Informatique Pour Tous

Cours 10 – Dérivation et intégration

L'outil informatique permet de modéliser le comportement de systèmes physiques en s'appuyant sur des équations différentielles (plus généralement, des équations aux dérivées partielles). Cette nouvelle séquence a pour objectif de vous familiariser avec les outils de base et les notations classiques de la modélisation de principes physiques. Ce chapitre sera très utile pour l'intégration des équations différentielles par la méthode d'Euler que l'on étudiera dans les prochains chapitres

A / Notations pour ce cours

Dans ce cours, nous supposons un intervalle $[t_{\min}, t_{\max}] \subset \mathbb{R}$ sur lequel sont définies deux fonctions.

A.1 / Une fonction y continue

En pratique, nous supposons dans ce cours une fonction

$$y : \begin{cases} [t_{\min}, t_{\max}] \to \mathbb{R} \\ t \longrightarrow y(t) \end{cases}$$ de classe $\mathcal{C}^1$ (dérivable).

→ **Informatiquement :** une telle fonction peut typiquement être définie par un code du type :

```
def y(t) :
    return ...
```

A.2 / Une fonction y discrète

Le plus souvent, nous ne travaillerons par avec des fonctions continues mais des fonctions discrètes, qui peuvent notamment être l'échantillonnage d'une fonction continue (car les capteurs, cartes électroniques, ports de communication, etc. fonctionnent tous en numérique, donc avec des grandeurs échantillonnées, cf. cours de physique de PT en électronique).

A.2.1 / Abscisses

Soit un ensemble discret comptant *N* points : $(t_k)_{0 \le k \le N-1}$

Par habitude et simplicité, on définira généralement une suite croissante (t_k) avec :

- $t_0 = t_{\min}$
- $t_{N-1} = t_{\max}$
- et $\forall k \in [\![1, N-1]\!]$, $t_{k+1} > t_k$

Généralement, on ajoutera même l'hypothèse qu'on discrétise l'intervalle avec **un pas de temps** constant, c'est-à-dire :

$$\forall k \in [\![1, N-1]\!],\ t_{k+1} - t_k = \delta t \qquad \text{càd} \qquad \forall k \in [\![0, N-1]\!], t_k = t_{\min} + k \cdot \delta t$$

(formulation implicite) (formulation explicite)

Remarque : dans les applications de type « temps réel », ce **pas de temps** δt (constant) sera souvent appelé :

période d'échantillonnage

Exemple 1 : Écrivons les lignes de code permettant de définir une liste `L_t` comptant `N_pts` points, régulièrement espacés entre `t_min` et `t_max`.

```
L_t = [t_min]                              # ou L_t = []
delta_t = (t_max - t_min)/(N_pts - 1)      # (pas de temps)
for k in range(1, N_pts) :                 # ou for k in range(N_pts)
    L_t.append( L_t[-1] + delta_t)         # ou     L_t.append( t_min + k*delta_t)
```

Exemple 2 : idem si on s'autorise l'utilisation de *Numpy* (bibliothèque pensée justement pour la modélisation physique) :

