In [1]:
```
%reset -f
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-Examen

# Choix d'un schéma

**Q2**
Soit un problème de Cauchy mathématiquement bien posé mais numériquement mal posé (i.e. sensible aux perturbations sur la donnée initiale). Parmi les schémas EE (Euler Explicite), EI (Euler Implicite) et H (Heun), lequel faut-il choisir? Justifier la réponse.

**Correction Q2**
Toute méthode numérique induisant une erreur sur la solution, cette erreur va s'amplifier avec les itérations succéssives puisque le problème est numériquement mal posé. La seule issue pour traiter ce genre de problème est d'utiliser des schémas d'ordre élévé (ici donc le schéma de Heun qui est d'ordre 2 tandis que les schéma d'Euler sont d'ordre 1) avec un pas $h$ petit (donc ce n'est pas la condition de A-stabilité qui limite le pas).

**Q3**
Soit le problème de Cauchy

$$\begin{cases} y'(t) = -10^5 y(t), \quad t \in [0;100] \\ y(0) = 1 \end{cases}$$

Parmi les schémas EE (Euler Explicite), EI (Euler Implicite) et H (Heun), lequel faut-il choisir? Justifier la réponse.

**Correction Q3**
Le problème est bien posé mathématiquement et numériquement.
On ne peut pas utiliser ni le schéma EE ni le schéma H car la condition de stabilité est trop contraignante: dans les deux cas il faut un pas $h < \frac{2}{10^5}$, i.e. une discrétisation avec $N > 100/h = \frac{10^7}{2}$ points.

Le schéma EI, en revanche, est inconditionnellement A-stable: on peut donc choisir le pas $h$ sans contrainte de stabilité.

# Solution exacte

Soit le problème de Cauchy

$$\begin{cases} y'(t) = 1 + \vartheta\left(t - y(t)\right), \quad t \in [0;2] \\ y(0) = 1 \end{cases}$$

Chaque sujet correspond à une valeur de $\vartheta$ parmi les suivantes:

In [2]:
```
THETA=list(range(10,54,5))
THETA
```

Out[2]:
```
[10, 15, 20, 25, 30, 35, 40, 45, 50]
```

**Q4** Calculer la solution exacte de ce problème de Cauchy.

**Correction Q4**
On peut calculer la solution à la main:

$$a(t)y'(t)+b(t)y(t)=g(t)$$

avec $a(t)=1$, $b(t)=\vartheta$ et $g(t)=1+\vartheta t$ donc

- $A(t)=\int \frac{b(t)}{a(t)}\mathrm{d}t=\int \vartheta \mathrm{d}t=\vartheta t$,
- $B(t)=\int \frac{g(t)}{a(t)} e^{A(t)}\mathrm{d}t$

$$=\int (1+\vartheta t) e^{\vartheta t}\mathrm{d}t$$
$$= t e^{\vartheta t}$$,

d'où $y(t)=ce^{-\vartheta t}+t$.

En imposant la condition $1=y(0)$ on trouve l'unique solution du problème de Cauchy donné:

$$\boxed{y(t)=e^{-\vartheta t}+t.}$$

Sinon, on peut utiliser le module de calcul symbolique mais **attention**, pour ne pas avoir des mauvaises intéractions entre le module `matplotlib` et le module `sympy`, on utilisera l'import de sympy avec un alias:

```
import sympy as sym
```

In [3]:
```
import sympy as sym
sym.init_printing()

sym.var('t,C1,theta')
u=sym.Function('u')
f=1+theta*t-theta*u(t)
edo=sym.Eq(sym.diff(u(t),t),f)
display(edo)

solgen=sym.dsolve(edo,u(t)).simplify()
display(solgen)

t0=0
u0=1
consts = sym.solve( [solgen.rhs.subs(t,t0)-u0 ],C1, dict=True)[0]
display(consts)

solpar=solgen.subs(consts)
display(solpar)
```

$$\frac{d}{dt}u(t)=t\theta-\theta u(t)+1$$

$$u(t)=C_1e^{-t\theta}+t$$

$$\{C_1:1\}$$

$$u(t)=t+e^{-t\theta}$$

# Schéma Predictor-corrector

Soit les deux schémas

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

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

**Q5** Écrire le schéma predictor-corrector associé à ce couple de schémas sous la forme d'une suite définie par récurrence.

**Correction Q5**

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

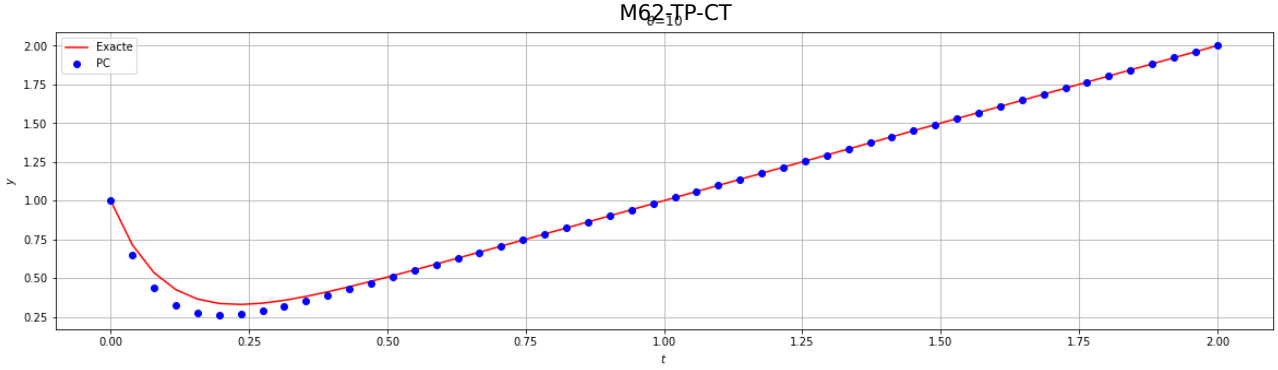
**Q6** Implémenter le schéma predictor-corrector de la question précédente et le tester avec le problème de Cauchy. On choisira $N = 1 + 5\vartheta$ points de discrétisation et on affichera la solution approchée et la solution exacte sur le même repère.

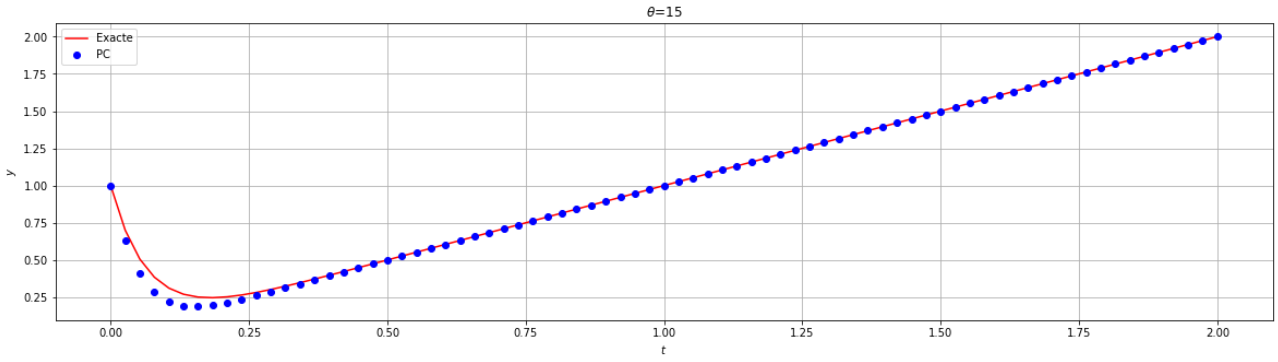
**Correction Q6**

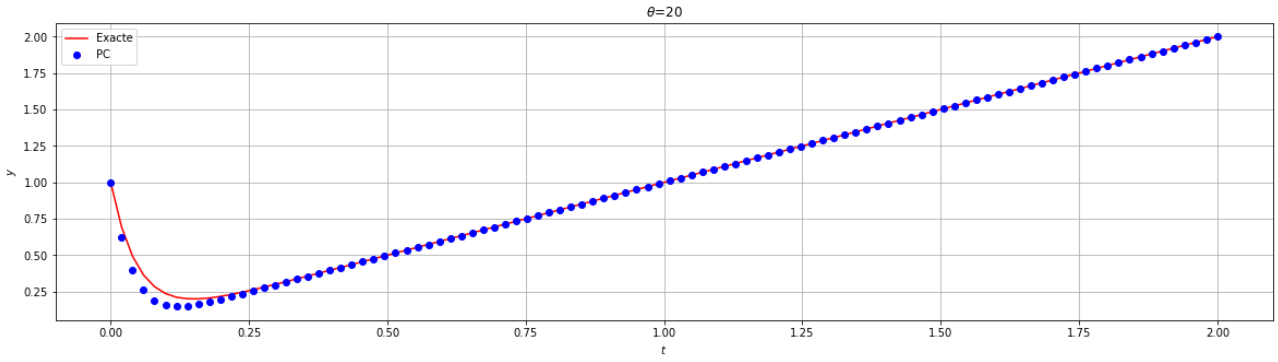
In [4]:
```
from matplotlib.pylab import *
```

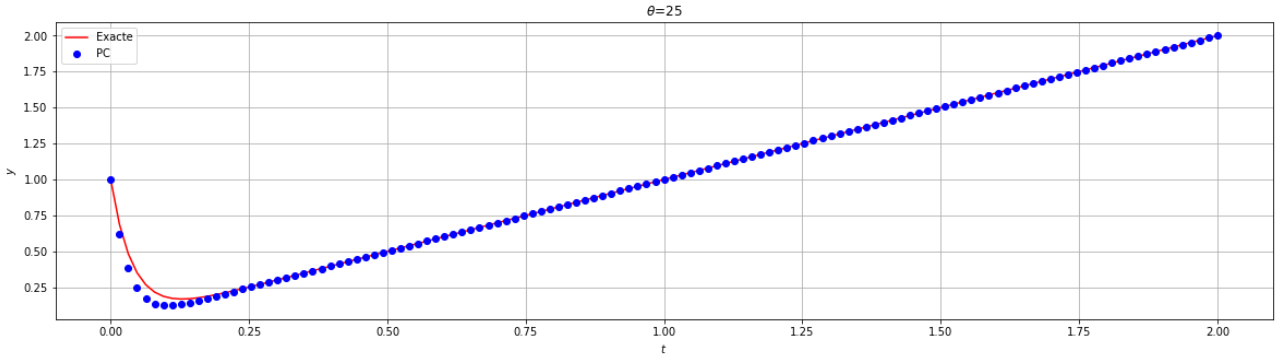
In [5]:
```
def PC(phi,tt):
    h = tt[1]-tt[0]
    uu = [y0]
    #uu.append( fsolve(lambda x : -x+uu[0]+h*phi(tt[1],x) , uu[0] )[0] )
    uu.append(uu[0]+h*phi(tt[0],uu[0]))
    for i in range(1,len(tt)-1):
        pred = uu[i] + h/2*( 3*phi(tt[i],uu[i]) - phi(tt[i-1],uu[i-1]) )
        uu.append( 4/3*uu[i] - uu[i-1]/3 + 2/3*h*phi(tt[i+1],pred) )
    return uu
```

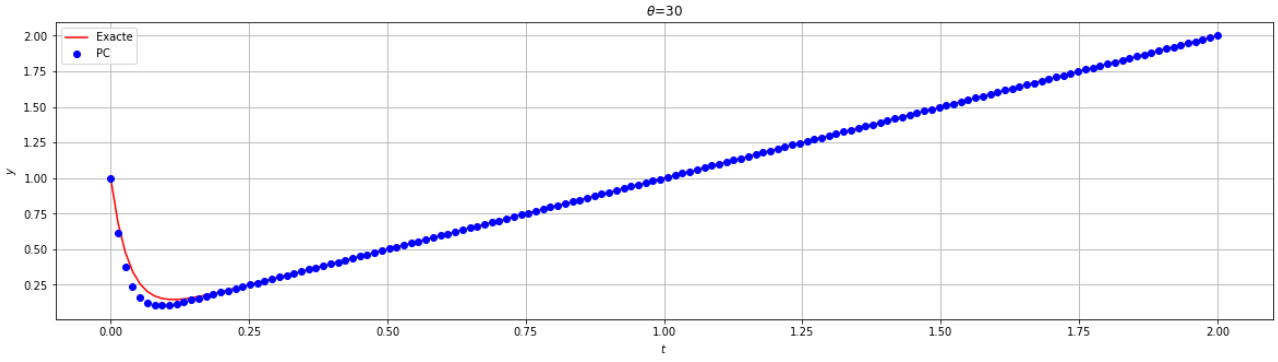
In [6]:
```
t0, y0, tfinal = 0, 1, 2

i=0
for theta in THETA:
    N = 1+5*theta
    tt = linspace(t0,tfinal,N+1)
    sol_exacte = lambda t : t+exp(-theta*t)
    phi = lambda t,y : 1+theta*(t-y)
    yy = [sol_exacte(t) for t in tt]
    uu = PC(phi,tt)
    i+=1
    figure(i,figsize=(20,5))
    plot(tt,yy,'r-',label=('Exacte'))
    plot(tt,uu,'bo',label=('PC'))
    title('$\\theta$={}'.format(theta))
    xlabel('$t$')
    ylabel('$y$')
    legend()
    grid(True)
```

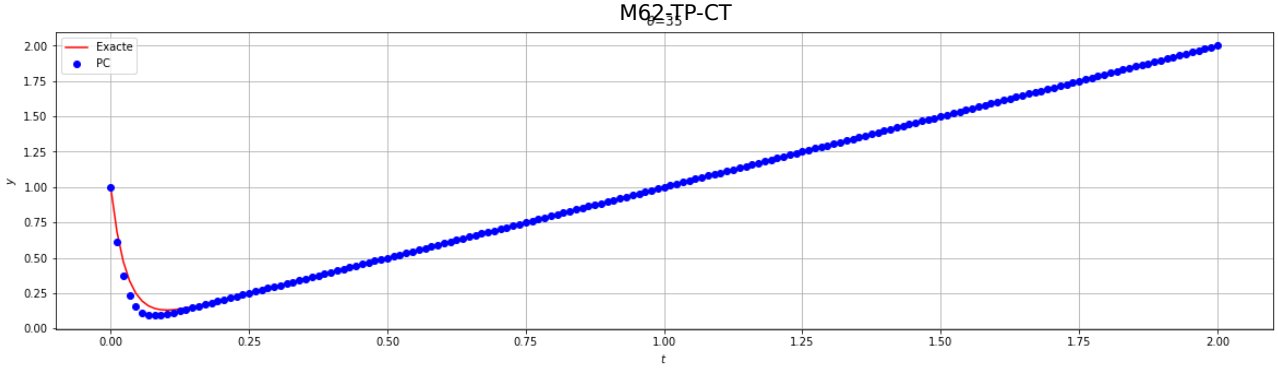

θ=10
Exacte
PC
y
t

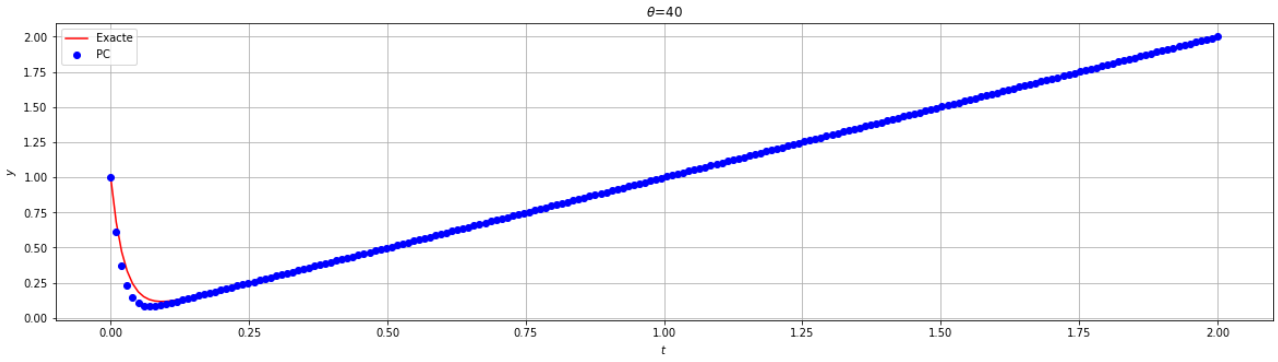

θ=15
Exacte
PC
y
t

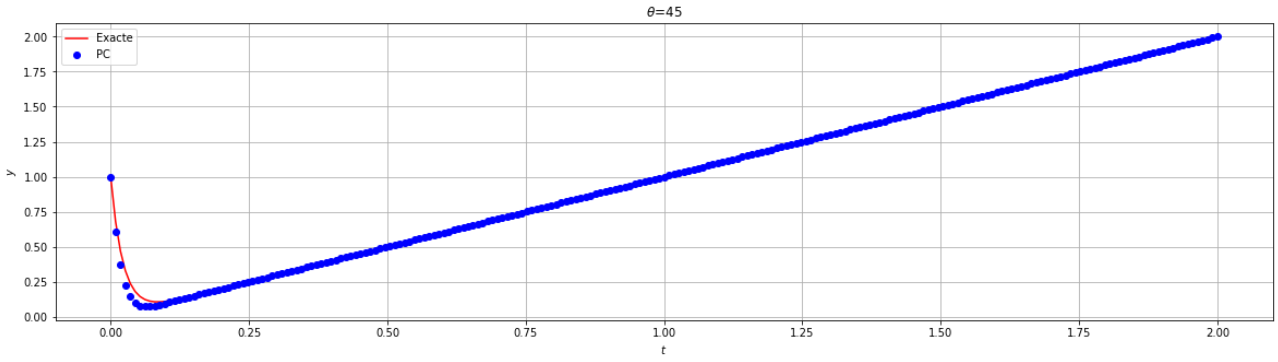

θ=20
Exacte
PC
y
t

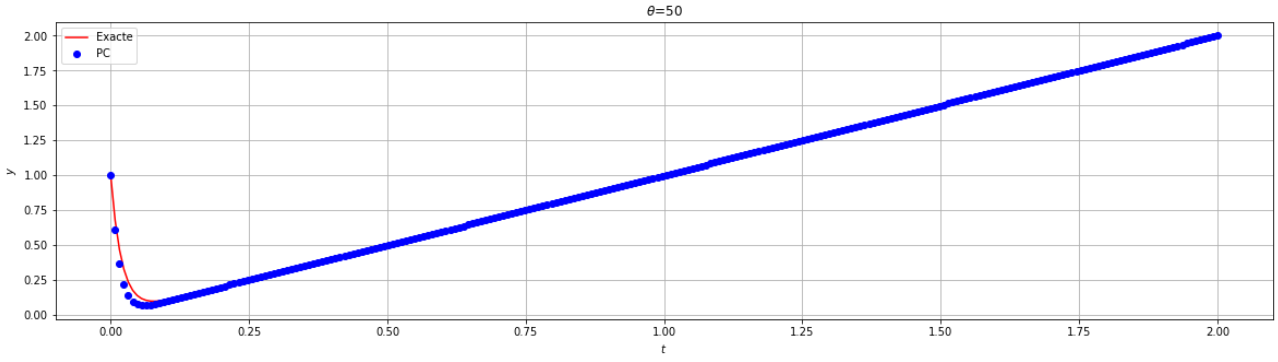

θ=25
Exacte
PC
y
t


θ=30
Exacte
PC
y
t




**Q7** Vérifier empiriquement l'ordre de convergence du schéma predictor-corrector de la question précédente en affichant la courbe de convergence et en estimant sa pente avec le problème de Cauchy.

**Correction Q7**

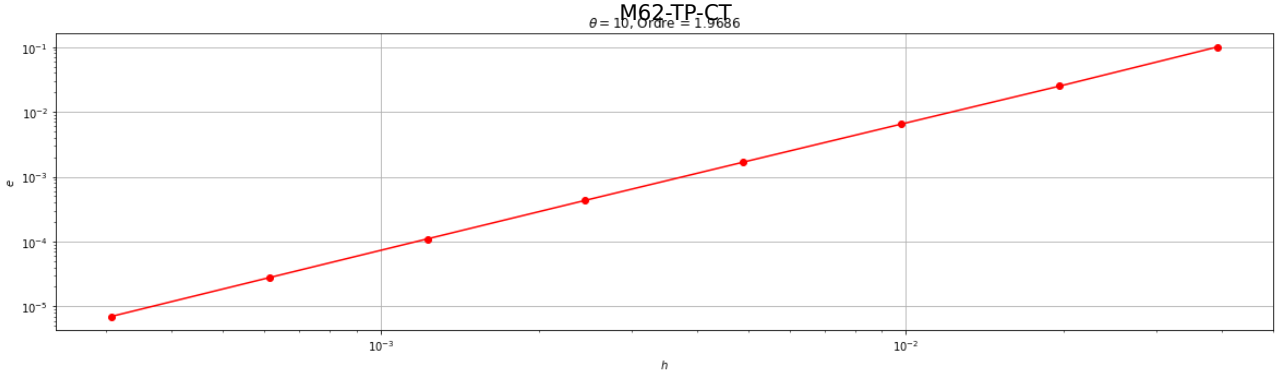
On calcule la solution approchée avec différentes valeurs de $h_k = 2/N_k$, à savoir $N_k = (1+5\vartheta)2^k$, $k = 0, \ldots 7$. On sauvegarde les valeurs de $h_k$ dans le vecteur `H`.

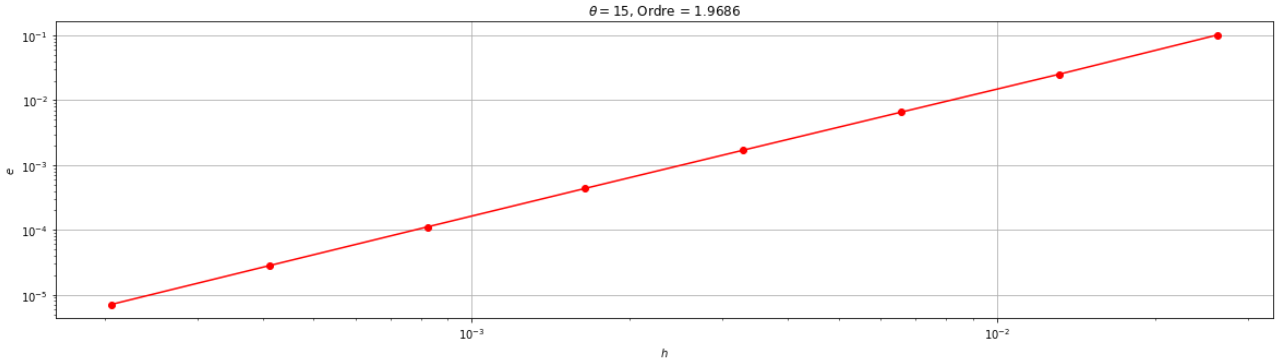
Pour chaque valeur de $h_k$, on calcule le maximum de la valeur absolue de l'erreur et on sauvegarde toutes ces erreurs dans le vecteur `err` de sort que `err[k]` contient $e_k = \max_{i=0,\ldots,N_k} |y(t_i) - u_i|$.

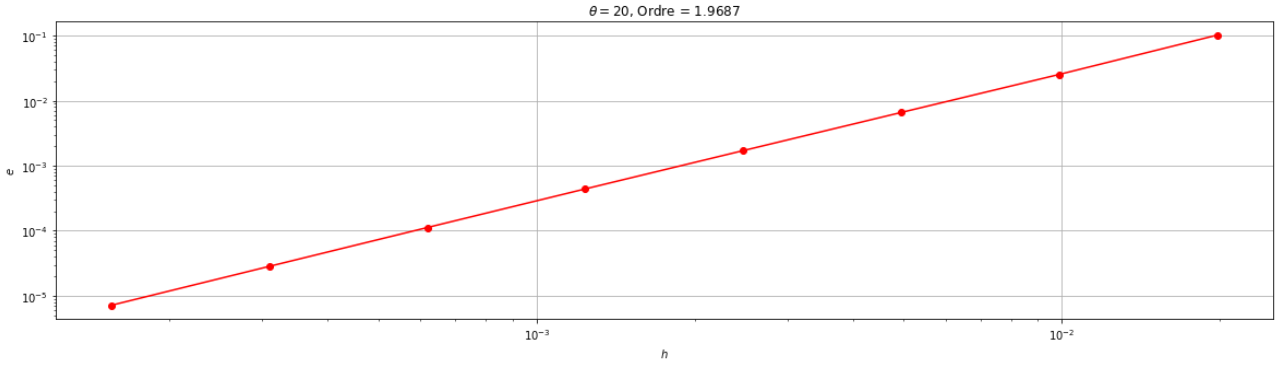
Pour afficher l'ordre de convergence on utilise une échelle logarithmique : on représente $\ln(h)$ sur l'axe des abscisses et $\ln(\mathrm{err})$ sur l'axe des ordonnées ainsi, si $\mathrm{err} = Ch^p$, alors $\ln(\mathrm{err}) = \ln(C) + p\ln(h)$. En échelle logarithmique, $p$ représente donc la pente de la ligne droite $\ln(\mathrm{err})$.

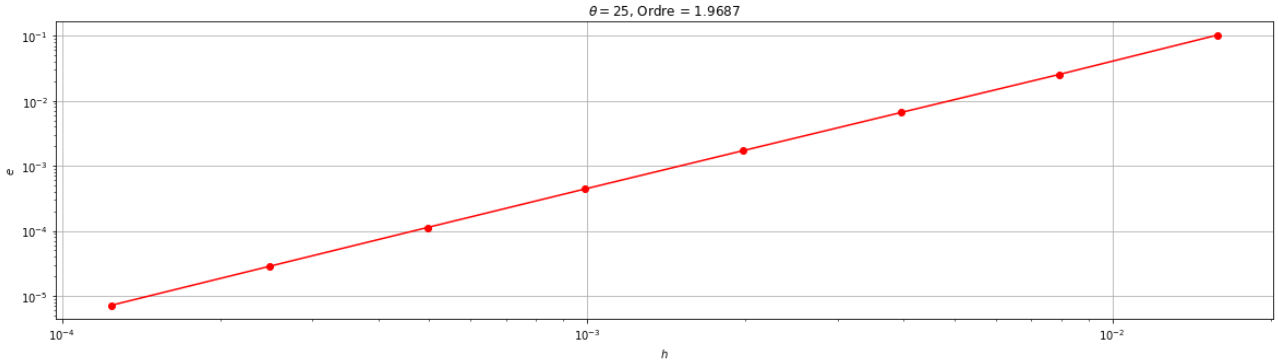
Pour estimer l'ordre de convergence on estime la pente de la droite qui relie l'erreur au pas $k$ à l'erreur au pas $k+1$ en echelle logarithmique en utilisant la fonction `polyfit` basée sur la régression linéaire.

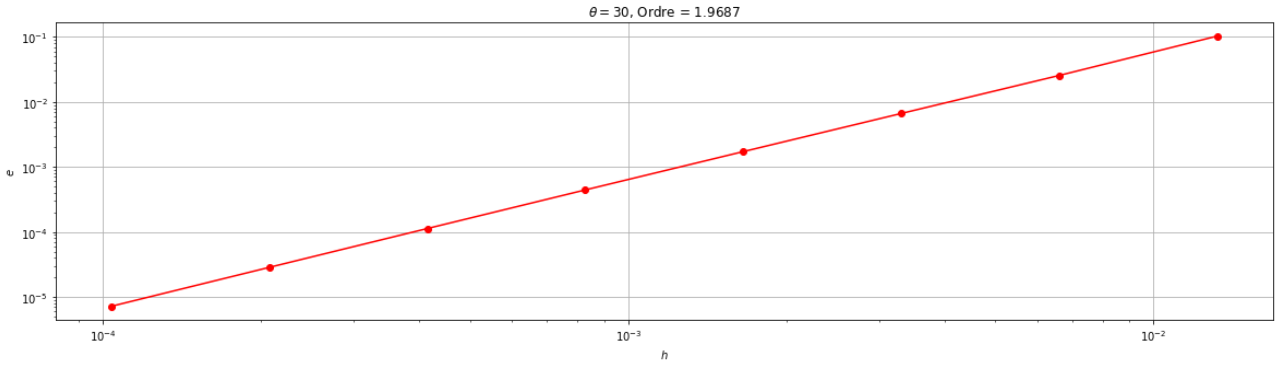
In [7]:
```
t0, y0, tfinal = 0, 1, 2

i=0
for theta in THETA:
    H = []
    err = []
    for k in range(8):
        N = (1+5*theta)*2**k
        tt = linspace(t0,tfinal,N+1)
        h = tt[1]-tt[0]
        H.append(h)
        sol_exacte = lambda t : t+exp(-theta*t)
        phi = lambda t,y : 1+theta*(t-y)
        yy = [sol_exacte(t) for t in tt]
        uu = PC(phi,tt)
        err.append(max([abs(uu[i]-yy[i]) for i in range(len(uu))]))
    i+=1
    figure(i,figsize=(20,5))
    loglog(H,err, 'r-o')
    title('$\\theta=${}, Ordre = {:1.4f}'.format(theta,polyfit(log(H),log(err), 1)[0]))
    xlabel('$h$')
    ylabel('$e$')
    legend()
    grid(True)
```

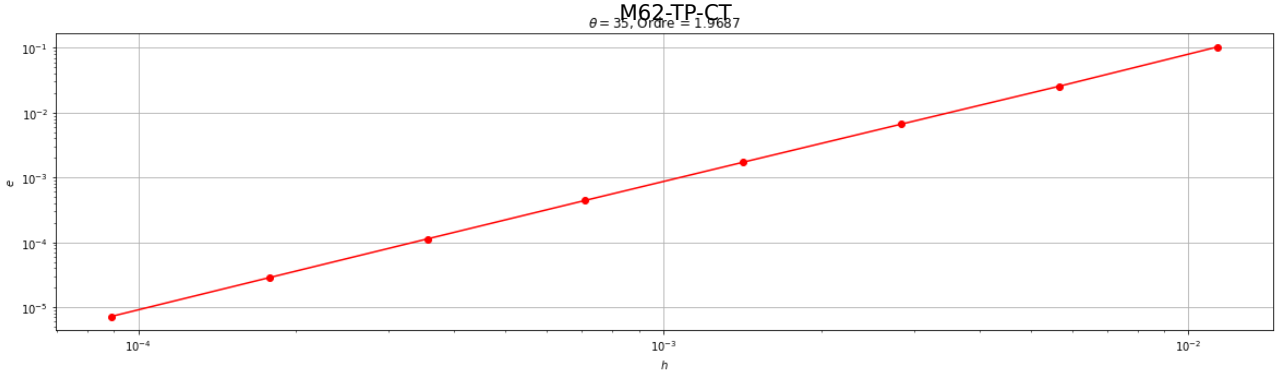

$\theta = 10$, Ordre = 1.9686
e
h

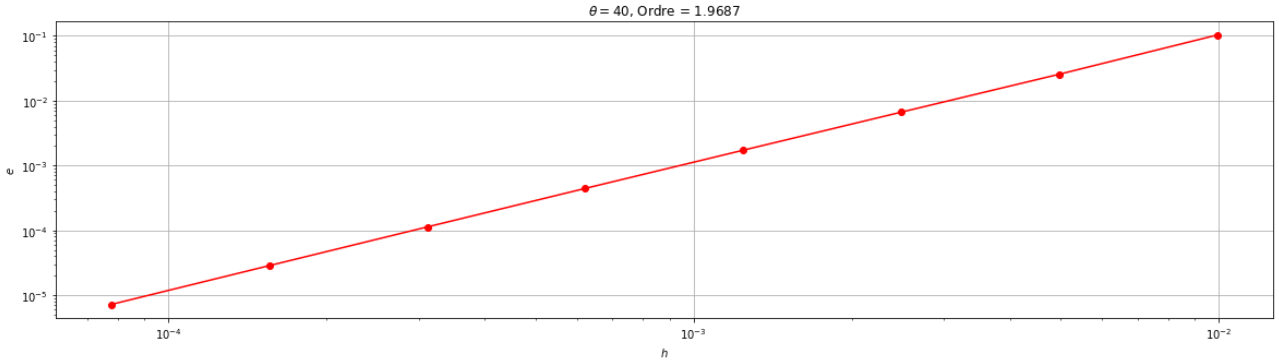

$\theta = 15$, Ordre = 1.9686
e
h

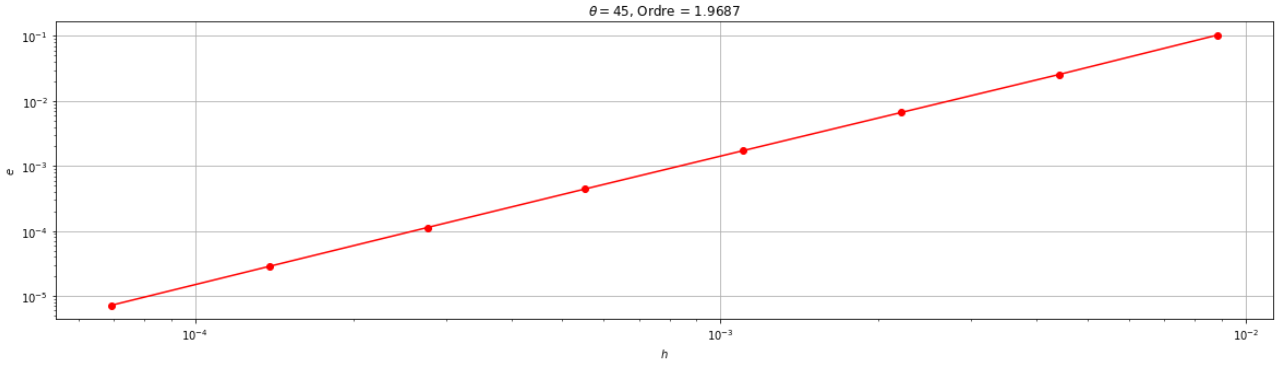

$\theta = 20$, Ordre = 1.9687
e
h

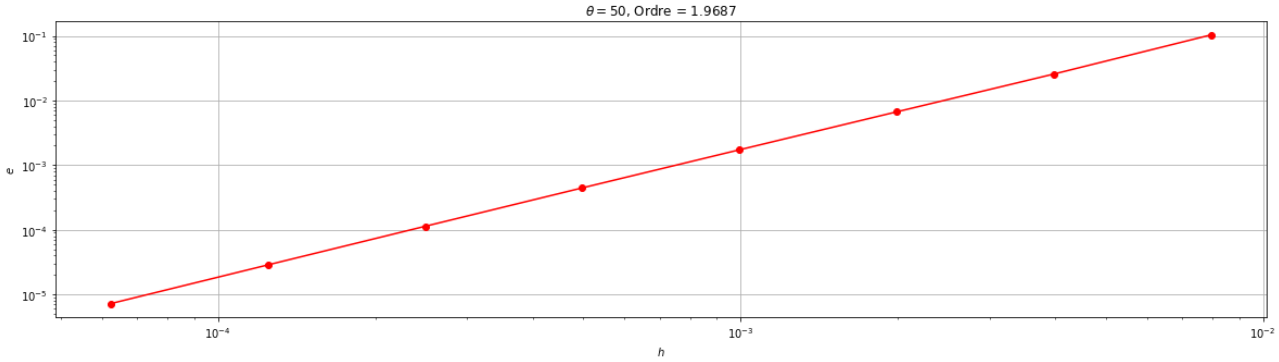

$\theta = 25$, Ordre = 1.9687
e
h


$\theta = 30$, Ordre = 1.9687
e
h




# Schéma Runge-Kutta

Soit le schéma de Runge-Kutta dont la *matrice de Butcher* est

$$\begin{array}{c|cc} 1 & 1 & 0 \\ \beta & \beta & 0 \\ \hline & (1-\gamma) & \gamma \end{array}$$

Chaque sujet correspond à un couple $(\beta, \gamma)$ choisi parmi les suivants:

In [8]:
```
from fractions import Fraction
alpha=1
GAMMA=[Fraction(13,24),Fraction(14,24),Fraction(15,24),Fraction(16,24),Fraction(17,24),
       Fraction(18,24),Fraction(19,24),Fraction(20,24),Fraction(21,24),Fraction(22,24),Fraction
(23,24)]
for gamma in GAMMA:
    beta=Fraction(Fraction(1,2)+alpha*(gamma-1),gamma)
    print("\t beta=",beta,"\t gamma=",gamma)
```

```
beta= 1/13      gamma= 13/24
beta= 1/7       gamma= 7/12
beta= 1/5       gamma= 5/8
beta= 1/4       gamma= 2/3
beta= 5/17      gamma= 17/24
beta= 1/3       gamma= 3/4
beta= 7/19      gamma= 19/24
beta= 2/5       gamma= 5/6
beta= 3/7       gamma= 7/8
beta= 5/11      gamma= 11/12
beta= 11/23     gamma= 23/24
```

**Q8** Écrire le schéma sous la forme d'une suite définie par récurrence.

**Correction Q8**

Considérons le problème de Cauchy

trouver une fonction $y$: $I \subset \mathbb{R} \to \mathbb{R}$ définie sur un intervalle $I = [t_0, T]$ telle que
$$\begin{cases} y'(t) = \varphi(t, y(t)), \quad \forall t \in I = [t_0, T], \\ y(t_0) = y_0, \end{cases}$$
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 $t_n \overset{\text{déf.}}{=} t_0 + nh$ avec $n = 0, 1, 2, \ldots, N$ une suite de $N+1$ nœuds de $I$ induisant une discrétisation de $I$ en $N$ 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 $N+1$ valeurs $\{t_0, t_1 = t_0 + h, \ldots, t_N = T\}$ représente les points de la discrétisation.
- L'ensemble de $N+1$ valeurs $\{y_0, y_1, \ldots, y_N\}$ représente la solution exacte.
- L'ensemble de $N+1$ valeurs $\{u_0 = y_0, u_1, \ldots, u_N\}$ représente la solution numérique. Cette solution approchée sera obtenue en construisant une suite récurrente.

Le schéma qu'on va construire permet de calculer implicitement $u_{n+1}$ à partir de $u_n$ par la formule de récurrence
$$\begin{cases} u_0 = y_0 \\ K_1 = \varphi\left(t_n + h, u_n + hK_1\right) \\ K_2 = \varphi\left(t_n + \beta h, u_n + \beta h K_1\right) \\ u_{n+1} = u_n + h\left((1-\gamma)K_1 + \gamma K_2\right) \quad n = 0, 1, \ldots N-1 \end{cases}$$

**Q9** Étudier théoriquement l'ordre du schéma.

**Correction Q9**

- C'est un schéma semi-implicite à deux étages: l'ordre est au plus 4.
- $c_1 = a_{11} + a_{12}$, $c_2 = a_{21} + a_{22}$ et $1 = b_1 + b_2 \implies$ il est consistante
- $c_1 b_1 + c_2 b_2 = \frac{1}{2} \implies$ il est d'ordre $2$
- $b_1 c_1^2 + b_2 c_2^2 = \frac{1}{2\gamma}$ et $b_1 a_{11} c_1 + b_2 a_{12} c_2 = \frac{1}{2} \implies$ il n'est pas d'ordre $3$

On peut vérifier ces calculs avec `sympy`:

In [9]:
```
import sympy as sym
sym.init_printing()

from IPython.display import display, Math

sym.var('gamma')
beta=(gamma-sym.S(1)/2)/gamma
c=[1,beta]
b=[1-gamma,gamma]
A=[[1,0],[beta,0]]
s=len(c)

display(Math(r'\sum_{j=1}^s b_j='+sym.latex(sum(b))     ))
for i in range(s):
    display(Math(r'i={}'+str(i)+'\quad \sum_{j=1}^s a_{ij}-c_i='+sym.latex(sum(A[i])-c[i])
))

display(Math(r'\sum_{j=1}^s b_j c_j='+sym.latex(sum([b[i]*c[i] for i in range(s)]))     ))

display(Math(r'\sum_{j=1}^s b_j c_j^2='+sym.latex(sum([b[i]*c[i]**2 for i in range(s)]).simplif
y())     ))
display(Math(r'\sum_{i,j=1}^s b_i a_{ij} c_j='+sym.latex(sum([b[i]*A[i][j]*c[j] for i in range(
s) for j in range(s)]))     ))
```

$$\sum_{j=1}^{s} b_j = 1$$

$$i = 0 \quad \sum_{j=1}^{s} a_{ij} - c_i = 0$$

$$i = 1 \quad \sum_{j=1}^{s} a_{ij} - c_i = 0$$

$$\sum_{j=1}^{s} b_j c_j = \frac{1}{2}$$

$$\sum_{j=1}^{s} b_j c_j^2 = \frac{1}{4\gamma}$$

$$\sum_{i,j=1}^{s} b_i a_{ij} c_j = \frac{1}{2}$$

**Q10** Implémenter le schéma de la question précédente et le tester avec le problème de Cauchy.
On choisira $N = 1 + 2\vartheta$ points de discrétisation et on affichera la solution approchée et la solution exacte sur le même repère.

Attention: le schéma est semi-implicite, la fonction `fsolve` du module `scipy.optimize` est votre amie...

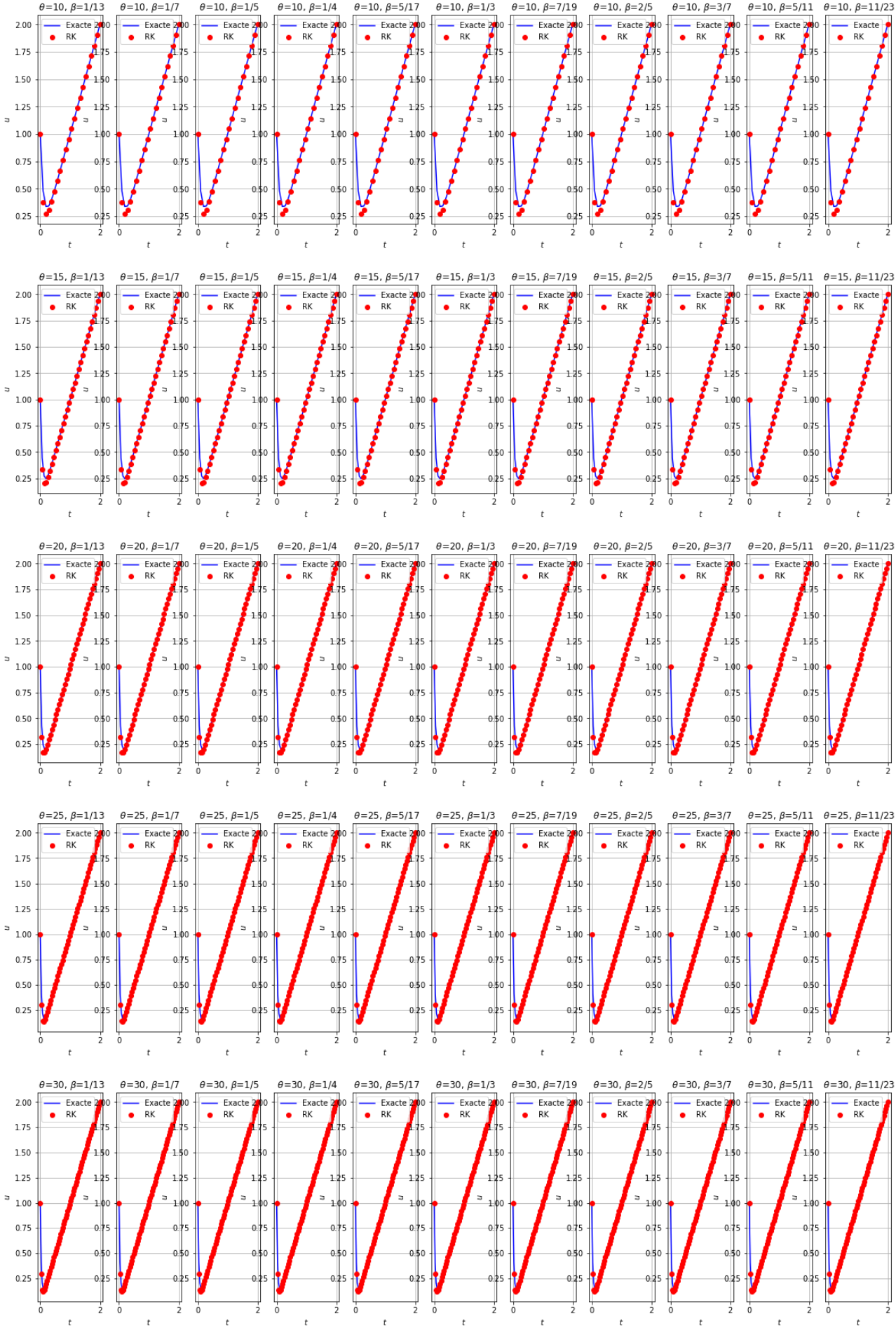
**Correction Q10**

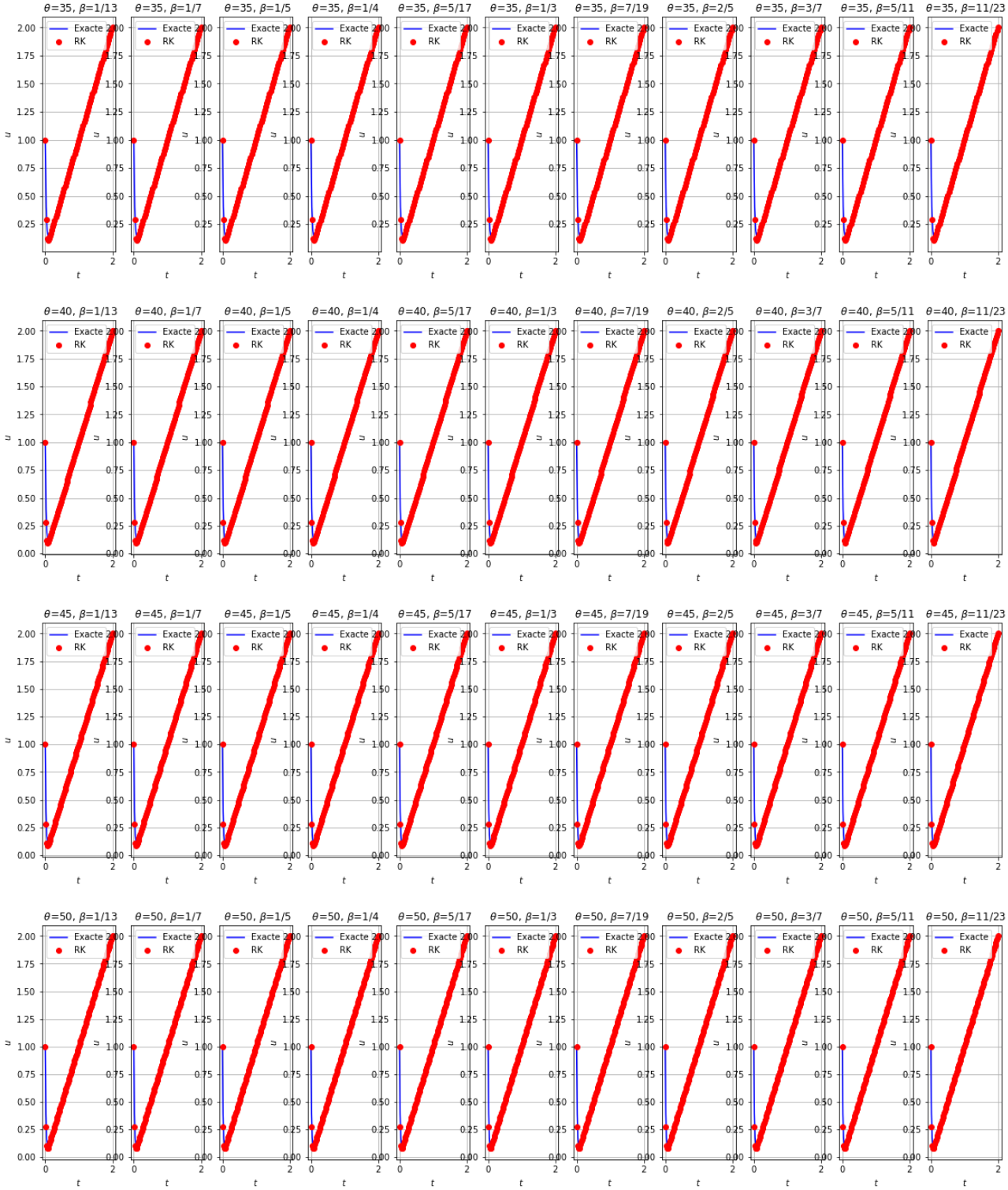
Dans chaque point $t_i$, il faut approcher $(K_1)_i$ en resolvant une équation. Si on utilise la fonction `fsolve` du module `scipy.optimize`, il faut initialiser `fsolve` avec une approximation de $(K_1)_i$. On choisira donc $\varphi(t_i, u_i)$:

In [10]:
```
from scipy.optimize import fsolve

def RK(phi,tt):
    uu = [y0]
    for i in range(len(tt)-1):
        k1 = fsolve( lambda x : x - phi( tt[i]+alpha*h, uu[i]+alpha*h*x ) , phi(tt[i],uu[i]) )[
0]
        k2 = phi( tt[i]+beta*h , uu[i]+beta*h*k1 )
        uu.append( uu[i] + h*((1-gamma)*k1+gamma*k2 ))
    return uu
```

Comparons solution exacte et solution approchée (pour trouver votre sujet vous devez chercher les valeurs de $\vartheta$ et $\beta$ correspondants):

In [11]:
```
from matplotlib.pylab import *

t0, y0, tfinal = 0, 1, 2

sol_exacte = lambda t : t+exp(-theta*t)
phi = lambda t,y : 1+theta*(t-y)


for theta in THETA:
    N = 1+2*theta
    tt = linspace(t0,tfinal,N+1)
    h = tt[1]-tt[0]
    yy = [sol_exacte(t) for t in tt]
    figure(theta,figsize=(20,5))
    g=len(GAMMA)
    i=0
    for gamma in GAMMA:
        beta=Fraction(Fraction(1,2)+alpha*(gamma-1),gamma)
        uu = RK(phi,tt)
        i+=1
        subplot(1,g,i)
        plot(tt,yy,'b-',label=("Exacte"))
        plot(tt,uu,'ro',label=("RK"))
        title(r' $\theta$={}, $\beta$={}'.format(theta,beta))
        xlabel('$t$')
        ylabel('$u$')
        legend()
        grid(True)
```


θ=10, β=1/13
θ=10, β=1/7
θ=10, β=1/5
θ=10, β=1/4
θ=10, β=5/17
θ=10, β=1/3
θ=10, β=7/19
θ=10, β=2/5
θ=10, β=3/7
θ=10, β=5/11
θ=10, β=11/23
θ=15, β=1/13
θ=15, β=1/7
θ=15, β=1/5
θ=15, β=1/4
θ=15, β=5/17
θ=15, β=1/3
θ=15, β=7/19
θ=15, β=2/5
θ=15, β=3/7
θ=15, β=5/11
θ=15, β=11/23
θ=20, β=1/13
θ=20, β=1/7
θ=20, β=1/5
θ=20, β=1/4
θ=20, β=5/17
θ=20, β=1/3
θ=20, β=7/19
θ=20, β=2/5
θ=20, β=3/7
θ=20, β=5/11
θ=20, β=11/23
θ=25, β=1/13
θ=25, β=1/7
θ=25, β=1/5
θ=25, β=1/4
θ=25, β=5/17
θ=25, β=1/3
θ=25, β=7/19
θ=25, β=2/5
θ=25, β=3/7
θ=25, β=5/11
θ=25, β=11/23
θ=30, β=1/13
θ=30, β=1/7
θ=30, β=1/5
θ=30, β=1/4
θ=30, β=5/17
θ=30, β=1/3
θ=30, β=7/19
θ=30, β=2/5
θ=30, β=3/7
θ=30, β=5/11
θ=30, β=11/23
Exacte
RK
u
t


θ=35, β=1/13
θ=35, β=1/7
θ=35, β=1/5
θ=35, β=1/4
θ=35, β=5/17
θ=35, β=1/3
θ=35, β=7/19
θ=35, β=2/5
θ=35, β=3/7
θ=35, β=5/11
θ=35, β=11/23
θ=40, β=1/13
θ=40, β=1/7
θ=40, β=1/5
θ=40, β=1/4
θ=40, β=5/17
θ=40, β=1/3
θ=40, β=7/19
θ=40, β=2/5
θ=40, β=3/7
θ=40, β=5/11
θ=40, β=11/23
θ=45, β=1/13
θ=45, β=1/7
θ=45, β=1/5
θ=45, β=1/4
θ=45, β=5/17
θ=45, β=1/3
θ=45, β=7/19
θ=45, β=2/5
θ=45, β=3/7
θ=45, β=5/11
θ=45, β=11/23
θ=50, β=1/13
θ=50, β=1/7
θ=50, β=1/5
θ=50, β=1/4
θ=50, β=5/17
θ=50, β=1/3
θ=50, β=7/19
θ=50, β=2/5
θ=50, β=3/7
θ=50, β=5/11
θ=50, β=11/23
Exacte
RK
u
t