In [1]:

```
from IPython.core.display import HTML
css_file = './custom.css'
HTML(open(css_file, "r").read())
```

Out[1]:

M62_CM4 : schémas "classiques" à un pas

Considérons le problème de Cauchy

> trouver une fonction y: $I \subset \mathbb{R} \to \mathbb{R}$ définie sur un intervalle $I = [t_0, T]$ telle que
> $$\begin{cases} y'(t) = \varphi(t, y(t)), & \forall t \in I = [t_0, T], \\ y(t_0) = y_0, \end{cases}$$
> avec y_0 une valeur donnée et supposons que l'on ait montré l'existence et l'unicité d'une solution y pour $t \in I$.

Pour $h > 0$ soit $t_n \equiv t_0 + nh$ avec $n = 0, 1, 2, \dots, N_h$ une suite de $N_h + 1$ nœuds de I induisant une discrétisation de I en N_h sous-intervalles $I_n = [t_n; t_{n+1}]$ chacun de longueur $h > 0$ (appelé le *pas de discrétisation*).

Pour chaque nœud t_n, on cherche la valeur inconnue u_n qui approche la valeur exacte $y_n \equiv y(t_n)$.

- L'ensemble de $N_h + 1$ valeurs $\{t_0, t_1 = t_0 + h, \dots, t_{N_h} = T\}$ représente les points de la discrétisation.
- L'ensemble de $N_h + 1$ valeurs $\{y_0, y_1, \dots, y_{N_h}\}$ représente la solution exacte.
- L'ensemble de $N_h + 1$ valeurs $\{u_0 = y_0, u_1, \dots, u_{N_h}\}$ représente la solution numérique. Cette solution approchée sera obtenue en construisant une suite récurrente.

Les schémas qu'on va construire permettent de calculer (explicitement ou implicitement) u_{n+1} à partir de $u_n, u_{n-1}, \dots, u_{n-k}$ et il est donc possible de calculer successivement u_1, u_2,..., en partant de u_0 par une formule de récurrence de la forme

$$\begin{cases} u_0 = y_0, \\ u_{n+1} = \Phi(u_{n+1}, u_n, u_{n-1}, \dots, u_{n-k}), & \forall n \in \mathbb{N}. \end{cases}$$

Méthodes explicites et méthodes implicites
Une méthode est dite *explicite* si la valeur u_{n+1} peut être calculée directement à l'aide des valeurs précédentes u_k, $k \leq n$ (ou d'une partie d'entre elles).
Une méthode est dite *implicite* si u_{n+1} n'est défini que par une relation implicite faisant intervenir la fonction φ.

Méthodes à un pas et méthodes multi-pas
Une méthode numérique pour l'approximation du problème de Cauchy est dite *à un pas* si pour tout $n \in \mathbb{N}$, u_{n+1} ne dépend que de u_n et éventuellement de lui-même.
Autrement, on dit que le schéma est une méthode *multi-pas* (ou à pas multiples).

Table of Contents

1 Construction de schémas à un pas

Si nous intégrons l'EDO $y'(t) = \varphi(t, y(t))$ entre t_n et t_{n+1} nous obtenons

$$y_{n+1} - y_n = \int_{t_n}^{t_{n+1}} \varphi(t, y(t))\mathrm{d}t.$$

On peut construire différentes schémas selon la formule d'approximation utilisée pour approcher le membre de droite. Cette solution approchée sera obtenue en construisant une suite récurrente comme suit:

$$\begin{cases} u_0 = y_0, \\ u_{n+1} = u_n + \displaystyle\int_{t_n}^{t_{n+1}} \text{un polynôme d'interpolation de } \varphi(t, u)\mathrm{d}t. \end{cases}$$

1.1 Schéma d'Euler explicite

Si on remplace une fonction f par une constante égale à la valeur de f en la borne gauche de l'intervalle $[a; b]$ (polynôme qui interpole f en le point $(a, f(a))$ et donc de degré 0), on a

$$\tilde{f}(x) = f(a)$$

$$\int_a^b f(x)\mathrm{d}x \approx \int_a^b \tilde{f}(x)\mathrm{d}x = (b-a)f(a).$$

Cette formule est dite *formule de quadrature du rectangle à gauche*.
En utilisant cette formule pour approcher la fonction $t \mapsto \varphi(t, y(t))$ on a

$$\int_{t_n}^{t_{n+1}} \varphi(t, y(t))\mathrm{d}t \approx h\varphi(t_n, y(t_n))$$

et on obtient le **schéma d'Euler progressif**

$$\begin{cases} u_0 = y(t_0) = y_0, \\ u_{n+1} = u_n + h\varphi(t_n, u_n) \quad n = 0, 1, 2, \dots N_h - 1 \end{cases}$$

Il s'agit d'un schéma à 1 pas explicite car il permet d'expliciter u_{n+1} en fonction de u_n.

In [2]:

```
def euler_progressif(phi,tt):
    h=tt[1]-tt[0]
    uu = [y0]
    for i in range(len(tt)-1):
        uu.append(uu[i]+h*phi(tt[i],uu[i]))
    return uu
```

1.2 Schéma d'Euler implicite

Si on remplace une fonction f par une constante égale à la valeur de f en la borne droite de l'intervalle $[a; b]$ (polynôme qui interpole f en le point $(b, f(b))$ et donc de degré 0), on a

$$\tilde{f}(x) = f(a)$$

$$\int_a^b f(x)\mathrm{d}x \approx \int_a^b \tilde{f}(x)\mathrm{d}x = (b-a)f(b).$$

Cette formule est dite *formule de quadrature du rectangle à droite*.
En utilisant cette formule pour approcher la fonction $t \mapsto \varphi(t, y(t))$ on a

$$\int_{t_n}^{t_{n+1}} \varphi(t, y(t))\mathrm{d}t \approx h\varphi(t_{n+1}, y(t_{n+1}))$$

et on obtient le **schéma d'Euler rétrograde**

$$\begin{cases} u_0 = y(t_0) = y_0, \\ u_{n+1} = u_n + h\varphi(t_{n+1}, u_{n+1}) \quad n = 0, 1, 2, \dots N_h - 1 \end{cases}$$

Il s'agit d'un schéma à 1 pas implicite car il ne permet pas d'expliciter directement u_{n+1} en fonction de u_n lorsque la fonction φ n'est pas triviale. Pour calculer u_{n+1} il faudra utiliser un schéma pour le calcul du zéro d'une fonction quelconque.

In [3]:

```
def euler_regressif(phi,tt)
    h = tt[1]-tt[0]:
    uu = [y0]
    for i in range(len(tt)-1):
        uu.append( fsolve(lambda x: -x+uu[i]+h*phi(tt[i+1],x),
    return uu
```

```
  File "<ipython-input-3-23235508af11>", line 1
    def euler_regressif(phi,tt)
                               ^
SyntaxError: invalid syntax
```

1.3 Schéma d'Euler modifié

Si on remplace une fonction f par une constante égale à la valeur de f au milieu de l'intervalle $[a;b]$ (polynôme qui interpole f en le point $\left(\frac{a+b}{2}, f\left(\frac{a+b}{2}\right)\right)$ et donc de degré 0), on a

$$\tilde{f}(x) = f\left(\frac{a+b}{2}\right)$$

$$\int_a^b f(x)\mathrm{d}x \approx \int_a^b \tilde{f}(x)\mathrm{d}x = (b-a)f\left(\frac{a+b}{2}\right).$$

Cette formule est dite *formule de quadrature du rectangle* ou *du point milieu*.

En utilisant cette formule pour approcher la fonction $t \mapsto \varphi(t, y(t))$ on a

$$\int_{t_n}^{t_{n+1}} \varphi(t, y(t))\mathrm{d}t \approx h\varphi\left(t_n + \frac{h}{2}, y\left(t_n + \frac{h}{2}\right)\right)$$

et on obtient

$$\begin{cases} u_0 = y(t_0) = y_0, \\ u_{n+1} = u_n + h\varphi\left(t_n + \frac{h}{2}, u_{n+1/2}\right) \quad n = 0, 1, 2, \dots N_h - 1 \end{cases}$$

où $u_{n+1/2}$ est une approximation de $y(t_n + h/2)$. Nous pouvons utiliser une prédiction d'Euler progressive sur un demi-pas pour approcher le $u_{n+1/2}$ dans le terme $\varphi(t_n + h/2, u_{n+1/2})$ par $\tilde{u}_{n+1/2} = u_n + (h/2)\varphi(t_n, u_n)$. Nous avons construit ainsi un nouveau schéma appelé **schéma d'Euler modifié** qui s'écrit

$$\begin{cases} u_0 = y(t_0) = y_0, \\ \tilde{u}_{n+1/2} = u_n + (h/2)\varphi(t_n, u_n), \\ u_{n+1} = u_n + h\varphi\left(t_n + \frac{h}{2}, \tilde{u}_{n+1/2}\right) \quad n = 0, 1, 2, \dots N_h - 1 \end{cases}$$

Il s'agit d'un schéma à 1 pas explicite car il permet d'expliciter u_{n+1} en fonction de u_n.

In [4]:

```
def euler_modifie(phi,tt):
    h=tt[1]-tt[0]
    uu = [y0]
    for i in range(len(tt)-1):
        uu.append( uu[i]+h*phi(tt[i]+h/2.,uu[i]+0.5*h*phi(tt[i]
    return uu
```

1.4 Schéma du trapèze ou de Crank-Nicolson

Si on remplace une fonction f par le segment qui relie $(a,f(a))$ à $(b,f(b))$ (polynôme qui interpole f en les points $(a,f(a))$ et $(b,f(b))$ et donc de degré 1), on a

$$\tilde{f}(x) = \frac{f(b)-f(a)}{b-a}(x-a)+f(a)$$

$$\int_a^b f(x)\mathrm{d}x \approx \int_a^b \tilde{f}(x)\mathrm{d}x = \frac{b-a}{2}(f(a)+f(b))\,.$$

Cette formule est dite *formule de quadrature du trapèze*.
En utilisant cette formule pour approcher la fonction $t \mapsto \varphi(t,y(t))$ on a

$$\int_{t_n}^{t_{n+1}} \varphi(t,y(t))\mathrm{d}t \approx \frac{h}{2}(\varphi(t_n,y(t_n))+\varphi(t_{n+1},y(t_{n+1})))$$

et on obtient le **schéma du trapèze ou de Crank-Nicolson**

$$\begin{cases} u_0 = y(t_0) = y_0, \\ u_{n+1} = u_n + \frac{h}{2}\varphi(t_n,u_n) + \frac{h}{2}\varphi(t_{n+1},u_{n+1}) \quad n = 0,1,2,\dots N_h-1 \end{cases}$$

Il s'agit à nouveau d'un schéma à 1 pas implicite car il ne permet pas d'expliciter directement u_{n+1} en fonction de u_n lorsque la fonction φ n'est pas triviale. En fait, ce schéma fait la moyenne des schémas d'Euler progressif et rétrograde.

In [5]:

```
def CN(phi,tt):
    h=tt[1]-tt[0]
    uu = [y0]
    for i in range(len(tt)-1):
        uu.append( fsolve(lambda x: -x+uu[i]+0.5*h*( phi(tt[i]
    return uu
```

1.5 Schéma de Heun

Pour éviter le calcul implicite de u_{n+1} dans le schéma du trapèze, nous pouvons utiliser une prédiction d'Euler progressive et remplacer le u_{n+1} dans le terme $\varphi(t_{n+1},u_{n+1})$ par $\tilde{u}_{n+1} = u_n + h\varphi(t_n,u_n)$. Nous avons construit ainsi un nouveau schéma appelé **schéma de Heun**. Plus précisément, la méthode de Heun s'écrit

$$\begin{cases} u_0 = y(t_0) = y_0, \\ \tilde{u}_{n+1} = u_n + h\varphi(t_n, u_n), \\ u_{n+1} = u_n + \frac{h}{2}\varphi(t_n, u_n) + \frac{h}{2}\varphi(t_{n+1}, \tilde{u}_{n+1}) \quad n = 0, 1, 2, \dots N_h - 1 \end{cases}$$

Il s'agit à nouveau d'un schéma à 1 pas explicite.

In [6]:

```
def heun(phi,tt):
    h=tt[1]-tt[0]
    uu = [y0]
    for i in range(len(tt)-1):
        k1 = phi( tt[i], uu[i] )
        k2 = phi( tt[i+1], uu[i] + k1  )
        uu.append( uu[i] + 0.5*h*(k1+k2) )
    return uu
```

1.6 Schéma de Simpson implicite

Si on remplace une fonction f par la parabole segment qui passe par $(a, f(a))$, $(b, f(b))$ et $\left(\frac{a+b}{2}, f\left(\frac{a+b}{2}\right)\right)$ (polynôme qui interpole f en les points $(a, f(a))$, $(b, f(b))$ et $\left(\frac{a+b}{2}, f\left(\frac{a+b}{2}\right)\right)$ et donc de degré 2), on a

$$\tilde{f}(x) = f(a)\frac{(x-b)\left(x-\frac{a+b}{2}\right)}{(a-b)\left(a-\frac{a+b}{2}\right)} + f\left(\frac{a+b}{2}\right)\frac{(x-a)(x-b)}{\left(\frac{a+b}{2}-a\right)\left(\frac{a+b}{2}-b\right.}$$

$$\int_a^b f(x)\mathrm{d}x \approx \int_a^b \tilde{f}(x)\mathrm{d}x = \frac{h}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right).$$

Cette formule est dite *formule de Simpson*.
En utilisant cette formule pour approcher la fonction $t \mapsto \varphi(t, y(t))$ on a

$$\int_{t_n}^{t_{n+1}} \varphi(t, y(t))\mathrm{d}t \approx \frac{h}{6}\left(\varphi(t_n, y(t_n)) + 4\varphi\left(t_n + \frac{h}{2}, y\left(t_n + \frac{h}{2}\right)\right) + q\right.$$

et on obtient

$$\begin{cases} u_0 = y(t_0) = y_0, \\ u_{n+1} = u_n + \frac{h}{6}\left(\varphi(t_n, u_n) + 4\varphi\left(t_n + \frac{h}{2}, u_{n+1/2}\right) + \varphi(t_{n+1}, u_{n+1})\right) \end{cases}$$

où $u_{n+1/2}$ est une approximation de $y(t_n + h/2)$. Nous pouvons utiliser une prédiction d'Euler progressive sur un demi-pas pour approcher le $u_{n+1/2}$ dans le terme $\varphi(t_n + h/2, u_{n+1/2})$ par $\tilde{u}_{n+1/2} = u_n + (h/2)\varphi(t_n, u_n)$. Nous avons construit ainsi un nouveau schéma qu'on appellera **schéma de Simpson implicite** qui s'écrit

$$\begin{cases} u_0 = y(t_0) = y_0, \\ \tilde{u}_{n+1/2} = u_n + (h/2)\varphi(t_n, u_n), \\ u_{n+1} = u_n + \frac{h}{6}\left(\varphi(t_n, u_n) + 4\varphi\left(t_n + \frac{h}{2}, \tilde{u}_{n+1/2}\right) + \varphi(t_{n+1}, u_{n+1})\right) \quad r \end{cases}$$

Il s'agit d'un schéma à 1 pas implicite car il ne permet pas d'expliciter u_{n+1} en fonction de u_n.

1.7 Schéma de Simpson explicite

Pour éviter le calcul implicite de u_{n+1} dans le schéma de Simpson implicite, nous pouvons utiliser une prédiction d'Euler progressive et remplacer le u_{n+1} dans le terme $\varphi(t_{n+1}, u_{n+1})$ par $\check{u}_{n+1} = u_n + h\varphi(t_n, u_n)$. Nous avons construit ainsi un nouveau schéma qu'on appellera **schéma de Simpson explicite** et qui s'écrit

$$\begin{cases} u_0 = y(t_0) = y_0, \\ \tilde{u}_{n+1/2} = u_n + \frac{h}{2}\varphi(t_n, u_n), \\ \check{u}_{n+1} = u_n + h\varphi(t_n, u_n), \\ u_{n+1} = u_n + \frac{h}{6}\left(\varphi(t_n, u_n) + 4\varphi\left(t_n + \frac{h}{2}, \tilde{u}_{n+1/2}\right) + \varphi(t_{n+1}, \check{u}_{n+1})\right) \end{cases} r$$

Il s'agit à nouveau d'un schéma à 1 pas explicite.

1.8 Schéma de Simpson explicite-variante

Notons qu'on aurait pu remplacer le u_{n+1} dans le terme $\varphi(t_{n+1}, u_{n+1})$ par une approximation utilisant $\tilde{u}_{n+1/2}$ comme par exemple une prédiction d'Euler progressive à partir de $t_n + h/2$, ce qui donne $\hat{u}_{n+1} = \tilde{u}_{n+1/2} + \frac{h}{2}\varphi(t_n + h/2, \tilde{u}_{n+1/2})$:

$$\begin{cases} u_0 = y(t_0) = y_0, \\ \tilde{u}_{n+1/2} = u_n + \frac{h}{2}\varphi(t_n, u_n), \\ \hat{u}_{n+1} = \tilde{u}_{n+1/2} + \frac{h}{2}\varphi(t_n + h/2, \tilde{u}_{n+1/2}), \\ u_{n+1} = u_n + \frac{h}{6}\left(\varphi(t_n, u_n) + 4\varphi\left(t_n + \frac{h}{2}, \tilde{u}_{n+1/2}\right) + \varphi(t_{n+1}, \hat{u}_{n+1})\right) \end{cases} r$$

La méthode d'Euler explicite construit la suite

$$\begin{cases} u_0 = y_0 = 0, \\ u_{n+1} = u_n + h\varphi(t_n, u_n) = u_n + hu_n^{1/2}, \quad n = 0, 1, 2, \dots N_h - 1 \end{cases}$$

par conséquent $u_n = 0$ pour tout n. La méthode d'Euler explicite approche la solution constante $y(t) =$ pour tout $t \in \mathbb{R}^+$.

La méthode d'Euler implicite construit la suite

$$\begin{cases} u_0 = y_0 = 0, \\ u_{n+1} = u_n + h\varphi(t_{n+1}, u_{n+1}) = u_n + hu_{n+1}^{1/2}, \quad n = 0, 1, 2, \dots N_h - 1 \end{cases}$$

par conséquent $u_0 = 0$ mais u_1 dépend de la méthode de résolution de l'équation implicite $x = 0 + h\sqrt{x}$. Bien sur $x = 0$ est une solution mais $x = h^2$ est aussi solution. Si le schéma choisit $u_1 = h^2$, alors $u_n > 0$ pour tout $n \in \mathbb{N}^*$.

Notons que le problème de Cauchy avec une CI $y(0) = y_0 > 0$ admet une et une seule solution, la fonction $y(t) = \frac{1}{4}(t - 2\sqrt{y_0})^2$. Dans ce cas, les deux schémas approchent forcement la même solution.

1.9 Remarques

1. À première vue, il semble que le schéma d'Euler progressif et le schéma de Heun soient préférable au schéma d'Euler rétrograde et de Crank-Nicolson puisque ces derniers ne sont pas explicites. Cependant, on verra que les méthodes d'Euler implicite et de Crank-Nicolson sont inconditionnellement A-stables. C'est aussi le cas de nombreuses autres méthodes implicites. Cette propriété rend les méthodes implicites attractives, bien qu'elles soient plus coûteuses que les méthodes explicites.
2. Pour la mise en application d'un schéma il faut aussi prendre en compte l'influence des erreurs d'arrondi. En effet, afin de minimiser l'erreur globale théorique, on pourrait être tenté d'appliquer une méthode avec un pas très petit, par exemple de l'ordre de 10^{-16}, mais ce faisant, outre que le temps de calcul deviendrait irréaliste, très rapidement les erreurs d'arrondi feraient diverger la solution approchée. En pratique il faut prendre h assez petit pour que la méthode converge assez rapidement, mais pas trop petit non plus pour que les erreurs d'arrondi ne donnent pas lieu à des résultats incohérent et pour que les calculs puissent être effectués en un temps raisonnable.
3. Pour vérifier nos calculs d'interpolation puis intégration, nous pouvons utiliser le package de calcul formel `SymPy` :

In [7]:

```
import sympy as symb
symb.init_printing()

symb.var('phi_np1,phi_n,phi_npm,')
symb.var('h,t,t_n')
t_np1=t_n+h
t_npm=t_n+h/2

p=symb.interpolate([(t_n,phi_n)], t)
EE=(symb.integrate(p,(t,t_n,t_np1)).simplify())
print("EE:")
display(EE)
print("\n")

p=symb.interpolate([(t_np1,phi_np1)], t)
EI=(symb.integrate(p,(t,t_n,t_np1)).simplify())
print("EI:")
display(EI)
print("\n")

p=symb.interpolate([(t_npm,phi_npm)], t)
PM=(symb.integrate(p,(t,t_n,t_np1)).simplify())
print("Point Milieu :")
display(PM)
print("\n")


p=symb.interpolate([(t_np1,phi_np1),(t_n,phi_n)], t)
CN=(symb.integrate(p,(t,t_n,t_np1)).simplify())
print("CN:")
display(CN)
print("\n")


p=symb.interpolate([(t_np1,phi_np1),(t_npm,phi_npm),(t_n,phi_n
SI=(symb.integrate(p,(t,t_n,t_np1)).simplify())
print("Simpson:")
display(SI)
print("\n")
```

EE:

$h\phi_n$

EI:

$h\phi_{np1}$

Point Milieu :

$h\phi_{npm}$

CN:

$$\frac{h}{2}\left(\phi_n + \phi_{np1}\right)$$

Simpson:

$$\frac{h}{6}\left(\phi_n + \phi_{np1} + 4\phi_{npm}\right)$$

In [ ]: