TP 4 : Flottants. Introduction aux modules classiques: Numpy, Matplotlib et Scipy

1 Petits exercices flottants

Commencer par importer toutes les fonctions du module `math`, en tapant `import matplotlib.pyplot as plt` en tête de votre script.

Exercice 1. *Approximation de* $\sqrt{2}$. La suite définie par $u_0 = 1$ et $u_{n+1} = \frac{1}{2}\left(u_n + \frac{2}{u_n}\right)$ converge (rapidement) vers $\sqrt{2}$. Écrire un script affichant à l'écran les approximations successives de $\sqrt{2}$ obtenue à l'aide de cette suite (un petit nombre suffit pour atteindre la précision maximale par défaut, moins de dix étapes sont nécessaires). Vérifier que même si mathématiquement, on a jamais $u_n = u_{n+1}$, ceci se produit avec des flottants à partir d'un certain rang.

Exercice 2. *Approximation de la dérivée.*

1. On sait que $f'(x) \simeq \frac{f(x+h)-f(x)}{h}$ pour h assez petit et f une fonction dérivable en x. Définir une fonction `derivee_exp(x,h)` calculant une approximation de $\exp'(x)$ en suivant cette méthode.

2. On sait que $\exp'(1) = \exp(1) = \mathrm{e}$. Tester votre fonction pour $h \in \{10^{-2}, 10^{-5}, 10^{-10}, 10^{-15}, 10^{-20}\}$.

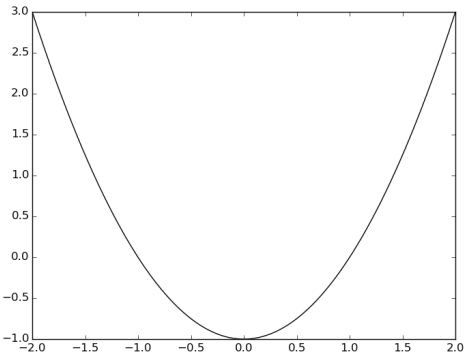
Remarque : même si mathématiquement, il est intéressant de prendre h le plus petit possible, ce n'est pas le cas en informatique : comme on ne dispose sur 64 bits que de 52 bits de mantisse, et que $2^{-52} \simeq 10^{-16}$, lorsque h s'approche de cette limite il se produit une grosse perte de précision. Dans le cas extrême $h = 10^{-20}$, $\exp(x+h)$ est arrondi à la même valeur que $\exp(x)$, d'où le résultat !

2 Tracé de graphes : utilisation de matplotlib

Pour tracer des courbes, on utilisera le sous-module `pyplot` du module `matplotlib`. Voici une manière de tracer le graphe de la fonction $x \mapsto x^2 - 1$ sur l'intervalle $[-2, 2]$ (le code se trouve sur la page web !)

```
import matplotlib.pyplot as plt #importation classique du module matplotlib.pyplot sous l'alias plt
n=1000 #nombre de points
a=-2
b=2
X=[a+(b-a)*k/(n-1) for k in range(n)] #des valeurs régulièrement espacées.
Y=[x*x-1 for x in X]
plt.plot(X,Y) #on relie les points (X[i],Y[i]) par des segments: c'est une bonne approximation du graphe.
plt.show() #et on l'affiche !
```

On obtient ainsi le graphe de la figure suivante.



Le fonctionnement de Matplotlib est simple : on crée deux listes de même taille n, l'une pour les abscisses, l'autre pour les ordonnées. Pour tout $i \in [\![0, n-2]\!]$, les points `(X[i],Y[i])` et `(X[i+1],Y[i+1])` sont alors reliés par des segments. Si le nombre de points est suffisamment important, on obtient un tracé correct.

Exercice 3. *Tracé des fonctions sinus et cosinus.* On rappelle que les fonctions `cos` et `sin` se trouvent dans le module `math`, ainsi que la constante π (`pi`). Il est facile de tracer plusieurs courbes sur un même graphique : il suffit d'afficher le graphe (avec `show()`) qu'une fois toutes les courbes tracées (avec `plot`).

1. En vous inspirant de l'exemple précédent, tracer sur le même graphe des fonctions sinus et cosinus sur l'intervalle $[-\pi, \pi]$. On créera une liste d'abscisses régulièrement espacées dans $[-\pi, \pi]$ en suivant le même mécanisme que le code ci-dessus, avec $a = -\pi$ et $b = \pi$.
2. Si vous avez tracé `cos` avant `sin`, ajoutez `plt.legend(('cos','sin'))` avant d'afficher le graphe, vous aurez ainsi une légende.
3. De même, utilisez `plt.title` (avec une chaîne de caractères de votre choix en argument) pour mettre un titre.

Il est hors de question de connaître toutes ces possibilités par cœur. Pour faire des graphiques évolués [1], jetez un œil à la documentation de `matplotlib.pyplot` [2].

3 Utilisation du module Numpy

Le module Numpy est un module que l'on utilisera souvent au deuxième semestre : il permet de faire de l'analyse numérique. Lorsqu'on utilise ce module, on manipule des données chiffrées sous une forme plus spécifique que les listes Python : les tableaux (vecteurs ou matrices) Numpy. On manipulera également les images à l'aide de ce module. Pour le moment on se restreint à des tableaux Numpy à une dimension.

```
>>> import numpy as np #importation classique de numpy avec l'alias np
>>> X=np.array([0,1,2]) #conversion d'une liste en tableau numpy
>>> print(X)
array([0, 1, 2])
```

La fonction `linspace` du module permet de produire directement des tableaux de flottants régulièrement espacés dans un intervalle donné. Voici une autre manière de tracer le graphe de la fonction $x \mapsto x^2 - 1$.

```
import numpy as np
X=np.linspace(-2,2,1000) #1000 points régulièrement espacés dans [-2,2]
plt.plot(X,[x*x-1 for x in X])
plt.show()
```

Un intérêt des tableaux Numpy est que les opérations `+` et `*` sur des tableaux Numpy de même taille s'appliquent terme à terme, contrairement aux listes classiques. On peut même rajouter une constante à tous les éléments d'un tableau. Ainsi la troisième ligne du script précédent peut se réécrire :

```
import numpy as np
X=np.linspace(-2,2,1000) #1000 points régulièrement espacés dans [-2,2]
plt.plot(X,X*X-1)
plt.show()
```

Exercice 4. *Quelques exemples issus des mathématiques.* Tracer les graphes des fonctions suivantes :

1. $x \mapsto x^{-x}$ sur l'intervalle $[0.01, 5]$.
2. $x \mapsto \frac{x^3}{(x-1)^2}$ sur l'intervalle $[-5, 5]$, et tracer sur le même graphique l'asymptote oblique $x \mapsto x + 2$. Comme la fonction n'est pas définie en 1, on tracera séparemment deux morceaux, l'un sur $[-5, 0.8]$, l'autre sur $[1.5, 5]$. Vérifier visuellement que 0 est un point d'inflexion.

Exercice 5. *De jolis graphiques.* Le code suivant permet de tracer et d'afficher un cercle de centre $(0, 0)$ et de rayon 1 (en effet, on relie des points de la forme $(\cos(\theta), \sin(\theta))$ avec θ prenant un ensemble de valeurs régulièrement espacés sur un intervalle de longueur 2π).

1. Pas aujourd'hui !
2. `http://matplotlib.org/users/pyplot_tutorial.html`

```
plt.axis("equal")
Theta=np.linspace(-np.pi,np.pi,1000)
X=[cos(x) for x in Theta]
Y=[sin(x) for x in Theta]
plt.plot(X,Y)
plt.show()
```

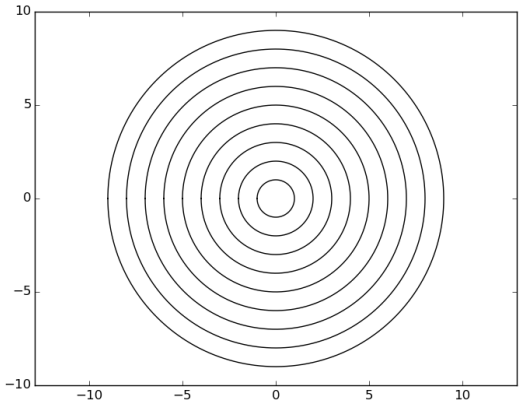
Remarquez l'option `plt.axis("equal")` qui forcent les axes à avoir des graduations égales, ce qui permet d'obtenir visuellement un cercle et non une ellipse.

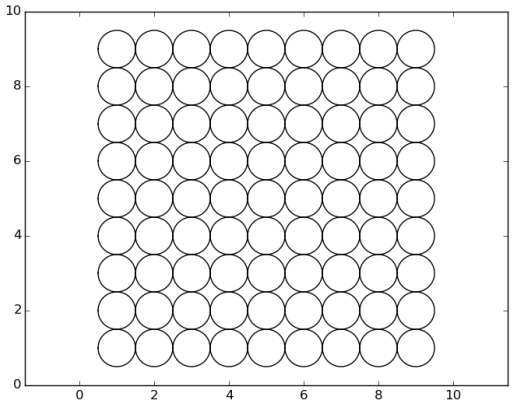
1. Une représentation paramétrique du cercle de centre (x_0, y_0) et de rayon r est donnée par $\begin{cases} x(\theta) = x_0 + r\cos\theta \\ y(\theta) = y_0 + r\sin\theta \end{cases}$ avec $\theta \in [-\pi, \pi]$. Écrire une fonction `cercle(x0,y0,r)` permettant de tracer le cercle de centre (x_0, y_0) et de rayon r. *Attention :* dans la fonction, on *tracera* uniquement le cercle, on ne l'affichera pas. On peut ainsi utiliser la fonction comme suit :

```
cercle(1,2,3) #trace le cercle de centre (1,2) et de rayon 3.
plt.show() #l'affiche.
```

Votre fonction agissant uniquement *par effets*, `return` est inutile. Essentiellement, on créera des listes de points comme ci-dessus, et on fera appel à `plot`.

2. En utilisant la fonction précédente (plusieurs fois !), débrouillez vous pour réaliser des figures comme celles ci-dessous. On fera évidemment usage de boucles `for` : une seule pour la première, deux boucles imbriquées pour la deuxième.





Dans la première figure, les cercles ont des rayons 1,2,3,... La deuxième est constituée de cercles tangents de même rayon 1/2.

4 Utilisation du module Scipy pour résoudre des équations différentielles

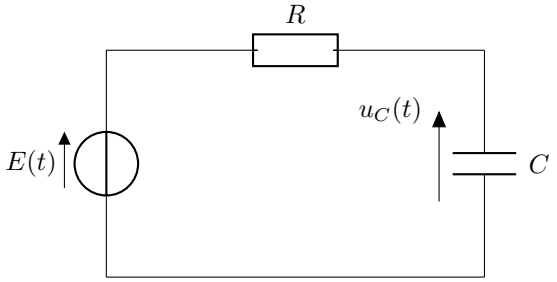


FIGURE 1: Circuit RC - Charge d'un condensateur

Le module Scipy permet de faire de l'analyse numérique : calculs d'intégrales, résolutions d'équations numériques ou différentielles... On verra en cours des algorithmes (basiques) permettant de résoudre ces problèmes, pour le moment on se contente d'utiliser des méthodes (évoluées) intégrées à Scipy. On travaillera dans cette section avec le problème d'un circuit RC classique, dont on cherche la tension u_C aux bornes du condensateur, en fonction des paramètres R et C ainsi que de la tension E aux bornes du générateur.

L'équation différentielle vérifiée par la tension u_C est la suivante :

$$\mathrm{RC}\frac{\mathrm{d}\,u_C}{\mathrm{d}t}(t) + u_C(t) = E(t)$$

En posant $\tau = \mathrm{RC}$, l'équation se réécrit sous forme *autonome* (le coefficient devant la dérivée est 1) comme :

$$\frac{\mathrm{d}\,u_C}{\mathrm{d}t}(t) + \frac{1}{\tau}u_C(t) = \frac{1}{\tau}E(t)$$

Scipy est capable de résoudre des équations différentielles scalaires (ou vectorielles, comme on verra plus tard) d'ordre 1 données sous la forme $y'(t) = f(y(t), t)$, c'est donc sous cette forme qu'il faut lui donner.

Pour commencer on affecte R, C et E (la fonction E est pour le moment prise constante égale à 5V, mais on changera après), et on calcule `tau` :

```
R=500 #500 Ohm
C=10**-6 #10^-6 Farad
tau=R*C #temps caractéristique

def E(t):
    return 5
```

Voici une définition de la fonction f utilisable par Scipy :

```
def f(y,t):
    return 1/tau*(E(t)-y)
```

Voici comment résoudre l'équation différentielle avec Scipy. La fonction `odeint` du sous-module `scipy.integrate` résout les équations différentielles. Elle prend en paramètres :

— une fonction f sous la même forme que celle donnée ;

— une valeur de la fonction y cherchée au temps t_0 (on prendra ici 0).

— un tableau de temps `T`.

La réponse fournie par Scipy est un tableau Numpy, contenant des approximations $y(t_i)$ des valeurs de la solution à l'équation différentielle $y'(t) = f(y(t), t)$ aux temps t_i donnés par le tableau `T`, (`t0` est le premier élément de `T`). Calculons la tension aux bornes du condensateur sur un intervalle de temps $[0, \alpha]$ en supposant $u_C(0) = 0$.

```
import scipy.integrate as sci
T=np.linspace(0,0.03,1000) #entre 0 et 30 ms !
solution=sci.odeint(f,0,T)
```

Voici comment tracer la tension aux bornes du générateur et celle aux bornes condensateur, et les tracer (voir figure 2).

```
plt.plot(T,[E(t) for t in T], "k-", label="générateur") #k pour blacK, - pour un trait
plt.plot(T, solution, "k--", label="condensateur") #-- pour des pointillés.
plt.legend(loc="lower right") #positionnement de la légende en bas à droite
plt.show()
```

Exercice 6. *Temps de réponse à 95%.* À l'aide d'un script, calculer le temps de réponse à 95%, c'est-à-dire le plus petit temps t_i du tableau `T` tel que la tension aux bornes du condensateur soit supérieure à $0.95 \times 5\mathrm{V}$. Vérifier que $t_i \simeq 3\tau$.

Exercice 7. *Changement de fonction E.* On va reprendre le tracé précédent avec une fonction E dépendant du temps.

1. définir une fonction E en créneaux ± 5V, de période 10ms et tracer à nouveau la tension aux bornes du générateur et celles aux bornes du condensateur.

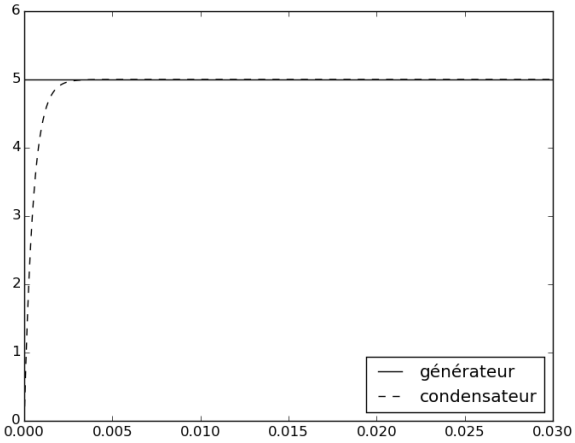
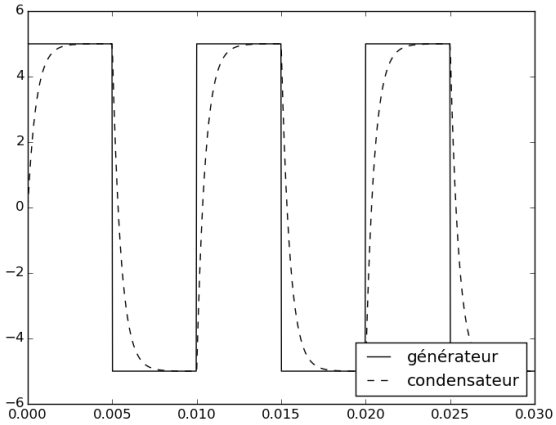
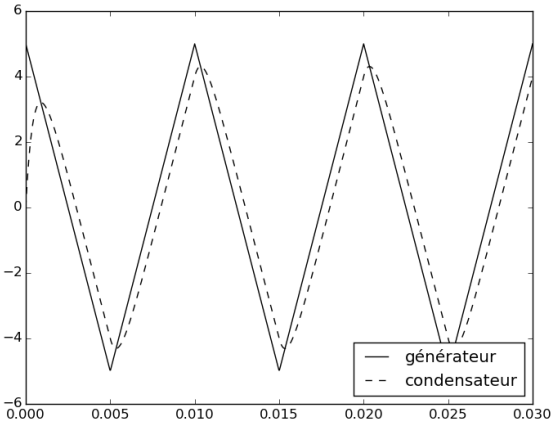


FIGURE 2: Tension aux bornes du condensateur avec Scipy

2. Faire de même avec une fonction E « en triangles ».

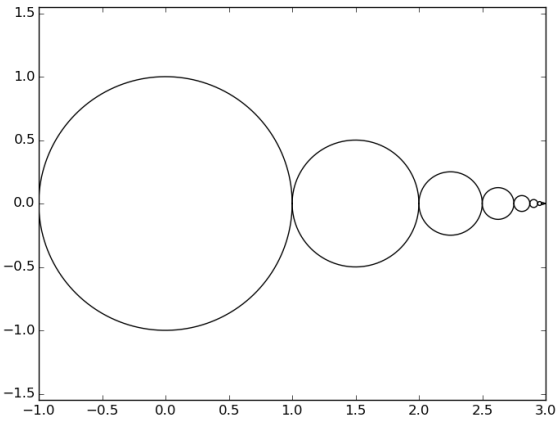
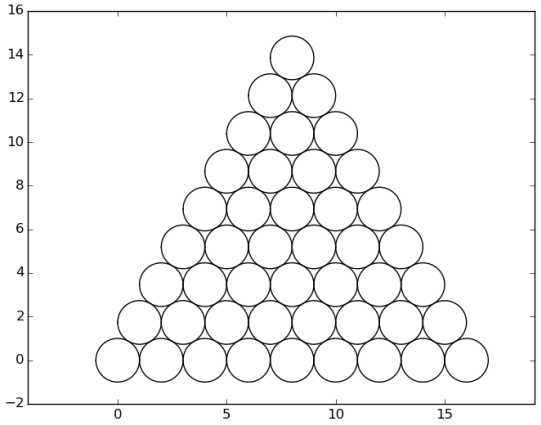
Rappel : le modulo `%` fonctionne aussi avec des flottants ! Le résultat attendu est tracé pour vous :





5 Exercices en plus

Exercice 8. *D'autres cercles*. Reproduisez les cercles des figures ci-dessous.

Exercice 9. *Des fractales (très difficile).* Essayez de dessiner ceci (utiliser des listes dans les deux cas) :