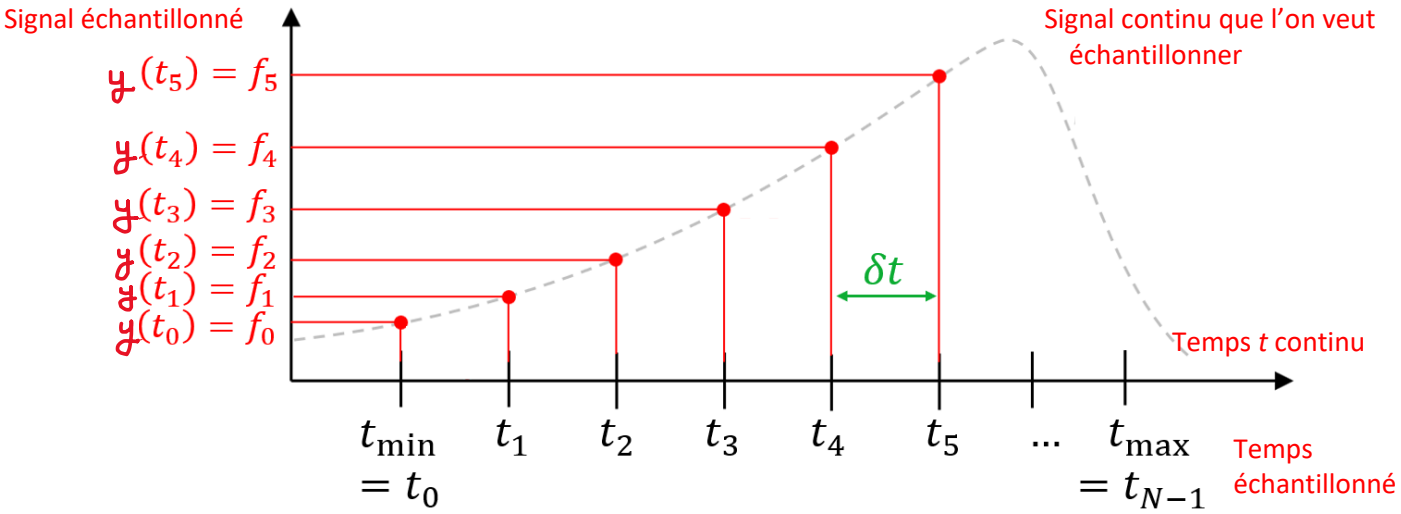
```
import numpy as np
V_t = np.linspace(t_min, t_max, N_pts)
```

A.2.2 / Ordonnées

On peut désormais définir la suite $(y_k)_{0\leq k\leq N-1}$ comptant autant de points que la suite $(t_k)_{0\leq k\leq N-1}$, avec :

$$\forall k \in [\![0, N-1]\!], y_k = y(t_k)$$

Remarque : la représentation d'une fonction à l'aide de `Matplotlib` s'appuie en pratique sur ce type de discrétisation. C'est-à-dire qu'on représente un nuage de points constitué des N points : $(t_k\ , y_k)$:



B / Dérivation

B.1 / Introduction à la dérivation : dérivée à droite

En mathématiques, si l'on a une fonction dérivable $f : \begin{cases} \mathbb{R} \longrightarrow \mathbb{R} \\ x \to f(x) \end{cases}$ on définit la valeur de la dérivée $f'(x_0)$ comme

$$f'(x_0) = \lim_{\varepsilon \to 0} \frac{f(x_0+\varepsilon) - f(x_0)}{\cancel{x_0} + \varepsilon - \cancel{x_0}}$$

C'est le taux d'accroissement de la droite tangente à $f(x)$ en $x \approx x_0$.

Application : Pour notre fonction y (de classe $\mathcal{C}^1$) définie en partie **A.1**, alors $\forall t \in]t_{\min}, t_{\max}]$ la dérivée est :

$$\frac{dy}{dt}(t) = \lim_{\tau \to 0} \frac{y(t+\tau) - y(t)}{\tau}$$

→ **Informatiquement :** on peut alors définir un code type d'une fonction :

```
def derivee( y, t) :          # une fonction peut prendre en argument une fonction !
    tau = 1E-10               # exemple d'une valeur très petite
    return (y(t+tau) - y(t))/ tau              # on renvoie ici un float
```

Cependant, comme nous l'avons vu en partie **A.2**, le plus souvent nous disposons plutôt d'une fonction discrétisée sous forme d'itérable $(y_k)_{0\leq k\leq N-1}$, sur un temps discret $(t_k)_{0\leq k\leq N-1}$.

Dans ce cas, si les pas de temps $t_{k+1}-t_k$ sont suffisamment rapprochés, on peut approximer que :

$$\frac{dy}{dt}(t_k) \approx \frac{y(t_{k+1})-y(t_k)}{t_{k+1}-t_k}$$

Remarque : il faudra faire attention, lors de la création de la liste des $\left(\frac{dy}{dt}(t_k)\right)$, aux bornes dans la boucle `for`. En effet,

$\frac{dy}{dt}(t_{N-1})$ n'est pas défini (car y_{t_N} n'existe pas).

La suite $\left(\frac{dy}{dt}(t_k)\right)_{0\leq k\leq N-2}$ compte donc $N-1$ termes, c'est-à-dire un de moins que les listes (y_k) et (t_k).

→ Il faudra donc être attentif si l'on veut tracer $\left(\frac{dy}{dt}(t_k)\right)$ en fonction de (t_k) car les 2 listes ne font pas la même taille.

B.2 / Dérivée à gauche

Reprenons, pour la suite de cette partie **B**, l'hypothèse d'un pas de temps constant.

Avec la **dérivée à droite** (qui calcule la dérivée en t_k en faisant appel à la valeur de $y_{k+1}=y(t_{k+1})$), nous aurions alors :

$$\frac{dy}{dt}(t_k) \approx \frac{y_{k+1}-y_k}{\delta t}$$

Si, ce qui est courant en Sciences de l'Ingénieur, nous travaillons sur un système piloté ou simulé en **temps réel**, il est impossible de connaître $y_{k+1}=y(t_{k+1})$ puisque t_{k+1} est l'instant futur.

➔ On privilégiera alors la **dérivée à gauche** (qui calcule la dérivée en t_k en faisant appel à la valeur de y_{k-1} , c'est-à-dire à l'instant précédent).

$$\frac{dy}{dt}(t_k) \approx \frac{y(t_k)-y(t_{k-1})}{t_k-t_{k-1}} = \frac{y_k-y_{k-1}}{\delta t}$$

Remarque : attention, cette fois la suite $\left(\frac{dy}{dt}(t_k)\right)$ sera définie pour $1\leq k\leq N-1$ (toujours $N-1$ termes).

B.3 / Dérivée centrée

La différence entre la définition mathématique (appel à la limite) et l'**approximation** de la dérivée en utilisant le pas de temps est très importante si le pas de temps δt, qui est fini (non epsilonesque) n'est pas suffisamment petit. Notamment, la dérivée $\frac{dy}{dt}(t_k)$ **estimée** par l'approximation à gauche, ou à droite, ne donne pas la même valeur. Selon que la courbe est concave ou convexe, l'une donnera une surestimation et l'autre une sous-estimation de la dérivée.

Une approximation plus juste de la dérivée est de faire appel à la dérivée centrée :

$$\frac{dy}{dt}(t_k) \approx \frac{y(t_{k+1})-y(t_{k-1})}{t_{k+1}-t_{k-1}} = \frac{y_{k+1}-y_{k-1}}{2\,\delta t}$$

Remarque 1 : notez qu'on a $\frac{1}{2}\left[\frac{y_{k+1}-y_k}{\delta t}+\frac{y_k-y_{k-1}}{\delta t}\right] = \frac{1}{2\,\delta t}[y_{k+1}-y_k+y_k-y_{k-1}] = \frac{y_{k+1}-y_{k-1}}{2\,\delta t}$

➔ donc la dérivée centrée est la moyenne entre la dérivée à gauche et la dérivée à droite.

Remarque 2 : attention, cette fois la suite $\left(\frac{dy}{dt}(t_k)\right)$ sera définie pour $1\leq k\leq N-2$ ($N-2$ termes).

B.4 / Dérivée seconde (ici estimée à droite)

Ce que nous venons de voir, dans les parties **B.1** à **B.3** s'applique aussi aux dérivées seconde, troisième, etc.

À titre d'introduction, imaginons que l'on souhaite estimer la dérivée seconde **à droite** : $\frac{d^2y}{dt^2}(t_k)$

- Par définition, puisque $\frac{d^2y}{dt^2} = \frac{d}{dt}\left[\frac{dy}{dt}\right]$, on a (dérivée à droite) : $\frac{d^2y}{dt^2}(t_k) \approx \frac{\frac{dy}{dt}(t_{k+1}) - \frac{dy}{dt}(t_k)}{\delta t}$
- Or (dérivée à droite) : $\frac{dy}{dt}(t_k) \approx \frac{y_{k+1} - y_k}{\delta t}$ et (dérivée à droite) : $\frac{dy}{dt}(t_{k+1}) \approx \frac{y_{k+2} - y_{k+1}}{\delta t}$
- D'où : $\frac{d^2y}{dt^2}(t_k) \approx \frac{1}{\delta t}\left(\frac{y_{k+2} - y_{k+1}}{\delta t} - \frac{y_{k+1} - y_k}{\delta t}\right) = \frac{y_{k+2} - 2y_{k+1} + y_k}{\delta t^2}$

Rmq : attention, cette fois la suite $\left(\frac{d^2y}{dt^2}(t_k)\right)$ sera définie pour $0 \leq k \leq N-3$ ($N-2$ termes).

B.5 / Exemple en *Python*

On veut écrire une fonction `derivee_g(L_Y, L_t)` prenant en argument une liste $(y_k = y(t_k))$ et une liste (t_k), toutes deux de même longueur, et renvoyant la liste des $\left(\frac{dy}{dt}(t_k)\right)$ approximée **à gauche**.
On admet que le pas de temps $t_k - t_{k-1}$ est constant.

```
def derivee_g(L_y, L_t) :
    delta_t = L_t[1] - L_t[0]          # car pas de temps supposé constant
    L_deriv = []
    for k in range(1, len(L_y) ) :
        y_prime = (L_y[k] - L_y[k-1] / delta_t
                                       # si pas de temps non constant, …/(L_t[k]-L_t[k-1])
        L_deriv.append( y_prime )
    return L_deriv
```

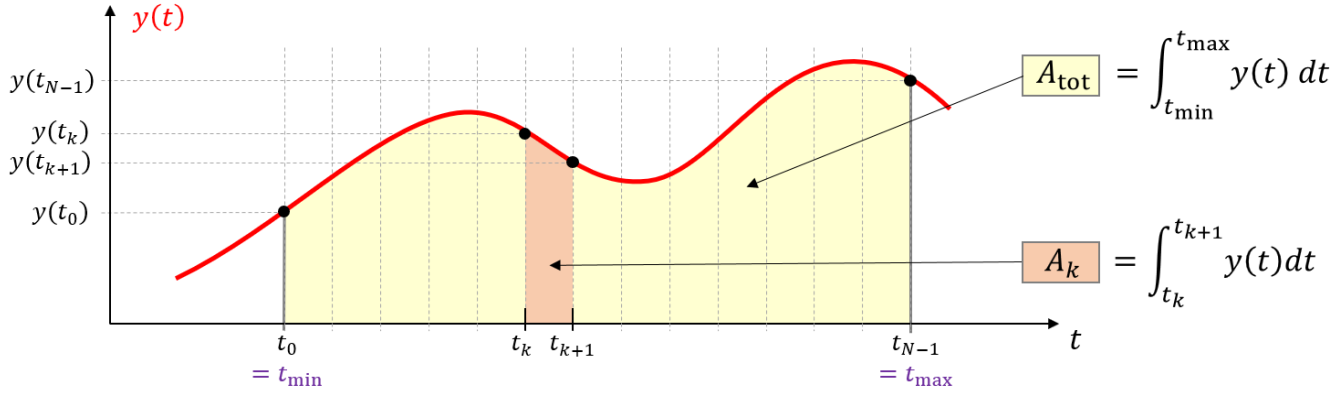
C / Intégrale et primitive

C.1 / Formulation

Soit une fonction y (qu'elle soit continue ou discrète !) dont on souhaite **estimer** l'intégrale entre 2 bornes :

$$A_{\text{tot}} = \int_{t_{\min}}^{t_{\max}} y(t)\, dt$$

On peut découper l'axe des abscisses en intervalles, comme illustré ci-dessous :



Si on a découpé le segment $[t_{\min}, t_{\max}]$ en une liste $(t_k)_{0 \le k \le N-1}$ (N points), il y a $N-1$ intervalles $[t_k, t_{k+1}]$.

En notant $A_k = \int_{t_k}^{t_{k+1}} y(t)dt$ l'intégrale sur le $k^{\text{ième}}$ intervalle, on a :

$$A_{\text{tot}} = \sum_{k=0}^{N-2} \int_{t_k}^{t_{k+1}} y(t)dt = \sum_{k=0}^{N-2} A_k$$

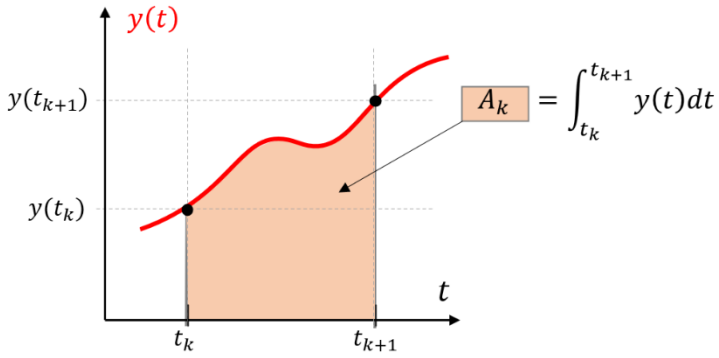
(somme de $N-1$ termes)

Remarque : ceci est une égalité stricte, exacte. Ce n'est pas une approximation. Ce qui nous reste à faire c'est à estimer chacun des A_k et c'est là que nous allons faire des approximations.

C.2 / Trois estimations possibles pour les A_k

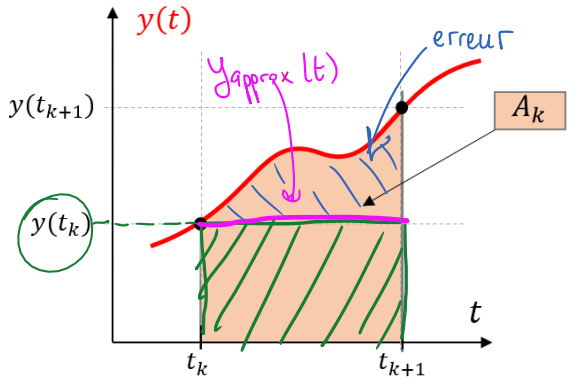
Nous venons de définir A_k comme étant $\int_{t_k}^{t_{k+1}} y(t)dt$

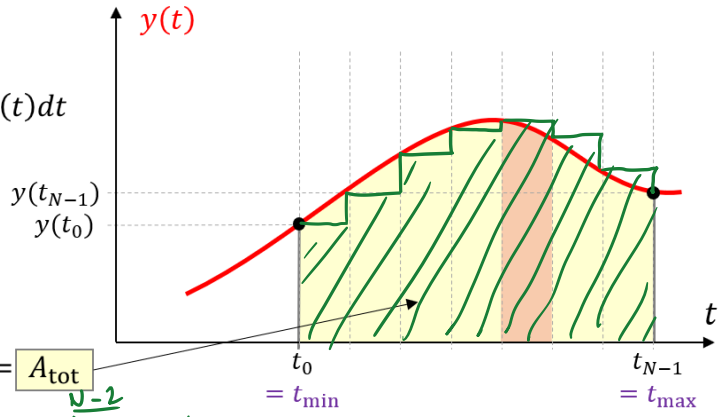
Que $y(t)$ soit définie de manière continue sur $[t_k, t_{k+1}]$, ou bien de manière discrète en $y_k = y(t_k)$ et $y_{k+1} = y(t_{k+1})$, nous allons approximer A_k.



C.2.1 / rectangles à gauche

Si on suppose que $y(t) \approx y(t_k)$ sur tout l'intervalle $[t_k, t_{k+1}]$, alors $A_k \approx y_k \times (t_{k+1} - t_k)$



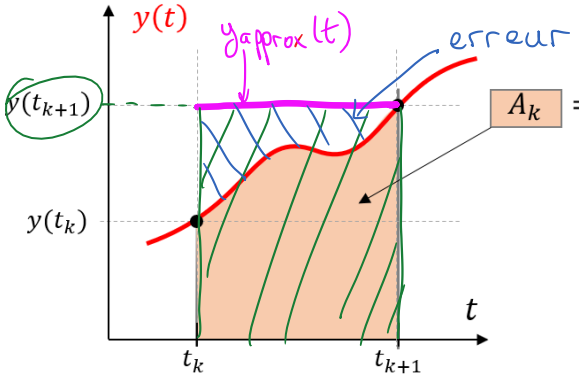


$$\sum A_k = A_{\text{tot}} \approx \sum_{k=0}^{N-2} y_k(t_{k+1} - t_k)$$

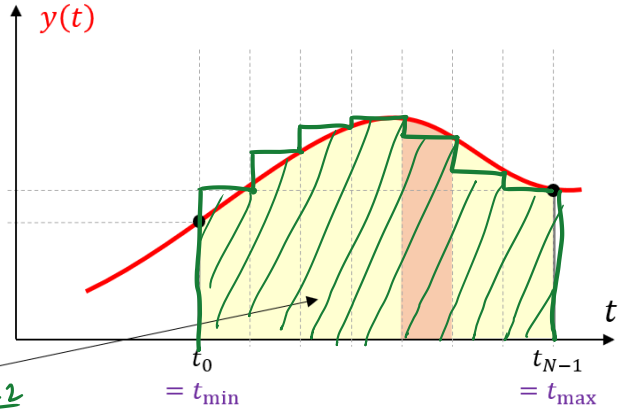
Si sort de la somme si pas de temps cst

C.2.2 / rectangles à droite

Si on suppose que $y(t) \approx y(t_{k+1})$ sur tout l'intervalle $[t_k, t_{k+1}]$, alors $A_k \approx y_{k+1} \times (t_{k+1} - t_k)$



$$A_k = \int_{t_k}^{t_{k+1}} y(t)dt$$

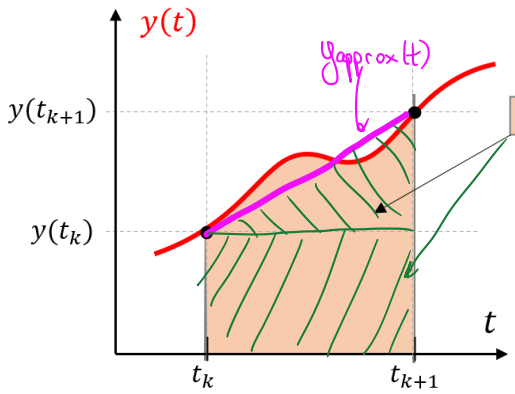


$$\sum A_k = A_{tot} \approx \sum_{k=0}^{N-2} y_{k+1}(t_{k+1} - t_k)$$

δt sort de la somme si pas de temps cst

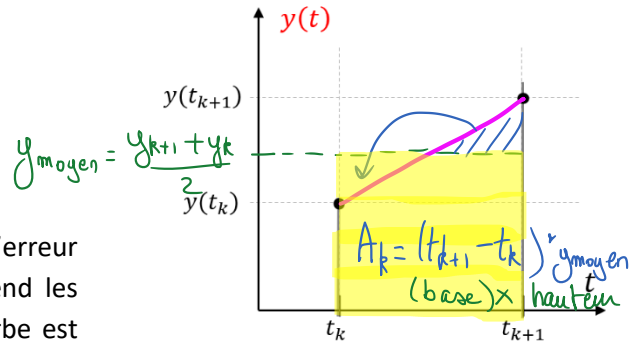
C.2.3 / méthode des trapèze

Si on suppose que $y(t)$ varie de façon affine entre t_k et t_{k+1} , cela donne :



$$A_k = (t_{k+1} - t_k)\, y_k + \frac{1}{2}(t_{k+1} - t_k) y_{k+1}$$

$$= (t_{k+1} - t_k)\frac{y_{k+1} + y_k}{2}$$

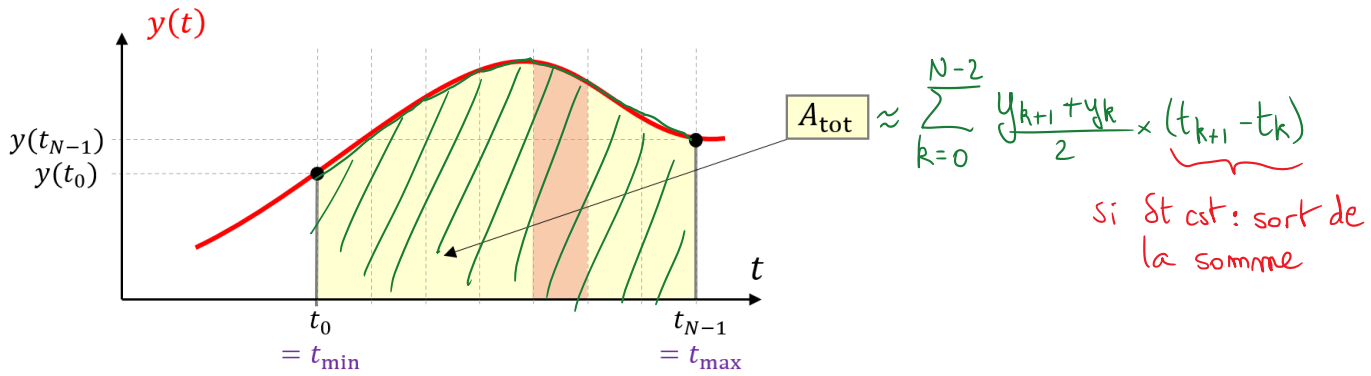


Rmq 1 : on peut aussi voir, de façon équivalente :

Rmq 2 : Sur les deux schémas des parties **C.2.1** et **C.2.2** on voit que l'erreur d'approximation de A_k n'a pas le même signe selon qu'on prend les rectangles à gauche ou à droite, qui peuvent (selon que la courbe est croissante ou décroissante) sur-évaluer ou sous-évaluer A_k.

➔ Mais remarquons alors que $A_{k\ \text{trapèze}} = \frac{A_{k\ \text{rect gauche}} + A_{k\ \text{rect droite}}}{2}$.

Conclusion : La méthode des trapèzes permet donc de compenser un peu les erreurs d'approximation des deux méthodes. Elle permet donc d'approximer une intégrale assez précisément avec moins de points (donc un pas de discrétisation δt plus grossier) que la méthode des rectangles.



C. 3 / Calcul d'une primitive

Si l'on veut déterminer une primitive Y d'une fonction y, on se ramène à un calcul intégral, puisque

$$Y(T) = \int_{t_{min}}^{T} y(t)dt + \text{cste}$$

Le plus souvent, on aura une fonction y discrète, décrite sous forme de liste $(y_k)_{0\leq k\leq N-1}$ avec une discrétisation sur une liste $(t_k)_{0\leq k\leq N-1}$, et l'on voudra une liste des $\big(Y_k = Y(t_k)\big)_{0\leq k\leq N-1}$.
Supposons qu'on nous fixe une condition initiale $Y(t_{min}) = Y_{init}$ qui nous servira pour définir la constante d'intégration.

Commençons par traduire la condition initiale :

$$Y_0 = Y(t_{min}) = \int_{t_{min}}^{t_{min}} y(t)dt + \text{cste} = \text{cste} = Y_{init}$$

Puis, pour simplifier le calcul, on peut remarquer la relation de récurrence suivante :

$$Y_{k+1} = \int_{t_{min}}^{t_{k+1}} y(t)dt + Y_{init} = \int_{t_{min}}^{t_k} y(t)dt + \int_{t_k}^{t_{k+1}} y(t)dt + Y_{init} = Y_k + A_k$$

➔ On obtient donc une relation de récurrence simple, et les termes A_k dépendent du type d'approximation (méthode des rectangles ou des trapèzes).

Exemple : On dispose d'une liste de vitesses `L_V` $= \big(V(t_k)\big)$ et d'une liste des temps correspondants `L_T` $= (t_k)$.
On veut renvoyer la liste des positions `L_X` $= \big(x(t_k)\big)$ correspondante, avec comme condition initiale que $x(t_0) = X_{init}$
On prendra la méthode des trapèzes.

```
def primitive( L_V, L_T, X_init) :

    N_pts = len(L_T)        # taille de L_V = taille de L_T

    L_X = [X_init]          # dans une récurrence, toujours initialiser ainsi

    for k in range(N-1) :

        Ak = ((L_V[k+1] + L_V[k]) / 2) * (L_T[k+1] - L_T[k]) # trapèzes

        L_X.append( L_X[-1] + Ak )

    return L_X
```