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 3 - Implémentation de schémas et 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 [32]:

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

```
Autosaving every 300 seconds
```

Table of Contents

1 Implémentation des schémas (cf. CM)

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 $y(t) = e^{t^2}$.

1. Calculer la solution approchée obtenue avec la méthode d'**Euler explicite** avec $h = 1/N$ et $N = 8$ (pour bien visualiser les erreurs);
2. Même exercice pour la méthode d'**Euler implicite** (pour résoudre à chaque pas de temps l'équation non linéaire on peut implémenter une méthode numérique comme la dichotomie ou la méthode de Newton ou, plus simplement, utiliser le module SciPy);

Correction

On commence par importer les modules `math` et `matplotlib` et la fonction `fsolve` du module `scipy.optimize` pour résoudre l'équation implicite présente dans le schéma implicite:

In [33]:

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

On initialise le problème de Cauchy

In [34]:

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

On définit la solution exacte:

In [35]:

```
sol_exacte = lambda t : y0*exp(t**2)
```

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 [36]:

```
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 [39]:

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

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 [40]:

```
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 [41]:

```
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 exacte et approchées:

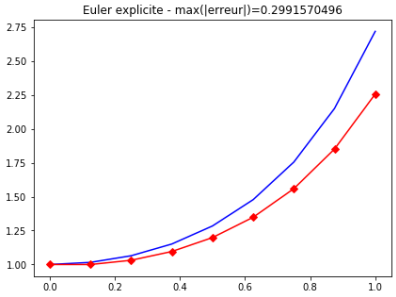
In [42]:

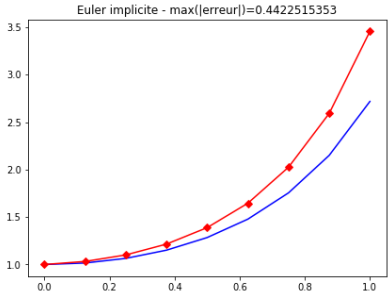
```
yy    = [sol_exacte(t) for t in tt]
uu_ep = euler_progressif(phi,tt)
uu_er = euler_regressif(phi,tt)
```

On compare les graphes des solutions exacte et approchées et on affiche le maximum de l'erreur:

In [43]:

```
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
```





2 Étude de la convergence (cf. CM)

Considérons le même problème de Cauchy.

1. On se propose d'estimer l'ordre de convergence de la méthode d'**Euler explicite**.
    - On calcule d'abord la solution approchée avec différentes valeurs de $h_k = 1/N_k$ (avec $N_k = 2^3, 2^4, \ldots 2^8$).
    - Pour chaque valeur de h_k, on calcule le maximum de la valeur absolue de l'erreur et on la sauvegarde dans le vecteur `err_ep` de sort que `err_ep[k]` contient $e_k = \max_{i=0,\ldots,N_k} |y(t_i) - u_i|$ avec $N_k = 2^{k+1}$.
    - Pour estimer l'ordre de convergence on affiche les points (`h[k]` , `err_ep[k]`) en echèlle logarithmique. On trouve ainsi une droite qui relie l'erreur au pas k à l'erreur au pas $k+1$. Pour estimer la pente globale de cette droite (par des moindres carrés) on utilisers la fonction `polyfit` basée sur la régression linéaire.
2. Même exercice pour estimer l'ordre de convergence de la méthode d'**Euler implicite**.

Correction

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 $N_k = 2^{k+1}$.

In [44]:

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

for k in range(7):
    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)
```

In [45]:

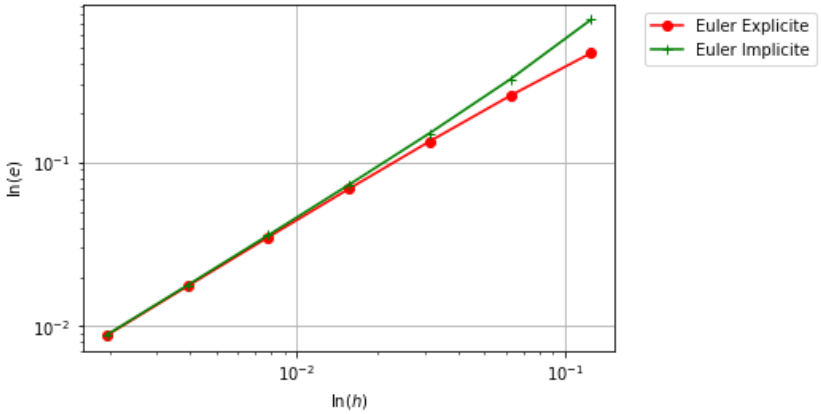
```
print ('Euler progressif %1.2f' %(polyfit(log(H),log(err_ep),
print ('Euler regressif %1.2f' %(polyfit(log(H),log(err_er), 1
```

```
Euler progressif 0.96
Euler regressif 1.05
```

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 $\ln(\text{err})$.

In [46]:

```
figure(1)
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)
show()
```



3 Exercice : Fonction intégrale

Il existe des fonctions définies par des intégrales qu'on ne peut pas calculer explicitement. Il est pourtant possible de calculer des valeurs approchées de ces fonctions.

Pour $x \in [0; \pi/2]$ calculer et afficher la table des valeurs et tracer le graphe de la fonction

$$x \mapsto f(x) = \int_0^x \sqrt{1 - \frac{1}{4}\sin^2(t)}\,\mathrm{d}t$$

en approchant numériquement un problème de Cauchy (lequel?) avec la méthode d'**Euler explicite**.

Vérifier que $f(\pi/6) \simeq 0.51788193$ et $f(\pi/2) \simeq 1.46746221$.

Correction

La fonction f est solution du problème de Cauchy

$$\begin{cases} y'(t) = \sqrt{1 - \frac{1}{4}\sin^2(t)}, \\ y(0) = 0. \end{cases}$$

En effet, si on intègre l'EDO entre 0 et x on a

$$f(x) = \int_0^x \sqrt{1 - \frac{1}{4}\sin^2(t)}\,\mathrm{d}t = \int_0^x y'(t)\mathrm{d}t = y(x) - y(0) = y(x).$$

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 [47]:

```
phi = lambda t,y : sqrt(1.0-0.25*(sin(t))**2)
```

et la condition initiale

In [48]:

```
t0, y0 = 0. , 0.
tfinal = pi/2
```

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

In [49]:

```
N=10
tt=linspace(t0,tfinal,N+1)
```

On calcule la solution approchée:

In [50]:

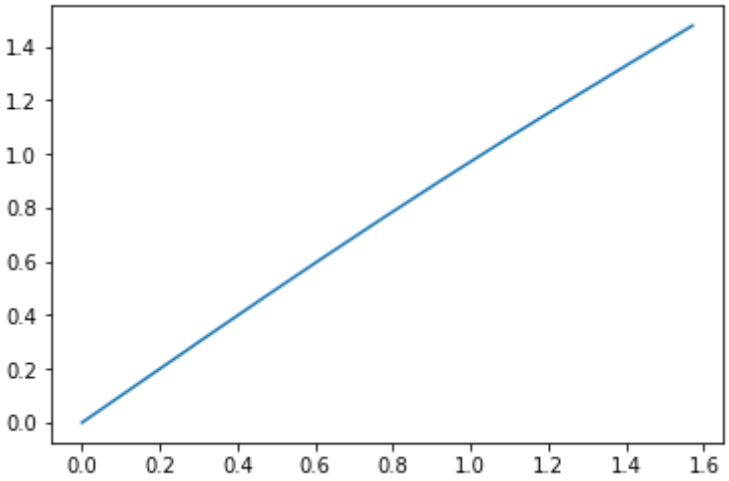
```
uu = [] # pour effacer la solution des autres exercices
uu = euler_progressif(phi,tt)
```

On vérifie que $f(\pi/6) \simeq 0.51788193$ et $f(\pi/2) \simeq 1.46746221$ et on affiche le graphe de la fonction approchée.

In [51]:

```
print("u(%1.2f *pi)=%1.10f" %(tt[-1]/pi, uu[-1]))
index=int(round((pi/6-tt[0])/(tt[1]-tt[0])))
print("u(%1.2f *pi)=%1.10f" %(tt[index]/pi, uu[index]))
plot(tt,uu);
```

```
u(0.50 *pi)=1.4779845495
u(0.15 *pi)=0.4688713622
```



4 Exercice : Problème mal conditionné (= numériquement mal posé)

Considérons le problème de Cauchy

$$\begin{cases} y'(t) = 10y(t) + 11t - 5t^2 - 1, & t \in [0;3] \\ y(0) = \varepsilon. \end{cases}$$

1. Pour $\varepsilon = 0$, tracer les graphes de la solution exacte et de la solution approchée obtenue avec les schémas d'Euler **explicite et implicite** avec un pas de discrétisation $h = \dfrac{1}{2^4}$.
2. Même question pour $\varepsilon = 10^{-10}$.
3. Calculer la solution exacte en fonction de la donnée initiale et expliquer les résultats obtenus.

Correction

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 [52]:

```
phi = lambda t,y : 10*y+11*t-5*t**2-1
```

et la condition initiale

In [53]:

```
t0=0.0
#y0=0.0
y0=1.e-10
tfinal = 3.0
```

On calcule la solution exacte: on a $a(t) = 1$, $b(t) = -10$ et $g(t) = 11t - 5t^2 - 1$ donc

- $A(t) = \int -10dt = -10t$,
- $B(t) = \int (11t - 5t^2 - 1)e^{-10t}dt = \frac{t(t-2)}{2}e^{-10t}$,

d'où $y(t) = ce^{10t} + \frac{t(t-2)}{2}$.

En imposant la condition $\varepsilon = y(0)$ on trouve l'unique solution du problème de Cauchy donné:

$$y(t) = \varepsilon e^{10t} + \frac{t(t-2)}{2}.$$

On constate qu'une petite perturbation de la condition initiale entraîne une grosse perturbation de la solution, en effet si $\varepsilon = 0$ la solution est une parabole, si $\varepsilon \neq 0$ le terme en e^{10t} est dominant.

Par exemple

$$y(4) = \varepsilon e^{40} + 4 = \begin{cases} 4 & \text{si } \varepsilon = 0, \\ > 10^7 & \text{si } \varepsilon = 10^{-10}. \end{cases}$$

Par conséquent, si $\varepsilon = 0$ le problème de Cauchy est numériquement mal posé car toute (petite) erreur de calcul a le même effet qu'une perturbation de la condition initiale: on *réveil* le terme dormant e^{10t}.

In [54]:

```
exact = lambda t : y0*exp(10.*t)+0.5*(t**2)-t
```

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

In [55]:

```
h=1./(2.0**4)
tt=arange(t0,tfinal+0.1,h)
```

On calcule les solutions exacte et approchée:

In [56]:

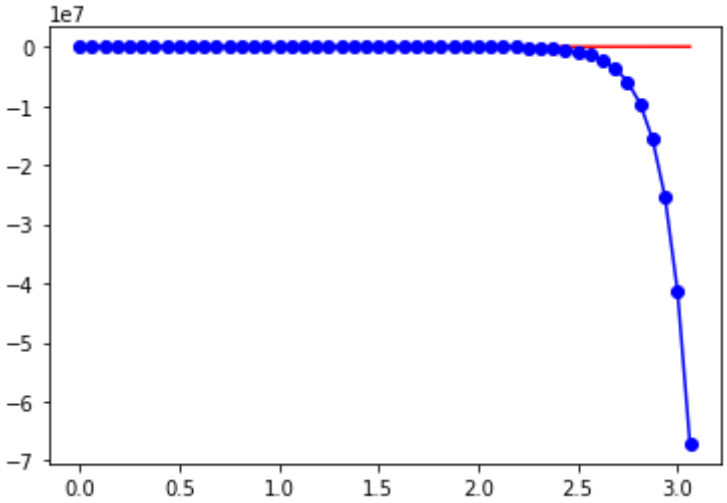
```
yy=[exact(t) for t in tt]
uu_ep=euler_progressif(phi,tt)
uu_er=euler_regressif(phi,tt)
```

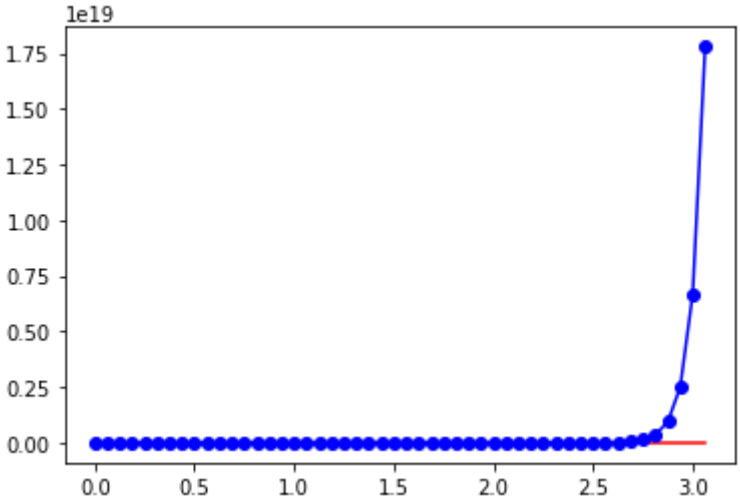
```
/home/minnolina/anaconda3/lib/python3.6/site-pack
ages/scipy/optimize/minpack.py:161: RuntimeWarnin
g: The iteration is not making good progress, as
measured by the
  improvement from the last ten iterations.
  warnings.warn(msg, RuntimeWarning)
```

On compare les graphes des solutions exacte et approchée et on affiche le maximum de l'erreur:

In [57]:

```
figure()
plot(tt,yy,'r-',tt,uu_ep,'b-o');
figure()
plot(tt,yy,'r-',tt,uu_er,'b-o');
```





5 Exercice : Système de deux EDO avec invariant

Considérons le problème de Cauchy

$$\begin{cases} x'(t) = \varphi_1(t,x,y), \\ y'(t) = \varphi_2(t,x,y), \quad t \in [t_0;T] \\ x(t_0) = x_0, \\ y(t_0) = y_0. \end{cases}$$

avec

$t_0 = 0, \qquad T = 4\pi, \qquad \varphi_1(t,x,y) = -y(t), \qquad \varphi_2(t,x,y) =$

La solution exacte est

$$x(t) = \cos(t),$$
$$y(t) = \sin(t).$$

Cette solution est périodique de période 2π et possède un invariant:

$$I(t) = x^2(t) + y^2(t) = 1, \qquad \forall t.$$

Dans une simulation numérique on aimerait que cette propriété soit préservée aussi bien que possible. Nous allons regarder ce qui se passe avec certaines méthodes vues en cours.

On notera $u_n \approx x_n = x(t_n)$ et $w_n \approx y_n = y(t_n)$. À chaque instant t_n, on calculera $J_n \approx I_n = I(t_n)$. Choisissons $h = t_{n+1} - t_n = \pi/48$.

1. Dans un premier temps on se propose d'appliquer la méthode d'**Euler explicite** à la résolution du système.
   A. Tracer (t_n, u_n) et (t_n, w_n) sur un même graphique;
   B. sur un autre graphique tracer (t_n, J_n);
   C. tracer enfin (u_n, w_n) sur un autre graphique et vérifier que la solution numérique tourne vers l'extérieur
   D. Après avoir constaté que l'invariant J_n n'est pas conservé mais augment au cours du temps, montrer analytiquement que $J_n = (1+h^2)^n$.
2. Même exercice pour le schéma d'**Euler implicite** (la linéarité du second membre permet de rendre se schéma explicite). Que peut-on constater? Commenter les résultats et écrire l'expression analytique de J_n.

Correction

In [58]:

```
t0 = 0.
tfinal = 4.*math.pi

x0 = 1.
y0 = 0.

phi1 = lambda t,x,y : -y
phi2 = lambda t,x,y : x
```

La solution exacte est

$$x(t) = \cos(t),$$
$$y(t) = \sin(t).$$

Cette solution est périodique de période 2π et possède un invariant :

$$I(t) = x^2(t) + y^2(t) = 1, \qquad \forall t.$$

Dans une simulation numérique on aimerait que cette propriété soit préservée aussi bien que possible. Nous allons regarder ce qui se passe avec certaines méthodes vues en cours.

On notera $u_n \approx x_n = x(t_n)$ et $w_n \approx y_n = y(t_n)$.

À chaque instant t_n, on calculera $J_n = u_n^2 + w_n^2 \approx I_n = I(t_n)$.

Choisissons $h = t_{n+1} - t_n = \pi/48$.

In [59]:

```
h = pi/48
tt = arange(t0,tfinal+h,h)
```

Euler explicite

$$\begin{cases} u_0 = 1, \\ w_0 = 0, \\ u_{n+1} = u_n + h\varphi_1(t_n, u_n, w_n) = u_n - hw_n, \\ w_{n+1} = w_n + h\varphi_2(t_n, u_n, w_n) = w_n + hu_n \end{cases}$$

In [60]:

```
def euler_progressif(phi1,phi2,tt):
    uu = [x0]
    ww = [y0]
    for i in range(len(tt)-1):
        uu.append(uu[i]+h*phi1(tt[i],uu[i],ww[i]))
        ww.append(ww[i]+h*phi2(tt[i],uu[i],ww[i]))
    return [uu,ww]
```

On a

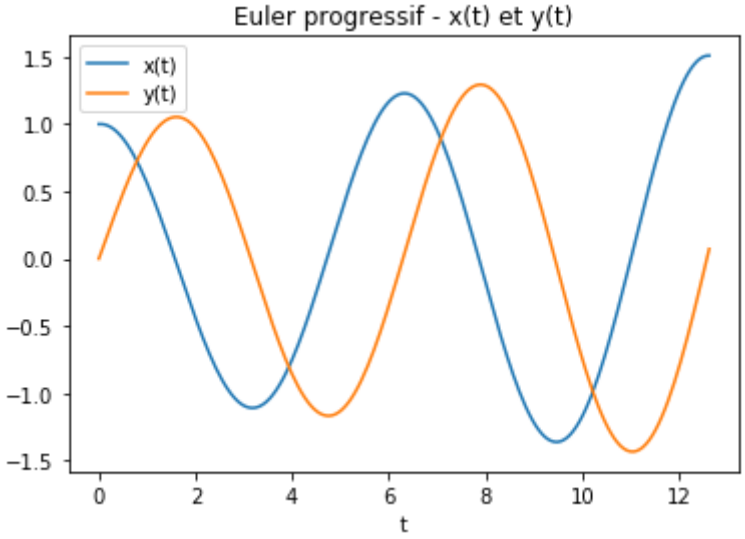
$J_{n+1} = u_{n+1}^2 + w_{n+1}^2 = (u_n - hw_n)^2 + (w_n + hu_n)^2 = (1 + h)(u_n^2 + w_n^2)$

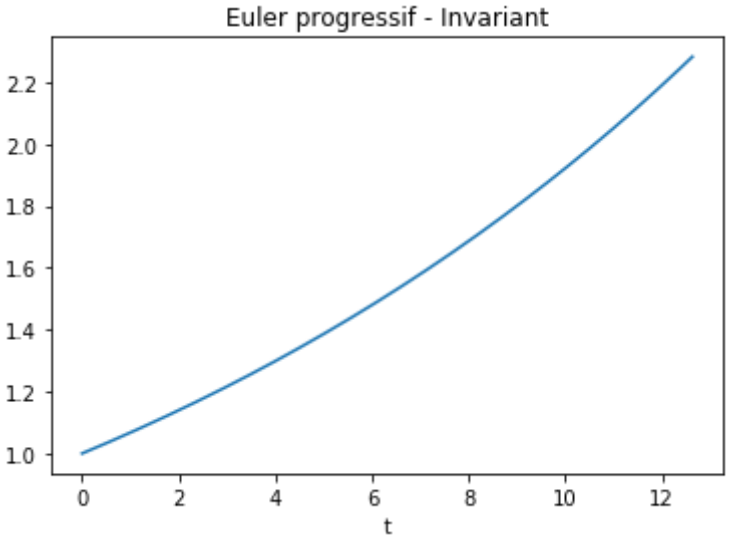
soit encore

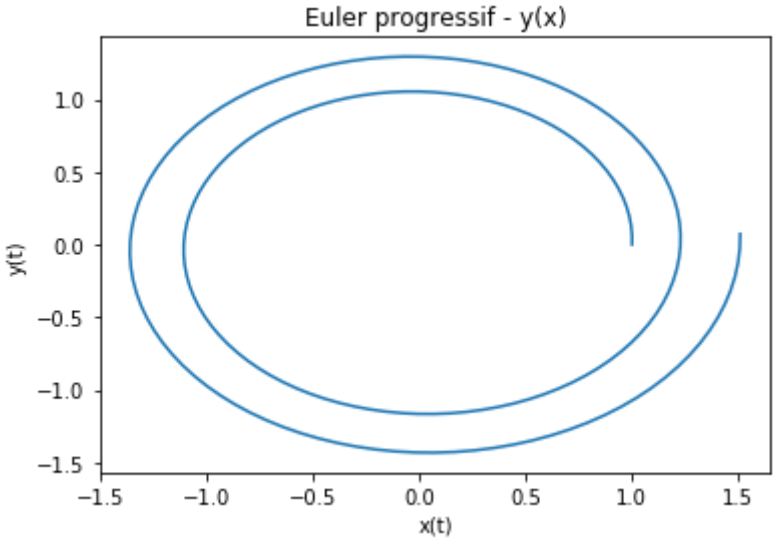
$$J_n = (1 + h)^n J_0 = (1 + h)^n.$$

In [61]:

```
[uu_ep, ww_ep] = euler_progressif(phi1,phi2,tt)
JJ_ep = [(uu_ep[i])**2+(ww_ep[i])**2 for i in range(len(tt))]

figure()
plot(tt,uu_ep,tt,ww_ep)
xlabel('t')
legend(['x(t)','y(t)'])
title('Euler progressif - x(t) et y(t)')

figure()
plot(tt,JJ_ep)
xlabel('t')
title('Euler progressif - Invariant')

figure()
plot(uu_ep,ww_ep)
xlabel('x(t)')
ylabel('y(t)')
title('Euler progressif - y(x)');
```







Euler implicite

$$\begin{cases} u_0 = 1, \\ w_0 = 0, \\ u_{n+1} = u_n + h\varphi_1(t_{n+1}, u_{n+1}, w_{n+1}) = u_n - hw_{n+1}, \\ w_{n+1} = w_n + h\varphi_2(t_{n+1}, u_{n+1}, w_{n+1}) = w_n + hu_{n+1} \end{cases}$$

En resolvant le système linéaire on obtient une écriture explicite

$$\begin{cases} u_0 = 1, \\ w_0 = 0, \\ u_{n+1} = \dfrac{u_n - hw_n}{1+h^2}, \\ w_{n+1} = \dfrac{w_n + hu_n}{1+h^2}. \end{cases}$$

In [62]:

```
def euler_regressif(phi1,phi2,tt):
    uu = [x0]
    ww = [y0]
    for i in range(len(tt)-1):
        uu.append((uu[i]-h*ww[i])/(1+h**2))
        ww.append((ww[i]+h*uu[i])/(1+h**2))
    return [uu,ww]
```

On a

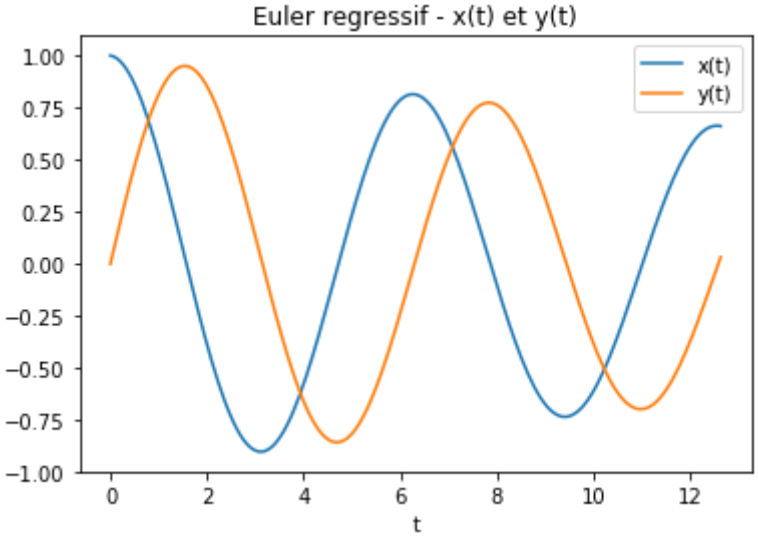
$$J_{n+1} = u_{n+1}^2 + w_{n+1}^2 = \frac{(u_n - hw_n)^2 + (w_n + hu_n)^2}{(1+h^2)^2} = (u_n^2 + w_n^2)\frac{1 +}{(1 +}$$

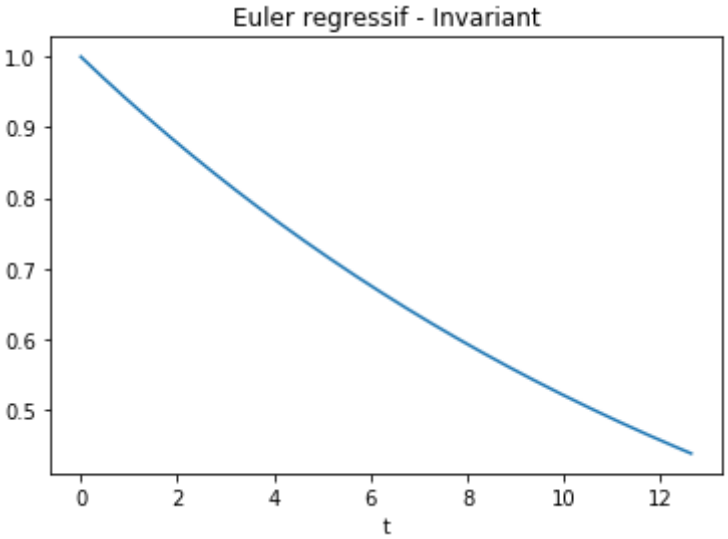
soit encore

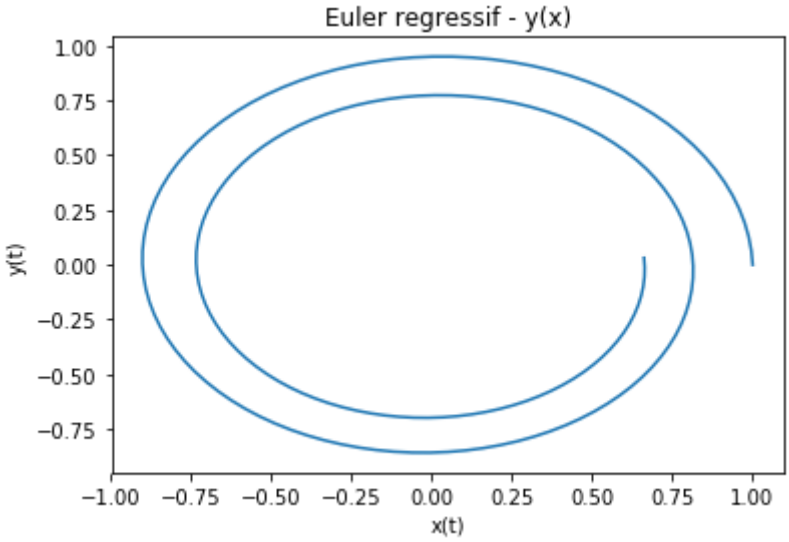
$$J_n = \left(\frac{1+h}{(1+h^2)^2}\right)^n J_0 = \left(\frac{1+h}{(1+h^2)^2}\right)^n.$$

In [63]:

```
[uu_er, ww_er] = euler_regressif(phi1,phi2,tt)
JJ_er = [(uu_er[i])**2+(ww_er[i])**2 for i in range(len(tt))]

figure()
plot(tt,uu_er,tt,ww_er)
xlabel('t')
legend(['x(t)','y(t)'])
title('Euler regressif - x(t) et y(t)')

figure()
plot(tt,JJ_er)
xlabel('t')
title('Euler regressif - Invariant')

figure()
plot(uu_er,ww_er)
xlabel('x(t)')
ylabel('y(t)')
title('Euler regressif - y(x)');
```






Euler regressif - y(x)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
y(t)
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x(t)