1}^{n} x_i$$

3.4.3 Méthode des trapèzes (rectangles centrés)

Cette méthode évite les surestimations et sous-estimation des rectangles à gauche et à droite en calculant l'aire de trapèzes. Cela revient à calculer l'aire d'un rectangle centré sur le point moyen :

$$I = \sum_{i=0}^{n-1} h\frac{x_i + x_{i+1}}{2} = \frac{h}{2}\sum_{i=0}^{n-1} x_i + x_{i+1}$$



3.4.4 Implémentations Python

Q.8. Compléter le code suivant de sorte que la fonction `integration_rectangles_droite` renvoie l'intégrale approximée par la méthode des rectangles à droite.

```
def f(x):
    """
    Fonction d'exemple à intégrer.
    """
    return x**2 - 4 # La primitive est x^3/3 - 4x


def integration_rectangles_droite(f, a, b, n):
  """
  Méthode des rectangles à droite pour approximer l'intégrale de f sur
  [a, b].
  Arguments:
  - f : fonction à intégrer
  - a : borne inférieure de l'intégration
  - b : borne supérieure de l'intégration
  - n : nombre de subdivisions de l'intervalle [a, b]
  Retourne:
  - L'approximation de l'intégrale de f sur [a, b].
  """


  h = (b - a) / n # Largeur de chaque sous-intervalle
  x = np.linspace(a + h, b, n) # Points d'évaluation à droite des
  rectangles




  return h * np.sum(f(x))
```

Q.9. Compléter le code suivant de sorte que la fonction `integration_rectangles_gauche` renvoie l'intégrale approximée par la méthode des rectangles à gauche.

```
def integration_rectangles_gauche(f, a, b, n):
    h = (b - a) / n # Largeur de chaque sous-intervalle
    x = np.linspace(a, b - h, n) # Points d'évaluation à gauche des rectangles
    return h * np.sum(f(x))
```

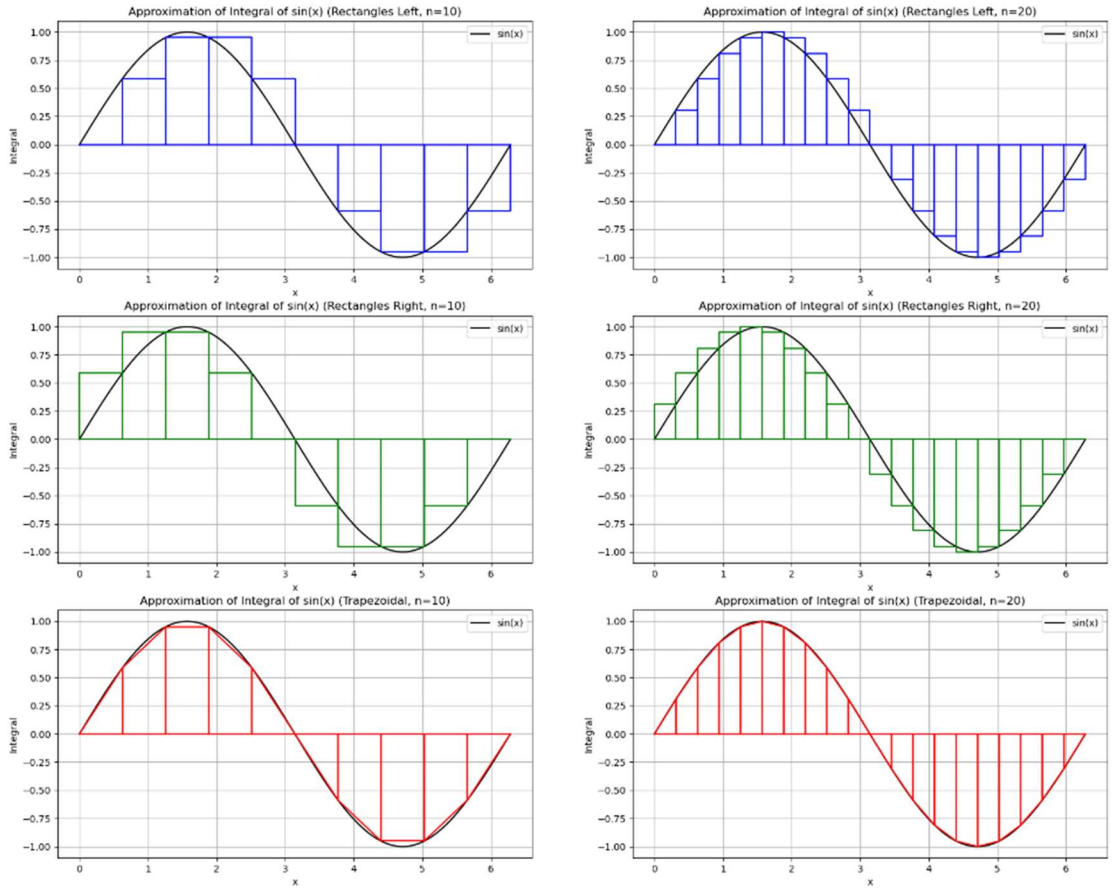
Q.10. Quelle devrait être la valeur approximée renvoyée par `integration_rectangles_gauche(f,0,10,200)` ?

La primitive de $x^2 - 4$ est $\frac{x^3}{3} - 4x$ d'où $I = \frac{10^3}{3} - 40 = 293$

3.4.5 Remarques

Calculer l'aire sous la courbe revient à calculer l'aire entre chaque point échantillonné, il a donc forcément $n - 1$ valeurs.

Plus n est grand, plus h est faible et plus l'erreur d'approximation est faible. La méthode centrée est plus précise.



4 Résolution d'équation différentielle

4.1 Problématique

On considère un système d'équations différentielles sous la forme du problème de Cauchy :

$$\begin{cases} \frac{dX}{dt} = F(t, X) \\ X(t_0) = X_0 \end{cases}$$

Où X est une fonction inconnue caractérisant l'état du système à un instant donné, t_0 et X_0 des données. Dans le cas général, il y a n équations différentielles et n conditions initiales : $X, X_0 \in \mathbb{R}^n$, ce sont des vecteurs. Le système n'est pas forcément linéaire.

4.2 Méthode d'Euler explicite

La méthode d'Euler explicite permet de résoudre numériquement un problème de Cauchy, en particulier une équation différentielle ordinaire. Prenons le cas simplifié où $X, X_0 \in \mathbb{R}^1$ avec $X = y$, $X_0 = y_0$ et $F(t, X) = f(t, y)$.

L'objectif est de déterminer la solution au pas suivant $y(t + h)$ en connaissant la solution au pas actuel $y(x)$ ainsi que sa dérivée $y'(x)$ (donnée par le problème). Le schéma d'Euler explicite utilise la méthode des différences finies progressives (avant).

Rappel : A partir d'une approximation sur le développement de Taylor à l'ordre 1 cette méthode donne :

$$y'(x) = \frac{y(x + h) - y(x)}{h}$$

On peut donc écrire :

$$y(x + h) = h.y'(x) + y(x)$$

Soit en numérique :

$$y_{k+1} = h.f(t, y_k) + y_k$$

4.2.1 Implémentation et exemple

Soit le problème de Cauchy :

$$\begin{cases} \frac{dy}{dt} = 2y \\ y(t_0) = y_0 \end{cases}$$

Q.11. Compléter le code suivant de sorte que la fonction renvoie la dérivée de y telle qu'annoncée par le problème de Cauchy.

```
# Définition de l'EDO y'(t) = f(t, y)
def f(t,y):
      return 2 * y
```

Q.12. Compléter le code suivant de sorte que la fonction `euler_explicite` réalise la résolution numérique du problème de Cauchy.

```
# Méthode d'Euler explicite
def euler_explicite(f, t0, y0, T, h):
    """
    Méthode d'Euler explicite pour résoudre un problème de Cauchy.
    Arguments:
    - f : fonction f(t, y) définissant l'EDO.
    - t0 : instant initial.
    - y0 : condition initiale (valeur de y à t0).
    - T : instant final.
    - h : pas de discrétisation.
    Retourne:
    - t_values : les instants discrétisés.
    - y_values : les approximations de y à chaque instant.
    """

    t_values = np.arange(t0, T + h, h) # Discrétisation des instants (~linspace)
    y_values = [y0] # Initialisation avec la condition initiale

    for t in t_values[:-1]:
        y = y_values[-1]
        y_values.append(y + h * f(t, y)) # Formule d'Euler explicite
    return t_values, y_values
```

Q.13. Quelle devrait être la fonction décrite par les listes renvoyées par `euler_explicite(f,0,2,2,0.1)` ?

On a $\frac{dy}{dt} = 2y \Leftrightarrow \frac{1}{y}dy = 2dt$ d'où par intégration $\ln|y| = 2t + C$, soit en appliquant l'exponentielle :

$y = e^C e^{2t}$, or pour $t = 0$ on a $y = y_0$ d'où $e^C = y_0$ soit $y = y_0 e^{2t}$

Remarques :

- La valeur de $y'(x) = f(t, y_k)$ est la pente permettant de passer de y_k à y_{k+1}.
- Il existe différentes méthodes numériques que celle des différences finies pour approximer la fonction $y'(x)$.
- La méthode de résolution d'un problème de Cauchy dans $\mathbb{R}^n$ est exactement la même.
- La méthode d'Euler explicite ne convient pas à tous les types d'équations différentielles. Parfois cette méthode donne une approximation très grossière, voire diverge. D'autres méthodes sont en pratique utilisées. Seule la méthode d'Euler explicite figure au programme.