In [1]:

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

Out[1]:

M62_CM3 Introduction à l'approximation numérique d'EDO

On ne peut expliciter des solutions analytiques que pour des équations différentielles ordinaires très particulières. Par exemple :

- dans certains cas, on ne peut exprimer la solution que sous forme implicite. C'est le cas par exemple de l'EDO $y'(t) = \dfrac{y(t)-t}{y(t)+t}$ dont les solutions vérifient la relation implicite
$$\frac{1}{2}\ln(t^2+y^2(t)) + \arctan\left(\frac{y(t)}{t}\right) = C,$$
où C est une constante arbitraire.
- dans d'autres cas, on ne parvient même pas à représenter la solution sous forme implicite.
C'est le cas par exemple de l'EDO $y'(t) = e^{-t^2}$ dont les solutions ne peuvent pas s'écrire comme composition de fonctions élémentaires.

Pour ces raisons, on cherche des méthodes numériques capables d'approcher la solution de toutes les équations différentielles qui admettent une solution.

Considérons le problème de Cauchy:

> trouver une fonction y: $I \subset \mathbb{R} \to \mathbb{R}$ définie sur un intervalle I 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 $n = 0, 1, 2, \ldots, N_h$ 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.

Table of Contents

1 Construction des méthodes d'Euler explicite et implicite

Une méthode classique, la **méthode d'Euler explicite** (ou *progressive*, de l'anglais *forward*), consiste à construire une solution numérique ainsi

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

Cette méthode est obtenue en considérant l'équation différentielle en chaque nœud t_n et en remplaçant la dérivée exacte $y'(t_n)$ par le taux d'accroissement

$$y'(t_n) \simeq \frac{y(t_{n+1}) - y(t_n)}{h}.$$

De même, en utilisant le taux d'accroissement

$$y'(t_{n+1}) \simeq \frac{y(t_{n+1}) - y(t_n)}{h}$$

pour approcher $y'(t_{n+1})$, on obtient la **méthode d'Euler implicite** (ou *rétrograde*, de l'anglais *backward*)

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

Ces deux méthodes sont dites à un pas: pour calculer la solution numérique u_{n+1} au nœud t_{n+1}, on a seulement besoin des informations disponibles au nœud précédent t_n. Plus précisément, pour la méthode d'Euler progressive, u_{n+1} ne dépend que de la valeur u_n calculée précédemment, tandis que pour la méthode d'Euler rétrograde, u_{n+1}

dépend aussi "de lui-même" à travers la valeur de $\varphi(t_{n+1}, u_{n+1})$. C'est pour cette raison que la méthode d'Euler progressive est dite explicite tandis que la méthode d'Euler rétrograde est dite implicite. Les méthodes implicites sont plus coûteuses que les méthodes explicites car, si la fonction φ est non linéaire, un problème non linéaire doit être résolu à chaque temps t_{n+1} pour calculer u_{n+1}. Néanmoins, nous verrons que les méthodes implicites jouissent de meilleures propriétés de stabilité que les méthodes explicites.

2 Implémentation des schémas d'Euler explicite et implicite

Voyons un exemple complet: considérons le problème de Cauchy

> trouver la fonction $y: I \subset \mathbb{R} \to \mathbb{R}$ définie sur l'intervalle $I = [0, 1]$ telle que
> $$\begin{cases} y'(t) = 2ty(t), & \forall t \in I = [0, 1], \\ y(0) = 1 \end{cases}$$
> dont la solution est $y(t) = e^{t^2}$.

On commence par importer les modules `math` et `matplotlib` ainsi que la fonction `fsolve` du module `scipy.optimize` pour résoudre les équations implicites présentes dans les schémas implicites:

In [2]:

```
from math import *
from matplotlib.pylab import *
from scipy.optimize import fsolve
```

On initialise le problème de Cauchy

In [3]:

```
t0 = 0.
tfinal = 1.
y0 = 1.
```

On définit l'équation différentielle : `phi` est une fonction python qui contient la fonction mathématique $\varphi(t, y) = 2ty$ dépendant des variables t et y.

In [4]:

```
phi = lambda t,y : 2.*y*t
```

On introduit la discrétisation: les nœuds d'intégration $[t_0, t_1, \dots, t_N]$ sont contenus dans le vecteur `tt`

In [5]:

```
N = 8
tt=linspace(t0,tfinal,N+1)
```

On écrit les schémas numériques : les valeurs $[u_0, u_1, \dots, u_N]$ pour chaque méthode sont contenues dans le vecteur `uu` .

Schéma d'Euler progressif :

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

In [6]:

```
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
```

Schéma d'Euler régressif :

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

Attention : u_{n+1} est solution de l'équation $x = u_n + h\varphi(t_{n+1}, x)$, c'est-à-dire un zéro de la fonction (en générale non linéaire)

$$x \mapsto -x + u_n + h\varphi(t_{n+1}, x)$$

In [7]:

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

On calcule les solutions approchées:

In [8]:

```
uu_ep = euler_progressif(phi,tt)
uu_er = euler_regressif(phi,tt)
```

Comme on la connait, on calcul la solution exacte pour calculer les erreurs:

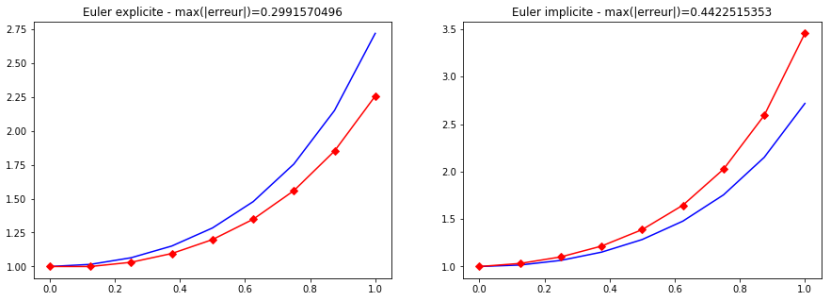
In [9]:

```
sol_exacte = lambda t : y0*math.exp(t**2)
yy    = [sol_exacte(t) for t in tt]
```

On compare les graphes des solutions exacte (en bleu) et approchées (en rouge) et on affiche le maximum de l'erreur:

In [10]:

```
figure(1, figsize=(15, 5))
subplot(1,2,1)
plot(tt,yy,'b-',tt,uu_ep,'r-D')
title('Euler explicite - max(|erreur|)=%1.10f'%(max([abs(uu_ep

subplot(1,2,2)
plot(tt,yy,'b-',tt,uu_er,'r-D')
title('Euler implicite - max(|erreur|)=%1.10f'%(max([abs(uu_er
```



3 Convergence des schémas d'Euler

Une méthode numérique est *convergente* si

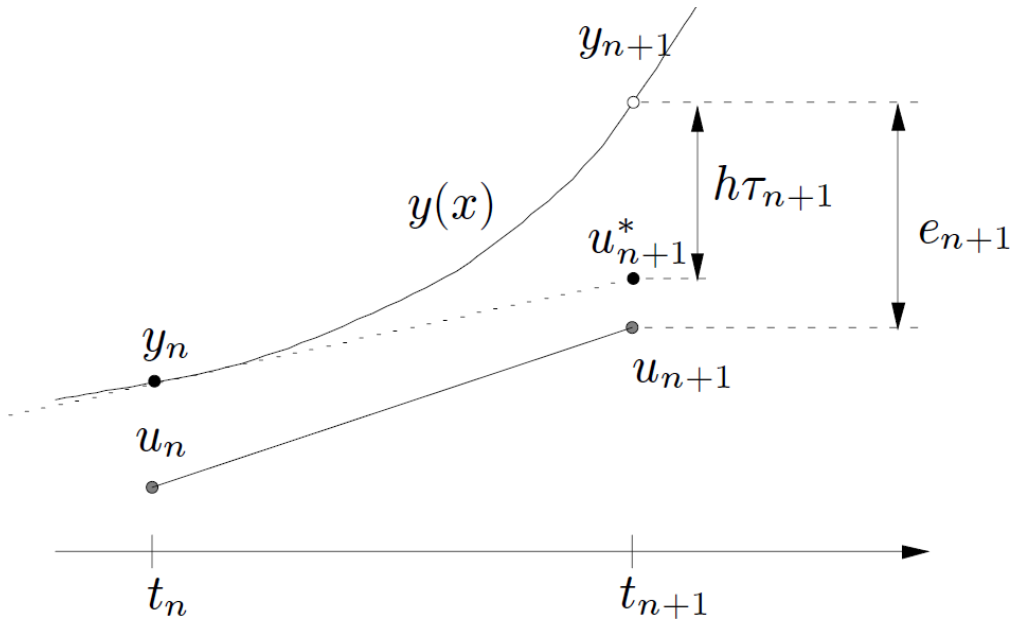
$$|y_n - u_n| \leq C(h) \qquad \forall n = 0, \dots, N_h$$

où $C(h)$ tend vers zéro quand h tend vers zéro.
Si $C(h) = \mathcal{O}(h^p)$ pour $p > 0$, on dit que la convergence de la méthode est d'ordre p.

Soit u^*_{n+1} la solution numérique au temps t_{n+1} qu'on obtiendrait en insérant de la solution exacte dans le schéma (par exemple, pour la méthode d'Euler explicite on a $u^*_{n+1} \equiv y_n + h\varphi(t_n, y_n)$). Pour vérifier qu'une méthode converge, on écrit l'erreur ainsi

$$e_n \equiv y_n - u_n = (y_n - u^*_n) + (u^*_n - u_n).$$

Si les deux termes $(y_n - u^*_n)$ et $(u^*_n - u_n)$ tendent vers zéro quand $h \to 0$ alors la méthode converge.



La quantité

$$\tau_{n+1}(h) \equiv \frac{y_{n+1} - u^*_{n+1}}{h}$$

est appelée **erreur de troncature locale**. Elle représente (à un facteur $1/h$ près) l'erreur qu'on obtient en insérant la solution exacte dans le schéma numérique.

L'**erreur de troncature globale** (ou plus simplement l'erreur de troncature) est définie par

$$\tau(h) = \max_{n=0,\ldots,N_h} |\tau_n(h)|.$$

Si $\lim_{h\to 0} \tau(h) = 0$ on dit que **la méthode est consistante**. On dit qu'elle est consistante d'ordre p si $\tau(h) = \mathcal{O}(h^p)$ pour un certain $p \geq 1$.

Remarque: la propriété de consistance est nécessaire pour avoir la convergence. En effet, si elle n'était pas consistante, la méthode engendrerait à chaque itération une erreur qui ne tendrait pas vers zéro avec h. L'accumulation de ces erreurs empêcherait l'erreur globale de tendre vers zéro quand $h \to 0$.

3.1 La méthode d'Euler explicite est convergente d'ordre 1.

On étudie séparément l'erreur de consistance et l'accumulation de ces erreurs.

- **Terme $y_n - u_n^*$.**
  Il représente l'erreur engendrée par une seule itération de la méthode d'\textsc{Euler} explicite. En supposant que la dérivée seconde de y existe et est continue, on écrit le développement de Taylor de y au voisinage de t_n:
  $$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(\eta_n)$$
  où η_n est un point de l'intervalle $]t_n\,;\,t_{n+1}[$. Donc il existe $\eta_n \in]t_n, t_{n+1}[$ tel que
  $$y_{n+1} - u_{n+1}^* = y_{n+1} - \Big(y_n + h\varphi(t_n, y_n)\Big) = y_{n+1} - y_n - hy'(t_n) =$$
  L'erreur de troncature de la méthode d'Euler explicite est donc de la forme
  $$\tau(h) = M\frac{h}{2}, \qquad M \equiv \max_{t\in[t_0,T]} |y''(t)|.$$
  On en déduit que $\lim_{h\to 0} \tau(h) = 0$: la méthode est consistante.
- **Terme $u_{n+1}^* - u_{n+1}$.**
  Il représente la propagation de t_n à t_{n+1} de l'erreur accumulée au temps précédent t_n. On a
  $u_{n+1}^* - u_{n+1} = (y_n + h\varphi(t_n, y_n)) - (u_n + h\varphi(t_n, u_n)) = e_n + h\,(\varphi($
  Comme φ est lipschitzienne par rapport à sa deuxième variable, on a
  $$|u_{n+1}^* - u_{n+1}| \le (1 + hL)|e_n|.$$
- **Convergence.**
  Comme $e_0 = 0$, les relations précédentes donnent
  $$\begin{aligned}
  |e_n| &\le |y_n - u_n^*| + |u_n^* - u_n| \\
  &\le h|\tau_n(h)| + (1 + hL)|e_{n-1}| \\
  &\le h|\tau_n(h)| + (1 + hL)\,(h|\tau_{n-1}(h)| + (1 + hL)|e_{n-2}|)\,| \\
  &\le \left(1 + (1 + hL) + \cdots + (1 + hL)^{n-1}\right) h\tau(h) \\
  &= \left(\sum_{i=0}^{n-1}(1 + hL)^i\right) h\tau(h) \\
  &= \frac{(1 + hL)^n - 1}{hL}h\tau(h) \\
  &\le \frac{(e^{hL})^n - 1}{hL}h\tau(h) \qquad \text{car } (1 + x) \le e^x \\
  &= \frac{(e^{hL})^{(t_n - t_0)/h} - 1}{L}\tau(h) \qquad \text{car } t_n - t_0 = nh \\
  &= \frac{e^{L(t_n - t_0)} - 1}{L}\tau(h) \\
  &= \frac{e^{L(t_n - t_0)} - 1}{L}\frac{M}{2}h
  \end{aligned}$$
  On peut conclure que la méthode d'Euler explicite est convergente d'ordre 1.

On remarque que l'ordre de cette méthode coïncide avec l'ordre de son erreur de troncature. On retrouve cette propriété dans de nombreuses méthodes de résolution numérique d'équations différentielles ordinaires.

Remarque: l'estimation de convergence est obtenue en supposant seulement φ lipschitzienne. On peut établir une meilleure estimation si $\partial_y \varphi$ existe et est non positive pour tout $t \in [t_0; T]$ et tout $y \in \mathbb{R}$. En effet dans ce cas

$$\begin{aligned} u_n^* - u_n &= (y_{n-1} + h\varphi(t_{n-1}, y_{n-1})) - (u_{n-1} + h\varphi(t_{n-1}, u_{n-1})) \\ &= e_{n-1} + h\left(\varphi(t_{n-1}, y_{n-1}) - \varphi(t_{n-1}, u_{n-1})\right) \\ &= e_{n-1} + h\left(e_{n-1}\partial_y \varphi(t_{n-1}, \eta_n)\right) \\ &= \left(1 + h\partial_y \varphi(t_{n-1}, \eta_n)\right) e_{n-1} \end{aligned}$$

où η_n appartient à l'intervalle dont les extrémités sont y_{n-1} et u_{n-1}. Ainsi, si

$$0 < h < \frac{2}{\max\limits_{t\in[t_0,T]} \partial_y \varphi(t, y(t))}$$

alors

$$|u_n^* - u_n| \leq |e_{n-1}|.$$

On en déduit $|e_n| \leq |y_n - u_n^*| + |e_{n-1}| \leq nh\tau(h) + |e_0|$et donc

$$|e_n| \leq M\frac{h}{2}(t_n - t_0).$$

La restriction sur le pas de discrétisation h est une condition de stabilité, comme on le verra dans la suite.

3.2 Étude empirique de la convergence

Considérons le même problème de Cauchy.
On se propose d'estimer l'ordre de convergence des méthodes d'**Euler**.

Pour chaque schéma, on calcule la solution approchée avec différentes valeurs de $h_k = 1/N_k$, à savoir $1/2$, $1/4$, $1/8$, ..., $1/1024$ (ce qui correspond à différentes valeurs de $N_k = 2^{k+1}$, à savoir 2, 2^2, 2^3, ... 2^{10}). On sauvegarde les valeurs de h_k dans le vecteur `H`.

Pour chaque valeur de h_k, on calcule le maximum de la valeur absolue de l'erreur et on sauvegarde toutes ces erreurs dans le vecteur `err_schema` de sort que `err_schema[k]` contient $e_k = \max_{i=0,\ldots,N_k} |y(t_i) - u_i|$ avec $N_k = 2^{k+1}$.

In [11]:

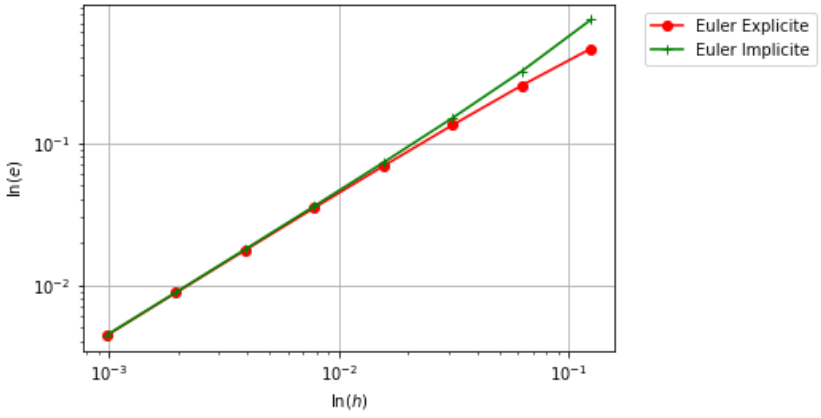
```
H = []
err_ep = []
err_er = []

for k in range(8):
    N = 2**(k+3)
    tt=linspace(t0,tfinal,N+1)
    h = tt[1]-tt[0]
    yy = [sol_exacte(t) for t in tt]
    uu_ep   = euler_progressif(phi,tt)
    uu_er   = euler_regressif(phi,tt)
    H.append(h)
    err_ep.append( max([abs(uu_ep[i]-yy[i]) for i in range(len
    err_er.append( max([abs(uu_er[i]-yy[i]) for i in range(len
```

```
/home/minnolina/anaconda3/lib/python3.6/site-pack
ages/scipy/optimize/minpack.py:161: RuntimeWarnin
g: The iteration is not making good progress, as
measured by the
  improvement from the last ten iterations.
  warnings.warn(msg, RuntimeWarning)
```

Pour afficher l'ordre de convergence on affiche les points (`h[k]` , `err_ep[k]`) en echèlle logarithmique: on représente $\ln(h)$ sur l'axe des abscisses et $\ln(\text{err})$ sur l'axe des ordonnées. Le but de cette représentation est clair: si $\text{err} = Ch^p$ alors $\ln(\text{err}) = \ln(C) + p\ln(h)$. En échelle logarithmique, p représente donc la pente de la ligne droite $\ln(\text{err})$.

In [12]:

```
loglog(H,err_ep, 'r-o',label='Euler Explicite')
loglog(H,err_er, 'g-+',label='Euler Implicite')
xlabel('$\ln(h)$')
ylabel('$\ln(e)$')
legend(bbox_to_anchor=(1.04,1),loc='upper left')
grid(True)
```



Pour estimer l'ordre de convergence on doit estimer la pente de la droite

qui relie l'erreur au pas k à l'erreur au pas $k+1$ en echelle logarithmique. Pour estimer la pente globale de cette droite (par des moindres carrés) on peut utiliser la fonction `polyfit` basée sur la régression linéaire.

In [13]:

```
print ('Euler progressif %1.2f' %(polyfit(log(H),log(err_ep),
print ('Euler regressif %1.2f' %(polyfit(log(H),log(err_er), 1
```

```
Euler progressif 0.96
Euler regressif 1.04
```