# Équations différentielles : Euler vs. Heun vs. RK4 vs. `odeint`

D'après un TP de Stéphane Gonnord

### Buts du TP

- Mettre en place deux méthodes concurrentes à la méthode d'Euler : la méthode de Heun, et la très fameuse méthode de Runge-Kutta d'ordre 4, *aka* « RK4 ».
- Comparer les performances respectives.
- Calculer des approximations de solutions d'équations différentielles dans des situations « réelles ».

Il n'est évidemment pas question de traiter l'ensemble du TP pendant le temps imparti ! Les cinq premiers exercices constituent l'objectif principal.

Créer (au bon endroit) un dossier associé à ce TP. Créer dans ce dossier un sous-dossier `jolis-dessins` dans lequel seront placés les différents (jolis) pdf produits pendant ce TP.

Lancer Spyder ou Pyzo, sauvegarder immédiatement au bon endroit le fichier `TP10.py`.

## 1 Contexte mathématique

On s'intéresse à des approximations de solutions d'équations différentielles de la forme

$$y'(t) = F\big(t, y(t)\big)$$

sur un segment $[a,b]$, avec $y(a)$ donné, et $F$ une application de $\mathbb{R}\times\mathbb{R}$ dans $\mathbb{R}$. Par exemple : « J'ai dans ma poche une fonction qui vaut 1 en 0, et vérifie $y'(t) = 2y(t)$ pour tout $t \in [0,1]$ ; que vaut $y(1)$ ? » Dans cet exemple, $F(t,z) = 2z$. Si cet exemple peut sembler un peu stupide [1]... il ne l'est pas tant que ça : le fait de connaître la valeur exacte de $y(1)$ va permettre d'évaluer la *qualité de l'approximation* calculée.

Si on prend l'exemple $y'(t) = te^{t^2 y(t)}$, on a cette fois $F(t,z) = te^{t^2 z}$, et ici, on ne sait pas donner d'expression analytique des solutions de l'équation différentielle, d'où l'intérêt d'une méthode d'approximation.

On peut aussi imaginer des équations d'ordre un... mais vectorielles :

$$\begin{cases} x'(t) &= (1-y(t))x(t) \\ y'(t) &= (x(t)-1)y(t) \end{cases}$$

Ici, en prenant $F(t,(a,b)) = ((1-b)a,(a-1)b)$ (donc $F$ va de $\mathbb{R}\times\mathbb{R}^2$ dans $\mathbb{R}^2$), on a bien une équation de la forme $Z'(t) = F(t,Z(t))$, pour peu qu'on pose $Z(t) = (x(t),y(t))$.

Enfin, les équations d'ordre deux se ramènent à l'ordre 1 par « vectorialisation » : l'équation du pendule $y''(t) = -\sin\big(y(t)\big)$, ou encore $\begin{pmatrix} y \\ y' \end{pmatrix}' = \begin{pmatrix} y' \\ -\sin y \end{pmatrix}$ est par exemple équivalente, en posant $Y(t) = \big(y(t), y'(t)\big)$, à $Y'(t) = F(t,Y(t))$, avec $F(t,(u,v)) = (v,-\sin u)$.

Dans la suite, on va donc se donner une fonction $F$ (de $\mathbb{R}\times\mathbb{R}$ dans $\mathbb{R}$ pour les équations scalaires d'ordre un, mais potentiellement de $\mathbb{R}\times\mathbb{R}^2$ dans $\mathbb{R}^2$ pour traiter les cas d'ordre deux/vectoriels). On va également se donner deux réels $a<b$, et une condition initiale : on suppose que $y(a)$ est connu, et on souhaite calculer une approximation de $y(b)$, en faisant des « petits pas », c'est-à-dire en calculant des approximations des $y(t_n)$, avec $a = t_0 < t_1 < \cdots < t_n = b$.

## 2 Les quatre candidats

Il s'agit à chaque fois de calculer les approximations $y_k$ de la solution de l'équation $y'(t) = F\big(t,y(t)\big)$ (avec condition initiale) aux différents temps $t_k = a + k\frac{b-a}{n}$. En notant $h = t_{k+1} - t_k = \frac{b-a}{n}$ le pas :

- Dans la méthode d'Euler (cf. un TP précédent), on exploite l'idée basique

$$F(t_k, y(t_k)) = y'(t_k) \simeq \frac{y(t_{k+1}) - y(t_k)}{t_{k+1} - t_k}$$

en posant $y_0 = y(a)$ et pour tout $k<n$ :

$$y_{k+1} = y_k + hF(t_k, y_k).$$

1. On connaît $y(t)$ : c'est $e^{2t}$ !



```
def euler(F, a, y0, b, n):
    les_yk = [y0]          # la liste des valeurs calculées
    t = a                  # le temps du dernier calcul
    h = float(b-a) / n  # le pas
    dernier = y0           # la dernière valeur calculée
    for i in range(n):
        suivant = dernier + h*F(t, dernier) # le nouveau terme
        les_yk.append(suivant)  # on le place à la fin des valeurs calculées
        t = t+h                 # le nouveau temps
        dernier = suivant       # et on met à jour le dernier terme calculé
    return les_yk  # c'est fini
```

- Pour la méthode de Heun, on prend :

$$y_{k+1} = y_k + \frac{h}{2}\big(F\big(t_k,y_k\big) + F\big(t_{k+1}, y_k + hF(t_k,y_k)\big)\big)$$

(on fait une moyenne entre la dérivée au temps $t_k$, et celle au temps $t_{k+1}$ en la valeur approchée calculée par Euler).

- Dans la méthode de Runge-Kutta d'ordre 4, on réinjecte successivement des premières approximations dans de nouvelles [2]. On calcule donc successivement :

$$\alpha_k = y_k + \frac{h}{2}F(t_k,y_k), \quad \beta_k = y_k + \frac{h}{2}F(t_k+h/2,\alpha_k), \quad \gamma_k = y_k + hF(t_k+h/2,\beta_k)$$

et enfin :

$$y_{k+1} = y_k + \frac{h}{6}\big(F(t_k,y_k) + 2F(t_k+h/2,\alpha_k) + 2F(t_k+h/2,\beta_k) + F(t_{k+1},\gamma_k)\big).$$

- Le dernier candidat est la fonction `odeint` de la librairie `scipy.integrate`.

  Tout d'abord, attention à l'ordre des arguments : on parle ici de l'équation $y'(t) = F(y(t),t)$ (ordre inversé par rapport à l'usage « commun » du matheux, exposé depuis le début de ce TP). On donne donc comme arguments à `odeint` cette fonction $F$, la valeur initiale $y(a)$, et une liste (ou un tableau `numpy`) de temps en lesquels on veut des approximations de la solution. Par exemple, pour avoir une approximation de la solution de $y' = y$ avec la condition initiale $y(0) = 1$ aux temps $0$, $\frac{1}{2}$ et $1$, on commence par définir $F(z,t) := z$ :

```
>>> def f0(z, t):
        return z
>>> les_y = odeint(f0, 1, [0, 0.5, 1])
...
```

EXERCICE 1 *Programmer la méthode de Heun*

```
def heun(F, a, y0, b, n):
    ...
    return les_yk
```

*Tester sur* $[0,1]$ *avec l'équation* $y' = y$ *et* $y(0) = 1$ *pour différentes valeurs de* $n$.

```
>>> def f1(t, z): # attention à l'ordre des arguments !
        return z
>>> heun(f1, 0, 1, 1, 2)
[1, 1.625, 2.640625]
```

EXERCICE 2 *Même chose avec RK4 !*

```
>>> RK4(f1, 0, 1, 1, 2)
[1, 1.6484375, 2.71734619140625]
```

EXERCICE 3 *Comparer les quatre méthodes sur ce même problème de Cauchy pour* $n = 10$, $n = 100$ *et* $n = 1000$, *en regardant l'approximation finale de* $y(1) \simeq \mathrm{e}$.

2. $\alpha_k$ est une approximation de $y(t_k + h/2)$, injectée dans une nouvelle approximation $\beta_k$ de $y(t_k+h/2)$, elle-même utilisée pour calculer une première approximation $\gamma_k$ de $y(t_{k+1})$ ; on utilise ensuite une intégration « à la Simpson ».

```
from math import exp
e = exp(1)

print("exp(1)=%.15f"%e)
for n in [10, 100, 1000]:
    eul = euler(f1, 0, 1, 1, n)[-1]
    heu = heun(f1, 0, 1, 1, n)[-1]
    rk4 = RK4(f1, 0, 1, 1, n)[-1]
    print("Pour n=%i :\n\t\t%.15f\t%.15f\t%.15f"%(n,eul,heu,rk4))
    print("Erreurs absolues : \n\t\t%.15f\t%.15f\t%.15f\n"\
          %(abs(eul-e), abs(heu-e), abs(rk4-e)))
------------------
exp(1)=2.718281828459045
Pour n=10 :
2.593742460100000 2.714080846608224 2.718279744135166
Erreurs absolues :
0.124539368359045 0.004200981850821 0.000002084323879

Pour n=100 :
2.704813829421526 2.718236862559957 2.718281828234404
...
```

On rappelle que pour représenter une solution approchée, on peut exécuter par exemple les lignes suivantes (après avoir importé `matplotlib.pyplot` sous l'alias `plt` par exemple) :

```
t = numpy.linspace(0, 1, 5)
# Les temps t_k. Il en faut le même nombre que les yk, attention...

yeuler = euler(f1, 0, 1, 1, 4)
yheun = heun(f1, 0, 1, 1, 4)
yrk4 = RK4(f1, 0, 1, 1, 4)

yodeint = odeint(f0, 1, t)

plt.plot(t, yeuler)
plt.plot(t, yheun)
plt.plot(t, yrk4)
plt.plot(t, yodeint)

plt.legend(['Euler', 'Heun', 'RK4', 'odeint'], loc = 'upper left')

plt.savefig('comparaison-methodes.pdf')
plt.show()
```

## 3 À l'ordre deux

Normalement, le code écrit précédemment doit fonctionner pour les équations à valeurs vectorielles (pour peu qu'on utilise des `array` de `numpy` pour que les additions se passent... comme des additions, et non des concaténations de listes).

EXERCICE 4 *Adapter, si besoin est, vos programmes pour qu'ils fonctionnent sur l'exemple* $y'' = -\sin(y)$ *:*

```
def f_pendule(t, z):
    y, yp = z # ou (y, yp) = (z[0], z[1]).
    return numpy.array([yp, -numpy.sin(y)])

Y = euler(f_pendule, 0, numpy.array([0, 1.5]), 10, 100)
```

Si tout se passe bien, `Y` doit être la liste des approximations de $(y, y')$ aux temps $t_k = \frac{10k}{100}$ $(0 \leqslant k \leqslant 100)$, avec $y$ la solution de $y'' = -\sin y$, sous les conditions initiales $y(0) = 0$ et $y'(0) = 1.5$. Si c'est bien le cas, on peut représenter la solution approchée en commençant par extraire la première colonne de $Y$, c'est-à-dire le premier élément de chaque ligne. On peut faire quelque chose comme `[ligne[0] for ligne in Y]`, ou bien utiliser le slicing des `array` de `numpy` :



```
Yarray = numpy.array(Y)
t = numpy.linspace(0, 10, 101)

plt.plot(t, Yarray[:, 0])  # sur chaque ligne de Y, on prend la première composante
```

EXERCICE 5 *Représenter les solutions de l'équation du pendule non amorti sur* $[0,10]$ *pour* $y(0) = 0$, $y'(0) = 1.5$, *et* $n \in \{20, 100, 1000\}$. *Pour chaque valeur de n, on tracera sur un même graphique le résultat des trois méthodes numériques.*

EXERCICE 6 *Avec les conditions initiales* $y(0) = 0$ *et* $y'(0) = 2$, *on peut montrer que la solution théorique de l'équation* $y'' = -\sin(y)$ *est strictement croissante et converge vers* $\pi$ (physiquement : le pendule a exactement l'énergie cinétique nécessaire et suffisante pour tendre vers le demi-tour... sans jamais l'atteindre, mais sans retomber non plus ! C'est bien entendu irréaliste, puisqu'en pratique il y a d'une part des frottements, et que d'autre part ça n'a pas de sens d'avoir *exactement* $y'(0) = 2$).

*Mettre en concurrence les quatre méthodes de résolution numérique sur cet exemple, et observer sur un même graphique les solutions calculées sur l'intervalle* $[0,30]$.

## 4 Un peu de physique-chimie

EXERCICE 7 *On jette un projectile M à partir du point O avec une vitesse* $\overrightarrow{v_0}$ *constante en norme, mais faisant un angle variable avec l'horizontale. Si le projectile est soumis seulement à la gravitation, la trajectoire est connue*[3] *: ce sera une parabole. On démontre même de façon classique que l'ensemble des trajectoires est « enveloppé » par une parabole, dite de sécurité. Cet exercice a pour objet la visualisation de cette parabole.*

1. *Faire un dessin !*
2. *Vectorialiser l'équation* $\frac{d^2}{dt^2}\overrightarrow{OM} = \overrightarrow{g}$ *(on doit se retrouver en dimension* 4, *non ?).*
3. *Résoudre numériquement l'équation* $\frac{d^2}{dt^2}\overrightarrow{OM} = \overrightarrow{g}$.
4. *Représenter une vingtaine de paraboles, avec des angles répartis sur* $[0, \pi/2]$.
5. *On suppose maintenant que le projectile est soumis à une force de frottement de la forme* $-k\overrightarrow{v}$. *Modifier les équations, les vectorialiser, et représenter à nouveau quelques trajectoires du projectile*[4].

EXERCICE 8 *On s'intéresse ici aux concentrations de trois produits (A, B et C) au cours du temps. Deux réactions entrent en jeu :* $A \longrightarrow B$ *d'une part et* $B \longrightarrow C$ *d'autre part. Ces réactions sont d'ordre un : leur vitesse est proportionnelle à la concentration du réactif. Les concentrations respectent donc des lois de la forme :*

$$\begin{cases} \frac{d[A]}{dt} &= -\alpha[A] \\ \frac{d[B]}{dt} &= \alpha[A] - \beta[B] \\ \frac{d[C]}{dt} &= \beta[B] \end{cases}$$

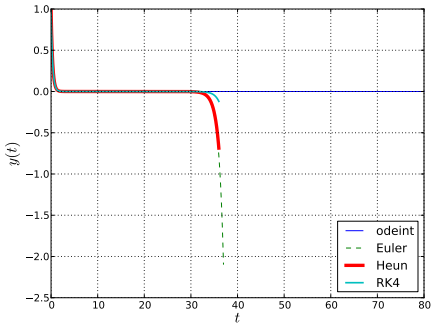
*On part de concentrations initiales* $[A] = 1$ *et* $[B] = [C] = 0$.

1. *Résoudre numériquement*[5] *ces équations différentielles, puis les représenter pour* $t \in [0,6]$, *avec* $(\alpha, \beta) = (1,1)$, $(\alpha, \beta) = (10,1)$ *puis* $(\alpha, \beta) = (1,10)$.
2. *On suppose maintenant que les vitesses des réactions sont de la forme* $\alpha'[A]^2$ *et* $\beta'[B]^2$ *(réactions d'ordre deux). Modifier le système différentiel et observer les évolutions des trois concentrations au cours du temps.*

## 5 Pour ceux qui s'ennuient

EXERCICE 9 *La résolution numérique de l'équation* $y'' = -2y' + 3y$ *avec les conditions initiales* $y(0) = 1$ *et* $y'(0) = -3$ *produit les résultats suivants :*

3. La résolution des équations $x'' = 0$ et $z'' = -g$ ne pose pas trop de problème...
4. Ici, on ne connaît pas de solution analytique de ces solutions.
5. Avec cette modélisation très simple, on peut même sans trop de mal déterminer une solution analytique.




*Tenter une explication. Que doit-il se passer avec l'équation translatée* $y'' = -2y' + 3y - 1$ *sous les conditions initiales* $y(0) = \frac{4}{3}$ *et* $y'(0) = -3$ *? Vérifier !*

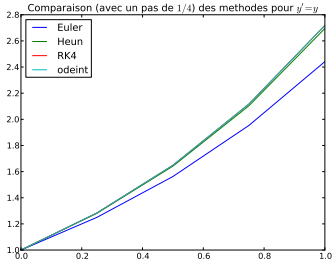
EXERCICE 10 *Représenter quelques portraits de phase de l'équation de van der Pol :*

$$x'' = \mu(1 - x^2)x' - x.$$

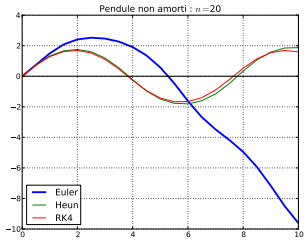
Il s'agit de représenter des trajectoires de $(x, x')$, avec différentes conditions initiales. On pourra prendre $\mu = 1$

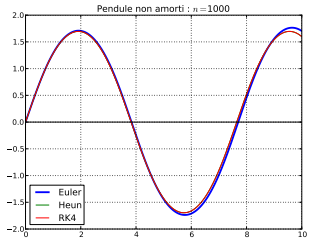
## 6 Les jolis dessins qu'on obtient

- Exercice 3. Pour un pas de 1/4, la méthode de Runge-Kutta d'ordre 4 donne déjà un résultat indiscernable de celui fourni par `odeint`.



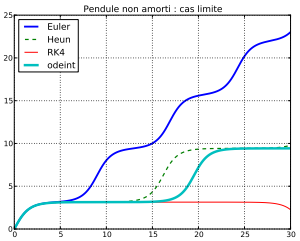

- Exercice 5. Pour $n = 20$ et $n = 1000$, on trouve :



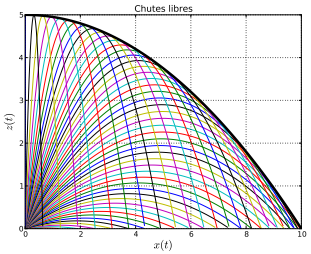



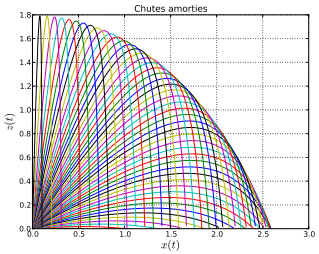



- Exercice 6. Aucune méthode [6] ne fournit une solution convergente, et c'est normal !

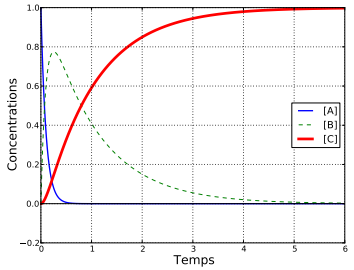


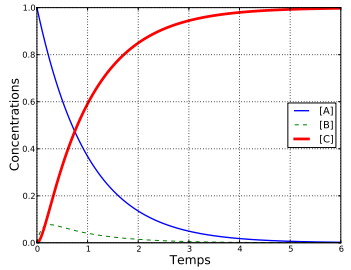

- Exercice 7. Sans frottement, on voit la parabole se dessiner (figure de gauche). Sur la figure de droite, on voit l'effet des frottements, qui raccourcissent les trajectoires :



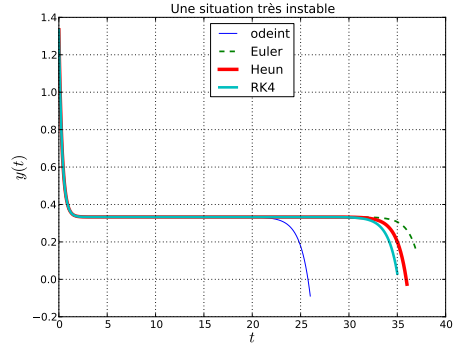

- Exercice 8. Pour $(\alpha, \beta) = (10, 1)$ puis $(\alpha, \beta) = (1, 10)$, on obtient :



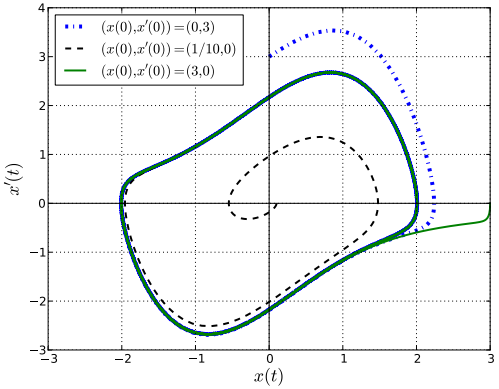




- Exercice 9. La solution théorique tend vers $\frac{1}{3}$, et sa solution approchée n'a aucune chance de passer accidentellement par cette valeur (codage binaire des flottants), donc va diverger :

6. $n = 10^4$ pour les trois premières.




- Exercice 10. Pour obtenir la figure suivante, on a pris $\mu = 1$. On voit apparaître le « cycle limite » bien connu des matheux.






