

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 (jolies) 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

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

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, $F(z, t) = 2z$. Si cet exemple peut sembler un peu stupide¹... 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 $F(z, t) = te^{t^2 z}$, 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 $F((a, b), t) = ((1 - b)a, (a - 1)b)$ (donc F va de $\mathbb{R}^2 \times \mathbb{R}$ dans \mathbb{R}^2), on a bien une équation de la forme $Z'(t) = F(Z(t), t)$, 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 $y''(t) = -\sin(y(t))$, ou encore $\begin{pmatrix} y \\ y' \end{pmatrix}' = \begin{pmatrix} y' \\ -\sin y \end{pmatrix}$ est par exemple équivalente, en posant $Y(t) = (y(t), y'(t))$, à $Y'(t) = F(Y(t), t)$, avec $F((u, v), t) = (v, -\sin u)$.

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 $\mathbb{R}^2 \times \mathbb{R}$ 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 $a = t_0 < t_1 < \dots < t_n = b$.

2 Les quatre candidats

Il s'agit à chaque fois de calculer les approximations y_k de la solution de l'équation $y'(t) = F(y(t), t)$ (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

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

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

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

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(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

```

- Pour la méthode de Heun, on prend :

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

(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 nouvelles². On calcule donc successivement :

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

et enfin :

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

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

On donne 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(z, t):
        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 $y(1) \simeq e$.

2. α_k est une approximation de $y(t_k + h/2)$, injectée dans une nouvelle approximation β_k de $y(t_k + h/2)$, elle-même utilisée pour calculer une première approximation γ_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` et `numpy` sous l'alias `np` par exemple) :

```

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

```

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 de l'équation différentielle du pendule simple $y'' = -\omega_0^2 \sin(y)$, avec $\omega_0 \simeq 5,6 \text{ rad.s}^{-1}$ (question subsidiaire : d'où vient cette valeur?) :

```

import numpy as np

def f_pendule(z, t):
    y, yp = z # ou (y, yp) = (z[0], z[1]).
    return np.array([yp, -31.36*np.sin(y)])

Y = euler(f_pendule, 0, np.array([0, 8]), 2, 100)

```

Si tout se passe bien, `Y` doit être la liste des approximations de (y, y') aux temps $t_k = \frac{10k}{100}$ ($0 \leq k \leq 100$), avec y la solution de $y'' = -\omega_0^2 \sin y$, sous les conditions initiales $y(0) = 0$ et $y'(0) = 8$. 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 = np.array(Y)
t = np.linspace(0, 2, 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 $[0, 2]$ pour $y(0) = 0$, $y'(0) = 8$, 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 $y'(0) = 2\omega_0$, on peut montrer que la solution théorique de l'équation $y'' = -\omega_0^2 \sin(y)$ est strictement croissante et converge vers π (physiquement : vous pourrez montrer (exercice!) que 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* $y'(0) = 2\omega_0$).

Mettre en concurrence les quatre méthodes de résolution numérique sur cet exemple, et observer sur un même graphique les solutions calculées sur l'intervalle $[0, 6]$.

4 Un peu de physique-chimie

EXERCICE 7 On jette un projectile M à partir du point O avec une vitesse \vec{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 connue³ : 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 $\frac{d^2}{dt^2} \vec{OM} = \vec{g}$ (on doit se retrouver en dimension 4, non?).
3. Résoudre numériquement l'équation $\frac{d^2}{dt^2} \vec{OM} = \vec{g}$.
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 $-k\vec{v}$. Modifier les équations, les vectorialiser, et représenter à nouveau quelques trajectoires du projectile⁴.

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 numériquement⁵ 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.

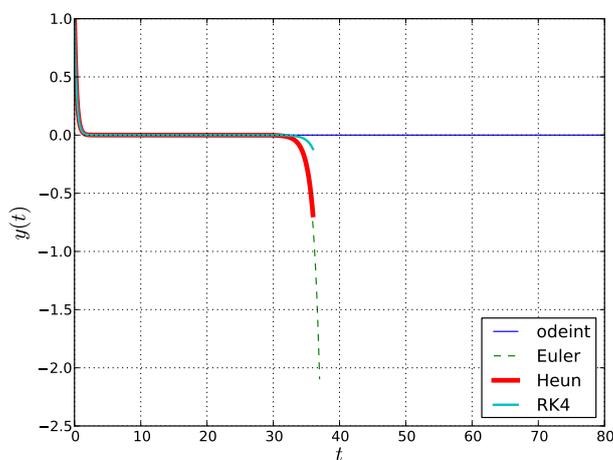
3. La résolution des équations $x'' = 0$ et $z'' = -g$ ne pose pas trop de problème...

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.

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 :



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!

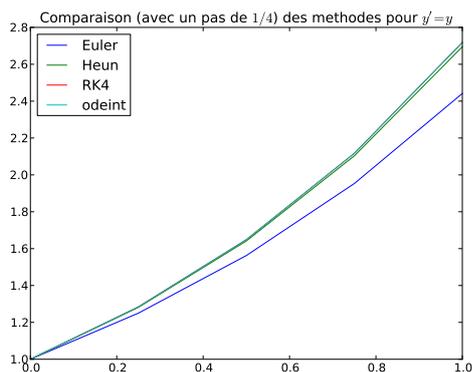
EXERCICE 10 Représenter quelques portraits de phase de l'équation de van der Pol :

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

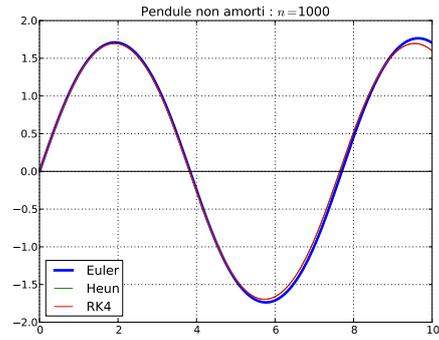
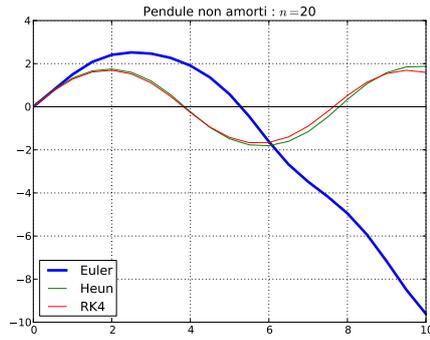
Il s'agit de représenter des trajectoires de (x, x') , avec différentes conditions initiales. On pourra prendre $\mu = 1$

6 Les jolis dessins qu'on obtient

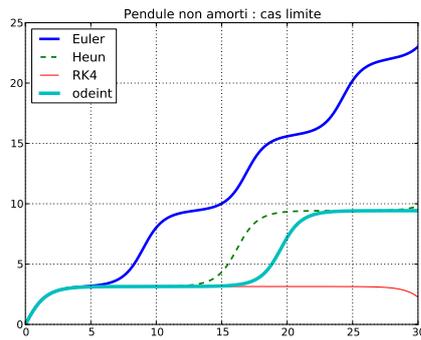
- Exercice 3. Pour un pas de $1/4$, la méthode de Runge-Kutta d'ordre 4 donne déjà un résultat indiscernable de celui fourni par odeint.



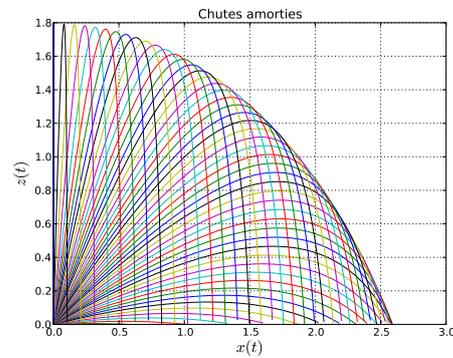
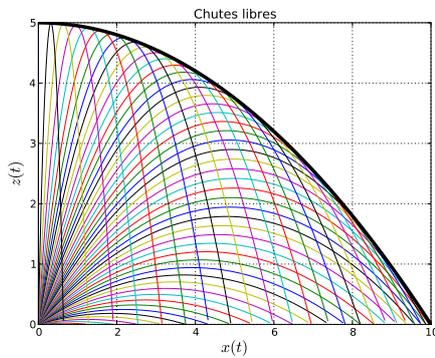
- Exercice 5. Pour $n = 20$ et $n = 1000$, on trouve :



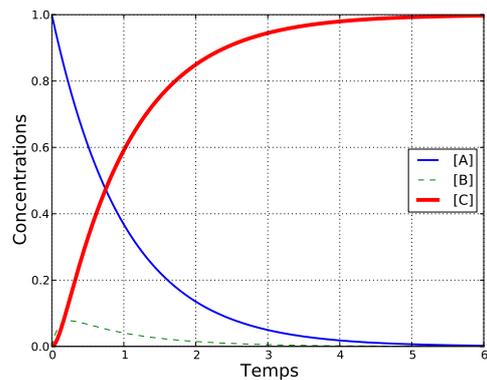
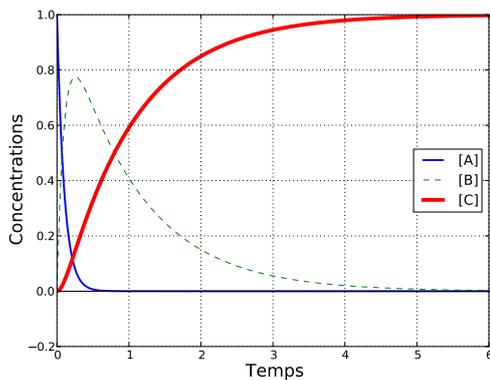
- Exercice 6. Aucune méthode ⁶ ne fournit une solution convergente, et c'est normal!



- 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 :

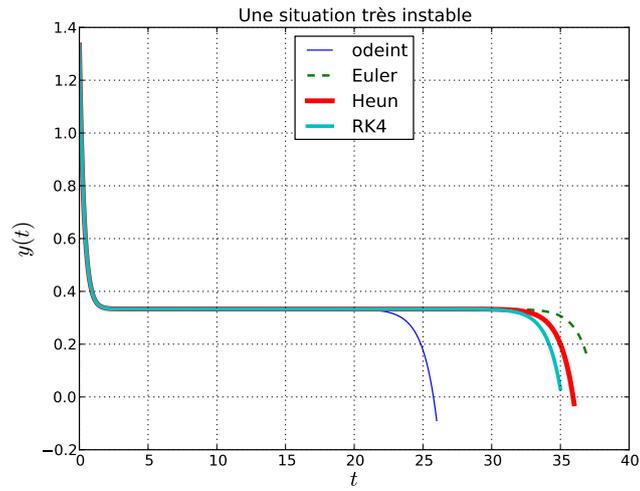


- Exercice 8. Pour $(\alpha, \beta) = (10, 1)$ puis $(\alpha, \beta) = (1, 10)$, on obtient :



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

- 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 :



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

