Chapitre 10

Utilisation de modules sous python. Tracé avec numpy et matplotlib.pyplot

10.1 Utilisation d'un module

10.1.1 Importer un module

• Un module est un fichier ayant pour extension `.py` contenant des définitions de constantes et fonctions. Importer un module permet d'utiliser ses constantes et fonctions.

Nous avons déjà utilisé les modules `math, fractions, random, time`.

• Pour importer un module :

1 Première méthode : A l'aide de l'instruction `from`
Pour importer une fonction ou constante :

```
>>> from math import sqrt
>>> sqrt(2)
1.4142135623730951
```

```
>>> from math import cos, pi
>>> cos(pi/4)
0.7071067811865476
```

Pour importer toute la bibliothèque :

```
>>> from math import *
>>> sin(pi/4)
0.7071067811865476
```

2 Deuxième méthode : Sans l'instruction `from`.

```
>>> import math
>>> math.sqrt(2)
1.4142135623730951
>>> math.cos(math.pi/4)
0.7071067811865476
```

Fonctions et constantes doivent être précédées d'un préfixe : le nom du module suivie d'un point `'.'`.

On peut définir un alias à l'aide de l'instruction `as` :

```
>>> import math as m        # Création d'un alias m
>>> m.sqrt(2)            # Utiliser le préfixe m.
1.4142135623730951
>>> m.cos(m.pi/4)
0.7071067811865476
```

10.1.2 Exemple : le module random

Le module `random` permet de générer des nombres pseudo-aléatoires.

Voici quelques fonctions fournies par le module `random` :

`randrange(a,b,k)`	Choisit un entier aléatoirement dans `range(a,b,k)`
`randint(a,b)`	Choisit un entier aléatoirement dans $[[a,b]]$
`choice(List)`	Choisit un entier aléatoirement dans la liste `List`
`random()`	Choisit un float aléatoirement dans $[0,1[$
`uniform(a,b)`	Choisit un float aléatoirement dans $[a,b[$

Ici aléatoirement signifie selon une loi uniforme (quasiment).

Exemple : Que fait le programme suivant ?

```
import random as rand
def de6(n):
    tirs = [0, 0, 0, 0, 0, 0]
    for i in range(n):
        t = rand.randint(1,6)
        tirs[t-1] += 1
    for j in range(6):
        tirs[j] = tirs[j]*100.0/n
        tirs[j] = str(round(tirs[j],2)) + '%'
    return tirs
```

La fonction `round(x,n)` pour `x` un flottant et `n` un entier renvoie l'arrondi de `x` à `n` chiffres après la virgule.

Réponse : le programme simule `n` lancers d'un dé 6, et compte le nombre de fois où chaque face apparaît pour finalement retourner le pourcentage d'apparition de chaque face.

Maintenant vérifions le théorème des grands nombres : pour un grand nombre de tirs les fréquences d'apparition de chaque face devraient tendre vers leur probabilité de tir.

```
>>> de6(10)
['30.00%', '0.00%', '20.00%', '20.00%', '10.00%', '20.00%']
>>> de6(100)
['10.00%', '22.00%', '19.00%', '20.00%', '15.00%', '14.00%']
>>> de6(1000)
['16.00%', '15.40%', '16.10%', '16.90%', '18.50%', '17.10%']
>>> de6(10000)
['16.45%', '16.66%', '16.50%', '16.84%', '17.37%', '16.18%']
>>> de6(100000)
['16.57%', '16.67%', '16.80%', '16.62%', '16.66%', '16.68%']
>>> de6(1000000)
['16.65%', '16.69%', '16.66%', '16.64%', '16.63%', '16.74%']
>>> de6(10000000)
['16.66%', '16.66%', '16.65%', '16.66%', '16.69%', '16.68%']
```

10.2 numpy et matplotlib

10.2.1 Modules scientifiques

Les modules à utiliser pour le tracé de courbes sous `python` :

1. `numpy` : Outils pour créer, manipuler, et appliquer de nombreuses opérations sur des tableaux de nombres. `http://docs.scipy.org/doc/numpy/reference/` (en anglais).
2. `matplotlib.pyplot` : Permet le tracé de graphes de fonctions. `http://matplotlib.org/users/pyplot_tutorial.html` (en anglais).

10.2.2 Le module numpy

• Le module `numpy` permet de créer, de manipuler, des tableaux homogènes non-redimensionnables de nombres et de leur appliquer des opérations mathématiques courantes.

• **La fonction** `array()` permet de créer un tableau, ou `array`, à partir d'un tableau `python` c'est à dire d'une liste de listes ou tuples de nombres :

```
>>> import numpy as np
>>> A = np.array([1, 2, 3, 4])
>>> A
array([1, 2, 3, 4])
>>> A[0] , A[-1]
 (1, 4)
```

• Toutes les opérations sur les listes et séquences `python`, en dehors de celles qui redimensionnent sont possible sur un tableau `numpy`.

• **La fonction** `arange()` crée un tableau de façon assez analogue à la fonction `range()`, à ceci près que les coefficients ne sont pas forcément entiers :

```
>>> v = np.arange(0, 1.5, 0.5)
>>> v
array([ 0., 0.5, 1. ])
>>> 2*v
array([ 0., 1., 2. ])
>>> (2*v) ** 2 + 100
array([ 100., 101., 104. ])
```

On peut appliquer sur les tableaux les opérations mathématiques, comprises terme à terme.

• **La fonction** `linspace(a, b, n)` crée le tableau des n valeurs régulièrement espacées prises entre a et b, tous deux inclus.

C'est à dire le tableau de la subdivision régulière de l'intervalle $[a, b]$ par n points (ou par $n-1$ segments). Exemple :

```
>>> v = np.linspace(0,1,10)
>>> v
array([ 0. , 0.11111111, 0.22222222, 0.33333333, 0.44444444,
0.55555556, 0.66666667, 0.77777778, 0.88888889, 1. ])
```

• `numpy` contient aussi toutes les fonctions mathématiques (aussi présentes dans `math`). **On peut appliquer sur les tableaux les fonctions de `numpy`, comprises terme à terme.** :

```
>>> np.cos(v)
array([ 1. , 0.99383351, 0.97541009, 0.94495695, 0.90284967,
0.84960756, 0.78588726, 0.71247462, 0.63027505, 0.54030231])
```

10.2.3 Commandes de création de tableaux sous `numpy`

• Tableaux (uni-dimensionnels) sous `numpy` :

`array(liste)`	convertit en tableau une liste ou séquence
`arange(a,b,k)`	crée le tableau de tous les `a+k.N` entre `a` (inclu) et `b` (exclu). L'écart entre deux points est `k`
`linspace(a,b,n)`	crée le tableau des `n` valeurs régulièrement espacées entre `a` et `b` (inclus). L'écart entre 2 points est `(b-a)/(n-1)`
`zeros(p)`	crée un tableau de taille `p` rempli de zéros
`empty(p)`	crée un tableau de taille `p` vide
`mean()`	retourne la moyenne d'un tableau
`size()`	retourne le nombre d'éléments d'un tableau

`len()` retourne aussi le nombre d'éléments d'un tableau unidimensionnel.

10.2.4 Tracé avec matplotlib.pyplot

• Pour le simple tracé de courbes nous n'utiliserons que le sous-module `pyplot`, importé, avec alias, à l'aide de la commande :

```
>>> import matplotlib.pyplot as plt
```

cf. documentation à : `http://www.matplotlib.org`.

• Les fonctions essentielles de `pyplot` sont :

1. `plot()` pour le tracé de points, de courbes, et
2. `show()` pour afficher le graphique créé.

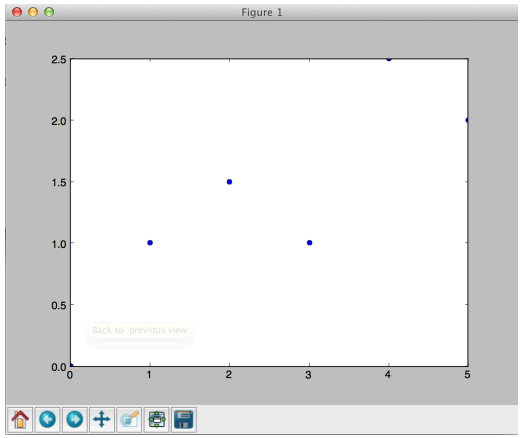
• Utiliser `plot()` avec :

1. en 1^{er} argument la liste des abscisses,
2. en 2^{eme} argument la liste des ordonnées,
3. en 3^{eme} argument (optionnel) le motif des points :
   (a) `'.'` pour un petit point,
   (b) `'o'` pour un gros point,
   (c) `'+'` pour une croix,
   (d) `'*'` pour une étoile,
   (e) `'-'` points reliés par des segments
   (f) `'--'` points reliés par des segments en pointillés
   (g) `'-o'` gros points reliés par des segments (on peut combiner les options)
   (h) `'b'`, `'r'`, `'g'`, `'y'` pour de la couleur (bleu, rouge, vert, jaune, etc...)
   (i) cf. `http://matplotlib.org/api/pyplot_api.html\#matplotlib.pyplot.plot`.

• Exemple : pour le tracé d'un nuage de points

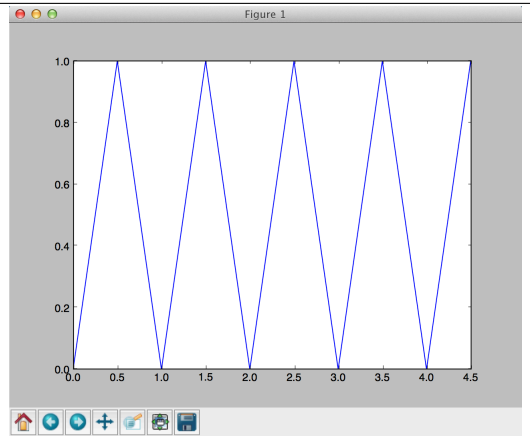
```
>>> import matplotlib.pyplot as plt
>>> abs = [0, 1, 2, 3, 4, 5]
>>> ord = [0, 1, 1.5, 1, 2.5, 2]
>>> plt.plot(abs, ord, 'o')
>>> plt.show()
```

produit un graphique (au format .png) :



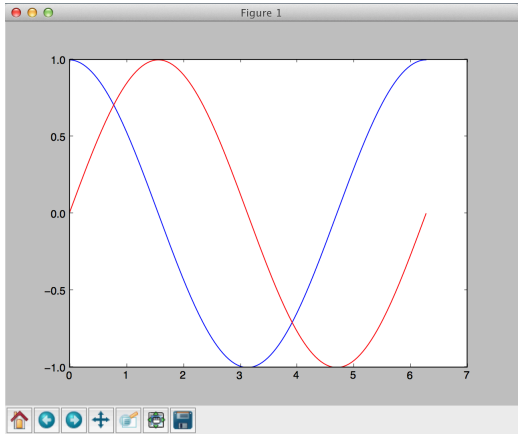
• Exemple : pour le tracé d'une ligne brisée :

```
>>> import matplotlib.pyplot as plt
>>> abs = [n/2 for n in range(10)]
>>> ord = [n % 2 for n in range(10)]
>>> plt.plot(abs,ord,'-b')
>>> plt.show()
```



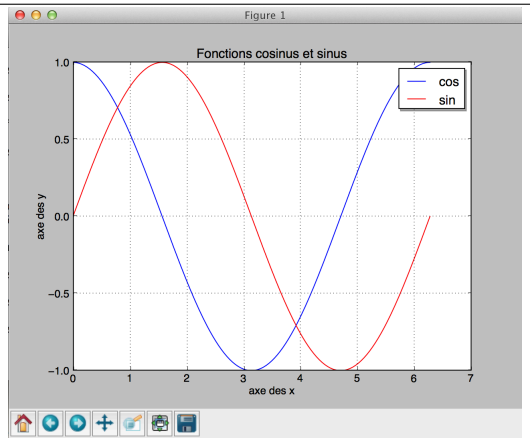
- Exemple : pour le tracé de courbes représentatives de fonctions réelles :

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> X = np.linspace(0, 2*np.pi, 1000) # abscisses
>>> Ycos = np.cos(X)    # image par cos
>>> Ysin = np.sin(X)    # image par sin
>>> plt.plot(X,Ycos,'b')    # tracé en bleu
>>> plt.plot(X,Ysin,'r')     # tracé en rouge
>>> plt.show()
```



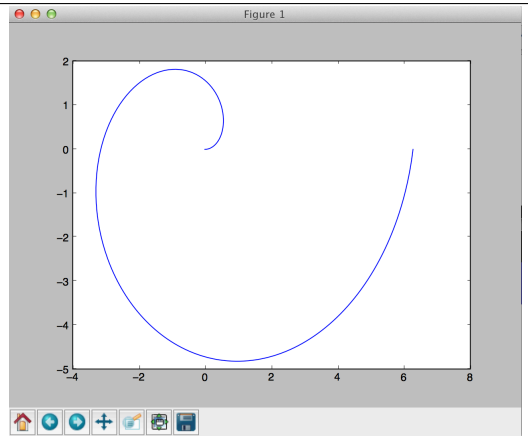
- On améliore le tracé en remplissant quelques options :

```
>>> plt.plot(X, Ycos, 'b', X, Ysin, 'r') # Tracé simultané
>>> plt.grid(True)    # Affiche une grille
>>> plt.legend(('cos','sin'), 'upper right')    # Légende
>>> plt.xlabel('axe des x')    # Label axe des abscisses
>>> plt.ylabel('axe des y')    # Label axe des ordonnées
>>> plt.title('Fonctions cosinus et sinus')    # Titre
>>> plt.savefig('Trace')    # sauvegarde fichier Trace.png
>>> plt.show()
```



- On peut tout aussi bien tracer des courbes paramétrées.

```
>>> T = np.linspace(0,2*np.pi,255)    # paramètre t
>>> X = T * np.cos(T)    # x(t) = t.cos(t)
>>> Y = T * np.sin(T)    # y(t) = t.sin(t)
>>> plt.plot(X,Y,'b') # Tracé de la courbe {(x(t),y(t))}
>>> plt.show()
```



10.3 Exemple : Méthode de Monte Carlo

• Les méthodes de simulation de Monte Carlo permettent le calcul approché de valeurs numériques par des procédés aléatoires.

• **Illustration : Approximation de π** • On tire au sort N points dans le domaine carré $[-1;1]\times[-1;1]$ selon une loi uniforme.

La probabilité pour un point d'être choisi dans le disque unitaire est égale au quotient de l'aire du disque unitaire par l'aire du domaine carré, soit :

$$\boxed{\frac{\pi}{4}}$$

Ainsi d'après le théorème des grands nombres, si l'on tire un grand nombre de points, c. à d. si N est suffisamment grand, la proportion des points tirés se trouvant dans le disque unitaire, c. à d. avec $x^2+y^2\leqslant 1$, est presque surement proche de $\pi/4$.

• **Algorithme :**

Choisir N points (x,y) dans le domaine [-1,1]×[-1,1] selon une loi uniforme.

Calculer la proportion de points choisis (x,y) vérifiant $x^2+y^2\leqslant 1$.

En multipliant par 4 ce résultat on obtient une approximation de π.

```
import matplotlib.pyplot as plt        # pyplot
import numpy as np                     # numpy
import random as rand                  # random
def monpi(N):
    X = [rand.uniform(-1,1) for i in range(N)]
    Y = [rand.uniform(-1,1) for i in range(N)]
    oui = non = 0    # Calcul de proportion dans disque
    for i in range(N):
        if X[i]**2 + Y[i]**2 <= 1:
            oui += 1
        else:
            non += 1
    print('approximation de pi trouvée :')
    print(4 * oui / (oui+non))
    print(np.pi)
```

```
    # Tracé du graphique
    plt.plot([-1,1,1,-1,-1],[-1,-1,1,1,-1], 'b-') # carré
    T = np.linspace(0,2*np.pi,1000)
    Xdisk = np.cos(T)
    Ydisk = np.sin(T)
    plt.plot(Xdisk,Ydisk,'b')    # tracé du disque
    for i in range(N):    # tracé du nuage de point
    if X[i]**2 + Y[i]**2 <= 1:
        plt.plot([X[i]],[Y[i]], 'g+') # vert si dedans
    else:
        plt.plot([X[i]],[Y[i]], 'r+') # rouge si dehors
    plt.axis('equal')
    plt.show()    # affichage du graphique
```

```
>>> monpi(10000)
approximation de pi trouvée :
3.152
3.14159265359
```

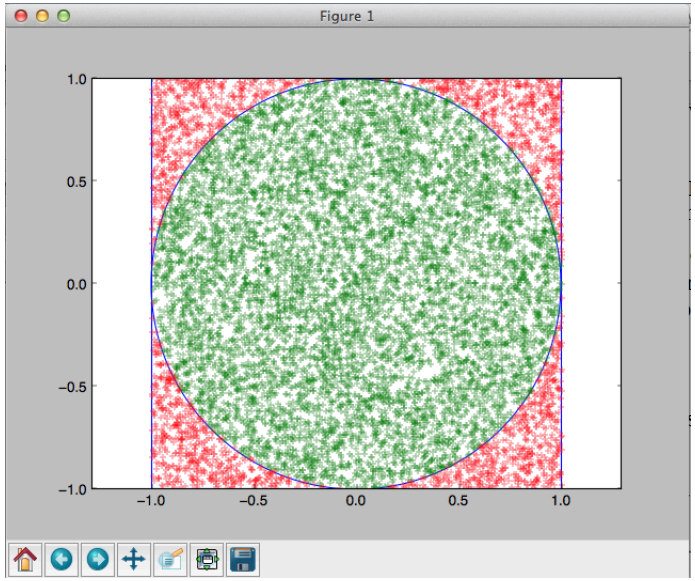


Figure 1
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0