In [1]:

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

Out[1]:

# M62 TP 5 - Étude de la convergence

Vous pouvez commencer à exécuter un serveur de notebook en écrivant dans un terminal :

`jupyter notebook &`

Cela imprimera des informations dans votre console et ouvrira un navigateur Web.
La page d'arrivée est le tableau de bord qui affiche les notebook actuellement disponibles (par défaut, le répertoire à partir duquel le serveur du bloc-notes a été démarré).

Vous pouvez

- soit créer un nouveau notebook à partir du tableau de bord avec le bouton (en haut à droite) `Nouveau`
- soit ouvrir un notebook existant en cliquant sur leur nom (par exemple, vous pouvez télécharger le notebook de présentation ici (http://nbviewer.jupyter.org/url/faccanoni.univ-tln.fr/user/enseignements/20182019/M62-CM1.ipynb).)

Pour ce TP, on importera tous les modules necessaires comme suit:

In [2]:

```
%reset -f
%matplotlib inline
%autosave 300

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

```
Autosaving every 300 seconds
```

# Table of Contents



# Étude de la convergence

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
> $$\begin{cases} y'(t) = y(t), & \forall t \in I = [0, 1], \\ y(0) = 1 \end{cases}$$
> dont la solution est $y(t) = e^t$.

On se propose d'estimer l'ordre de convergence d'une méthode numérique.

- Pour chaque schéma, on calcule la solution approchée avec différentes valeurs de $h_k = 1/N_k$, à savoir $1/8$, $1/16$, $1/32$, etc. (ce

qui correspond à différentes valeurs de $N_k = 2^{k+3}$). 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 $e_k = \max_{i=0,\dots,N_k} |y(t_i) - u_i|$ avec $N_k = 2^{k+1}$.
- Pour afficher l'ordre de convergence on utilise une échelle logarithmique, i.e. on représente $\ln(h)$ sur l'axe des abscisses et $\ln(\text{err})$ sur l'axe des ordonnées.
  En effet, 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 $\ln(\text{err})$.

Remarque: puisque la fonction $\varphi(t, y) = y$ est linéaire, toute méthode implicite peut être rendue explicite par un calcul élémentaire en explicitant directement pour chaque schéma l'expression de $u_{n+1}$. Cependant, nous pouvons utiliser le le module `SciPy` sans modifier l'implémentation des schémas (mais on payera l'ordre de convergence de `fsolve`).

Estimer l'ordre de convergence des méthodes:

1. EE, $AB_2$, $AB_3$, $AB_4$, $AB_5$, $N_2$, $N_3$, $N_4$
2. EI, CN, $AM_1$, $AM_2$, $AM_3$, $AM_4$, $BDF_2$, $BDF_3$
3. EM, Heun, RK4-1, RK4-2
4. $AM_4$-$AB_1$, $AM_4$-$AB_2$, $AM_4$-$AB_3$

**Attention: les schémas multistep ont besoin d'initialiser plusieurs pas de la suite définie pas récurrence pour pouvoir démarrer. Dans cette étude, au lieu d'utiliser un schéma d'ordre inférieur pour initialiser la suite, on utilisera la solution exacte (en effet, l'utilisation d'un schéma d'ordre inférieur dégrade l'ordre de précision).**

On écrit les schémas numériques :

- les nœuds d'intégration $[t_0, t_1, \dots, t_N]$ sont contenus dans le vecteur `tt` (qui change en fonction de `h`)
- les valeurs $[u_0, u_1, \dots, u_N]$ pour chaque méthode sont contenues dans le vecteur `uu`.

# 1 Schémas explicites

## 1.1 Schéma de Adam-Bashforth à 1 pas = 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 [3]:

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

## 1.2 Schéma de Adam-Bashforth à 2 pas

$$\begin{cases} u_0 = y_0, \\ u_1 = u_0 + h\varphi(t_0, u_0), \\ u_{n+1} = u_n + \frac{h}{2}\Big(3\varphi(t_n, u_n) - \varphi(t_{n-1}, u_{n-1})\Big) \quad n = 1, 2, 3, 4, 5, \dots N \end{cases}$$

In [4]:

