Informatique PT. 2025 - 2026

Mathématiques & Informatique - F. Gaunard
http://frederic.gaunard.com
Lycée Voltaire, Paris 11e.

2 Algorithmes classiques en ingénierie numérique

Tous les algorithmes de ce chapitre sont au programme officiel ; il faut donc connaître leur fonctionnement (et savoir les implémenter) sur le bout de doigts.

1 Algorithmes de résolution d'équation de la forme $f(x) = 0$

Dans toute cette section, on considère une fonction $f : [a, b] \to \mathbb{R}$ continue. On cherche à résoudre l'équation $f(x) = 0$ d'inconnue $x \in [a, b]$. Dans toute la suite, on supposera qu'on est en mesure de justifier de l'existence et unicité d'une solution $c \in [a, b]$, telle que $f(c) = 0$.
On renvoie notamment pour cela au **Chapitre 1** (**Section 3**) du cours de mathématiques.

1.1 Algorithme de recherche par dichotomie (associé au TVI)

L'algorithme de recherche par dichotomie repose sur l'observation suivante : si $f(a)f(b) < 0$ alors f change de signe entre a et b et donc la solution de $f(x) = 0$ est entre a et b (par application du TVI, car f est continue). Ainsi, à chaque étape de cet algorithme, on regarde si f change de signe entre l'extrémité de gauche et le milieu de l'intervalle de recherche. Si oui, on cherche donc à gauche du milieu à l'étape d'après, sinon on cherche à droite.
Et on continue jusqu'à ce que la taille de l'intervalle de recherche soit inférieure à la précision voulue. Alors, n'importe quel élément de l'intervalle de recherche fournit une valeur approchée de la solution. On prendra à nouveau le milieu.

☞ Pour une valeur approchée à ε près, le nombre d'itérations de l'algorithme est $N = \left\lceil \log_2\left(\frac{b-a}{\varepsilon}\right)\right\rceil$.

```
def dichotomie_TVI(f, a,b, epsilon) :
    A, B, n = a, b, 0
    c=(A+B)/2
    while B-A > epsilon :
        if f(A)*f(c) < 0 :
            B=c
        else :
            A=c
        n=n+1
        c=(A+B)/2
    return c
```

☞ On peut aussi demander à la fonction de renvoyer le nombre d'itérations en remplaçant la dernière ligne par

```
    return (c, n)
```

Par exemple, avec cette modification et la fonction f ci-dessous, on obtient une valeur approchée de $\sqrt{2}$ à 10^{-4} près.

```
def f(x):
    return x**2-2
>>> dichotomie_TVI(f, 1, 2, 10**(-4))
(1.414215087890625, 14)
```

Cette méthode, robuste, a pour principal défaut sa vitesse de convergente linéaire (donc assez lente). La méthode de Newton, présentée ci-après, elle a une vitesse de convergence quadratique.

1.2 Méthode de Newton

Cette méthode repose sur le développement limité à l'ordre 1 de f au voisinage d'un point x_0 que l'on suppose près de la solution (ce qu'on peut trouver par des estimations grossières). On suppose donc que f est dérivable, ce qui garantit l'existence du développement limité (et des nombres dérivés).
Au voisinage de x_0, on a

$$f(x) \simeq f(x_0) + f'(x_0)(x - x_0).$$

Un valeur approchée de la solution de $f(x) = 0$ peut alors être trouvée en résolvant $f(x_0) + f'(x_0)(x - x_0) = 0$, ce qui donne

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

si toutefois $f'(x_0) \neq 0$. On répète alors le processus : $x_{k+1} = x_k - \dfrac{f(x_k)}{f'(x_k)}$.

Il reste alors toujours la même question: si l'on cherche une valeur approchée de $c \in [a, b]$ à $\varepsilon > 0$ près, quand doit-on arrêter l'algorithme ?
Autrement dit, comment calculer l'entier n tel que x_n soit une approximation de c à ε près? Il faudrait entrer dans des considérations mathématiques pour répondre à cette question.

L'inégalité de Taylor-Lagrange permet de voir que la convergence de (x_n) vers la solution c de $f(x) = 0$ est quadratique, à condition que le point de départ x_0 soit assez proche de la solution c, plus précisément si

$$|x_0 - c| < 1/K, \qquad \text{où} \qquad K = \frac{\max_{[a,b]} |f''(x)|}{2 \min_{[a,b]} |f'(x)|}.$$

En pratique, on utilise souvent une condition naïve d'arrêt, comme par exemple $|f(x_n)| \leqslant \varepsilon$ (on s'arrête lorsque $f(x_n)$ est "suffisamment proche" de 0) où $|x_n - x_{n+1}| \leqslant \varepsilon$ (on s'arrête lorsque l'algorithme semble "stagner").
Évidemment, l'inconvénient de ces conditions d'arrêt est que l'on perd le contrôle de la marge d'erreur (la valeur finale x_n n'est pas une valeur approchée de c à ε près), mais elles présentent l'avantage de simplifier l'analyse préalable du problème, et donnent des résultats satisfaisants si l'on n'est pas très exigeant sur la valeur approchée recherchée.

Le code `Python` suivant est une implémentation de la méthode de Newton avec la condition d'arrêt naïve $|f(x_n)| \leqslant \varepsilon$.

```
def Newton(f, fprime, x0, epsilon) :
    x=x0
    n=0
    while abs(f(x)) > epsilon :
        x=x-f(x)/fprime(x)
        n=n+1
    return (x, n) # ou bien juste return x
```

On applique alors l'algorithme à la fonction définie par $f(x) = x^2 - 2$ partant de $x_0 = 3$, afin d'obtenir une valeur approchée de $\sqrt{2}$.
Il faut seulement trois itérations (contre 14 avec la méthode par dichotomie) pour une précision à 10^{-4} et seulement 5 pour une précision à 10^{-12}.

```
def f(x):
    return x**2-2
def fprime(x):
    return 2*x
>>> Newton(f, fprime, 2, 10**(-4))
(1.4142156862745099, 3)
>>> Newton(f, fprime, 2, 10**(-12))
(1.4142135623730951, 5)
```

On illustre les deux méthodes ci-dessous pour résoudre l'équation $f(x) = 0$ sur $[0, 2]$ avec $f : x \mapsto \dfrac{(2x-3)^2}{8}$.

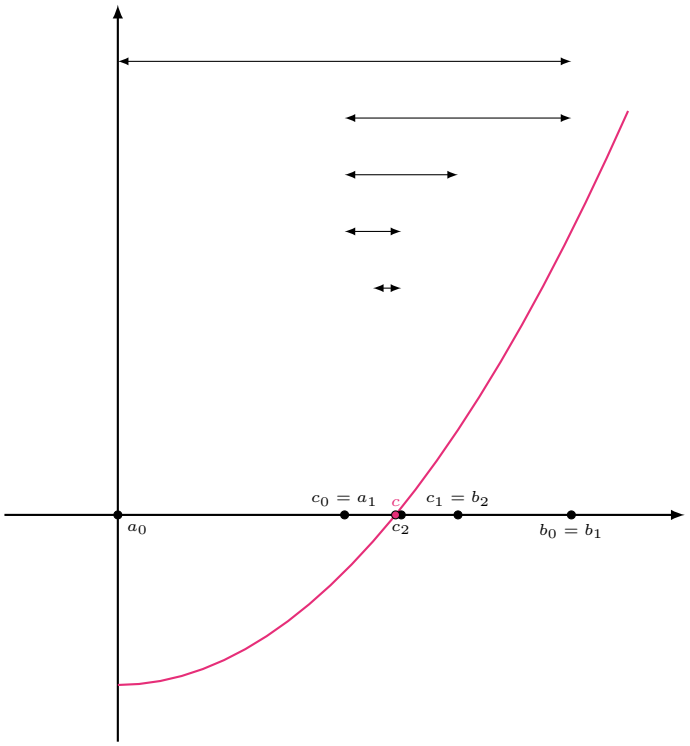


Illustration de la recherche par dichotomie

Si c_2 parait très proche de c, on poursuit quand même l'algorithme car la taille de l'intervalle à la deuxième étape est encore trop grande... Il faut faire tourner l'algorithme pendant 15 itérations pour avoir une précision à 10^{-4}.

$$\begin{aligned}
c_0 &= 1 \quad (a_0 = 0,\ b_0 = 2)\\
c_1 &= 1.5 \quad (a_1 = 1,\ b_1 = 2)\\
c_2 &= 1.25 \quad (a_2 = 1,\ b_2 = 1.5)\\
c_3 &= 1.125 \quad (a_3 = 1,\ b_3 = 1.25)\\
c_4 &= 1.1875 \quad (a_4 = 1.125,\ b_4 = 1.25)\\
&\vdots\\
c_{15} &= 1.224761962890625\\
c &= 1.224744871391589
\end{aligned}$$

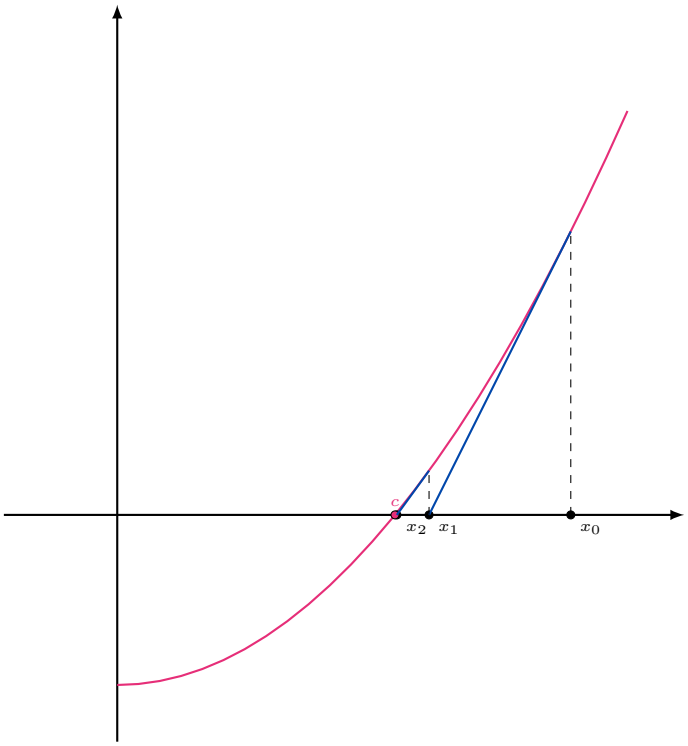


Illustration de la méthode de Newton

Dès la deuxième étape, x_2 est proche de c. Mais on continue car l'écart entre x_1 et x_2 est encore grand. Néanmoins, une étape de plus suffit à avoir un écart inférieur à 10^{-4}.

$$\begin{aligned}
x_0 &= 2\\
x_1 &= 1.375\\
x_2 &= 1.2329545454545454\\
x_3 &= 1.2247722036028488\\
&\vdots\\
c &= 1.224744871391589
\end{aligned}$$

2 Algorithmes de calcul d'une intégrale

2.1 La méthode des rectangles

Soit f une fonction continue définie sur un intervalle $[a, b]$ avec $a < b$.

Pour calculer une valeur approchée de l'intégrale $\displaystyle\int_a^b f(t)\mathrm{d}t$, on peut procéder comme suit.

Soit $n \in \mathbb{N}^*$.

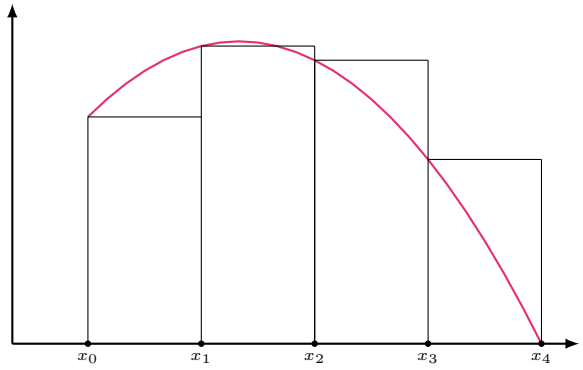
Posons, pour tout $k \in [\![0, n]\!]$, $x_k = a + k\dfrac{b-a}{n}$.

Il vient alors $a = x_0 < x_1 < \cdots < x_n = b$: on a subdivisé l'intervalle $[a, b]$ en n intervalles de longueurs égales $[x_0, x_1], [x_1, x_2], \ldots, [x_{n-1}, x_n]$.

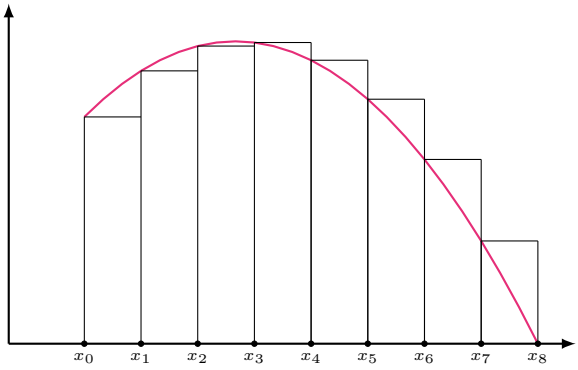
Géométriquement, l'intégrale $\int_a^b f(t)\mathrm{d}t$ correspond à l'aire algébrique délimitée par la courbe représentative de f, l'axe des abscisses et les droites verticales d'équations $x = a$ et $x = b$. On peut approcher cette aire en sommant, l'aire des rectangles dont la base est l'intervalle $[x_k, x_{k+1}]$ et la hauteur est $f(x_k)$, comme indiqué sur les dessins ci dessous (où on a

pris $n = 4$ puis $n = 8$). D'ailleurs, le résultat du cours sur les sommes de Riemann à pas constant (on renvoie au cours de première année ou au début du **Chapitre 5**) stipule alors que

$$\lim_{n\to+\infty} \frac{b-a}{n}\sum_{k=0}^{n-1} f\left(a+k\frac{b-a}{n}\right) = \int_a^b f(t)\mathrm{d}t.$$



$n = 4$



$n = 8$

L'algorithme calcule donc la somme des aires des n rectangles.

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

On l'applique ensuite pour obtenir une valeur approchée de π via la relation $4\displaystyle\int_0^1 \frac{\mathrm{d}x}{1+x^2} = \pi$.
On observe que *la méthode converge*, bien qu'elle soit relativement lente.

```
def g(x):
    return 4/(1+x**2)
>>> rectangle(g,0,1,10**3) # 1000 rectangles
3.1425924869231237
>>> rectangle(g,0,1,10**4) # 10 000 rectangles
3.1416926519232278
```

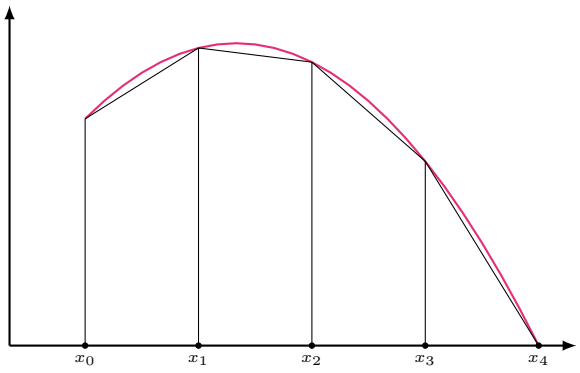
Intuitivement il est clair que plus n est grand (c'est-à-dire plus le pas $\frac{b-a}{n}$ est petit), plus $\displaystyle\frac{b-a}{n}\sum_{k=0}^{n-1} f\left(a+k\frac{b-a}{n}\right)$ est une valeur approchée précise de $\displaystyle\int_a^b f(t)\mathrm{d}t$.
Mais si l'on souhaite obtenir une valeur approchée à $\varepsilon > 0$ près, quelle valeur de n faut-il choisir? Pour pouvoir y répondre, il faut être capable de calculer (ou à défaut de majorer) l'erreur

$$\left|\int_a^b f(t)\mathrm{d}t - \frac{b-a}{n}\sum_{k=0}^{n-1} f\left(a+k\frac{b-a}{n}\right)\right|$$

associée à l'approximation considérée (cela peut être fait ...).

2.2 Méthode des trapèzes

Au lieu d'utiliser des rectangles, on peut construire des trapèzes, comme sur le dessin suivant :



Méthode des trapèzes (ici $n = 4$)

On approche alors l'intégrale par la somme des aires algébriques des trapèzes, qui vaut

$$\sum_{k=0}^{n-1}(x_{k+1}-x_k)\times\frac{1}{2}\left(f(x_k)+f(x_{k+1})\right)=\frac{b-a}{2n}\sum_{k=0}^{n-1}\left(f(x_k)+f(x_{k+1})\right).$$

Ceci s'implémente sans difficulté

```
def trapeze(f,a,b,n):
    A,x=0,a
    for k in range(n):
        A=A+f(x)+f(x+(b-a)/n)
        x=x+(b-a)/n
    return ((b-a)/(2*n))*A
```

La méthode des trapèzes semble converger plus vite que la méthode des rectangles: pour avoir une valeur approchée de π à 10^{-4} près avec la méthode des trapèzes la valeur $n = 10^2$ convient, alors que $n = 10^3$ ne convient pas pour la méthode des rectangles.

```
>>> trapeze(g,0,1,10)
3.1399259889071587
>>> trapeze(g,0,1,100)
3.141575986923126
```

3 Résolution d'une équation différentielle: la méthode d'Euler

Le calcul explicite des solutions d'une équation différentielle n'est pas toujours aisé, et il existe un grand nombre de cas que l'on ne sait pas traiter. On dispose néanmoins de *méthodes numériques* qui permettent d'obtenir une valeur approchée de la solution, ce qui est suffisant dans le cadre des sciences expérimentales. La plus simple d'entre elles est la *méthode d'Euler*.

Le Cadre. Considérons une fonction $f : \mathbb{R}^2 \to \mathbb{R}$, et I un intervalle non trivial de $\mathbb{R}$. On cherche à déterminer les fonctions $y : I \to \mathbb{R}$ qui vérifient l'équation différentielle

$$\forall t \in I,\ y'(t) = f(y(t), t).$$

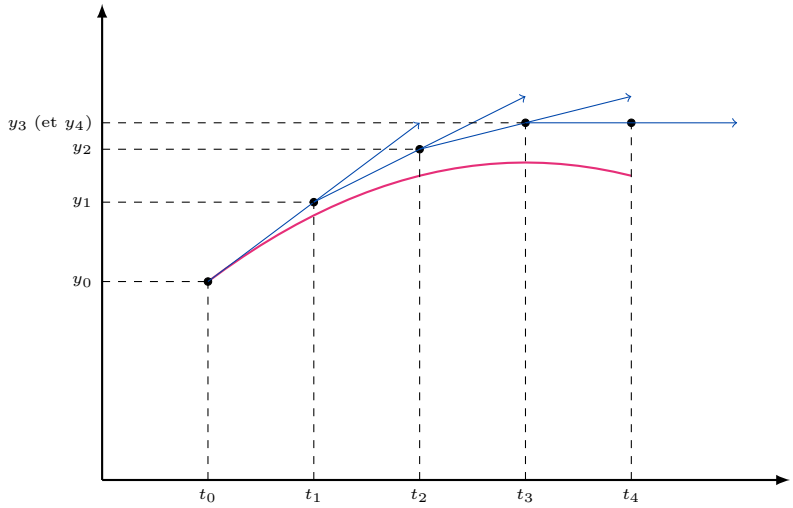
On a ici affaire à une équation différentielle du premier ordre (non linéaire et à coefficients non constants *a priori*). On **admet**[1] que, si la fonction f est de classe $\mathcal{C}^1$, alors quels que soient $(y_0, t_0) \in \mathbb{R} \times I$, cette équation possède une unique solution y qui vérifie $y(t_0) = y_0$.

La méthode : Approximation de la courbe représentative de la solution par une ligne polygonale.
Le principe de la méthode d'Euler est le suivant. Soit y la solution de l'équation différentielle considérée. On commence par se donner une subdivision $(t_0, ..., t_n)$ régulière de I, et on cherche une valeur approchée y_i de chacun des réels $y(t_i)$. En reliant par des segments de droite les points de coordonnées (t_i, y_i) on obtient alors une ligne polygonale qui est une approximation de la courbe représentative de f. A priori, plus le nombre n de points que l'on s'est donné est important, plus le graphe obtenu sera une approximation de qualité.

Principe de la méthode, ou approximation d'une dérivée par la pente d'une corde.
Naïvement, le principe de la méthode peut être exprimé comme suit: partant d'une valeur approchée de y_i de $y(t_i)$, il suffit

[1] Ce résultat est une conséquence du théorème de Cauchy-Lipschitz

de suivre la tangente en $(t_i, y(t_i))$ (à la courbe représentative $\mathcal{C}_y$ de y) jusqu'à l'abscisse t_{i+1}; l'ordonnée obtenue est alors une valeur approchée de $y(t_{i+1})$, que l'on appelle y_{i+1}.
On illustre cette idée par la figure ci-dessous :



Pour formaliser cette méthode, on utilise le fait que si y désigne une fonction dérivable sur un intervalle $[t_i, t_{i+1}]$, lorsque la valeur de $h = t_{i+1} - t_i$ est "suffisamment petite", on peut écrire:

$$y'(t_i) \simeq \frac{y(t_{i+1}) - y(t_i)}{t_{i+1} - t_i} = \frac{y(t_{i+1}) - y(t_i)}{h},$$

ce qui équivaut à $y(t_{i+1}) \simeq y(t_i) + hy'(t_i)$.
En d'autres termes, si h est "suffisamment petit", c'est-à-dire si t_i et t_{i+1} sont suffisamment proches, $y(t_i) + hy'(t_i)$ est une valeur approchée de $y(t_{i+1})$, obtenue en suivant la tangente en $(t_i, y(t_i))$ jusqu'à l'abscisse t_{i+1} (on reconnait l'équation de la tangente en $(t_i, y(t_i))$ dans le second membre).

Mais ce n'est pas si simple, car on ne connait pas la valeur de $y'(t_i)$ et donc pas l'équation de la tangente en $(t_i, y(t_i))$.
Mais comme y vérifie l'équation $y'(t) = f(y(t), t)$, on en déduit $y'(t_i) = f(y(t_i), t_i)$.
Avec la relation $y(t_{i+1}) \simeq y(t_i) + hy'(t_i)$, on obtient alors $y(t_{i+1}) \simeq y(t_i) + hf(y(t_i), t_i)$.
La suite des approximations $(y_i)_{0 \leqslant i \leqslant n}$ peut alors être définie par récurrence, en *imitant ce comportement*:

$$y_{i+1} = y_i + hf(y_i, t_i)\,.$$

Implémentation de la méthode d'Euler.
Considérons une équation différentielle $y'(t) = f(y(t), t)$, à résoudre sur un intervalle $[a, b]$ avec la condition initiale $y(a) = \alpha$.
On commence par se donner une subdivision $(t_0, ..., t_n)$ de $[a, b]$ de pas constant $h = \frac{b-a}{n}$ **suffisamment petit**, et on définit alors notre suite d'approximation $y_0, \ldots, y_n$ en posant

$$y_0 = \alpha \quad \text{et} \quad \forall i \in [\![0, n-1]\!],\ y_{i+1} = y_i + hf(y_i, t_i).$$

On a ainsi défini par récurrence une suite $(y_0, \ldots, y_n)$ qui est une approximation de la suite $y(t_0), \ldots, y(t_n)$, où y désigne la solution de l'équation considérée. En effet, y vérifie la relation

$$y(t_{i+1}) = y(t_i) + hf(y(t_i), t_i) + o(h),$$

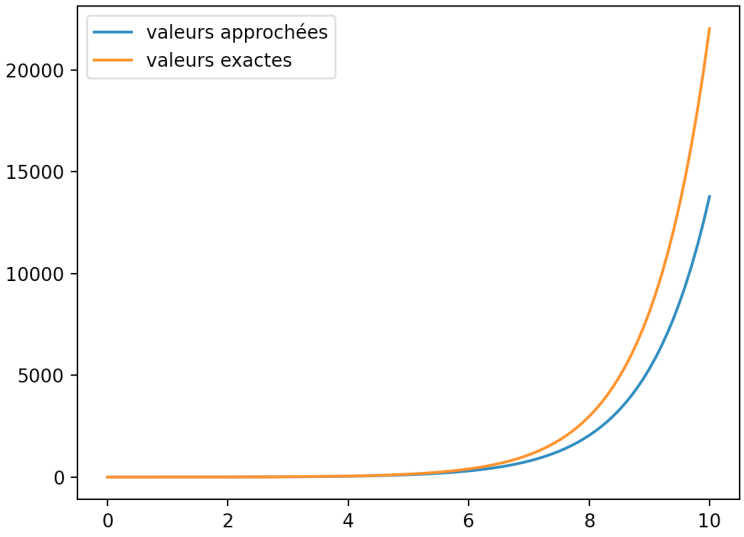
puisque l'on a $y'(t_i) = f(y(t_i), t_i)$ pour tout entier $i \in [\![0, n]\!]$.

```
def Euler(f, a, b, y0, h):
    liste_y=[y0]
    liste_t=[a]
    k=0
    while liste_t[k]+h<=b:
        y=liste_y[k]
        t=liste_t[k]
        liste_y.append(y+h*f(y,t))
        liste_t.append(t+h)
        k=k+1
    return (liste_t,liste_y)
```

Exemple 1.

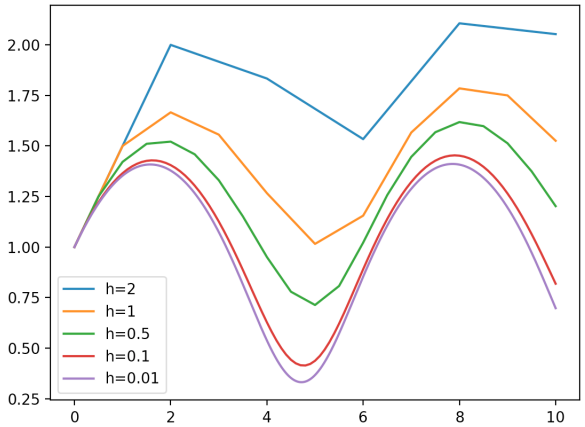
On implémente la méthode d'Euler, puis on l'applique à l'équation différentielle $y' = y$ sur $[0, 10]$ munie de la condition initiale $y(0) = 1$ (l'équation différentielle est donc ici associée à la fonction définie par $f(y, t) = y$).
La solution recherchée est donc l'exponentielle, et on représente les valeurs approchées sur un graphe pour observer l'erreur commise.

```
import matplotlib.pyplot as plt
import math as ma

def f(y,t) :
    return y

a, b, h=0,10,10**(-1)
liste_t,liste_y=Euler(f,a,b,1,h)
plt.plot(liste_t, liste_y, label="valeurs approchées")

liste_exp=[ma.exp(t) for t in liste_t]
plt.plot(liste_t, liste_exp, label="valeurs exactes")
plt.legend(loc=2)
plt.show()
```



Exercice 1.

Soit $y' = \dfrac{\cos t}{1 + y^2}$ à résoudre sur $[0, 10]$, avec la condition initiale $y(0) = 1$.



Le graphique ci-contre fait apparaitre la courbe représentative de la solution, ainsi que les courbes obtenues en appliquant la méthode d'Euler, avec les différents pas suivants: $h = 2$, $h = 1$, $h = \frac{1}{2}$, $h = \frac{1}{10}$ et $h = \frac{1}{100}$.
On observe bien la convergence de la ligne polygonale vers la courbe représentative de la solution.

Écrire un programme `Python` qui permet d'obtenir le graphique ci-dessus.

☞ Pour associer le nom $h = 0.1$ à la courbe calculée avec le pas $1/10$, on utilisera la commande `label="h="+str(h)`. En effet, `"h="+str(h)` est la chaîne de caractère `h=x` où `x` est la valeur contenue dans la variable `h` (qui peut être un entier, un réel, une chaîne de caractères,...).
La commande `plt.legend(loc='best')` permet d'ajouter la légende, à la "meilleure" position possible sur la figure.

Remarque 1.

Intuitivement, il est clair que plus le pas h est petit, plus l'approximation de la solution est précise: on peut démontrer que *la méthode converge* lorsque h tend vers 0.
En pratique, l'implémentation de la méthode pose d'autre difficultés, des erreurs d'arrondis pouvant apparaître à chaque étape: en diminuant la valeur de h on augmente le nombre d'étapes lors de l'exécution de l'algorithme, et ainsi le nombre d'erreurs commises, ce qui peut faire diverger la méthode.

3.1 Cas des équations d'ordre deux

Dans le cas d'une équation d'ordre deux, il faut d'abord se ramener à une équation d'ordre un avant de pouvoir appliquer la méthode d'Euler. On utilise pour cela un changement de variable vectoriel, comme l'illustre l'exemple ci-dessous.

Considérons l'équation différentielle suivante, à résoudre sur l'intervalle $I = [a, b]$:

$$(\mathcal{E}) : \ y''(t) + c_1(t)y'(t) + c_0(t)y(t) = d(t)$$

où c_1, c_0, d sont des fonctions quelconques sur I. On se donne encore les conditions initiales $y(0) = \alpha$ et $y'(0) = \alpha'$.

Considérer la variable vectorielle définie par $Y(t) = \begin{pmatrix} y(t) \\ y'(t) \end{pmatrix}$. La dérivée de Y est alors $Y'(t) = \begin{pmatrix} y'(t) \\ y''(t) \end{pmatrix}$, et on peut écrire:

$$\begin{aligned} y \text{ solution de } (\mathcal{E}) &\iff y''(t) + c_1(t)y'(t) + c_0(t)y(t) = d(t) \\ &\iff \begin{pmatrix} y'(t) \\ y''(t) \end{pmatrix} = \begin{pmatrix} y'(t) \\ d(t) - c_1(t)y'(t) - c_0(t)y(t) \end{pmatrix} \\ &\iff Y'(t) = f(Y(t), t) \\ &\iff Y \text{ solution de } (\mathcal{E}') \end{aligned}$$

où on a posé $f\left(\begin{pmatrix} u_0 \\ u_1 \end{pmatrix}, t\right) = \begin{pmatrix} u_1 \\ d(t) - c_1(t)u_1 - c_0(t)u_0 \end{pmatrix}$ et $(\mathcal{E}') : \ Y'(t) = f(Y(t), t)$.
On peut alors appliquer la méthode d'Euler à l'équation différentielle $Y' = f(Y, t)$, la condition initiale étant $Y_0 = (\alpha, \alpha')$.

$$Y_{i+1} = Y_i + hf(Y_i, t_i) .$$

Pour modéliser le vecteur $Y(t)$, on utilisera un tableau 1D, ce qui permettra d'effectuer naturellement du calcul vectoriel.
☞ On garde ensuite la première colonne de la `liste_Y` en sortie.

Exemple 2.

On veut appliquer cette méthode à la résolution du problème de Cauchy, défini sur $[0; 10]$, par

$$(\mathcal{P}) \qquad \begin{cases} y'' + y &= 0 \\ y(0) &= 1 \\ y'(0) &= -1 \end{cases}$$

dont on sait déterminer la solution avec la méthode du cours. En effet, la solution est $y : t \longmapsto \cos(t) - \sin(t)$.
On reformule le problème en

$$(\mathcal{P}') \qquad \begin{cases} Y' &= f(Y, t) \\ Y(0) &= \begin{pmatrix} 1 \\ -1 \end{pmatrix} \end{cases} \qquad \text{avec} \qquad f\left(\begin{pmatrix} y_0 \\ y_1 \end{pmatrix}, t\right) = \begin{pmatrix} y_1 \\ -y_0 \end{pmatrix}$$

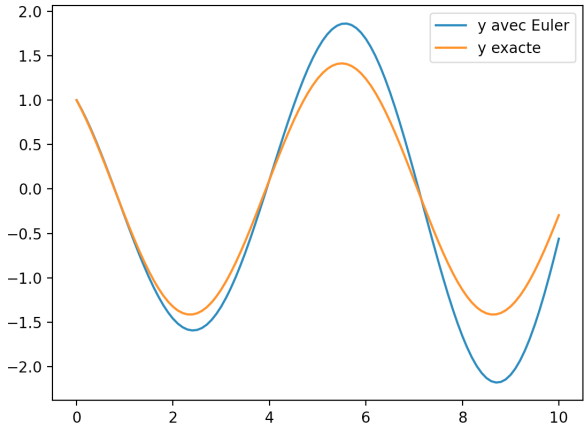
☞ Il faut bien s'assurer que l'on utilise des tableaux 1D donc les listes de la fonction `Euler` doivent être converties.

On n'oublie pas non plus d'importer les bibliothèques nécessaires (notamment `numpy`).

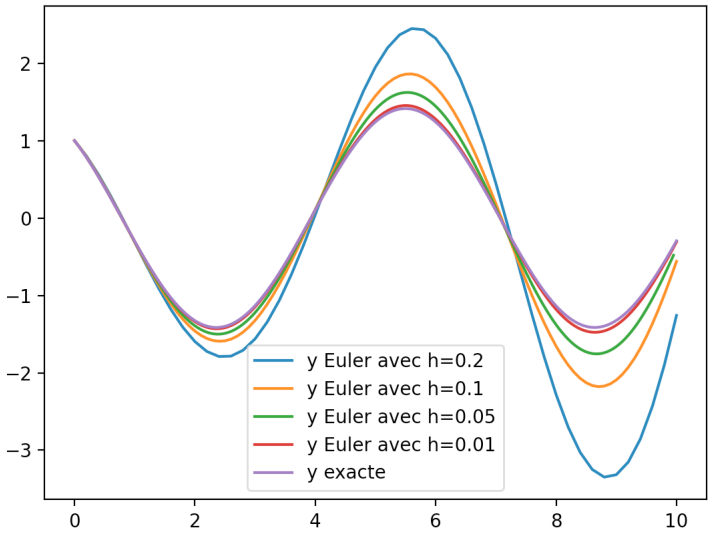
```
import matplotlib.pyplot as plt
import math as ma
import numpy as np
```

```
def Euler(f, a, b, Y0, h):
    liste_Y=[Y0]
    liste_t=[a]
    k=0
    while liste_t[k]+h<=b:
        Y=liste_Y[k]
        t=liste_t[k]
        liste_y.append(Y+h*f(Y,t))
        liste_t.append(t+h)
        k=k+1
    return (np.array(liste_t),np.array(liste_Y))
```

```
def f(Y,t) :
    return np.array([Y[1], -Y[0]])

a, b = 0, 10
Y0 = np.array([1, -1])
h = .1
T, Y = Euler(f,a,b,Y0,h)

plt.plot(T, Y[:,0], label="y Euler")

Y_e=[ma.cos(t)-ma.sin(t) for t in T]
plt.plot(T, Y_e, label="y exacte")

plt.legend(loc='best')
plt.show()
```



☞ Naturellement, plus le pas est petit, plus la solution est précise, comme on peut encore le voir sur la figure ci-dessous :



4 Annales de l'oral

On proposes quelques exercices tombés à l'oral faisant référence aux notions de ce chapitre.

Exercice 2. — Oral Math II 2021

1. Faire tracer les courbes des fonctions $g : x \mapsto x\ln(x) - 2$ et $h : x \mapsto x^3 - 4x^2 + 2x - 1$ sur l'intervalle $[1, 5]$.
2. Rappeler le principe de la **méthode de dichotomie** pour résoudre numérique l'équation $g(x) = 0$.
3. Déterminer un encadrement $[a, b]$ de la solution de l'équation $g(x) = 0$ de sorte que $0 \leqslant b - a \leqslant 2^{-10}$.
4. Reprendre la question précédente avec la fonction h.
5. On va maintenant utiliser la fonction `bisect` du module `scipy.optimize` de `Python`. Lire l'aide sur cette fonction, puis l'utiliser pour résoudre à nouveau les deux équations précédentes.

Exercice 3. — Math II 2023

1. Écrire une fonction `trapeze1` de paramètres f, a, b, n qui calcule l'intégrale d'une fonction f sur l'intervalle $[a, b]$ en utilisant la **méthode des trapèzes** sur n intervalles.
2. Vérifier que le programme fonctionne en calculant $\displaystyle\int_0^{\pi} \sin(t)\mathrm{d}t$.
3. Améliorer ce programme en écrivant une fonction `trapeze2` qui renvoie en outre la grandeur
$$\varepsilon = \frac{b-a}{12} \max |t_{k+1} - 2t_k + t_{k-1}|,$$
où t_k est la valeur de f au k-ième point du découpage (ε est l'erreur associée).
4. Tracer f définie par $f(t) = \dfrac{e^{-t}}{1+t^2}$ sur $[0, 4]$.
5. Trouver le plus petit réel T de la forme $T = \frac{i}{10}$, avec i entier, tel que $f(T) < 5 \cdot 10^{-6}$.
En déduire une valeur approchée de $\displaystyle\int_0^{+\infty} f(t)\mathrm{d}t$.
6. Comparer la valeur obtenue avec celle donnée par la fonction `quad` du module `scipy.integrate`.

Exercice 4. — Oral Math II 2019

On cherche à étudier numériquement les solutions de l'équation différentielle suivante :
$$y'(t) = t^2 - y(t)^3 \quad \text{avec} \quad y(-1.5) = a \qquad (1)$$

1. Pour $a = 2$ et un pas de discrétisation $h = 1/4$, puis $h = 1/8$, représenter les solutions approchées du problème (1) obtenues par la **méthode d'Euler** sur l'intervalle$[-1.5, 2.5]$.
2. En utilisant la fonction `odeint` du module `scipy.integrate` de `Python`, résoudre numériquement le problème (1) pour $a = 2$. On prendra garde à définir soigneusement les arguments de cette fonction en lisant attentivement l'aide en ligne.
3. Sur la figure existante, rajouter la courbe de la solution numérique obtenue à la question précédente.
4. Rajouter ensuite les courbes des solutions numériques pour $a = 1.3$ et $a = 0.1$. Qu'observe-t-on?
5. Pour résoudre $y'(t) = f(t, y(t))$ avec $y(t_0) = a$, on utilise maintenant la suite (y_n) définie par $y_0 = a$ et
$$y_{k+1} = y_k + \frac{h}{2}\left(f(t_k, y_k) + f(t_{k+1}, y_k + hf(t_k, y_k))\right)$$
Représenter, pour $a = 2$ et les mêmes pas de discrétisation qu'à la Question**1.** , les solutions approchées obtenues par cette méthode. Expliquer pourquoi le résultat semble meilleur.