Feuille de travaux pratiques # 1

1 Python, numpy, scipy, et matplotlib

Les TPs se font en `python`, à l'aide des modules `numpy`, `scipy`, et `matplotlib`. On utilisera le systeme de feuilles de calcul `jupyter`. Tous ces logiciels sont des logiciels libres, disponibles sous Windows, Mac, et autres.

Pour lancer `jupyter`, ouvrir un terminal, puis taper la commande `jupyter notebook` ou bien `jupyter lab`.

Une documentation bien faite sur python, numpy, scipy et matplotlib se trouve sur `python-simple.com`. On pourra aussi consulter le livre de Vincent Vigon *python proba stat.*

Hello World

Il faut importer numpy avant de l'utiliser (1ere ligne: on declare np comme raccourci pour numpy). On importe aussi les sous-modules linalg (algèbre linéaire) et stats de scipy (raccourcis: linalg, stats) et la librairie graphique matplotlib (raccourci: plt). Ici, création d'une matrice `M`. Attention, la numerotation des lignes et colonnes commence à 0, c'est le standard python.

```
import numpy as np
import scipy.linalg as linalg
import scipy.stats as stats
import matplotlib.pyplot as plt

print("Hello World!")
#Ceci est un commentaire

M=np.array([[1,2,3],[4,5,6]])
print(M[0,0])    # affiche 1
print(M[1,2])    # affiche 6
```

En python, la délimitation des blocs se fait par l'indentation (utiliser la touche "Tab" pour indenter, "shift-Tab" pour desindenter). Les blocs commencent souvent par " : " comme ci-dessous pour la boucle `for`, la definition d'une fonction, les tests `if` / `then` / `else`. Remarquer l'absence du mot "then" dans le test `if`.

```
#affiche les carres des entiers de 0 a 4
for i in range(5):
    j=i**2
    print(i,j)

def f(x,y):
    # definition d'une fonction qui renvoie la
        norme de (x,y)
    return np.sqrt(x**2+y**2)

def monmax(x,y):
    if x>y:
        return x
    else:
        return y

print(f(3,4))
print(monmax(5,6))
```

On peut avoir de l'aide sur une fonction existante, mettre le curseur sur le nom de la fonction, et utiliser le raccourci `ctrl-I` de spyder. Sinon, on peut utiliser la commande `help`, ou ajouter un "?" a la fin de la commande. Par exemple: `help(np.random)` ou `M.dot?`.

On peut aussi utiliser la complétion automatique pour voir quelles sont les fonctions disponibles. Par exemple, on peut taper `linalg.d`–*tab* et voir qu'il y a linalg.det (déterminant) comme complétion possible.

Vecteurs et matrices numpy

Les `array` de numpy sont des tableaux de nombres tous du meme type (flottant, entier, chaine de caracteres de taille bornee). Ils peuvent avoir 1 dimension (vecteur), 2 dimensions (matrice) ou plus. Attention: les opérations standard se font coefficient par coefficient. Pour calculer un produit de matrices $A * B$, utiliser `A.dot(B)`.

```
M=np.array([[1,2,3],[4,5,6]],dtype=
        float)
N=np.array([[1,-1],[2,-2],[3,-3]],dtype
        =int)
P=np.array([[1.0,1,1],[2,2,2]]) #
        flottants à cause du
        1.0

print(M+0.1)  # ajoute 0.1 a tous les coefts
N2=N.transpose()
print(M*N2)  # produit terme a terme !!
print(M.dot(N)) # produit matriciel
```

On peut créer une matrice de zéros ou de uns d'une taille donnée ou de la meme taille qu'une matrice donnée. On peut aussi remplir une matrice avec une valeur, ou créer la matrice identité.

```
A=np.zeros((2,3)) #matrice 2x3
M0=np.zeros_like(M)
M1=np.ones_like(M)
```

```
M2=M0.copy()
M2[:]=np.pi
I=np.identity(3)
print(M0,"\n",M1,"\n",M2,"\n",I)
```

Les numéros de lignes et de colonnes commencent à 0.

```
M=np.array([[1.0,2,3],[4,5,6]])
print(M[0,0])
M[1,1]=np.pi*10
print(M)
```

Attention si on assigne un flottant à une matrice d'entiers !

```
M2=np.array([[1,2,3],[4,5,6]])
M2[0,0]=np.pi*10
print(M2)
```

On peut changer le format d'affichage des nombres

```
print(M)
np.set_printoptions(precision=3,
        suppress=True)
print(M)
```

Appliquer une fonction a un tableau numpy

La plupart des fonctions standard peuvent s'appliquer a un tableau numpy:

```
l=np.random.uniform(size=10) # 10 nombres
        aléatoires dans [0,1[
l2=np.exp(l) # les e^i pour i dans l
```

Cette possiblité est intéressante par sa simplicité: elle évite de faire une boucle et peut être parfois plus rapide (selon comment on programme la boucle). Il est important que l'entrée soit un tableau numpy (pas une liste python). La sortie est un tableau numpy. Mais pour appliquer une fonction qu'on a défini soi même il faut en général la "vectoriser": ici, à partir de la fonction f, on en crée une autre fv qu'on peut appliquer sur un tableau numpy.

```
def f(x):
    if x<5:
        return x**2-3*x+7
    else:
        return -1

fv=np.vectorize(f) #version vectorisee de f
l2=fv(l)
```

La plupart des fonctions numpy peuvent s'appeler en syntaxe "fonctionnelle" ou "objet". Typiquement, pour calculer la somme des coefficients d'un array `M`, on peut appeler `M.sum()` (syntaxe objet) ou bien `np.sum(M)` (syntaxe fonctionnelle).

```
print(M.sum()) #somme des coefts de M
print(np.sum(M)) #idem
```

La fonction `cumsum` est très utile, elle calcule la somme cumulée d'une liste (ligne par ligne pour une matrice). `np.arange(i,j)` est un tableau numpy contenant la liste des entiers dans l'intervalle $[i, j[$.

```
  l=np.arange(0,7)
  s=l.cumsum() #somme cumulée
  print(l)
  print(s)
```

Vues sur un tableau...

On peut créér plusieurs "vues" sur un meme jeu de données (parfois sans le vouloir !). La notation `a:b` représente l'intervalle $[a, b[$ (ouvert en b!). Dans l'exemple ci-dessous, changer une valeur dans N modifie M également.

```
M=np.array([[1,2,3],[4,5,6],[7,8,9]])
N=M[0:2,0:2]
# N est une ''vue'' sur la sous-matrice 2x2 de
        M.
N[1,1]=-7
print(M[1,1]) # renvoie -7
N[:,:]=np.identity(2) #remplace les coefs
        de N par id
print(M)
```

Attention, une affectation `N=M` crée une autre "vue" sur un même jeu de données. Ainsi, si on modifie ensuite un coef de N, cela modifiera M. Pour dupliquer les données en mémoire, utiliser `copy()`.

```
M=np.array([[1,2,3],[4,5,6]])
print(M[0,0]) # renvoie 1
N=M
N[0,0]=-7
print(M[0,0]) # renvoie -7
N2=M.copy()
M[0,0]=3
print(N2[0,0])  # renvoie -7
```

Algèbre linéaire

Résolution de systeme et inverse avec le module linalg de scipy

```
M=np.array([[1.0,2,3],[4,5,6],[7,8,10]]
        )
v=linalg.solve(M,[2,3,4])
print(v)
#inverse
MM=linalg.inv(M)
v2=MM.dot([2,3,4])
print(v2)
```

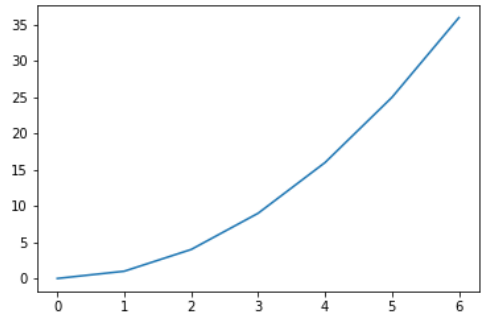
Valeur propres et vecteurs propres (eigenvalues, eigenvectors en anglais).

```
vap,vep=linalg.eig(M)
print(vap)  # les 3 valeurs propres
print(vep)  # la matrice contenant les vep
```

Graphiques

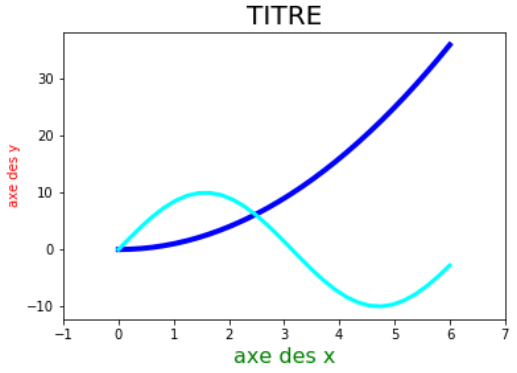
On utilise le module `mathplotlib`. Quelques points sur une parabole:

```
x=np.array([0, 1.0, 2, 3, 4, 5, 6])
y=x**2   # y est encore un array
plt.plot(x,y)
plt.show()
```



Le meme avec plus de points, en superposant une sinusoide, avec des titres et en changeant quelques parametres graphiques.

```
x=np.linspace(0,6,30)
#linspace renvoie ici 30 valeurs regulierement
        espacees entre 0 et 6
# x est un array (=liste) de 30 valeurs
y=x**2   # y est encore un array
plt.plot(x,y,linewidth=4,color='blue')
y2=10*np.sin(x) # y2 est aussi un array
plt.plot(x,y2,linewidth=3,color='cyan')
plt.xlabel('axe des x', color = 'green'
        , fontsize = 16)
plt.ylabel('axe des y', color = 'red',
        fontsize = 10)
plt.title('TITRE',fontsize=20)
plt.xlim(xmin=-1,xmax=7)   #taille du
        graphique en x
plt.show()
```



Pour fermer un graphique, on peut utiliser `plt.close()`

La "commande magique" `%matplotlib auto` permet que les figures s'ouvrent dans leur propre fenetre, ce qui permet de les enregistrer, de modifier quelques parametres. Ce n'est pas une commande python, mais "ipython", (à executer dans la console dans spyder/pyzo). Pour revenir au mode normal, `%matplotlib inline`

Générateurs aléatoires, lois classiques

Le générateur aléatoire le plus simple tire des valeurs au hasard dans un ensemble fini. On peut préciser l'ensemble des valeurs possibles et les probabilités respectives. Par exemple, dans le programme ci-dessous, on tire 7 fois un prénom, indépendamment, avec la loi ci-dessus. La commande `np.random.seed` permet d'initialiser la *graine* d'aléa, et d'avoir des résultats reproductibles.

```
np.random.seed(10)
espace=np.array(['Albert','Becassine','
        Cristobal','Desdemone'])
loi=np.array([.2,.3,.4,.1])
l=np.random.choice(espace, 7, p=loi)
print(l)
```

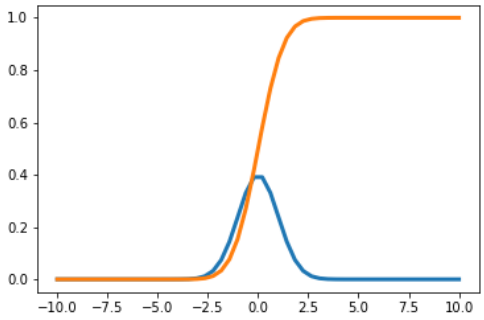
On peut aussi générer des variables aléatoires continues. Les parametres `loc` et `scale` permettent de faire une transformation affine: si X est la variable aléatoire renvoyée par `loixyz()`, alors $aX + b$ est la variable aléatoire renvoyée par `loixyz(loc=b,scale=a)`. Exemple avec la loi normale.

```
#tire une liste de 7 nombres aleatoires,
        suivant la loi normale
l1=np.random.standard_normal(7)
# 1000000 nombres aleatoires, suivant la loi
        normale de moyenne 3, et d'ecart-type
        4
l2=np.random.normal(size=1000000,loc=3,
        scale=4)
l3=np.random.normal(3,4,1000000) #idem
```

Exemple: on définit f comme la densité de la loi normale (en anglais: Probability Density Function= pdf), et g comme la fonction de répartition (en anglais: Cumulative Distribution Function: cdf)

```
f=stats.norm.pdf   # ceci est une fonction
g=stats.norm.cdf   # ceci aussi
x=np.linspace(-10,10,50)
y1=f(x)
y2=g(x)

plt.plot(x,y1,linewidth=3)
plt.plot(x,y2,linewidth=3)
plt.show()
```



Voir la doc de reference pour la (longue) liste de lois disponible. En particulier `geometric`, `poisson`, `uniform`, `exponential`, `gamma`.

Pour se rappeler rapidement l'ordre et la signification des parametres, on peut appeler l'aide ou utiliser le point d'interrogation, e.g. `np.random.gamma?`

Le module `scipy.stats` permet de travailler avec ces lois: moyennes, variances, fonctions de répartitions, densités, moments, etc. On peut voir les nombreuses lois disponibles ici.

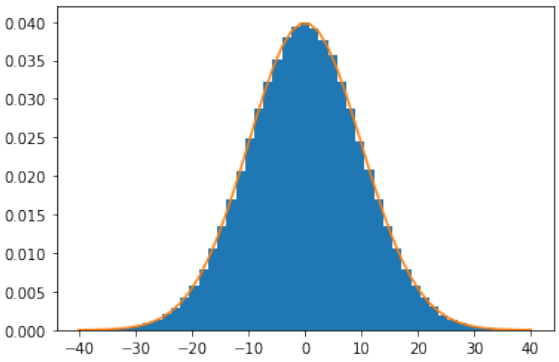
```
print(stats.expon.mean(scale=2)) #
        moyenne de la loi exponentielle
        (parametre 2)
print(stats.poisson.var(3)) #variance de
        la loi de poisson, parametre 3
```

Histogrammes

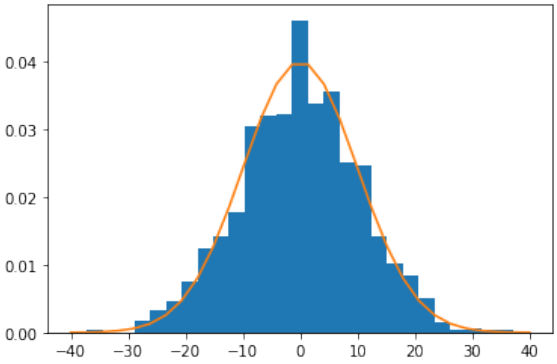
On tire 1,000,000 valeurs selon une loi normale, et on affiche l'histogramme correspondant avec la commande `plt.hist`. On met 50 cases (bins=bacs en anglais) entre -40 et 40. On trace ensuite sur le même graphique la densité de la loi normale. L'option `normed=True` (ou `density=True` pour les versions récentes) normalise les données pour faire en sorte que la somme des aires des rectangles dessinés fasse 1.

```
ecarttype=10
N=1000000
l=np.random.normal(0,ecarttype,N) #on
        tire plein de valeurs
intervalles=np.linspace(-40,40,50) #50
        cases pour l'histogramme
plt.hist(l,normed=True,bins=intervalles
        )

densite=stats.norm(scale=10).pdf #
        x \mapsto \frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/2\sigma^2}, \sigma = 10
plt.plot(intervalles,densite(
        intervalles))
plt.show()
```



Le même histogramme avec seulement 1,000 valeurs et 30 cases



2 Générer des variables aléatoires.

Le but est de voir plusieurs méthodes pour générer des variables aléatoires de loi donnée.

2.1 Simulation par fonction de répartition inverse

Proposition. *$F : \mathbb{R} \to [0,1]$ une fonction de répartition. Pour $u \in [0,1]$, on désigne par $F^{-1}(u) := \inf\{x \in \mathbb{R}, F(x) \geq u\}$ l'inverse généralisée de la fonction de répartition F. Si X est une variable aléatoire de loi uniforme $X \sim \mathcal{U}_{[0,1]}$ alors $F^{-1}(X)$ a pour fonction de répartition F.*

1. Que vaut la fonction de répartition de la loi de Bernoulli $\mathcal{B}(p)$? À partir de variables uniformes sur $[0,1]$, générer un $n-$échantillon de variables de loi $\mathcal{B}(1/2)$ à valeurs dans $E = \{0,1\}$. Meme question avec $E = \{-1,1\}$ et $p = 0.3$.

2. Simuler une variable aléatoire Y de loi uniforme à valeurs dans $\{0,1,2,3,4\}$.

3. À partir de variables uniformes, simuler un $n-$échantillon $(X_1, \ldots, X_n)$ de variables aléatoires de loi exponentielle de paramètre 2. Tracer un histogramme, et superposer la courbe avec la densité.

4. Mettre en évidence la loi des grands nombres en traçant la fonction $k \mapsto (X_1 + \ldots + X_k)/k$ pour k allant de 1 à n.

2.2 Simulation par rejet

Soit leb la mesure de Lebsgue sur $\mathbb{R}^n$. Etant donné $A \in \mathcal{B}(\mathbb{R}^d)$ tel que $\text{leb}(A) > 0$, la mesure uniforme sur A est la mesure de probabilité $\frac{1}{\text{leb}(A)}\text{leb}_{|A}$.

Proposition. *Soient A, $D \in \mathcal{B}(\mathbb{R}^d)$ tels que $A \subset D$ et $\text{leb}(A) > 0$. Soit $(X_i)_{i\in\mathbb{N}}$ une suite i.i.d. de variables aléatoires définies sur un espace de probabilité $(\Omega, \mathcal{F}, \mathbb{P})$, de loi uniforme sur D. Introduisons le temps $\tau := \inf\{i \in \mathbb{N}^*, X_i \in A\}$.*

Alors la loi de X_τ est la loi uniforme sur A.

De plus τ suit une loi géométrique $\mathcal{G}(p)$, avec $p = \frac{\text{leb}(A)}{\text{leb}(D)}$ donc $E(\tau) = \frac{\text{leb}(D)}{\text{leb}(A)}$, et on peut montrer que τ et X_τ sont indépendantes.

En d'autres termes, en tirant des points aléatoirement dans D jusqu'à trouver un point dans A, il faut effectuer $\frac{\text{leb}(D)}{\text{leb}(A)}$ tirages en moyenne pour obtenir un point dans A, et le point obtenu est uniformément réparti.

De la proposition, on déduit la méthode de simulation suivante. Soit X une variable aléatoire réelle de loi μ possèdant une densité continue f à support compact inclus dans $[a,b] \subset \mathbb{R}$ et telle que son graphe soit inclus dans $[a,b]\times[0,M]$, pour un certain $M \in \mathbb{R}^+$. On considère une suite $(Z_i)_{i\in\mathbb{N}} = (X_i, Y_i)_{i\in\mathbb{N}}$ de variables aléatoires uniformes dans $[a,b] \times [0,M]$ et l'on définit $\tau = \inf\{i \in \mathbb{N}, f(X_i) > Y_i\}$. Alors X_τ a pour loi μ.

1. Utiliser la méthode de simulation ci-dessus pour générer une variable aléatoire de densité $x \mapsto \frac{\pi}{2}\sin(\pi x)$ sur l'intervalle $[0,1]$.

2. Faire un histogramme

On peut généraliser la méthode de rejet de la façon suivante. Soit X une variable aléatoire réelle de loi μ qui possède une densité continue f majorée uniformément par une densité g, i.e. il existe une constante $C \geq 1$ telle que

$$\forall x \in \mathbb{R},\ f(x) \leq Cg(x), \text{ avec } \int g(y)dy = 1.$$

On suppose que l'on sait facilement simuler des variables aléatoires de loi de densité g. Soient donc $(X_i)_{i\in\mathbb{N}}$ une suite i.i.d. de variables aléatoires de loi de densité g et $(U_i)_{i\in\mathbb{N}}$ une suite i.i.d. de variables aléatoires uniformes sur l'intervalle $[0,1]$, indépendantes de $(X_i)_{i\in\mathbb{N}}$. Si l'on considère le temps $\tau := \inf\{i \geq 1,\ f(X_i) > Cg(X_i)U_i\}$, alors X_τ a pour loi μ.

Application : on souhaite simuler une variable aléatoire X de loi $\mathcal{N}(0,1)$. On montre sans trop de difficulté la majoration suivante :

$$\forall x \in \mathbb{R},\ \underbrace{\left(\frac{e^{-x^2/2}}{\sqrt{2\pi}}\right)}_{:=f_X(x)} \leq \underbrace{\sqrt{\frac{2e}{\pi}}}_{:=C} \times \underbrace{\left(\frac{1}{2}e^{-|x|}\right)}_{:=g(x)}.$$

3. a. Montrer que si ε suit une loi de Bernoulli de paramètre $1/2$ dans $\{-1,1\}$ et si Z est une variable exponentielle de paramètre 1 indépendante de ε, alors εZ a pour densité la fonction g sur $\mathbb{R}$;

   b. Implémenter un algorithme qui simule une variable gaussienne via la méthode de rejet ;

c. Simuler 1,000,000 variables selon cet algorithme et vérifier que l'histogramme correspondant approche bien la densité gaussienne.

d. A quelle proportion de rejet s'attend-on ? Quelle est la proportion de rejet observée ?

2.3 Méthode de Monte-Carlo

La méthode de Monte-Carlo est basée sur la loi des grands nombres. Elle permet par exemple de calculer des valeurs approchées d'intégrales ou d'espérances, en utilisant des réalisations i.i.d. d'une loi que l'on sait simuler. Par exemple, si $(X_n)_{n\geq 1}$ est une suite de variables aléatoires i.i.d. de loi uniforme sur $[0,1]^m$ et si $f : [0,1]^m \to \mathbb{R}$ est une fonction intégrable par rapport à la mesure de Lebesgue sur $[0,1]^m$, alors la loi des grands nombres entraîne la convergence presque-sûre suivante :

$$\frac{1}{n}\left(f(X_1) + \ldots + f(X_n)\right) \to \mathbb{E}(f(X_1)) \ \ p.s.$$

$$= \int_{[0,1]^m} f(x)dx.$$

Si l'on sait majorer la variance de $f(X_1)$, on est de plus en mesure de fournir des intervalles de confiance pour contrôler l'erreur commise dans l'approximation. La vitesse de convergence de cette méthode (de l'ordre de $\sqrt{n}$) est lente par rapport à des méthodes déterministes. Cependant cette vitesse ne dépend pas de la régularité de l'intégrande f et dépend plus faiblement de la dimension m que les méthodes déterministes.

Calculer les approximations des intégrales suivantes.

1. $\int_0^1 4\sqrt{1-x^2}dx$ (aire du disque unité)
2. $\int_{[-1,1]^3} \mathbb{1}_{\{x^2+y^2+z^2\leq 1\}} dxdydz$ (volume de la boule unité).