# Corrigé du TP 12 Schémas numériques de résolution d'équations différentielles

Chenevois-Jouhet-Junier

## 1 Imports de bibliothèques et réglage du répertoire de travail

- On va utiliser les bibliothéques/modules suivant(e)s :

```
#imports pour tout le TP
import os.path
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
```

- Toutes les figures seront rangées dans un répertoire `jolis-dessins` situé dans le répertoire de travail `TP12` contenant le script. Avec `Pyzo`, il faut régler le répertoire de travail (point de départ du chemin relatif des ressources externes) en exécutant le script avec l'option `Run as script` ou la séquence de touches `CTRL + SHIFT + E`.

```
REPERTOIRE = 'jolis-dessins' #nom du répertoire de stockage des jolis dessins
```

- On peut vérifier la valeur du répertoire de travail avec la fonction `pwd` si l'interpréteur est Ipython.

```
"""
Running script: "/home/fjunier/InfoPCSI/843/TP843/12-euler-rk4/2015-2016/corrige-tp12-euler-rk4-201-20

In [1]: pwd
Out[1]: '/home/fjunier/InfoPCSI/843/TP843/12-euler-rk4/2015-2016'
"""
```

## 2 Rappel sur le schéma numérique d'Euler (ordre 1)

```
def euler(F, a, y0, b, n):
    """Retourne le tableau des approximations
    de y'=F(t,y) avec y(a)=y0 par la methode d'Euler avec un pas  de 1/n
    """
```

```
    les_yk = [y0]         # la liste des valeurs calculées
    t = a                # le temps du dernier calcul
    h = float(b-a) / n  # le pas
    dernier = y0          # la dernière valeur calculée
    for i in range(n):
        suivant = dernier + h*F(dernier, t) # le nouveau terme
        les_yk.append(suivant)  # on le place à la fin des valeurs calculées
        t = t+h                 # le nouveau temps
        dernier = suivant       # et on met à jour le dernier terme calculé
    return les_yk  # c'est fini


"""
In [6]: euler(lambda y, t : y, 0, 1, 1, 10)
Out[6]:
[1,1.1,1.2100000000000002,1.3310000000000002,1.4641000000000002,1.61051,
 1.7715610000000002,1.9487171,2.1435888100000002,2.357947691,2.5937424601]
"""
```

## 3 Exercice 1 Schéma numérique de Heun (ordre 2)

```
def heun(F, a, y0, b, n):
    """Retourne le tableau des approximations
    de y'=F(t,y) avec y(a)=y0 par la methode de Heun avec un pas  de 1/n
    """
    les_yk = [y0]         #la liste des valeurs calculées
    t = a                #le temps du dernier calcul
    h = float(b-a) / n  #le pas
    dernier = y0          # la dernière valeur calculée
    for i in range(n):
        eul = dernier + h*F(dernier, t) # ce que propose Euler
        suivant = dernier + h/2*(F(dernier, t) + F(eul, t+h)) #nouveau terme
        les_yk.append(suivant)  #on le place à la fin des valeurs calculées
        t = t+h                 #nouveau temps
        dernier = suivant       #on met à jour le dernier terme calculé
    return les_yk


def f1(z, t):
    return z

"""
In [8]: heun(f1, 0, 1, 1, 2)
Out[8]: [1, 1.625, 2.640625]
"""
```

## 4 Exercice 2 Schéma numérique de Runge-Kutta (ordre 4)

```
def RK4(F, a, y0, b, n):
    """Retourne le tableau des approximations
    de y'=F(t,y) avec y(a)=y0 par la methode de RK4 avec un pas  de 1/n
    """
    les_yk = [y0]         # la liste des valeurs calculées
    t = a                # le temps du dernier calcul
    h = float(b-a) / n  # le pas
    dernier = y0          # la dernière valeur calculée
    for i in range(n):
        alpha = dernier + h/2*F(dernier, t)
        beta = dernier + h/2*F(alpha, t+h/2)
        gamma = dernier + h*F(beta, t+h/2)
        suivant = dernier + h/6*(F(dernier, t) + 2*F(alpha, t+h/2) +
                        2*F(beta, t+h/2) + F(gamma,t+h)) # le nouveau terme
        les_yk.append(suivant)  # on le place à la fin des valeurs calculées
        t = t+h                 # le nouveau temps
        dernier = suivant       # et on met à jour le dernier terme calculé
    return les_yk


"""
In [10]: RK4(f1, 0, 1, 1, 2)
Out[10]: [1, 1.6484375, 2.71734619140625]
"""
```

## 5 Exercice 3 Comparaison des quatre schémas numériques pour la résolution approchée de $y' = y$ avec $y(0) = 1$

```
def exo3():
    """Comparaison des méthodes Euler, Heun, RK4, odeint (pas variable)
    sur le problème de Cauchy y'=y avec y(0)=1 pour n=4, n=10, n=100, n=1000"""
    e = np.exp(1)
    print("exp(1)=%.15f"%e)
    #500 temps pour représenter la fonction exponentielle sur les graphiques
    tcontinu = np.linspace(0, 1, 500)
    #y exact (ou presque)
    yexpo = np.exp(tcontinu)
    #comparaison numérique pour 10, 100, 1000 temps
    for n in [4, 10, 100, 1000]:
        t = np.linspace(0,1,n+1)
        #pour odeint l'ordre des paramètres de la fonction change
        yodeint = odeint(lambda z, t : z, 1, t)
        yeuler = euler(f1, 0, 1, 1, n)
        yheun = heun(f1, 0, 1, 1, n)
        yrk4 = RK4(f1, 0, 1, 1, n)
        eul, heu, rk4, od = [y[-1] for y in [yeuler, yheun, yrk4, yodeint]]
        print("Pour n=%i :\n\t\t%.15f\t%.15f\t%.15f\t%.15f"%(n,eul,heu,rk4,od))
        print("Erreurs absolues : \n\t\t%.15f\t%.15f\t%.15f\t%.15f\n"
              %(abs(eul-e), abs(heu-e), abs(rk4-e),abs(od-e)))
        #graphique
        #les temps t_k, il en  faut le meme nombre que les y_k
        t = np.linspace(0, 1, n+1)
```

```
        plt.plot(t, yeuler,color='red', linestyle='dashed', label='Euler')
        plt.plot(t, yheun,color='navy',linestyle='dotted', label='Heun')
        plt.plot(t, yrk4,color='blue',linestyle='solid', label='RK4')
        plt.plot(t, yodeint,color='green',linestyle='dashdot', label='odeint')
        plt.plot(tcontinu, yexpo,color='black',linestyle='solid',linewidth=1.5, label='expo')
        plt.legend(loc = 'upper left')
        plt.title(r"Comparaison (avec un pas de $1/%s$) des methodes pour $y'=y$"%n)
        plt.savefig(os.path.join(REPERTOIRE,
        'comparaison-methodes-%s-pas.png'%n), dpi=80)
        #plt.show()
        plt.clf() # faire de la place pour une nouvelle figure


"""
>>> exo3()
exp(1)=2.718281828459045
Pour n=4 :
2.441406250000000   2.694855690002441   2.718209939201323   2.718281903130344
Erreurs absolues :
0.276875578459045   0.023426138456604   0.000071889257722   0.000000074671299

Pour n=10 :
2.593742460100000   2.714080846608224   2.718279744135166   2.718281900949100
Erreurs absolues :
0.124539368359045   0.004200981850821   0.000002084323879   0.000000072490055

Pour n=100 :
2.704813829421526   2.718236862559957   2.718281828234404   2.718281899096809
Erreurs absolues :
0.013467999037519   0.000044965899088   0.000000000224641   0.000000070637764

Pour n=1000 :
2.716923932235896   2.718281375751763   2.718281828459025   2.718281949830612
Erreurs absolues :
0.001357896223149   0.000000452707282   0.000000000000020   0.000000121371567
"""
```



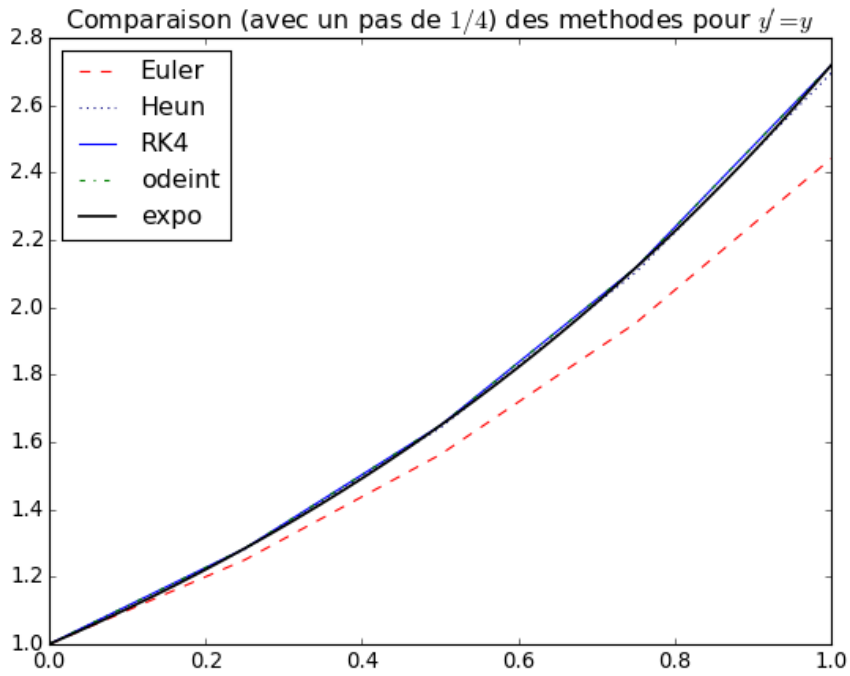



Figure 1:

## 6 Exercice 4 Passage à l'ordre 2, résolution de l'équation du pendule non amorti $y'' = -sin(y)$ avec le schéma d'Euler

On vectorialise en posant $Y(t) = (y(t), y'(t))$ et $F(Y(t), t) = (y'(t), -sin(y(t)))$

On a donc F de $\mathbb{R}^2 \times \mathbb{R}$ dans $\mathbb{R}^2$ telle que $F((a, b), t) = (b, -sin(a))$.

```
def f_pendule(z, t):
    y, yp = z  #y=z[0] et yp=z[1]
    return np.array([yp, -np.sin(y)])


def exo4():
    """Fonction test de l'exo 4
    Approximation de la solution de y'' = -sin(y) sur [0;10]
```

```
    avec y(0)=0 et y'(0)=1.5"""
    #Y est un tableau d'array qui sont les approximations
    #de (y,y') aux temps tk=10k/100  (0<=k<=100)
    Y = euler(f_pendule, 0, np.array([0,1.5]), 10, 100)
    t = np.linspace(0, 10, 101)
    Yarray = np.array(Y)
    plt.plot(t, Yarray[:, 0]) #on extrait la première colonne de Y
    plt.title(r"Résolution (Euler) de  $y''=-\sin(y)$ avec $y(0)=0$ et $y'(0)=1,5$ sur $[0;10]$, $100$
    plt.savefig(os.path.join(REPERTOIRE, 'exo4-pendule-amorti.png'), dpi=60)
    plt.clf() # faire de la place pour une nouvelle figure
```

## 7 Exercice 5 Comparaison des quatre méthodes pour la résolution approchée de $y'' = -\sin(y)$ avec $y(0) = 0$ et $y'(0) = 1,5$ sur $[0; 10]$

```
def exo5():
    """Fonction test de l'exo 5
    Solutions  de y'' = -sin(y) sur [0;10]
    avec y(0)=0 et y'(0)=1.5
    pour n dans {20,100,1000}"""
    #Y est un tableau d'array qui sont les approximations
    #de (y,y') aux temps tk=10/k  (0<=k<=100)
    methode = [euler, heun, RK4, odeint]
    #couleur = ['red', 'blue', 'green',  'black']
    style = ['--','-.','-','-']
    width = [2, 2, 2, 4]
    etiquette = ['Euler', 'Heun', 'RK4', 'odeint']
    for n in [20, 100, 1000]:
        t = np.linspace(0, 10, n+1)
        for i in range(4):
            if i < 3:
                Y = methode[i](f_pendule, 0, np.array([0,1.5]), 10, n)
                Yarray = np.array(Y)
                #on extrait la première colonne de Y
                plt.plot(t, Yarray[:, 0], linestyle=style[i],
                                    linewidth=width[i], label=etiquette[i])
            else:
                Y = odeint(f_pendule, np.array([0,1.5]), t)
                Yarray = np.array(Y)
                #on extrait la première colonne de Y
                plt.plot(t, Yarray[:, 0], linestyle=style[i],
                                    linewidth=width[i], label=etiquette[i])
        plt.title(r"Résolution de  $y''=-\sin(y)$ avec $y(0)=0$ et $y'(0)=1,5$ sur $[0;10]$, %d pas"%n
        plt.legend(loc = 'upper left')
        plt.savefig(os.path.join(REPERTOIRE,'exo5-%d-pas.png'%n), dpi=60)
        plt.clf() # faire de la place pour une nouvelle figure
```



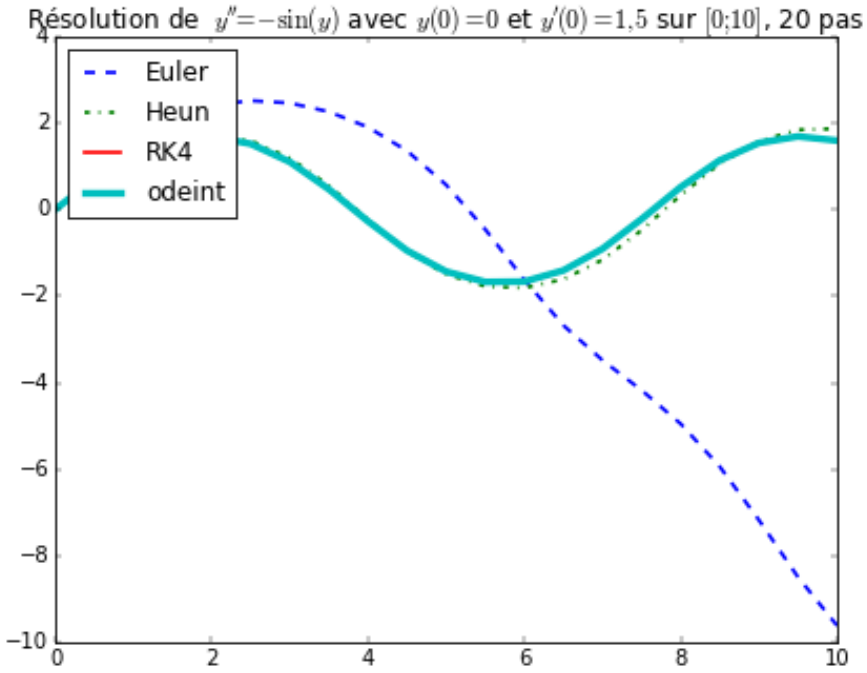



Figure 2:

## 8 Exercice 6

```
def exo6():
    """Fonction test de l'exo 6
    Solutions  de y'' = -sin(y) sur [0;30]
    avec y(0)=0 et y'(0)=2
    La solution théorique est strictement croissante
    et converge vers pi
    """
    #Y est un tableau d'array qui sont les approximations
    #de (y,y') aux temps tk=10/k  (0<=k<=100)
    methode = [euler, heun, RK4, odeint]
    #couleur = ['red', 'blue', 'green',  'black']
    style = ['--','-.','-','-']
    width = [2, 2, 2, 4]
    n = 1000
    etiquette = ['Euler', 'Heun', 'RK4', 'odeint']
    t = np.linspace(0, 30, 1+n)
    for i in range(4):
        if i < 3:
            Y = methode[i](f_pendule, 0, np.array([0,2]), 30, n)
        else:
            Y = methode[i](f_pendule, np.array([0,2]), t)
        Yarray = np.array(Y) #conversion en array pour le slicing
        #on extrait la première colonne de Y
        plt.plot(t, Yarray[:, 0], linestyle=style[i],
                                  linewidth=width[i], label=etiquette[i])
    plt.legend(loc = 'upper left')
    plt.title(r"Résolution de  $y''=-\sin(y)$ avec $y(0)=0$ et $y'(0)=2$ sur $[0;30]$, $1000$ pas")
    plt.savefig(os.path.join(REPERTOIRE,'exo6-pendule-limite.png'), dpi=60)
    plt.clf() # faire de la place pour une nouvelle figure
```

## 9 Exercice 7 Un peu de physique : Chute libre et parabole de sécurité

- On va résoudre numériquement l'équation différentielle d'ordre 2 : $\frac{d^2}{dt^2}\overrightarrow{OM(t)} = g$ où $g$ est l'accélération de la pesanteur.
- On vectorialise en dimension 4 en résolvant :

$$Y' = F(Y,t) \text{ où } Y = (x, z, x', z') \text{ et } F(Y,t) = (x', z', x_g, z_g)$$

- Les constantes de l'accélération de la pesanteur à la surface terrestre sont environ $x_g = 0$ et $z_g = -10$ et puisque le projectile est lancé avec une vitesse $v_0$ constante et avec un angle $\alpha$ variable avec l'horizontale, les composantes de la vitesse initiale sont $(v_0\cos(\alpha), v_0\sin(\alpha))$.



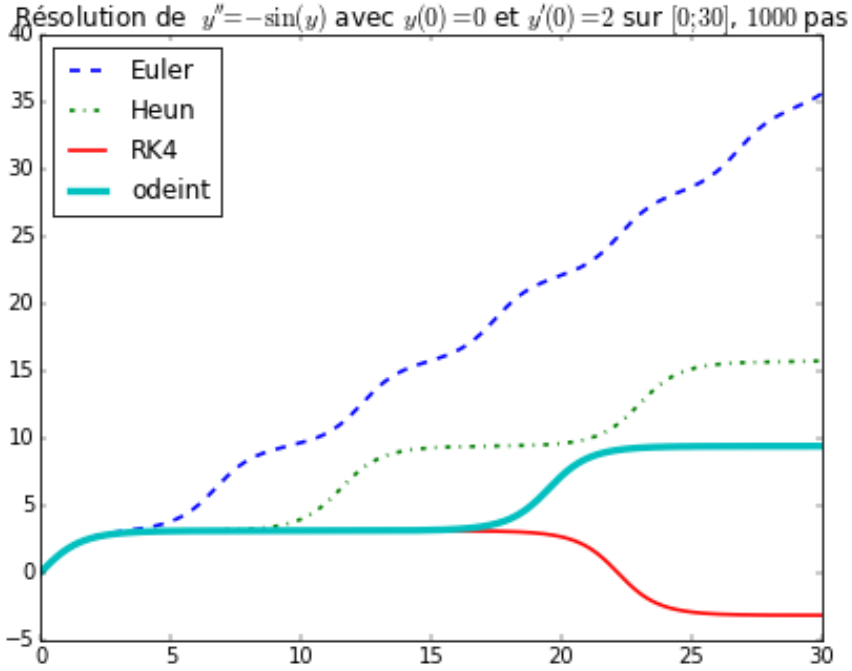



Figure 3:

```
g = 10. #accélération de la pesanteur

def Flibre(V, t):
    x, z, xp, zp = V
    return np.array([xp, zp, 0, -g])

def chutelibre(v0, alpha, tmax, n):
    """
    parabole approchée avec odeint
    sur l'intervalle entre 0 et tmax avec n temps
    et les conditions initiales vo et alpha
    """
    t = np.linspace(0, tmax, n+1)
    yodeint = odeint(Flibre, np.array([0, 0, v0*np.cos(alpha),
                                v0*np.sin(alpha)]), t)
    #on ne garde que les z>=0
    #puis on convertit en array  pour le slicing
    Yodeint = np.array([y[:2] for y in yodeint if y[1] >= 0])
    plt.plot(Yodeint[:,0], Yodeint[:,1])

def exo7question3(v0, tmax, n):
    angle = np.linspace(0, np.pi/2, 50)
    for alpha in angle:
        chutelibre(v0, alpha, tmax, n)
    #tracé de la parabole de surete
    #d'équation z = v0**2/(2*g)-g/(2*v0**2)*x**2
    #elle coupe l'axe des abscisses en v0**2/g
    x=np.linspace(0,v0**2/g,100)
    plt.plot(x,v0**2/2/g-g/v0**2*x*x/2,linewidth=4,color='black')
    plt.xlabel(r'$x(t)$',fontsize=18)
    plt.ylabel(r'$z(t)$',fontsize=18)
    plt.title('Chutes libres')
    plt.savefig(os.path.join(REPERTOIRE,'exo7-libre-vitesse-%i.png'%v0), dpi=60)
    plt.clf() # faire de la place pour une nouvelle figure


"""
>>> exo7question3(10, 10, 1000)
"""
```

---

- On suppose maintenant que le projectile est soumis à des forces de frottement de la forme $-kv$
- On va donc résoudre numériquement $\frac{d^2}{dt^2}\overrightarrow{OM(t)} = g - k_m \times v$
- On considère :

$$Y' = F(Y,t) \text{ où } Y = (x, z, x', z') \text{ et } F(Y,t) = (x', z', -k_m \times x_p, -g - k_m \times z_p)$$

```
k_m = 4 #coefficient de frottement
```



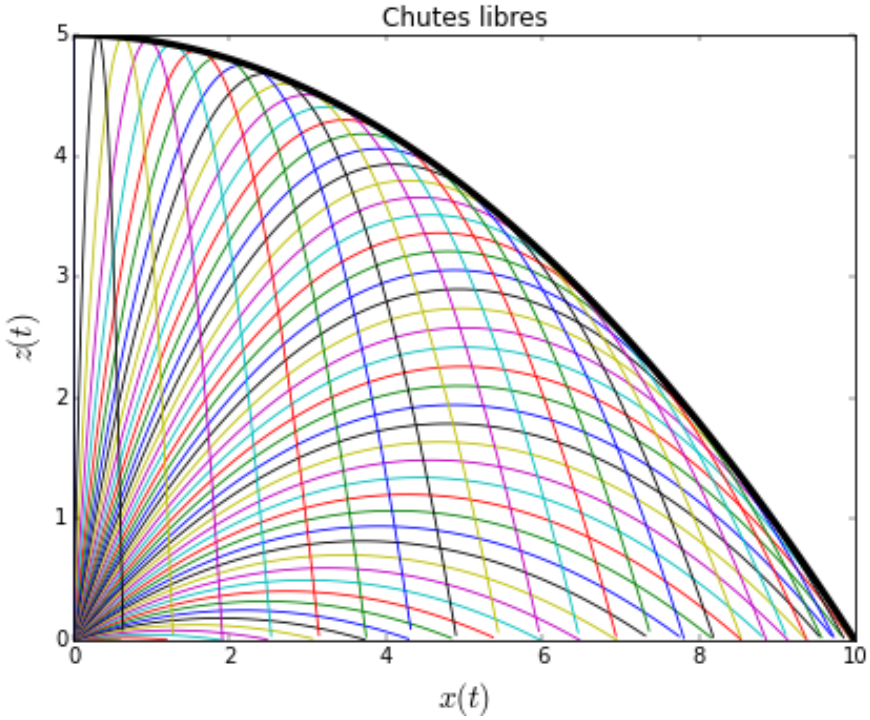



Figure 4:

```
def Famortie(V,t):
    x, z, xp, zp = V
    return np.array([xp, zp,-k_m*xp,-g-k_m*zp])

def chuteamortie(v0, alpha, tmax, n):
    t = np.linspace(0, tmax, n+1)
    yodeint = odeint(Famortie, np.array([0, 0, v0*np.cos(alpha),
                                v0*np.sin(alpha)]), t)
    #on ne garde que les z>=0
    #puis on convertit en array  pour le slicing
    Yodeint = np.array([y[:2] for y in yodeint if y[1] >= 0])
    plt.plot(Yodeint[:,0], Yodeint[:,1])

def exo7question5(v0, tmax, n):
    angle = np.linspace(0, np.pi/2, 50)
    for alpha in angle:
        chuteamortie(v0, alpha, tmax, n)
    plt.xlabel(r'$x(t)$',fontsize=18)
    plt.ylabel(r'$z(t)$',fontsize=18)
    plt.title('Chutes amorties')
    plt.savefig(os.path.join(REPERTOIRE,'exo7-amortie-vitesse-%i.png'%v0), dpi=60)
    plt.clf() # faire de la place pour une nouvelle figure


"""
>>> exo7question5(10, 10, 1000)
"""
```



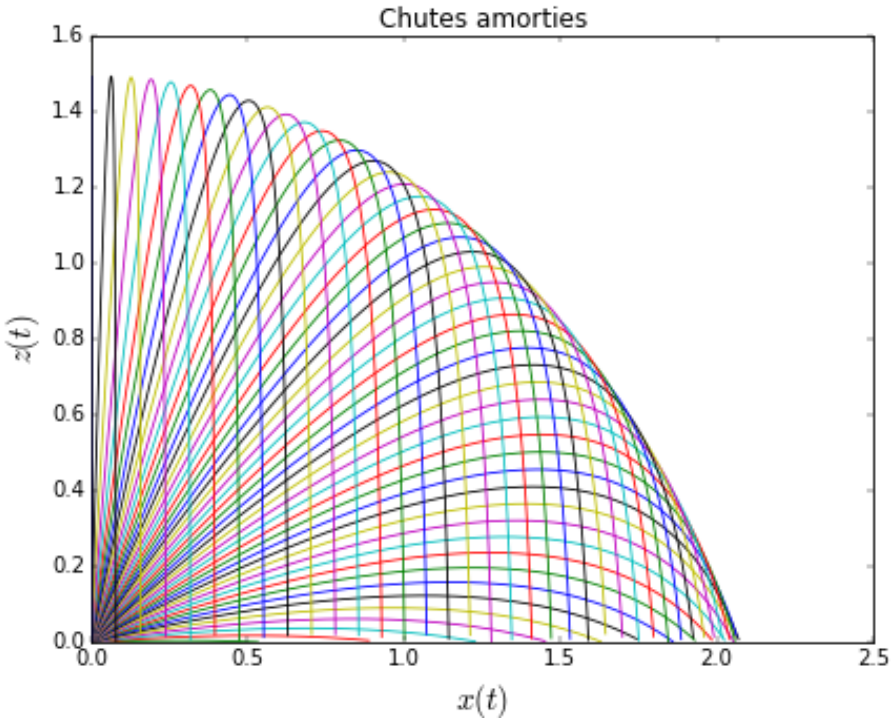



Figure 5:

## 10 Exercice 8 Cinétique chimique

```
def Ford(Y,t,ordre):
    A, B, C = Y
    return np.array([-alpha*A**ordre, alpha*A**ordre - beta*B**ordre,
     beta*B**ordre])

def exo8(F, ordre=1):
    global alpha, beta #alpha, beta variables globales
    Fbis = lambda Y, t : F(Y, t, ordre)
    for alpha, beta in [(1, 1), (10, 1), (1, 10)]:
        t = np.linspace(0, 6, 1000)
        yodeint = odeint(Fbis, np.array([1, 0, 0]), t)
        Yodeint = np.array(yodeint)
```

```
plt.plot(t, Yodeint[:,0], linestyle='--') #tracé de la concentration de A
plt.plot(t, Yodeint[:,1],linestyle='-.') #tracé de la concentration de B
plt.plot(t, Yodeint[:,2]) #tracé de la concentration de C
plt.title(r'Concentrations chimiques $\alpha$=%d, $\beta$=%d'%(alpha,beta))
plt.legend(['[A]', '[B]', '[C]'], loc = 'center right')
plt.grid(True)
plt.savefig(os.path.join(REPERTOIRE,'exo8-ordre%d-alpha%d-beta%d.pdf'%(ordre,alpha,beta)))
plt.clf() # faire de la place pour une nouvelle figure
```

## 11 Exercice 9 Instabilité des schémas numériques

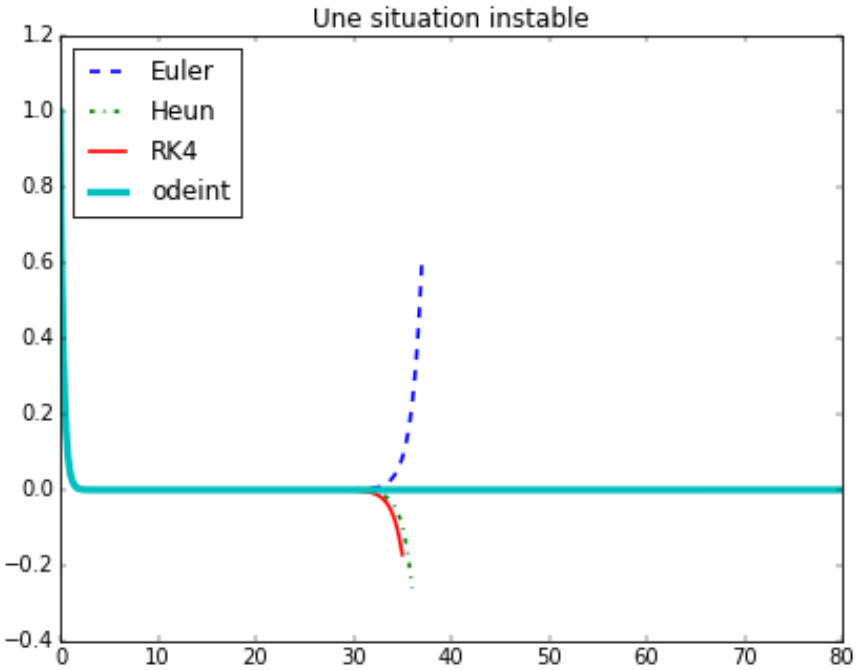



Figure 6:

- La solution générale de $y'' = -2y' + 3y$ est de la forme $y(x) = C_1 e^{-3x} + C_2 e^{x}$. On peut le vérifier avec la fonction `sympy.dsolve` du module de calcul formel `sympy`



```
"""
In [18]: import sympy

In [19]: from sympy.abc import f, x

In [20]: diffeq = sympy.Eq(f(x).diff(x,x) + 2*f(x).diff(x) - 3*f(x),0)

In [21]: diffeq
Out[21]: -3*f(x) + 2*Derivative(f(x), x) + Derivative(f(x), x, x) == 0

In [22]: sympy.dsolve(diffeq, f(x))
Out[22]: f(x) == C1*exp(-3*x) + C2*exp(x)
"""
```

- La solution particulière pour les conditions initiales $y(0) = 1$ et $y'(0) = -3$ est $y(x) = e^{-3x}$
- Elle vérifie pour tout réel $x$, $y'(x) = -3 \times e^{-3x}$ et donc $y'(x) + 3y(x) = 0$.

  C'est vrai pour $x = 0$.

  Dès les premières étapes du schéma, avec les petites approximations, cette égalité n'est plus vérifiée (les réels sont représentés de façon approchée par des flottants).

  A chaque étape résout alors numériquement des équations différentielles vérifiant des conditions initiales légèrement différentes.

  Leurs solutions vont comporter un terme en $e^x$ qui va diverger quand $x$ va tendre vers $+\infty$.

```
def instable1(Y, t):
    y, yp = Y
    return np.array([yp, -2*yp + 3*y])
```

```
def exo9(F, y0, yp0):
    methode = [euler, heun, RK4, odeint]
    if F.__name__ == "instable1":
        tmax = [37, 36, 35, 80] #tmax varie selon le schéma
    else:
        tmax = [37, 36, 35, 26]
    style = ['--','-.','-','-']
    width = [2, 2, 2, 4]
    n = 1000
    for i in range(4):
        t = np.linspace(0, tmax[i], 1+n)
        if i < 3:
            Y = methode[i](F, 0, np.array([y0, yp0]), tmax[i], n)
        else:
            Y = methode[i](F, np.array([y0, yp0]), t)
        Yarray = np.array(Y) #conversion en array pour le slicing
        #on extrait la première colonne de Y
        plt.plot(t, Yarray[:, 0], linestyle=style[i],linewidth=width[i])
    plt.legend(['Euler', 'Heun', 'RK4', 'odeint'], loc = 'upper left')
    plt.title('Une situation instable')
```

```
    plt.savefig(os.path.join(REPERTOIRE,'exo9-%s.png'%F.__name__), dpi=60)
    plt.clf() # faire de la place pour une nouvelle figure

"""
>>> exo9(instable1,1,-3)
"""
```

---

- On résout l'équation translatée $y'' = -2y' + 3y - 1$ en prenant comme conditions initiales $y(0) = \frac{4}{3}$ et $y'(0) = -3$
- La solution générale de cette équation est $y(x) = C_1 e^{-3x} + C_2 e^{x} + \frac{1}{3}$.
- La solution particulière est $y(x) = e^{-3x}$.
- On observe encore une instabilité des schémas numériques de résolution.

```
def instable2(Y, t):
    y, yp = Y
    return np.array([yp, -2*yp + 3*y - 1])


"""
>>> exo9(instable2,4/3.,-3)
"""
```

## 12 Exercice 10 Équation de Vanderpol $x'' = \mu \times (1 - x^2) \times x' - x$

Portrait de phase $(x, x')$ la résolution numérique avec `odeint` de l'équation de Vanderpol $x'' = \mu \times (1 - x^2) \times x' - x$

```
mu = 1

def vanderpol(X,t):
    x, xp = X
    return np.array([xp, xp*mu*(1 - x**2)-x])

def exo10():
    style = ['--','-.','-']
    width = [2, 2, 4]
    cinit = [[0, 3], [0.1, 0], [3, 0]]
    tmax = 50
    n = 1000
    for i in range(3):
        t = np.linspace(0, tmax, 1+n)
        Y = odeint(vanderpol, cinit[i], t)
        Yarray = np.array(Y) #conversion en array pour le slicing
        #on extrait la première colonne de Y
        plt.plot(Yarray[:, 0], Yarray[:, 1], linestyle=style[i],
                                            linewidth=width[i])
    plt.axhline(color='black')
    plt.axvline(color='black')
    plt.xlabel(r'$x(t)$',fontsize=18)
```



```
plt.ylabel(r"$x'(t)$",fontsize=18)
plt.grid()
plt.legend([r"$(x(0),x'(0))=(0,3)$", r"$(x(0),x'(0))=(1/10,0)$",
            r"$(x(0),x'(0))=(3,0)$"], loc = 'upper left')
plt.savefig(os.path.join(REPERTOIRE,'exo10.png'), dpi=80)
plt.clf() # faire de la place pour une nouvelle figure
```

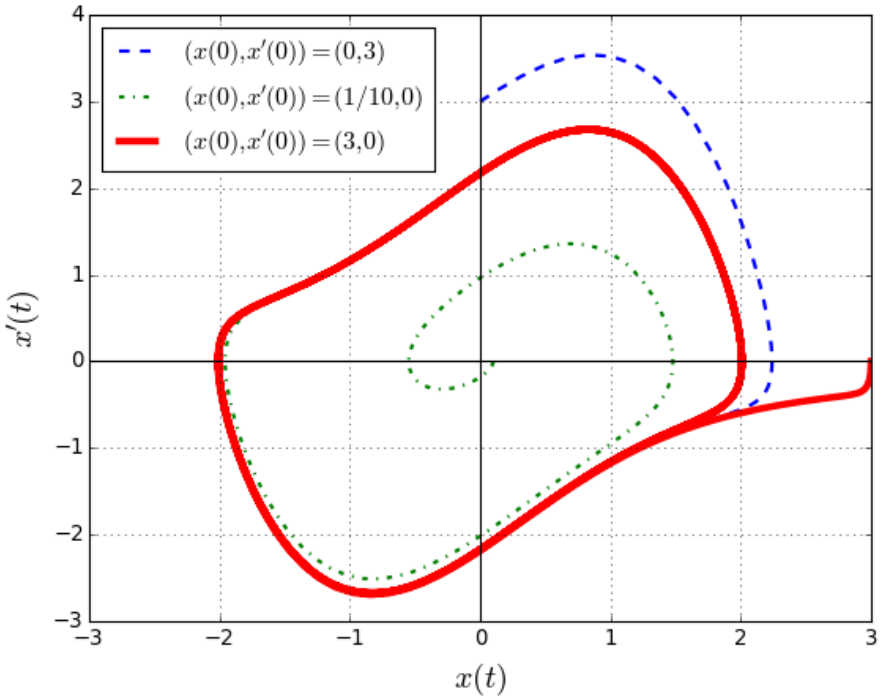



Figure 7: