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_CM9 : Révisions

Table of Contents

1 Exercice : construction d'un schéma

Considérons le problème de Cauchy:

> trouver y: $[t_0, T] \subset \mathbb{R} \to \mathbb{R}$ tel que
> $$\begin{cases} y'(t) = \varphi(t, y(t)), & \forall t \in [t_0, T], \\ y(t_0) = y_0. \end{cases}$$
> Supposons que l'on ait montré l'existence d'une unique solution y.

On subdivise l'intervalle $[t_0, T]$ en N intervalles de longueur $h = (T - t_0)/N = t_{n+1} - t_n$. Pour chaque nœud $t_n = t_0 + nh\,(1 \leq n \leq N)$ on cherche la valeur inconnue u_n qui approche $y(t_n)$. L'ensemble de $N + 1$ valeurs $\{u_0 = y_0, u_1, \ldots, u_N\}$ représente la solution numérique.

Dans cette exercice on va construire des nouveaux schémas numériques basés sur l'intégration de l'EDO $y'(t) = \varphi(t, y(t))$ entre t_n et t_{n+2}:

$$y(t_{n+2}) = y(t_n) + \int_{t_n}^{t_{n+2}} \varphi(t, y(t))\mathrm{d}t.$$

1. En utilisant la formule de quadrature du point milieu pour approcher le membre de droite écrire un schéma numérique explicite permettant de calculer u_{n+2} à partir de u_{n+1} et u_n. Notons que ce schéma a besoin de deux valeurs initiales; on posera alors $u_0 = y_0$ et u_1 sera approché par une prédiction d'Euler progressive.
2. En utilisant la formule de quadrature de Cavalieri-Simpson pour approcher le membre de droite écrire un schéma numérique implicite permettant de calculer u_{n+2} à partir de u_{n+1} et u_n. Notons que ce schéma a besoin de deux valeurs initiales; on posera alors $u_0 = y_0$ et u_1 sera approché par une prédiction d'Euler progressive.
3. Proposer une modification du schéma au point précédent pour qu'il devient explicite.

Rappels: une formule de quadrature interpolatoire approche l'intégrake d'une fonction f par l'intègrale dun polynôme qui interpole f en des points donnés: $\int_a^b f(t)\mathrm{d}t \approx \int_a^b p(t)\mathrm{d}t$

- la formule de quadrature du point milieu considère comme points d'interpolation le point $\frac{a+b}{2}$ ainsi
$$\int_a^b f(t)\mathrm{d}t \approx \int_a^b f\left(\frac{a+b}{2}\right)\mathrm{d}t = (b-a)f\left(\frac{a+b}{2}\right)$$
- la formule de Simpson considère comme points d'interpolation les points $\left\{a, \frac{a+b}{2}, b\right\}$ ainsi
$$\int_a^b f(t)\mathrm{d}t \approx \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

1.1 Correction

Si on utilise la formule de quadrature du point milieu sur l'intervalle $[t_n\,;\,t_{n+2}]$, ie

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

on obtient

$$\begin{cases} u_0 = y(t_0) = y_0, \\ u_1 \approx y(t_1) \\ u_{n+2} = u_n + 2h\varphi(t_{n+1}, u_{n+1}) \quad n = 0, 1, \dots N-2 \end{cases}$$

où u_1 est une approximation de $y(t_1)$. Nous pouvons utiliser une prédiction d'Euler progressive pour approcher u_1. Nous avons construit ainsi un nouveau schéma explicite à 2 pas:

$$\begin{cases} u_0 = y(t_0) = y_0, \\ u_1 = u_0 + h\varphi(t_0, u_0), \\ u_{n+2} = u_n + 2h\varphi(t_{n+1}, u_{n+1}) \quad n = 0, 1, \dots N-2 \end{cases}$$

1.2 Correction

Si on utilise la formule de quadrature de Cavalieri-Simpson sur l'intervalle $[t_n\,;\,t_{n+2}]$, ie

$$\int_{t_n}^{t_{n+2}} \varphi(t, y(t))\mathrm{d}t \approx \frac{h}{3}(\varphi(t_n, y(t_n)) + 4\varphi(t_{n+1}, y(t_{n+1})) + \varphi(t_{n+2}, y(t_{n+2}))),$$

et avec une prédiction d'Euler explicite pour approcher u_1, nous obtenons un nouveau schéma implicite à 3 pas:

$$\begin{cases} u_0 = y(t_0) = y_0, \\ u_1 = u_0 + h\varphi(t_0, u_0), \\ u_{n+2} = u_n + \frac{h}{3}(\varphi(t_n, u_n) + 4\varphi(t_{n+1}, u_{n+1}) + \varphi(t_{n+2}, u_{n+2})) \quad n = 0, 1, \dots N-2 \end{cases}$$

1.3 Correction

Pour éviter le calcul implicite de u_{n+2}, nous pouvons utiliser une technique de type predictor-corrector et remplacer le u_{n+2} dans le terme $\varphi(t_{n+2}, u_{n+2})$ par exemple par $\tilde{u}_{n+2} = u_{n+1} + h\varphi(t_{n+1}, u_{n+1})$. Nous avons construit ainsi un nouveau schéma explicite à 3 pas:

$$\begin{cases} u_0 = y(t_0) = y_0, \\ u_1 = u_0 + h\varphi(t_0, u_0), \\ u_{n+2} = u_n + \frac{h}{3}(\varphi(t_n, u_n) + 4\varphi(t_{n+1}, u_{n+1}) + \varphi(t_{n+2}, u_{n+1} + h\varphi(t_{n+1}, u_{n+1})) \quad n = 0, 1, \dots N-2 \end{cases}$$

2 Exercice : Modèle de Malthus

On modélise la croissance d'une population d'effectif $y(t) \geq 0$ par l'équation différentielle

$$y'(t) = ay(t), \qquad a \in \mathbb{R}.$$

Dans ce modèle la variation de population est proportionnelle à la population elle-même: le coefficient a est appelé paramètre de Malthus ou potentiel biologique et représente la différence entre le nombre de nouveaux nés par individu en une unité de temps et la fraction d'individus qui meurent en une unité de temps. On s'intéresse à l'unique solution de cette équation vérifiant $y(t_0) = y_0$ où y_0 et t_0 sont des réels donnés.

1. Donner l'unique solution théorique à ce problème de Cauchy. Calculer $\lim_{t\to+\infty} y(t)$ en fonction du paramètre a.
2. Calculer analytiquement la solution obtenue avec la méthode d'Euler explicite sur l'intervalle $[0; 2]$ avec $N_h = 5$, ensuite la tracer avec Python et la comparer à la solution exacte.
3. Soit $u_{N_h}(2)$ l'approximation de $y(2)$ obtenue avec la méthode d'Euler explicite et N_h nœuds. Calculer $u_{N_h}(2)$ avec $N_h = 2^i$ et montrer analytiquement que $\lim_{N_h\to+\infty} u_{N_h}(2) = y(2)$.
4. Tracer le graphe de $\max_{0\leq n\leq N_h} |y(t_n) - u_n|$ en fonction de N_h ainsi que le graphe des courbes $1/N_h$, $1/N_h^2$ ... avec une progression logarithmique en abscisse et en ordonnées. On prendra par exemple $N_h = 200$, puis $N_h = 300, 400, \ldots 2000$.

Correction

L'unique solution du problème de Cauchy $\begin{cases} y'(t) = ay(t) \\ y(t_0) = y_0 > 0 \end{cases}$ pour $t \geq t_0$ est la fonction $y(t) = y_0 e^{at}$. On a

$$\lim_{t\to+\infty} y(t) = \begin{cases} +\infty & \text{si } a > 0, \\ y_0 & \text{si } a = 0, \\ 0^+ & \text{si } a < 0, \end{cases}$$

Pour $a = 1$, $t_0 = 0$ et $y_0 = 1$, la solution exacte est $y(t) = e^t$.

On subdivise l'intervalle $[t_0; T]$ en N intervalles $[t_n; t_{n+1}]$ de largeur $h = (T - t_0)/N$ avec $t_n = t_0 + nh$ pour $n = 0, \ldots, N$.
Pour chaque nœud t_n, on note $y_n = y(t_n)$; l'ensemble des valeurs $\{y_0, y_1, \ldots, y_N\}$ représente la solution exacte discrète.
Pour chaque nœud t_n, on cherche la valeur inconnue u_n qui approche la valeur exacte y_n.
L'ensemble des valeurs $\{u_0 = y_0, u_1, \ldots, u_N\}$ représente la solution numérique.

Dans la méthode d'Euler explicite, cette solution approchée est obtenue en construisant une suite récurrente comme suit

$$\begin{cases} u_0 = y_0, \\ u_{n+1} = u_n + h\varphi(t_n, u_n). \end{cases}$$

En utilisant ce schéma pour notre EDO on obtient

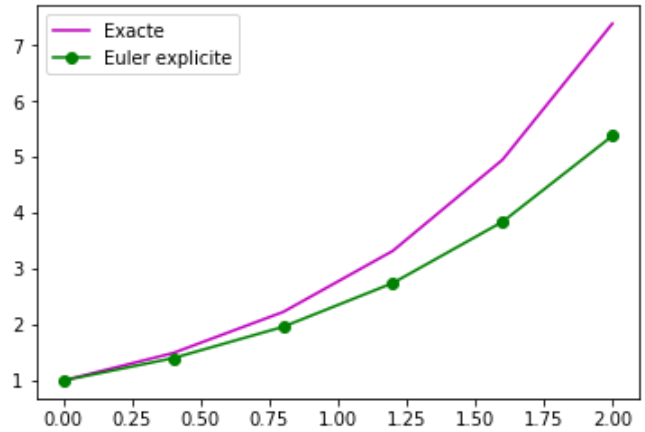
$$\begin{cases} u_0 = y_0, \\ u_{n+1} = (1 + ah)u_n \end{cases}$$

soit encore $u_n = (1 + ah)^n u_0 = \left(1 + \frac{2}{5}\right)^n$.

In [2]:

```
%reset -f
%matplotlib inline

from math import *
from matplotlib.pylab import *

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

phi = lambda t,y : a*y # EDO
sol_exacte = lambda t : y0*math.exp(a*t) # SOLUTION EXACTE

# INITIALISATION
a  = 1.
Nh = 5
t0, y0 = 0.0, 1.0
tfinal = 2.

# CALCUL SOLUTION APPROCHEE
tt1 = linspace(t0,tfinal,Nh+1)
uu1 = euler_progressif(phi,tt1)

# CALCUL SOLUTION EXACTE POUR PLOT
yy = [sol_exacte(t) for t in tt1]

# PLOT
plot(tt1,yy,'m-',tt1,uu1,'go-')
legend(['Exacte','Euler explicite'],loc='upper left')

# PRINT
h1=tt1[1]-tt1[0]
for n in range(Nh):
    print ('y_%1d=%1.14f u_%1d=%1.14f (1+h)^%1d=%1.14f' %(n,yy[n],n,uu1[n],n,(1.+h1)**n))
```

```
y_0=1.00000000000000 u_0=1.00000000000000 (1+h)^0=1.00000000000000
y_1=1.49182469764127 u_1=1.40000000000000 (1+h)^1=1.40000000000000
y_2=2.22554092849247 u_2=1.96000000000000 (1+h)^2=1.96000000000000
y_3=3.32011692273655 u_3=2.74400000000000 (1+h)^3=2.74400000000000
y_4=4.95303242439511 u_4=3.84160000000000 (1+h)^4=3.84160000000000
```



On compare les approximations de $y(2)$ avec plusieurs valeurs de N_h :

$$u_{N_h} = (1+ah)^{N_h} u_0 = \left(1 + \frac{2}{N_h}\right)^{N_h} \xrightarrow[N_h \to +\infty]{} e^2 = y(2).$$

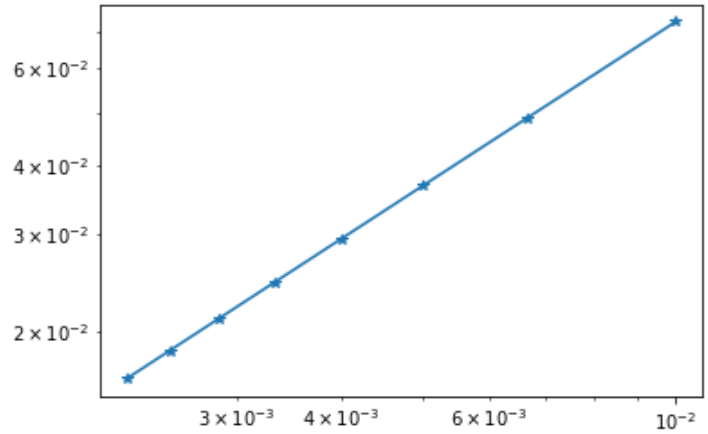
In [3]:
```
# MAIN
y_2=sol_exacte(tfinal)
print ('\nSolution exacte en t=2:')
print ('y(2) =',y_2)
print ('\nSolution approchee en t=2 avec N points:')
print ('N\t sol_approx\t\t\t |erreur|')
for i in range(3,15):
    N=2**i
    tt = linspace(t0,tfinal,N+1)
    uu = euler_progressif(phi,tt)
    print (N,'\t',uu[-1], '\t',abs(uu[-1]-y_2))
```

```
Solution exacte en t=2:
y(2) = 7.38905609893065

Solution approchee en t=2 avec N points:
N        sol_approx                      |erreur|
8        5.9604644775390625      1.428591621391588
16       6.583250172027423       0.8058059269032274
32       6.958666757218805       0.4303893417118454
64       7.166276152788222       0.22277994614242846
128      7.275669793128417       0.11338630580223352
256      7.3318505987410365      0.0572055001896139
512      7.3603235532692795      0.028732545661370956
1024     7.374657160341845       0.014398938588805699
2048     7.381848435880501       0.007207663050149193
4096     7.385450215539008       0.0036058833916428057
8192     7.38725264383889        0.0018034550917604975
16384    7.388154242982074       0.0009018559485767241
```

Vérifions la convergence linéaire:

In [4]:
```
error=[]
H=[]
N=arange(200,1000,100)
for n in N:
    tt=linspace(t0,tfinal,n+1)
    H.append(tt[1]-tt[0])
    yy=[sol_exacte(t) for t in tt]
    uu = euler_progressif(phi,tt)
    error_vec=[abs(uu[i]-yy[i]) for i in range(len(uu))]
    error.append(max(error_vec))

# ESTIMATION ORDRES DE CONVERGENCE par regression
print ('Pente par regression lineaire', polyfit(log(H),log(error), 1)[0])

# PLOT ERREUR
loglog(H,error,'*-');
```

```
Pente par regression lineaire 0.994213737099799
```



3 Exercice : Modèle de Verhulst (croissance logistique).

Dans le modèle logistique, on suppose que la croissance de la population d'effectif $y(t) \geq 0$ est modélisée par l'équation différentielle:

$$y'(t) = (a - by(t))y(t), \qquad a, b > 0.$$

Ce modèle prend en compte une éventuelle surpopulation. En effet, le taux de croissance de la population est désormais de la forme $a - by(t)$: c'est la différence entre un taux de fertilité a et un taux de mortalité de la forme $by(t)$ qui augmente avec l'effectif de la population.

1. Calculer les solutions exactes de l'équation différentielle.
2. Écrire la relation entre u_{n+1} et u_n obtenue obtenue en appliquant la méthode d'\textsc{Euler} explicite.
3. Pour $a = b = 0.5$ et $t_0 = 0$, tracer avec python les solutions exactes sur l'intervalle de temps $[0; 20]$ pour les valeurs de y_0 suivantes:
   $$y_0 = 0.01 \qquad y_0 = 0.1 \qquad y_0 = 0.4 \qquad y_0 = 0.8 \qquad y_0 = 1 \qquad y_0 = 1.5 y_0 = 2$$
   Sur un autre graphique à coté, tracer les solutions correspondantes obtenues par la méthode d'Euler explicite avec $N_h = 50$.
   Sur un autre graphique à coté, tracer les solutions correspondantes obtenues par la méthode d'Euler explicite avec $N_h = 100$.

Correction

On cherche l'unique solution du problème de Cauchy $\begin{cases} y'(t) = (a - by(t))y(t) \\ y(t_0) = y_0 \geq 0 \end{cases}$ pour $t \geq t_0$.

On écrit l'EDO sous la forme $y'(t) = -by(t)\left(y(t) - \frac{a}{b}\right)$.

Si $y(t) = A$ pour tout $t > 0$ est solution de l'EDO, alors on doit avoir $A = (a - bA)A$. Donc, soit $A = 0$ soit $A = \frac{a}{b}$. Ainsi la solution constante $y(t) = \frac{a}{b} > 0$ représente l'effectif d'équilibre de la population, directement liée aux contraintes environnementale (densité, ressources, etc.).

De plus, les solutions sont des fonctions strictement croissantes si $y(t) \in]0, a/b[$ et décroissantes si $y(t) > a/b$.

Calculons la solution exacte en remarquant qu'il s'agit d'une EDO à variables séparables. Pour $y(t) \neq 0$ et $y(t) \neq \frac{a}{b}$, on remarque que

$$-b = \frac{y'(t)}{\left(y(t) - \frac{a}{b}\right) y(t)} = \frac{\frac{b}{a}y'(t)}{y(t) - \frac{a}{b}y'(t)} - \frac{\frac{b}{a}}{y(t)}.$$

On doit alors calculer

$$\int \frac{1}{y - \frac{a}{b}} \mathrm{d}y - \int \frac{1}{y} \mathrm{d}y = -a \int 1 \mathrm{d}t.$$

On obtient

$$\ln\left(1 - \frac{b/a}{y}\right) = -at + C$$

et on en déduit

$$y(t) = \frac{a/b}{1 - De^{-at}}.$$

En imposant la condition initiale $y(0) = y_0$ on trouve la constante d'intégration D et on conclut que toutes les solutions du problème de Cauchy pour $t \geq 0$ sont

$$y(t) = \begin{cases} 0 & \text{si } y_0 = 0, \\ \dfrac{a}{b} & \text{si } y_0 = \dfrac{a}{b}, \\ \dfrac{1}{\left(\frac{1}{y_0} - \frac{b}{a}\right) e^{-at} + \frac{b}{a}} & \text{sinon.} \end{cases}$$

Remarquons que $\lim_{t \to +\infty} y(t) = \frac{a}{b}$: une population qui évolue à partir de y_0 individus à l'instant initiale tend à se stabiliser vers un nombre d'individus d'environ a/b, ce qui représente la capacité de l'environnement.

La méthode d'Euler explicite (ou progressive) est une méthode d'intégration numérique d'EDO du premier ordre de la forme $y'(t) = \varphi(t, y(t))$.

On subdivise l'intervalle $[t_0\,; T]$ en N intervalles $[t_n\,; t_{n+1}]$ de largeur $h = \dfrac{T - t_0}{N}$ avec $t_n = t_0 + nh$ pour $n = 0, \ldots, N$. La longueur h est appelé le pas de discrétisation.
Pour chaque nœud t_n, on note $y_n = y(t_n)$; l'ensemble des valeurs $\{y_0, y_1, \ldots, y_N\}$ représente la solution exacte discrète.
Pour chaque nœud t_n, on cherche la valeur inconnue u_n qui approche la valeur exacte y_n.
L'ensemble des valeurs $\{u_0 = y_0, u_1, \ldots, u_{N_h}\}$ représente la solution numérique.

Dans la méthode d'Euler explicite, cette solution approchée est obtenue en construisant une suite récurrente comme suit

$$\begin{cases} u_0 = y_0, \\ u_{n+1} = u_n + h\varphi(t_n, u_n). \end{cases}$$

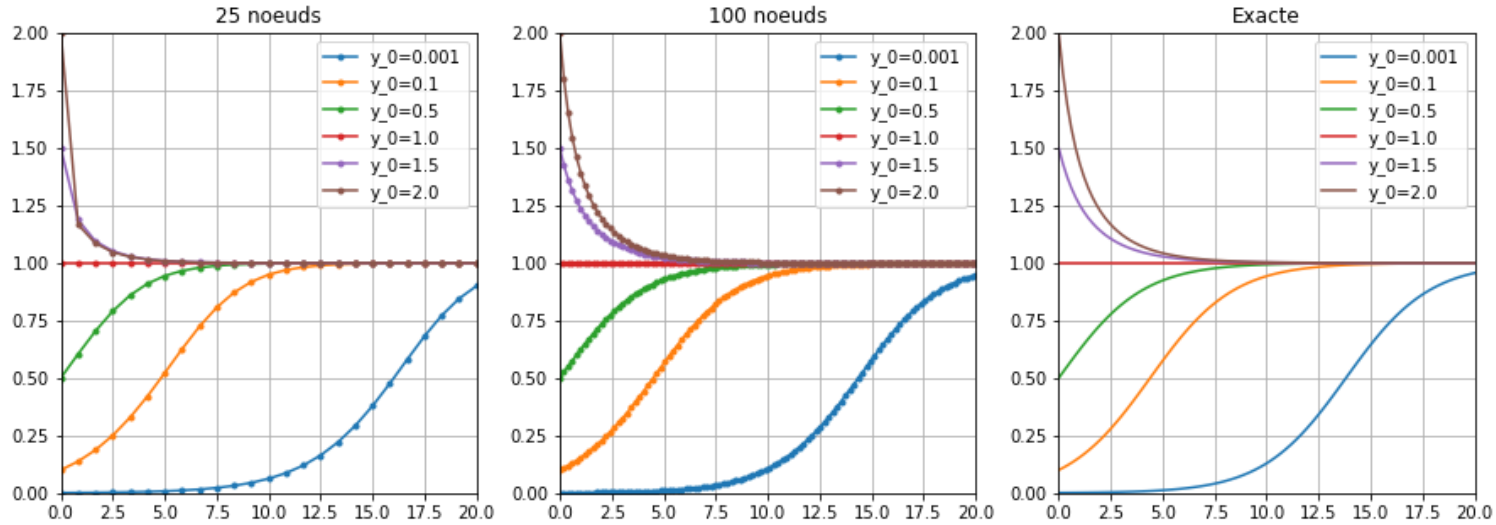
En utilisant cette construction pour notre EDO on obtient

$$\begin{cases} u_0 = y_0, \\ u_{n+1} = u_n + h(a - bu_n)u_n \end{cases}$$

In [5]:

```
from math import *
from matplotlib.pylab import *

def euler_progressif(phi,tt,y0):
    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

phi = lambda t,y : (a-b*y)*y

def sol_exacte(t,y0):
    if abs(y0-0.)<=1.e-10:
        return 0
    elif abs(y0-a/b)<=1.e-10:
        return a/b
    else:
        return 1./((1./y0-b/a)*exp(-a*t)+b/a)

# INITIALISATION
a  = 0.5
b  = 0.5
t0 = 0.
tfinal = 20.
yy_0=[0.001, 0.1,0.5,1.0,1.5,2.0]
leg=[]
for y0 in yy_0:
    leg.append('y_0='+str(y0))

# MAIN
figure(1, figsize=(15, 5))
subplot(1,3,1)
tt=linspace(t0,tfinal,25)
for i in range(len(yy_0)):
    uu=euler_progressif(phi,tt,yy_0[i])
    plot(tt,uu,'.-')
axis([t0,tfinal,0,2.])
title('25 noeuds')
grid()
legend(leg)

subplot(1,3,2)
tt=linspace(t0,tfinal,100)
for i in range(len(yy_0)):
    uu=euler_progressif(phi,tt,yy_0[i])
    plot(tt,uu,'.-')
title('100 noeuds')
```

```
axis([t0,tfinal,0,2.])
grid()
legend(leg)

subplot(1,3,3)
for i in range(len(yy_0)):
    yy=[sol_exacte(t,yy_0[i]) for t in tt]
    plot(tt,yy,'-')
axis([t0,tfinal,0,2.])
title('Exacte')
grid()
legend(leg);
```



4 Exercice

L'évolution de la concentration de certaines réactions chimiques au cours du temps peut être décrite par l'équation différentielle

$$y'(t) = -\frac{1}{1+t^2}y(t).$$

Sachant qu'à l'instant $t = 0$ la concentration est $y(0) = 5$, déterminer la concentration à $t = 2$ à l'aide de la méthode d'**Euler implicite** avec un pas $h = 0.5$.

Correction

Dans ce cas, le schéma d'Euler implicite s'écrit

$$u_{n+1} = u_n - \frac{h}{1+t_{n+1}^2} u_{n+1}.$$

Elle peut être résolue algébriquement et cela donne la suite

$$u_{n+1} = \frac{u_n}{1 + \frac{h}{1+t_{n+1}^2}}.$$

Si à l'instant $t = 0$ la concentration est $y(0) = 5$, et si $h = 1/2$, alors $t_n = n/2$ et

$$u_{n+1} = \frac{4 + (n+1)^2}{6 + (n+1)^2} u_n.$$

On obtient donc

n	t_n	u_n
0	0	5
1	0.5	$\frac{4+1^2}{6+1^2}5 = \frac{5}{7}5 = \frac{25}{7} \approx 3.57$
2	1.0	$\frac{4+2^2}{6+2^2}\frac{25}{7} = \frac{8}{10}\frac{25}{7} = \frac{20}{7} \approx 2.86$
3	1.5	$\frac{4+3^2}{6+3^2}\frac{20}{7} = \frac{13}{15}\frac{20}{7} = \frac{52}{21} \approx 2.48$
4	2.0	$\frac{4+4^2}{6+4^2}\frac{52}{21} = \frac{20}{22}\frac{52}{21} = \frac{520}{231} \approx 2.25$

La concentration à $t = 2$ est d'environ 2.25 qu'on peut comparer avec le calcul exact $y(2) = 5e^{-\arctan(2)} \approx 1.652499838$.

5 Exercice : EDO d'ordre 2

Considérons le problème de Cauchy

$$\begin{cases} y''(t) = ty(t), & t \in [0; 4.5] \\ y(0) = 0.3550280538887817, \\ y'(0) = -0.2588194037928 07. \end{cases}$$

Transformer l'EDO d'ordre 2 en un système de deux EDO d'ordre 1.
Calculer la valeur approchée de y en $t = 4.5$ obtenue avec le schéma RK4 avec un pas de discrétisation $h = 0.001$.

5.1 Correction

On appelle $x(t) = y(t)$ et $z(t) = y'(t)$, alors on a $x'(t) = y'(t) = z(t)$ et $z'(t) = y''(t) = ty(t) = tx(t)$ donc

$$\begin{cases} x'(t) = z(t) \\ z'(t) = tx(t) \\ x(0) = 0.355028053887817, \\ z(0) = -0.258819403792807. \end{cases}$$

In [6]:
```
def RK4(phi1,phi2,tt):
    uu1 = [y0]
    uu2 = [yp0]
    for i in range(len(tt)-1):
        k1_1 = phi1( tt[i]     , uu1[i]          , uu2[i] )
        k1_2 = phi2( tt[i]     , uu1[i]          , uu2[i] )
        k2_1 = phi1( tt[i]+h/2 , uu1[i]+h*k1_1/2, uu2[i]+h*k1_2/2 )
        k2_2 = phi2( tt[i]+h/2 , uu1[i]+h*k1_1/2, uu2[i]+h*k1_2/2 )
        k3_1 = phi1( tt[i]+h/2 , uu1[i]+h*k2_1/2, uu2[i]+h*k2_2/2 )
        k3_2 = phi2( tt[i]+h/2 , uu1[i]+h*k2_1/2, uu2[i]+h*k2_2/2 )
        k4_1 = phi1( tt[i+1]   , uu1[i]+h*k3_1  , uu2[i]+h*k3_2 )
        k4_2 = phi2( tt[i+1]   , uu1[i]+h*k3_1  , uu2[i]+h*k3_2 )
        uu1.append( uu1[i] + (k1_1+2.0*k2_1+2.0*k3_1+k4_1)*h/6 )
        uu2.append( uu2[i] + (k1_2+2.0*k2_2+2.0*k3_2+k4_2)*h/6 )
    return [uu1,uu2]

phi1 = lambda t,y1,y2 : y2
phi2 = lambda t,y1,y2 : t*y1

t0=0.0
y0=0.355028053887817
yp0=-0.258819403792807
h=0.001

tt=[t0+i*h for i in range(4501)]
[uu1,uu2]=RK4(phi1,phi2,tt)
print("u(%1.2f)=%1.16f"%(tt[-1],uu1[-1]))
```

```
u(4.50)=0.0003302503231810
```

6 Exercice : Étude d'un problème numériquement mal posé

Considérons le problème de Cauchy

$$\begin{cases} y'(t) = 3y(t) - 4e^{-t}, & t \in [0;1], \\ y(0) = 1 + \varepsilon. \end{cases}$$

1. Calculer la solution exacte.
2. Utiliser la méthode RK4 pour calculer la solution approchée avec un pas $h = \frac{1}{10}$ et comparer à la solution exacte pour trois valeurs des ε, à savoir $\varepsilon = 1$, $\varepsilon = \frac{1}{10}$ et $\varepsilon = 0$.
3. Même exercice avec la méthode d'Euler explicite.
4. Commenter les résultats.

6.1 Correction : solution exacte (avec **SymPy**)

La solution exacte est $y(t) = \varepsilon e^{3t} + e^{-t}$.

Si $\varepsilon = 0$, la solution exacte devient donc $y(t) = e^{-t}$ mais le problème est **mal conditionné** car toute (petite) erreur de calcul a le même effet qu'une perturbation de la condition initiale: on "réveil" le terme dormant e^{3t}.

In [7]:
```
%reset -f

import sympy as sym

sym.init_printing(pretty_print=True)

sym.var('t,epsilon')
y=sym.Function('y')

edo=sym.Eq(sym.diff(y(t),t),  3*y(t)-4*sym.exp(-t) )

print("EDO")
display(edo)

print("\nSolution generale")
soln=sym.dsolve(edo,y(t))
display(soln.expand())

print("\nCI =>")
t0 = 0
y0 = 1+epsilon
constants = sym.solve([soln.rhs.subs(t,t0) - y0])
display(constants)

print("\nSolution particuliere")
sym.var('C1')
solp = soln.subs(constants)
display(solp.expand())
```

EDO

$$\frac{d}{dt}y(t) = 3y(t) - 4e^{-t}$$

Solution generale

$$y(t) = C_1 e^{3t} + e^{-t}$$

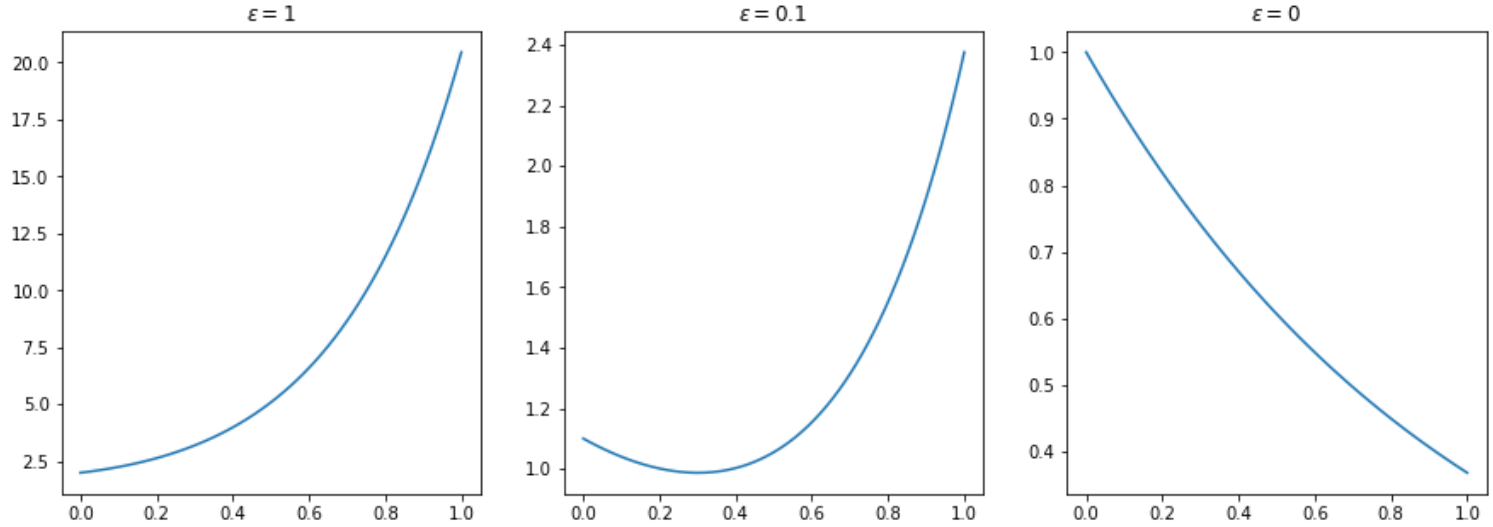
CI =>

$$\{C_1 : \epsilon\}$$

Solution particuliere

$$y(t) = \epsilon e^{3t} + e^{-t}$$

In [8]:
```
func = sym.lambdify((t,epsilon),solp.rhs,'numpy')

import matplotlib.pylab as pl
tt=pl.linspace(0,1,101)

pl.figure(1,figsize=(15,5))
EPSILON=[1,1.e-1,0]
i=1
for epsi in EPSILON:
    pl.subplot(1,3,i)
    pl.plot(tt,func(tt,epsi))
    pl.title("$\epsilon=$"+str(epsi))
    pl.legend();
    i+=1
```

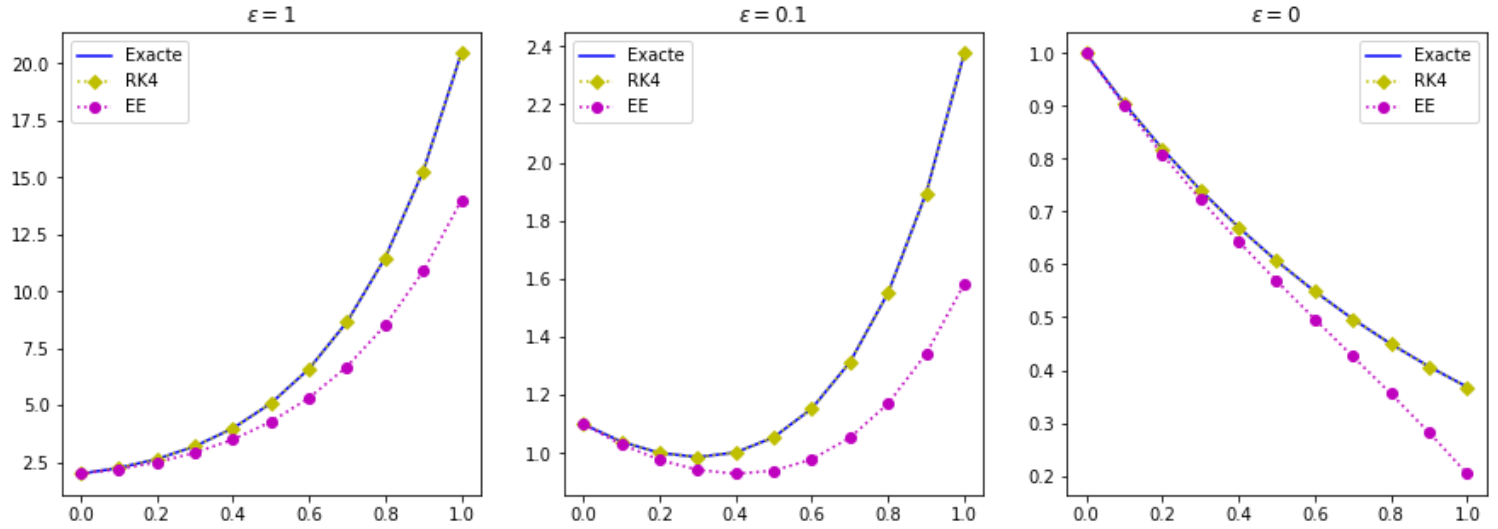


6.2 Correction : solution approchée (avec RK41 et EE)

In [9]:

```
%reset -f
%matplotlib inline

from matplotlib.pylab import *


phi = lambda t,y : 3*y-4*exp(-t)
sol_exacte = lambda t,y0 : (y0-1)*exp(3.*t)+exp(-t)


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

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] + (k1+2*k2+2*k3+k4)*h/6 )
    return uu

# INITIALISATION
t0 = 0.
tfinal = 1.
# DISCRETISATION
N = 10
h = (tfinal-t0)/N
tt = [ t0+i*h for i in range(N+1) ]


EPSILON=[1,1.e-1,0]

figure(1,figsize=(15,5))
i=1
for epsilon in EPSILON:
    y0 = 1.+epsilon
    yy = [sol_exacte(t,y0) for t in tt]
    uu_RK4 = RK4(phi,tt)
    uu_EE = EE(phi,tt)
    subplot(1,3,i)
    plot(tt,yy,'b-',tt,uu_RK4,'y:D',tt,uu_EE,'m:o')
    legend(['Exacte','RK4','EE']);
    title("$\epsilon=$"+str(epsilon))
```

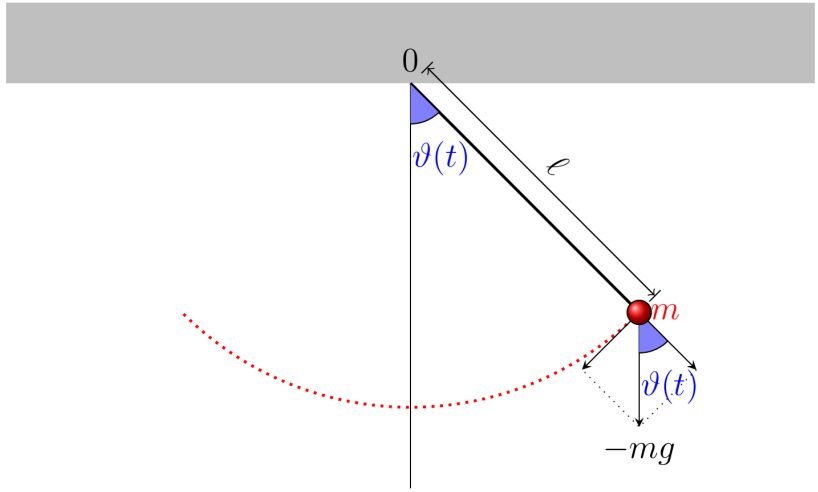
```
    i+=1
```



Type *Markdown* and LaTeX: α^2

7 Exercice : Étude théorique et numérique du mouvement d'un pendule

Considérons le problème du mouvement d'un pendule de masse m suspendu au point O par un fil non pesant de longueur ℓ.
On se propose d'étudier l'évolution au cours du temps $t \geq 0$ de l'angle $\vartheta(t)$ autour du point 0.
L'angle $\vartheta(t)$ est mesuré par rapport à une verticale passante par 0.
Considérons pour simplifier seulement la force de gravité g.



La fonction ϑ est alors solution de l'EDO d'ordre 2

$$\vartheta''(t) = -\frac{g}{\ell}\sin(\vartheta(t)).$$

Si on introduit le paramètre ω tel que $\omega^2 = \frac{g}{\ell}$, l'équation devient

$$\vartheta''(t) + \omega^2 \sin(\vartheta(t)) = 0.$$

Pour simplicité on pose $\omega = 1$ et on note $q = \vartheta$ et $p = \vartheta'$. On peut alors transformer l'EDO d'ordre 2 en un système de deux EDO d'ordre 1:

$$\begin{cases} q'(t) = p(t), \\ p'(t) = -\sin(q(t)). \end{cases}$$

Considérons l'énergie totale du système:

$$E(q,p) = \underbrace{-\cos(q)}_{\text{Énergie potentielle}} + \underbrace{\frac{1}{2}p^2}_{\text{Énergie cinétique}}.$$

1. **Étude théorique**
   - Montre analytiquement que l'énergie totale reste constante au cours du temps, i.e. que $\frac{\mathrm{d}}{\mathrm{d}t}E(q(t),p(t)) = 0$.
   - Tracer avec Python les courbes de niveau de la fonction $(q,p) \mapsto E(q,p)$, i.e. les ensembles
     $$\mathcal{N}_\kappa = \{(q,p) \mid E(q,p) = \kappa\} = \{(q,p) \mid p^2 = 2(\kappa + \cos(q))\}.$$
     pour $(q,p) \in [-\pi;\pi] \times \mathbb{R}$.
   - Lorsque ϑ est petit on peut considérer l'approximation $\sin(\vartheta) \simeq \vartheta$. Calculer la solution exacte de cette équation approchée.
2. **Étude numérique**
   Dans une simulation numérique on aimerait que la propriété de conservation de l'énergie soit préservée aussi bien que possible. Nous allons regarder ce qui se passe avec certaines méthodes vues en cours.

On notera $x_n \approx q(t_n) = \vartheta(t_n)$ et $y_n \approx p(t_n) = \vartheta'(t_n)$.
À chaque instant t_n, on calculera les valeurs approchées de l'énergie cinétique $E_c(t_n) = \frac{1}{2}y_n^2$, de l'énergie potentielle $E_p(t_n) = -\cos(x_n)$ et de l'énergie totale $E(t_n) = E_c(t_n) + E_p(t_n)$.
Choisissons $h = t_{n+1} - t_n = 0.1$ et $(x_0, y_0) = (\pi/4, 0)$.

A. Dans un premier temps on se propose d'appliquer la méthode d'**Euler explicite** à la résolution du système.

   - Tracer (t_n, x_n) et (t_n, y_n) sur un même graphique;
   - sur un autre graphique représenter les trois énergies simultanément en fonction du temps;
   - tracer enfin (x_n, y_n) sur un autre graphique et vérifier que la solution numérique tourne vers l'extérieur (i.e. l'énergie augmente au cours du temps).

B. Considérons le schéma obtenu en écrivant le schéma d'**Euler explicite** pour la première équation et le schéma d'**Euler implicite** pour la seconde.

   - Montrer que ce schéma peut s'écrire sous forme parfaitement explicite;
   - tracer (t_n, x_n) et (t_n, y_n) sur un même graphique;
   - sur un autre graphique représenter les trois énergies simultanément en fonction du temps;
   - tracer enfin (x_n, y_n) sur un autre graphique. Que peut-on constater? Commenter les résultats.

C. On se propose d'appliquer les méthodes d'**Euler modifiée, de Heun, AB2, AB3 et RK4** à la résolution du système. Pour chaque méthode:

   - Tracer (t_n, x_n) et (t_n, y_n) sur un même graphique;
   - sur un autre graphique représenter les trois énergies simultanément en fonction du temps;
   - tracer enfin (x_n, y_n) sur un autre graphique. Que peut-on constater? Commenter les résultats.

7.1 Correction étude théorique

Il s'agit d'un système de deux EDO scalaires d'ordre 1 du type

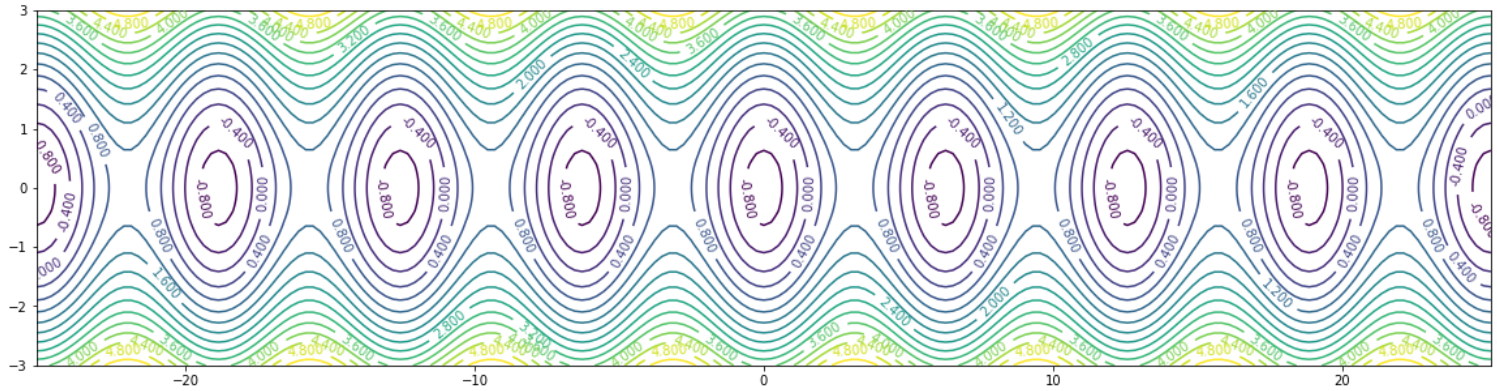
$$\begin{cases} q'(t) = \varphi_1(t, q(t), p(t)), \\ p'(t) = \varphi_2(t, q(t), p(t)), \end{cases} \quad \text{avec} \quad \begin{cases} \varphi_1(t, q(t), p(t)) = p(t), \\ \varphi_2(t, q(t), p(t)) = -\sin(q(t)). \end{cases}$$

- L'énergie totale reste constante au cours du temps:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}E(q(t), p(t)) &= \sin(q(t))q'(t) + p(t)p'(t) = \sin(q(t))p(t) + p(t)p'(t) \\ &= p(t)\left(\sin(q(t)) + p'(t)\right) = p(t)\left(\sin(q(t)) - \sin(q(t))\right) = 0. \end{aligned}$$

- Courbes de niveau de la fonction $(q, p) \mapsto E(q, p)$:

In [10]:
```
from matplotlib.pylab import *

Etot = lambda q,p : -cos(q)+p**2/2

q,p = meshgrid(linspace(-8*pi,8*pi,201),linspace(-3,3,201))
E = Etot(q,p)

figure(1,figsize=(20,5))
graphe=contour(q,p,E,20)
clabel(graphe,inline=1);
```



- $\vartheta''(t) + \omega^2 \vartheta(t) = 0$ est une EDO d'ordre 2 homogène à coefficients constants dont la solution est $\vartheta(t) = \vartheta(0)\cos(\omega t) + \frac{\vartheta'(0)}{\omega}\sin(\omega t)$.

7.2 Correction étude numérique

In [11]:
```
t0 = 0
tfinal = 8*pi
y0 = [pi/4,0]

N = 100
tt = linspace(t0,tfinal,N+1)
h = tt[1]-tt[0]

phi1 = lambda t,q,p : p
phi2 = lambda t,q,p : -sin(q)
```

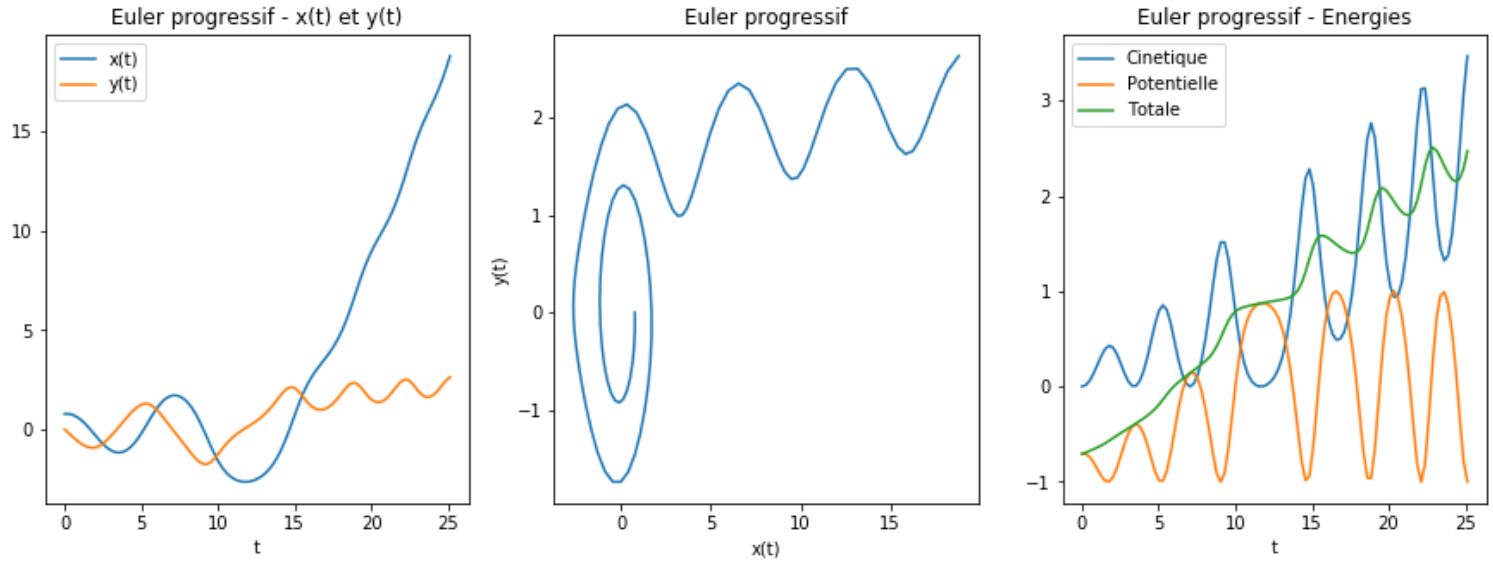
7.2.1 Méthode d'Euler explicite

$$\begin{cases} x_0 = q(0), \\ y_0 = p(0), \\ \\ x_{n+1} = x_n + h\varphi_1(t_n, x_n, y_n), \\ y_{n+1} = y_n + h\varphi_2(t_n, x_n, y_n). \end{cases}$$

In [12]:

```
def euler_progressif(phi1,phi2,tt):
    q = [y0[0]]
    p = [y0[1]]
    for i in range(len(tt)-1):
        q.append(q[i]+h*phi1(tt[i],q[i],p[i]))
        p.append(p[i]+h*phi2(tt[i],q[i],p[i]))
    return [q,p]
[q_ep, p_ep] = euler_progressif(phi1,phi2,tt)
Ec_ep = [p**2/2 for p in p_ep]
Ep_ep = [-math.cos(q) for q in q_ep]
Et_ep = [Ec_ep[i]+Ep_ep[i] for i in range(len(Ec_ep))]
print ('Euler explicite : |energie totale (t=Tfinal) - Energie totale (t=0)|=', abs(Et_ep[-1]-Et_ep[

figure(1,figsize=(15,5))
subplot(131)
plot(tt,q_ep,tt,p_ep)
xlabel('t')
legend(['x(t)','y(t)'])
title('Euler progressif - x(t) et y(t)')
subplot(132)
plot(q_ep,p_ep)
xlabel('x(t)')
ylabel('y(t)')
title('Euler progressif')
subplot(133)
plot(tt,Ec_ep,tt,Ep_ep,tt,Et_ep)
xlabel('t')
legend(['Cinetique','Potentielle','Totale'])
title('Euler progressif - Energies');
```

```
Euler explicite : |energie totale (t=Tfinal) - Energie totale (t=0)|= 3.1764943624320936
```



7.2.2 Méthode d'Euler explicite/implicite

Schéma obtenu en écrivant le schéma d'Euler explicite pour la première équation et le schéma d'Euler implicite pour la seconde:

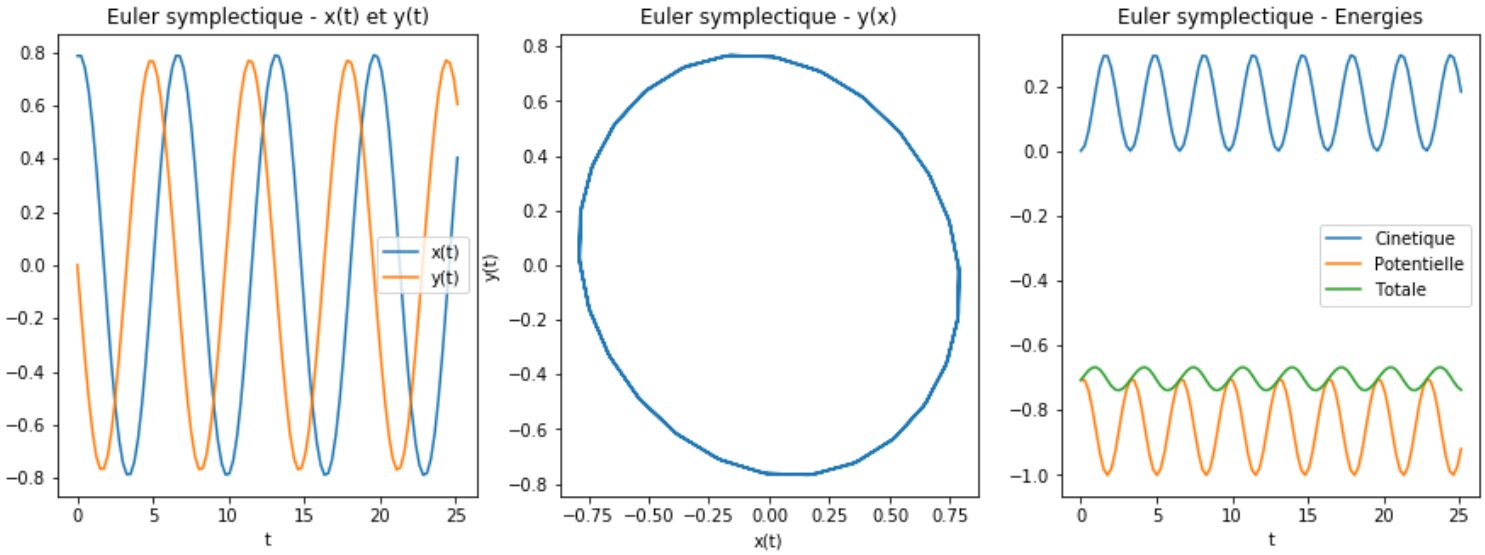
$$\begin{cases} x_0 = q(0), \\ y_0 = p(0), \\ \\ x_{n+1} = x_n + h\varphi_1(t_n, x_n, y_n), \\ y_{n+1} = y_n + h\varphi_2(t_{n+1}, x_{n+1}, y_{n+1}). \end{cases}$$

Dans notre cas $\varphi_2(t_{n+1}, x_{n+1}, y_{n+1}) = -\sin(x_{n+1})$ donc le schéma est parfaitement explicite:

In [13]:

```
def euler_symplectique(phi1,phi2,tt):
    q = [y0[0]]
    p = [y0[1]]
    for i in range(len(tt)-1):
        q.append(q[i]+h*phi1(tt[i],q[i],p[i]))
        p.append(p[i]+h*phi2(tt[i+1],q[i+1],p[i])) # il faudrait p[i+1] (schema implicite) mais comm
    return [q,p]
[q_es, p_es] = euler_symplectique(phi1,phi2,tt)
Ec_es = [p**2/2 for p in p_es]
Ep_es = [-math.cos(q) for q in q_es]
Et_es = [Ec_es[i]+Ep_es[i] for i in range(len(Ec_es))]
print ('Euler symplectique : |energie totale (t=Tfinal)-Energie totale (t=0)|=', abs(Et_es[-1]-Et_es

figure(1,figsize=(15,5))
subplot(131)
plot(tt,q_es,tt,p_es)
xlabel('t')
legend(['x(t)','y(t)'])
title('Euler symplectique - x(t) et y(t)')
subplot(132)
plot(q_es,p_es)
xlabel('x(t)')
ylabel('y(t)')
title('Euler symplectique - y(x)')
subplot(133)
plot(tt,Ec_es,tt,Ep_es,tt,Et_es)
xlabel('t')
legend(['Cinetique','Potentielle','Totale'])
title('Euler symplectique - Energies');
```

```
Euler symplectique : |energie totale (t=Tfinal)-Energie totale (t=0)|= 0.030048843024607974
```



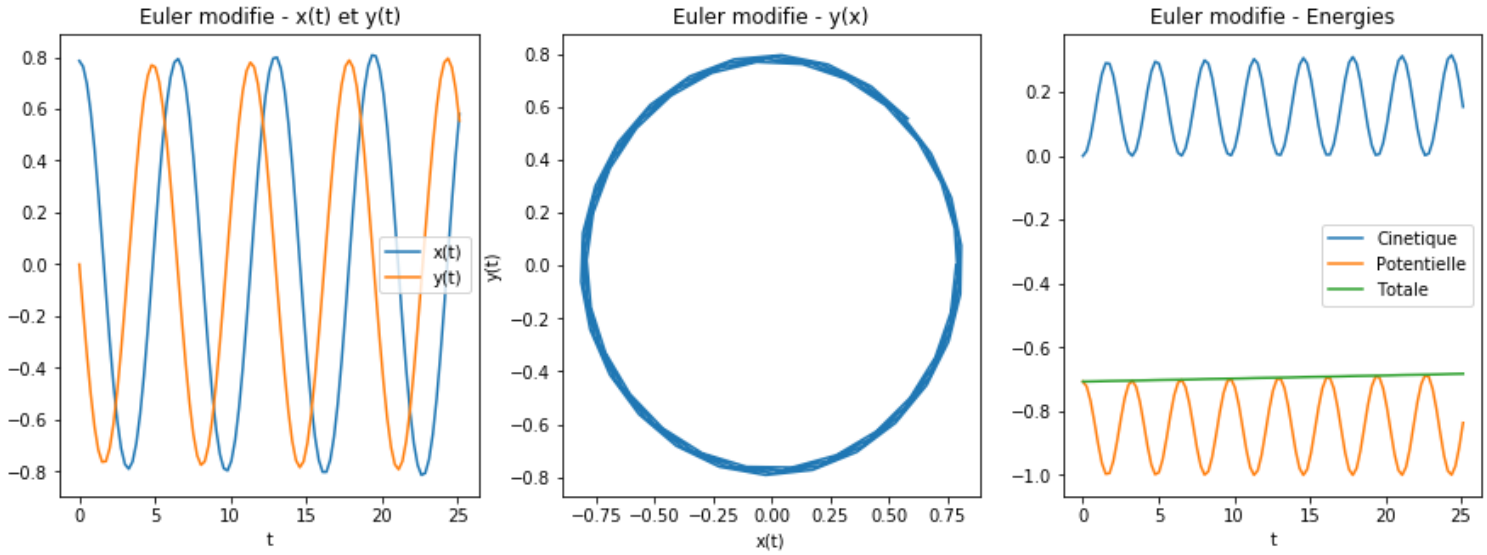
7.2.3 Méthodes d'Euler modifiée

$$\begin{cases} x_0 = q(0), \\ y_0 = p(0), \\ \\ \tilde{x}_n = x_n + \frac{h}{2}\varphi_1(t_n, x_n, y_n), \\ \tilde{y}_n = y_n + \frac{h}{2}\varphi_2(t_n, x_n, y_n), \\ \\ x_{n+1} = x_n + h\varphi_1(t_n, \tilde{x}_n, \tilde{y}_n), \\ y_{n+1} = y_n + h\varphi_2(t_n, \tilde{x}_n, \tilde{y}_n). \end{cases}$$

In [14]:

```
def euler_modifie(phi1,phi2,tt):
    q = [y0[0]]
    p = [y0[1]]
    for i in range(len(tt)-1):
        qtilde = phi1( tt[i], q[i], p[i] )
        ptilde = phi2( tt[i], q[i], p[i] )
        q.append( q[i]+h*phi1(tt[i]+h/2.,q[i]+qtilde*h/2.,p[i]+ptilde*h/2.) )
        p.append( p[i]+h*phi2(tt[i]+h/2.,q[i]+qtilde*h/2.,p[i]+ptilde*h/2.) )
    return [q,p]
[q_em, p_em] = euler_modifie(phi1,phi2,tt)
Ec_em = [p**2/2 for p in p_em]
Ep_em = [-math.cos(q) for q in q_em]
Et_em = [Ec_em[i]+Ep_em[i] for i in range(len(Ec_em))]
print ('Euler modifie : |energie totale (t=Tfinal)-Energie totale (t=0)|=', abs(Et_em[-1]-Et_em[0]))

figure(1,figsize=(15,5))
subplot(131)
plot(tt,q_em,tt,p_em)
xlabel('t')
legend(['x(t)','y(t)'])
title('Euler modifie - x(t) et y(t)')
subplot(132)
plot(q_em,p_em)
xlabel('x(t)')
ylabel('y(t)')
title('Euler modifie - y(x)')
subplot(133)
plot(tt,Ec_em,tt,Ep_em,tt,Et_em)
xlabel('t')
legend(['Cinetique','Potentielle','Totale'])
title('Euler modifie - Energies');
```

```
Euler modifie : |energie totale (t=Tfinal)-Energie totale (t=0)|= 0.02457609735191002
```

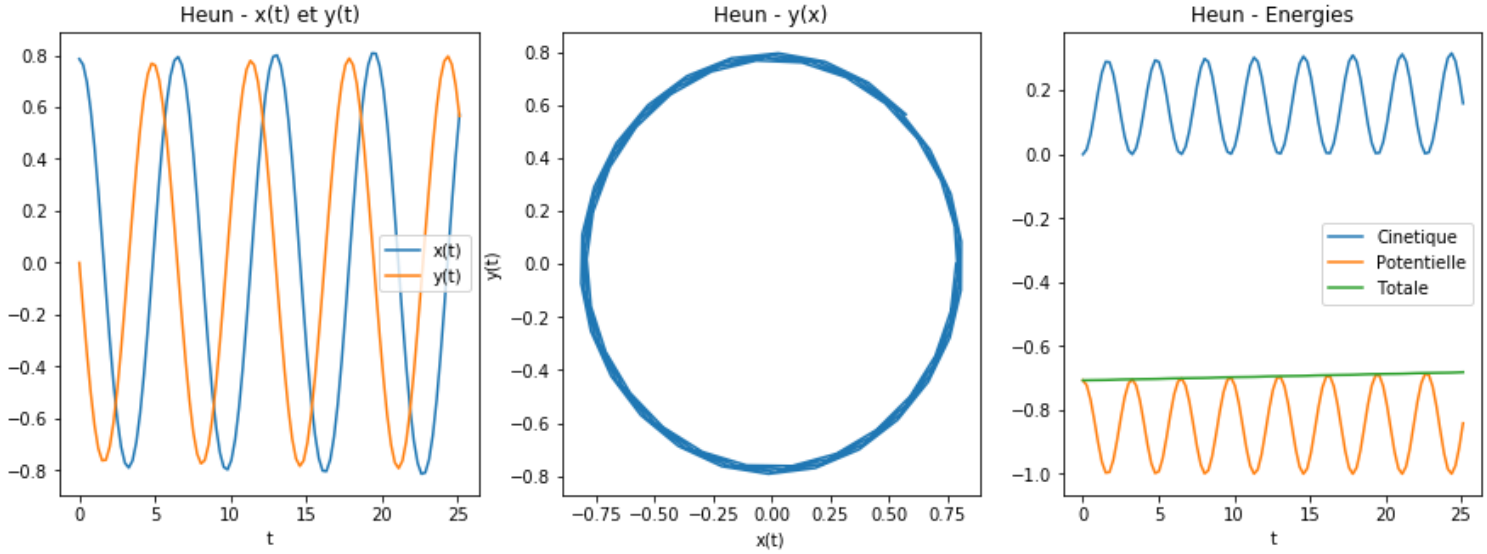


7.2.4 Méthodes de Heun

$$
\begin{cases}
x_0 = q(0), \\
y_0 = p(0), \\
\tilde{x}_n = x_n + h\varphi_1(t_n, x_n, y_n), \\
\tilde{y}_n = y_n + h\varphi_2(t_n, x_n, y_n), \\
x_{n+1} = x_n + \frac{h}{2}\left(\varphi_1(t_n, x_n, y_n) + \varphi_1(t_{n+1}, \tilde{x}_n, \tilde{y}_n)\right), \\
y_{n+1} = y_n + \frac{h}{2}\left(\varphi_2(t_n, x_n, y_n) + \varphi_2(t_{n+1}, \tilde{x}_n, \tilde{y}_n)\right).
\end{cases}
$$

In [15]:
```
def heun(phi1,phi2,tt):
    q = [y0[0]]
    p = [y0[1]]
    for i in range(len(tt)-1):
        qtilde = phi1( tt[i], q[i], p[i] )
        ptilde = phi2( tt[i], q[i], p[i] )
        q.append( q[i]+h*0.5*( phi1(tt[i],q[i],p[i]) + phi1(tt[i+1],q[i]+h*qtilde,p[i]+h*ptilde) ) )
        p.append( p[i]+h*0.5*( phi2(tt[i],q[i],p[i]) + phi2(tt[i+1],q[i]+h*qtilde,p[i]+h*ptilde) ) )
    return [q,p]
[q_heun, p_heun] = heun(phi1,phi2,tt)
Ec_heun = [p**2/2 for p in p_heun]
Ep_heun = [-math.cos(q) for q in q_heun]
Et_heun = [Ec_heun[i]+Ep_heun[i] for i in range(len(Ec_heun))]
print ('Heun : |energie totale (t=Tfinal)-Energie totale (t=0)|=', abs(Et_heun[-1]-Et_heun[0]))

figure(1,figsize=(15,5))
subplot(131)
plot(tt,q_heun,tt,p_heun)
xlabel('t')
legend(['x(t)','y(t)'])
title('Heun - x(t) et y(t)')
subplot(132)
plot(q_heun,p_heun)
xlabel('x(t)')
ylabel('y(t)')
title('Heun - y(x)')
subplot(133)
plot(tt,Ec_heun,tt,Ep_heun,tt,Et_heun)
xlabel('t')
legend(['Cinetique','Potentielle','Totale'])
title('Heun - Energies');
```

```
Heun : |energie totale (t=Tfinal)-Energie totale (t=0)|= 0.025432751256537878
```



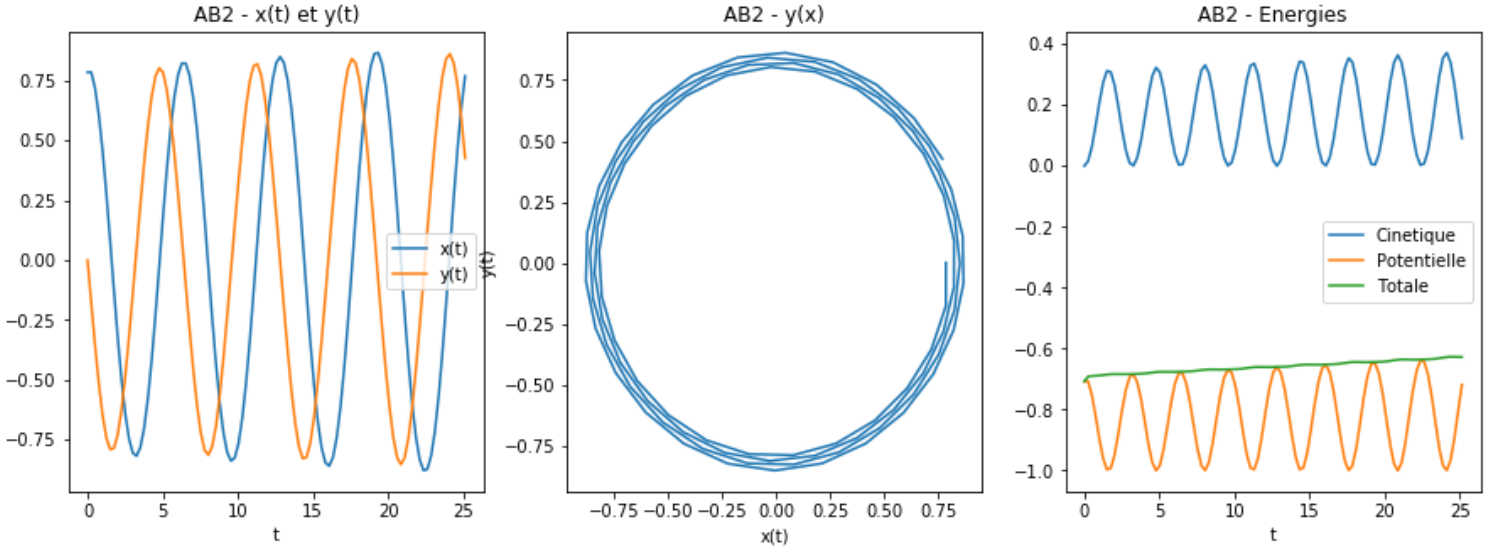
7.2.5 Méthode AB2

$$\begin{cases} x_0 = q(0), \\ y_0 = p(0), \\ \\ x_1 = x_0 + h\varphi_1(t_0, x_0, y_0), \\ y_1 = y_0 + h\varphi_2(t_0, x_0, y_0), \\ \\ x_{n+1} = x_n + \frac{h}{2}(3\varphi_1(t_n, x_n, y_n) - \varphi_1(t_{n-1}, x_{n-1}, y_{n-1})), \\ y_{n+1} = y_n + \frac{h}{2}(3\varphi_2(t_n, x_n, y_n) - \varphi_2(t_{n-1}, x_{n-1}, y_{n-1})). \end{cases}$$

In [16]:

```
def AB2(phi1,phi2,tt):
    q = [y0[0]]
    p = [y0[1]]
    q.append(q[0]+h*phi1(tt[0],q[0],p[0]))
    p.append(p[0]+h*phi2(tt[0],q[0],p[0]))
    for i in range(1,len(tt)-1):
        qtilde1 = phi1( tt[i],q[i],p[i] )
        qtilde2 = phi1( tt[i-1],q[i-1],p[i-1] )
        ptilde1 = phi2( tt[i],q[i],p[i] )
        ptilde2 = phi2( tt[i-1],q[i-1],p[i-1] )
        q.append( q[i] + (3*qtilde1-qtilde2)*h/2 )
        p.append( p[i] + (3*ptilde1-ptilde2)*h/2 )
    return [q,p]
[q_AB2, p_AB2] = AB2(phi1,phi2,tt)
Ec_AB2 = [p**2/2. for p in p_AB2]
Ep_AB2 = [-math.cos(q) for q in q_AB2]
Et_AB2 = [Ec_AB2[i]+Ep_AB2[i] for i in range(len(Ec_AB2))]
print ('AB2 : |energie totale (t=Tfinal)-Energie totale (t=0)|=', abs(Et_AB2[-1]-Et_AB2[0]))

figure(1,figsize=(15,5))
subplot(131)
plot(tt,q_AB2,tt,p_AB2)
xlabel('t')
legend(['x(t)','y(t)'])
title('AB2 - x(t) et y(t)')
subplot(132)
plot(q_AB2,p_AB2)
xlabel('x(t)')
ylabel('y(t)')
title('AB2 - y(x)')
subplot(133)
plot(tt,Ec_AB2,tt,Ep_AB2,tt,Et_AB2)
xlabel('t')
legend(['Cinetique','Potentielle','Totale'])
title('AB2 - Energies');
```

```
AB2 : |energie totale (t=Tfinal)-Energie totale (t=0)|= 0.07960924681985027
```

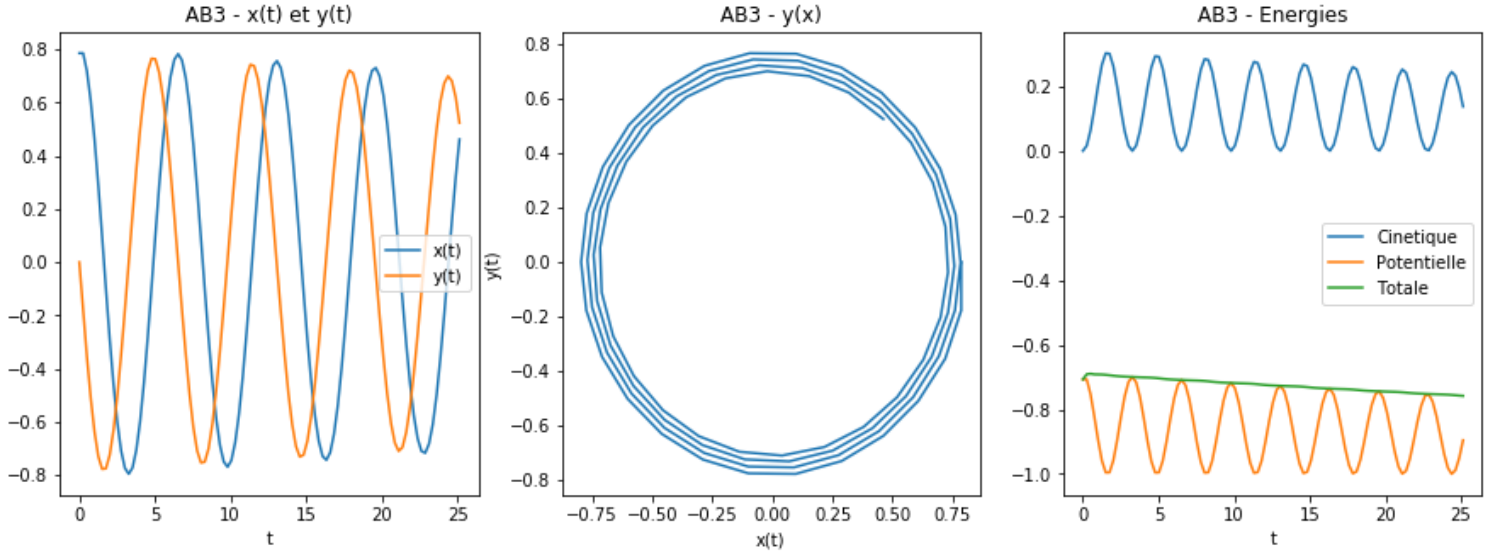


7.2.6 Méthode AB3

$$
\begin{cases}
x_0 = q(0), \\
y_0 = p(0), \\
\\
x_1 = x_0 + h\varphi_1(t_0, x_0, y_0), \\
y_1 = y_0 + h\varphi_2(t_0, x_0, y_0), \\
\\
x_2 = x_1 + \frac{h}{2}(3\varphi_1(t_1, x_1, y_1) - \varphi_1(t_0, x_0, y_0)), \\
y_2 = y_1 + \frac{h}{2}(3\varphi_2(t_1, x_1, y_1) - \varphi_2(t_0, x_0, y_0)), \\
\\
x_{n+1} = x_n + \frac{h}{12}(23\varphi_1(t_n, x_n, y_n) - 16\varphi_1(t_{n-1}, x_{n-1}, y_{n-1}) + 5\varphi_1(t_{n-2}, x_{n-2}, y_{n-2})), \\
y_{n+1} = y_n + \frac{h}{12}(23\varphi_2(t_n, x_n, y_n) - 16\varphi_2(t_{n-1}, x_{n-1}, y_{n-1}) + 5\varphi_2(t_{n-2}, x_{n-2}, y_{n-2})).
\end{cases}
$$

In [17]:
```
def AB3(phi1,phi2,tt):
    q = [y0[0]]
    p = [y0[1]]
    q.append(q[0]+h*phi1(tt[0],q[0],p[0]))
    p.append(p[0]+h*phi2(tt[0],q[0],p[0]))
    q.append(q[1]+h*(3*phi1(tt[1],q[1],p[1])-phi1(tt[0],q[0],p[0]))/2. )
    p.append(p[1]+h*(3*phi2(tt[1],q[1],p[1])-phi2(tt[0],q[0],p[0]))/2.)
    for i in range(2,len(tt)-1):
        qtilde1 = phi1( tt[i],q[i],p[i] )
        qtilde2 = phi1( tt[i-1],q[i-1],p[i-1] )
        qtilde3 = phi1( tt[i-2],q[i-2],p[i-2] )
        ptilde1 = phi2( tt[i],q[i],p[i] )
        ptilde2 = phi2( tt[i-1],q[i-1],p[i-1] )
        ptilde3 = phi2( tt[i-2],q[i-2],p[i-2] )
        q.append( q[i] + (23*qtilde1-16*qtilde2+5*qtilde3)*h/12 )
        p.append( p[i] + (23*ptilde1-16*ptilde2+5*ptilde3)*h/12 )
    return [q,p]
[q_AB3, p_AB3] = AB3(phi1,phi2,tt)
Ec_AB3 = [p**2/2. for p in p_AB3]
Ep_AB3 = [-math.cos(q) for q in q_AB3]
Et_AB3 = [Ec_AB3[i]+Ep_AB3[i] for i in range(len(Ec_AB3))]
print( 'AB3 : |energie totale (t=Tfinal)-Energie totale (t=0)|=', abs(Et_AB3[-1]-Et_AB3[0]))

figure(1,figsize=(15,5))
subplot(131)
plot(tt,q_AB3,tt,p_AB3)
xlabel('t')
legend(['x(t)','y(t)'])
title('AB3 - x(t) et y(t)')
subplot(132)
plot(q_AB3,p_AB3)
xlabel('x(t)')
ylabel('y(t)')
title('AB3 - y(x)')
subplot(133)
plot(tt,Ec_AB3,tt,Ep_AB3,tt,Et_AB3)
xlabel('t')
legend(['Cinetique','Potentielle','Totale'])
title('AB3 - Energies');
```

```
AB3 : |energie totale (t=Tfinal)-Energie totale (t=0)|= 0.051119920140796915
```



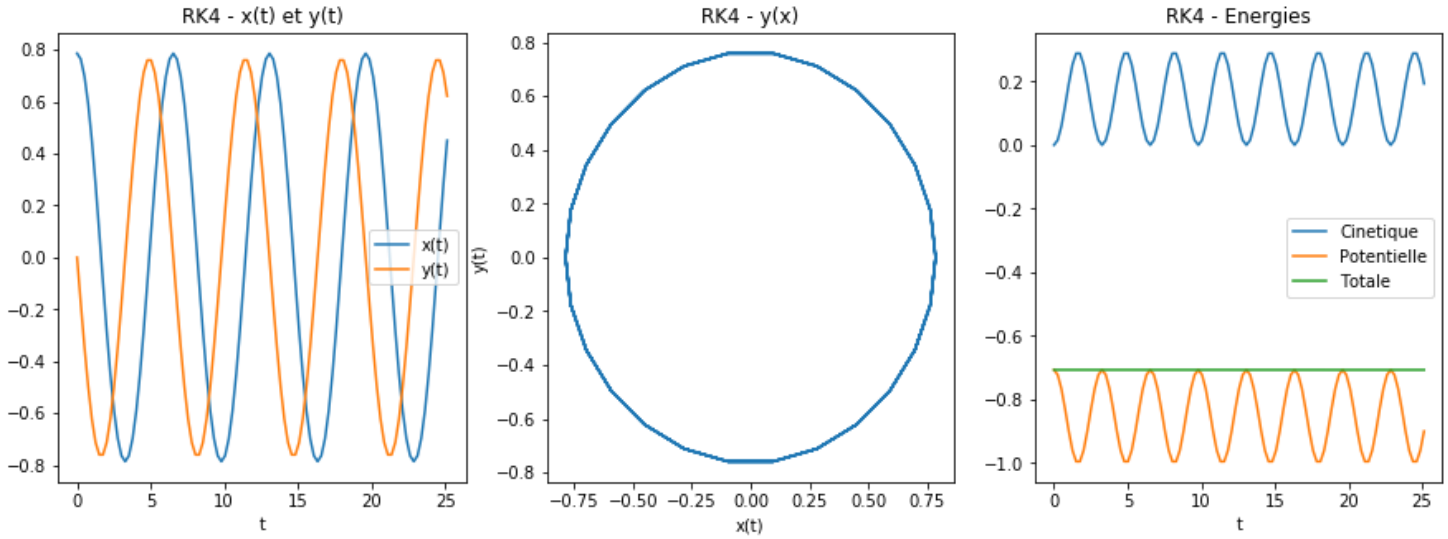
7.2.7 Méthode RK4

$$
\begin{cases}
x_0 = q(0), \\
y_0 = p(0), \\
\tilde{x}_n = x_n + \frac{h}{2}\varphi_1(t_n, x_n, y_n), \\
\tilde{y}_n = y_n + \frac{h}{2}\varphi_2(t_n, x_n, y_n), \\
\check{x}_n = x_n + \frac{h}{2}\varphi_1(t_n + h/2, \tilde{x}_n, \tilde{y}_n), \\
\check{y}_n = y_n + \frac{h}{2}\varphi_2(t_n + h/2, \tilde{x}_n, \tilde{y}_n), \\
\hat{x}_n = x_n + h\varphi_1(t_{n+1}, \check{x}_n, \check{y}_n), \\
\hat{y}_n = y_n + h\varphi_2(t_{n+1}, \check{x}_n, \check{y}_n), \\
x_{n+1} = x_n + \frac{h}{6}\left(\varphi_1(t_n, x_n, y_n) + 2\varphi_1(t_n + h/2, \tilde{x}_n, \tilde{y}_n) + 2\varphi_1(t_n + h/2, \check{x}_n, \check{y}_n) + \varphi_1(t_{n+1}, \hat{x}_n, \hat{y}_n)\right), \\
y_{n+1} = y_n + \frac{h}{6}\left(\varphi_2(t_n, x_n, y_n) + 2\varphi_2(t_n + h/2, \tilde{x}_n, \tilde{y}_n) + 2\varphi_2(t_n + h/2, \check{x}_n, \check{y}_n) + \varphi_2(t_{n+1}, \hat{x}_n, \hat{y}_n)\right).
\end{cases}
$$

In [18]:

```
def RK4(phi1,phi2,tt):
    q = [y0[0]]
    p = [y0[1]]
    for i in range(len(tt)-1):
        qtilde = q[i] + 0.5*h * phi1( tt[i], q[i], p[i] )
        ptilde = p[i] + 0.5*h * phi2( tt[i], q[i], p[i] )
        qv = q[i] + 0.5*h * phi1( tt[i]+h*0.5, qtilde, ptilde )
        pv = p[i] + 0.5*h * phi2( tt[i]+h*0.5, qtilde, ptilde )
        qhat = q[i] + h * phi1( tt[i+1], qv, pv )
        phat = p[i] + h * phi2( tt[i+1], qv, pv )
        q.append( q[i]+h/6*( phi1(tt[i],q[i],p[i]) + 2*phi1(tt[i]+0.5*h,qtilde,ptilde)+ 2*phi1(tt[i]
        p.append( p[i]+h/6*( phi2(tt[i],q[i],p[i]) + 2*phi2(tt[i]+0.5*h,qtilde,ptilde)+ 2*phi2(tt[i]
    return [q,p]
[q_RK4, p_RK4] = RK4(phi1,phi2,tt)
Ec_RK4 = [p**2/2. for p in p_RK4]
Ep_RK4 = [-math.cos(q) for q in q_RK4]
Et_RK4 = [Ec_RK4[i]+Ep_RK4[i] for i in range(len(Ec_RK4))]
print ('RK4 : |energie totale (t=Tfinal)-Energie totale (t=0)|=', abs(Et_RK4[-1]-Et_RK4[0]))

figure(1,figsize=(15,5))
subplot(131)
plot(tt,q_RK4,tt,p_RK4)
xlabel('t')
legend(['x(t)','y(t)'])
title('RK4 - x(t) et y(t)')
subplot(132)
plot(q_RK4,p_RK4)
xlabel('x(t)')
ylabel('y(t)')
title('RK4 - y(x)')
subplot(133)
plot(tt,Ec_RK4,tt,Ep_RK4,tt,Et_RK4)
xlabel('t')
legend(['Cinetique','Potentielle','Totale'])
title('RK4 - Energies');
```

```
RK4 : |energie totale (t=Tfinal)-Energie totale (t=0)|= 8.32124615898211e-05
```


RK4 - x(t) et y(t)
x(t)
y(t)
t
RK4 - y(x)
y(t)
x(t)
RK4 - Energies
Cinetique
Potentielle
Totale
t

In [ ]: