Équations différentielles

Calcul formel – TP 9

1. Une équation avec condition initiale

1. Pour obtenir une solution exacte, il suffit d'utiliser la commande `desolve` comme suit :

```
var('x')
y = function('y')(x)
yy = diff(y,x)
equadiff = yy == 2*y + exp(x)

desolve(equadiff, y, ivar=x)
```

Sage retourne :

```
(_C - e^(-x))*e^(2*x)
```

La famille des solutions est donc $-e^x + Ce^{2x}$, pour les constantes $C \in \mathbb{R}$.

2. **Énigme.** On commence par trouver l'unique solution vérifiant $y(\ln(7)) = 123456$:

```
x1 = ln(7)
y1 = 123456
sol=desolve(equadiff, y, ics=[x1,y1], ivar=x)
sol
```

ce qui donne la solution

$$y(x) = \frac{123463}{49} e^{2x} - e^x.$$

Puis on l'évalue en $x = 0$:

```
x0 = 0
print sol(x=x0).full_simplify()
```

Sage renvoie $\frac{123414}{49}$.

Réponse attendue : 12341449.

2. Le poulet de Newton

1. On commence par définir les variables, la fonction température et sa dérivée, les constantes correspondant aux conditions initiales, l'équation différentielle ; puis on résout cette dernière :

```
var('c')   # Constante de l'equation de Newton
var('t')   # Temps
T = function('T')(t)  # T(t) : temperature du poulet
TT = diff(T,t)  # sa derivee T'(t)
T0 = 5     # Temperature initiale du poulet
F0 = 250   # Temperature du four
equadiff = TT == c*(F0-T)
sol = desolve(equadiff, T, ivar=t, ics=[0,T0])
sol
```

   Sage renvoie `5*(50*e^(c*t) - 49)*e^(-c*t)`. Il ne reste plus qu'à utiliser la donnée supplémentaire sur le temps de cuisson pour déterminer c :

```
Tcuit = 200
tcuisson = 100
equac = sol(t=tcuisson) == Tcuit
find_root(equac,0,1)
```

   On prend l'approximation numérique :

$$c = 0.0158923520512\ldots$$

2. **Énigme.** On procède exactement de la même manière, la seule différence étant que la température du four dépend maintenant du temps :

```
var('t')   # Temps
T = function('T')(t)  # T(t) : temperature du poulet
TT = diff(T,t)  # sa derivee T'(t)
F = 250 - 230*exp(-1/5*t) # Temperature du four
T0 = 5     # Temperature initiale du poulet
c = 0.0158923520512
equadiff = TT == c*(F-T)
sol = desolve(equadiff, T, ivar=t, ics=[0,T0])
sol
```

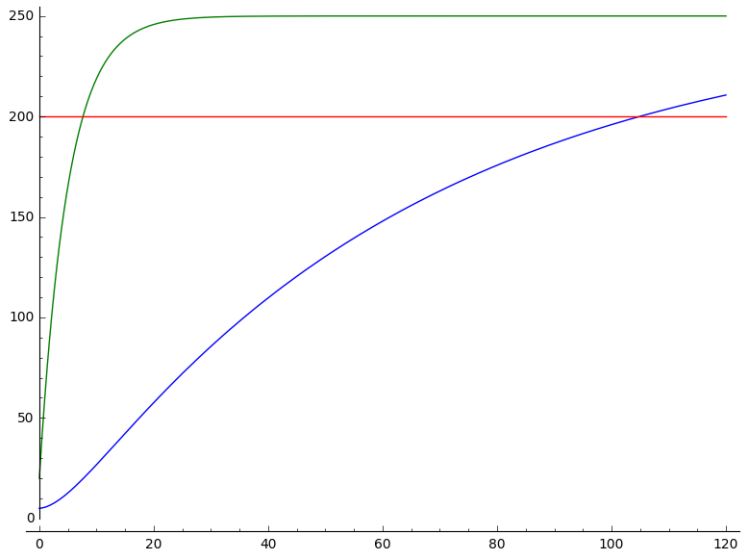
   On obtient une solution assez compliquée. L'inconnue du problème est maintenant le temps t, l'équation se résout tout aussi facilement :

```
Tcuit = 200
equat = sol == Tcuit
find_root(equat,1,200)
```

   Sage trouve retourne $t = 104.9029783543448\ldots$

   Réponse attendue : 105.

Sur la figure, en vert la température du four, en bleu celle du poulet, qui atteint la température souhaitée (en rouge) pour t proche de 105 minutes.



3. Balistique

1. **Sans frottements.**

   On commence par définir comme variables l'angle de tir, la vitesse initiale et la constante de gravitation universelle, puis on en déduit les composantes de la vitesse initiale.

```
# Angle de tir
var('alpha')
# Vitesse initiale
var('v0')
# Constante de gravitation
var('g')
# Composantes de la vitesse initiale
vx0 = v0*cos(alpha)
vz0 = v0*sin(alpha)
```

   On définit ensuite la variable t temporelle, la fonction $x(t)$ et l'équation différentielle qu'elle vérifie, qu'on résout, en tenant compte des conditions initiales, à l'aide de la commande `desolve` :

```
# Equation differentielle en x
t = var('t')
x = function('x')(t)
xx = diff(x,t)
xxx = diff(x,t,2)
equadiffx = xxx == 0
solx = desolve(equadiffx, x, ivar=t, ics=[0,0,vx0])
solx
```

   On obtient `t*v0*cos(alpha)`. On fait ensuite de même avec la seconde coordonnée z :

```
# Equation differentielle en z
z = function('z')(t)
zz = diff(z,t)
zzz = diff(z,t,2)
equadiffz = zzz == -g
```

```
solz = desolve(equadiffz, z, ivar=t, ics=[0,0,vz0])
solz
```

ce qui donne : `-1/2*g*t^2 + t*v0*sin(alpha)`. On peut alors tracer les courbes paramétrées

$$\big(x(t),z(t)\big)=\left(v_0\cos(\alpha)t, v_0\sin(\alpha)t-\frac{1}{2}gt^2\right)$$

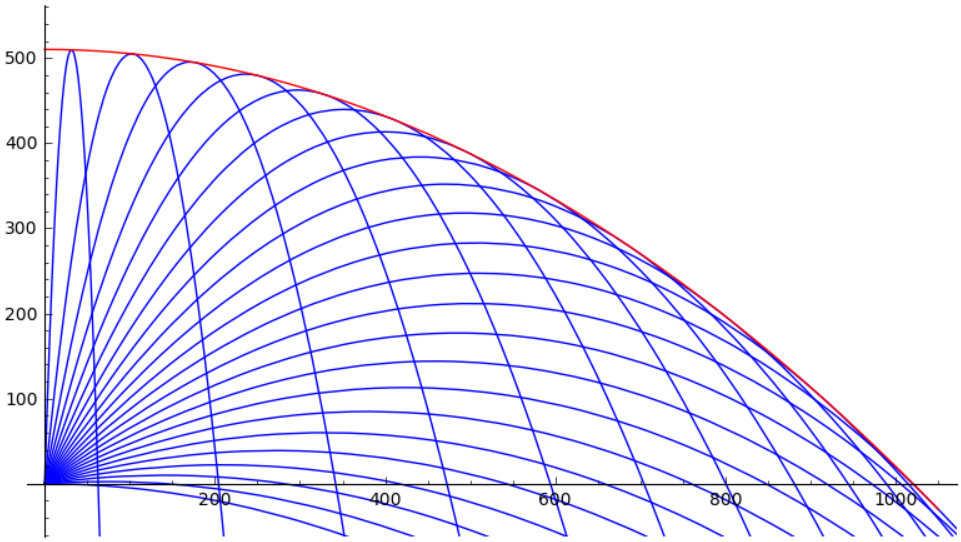
pour différentes valeurs de l'angle α grâce à une boucle et à la commande `parametric_plot` :

```
# Dessin des solutions pour differentes valeur d'angle
G = Graphics()
myg = 9.8
myv0 = 100
for myalpha in srange(0,pi/2,0.07):
    plotsolx = solx(g=myg,v0=myv0)
    plotsolz = solz(g=myg,v0=myv0)
    G = G + parametric_plot((plotsolx(alpha=myalpha),plotsolz(alpha=myalpha)),
                            (t,0,50))
G.show(aspect_ratio=1, xmin=0,xmax=1050,ymin=-50,ymax=550)
```

2. On ajoute l'équation de la parabole de sécurité, puis on trace le tout :

```
var('u')
h = v0^2/(2*g)
ploth = h(g=myg,v0=myv0)
G = G + parametric_plot((u,ploth-u^2/(4*ploth)),(u,0,1050),color='red')
G.show(aspect_ratio=1, xmin=0,xmax=1050,ymin=-50,ymax=550)
```

Voici le résultat :



3. **Avec frottements.**
La démarche est la même qu'à la première question. On définit les variables :

```
var('v0')
vx0 = v0*cos(alpha)
vz0 = v0*sin(alpha)
var('alpha')
var('g')
var('f')
assume(f>0)
```

On définit l'équation différentielle en $x(t)$, puis on la résout :

```
t = var('t')
x = function('x')(t)
xx = diff(x,t)
xxx = diff(x,t,2)
equadiffx = xxx == -f*xx
solx = desolve(equadiffx, x, ivar=t, ics=[0,0,vx0])
solx
```

On obtient : `-v0*cos(alpha)*e^(-f*t)/f + v0*cos(alpha)/f`. Puis on passe à l'équation différentielle en $z(t)$:

```
z = function('z')(t)
zz = diff(z,t)
zzz = diff(z,t,2)
equadiffz = zzz == -g-f*zz
solz = desolve(equadiffz, z, ivar=t, ics=[0,0,vz0])
solz
```

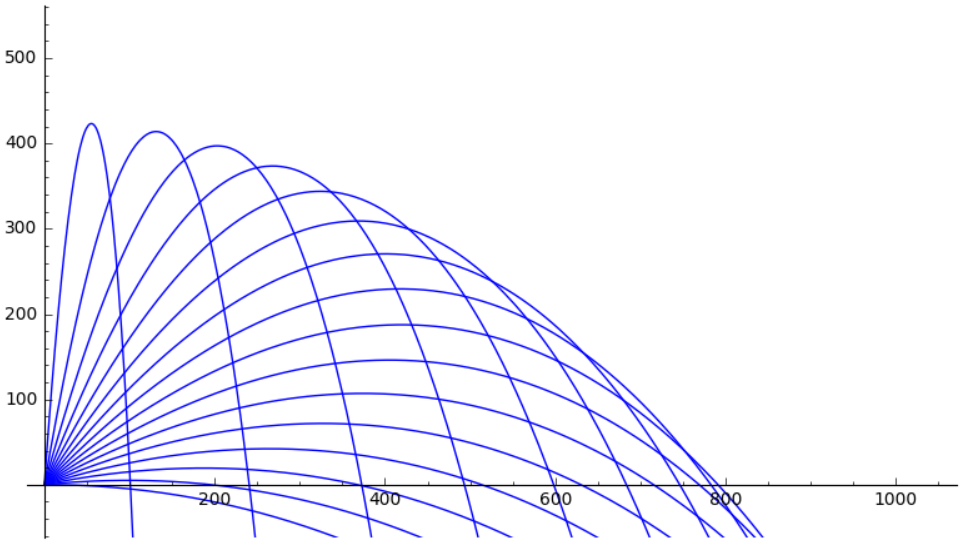
Ce qui donne `v0*sin(alpha)/f - (f*v0*sin(alpha)+g)*e^(-f*t)/f^2 - (f*g*t-g)/f^2`. L'équation paramétrique est donc :

$$\big(x(t),z(t)\big)=\left(\frac{v_0}{f}\cos(\alpha)-\frac{v_0}{f}\cos(\alpha)e^{-f\,t},\frac{v_0}{f}\sin(\alpha)+\frac{g}{f^2}-\frac{g}{f}t-\frac{1}{f^2}(f\,v_0\sin(\alpha)+g)e^{-f\,t}\right).$$

Enfin, on définit les courbes pour différentes valeurs d'angle :

```
G = Graphics()
myg = 9.8
myv0 = 100
myf = 0.03
for myalpha in srange(0,pi/2,0.1):
    plotsolx = solx(g=myg,v0=myv0,f=myf)
    plotsolz = solz(g=myg,v0=myv0,f=myf)
    G = G + parametric_plot((plotsolx(alpha=myalpha),plotsolz(alpha=myalpha)),
                            (t,0,50))
G.show(aspect_ratio=1, xmin=0,xmax=1050,ymin=-50,ymax=550)
```

Sage affiche :

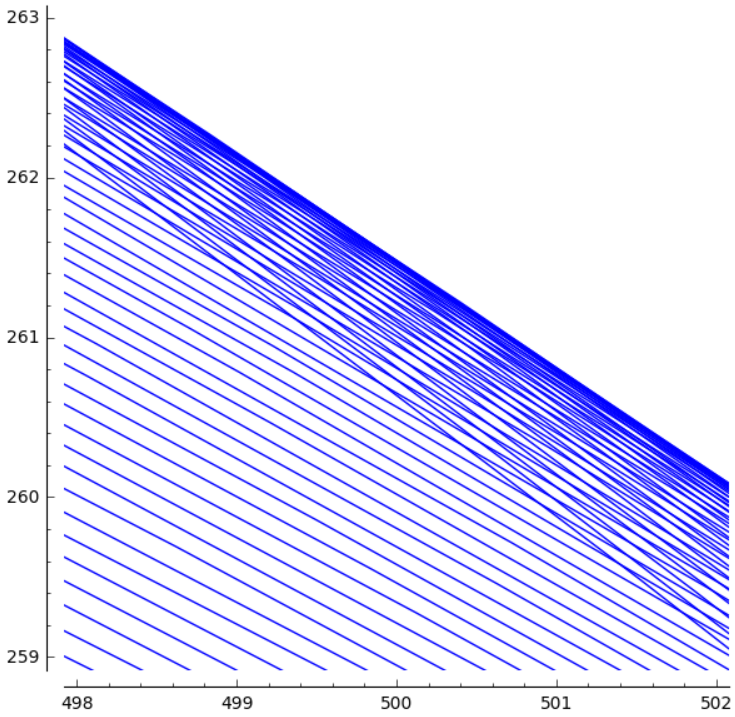


4. **Énigme.** On fait un zoom sur la zone qui nous intéresse. Il faut aussi ajuster la valeur de l'angle α de façon à avoir assez de courbes au voisinage du point d'abscisse 500 :

```
G = Graphics()
myg = 9.8
myv0 = 100
myf = 0.03

for myalpha in srange(0.9,1.0,0.001):
    plotsolx = solx(g=myg,v0=myv0,f=myf)
    plotsolz = solz(g=myg,v0=myv0,f=myf)
    G = G + parametric_plot((plotsolx(alpha=myalpha),plotsolz(alpha=myalpha)),
                            (t,0,50))
G.show(aspect_ratio=1, xmin=498,xmax=502,ymin=259,ymax=262)
```

On obtient le graphe suivant :



On constate que l'altitude critique est entre 261 et 262.

Réponse attendue : 262.

4. Méthode d'Euler

Partie I. Méthode d'Euler classique.

1. On définit, comme nous en avons l'habitude, une fonction contenant une boucle.

```
def euler(f,x0,y0,xmax,h):
    xn = x0
    yn = y0
    courbe = [(x0,y0)]
    while xn < xmax:
        xn, yn = xn+h, yn + h*f(x=xn,y=yn)
```

```
        courbe = courbe + [(xn,yn)]
    return courbe
```

Notez cependant le procédé de double affectation !

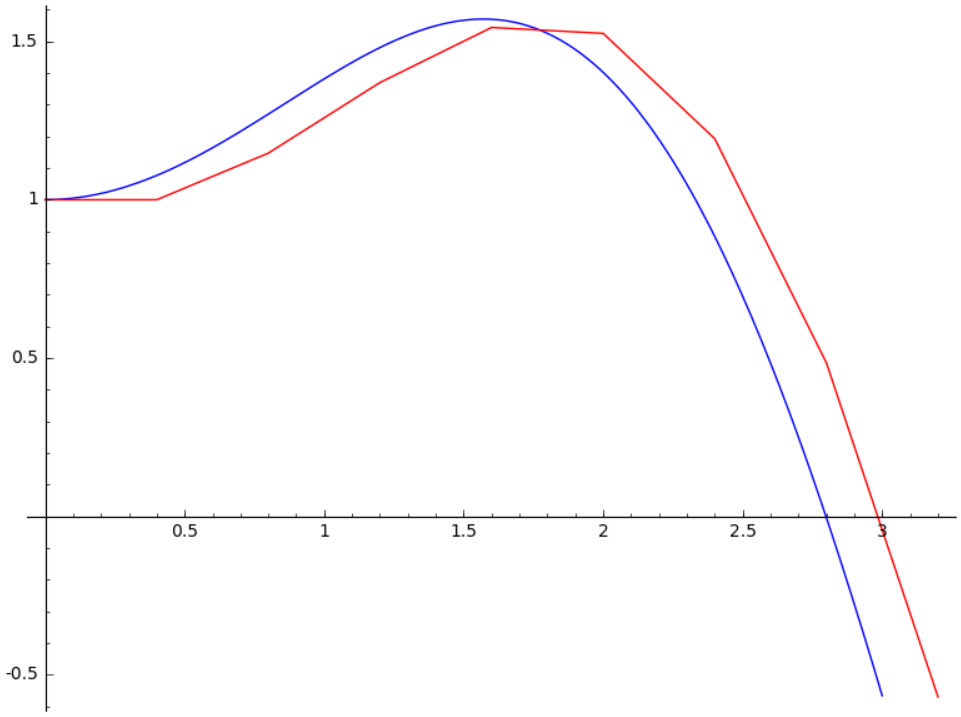
2. On commence par calculer la solution exacte et par définir son graphe :

```
var('x')
y = function('y')(x)
yy = diff(y,x)
f = x*cos(x)
x0 = 0
y0 = 1
sol_sage = desolve(yy == f, y, ics=[x0,y0])
print sol_sage
G = plot(sol_sage,(x,0,3))
```

On obtient : `x*sin(x) + cos(x)`. On passe ensuite à la solution approchée :

```
xmax = 3
h = 0.4
courbe = euler(f,x0,y0,xmax,h)
G = G + line(courbe, color='red')
G.show()
```

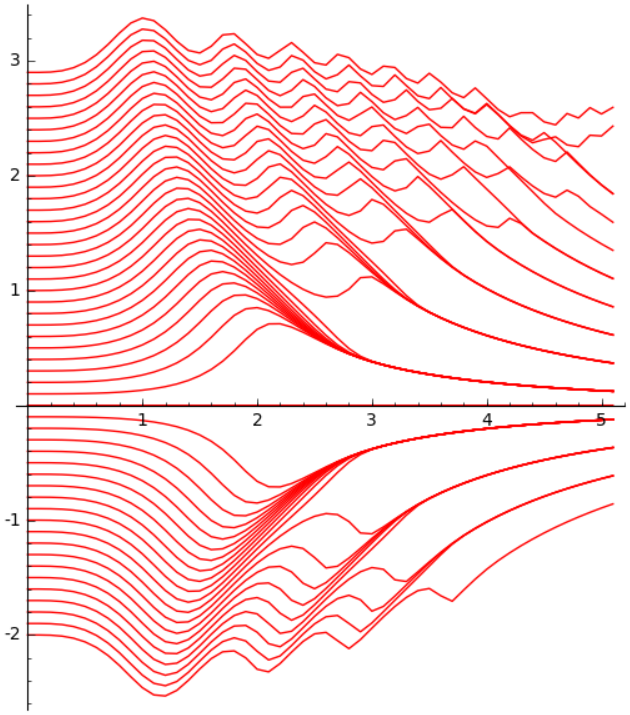
Sage affiche en bleu la solution exacte et en rouge la solution approchée :



3. On utilise une boucle pour faire varier la condition initiale :

```
var('y')
g = sin(x^2*y)
x0_g = 0
xmax_g = 5
h_g = 0.1
```

```
G_g = Graphics()
for y0_g in srange(-2,3,0.1):
    courbe = euler(g,x0_g,y0_g,xmax_g,h_g)
    G_g = G_g + line(courbe, color='red')
G_g.show(aspect_ratio=1)
```



Partie II. Méthode d'Euler améliorée.

1. On définit une fonction qui améliore la méthode précédente

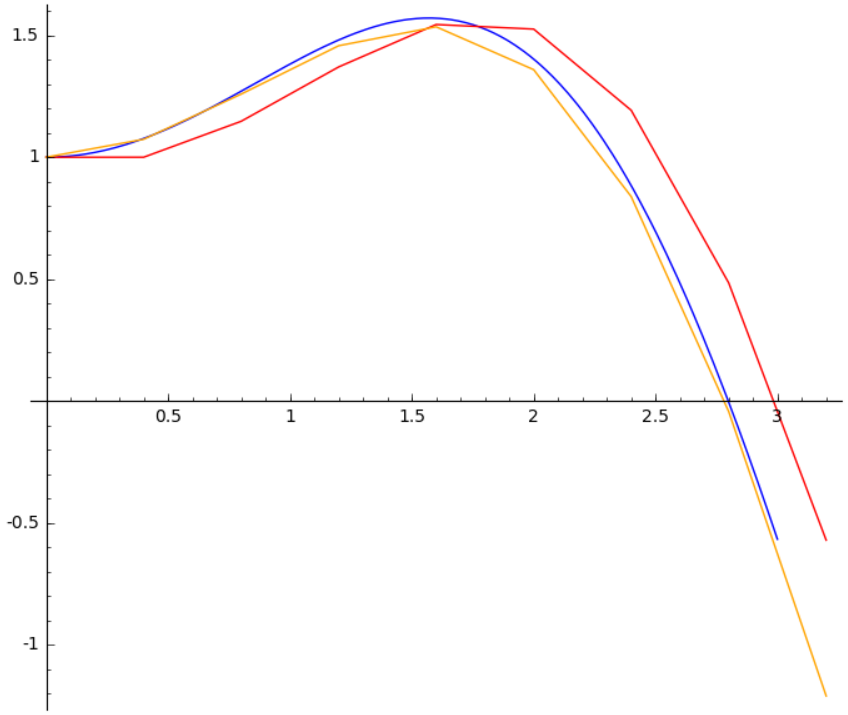
```
var('x,y')

def euler_ameliore(f,x0,y0,xmax,h):
    xn = x0
    yn = y0
    courbe = [(x0,y0)]
    while xn < xmax:
        xn, yn = xn+h, yn+1/2*h*(f(x=xn,y=yn)+f(x=xn+h,y=yn+h*f(x=xn,y=yn)))
        courbe = courbe + [(xn,yn)]
    return courbe
```

puis, afin de comparer les deux méthodes, on ajoute au graphe calculé précédemment la courbe représentative de la nouvelle solution approchée :

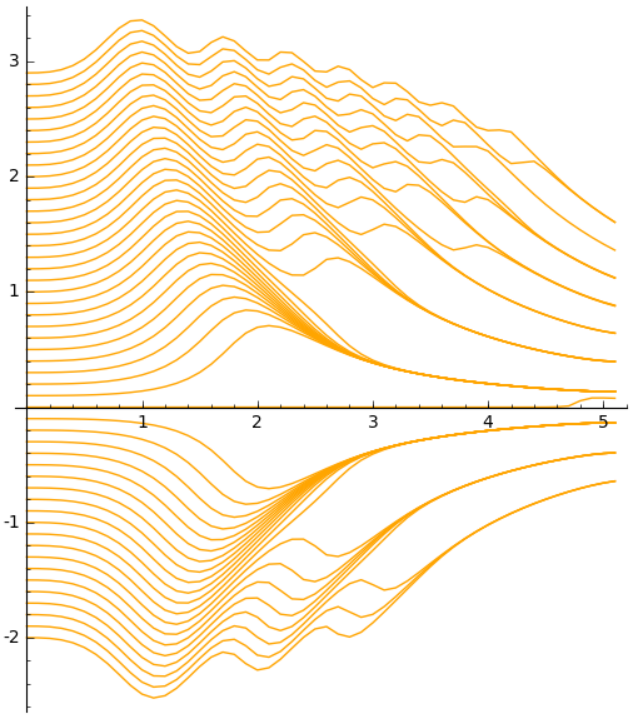
```
courbe = euler_ameliore(f,x0,y0,xmax,h)
G = G + line(courbe, color='orange')
G.show(aspect_ratio=1)
```

En bleu la solution exacte, en rouge la méthode d'Euler classique et en orange la méthode d'Euler améliorée.



Pour l'équation $y' = \sin(x^2 y)$:

```
for y0_g in srange(-2,3,0.1):
    courbe = euler_ameliore(g,x0_g,y0_g,xmax_g,h_g)
    G_g = G_g + line(courbe, color='orange')
G_g.show(aspect_ratio=1)
```



2. **Énigme.** On utilise notre fonction améliorée :

```
var('x,y')

f = y
```

```
x0 = 0
y0 = 1
xmax = 1
h = 1/1000

courbe = euler_ameliore(f,x0,y0,xmax,h)
xn,yn = courbe[-1]
```

L'instruction `courbe[-1]` permet de sélectionner le dernier élément de la liste `courbe`.

L'équation différentielle étant $y' = y$, les solutions sont les $y(x) = ke^x$. L'unique solution vérifiant $y(0) = 1$ est $y(x) = e^x$. Donc $y(1) = e$.

La méthode d'Euler améliorée fournit donc une approximation de $y(1)$ par `yn.n()` (donc de e) :

$$2.71828137575176\ldots$$

alors que $e = 2.71828182845905\ldots$.

Nous obtenons 6 chiffres exacts après la virgule.

Réponse attendue : 6.