TD Python 1 : Intégration

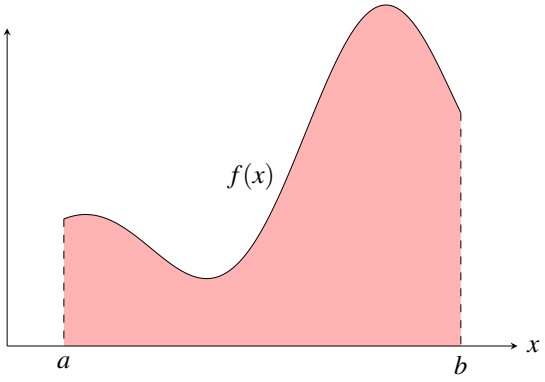
1 Calcul approché d'une intégrale

On cherche à évaluer numériquement l'intégrale $I = \int_a^b f(x)\mathrm{d}x$. Nous allons voir plusieurs méthodes qui permettent d'obtenir une valeur approchée de cette intégrale en remplaçant la fonction f par une fonction plus simple à manipuler.

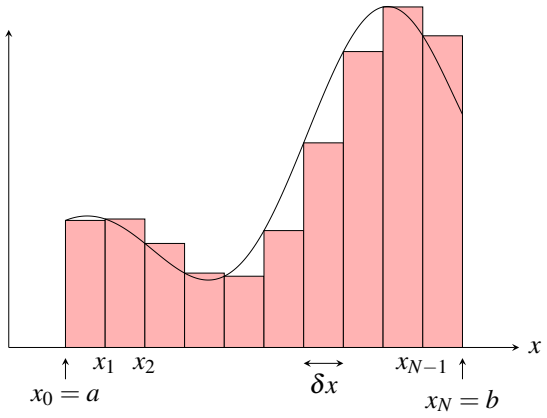
1.1 Méthode des rectangles

Cette méthode consiste à remplacer la fonction f à intégrer par une fonction $\tilde{f}$ «en marches d'escalier» (c'est-à-dire constante par morceaux) avec un **pas de discrétisation** δx régulier. On parle de méthode des rectangles à gauche (resp. à droite) lorsque la fonction à intégrer intersecte les paliers à leur extrémité gauche (resp. à leur extrémité droite, voir figures ci-dessous). Supposons par exemple que l'intervalle $[a,b]$ est découpé en N subdivisions régulières et notons $(x_0, x_1, ..., x_{N-1}, x_N)$ les extrémités de ces sous-intervalles (avec $x_0 = a$ et $x_N = b$ comme indiqués sur les figures ci-dessous). La fonction approchée est telle que sur tout intervalle $[x_k, x_{k+1}]$, $k \in [\![0, N-1]\!]$:

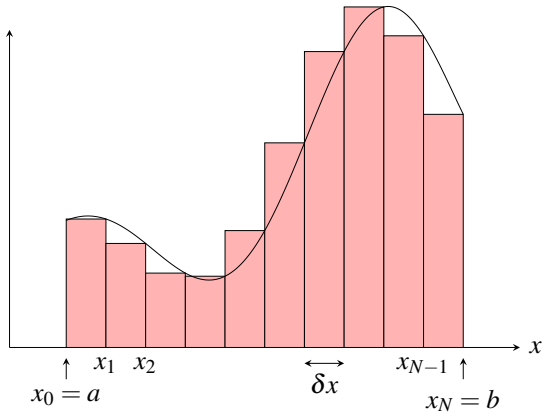
- $\tilde{f}(x) = f(x_k)$ pour la méthode des rectangles à **gauche** ;
- $\tilde{f}(x) = f(x_{k+1})$ pour la méthode des rectangles à **droite**.



Intégrale à calculer



Méthode des rectangles à gauche



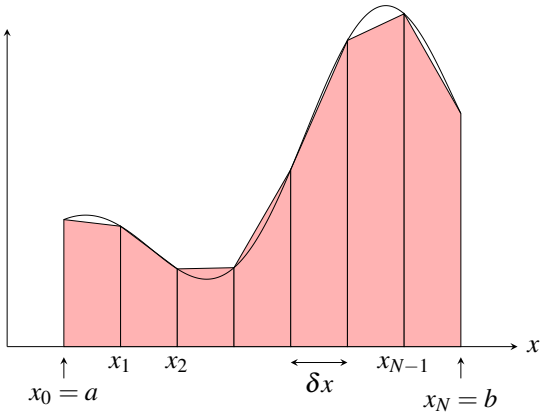
Méthode des rectangles à droite

Notons $\mathscr{A}_k$ l'aire du rectangle situé entre les abscisses x_k et x_{k+1} ($k \in [\![0, N-1]\!]$). La valeur approchée de l'intégrale I (que l'on notera $\tilde{I}$ par la suite), s'obtient en sommant les aires des différents rectangles :

$$\boxed{\tilde{I} = \sum_{k=0}^{N-1} \mathscr{A}_k}$$

1.2 Méthode des trapèzes

La méthode des trapèzes est semblable à la méthode des rectangles, à la différence que l'on approche la fonction f par une fonction $\tilde{f}$ **affine par morceaux** (au lieu de constante par morceaux, voir figure ci-dessous), de pas de discrétisation régulier δx. La fonction approchée est choisie de telle sorte que $\tilde{f}(x_k) = f(x_k)\ \forall k \in [\![0, N]\!]$.



Méthode des trapèzes

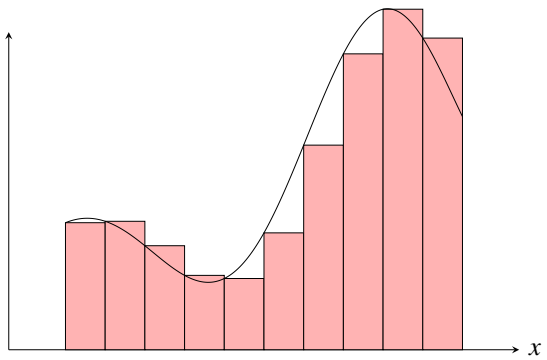
Là encore la valeur approchée $\tilde{I}$ est obtenue en sommant les aires $\mathscr{A}_k$ des trapèzes situés entre les abscisses x_k et x_{k+1} ($k \in [\![0, N-1]\!]$) :

$$\boxed{\tilde{I} = \sum_{k=0}^{N-1} \mathscr{A}_k}$$

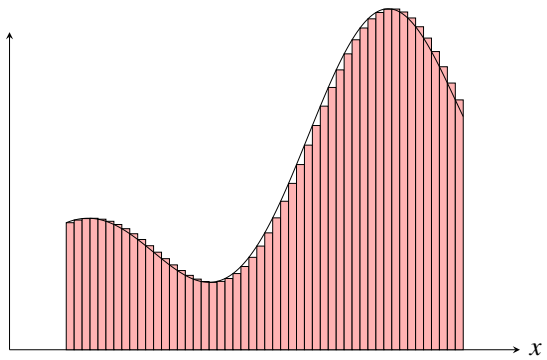
1.3 Comparaison des deux méthodes

Les deux méthodes sont simples à mettre en œuvre avec Python car l'aire d'un rectangle ou d'un trapèze est facile à calculer.

Quelque soit la méthode choisie $\tilde{I}$ converge vers I dans la limite $\delta x \to 0$ (c'est-à-dire $N \to +\infty$). Autrement dit le résultat obtenu est d'autant plus précis que le pas de discrétisation δx est faible, c'est-à-dire que le nombre N de subdivisions est élevé (voir figures ci-dessous).



Méthode des rectangles à gauche ($N = 10$)

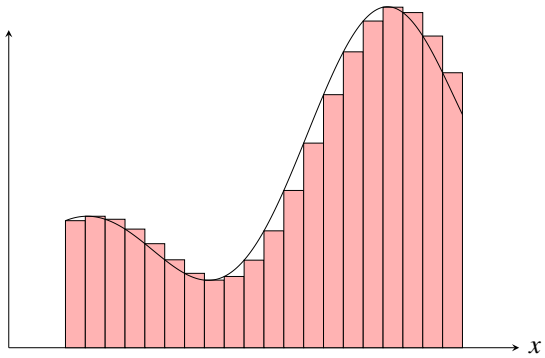


Méthode des rectangles à gauche ($N = 50$)

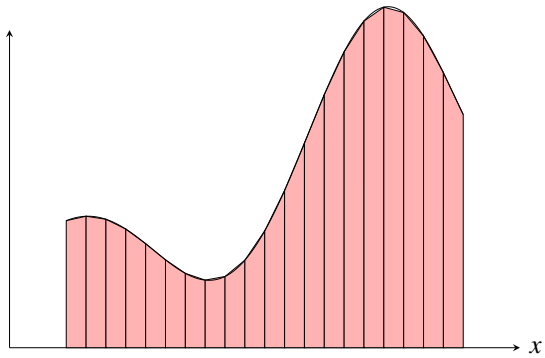
La différence principale entre les deux méthodes se situe au niveau de leur **vitesse de convergence**.

- la méthode des rectangles est dite **d'ordre 1** car l'erreur $\left|I-\tilde{I}\right|$ est $\boxed{\text{proportionnelle à } \delta x}$;
- la méthode des trapèzes est dite **d'ordre 2** car l'erreur $\left|I-\tilde{I}\right|$ est $\boxed{\text{proportionnelle à } (\delta x)^2}$.

Le résultat $\tilde{I}$ converge donc **beaucoup plus vite** vers I avec la méthode des trapèzes qu'avec la méthode des rectangles quand $\delta x \to 0$. Cela est dû au fait que, pour approcher la fonction f à intégrer, une fonction affine par morceaux est bien plus efficiente qu'une fonction constante par morceaux (voir figures ci-dessous).

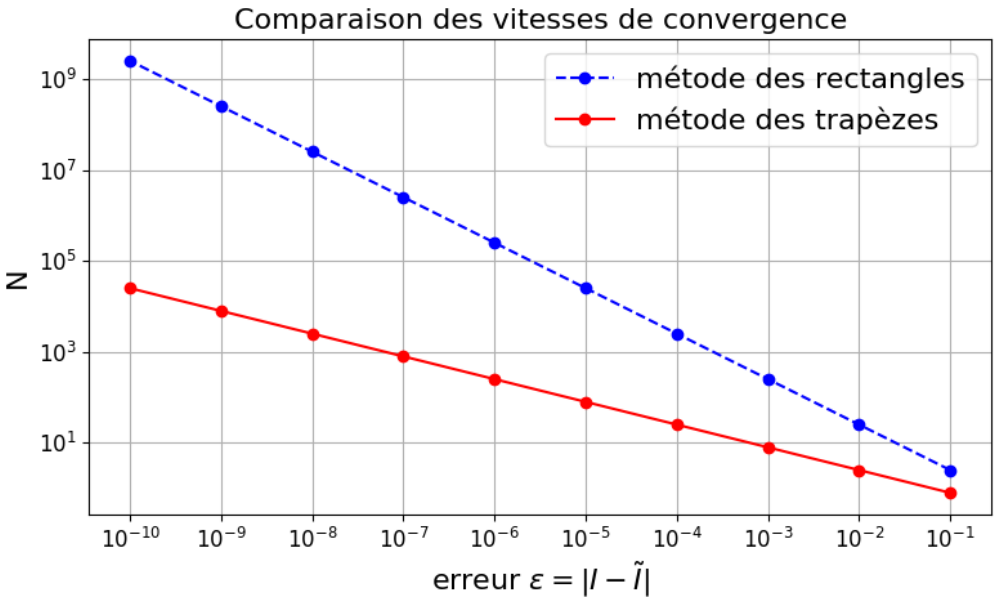


Méthode des rectangles à gauche ($N = 20$)



Méthode des trapèzes ($N = 20$)

Prenons l'exemple de l'intégrale $I = \int_0^1 \frac{x}{x+1} \mathrm{d}x$, dont la valeur exacte est $I = 1 - \ln 2$. Le graphe ci-dessous montre le nombre de subdivisions N nécessaire pour obtenir un résultat de précision donnée, mesurée par l'erreur $\varepsilon = \left|I-\tilde{I}\right|$ commise (on notera que le graphe est en échelle logarithmique sur les deux axes). Sachant que le temps de calcul est proportionnel à N (car il y a N rectangles (ou trapèzes) dont il faut calculer l'aire), on peut considérer que l'axe des ordonnées permet de lire les variations du **temps de calcul** nécessaire pour obtenir une précision donnée.



On remarque les points suivants :

- plus ε est petit (autrement dit meilleure est la précision attendue) et **plus le temps de calcul augmente** ;
- la méthode des trapèzes est **beaucoup plus rapide** que la méthode des rectangles, et l'écart entre les deux s'accentue à mesure que le degré de précision attendu augmente.

Pour mesurer à quel point l'écart est important entre les deux méthodes en termes de temps de calcul, notons que :

- pour un résultat précis à $\varepsilon = 10^{-4}$ près la méthode des trapèzes est **cent fois plus rapide** que la méthode des rectangles ;
- pour $\varepsilon = 10^{-6}$ elle est **mille fois plus rapide** ;
- pour $\varepsilon = 10^{-8}$ elle est **dix-mille fois plus rapide**.

À retenir : la méthode des trapèzes (d'ordre 2) est **beaucoup** plus avantageuse que la méthode des rectangles (d'ordre 1) en termes de temps de calcul, et l'écart entre les deux s'accentue à mesure que N augmente (autrement dit que l'on souhaite une meilleure précision).

Travail préparatoire

On souhaite calculer l'intégrale $I = \int_0^1 \frac{e^{-x}}{x+1} dx$. On présente ci-dessous un code Python incomplet.

```
1  import numpy as np
2
3  a, b = 0, 1            # bornes d'intégration
4  N = 1000               # nombre de subdivisions
5  dx =                   # pas de discrétisation
6
7  # Définition du tableau x qui divise l'intervalle [0,1] en N subdivisions régulières
8  x =
9
10 # Définition de la fonction à intégrer
11 def f(x):
12     return np.exp(-x) / (x + 1)
13
14 # Initialisation de l'intégrale
15 I = 0
16
17 # Calcul de l'intégrale
18 for k in range( ):
19     I +=
```

1. Compléter la ligne 5 qui calcule le pas de discrétisation.

2. Combien d'éléments contient le tableau `x` si l'on veut découper l'intervalle $[0,1]$ en $N = 1000$ subdivisions régulières ? Compléter la ligne 8 qui définit ce tableau.

3. On note $f(x) = \frac{e^{-x}}{x+1}$ la fonction à intégrer. Donner l'expression de l'aire $\mathscr{A}_k$ du rectangle compris entre les abscisses x_k et x_{k+1} dans le cas de la **méthode des rectangles à gauche**.

4. La boucle `for` permet de calculer la valeur approchée $\tilde{I}$ de l'intégrale I en sommant les aires des différents rectangles.

a) Compléter la ligne 18 en écrivant l'argument qui convient dans la fonction `range()`.

b) Compléter la ligne 19.
La syntaxe `I +=` est équivalente à `I = I +` .
On rappelle que `x[k]` appelle le terme de rang `k` dans le tableau `x`.

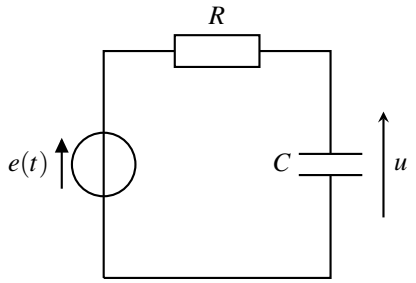
5. Reprendre les questions 3. et 4.b) dans le cas de la méthode des rectangles à droite puis la méthode des trapèzes.

6. On souhaite maintenant calculer $J = \int_0^\pi \frac{\sin x}{x} dx$ (on admet qu'elle est définie). Quel problème se pose si l'on utilise la méthode des rectangles à gauche ? Le problème se pose-t-il encore avec la méthode des rectangles à droite ? des trapèzes ?

2 Résolution numérique d'une équation différentielle par la méthode d'Euler

2.1 Position du problème

On étudie un circuit RC série alimenté par un générateur de force électromotrice quelconque $e(t)$.



On a vu en cours que la tension $u(t)$ était solution de l'équation différentielle :

$$\frac{\mathrm{d}u}{\mathrm{d}t} + \frac{u}{\tau} = \frac{e(t)}{\tau} \quad \text{avec} \quad \tau = RC$$

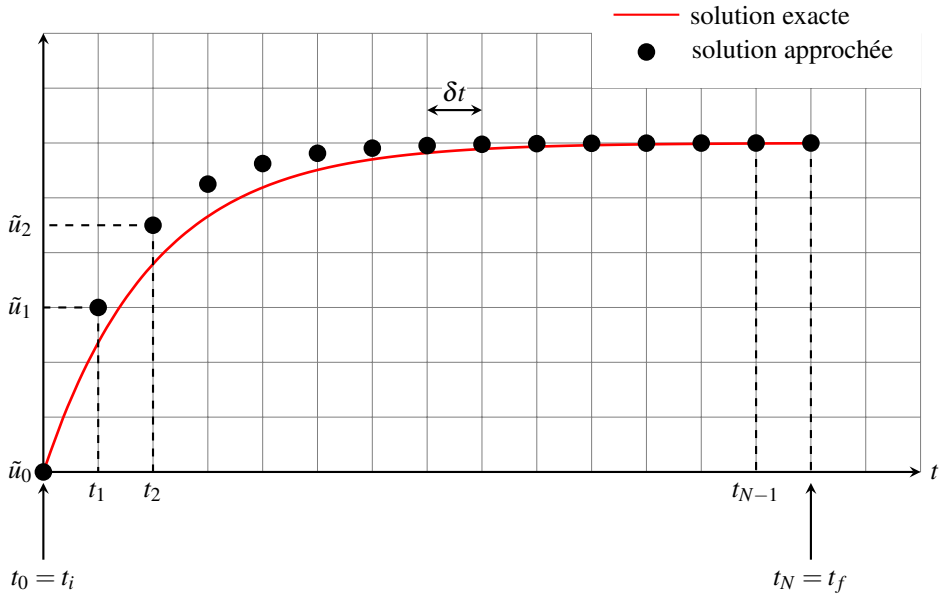
Les données du problème sont :

- les valeurs de R et C ;
- la tension d'alimentation $e(t)$;
- l'intervalle $[t_i, t_f]$ sur lequel on souhaite calculer la tension $u(t)$;
- la condition initiale $u(t_i)$ (on choisira ici $t_i = 0^+$).

On se fixe comme objectif d'obtenir à l'aide de Python **une approximation de la solution de cette équation différentielle** sur l'intervalle $[t_i, t_f]$.

2.2 Discrétisation

Par la suite nous noterons $u(t)$ la solution **exacte** de l'équation différentielle et $\tilde{u}(t)$ la solution **approchée**. La méthode consiste à découper l'intervalle $[t_i, t_f]$ en subdivisions régulières avec un **pas de discrétisation** δt **arbitraire** et à chercher des valeurs approchées de la solution de l'équation différentielle pour les dates $t_0 = t_i$, $t_1 = t_i + \delta t$, $t_2 = t_i + 2\delta t$,...,$t_N = t_i + N\delta t = t_f$.



La résolution numérique de l'équation différentielle se présente donc sous la forme d'un **tableau de nombres** $(\tilde{u}_0, \tilde{u}_1, ..., \tilde{u}_N)$ qui approche la solution exacte pour les dates $(t_0, t_1, ..., t_N)$.

Pour comprendre comment calculer les valeurs approchées $\tilde{u}_k$, $k \in [\![0,N]\!]$, commençons par réécrire l'équation différentielle sous la forme ci-dessous :

$$\frac{\mathrm{d}u}{\mathrm{d}t} = f(u(t),e(t)) \qquad \text{avec} \qquad f(u(t),e(t)) = \frac{e(t)-u(t)}{\tau}$$

La condition initiale $u(t_0)$ étant connue, la manière **exacte** de calculer $u(t)$ pour $t > 0$ quelconque est la suivante :

$$u(t) - u(t_0) = \int_{t_0}^{t} \frac{\mathrm{d}u}{\mathrm{d}t'}\mathrm{d}t' = \int_{t_0}^{t} f(u(t'),e(t'))\mathrm{d}t'$$

Ce qui revient à dire que la solution **exacte** de l'équation différentielle s'exprime de la manière suivante :

$$\boxed{u(t) = u(t_0) + \int_{t_0}^{t} f(u(t'),e(t'))\mathrm{d}t'}$$

La résolution de l'équation différentielle passe donc par le calcul d'une intégrale de la fonction $f(u(t),e(t))$. Comme nous l'avons vu dans la première partie il est possible de calculer numériquement avec Python la valeur approchée d'une intégrale en utilisant des algorithmes simples. C'est ce que nous allons faire pour obtenir les valeurs approchées $\tilde{u}_k$.

Remarque : la variable d'intégration est notée t' pour éviter toute confusion avec la date t à laquelle on veut calculer la solution et qui se trouve dans les bornes d'intégration. Une variable d'intégration est une variable **muette** ; on pourrait utiliser arbitrairement n'importe quel symbole pour l'écrire.

2.3 Méthode d'Euler explicite

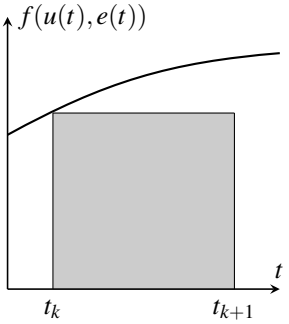
La résolution numérique de l'équation différentielle s'effectue de manière **itérative**. Connaissant la condition initiale $\tilde{u}_0 = u(t_i)$ on effectue une approximation pour calculer $\tilde{u}_1$ puis $\tilde{u}_2$ et ainsi de suite jusqu'à $\tilde{u}_N$ grâce à une **relation de récurrence** que nous allons expliciter dans ce paragraphe.

On appelle **méthode d'Euler explicite** l'algorithme qui détermine une solution approchée de l'équation différentielle en utilisant la **méthode des rectangles à gauche** pour calculer l'intégrale de la fonction $f(u(t),e(t))$.

Voici comment obtenir la relation de récurrence ; la relation **exacte** entre $u(t_{k+1})$ et $u(t_k)$ est :

$$u(t_{k+1}) - u(t_k) = \int_{t_k}^{t_{k+1}} f(u(t),e(t))\mathrm{d}t$$

La méthode des rectangles à gauche consiste à remplacer la fonction $f(u(t),e(t))$ par une fonction constante égale à $f(u(t_k),e(t_k))$ (sa valeur "à gauche" de l'intervalle $[t_k,t_{k+1}]$, voir figure ci-contre). On peut alors écrire **de manière approchée** :



$$\tilde{u}_{k+1} - \tilde{u}_k = f(\tilde{u}_k,e(t_k)) \times (t_{k+1}-t_k) = \delta t \times f(\tilde{u}_k,e(t_k))$$

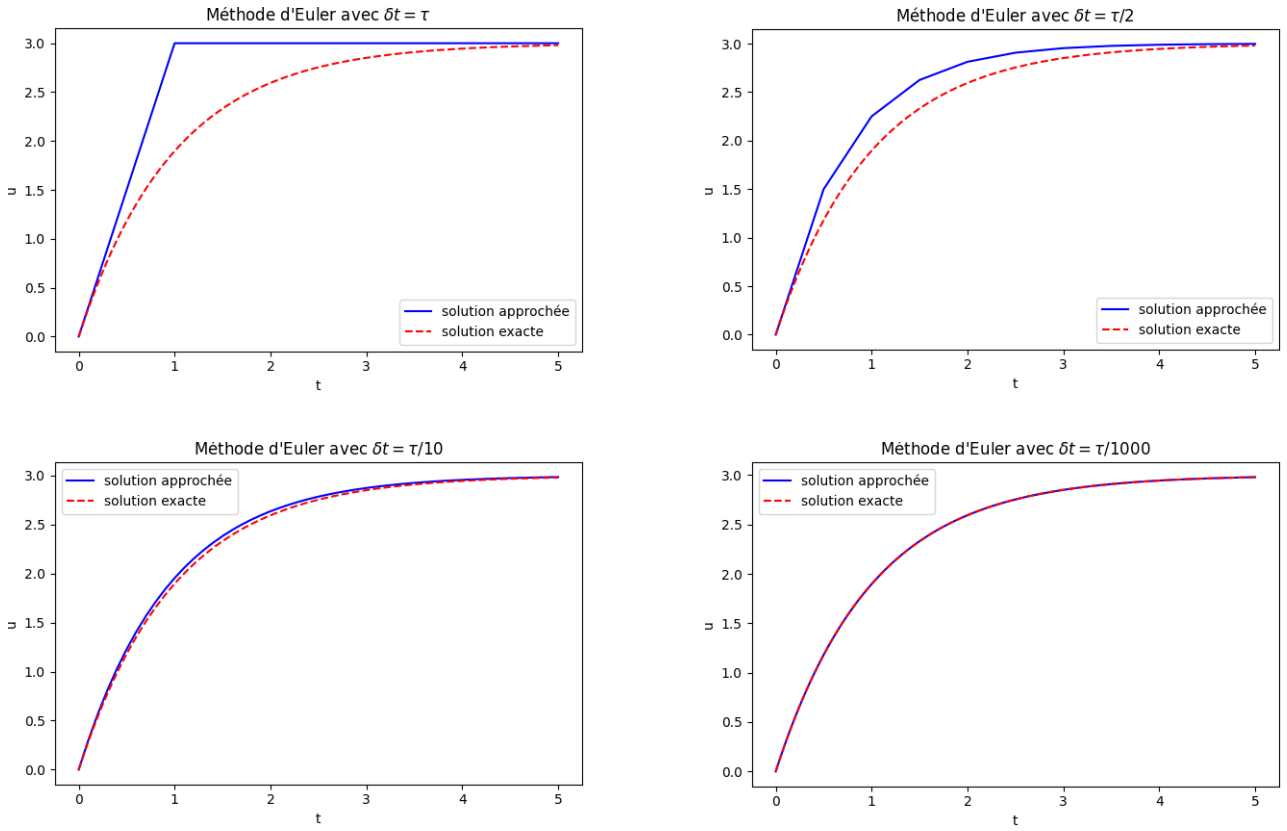
La relation de récurrence est donc la suivante :

$$\boxed{\tilde{u}_{k+1} = \tilde{u}_k + \delta t \times f(\tilde{u}_k,e(t_k))}$$

Cette relation est valable quelle que soit la fonction f à intégrer. La relation de récurrence permet donc, à partir de la condition initiale connue $\tilde{u}_0 = u(t_i)$, de déterminer toutes les valeurs suivantes de la solution approchée.

Plus le pas de discrétisation δt est faible et plus la solution numérique $\tilde{u}$ s'approche de la solution exacte u, mais en contrepartie le temps de calcul augmente.

Vous pouvez retenir sur l'exemple du circuit RC série que le calcul numérique est une excellente approximation si jamais **le pas de discrétisation δt est très faible comparé au temps caractéristique τ**. On choisira typiquement $\delta t \sim \tau/1000$ pour obtenir un résultat précis avec un temps de calcul modéré.



Remarque : les graphes ci-dessus ont été tracés dans le cas où la fonction $e(t)$ est un échelon de tension montant (voir cours sur la charge d'un circuit RC série). On peut toutefois appliquer la méthode d'Euler quelle que soit l'allure de la tension $e(t)$ délivrée par le générateur. Les valeurs données sur les axes sont dans des unités arbitraires.

Remarque : la méthode d'intégration que l'on vient de présenter s'applique à **une équation différentielle du premier ordre**. Nous verrons dans un prochain TD Python qu'il est possible de l'adapter pour résoudre une équation différentielle du deuxième ordre.

Remarque : pour minimiser l'erreur de calcul il est souvent préférable de ne pas découper l'intervalle $[t_i, t_f]$ de manière régulière. La façon de choisir les t_k pour que le calcul soit le plus efficient possible est complexe, nous n'évoquerons pas cette question.

Remarque : il est tout à fait possible d'utiliser la méthode des rectangles à droite ou bien la méthode des trapèzes pour calculer de manière approchée l'intégrale de la fonction $f(u(t), e(t))$.

- On appelle **méthode d'Euler implicite** l'algorithme qui détermine une solution approchée de l'équation différentielle en utilisant la **méthode des rectangles à droite** pour calculer l'intégrale de la fonction $f(u(t), e(t))$.
- On appelle **méthode de Runge-Kutta d'ordre 2** l'algorithme qui détermine une solution approchée de l'équation différentielle en utilisant la **méthode des trapèzes** pour calculer l'intégrale de la fonction $f(u(t), e(t))$.

Nous ne détaillerons pas ces méthodes mais retenez que la méthode de Runge-Kutta d'ordre 2 est beaucoup plus précise que la méthode d'Euler (explicite ou implicite) ; en effet la solution approchée $\tilde{u}$ converge beaucoup plus vite vers la solution exacte u lorsque le pas de discrétisation $\delta t \to 0$ (autrement dit pour une même précision elle nécessite un temps de calcul bien plus faible).

Travail préparatoire

On présente ci-dessous un programme Python incomplet destiné à calculer la tension $u(t)$ aux bornes du condensateur dans un circuit RC série, dans le cas où la tension $e(t)$ est un échelon montant :

$$e(t) = \begin{cases} 0 & t < 0 \\ E & t > 0 \end{cases}$$

avec $E = \text{Cste} = 3\,\text{V}$. On résout cette équation différentielle pour $t \in [0, 5\tau]$, avec la condition initiale $u(0) = 0$.

```
1  E = 3                    # f.e.m. du générateur (en V)
2  u0 = 0                   # tension initiale
3  tau = 1                  # valeur de la constante de temps (unité arbitraire)
4  t_i, t_f = 0, 5 * tau    # intervalle de calcul
5  dt = tau / 1000          # pas de discrétisation
6
7  t =                      # tableau des dates de calcul
8
9  # Initialisation de la solution approchée
10 u = [u0]
11
12 # Calcul de la solution approchée par récurrence
13 for k in range(  ):
14     u.append(  )
```

7. Compléter la ligne 7 qui définit le tableau des dates de calcul allant de `ti` à `tf` avec un pas de discrétisation `dt`. Il n'est pas nécessaire que `tf` soit inclue dans le tableau.

8. Montrer que pour $t > 0$ la relation de récurrence qui permet de calculer la solution approchée est :

$$\tilde{u}_{k+1} = \left(1 - \frac{\delta t}{\tau}\right)\tilde{u}_k + \frac{\delta t}{\tau}E$$

9. Compléter les lignes 13 et 14 qui permettent de calculer l'ensemble des $\tilde{u}_k$ pour les dates du tableau `t`.

Indications : Pour définir la taille du `range()` on pourra utiliser le fait que `len(t)` renvoie la taille du tableau `t`, autrement dit son nombre d'éléments.

On montre ci-dessous des exemples d'utilisation de `len()` (fonctionne aussi bien pour les listes que pour les tableaux numpy) :

```
>>> A = [0, 3, 2, 9]
>>> B = [12, 24, 224]
>>> C = np.linspace(1, 2, 100)
>>> print(len(A), len(B), len(C))
4 3 100
```

La fonction `u.append(a)` rajoute l'élément `a` à la fin de la liste `u`. Exemple :

```
>>> A = [2, 5, 3, 7]
>>> A.append(8)
>>> print(A)
[2, 5, 3, 7, 8]
```