Intégration numérique et méthode d'Euler

D'après un TP de Stéphane Gonnord

Buts du TP

- ☞ Programmer et tester différentes formules pour approcher numériquement une intégrale.
- ☞ Programmer la méthode d'Euler pour approcher numériquement une solution d'équation différentielle.
- ☞ Représenter le résultat d'une telle approximation.
- ☞ Se confronter à un extrait de sujet de concours.

Les différentes bibliothèques utilisées seront :

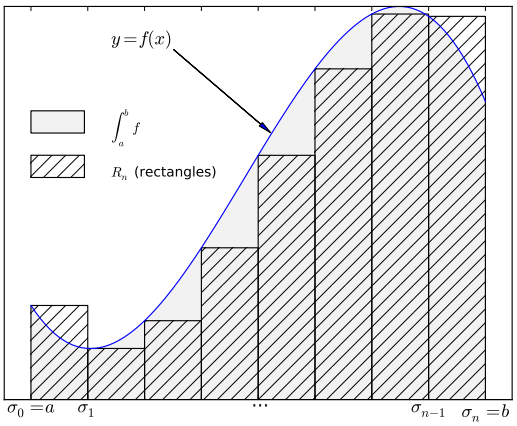
- `math` pour `cos`, `exp`, `sin`, `log`.
- `scipy.integrate` pour `quad`.
- `time` pour `time`...
- `numpy` pour `array`, `linspace` importée sous l'alias `np`.
- `matplotlib.pyplot` pour `plot`, `show`, et autres bricoles : importée sous l'alias `plt`.
- `scipy.optimize` pour `fsolve`.
- `test` pour les fonctions de test des méthodes d'intégration

Créer (au bon endroit) un dossier associé à ce TP.
Lancer Spyder ou Pyzo, sauvegarder immédiatement au bon endroit le fichier `TP9.py`.

1 Trois méthodes d'intégration numérique

Le théorème de Riemann dit que si f est continue (par morceaux) sur un segment $[a,b]$, alors on peut approcher l'aire située sous le graphe de f par la somme des aires des rectangles approchant f en n points uniformément répartis $\sigma_k = a + k\dfrac{b-a}{n}$, pour $0 \leqslant k \leqslant n-1$:

$$R_n = \frac{b-a}{n}\sum_{k=0}^{n-1} f(\sigma_k) \underset{n\to+\infty}{\longrightarrow} \int_a^b f(t)\,dt.$$



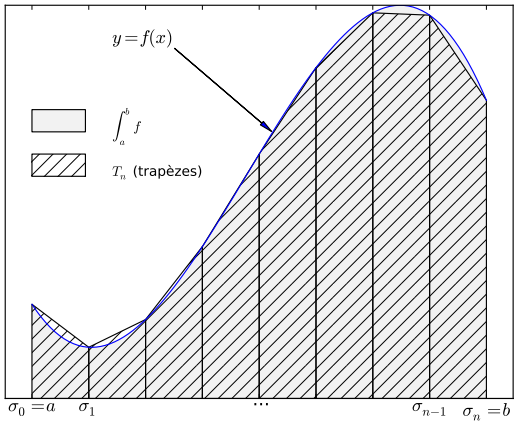
EXERCICE 1 *Écrire une fonction prenant en entrée une fonction, deux bornes et un nombre de pas n, et renvoyant l'approximation R_n de l'intégrale de la fonction.*

```
def rectangles(f, a, b, n):
    ...
```

EXERCICE 2 *Copier/Coller le fichier* `test.py` *depuis le répertoire* `Donnees` *de la classe vers le répertoire courant du TP.*

1. *Ouvrir le fichier* `test.py` *pour analyser son code.*
2. *Insérer la commande* `from test import *` *avec les autres imports de modules au début de* `TP9.py`*, sauvegarder puis exécuter.*
3. *Éditer le fichier* `test.py` [1] *puis choisir la fonction avec les paramètres adaptés pour le test de la méthode des rectangles avec différentes valeurs de n pour approcher des intégrales connues :*
   - $t \mapsto t$ *sur* $[0,1]$ *;*
   - $t \mapsto t^2$ *sur* $[0,1]$ *;*
   - $t \mapsto t^3$ *sur* $[0,1]$ *;*
   - $t \mapsto t^{10}$ *sur* $[0,1]$ *;*
   - $t \mapsto \cos t$ *sur* $[0,\pi/2]$ *;*
   - $t \mapsto \mathrm{e}^t$ *sur* $[-3,3]$.
4. *Comparer les valeurs approchées avec la valeur retournée par* `scipy.integrate.quad` *et conjecturer un ordre de grandeur d'un majorant de l'erreur commise par la méthode des rectangles en fonction du nombre n de subdivisions.*

On peut aussi décider d'approcher f (puis son intégrale) en l'interpolant sur chaque $[\sigma_k, \sigma_{k+1}]$ par des fonctions affines (plutôt que des constantes), ce qui conduit à considérer l'aire de trapèzes :



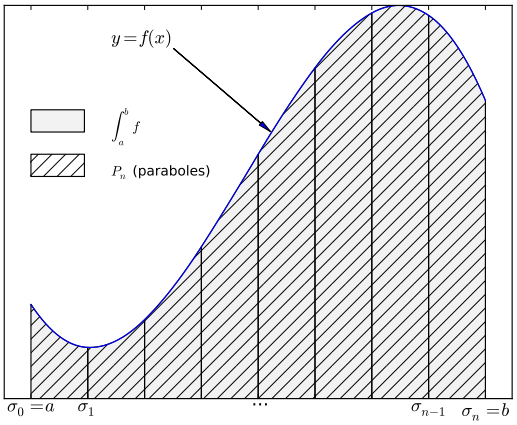
$$T_n = \frac{b-a}{2n}\sum_{k=0}^{n-1}\big(f(\sigma_k)+f(\sigma_{k+1})\big) \underbrace{=}_{\text{exercice}} R_n + \frac{b-a}{2n}\big(f(b)-f(a)\big).$$

EXERCICE 3 *Écrire une fonction prenant en entrée une fonction, deux bornes et un nombre de pas, et renvoyant l'approximation* T_n *de l'intégrale de la fonction.*

Effectuer les mêmes tests que pour la méthode des rectangles et conjecturer un ordre de grandeur d'un majorant de l'erreur commise par la méthode des trapèzes en fonction du nombre n de subdivisions.

On peut enfin (méthode de Simpson) interpoler f à l'aide des valeurs en σ_k, σ_{k+1} et $\frac{\sigma_k+\sigma_{k+1}}{2}$: on a alors localement des paraboles vraiment proches de f.

1. Pour utiliser l'introspection avec `help(test)`, il faut importer le module avec `import test`



Un savant calcul [2] fournit :

$$P_n = \frac{b-a}{6n}\sum_{k=0}^{n-1}\left(f(\sigma_k)+4f\left(\frac{\sigma_k+\sigma_{k+1}}{2}\right)+f(\sigma_{k+1})\right).$$

EXERCICE 4 *Écrire une fonction prenant en entrée une fonction, deux bornes et un nombre de pas, et renvoyant l'approximation P_n de l'intégrale de la fonction.*

Effectuer les mêmes tests que pour la méthode des rectangles et conjecturer un ordre de grandeur d'un majorant de l'erreur commise par la méthode de Simpson en fonction du nombre n de subdivisions.

Comme on s'en doute (voir les dessins...) l'approximation par des paraboles est diaboliquement précise. Des résultats plus précis sur les ordres de convergences... seront données en cours de maths sur demande!

L'exercice suivant est plutôt à faire AVANT le TP...

EXERCICE 5 *Donner le nombre d'évaluations de f nécessaires pour calculer :*

- *R_n ;*
- *T_n avec la formule initiale ;*
- *T_n avec la formule utilisant R_n ;*
- *P_n sans finasser ;*
- *P_n en finassant.*

EXERCICE 6 *Résumer les qualités d'approximation et temps de calcul nécessaires à l'évaluation de l'intégrale $\int_0^{\pi/2}\cos t\,dt = 1$ pour les différentes méthodes et différentes valeurs de n.*

- *Rectangles :*

n	10^2	10^4	10^6
Temps (secondes)			
$\lvert R_n-1\rvert$			

- *Trapèzes :*

n	10^2	10^4	10^6
Temps (secondes)			
$\lvert T_n-1\rvert$			

- *Paraboles (Simpson) :*

n	10^2	10^4	10^6
Temps (secondes)			
$\lvert P_n-1\rvert$			

Indication : pour chronométrer un calcul, on peut opérer ainsi..

2. Faire une recherche Google avec les mots clefs *formule des trois niveaux Simpson*

```
t1 = time.perf_counter()
<mon_calcul>
t2 = time.perf_counter()
```

Le temps du calcul est alors `t2-t1`.
On peut aussi utiliser la fonction `chrono` du module `test` :

```
>>> a, b, f, intexact = 0,math.pi/2.,math.cos, 1
>>> chrono(rectangles, intexact)(f,a,b,10**2)
'1.0078334198735823 en 6.413459777832031e-05 secondes avec 2 décimales justes'
```

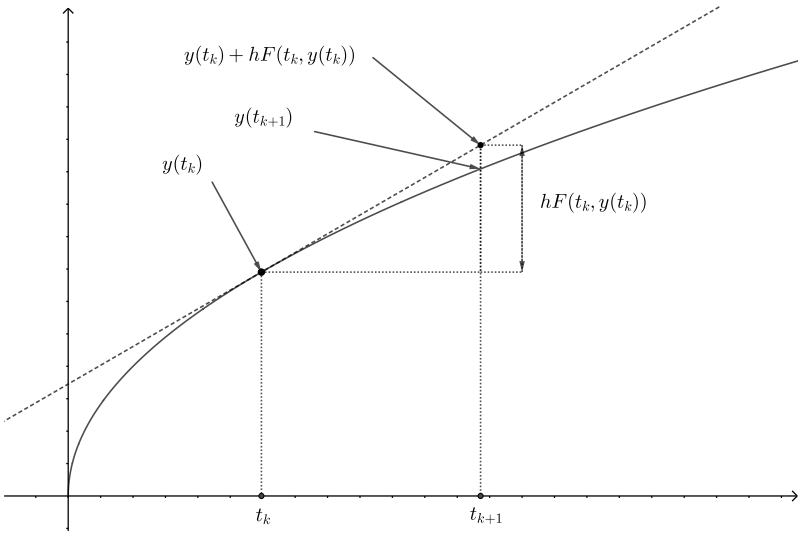
2 Méthode d'Euler

Des théorèmes (désormais complètement hors programme) assurent que sous des conditions raisonnables, il existe une unique application y de classe $\mathscr{C}^1$ sur $[a,b]$ dont la valeur est imposée en a et qui vérifie une équation de la forme $y'(t) = F(t, y(t))$ pour tout $t \in [a,b]$. L'objet des *schémas numériques* est d'obtenir des approximations de ces solutions dont la théorie donne l'existence de façon non constructive. En pratique, on tente en général d'approcher y en un certain nombre de points répartis sur l'intervalle $[a,b]$.

Plus précisément, on veut calculer une approximation y_k des $y(t_k)$, avec $t_k = a + k \times h$, où $h = \dfrac{b-a}{n}$ est un *pas* qu'il conviendra d'ajuster. De façon très simple, si on écrit :

$$y(t_{k+1}) - y(t_k) = \int_{t_k}^{t_{k+1}} y'(u)\,du = \int_{t_k}^{t_{k+1}} F(u, y(u))\,du \simeq hF(t_k, y(t_k))$$

alors on obtient la *méthode d'Euler*.
Les approximations sont calculées de proche en proche via la formule suivante : $y_{k+1} = y_k + hF(t_k, y_k)$.
On initialise bien entendu avec $y_0 = y(a)$, qui sera la seule valeur « exacte » calculée.



EXERCICE 7 *Écrire une fonction calculant les valeurs approchées d'une solution d'équation différentielle, dans le cadre précédent :*

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

La fonction devra renvoyer le tableau des $n+1$ valeurs approchées de y (solution de $y'(t) = F(t, y(t))$ avec $y(t_0) = y_0$) aux $n+1$ temps $t_k = t_0 + k\frac{b-a}{n}$.

Indication : il suffit de calculer les termes de proche en proche. On peut soit créer la liste immédiatement à la bonne taille, soit ajouter chaque nouvel élément en bout de liste via la méthode `append`.

EXERCICE 8 *Tester la fonction précédente sur $[0,1]$ avec l'équation $y' = y$ et la condition initiale $y(0) = 1$ pour différentes valeurs de n.*

```
def f0(t, y):
    return y

y10 = euler(f0, 0, 1, 1, 10)
```

Si tout se passe bien, `y10` *représente alors les valeurs approchées de la fonction exponentielle aux temps* $t_k = \frac{k}{n}\cdot$ *En particulier, l'approximation de la solution en* 1 *est la dernière valeur de ce tableau, et doit donc être relativement proche de* e.

Importer les bibliothèques `numpy` *et* `matplotlib.pyplot` *(par exemple sous les alias* `np` *et* `plt`*); représenter la ligne polygonale rejoignant les points calculés précédemment :*

```
import numpy as np
import matplotlib.pyplot as plt

t10 = np.linspace(0,1,11) #10 subdivision donc 11 bornes
plt.plot(t10, y10)

plt.savefig('exp10.pdf')
plt.show()
```

On pourra mettre sur le même graphique le graphe de la fonction exponentielle :

```
t100 = np.linspace(0, 1, 100)
exp100 = np.exp(t100)
plt.plot(t100, exp100)
```

Vous pourrez voir dans le corrigé comment ajouter des décorations au graphique.

EXERCICE 9 *L'équation du deuxième ordre* $y'' = -y$ *se ramène à la théorie du premier ordre en considérant* $Y(t) = [y(t), y'(t)]$*, qui vérifie* $Y'(t) = F(t, Y(t))$*, où* $F(t, [a, b]) = [b, -a]$.

À l'aide de ces considérations, calculer une approximation de la solution de $y'' = -y$ *sur* $[0, 4\pi]$*, avec* $y(0) = 0$ *et* $y'(0) = 1$ *(la solution de ce problème de Cauchy est la fonction sinus). On découpera l'intervalle en* 100.

On va passer par des `array` de `numpy` pour que l'addition des objets se passe comme dans $\mathbb{R}^2$ lors du calcul de `yk + h*F(t0+k*h, yk)`

```
def f1(t, y):
    [a, b] = y # pratique, non ?
    return np.array([b, -a])

Y100 = euler(f1, 0, np.array([0, 1]), 4*pi, 100)
```

EXERCICE 10 *Représenter la solution calculée dans l'exercice précédent de deux façons différentes :*

- *le graphe de* y *: les points considérés ont donc pour abscisses les* t_k *et pour ordonnées les premières composantes de* `Y100`*, qu'on obtient par slicing* `np.array(Y100)[:, 0]` *;*
- *le portrait de phase, qui est la courbe paramétrée reliant les points de coordonnées* $Y_k = (y_k, y'_k)$.

Le résultat est-il celui attendu ?

Découper l'intervalle en 100, 10^3 *ou* 10^4 *sous-intervalles et observer les effets sur la précision et le temps de calcul.*

3 Extrait du sujet posé au concours Centrale 2015

Soit y une fonction de classe $\mathscr{C}^2$ sur $\mathbb{R}$ et $t_{\min}$ et $t_{\max}$ deux réels tels que $t_{\min} < t_{\max}$. On note I l'intervalle $[t_{\min}\,;\, t_{\max}]$. On s'intéresse à une équation différentielle du second ordre de la forme :

$$\forall t \in I, \qquad y''(t) = f(y(t)) \tag{1}$$

où f est une fonction donnée, continue sur $\mathbb{R}$. De nombreux systèmes physiques peuvent être décrits par une équation de ce type.

On suppose connues les valeurs $y_0 = y(t_{\min})$ et $z_0 = y'(t_{\min})$. On suppose également que le système physique étudié est conservatif. Ce qui entraîne l'existence d'une quantité indépendante du temps (énergie, quantité de mouvement, ...), notée E , qui vérifie l'équation (2) où $g' = -f$.

$$\forall t \in I, \qquad \frac{1}{2}y'(t)^2 + g(y(t)) = E \tag{2}$$

3.1 Mise en forme du problème

Pour résoudre numériquement l'équation différentielle (1), on introduit la fonction $z : I \to \mathbb{R}$ définie par $\forall t \in I,\ z(t) = y'(t)$.

EXERCICE 11

1. *Montrer que l'équation* (1) *peut se mettre sous la forme d'un système différentiel du premier ordre en* $z(t)$ *et* $y(t)$, *noté* (S).

2. *Soit* n *un entier strictement supérieur à* 1 *et* $J_n = [\![0\,;\, n-1]\!]$.
   On pose $h = \frac{t_{max}-t_{min}}{n-1}$ *et* $\forall i \in J_n,\ t_i = t_{min} + ih$. *Montrer que pour tout entier* $i \in [\![0\,;\, n-2]\!]$:

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} z(t)\,\mathrm{d}t \qquad et \qquad z(t_{i+1}) = z(t_i) + \int_{t_i}^{t_{i+1}} f(y(t))\,\mathrm{d}t \tag{3}$$

3.2 Schéma d'Euler explicite

Dans le schéma d'Euler explicite, chaque terme sous le signe intégrale est remplacé par sa valeur prise en la borne inférieure.

EXERCICE 12

1. *Dans ce schéma, montrer que les équations* (3) *permettent de définir deux suites* $(y_i)_{i\in J_n}$ *et* $(z_i)_{i\in J_n}$ *où* y_i *et* z_i *sont des valeurs approchées de* $y(t_i)$ *et* $z(t_i)$. *Donner les relations de récurrence permettant de déterminer les valeurs de* y_i *et* z_i *connaissant* y_0 *et* z_0.

2. *Écrire une fonction* `euler` *qui reçoit en argument les paramètres qui semblent pertinents et qui renvoie deux listes de nombres correspondant aux valeurs associées aux suites* $(y_i)_{i\in J_n}$ *et* $(z_i)_{i\in J_n}$. *Justifier le choix des paramètres transmis à la fonction.*

3. *Pour illustrer cette méthode, on considère l'équation différentielle* $\forall t \in I,\ y''(t) = -\omega^2 y(t)$ *dans laquelle* ω *est un nombre réel.*

   (a) *Montrer qu'on peut définir une quantité* E, *indépendante du temps, vérifiant une équation de la forme* (2).

   (b) *On note* E_i *la valeur approchée de* E *à l'instant* t_i, $i \in J_n$, *calculée en utilisant les valeurs approchées de* $y(t_i)$ *et* $z(t_i)$ *obtenues à la question* 1.
   Montrer que $E_{i+1} - E_i = h^2\omega^2 E_i$.

   (c) *Qu'aurait donné un schéma numérique qui satisfait à la conservation de* E *?*

   (d) *En portant les valeurs de* y_i *et* z_i *sur l'axe des abscisses et l'axe des ordonnées respectivement, quelle serait l'allure du graphe qui respecte la conservation de* E *?*

   (e) *La mise en œuvre de la méthode d'Euler explicite génère le résultat graphique donné sur la figure* 1 *ci-dessous à gauche. Dans un système d'unités adapté, les calculs ont été menés en prenant* $y_0 = 3$, $z_0 = 0$, $t_{min} = 0$, $t_{max} = 3$, $\omega = 2\pi$ *et* $n = 100$.

```
>>> yy, zz = euler(lambda y : -(2*np.pi)**2*y, 100, 0, 3, 3, 0)
>>> print(yy[-3:], zz[-3:])
[13.933084621185467, 15.745054071756226, 17.051921471645617] [59.79499186883504,
     43.12662419634988, 24.290569049441377]
```

En quoi ce graphe confirme-t-il que le schéma numérique ne conserve pas E *? Pouvez-vous justifier son allure ?*

Figure 1

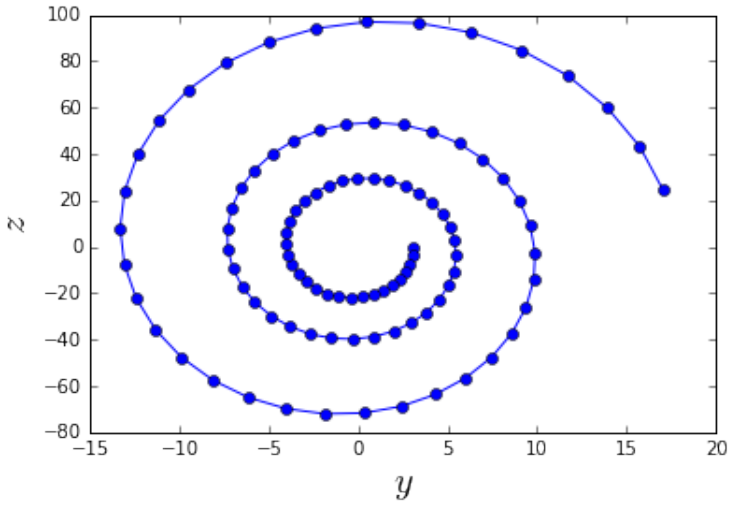
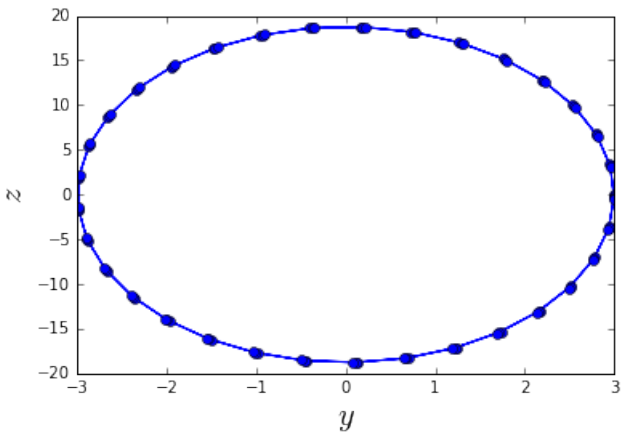


Figure 2



3.3 Schéma de Verlet

Le physicien français Loup Verlet a proposé en 1967 un schéma numérique d'intégration d'une équation de la forme (1) dans lequel, en notant $f_i = f(y_i)$ et $f_{i+1} = f(y_{i+1})$, les relations de récurrence s'écrivent :

$$y_{i+1} = y_i + hz_i + \frac{h^2}{2} f_i \qquad \text{et} \qquad z_{i+1} = z_i + \frac{h}{2}\left(f_i + f_{i+1}\right)$$

EXERCICE 13 *1. Écrire une fonction* `verlet` *qui reçoit en argument les paramètres qui semblent pertinents et qui renvoie deux listes de nombres correspondant aux valeurs associées aux suites* $(y_i)_{i\in J_n}$ *et* $(z_i)_{i\in J_n}$.

2. On reprend l'exemple de l'oscillateur harmonique (question 3. *de l'exercice* 12*) et on compare les résultats obtenus à l'aide des schémas d'Euler et de Verlet.*

(a) Montrer que dans le schéma de Verlet, on a $E_{i+1} - E_i = O(h^3)$.

(b) La mise en œuvre du schéma de Verlet avec les mêmes paramètres que ceux utilisés à la question 3. *de l'exercice* 12 *donne le schéma de la figure* 2 *ci-dessus. Interpréter l'allure de ce graphe.*

```
>>> yy, zz = verlet(lambda y : -(2*np.pi)**2*y, 100, 0, 3, 3, 0)
>>> print(yy[-3:], zz[-3:])
[2.8152395363671467, 2.9606719372849324, 2.998774087394971] [6.483226837453586,
     3.028320091959096, -0.5363692846006165]
```

(c) Que peut-on conclure sur le schéma de Verlet ?

4 Pour ceux qui s'ennuient

EXERCICE 14 *On s'intéresse à l'équation* $y'' = y' + 6y$*, avec les conditions* $y(0) = -1$ *et* $y'(0) = 2$.

1. Éteindre l'écran ; trouver l'unique solution de ce problème de Cauchy.

2. Allumer l'écran ! Résoudre numériquement cette équation et représenter le résultat sur l'intervalle $[0,10]$.

```
def f4(t, y):
    [a,b] = y
    return np.array([b, b+6*a])

t = np.linspace(0, 10, 1001)
Y = euler(f4, 0, np.array([-1,2]), 10, 1000)
y = np.array(Y)[:, 0]

plt.plot(t, y)
plt.savefig('ordre2-converge.pdf')
plt.show()
plt.clf()
```

3. Recommencer sur $[0,20]$ *avec le même pas.*

4. (difficile) Prendre un air surpris... réfléchir... puis tenter une explication en prenant un air intelligent !

EXERCICE 15 *Résoudre numériquement sur* $[0,3]$ *l'équation* $y' = -3y$ *avec la condition* $y(0) = 1$. *On prendra pour* n *les valeurs* $\{3,4,5,30\}$ *pour avoir des pas de* 1, 0.75, 0.6 *et* 0.1.

Quand on représente plusieurs graphiques sur le même dessin, on peut souhaiter différencier les traits. On peut par exemple modifier l'épaisseur (de 1 par défaut) ou le style (pointillé, continu...)

```
plot(t, y1, linewidth=3)         # linewidth == 1 par défaut
plot(t, y2, '-')                 # trait plein
plot(t, y3, '--')                # pointillés
plot(t, y4, '-.', linewidth=2)   # devinez !
plot(t, y5, '--', color = 'red') # ...
```

EXERCICE 16 *Le schéma d'Euler* implicite, *ou* rétrograde *consiste à remplacer la relation :*

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

par :

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

Bien entendu, cette relation ne donne pas explicitement y_{k+1} en fonction de y_k (d'où son nom) et nécessite donc à chaque étape une résolution d'équation de la forme $\Phi(y_{k+1}) = 0$. Puisque y_{k+1} est censé être proche de y_k, on dispose d'une bonne première approximation y_k qu'on passe comme paramètre à la fonction de bibliothèque `fsolve`[3].

1. *Programmer la méthode d'Euler implicite.*

   Indication : pour demander à Python une (approximation d'une) solution de l'équation implicite $y = y_k + hF(t_{k+1}, y)$, on pourra exécuter : `fsolve(lambda y:y-yk-h*F(tkp1,y), yk)`, avec `fsolve` issue de la bibliothèque `scipy.optimize`

2. *Tester cette méthode sur l'équation $y' = -3y$ sur $[0,3]$, avec la condition initiale $y(0) = 1$. Tester des pas entre 0.1 et 2.*

3. *Comparer sur ce même exemple avec la méthode d'Euler usuelle (explicite) ; voir l'exercice 15.*

5 Pour information...

Les jolis dessins de cet énoncé ont été faits avec `matplotlib`. Par exemple, voici le tracé les trapèzes :

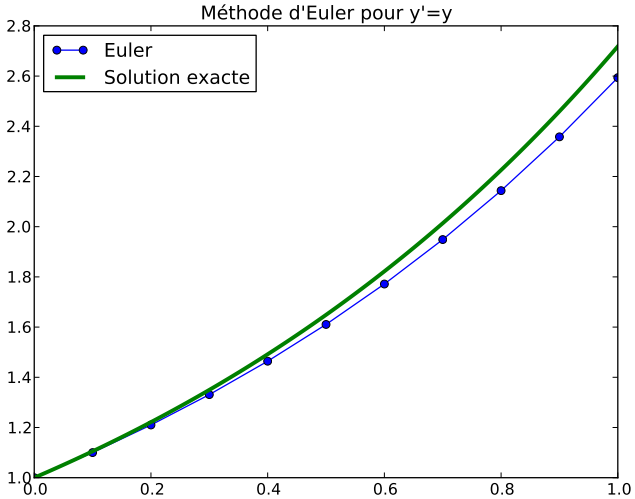
```
def un_trapeze(g,d):   # bords gauche et droit
    plt.fill([g,d,d,g] , [0,0,f(d),f(g)], fill = False, hatch = '/')

sigma = np.linspace(a,b,n+1) # création de la subdivision régulière
for i in range(n):
    un_trapeze(sigma[i],sigma[i+1])
```

Pour les paraboles, on résout d'abord un système linéaire pour trouver le polynôme interpolant la fonction aux trois points.

```
def une_parabole(g,d): # gauche et droite
    m = (g+d)/2 # milieu; on veut que P=a2*X^2+a1*X+a0 interpole f aux trois points
    [a2,a1,a0] = np.linalg.solve([[g**2,g,1],[m**2,m,1],[d**2,d,1]],[f(g),f(m),f(d)])
    xl = np.linspace(g,d,100)
    yl = a2*xl**2+a1*xl+a0
    plt.fill([d,g]+list(xl),[0,0]+list(yl), fill = False, hatch = '/')
```

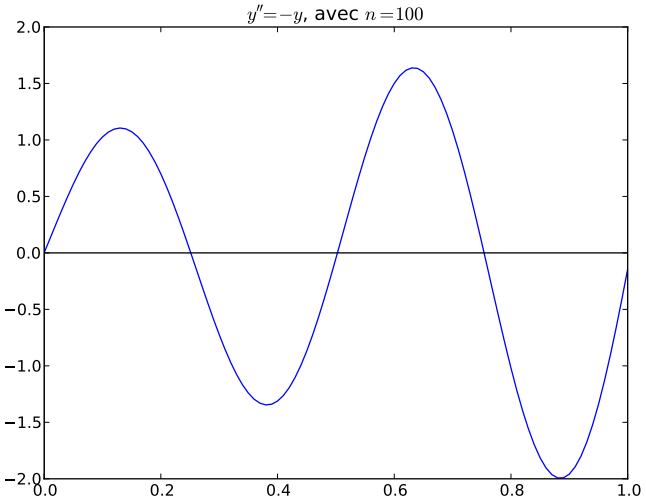
6 Quelques graphiques

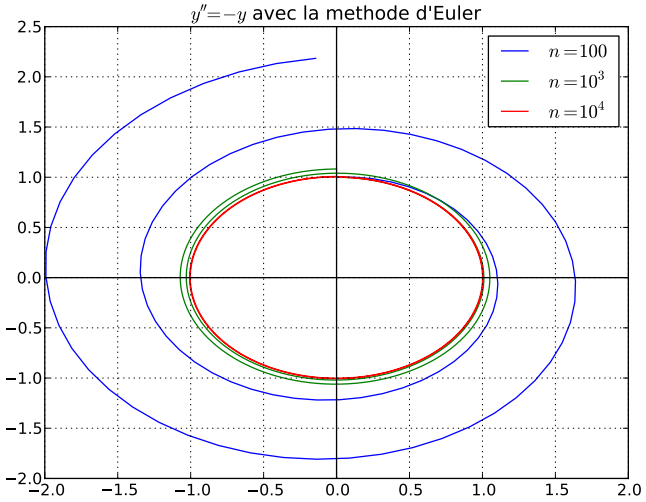
- Exercice 8 :



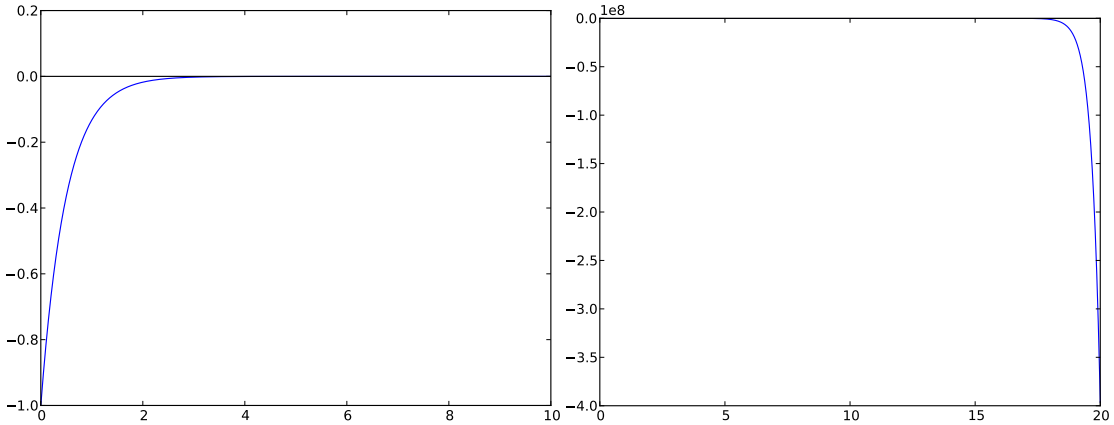
3. Documentation : `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fsolve.html`

- Exercice 10 :



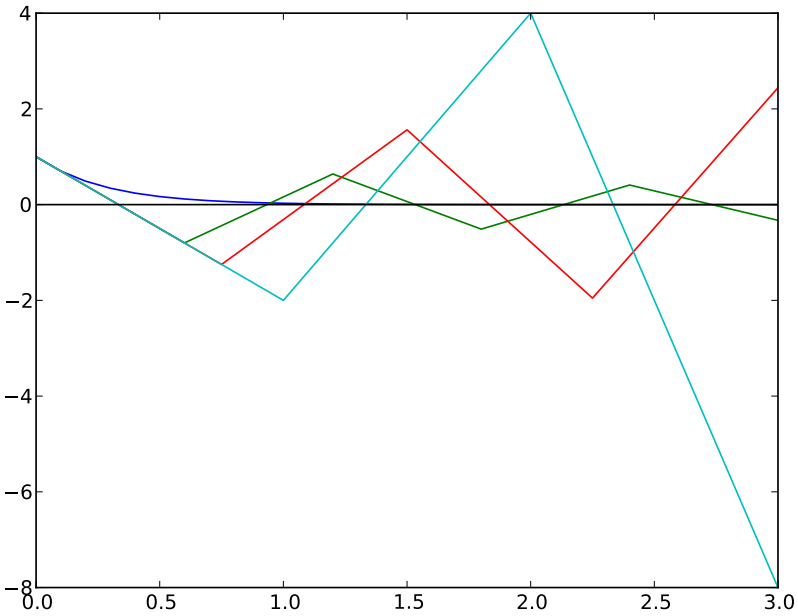


- Exercice 14 :



Ça converge? Ha non!

- Exercice 15 :



- Exercice 16 :