```
def AB2(phi,tt):
    uu = [y0]
    uu.append(sol_exacte(tt[1]))
    for i in range(1,N):
        k1 = h * phi( tt[i], uu[i] )
        k2 = h * phi( tt[i-1], uu[i-1] )
        uu.append( uu[i] + (3.*k1-k2) /2.0 )
    return uu
```

## 1.3 Schéma de Adam-Bashforth à 3 pas

$$\begin{cases} u_0 = y_0, \\ u_1 = u_0 + h\varphi(t_0, u_0), \\ u_2 = u_1 + \frac{h}{2}\Big(3\varphi(t_1, u_1) - \varphi(t_0, u_0)\Big), \\ u_{n+1} = u_n + \frac{h}{12}\Big(23\varphi(t_n, u_n) - 16\varphi(t_{n-1}, u_{n-1}) + 5\varphi(t_{n-2}, u_{n-2})\Big) \end{cases}$$

In [5]:

```
def AB3(phi,tt):
    uu = [y0]
    uu.append(sol_exacte(tt[1]))
    uu.append(sol_exacte(tt[2]))
    for i in range(2,N):
        k1 = h * phi( tt[i], uu[i] )
        k2 = h * phi( tt[i-1], uu[i-1] )
        k3 = h * phi( tt[i-2], uu[i-2] )
        uu.append( uu[i] + (23.*k1-16.*k2+5.*k3) /12.0 )
    return uu
```

## 1.4 Schéma de Adam-Bashforth à 4 pas

$$\begin{cases} u_0 = y_0, \\ u_1 = u_0 + h\varphi(t_0, u_0), \\ u_2 = u_1 + \frac{h}{2}\Big(3\varphi(t_1, u_1) - \varphi(t_0, u_0)\Big), \\ u_3 = u_2 + \frac{h}{12}\Big(23\varphi(t_2, u_2) - 16\varphi(t_1, u_1) + 5\varphi(t_0, u_0)\Big), \\ u_{n+1} = u_n + \frac{h}{24}\Big(55\varphi(t_n, u_n) - 59\varphi(t_{n-1}, u_{n-1}) + 37\varphi(t_{n-2}, u_{n-2}) - \end{cases}$$

In [6]:

```
def AB4(phi,tt):
    uu = [y0]
    uu.append(sol_exacte(tt[1]))
    uu.append(sol_exacte(tt[2]))
    uu.append(sol_exacte(tt[3]))
    for i in range(3,N):
        k1 = h * phi( tt[i], uu[i] )
        k2 = h * phi( tt[i-1], uu[i-1] )
        k3 = h * phi( tt[i-2], uu[i-2] )
        k4 = h * phi( tt[i-3], uu[i-3] )
        uu.append( uu[i] + (55.*k1-59.*k2+37.*k3-9.*k4) /24.0
    return uu
```

## 1.5 Schéma de Adam-Bashforth à 5 pas

$$\begin{cases} u_0 = y_0, \\ u_1 = u_0 + h\varphi(t_0, u_0), \\ u_2 = u_1 + \frac{h}{2}\Big(3\varphi(t_1, u_1) - \varphi(t_0, u_0)\Big), \\ u_3 = u_2 + \frac{h}{12}\Big(23\varphi(t_2, u_2) - 16\varphi(t_1, u_1) + 5\varphi(t_0, u_0)\Big), \\ u_4 = u_3 + \frac{h}{24}\Big(55\varphi(t_3, u_3) - 59\varphi(t_2, u_2) + 37\varphi(t_1, u_1) - 9\varphi(t_0, u_0)\Big) \\ u_{n+1} = u_n + \frac{h}{720}\Big(1901\varphi(t_n, u_n) - 2774\varphi(t_{n-1}, u_{n-1}) + 2616\varphi(t_{n-2},\end{cases}$$

In [7]:

```
def AB5(phi,tt):
    uu = [y0]
    uu.append(sol_exacte(tt[1]))
    uu.append(sol_exacte(tt[2]))
    uu.append(sol_exacte(tt[3]))
    uu.append(sol_exacte(tt[4]))
    for i in range(4,N):
        k1 = h * phi( tt[i], uu[i] )
        k2 = h * phi( tt[i-1], uu[i-1] )
        k3 = h * phi( tt[i-2], uu[i-2] )
        k4 = h * phi( tt[i-3], uu[i-3] )
        k5 = h * phi( tt[i-4], uu[i-4] )
        uu.append( uu[i] + (1901.*k1-2774.*k2+2616.*k3-1274.*k
    return uu
```

## 1.6 Schéma de Nylström à 2 pas

$$\begin{cases} u_0 = y_0, \\ u_1 = u_0 + h\varphi(t_0, u_0), \\ u_{n+1} = u_{n-1} + 2h\varphi(t_n, u_n) \quad n = 1, 2, 3, 4, 5, \dots N-1 \end{cases}$$

In [8]:

```
def N2(phi,tt):
    uu = [y0]
    uu.append(sol_exacte(tt[1]))
    for i in range(1,N):
        k1 = h * phi( tt[i], uu[i] )
        uu.append( uu[i-1] + 2.0*k1 )
    return uu
```

## 1.7 Schéma de Nylström à 3 pas

$$\begin{cases} u_0 = y_0, \\ u_1 = u_0 + h\varphi(t_0, u_0), \\ u_2 = u_0 + 2h\varphi(t_1, u_1), \\ u_{n+1} = u_{n-1} + \frac{h}{3}\Big(7\varphi(t_n, u_n) - 2\varphi(t_{n-1}, u_{n-1}) + \varphi(t_{n-2}, u_{n-2})\Big) \quad n \end{cases}$$

In [9]:

```
def N3(phi,tt):
    uu = [y0]
    uu.append(sol_exacte(tt[1]))
    uu.append(sol_exacte(tt[2]))
    for i in range(2,N):
        k1 = h * phi( tt[i], uu[i] )
        k2 = h * phi( tt[i-1], uu[i-1] )
        k3 = h * phi( tt[i-2], uu[i-2] )
        uu.append( uu[i-1] + (7.*k1-2.*k2+k3)/3. )
    return uu
```

## 1.8 Schéma de Nylström à 4 pas

$$\begin{cases} u_0 = y_0, \\ u_1 = u_0 + h\varphi(t_0, u_0), \\ u_2 = u_0 + 2h\varphi(t_1, u_1), \\ u_3 = u_1 + \frac{h}{3}\Big(7\varphi(t_2, u_2) - 2\varphi(t_1, u_1) + \varphi(t_0, u_0)\Big), \\ u_{n+1} = u_{n-1} + \frac{h}{3}\Big(8\varphi(t_n, u_n) - 5\varphi(t_{n-1}, u_{n-1}) + 4\varphi(t_{n-2}, u_{n-2}) - \varphi($$

In [10]:

```
def N4(phi,tt):
    uu = [y0]
    uu.append(sol_exacte(tt[1]))
    uu.append(sol_exacte(tt[2]))
    uu.append(sol_exacte(tt[3]))
    for i in range(3,N):
        k1 = h * phi( tt[i], uu[i] )
        k2 = h * phi( tt[i-1], uu[i-1] )
        k3 = h * phi( tt[i-2], uu[i-2] )
        k4 = h * phi( tt[i-3], uu[i-3] )
        uu.append( uu[i-1] + (8.*k1-5.*k2+4.*k3-k4)/3. )
    return uu
```

## 1.9 Schéma de Runge-Kutta RK4-1

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

In [11]:

```
def RK4(phi,tt):
    uu = [y0]
    for i in range(N):
        k1 = phi( tt[i], uu[i] )
        k2 = phi( tt[i]+h/2 , uu[i]+h*k1/2 )
        k3 = phi( tt[i]+h/2 , uu[i]+h*k2/2 )
        k4 = phi( tt[i+1]    , uu[i]+h*k3 )
        uu.append( uu[i] + h*(k1+2*k2+2*k3+k4) /6 )
    return uu
```

# 2 Schémas implicites

## 2.1 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)
$\lambda : x \mapsto -x + u_n + h\varphi(t_{n+1}, x)$.

In [12]:

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

## 2.2 Schéma de Crank-Nicolson

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

Attention : $u_{n+1}$ est solution de l'équation $x = u_n + \frac{h}{2}(\varphi(t_n, u_n) + \varphi(t_{n+1}, x))$, c'est-à-dire un zéro de la fonction (en générale non linéaire) $\lambda : x \mapsto -x + u_n + \frac{h}{2}(\varphi(t_n, u_n) + \varphi(t_{n+1}, x))$

In [13]:

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

## 2.3 Schéma de AM-2

$$\begin{cases} u_0 = y_0, \\ u_1 = u_0 + \frac{h}{2}\Big(\varphi(t_1, u_1) + \varphi(t_0, u_0)\Big), \\ u_{n+1} = u_n + h\Big(5\varphi(t_{n+1}, u_{n+1}) + 8\varphi(t_n, u_n) - \varphi(t_{n-1}, u_{n-1})\Big) \quad n = \end{cases}$$

In [14]:

```
def AM2(phi,tt):
    uu = [y0]
    uu.append(sol_exacte(tt[1]))
    for i in range(1,N):
        temp = fsolve(lambda x: -x+uu[i]+h*( 5.*phi(tt[i+1],x)
        uu.append(temp)
    return uu
```

## 2.4 Schéma de AM-3

$$\begin{cases} u_0 = y_0, \\ u_1 = u_0 + \frac{h}{2}\Big(\varphi(t_1, u_1) + \varphi(t_0, u_0)\Big), \\ u_2 = u_n + h\Big(5\varphi(t_2, u_2) + 8\varphi(t_1, u_1) - \varphi(t_0, u_0)\Big), \\ u_{n+1} = u_n + h\Big(9\varphi(t_{n+1}, u_{n+1}) + 19\varphi(t_n, u_n) - 5\varphi(t_{n-1}, u_{n-1}) + \varphi(t_{n-2}, u_{n-2})\Big) \end{cases}$$

In [15]:

```
def AM3(phi,tt):
    uu = [y0]
    uu.append(sol_exacte(tt[1]))
    uu.append(sol_exacte(tt[2]))
    for i in range(2,N):
        temp = fsolve(lambda x: -x+uu[i]+h*( 9.*phi(tt[i+1],x)
        uu.append(temp)
    return uu
```

## 2.5 Schéma de AM-4

$$\begin{cases} u_0 = y_0, \\ u_1 = u_0 + \frac{h}{2}\Big(\varphi(t_1,u_1) + \varphi(t_0,u_0)\Big), \\ u_2 = u_1 + h\Big(5\varphi(t_2,u_2) + 8\varphi(t_1,u_1) - \varphi(t_0,u_0)\Big), \\ u_3 = u_2 + h\Big(9\varphi(t_3,u_3) + 19\varphi(t_2,u_2) - 5\varphi(t_1,u_1) + \varphi(t_0,u_0)\Big), \\ u_{n+1} = u_n + h\Big(251\varphi(t_{n+1},u_{n+1}) + 646\varphi(t_n,u_n) - 264\varphi(t_{n-1},u_{n-1})\end{cases}$$

In [16]:

```
def AM4(phi,tt):
    uu = [y0]
    uu.append(sol_exacte(tt[1]))
    uu.append(sol_exacte(tt[2]))
    uu.append(sol_exacte(tt[3]))
    for i in range(3,N):
        temp = fsolve(lambda x: -x+uu[i]+h*( 251.*phi(tt[i+1],
        uu.append(temp)
    return uu
```

## 2.6 Schéma de AM-5

$$\begin{cases} u_0 = y_0, \\ u_1 = u_0 + \frac{h}{2}\Big(\varphi(t_1,u_1) + \varphi(t_0,u_0)\Big), \\ u_2 = u_1 + h\Big(5\varphi(t_2,u_2) + 8\varphi(t_1,u_1) - \varphi(t_0,u_0)\Big), \\ u_3 = u_2 + h\Big(9\varphi(t_3,u_3) + 19\varphi(t_2,u_2) - 5\varphi(t_1,u_1) + \varphi(t_0,u_0)\Big), \\ u_4 = u_3 + h\Big(251\varphi(t_4,u_4) + 646\varphi(t_3,u_3) - 264\varphi(t_2,u_2) + 106\varphi(t_1 \\ u_{n+1} = u_n + h\Big(475\varphi(t_{n+1},u_{+1}) + 1427\varphi(t_n,u_n) - 798\varphi(t_{n-1},u_{n-1}) \\ \qquad\qquad + 27\varphi(t_{n-4},u_{n-4})\Big), \end{cases}$$

In [17]:

```
def AM5(phi,tt):
    uu = [y0]
    uu.append(sol_exacte(tt[1]))
    uu.append(sol_exacte(tt[2]))
    uu.append(sol_exacte(tt[3]))
    uu.append(sol_exacte(tt[4]))
    for i in range(4,N):
        temp = fsolve(lambda x: -x+uu[i]+h*( 475.*phi(tt[i+1],
        uu.append(temp)
    return uu
```

## 2.7 Schéma BDF2

$$\begin{cases} u_0 = y_0, \\ u_1 = u_0 + h\varphi(t_1,u_1), \\ u_{n+1} = \frac{4}{3}u_n - \frac{1}{3}u_{n-1} + \frac{2}{3}\varphi(t_{n+1},u_{n+1}) \quad n = 1,2,\dots N-1 \end{cases}$$

In [18]:

```
def BDF2(phi,tt):
    uu = [y0]
    uu.append(sol_exacte(tt[1]))
    for i in range(1,N):
        temp = fsolve(lambda x: -x+4./3.*uu[i]-1./3.*uu[i-1] +
        uu.append(temp)
    return uu
```

## 2.8 Schéma BDF3

$$\begin{cases} u_0 = y_0, \\ u_1 = u_0 + h\varphi(t_1, u_1), \\ u_2 = \frac{4}{3}u_1 - \frac{1}{3}u_0 + \frac{2}{3}\varphi(t_2, u_2), \\ u_{n+1} = \frac{4}{3}u_n - \frac{1}{3}u_{n-1} + \frac{2}{3}\varphi(t_{n+1}, u_{n+1}) \quad n = 2, 3, \dots N-1 \end{cases}$$

In [19]:

```
def BDF3(phi,tt):
    uu = [y0]
    uu.append(sol_exacte(tt[1]))
    uu.append(sol_exacte(tt[2]))
    for i in range(2,N):
        temp = fsolve(lambda x: -x+18./11.*uu[i]-9./11.*uu[i-1
        uu.append(temp)
    return uu
```

# 3 Schémas predicteur-correcteur

## 3.1 Schéma d'Euler modifié

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

In [20]:

```
def euler_modifie(phi,tt):
    uu = [y0]
    for i in range(N):
        k1 = h * phi( tt[i], uu[i] )
        uu.append( uu[i]+h*phi(tt[i]+h/2.,uu[i]+k1/2.) )
    return uu
```

## 3.2 Schéma de Heun

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

In [21]:

```
def heun(phi,tt):
    uu = [y0]
    for i in range(N):
        k1 = h * phi( tt[i], uu[i] )
        k2 = h * phi( tt[i+1], uu[i] + k1  )
        uu.append( uu[i] + (k1+k2) /2.0 )
    return uu
```

## 3.3 Schéma AM-2 AB-1

$$\begin{cases} u_0 = y_0, \\ u_1 = u_0 + h\varphi(t_0, u_0), \\ \tilde{u} = u_n + h\varphi(t_n, u_n), \\ u_{n+1} = u_n + \frac{h}{12}\left(5\varphi(t_{n+1}, \tilde{u}) + 8\varphi(t_n, u_n) - \varphi(t_{n-1}, u_{n-1})\right) \quad n = 1, \end{cases}$$

In [22]:

```
def AM2AB1(phi,tt):
    uu = [y0]
    uu.append(sol_exacte(tt[1]))
    for i in range(1,N):
        pred = uu[i] + h*phi(tt[i],uu[i])
        uu.append(uu[i]+h*(5.*phi(tt[i+1],pred)+8.*phi(tt[i],u
    return uu
```

## 3.4 Schéma AM-3 AB-2

$$\begin{cases} u_0 = y_0, \\ u_1 = u_0 + h\varphi(t_0, u_0), \\ u_2 = u_0 + \frac{h}{2}(3\varphi(t_1, u_1) - \varphi(t_0, u_0)), \\ \tilde{u} = u_n + \frac{h}{2}(3\varphi(t_n, u_n) - \varphi(t_{n-1}, u_{n-1})), \\ u_{n+1} = u_n + \frac{h}{24}\left(9\varphi(t_{n+1}, \tilde{u}) + 19\varphi(t_n, u_n) - 5\varphi(t_{n-1}, u_{n-1}) + \varphi(t_{n-

In [23]:

```
def AM3AB2(phi,tt):
    uu = [y0]
    uu.append(sol_exacte(tt[1]))
    uu.append(sol_exacte(tt[2]))
    for i in range(2,N):
        k1 = h * phi( tt[i], uu[i] )
        k2 = h * phi( tt[i-1], uu[i-1] )
        pred = uu[i] + (3.*k1-k2) /2.0
        uu.append(uu[i]+h*(5.*phi(tt[i+1],pred)+8.*phi(tt[i],u
    return uu
```

In [24]:

```
def AM4AB2(phi,tt):
    uu = [y0]
    uu.append(sol_exacte(tt[1]))
    uu.append(sol_exacte(tt[2]))
    uu.append(sol_exacte(tt[3]))
    for i in range(3,N):
        k1 = h * phi( tt[i], uu[i] )
        k2 = h * phi( tt[i-1], uu[i-1] )
        pred = uu[i] + (3.*k1-k2) /2.0
        uu.append(uu[i]+h*( 251.*phi(tt[i+1],pred)+646.*phi(tt
    return uu

def AM4AB3(phi,tt):
    uu = [y0]
    uu.append(sol_exacte(tt[1]))
    uu.append(sol_exacte(tt[2]))
    uu.append(sol_exacte(tt[3]))
    for i in range(3,N):
        k1 = h * phi( tt[i], uu[i] )
        k2 = h * phi( tt[i-1], uu[i-1] )
        k3 = h * phi( tt[i-2], uu[i-2] )
        pred = uu[i] + (23.*k1-16.*k2+5.*k3) /12.0
        uu.append(uu[i]+h*( 251.*phi(tt[i+1],pred)+646.*phi(tt
    return uu

def AM4AB4(phi,tt):
    uu = [y0]
    uu.append(sol_exacte(tt[1]))
    uu.append(sol_exacte(tt[2]))
    uu.append(sol_exacte(tt[3]))
    for i in range(3,N):
        k1 = h * phi( tt[i], uu[i] )
        k2 = h * phi( tt[i-1], uu[i-1] )
        k3 = h * phi( tt[i-2], uu[i-2] )
        k4 = h * phi( tt[i-3], uu[i-3] )
        pred = uu[i] + (55.*k1-59.*k2+37.*k3-9.*k4) /24.0
        uu.append(uu[i]+h*( 251.*phi(tt[i+1],pred)+646.*phi(tt
    return uu
```

# 4 Convergence

On initialise le problème de Cauchy

In [25]:

```
t0, y0, tfinal = 0, 1, 3
```

On définit la solution exacte:

In [26]:

```
def sol_exacte(t):
    return exp(t)
    #return exp(-t)
    #return sqrt(2.*t+1.)
    #return sqrt(t**2+1.)
    #return 1./sqrt(1.-2.*t)
```

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

In [27]:

```
def phi(t,y):
    return y
    #return -y
    #return 1./y
    #return t/y
    #return y**3
```

Pour chaque schéma, on calcule la solution approchée avec différentes valeurs de $h_k = 1/N_k$, à savoir $N_k = 2, 2^2, 2^3, ... 2^{10}$). 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 $e_k = \max_{i=0,...,N_k} |y(t_i) - u_i|$ avec $N_k = 2^{k+1}$.

In [28]:

```
H = []

err_ep   = []
err_AB2  = []
err_AB3  = []
err_AB4  = []
err_AB5  = []
err_N2   = []
err_N3   = []
err_N4   = []
err_RK4  = []

err_er   = []
err_CN   = []
err_AM2  = []
err_AM3  = []
err_AM4  = []
err_AM5  = []
err_BDF2 = []
err_BDF3 = []

err_em   = []
err_heun = []
err_AM4AB2  = []
err_AM4AB3  = []
err_AM4AB4  = []

for k in range(8):
    N = 2**(k+3)
    h = (tfinal-t0)/N
    H.append(h)
    tt = [t0+i*h for i in range(N+1)]
    yy = [sol_exacte(t) for t in tt]
    # schemas explicites
    uu_ep   = euler_progressif(phi,tt)
    uu_AB2  = AB2(phi,tt)
    uu_AB3  = AB3(phi,tt)
    uu_AB4  = AB4(phi,tt)
    uu_AB5  = AB5(phi,tt)
    uu_N2   = N2(phi,tt)
    uu_N3   = N3(phi,tt)
    uu_N4   = N4(phi,tt)
    uu_RK4  = RK4(phi,tt)
    # schemas implicites
    uu_er   = euler_regressif(phi,tt)
    uu_CN   = CN(phi,tt)
    uu_AM2  = AM2(phi,tt)
    uu_AM3  = AM3(phi,tt)
    uu_AM4  = AM4(phi,tt)
    uu_AM5  = AM5(phi,tt)
    uu_BDF2 = BDF2(phi,tt)
    uu_BDF3 = BDF3(phi,tt)
    # schemas predictor-corrector
    uu_em   = euler_modifie(phi,tt)
    uu_heun = heun(phi,tt)
    uu_AM4AB2 = AM4AB2(phi,tt)
    uu_AM4AB3 = AM4AB3(phi,tt)
    uu_AM4AB4 = AM4AB4(phi,tt)
```

```
    # erreurs
    err_ep.append(max([abs(uu_ep[i]-yy[i]) for i in range(N+1)
    err_AB2.append(max([abs(uu_AB2[i]-yy[i]) for i in range(N+
    err_AB3.append(max([abs(uu_AB3[i]-yy[i]) for i in range(N+
    err_AB4.append(max([abs(uu_AB4[i]-yy[i]) for i in range(N+
    err_AB5.append(max([abs(uu_AB5[i]-yy[i]) for i in range(N+
    err_N2.append(max([abs(uu_N2[i]-yy[i]) for i in range(N+1)
    err_N3.append(max([abs(uu_N3[i]-yy[i]) for i in range(N+1)
    err_N4.append(max([abs(uu_N4[i]-yy[i]) for i in range(N+1)
    err_RK4.append(max([abs(uu_RK4[i]-yy[i]) for i in range(N+

    err_er.append(max([abs(uu_er[i]-yy[i]) for i in range(N+1)
    err_CN.append(max([abs(uu_CN[i]-yy[i]) for i in range(N+1)
    err_AM2.append(max([abs(uu_AM2[i]-yy[i]) for i in range(N+
    err_AM3.append(max([abs(uu_AM3[i]-yy[i]) for i in range(N+
    err_AM4.append(max([abs(uu_AM4[i]-yy[i]) for i in range(N+
    err_AM5.append(max([abs(uu_AM5[i]-yy[i]) for i in range(N+
    err_BDF2.append(max([abs(uu_BDF2[i]-yy[i]) for i in range(
    err_BDF3.append(max([abs(uu_BDF3[i]-yy[i]) for i in range(

    err_em.append(max([abs(uu_em[i]-yy[i]) for i in range(N+1)
    err_heun.append(max([abs(uu_heun[i]-yy[i]) for i in range(
    err_AM4AB2.append(max([abs(uu_AM4AB2[i]-yy[i]) for i in ra
    err_AM4AB3.append(max([abs(uu_AM4AB3[i]-yy[i]) for i in ra
    err_AM4AB4.append(max([abs(uu_AM4AB4[i]-yy[i]) for i in ra
```

Pour estimer l'ordre de convergence on estime la pente de la droite qui relie l'erreur au pas $k$ à l'erreur au pas $k+1$ en echelle logarithmique en utilisant la fonction `polyfit` basée sur la régression linéaire.

In [29]:

```
print ('EE\t %1.2f' %(polyfit(log(H),log(err_ep), 1)[0]))
print ('AB2\t %1.2f' %(polyfit(log(H),log(err_AB2), 1)[0]))
print ('AB3\t %1.2f' %(polyfit(log(H),log(err_AB3), 1)[0]))
print ('AB4\t %1.2f' %(polyfit(log(H),log(err_AB4), 1)[0]))
print ('AB5\t %1.2f' %(polyfit(log(H),log(err_AB5), 1)[0]))
print ('N2\t %1.2f' %(polyfit(log(H),log(err_N2), 1)[0]))
print ('N3\t %1.2f' %(polyfit(log(H),log(err_N3), 1)[0]))
print ('N4\t %1.2f' %(polyfit(log(H),log(err_N4), 1)[0]))
print ('RK4\t %1.2f' %(polyfit(log(H),log(err_RK4), 1)[0]))
print('\n')
print ('EI\t %1.2f' %(polyfit(log(H),log(err_er), 1)[0]))
print ('CN\t %1.2f' %(polyfit(log(H),log(err_CN), 1)[0]))
print ('AM2\t %1.2f' %(polyfit(log(H),log(err_AM2), 1)[0]))
print ('AM3\t %1.2f' %(polyfit(log(H),log(err_AM3), 1)[0]))
print ('AM4\t %1.2f' %(polyfit(log(H),log(err_AM4), 1)[0]))
print ('AM5\t %1.2f' %(polyfit(log(H),log(err_AM5), 1)[0]))
print ('BDF2\t %1.2f' %(polyfit(log(H),log(err_BDF2), 1)[0]))
print ('BDF3\t %1.2f' %(polyfit(log(H),log(err_BDF3), 1)[0]))
print('\n')
print ('EM\t %1.2f' %(polyfit(log(H),log(err_em), 1)[0]))
print ('Heun\t %1.2f' %(polyfit(log(H),log(err_heun), 1)[0]))
print ('AM4AB2\t %1.2f' %(polyfit(log(H),log(err_AM4AB2), 1)[0
print ('AM4AB3\t %1.2f' %(polyfit(log(H),log(err_AM4AB3), 1)[0
print ('AM4AB4\t %1.2f' %(polyfit(log(H),log(err_AM4AB4), 1)[0
```

```
EE       0.92
AB2      1.92
AB3      2.87
AB4      3.82
AB5      4.75
N2       1.97
N3       2.92
N4       3.86
RK4      3.95


EI       1.12
CN       2.01
AM2      2.95
AM3      3.90
AM4      4.85
AM5      5.28
BDF2     1.95
BDF3     2.87


EM       1.95
Heun     1.95
AM4AB2   2.86
AM4AB3   3.82
AM4AB4   4.80
```

Pour afficher l'ordre de convergence on utilise une échelle 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 [30]:

```
figure(1,figsize=(20,5))

subplot(1,3,1)
loglog(H,err_ep, 'r-o',label='AB-1=EE')
loglog(H,err_AB2, 'g-+',label='AB-2')
loglog(H,err_AB3, 'c-D',label='AB-3')
loglog(H,err_AB4, 'y-*',label='AB-4')
loglog(H,err_AB5, 'r-.',label='AB-5')
loglog(H,err_N2, 'y-*',label='N-2')
loglog(H,err_N3, 'r-<',label='N-3')
loglog(H,err_N4, 'b-+',label='N-4')
loglog(H,err_RK4, 'g-o',label='RK41')
xlabel('$h$')
ylabel('$e$')
title("Schemas explicites")
legend()
grid(True)

subplot(1,3,2)
loglog(H,err_er, 'r-o',label='AM-0=EI')
loglog(H,err_CN, 'b-v',label='AM-1=CN')
loglog(H,err_AM2, 'm->',label='AM-2')
loglog(H,err_AM3, 'c-D',label='AM-3')
loglog(H,err_AM4, 'g-+',label='AM-4')
loglog(H,err_AM5, 'b->',label='AM-5')
loglog(H,err_BDF2, 'y-*',label='BDF-2')
loglog(H,err_BDF3, 'r-<',label='BDF-3')
xlabel('$h$')
ylabel('$e$')
title("Schemas implicites")
legend()
grid(True)

subplot(1,3,3)
loglog(H,err_em, 'c-o',label='EM')
loglog(H,err_heun, 'y->',label='Heun')
loglog(H,err_AM4AB2, 'r-<',label='AM4-AB2')
loglog(H,err_AM4AB3, 'b-v',label='AM4-AB3')
loglog(H,err_AM4AB4, 'm-*',label='AM4-AB4')
xlabel('$h$')
ylabel('$e$')
title("Schemas predicteur-correcteur")
legend()
grid(True)

show()
```