Fiche 1 : Matplotlib et probabilités

Proposition d'exo 1 : rencontres

Thèmes concernés : Probas, Programmation, Aires, (éventuellement équation de droites)

Alice et Bob se sont donné rendez-vous. Chacun arrive au hasard entre 8h et 9h mais le premier arrivé n'attend pas l'autre plus de 10 minutes.
Quelle est la probabilité qu'Alice et Bob se rencontrent ?

Partie A : Simulation avec Python

Chaque rendez-vous peut être associé à un point de coordonnées (x, y) dans un repère orthonormé où x et y représentent respectivement les temps d'arrivée de Bob et Alice en minutes écoulées depuis 8h.
Par exemple, si Bob arrive à 8h30 et Alice à 8h45, le point associé a pour coordonnées $(30; 45)$
On veut créer une fonction Python `simu(nb_simul)` qui simule plusieurs rencontres entre Alice et Bob et qui affiche dans un repère orthonormé les points associés. S'il y a rencontre, le point associé est en rouge, sinon il est en vert.
Le nombre de simulations est passé en paramètre.

1. A quelles conditions sur (x, y) y'a-t-il rencontre ?
2. On a besoin de choisir aléatoirement une rencontre. Quel module importer ?
3. On veut afficher des points dans un repère. Quel module importer ?
4. Compléter la fonction (*on peut guider un peu moins les élèves éventuellement*)

```
def simu(nb_simul):

    for k in range(.....):
        x = ..... # heure d'arrivée de Bob
        y = ..... # heure d'arrivée d'Alice
        if ..... < 10:
            plot(x,y,"gx") # on affiche le point en « green » avec un « rond »
        else:
            plot(x,y,"rx") # rouge et rond

    # avec xlim et ylim, on règle la fenêtre
    xlim(-15,75)
    ylim(-5,65)
    show()
```

Partie B : Exploitation de la figure obtenue pour calculer la probabilité

On obtient par exemple la figure ci-contre en prenant **10 000** simulations (on peut se limiter à 5000 aussi). On peut admettre avec les élèves que la probabilité correspond à $\frac{\text{aire du domaine vert}}{\text{aire du domaine rouge}}$ puis réaliser le calcul….
Les deux triangles rouges sont rectangles isocèles et les deux plus petits côtés ont pour longueur 50 donc on peut calculer leur aire puis l'aire du domaine vert.

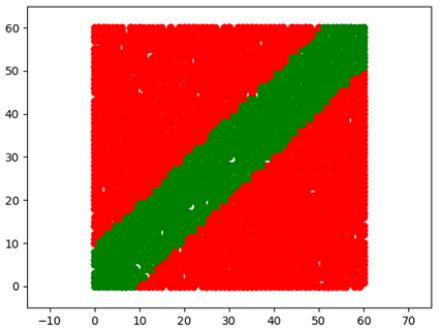


Figure obtenue avec 10 000 simulations

Commentaire: Eventuellement faire une partie C pour calculer plus rigoureusement…

Proposition d'exo 2 : quelques idées autour de l'échantillonnage

Un exo autour de l'échantillonnage et plus particulièrement d'intervalle de fluctuation (seconde, première ou terminale). Par exemple, voici ce que dit le programme de 2^{nde}.

Capacités attendues

- Lire et comprendre une fonction Python renvoyant le nombre ou la fréquence de succès dans un échantillon de taille n pour une expérience aléatoire à deux issues.
- Observer la loi des grands nombres à l'aide d'une simulation sur Python ou tableur.
- Simuler N échantillons de taille n d'une expérience aléatoire à deux issues. Si p est la probabilité d'une issue et f sa fréquence observée dans un échantillon, calculer la proportion des cas où l'écart entre p et f est inférieur ou égal à $\frac{1}{\sqrt{n}}$.

Un contexte simple : Dans une urne, on a placé 10 boules indiscernables au toucher : 3 boules rouges et 7 boules vertes. On tire au hasard une boule dans l'urne.

Partie A : Une fonction intermédiaire

Dans un premier temps, on fait faire aux élèves une fonction `frequence` qui simule un certain nombre de fois le tirage d'une boule dans cette urne (on note n ce nombre de tirages). Elle retourne la fréquence d'apparition de la boule rouge à l'issu de ces n tirages.

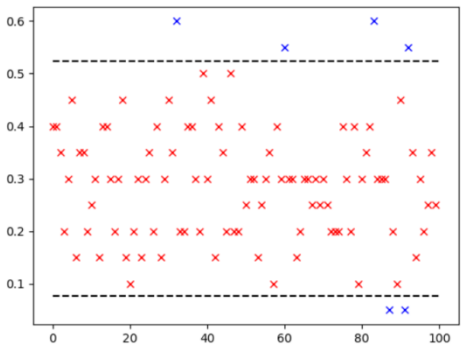
```
import random
def frequence_rouge(n):

    """ n (entier, type int) représente le nombre de tirages. Pour chacune des n expériences, on simule de façon suivante : on choisit aléatoirement un nombre entre 1 et 10. Les entiers 1,2,3 représentent la      boule rouge et les autres, la verte. """

    somme = 0
    for k in range(n):
        alea = random.randint(1,10)
        if alea <=3:
            somme = somme+1

    return somme/n
```

Partie B : Echantillonnage

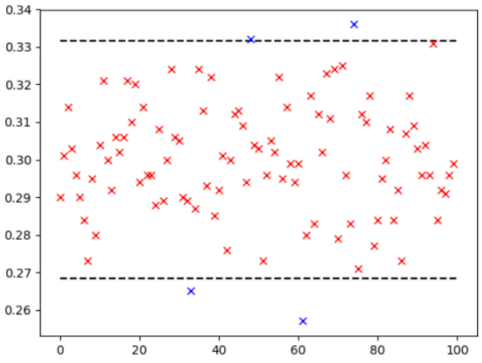
On va réaliser 100 fois l'expérience qui consiste à réaliser n tirages d'une urne dans la boule. Pour chacune des 100 expériences, on s'intéresse à la fréquence de boules rouges obtenue à l'issu des n tirages.
On veut représenter graphiquement ces 100 fréquences obtenues avec 100 points dont les ordonnées correspondent aux fréquences trouvées et les abscisses aux entiers $1,2,3,\ldots,100$.

1. Pour cela, créer une fonction `echantillon(n)`. On utilisera la fonction `frequence_rouge`.

Commentaire pour le prof : Si on part sur une première activité où les élèves découvrent le fait que les fréquences fluctuent autour de 0,3, on peut leur demander d'observer ensuite les figures obtenues en prenant des valeurs de plus en plus grandes de n.
Sinon, si les élèves connaissent déjà les intervalles de fluctuation (par exemple, en seconde, l'intervalle $\left[0{,}3-\frac{1}{\sqrt{n}}\,,0{,}3+\frac{1}{\sqrt{n}}\right]$*), on peut leur demander d'afficher le point en rouge si la fréquence est dans l'intervalle de fluctuation et en bleu sinon.*
Puis éventuellement afficher deux segments pour délimiter l'intervalle de fluctuation. Quelques exemples de figures obtenues.



`echantillon(20)`



`echantillon(1000)`

Proposition d'exo 3 : marche aléatoire à une dimension et applications

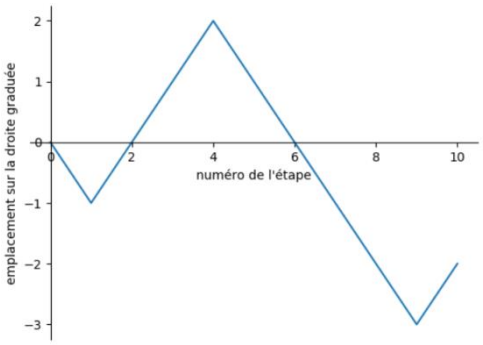
On se propose d'étudier un phénomène de marche aléatoire sur une droite graduée : un objet se trouve initialement à l'emplacement 0 de la droite et à chaque étape, il a une chance sur deux d'avancer d'une unité et une chance sur deux de reculer d'une unité. Le nombre d'étapes est appelé le **nombre de pas** de la marche aléatoire.
A l'étape 0, l'objet se trouve à l'emplacement 0, à l'étape 1, il se trouve à l'emplacement 1 ou -1.

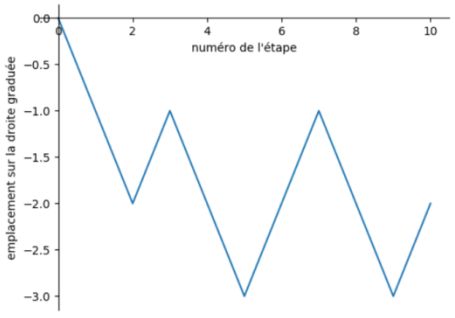
1. Quels sont les emplacements possibles de l'objet à l'étape 2 ?
2. Ecrire une fonction Python `deplacement_aleatoire()` qui renvoie aléatoirement -1 ou 1 (avec la même probabilité $\frac{1}{2}$).

```
import random
def deplacement_aleatoire():
```

3. On souhaite représenter graphiquement une marche aléatoire à n pas dans un repère de la façon suivante :
   - on représente n points, chaque point étant relié au précédent ;
   - les abscisses de ces points correspondent au numéro de l'étape ;
   - l'ordonnée de chaque point est l'emplacement sur lequel se trouve l'objet à un instant donné par l'abscisse.

Voici deux exemples de simulation d'une marche aléatoire à **10** pas.



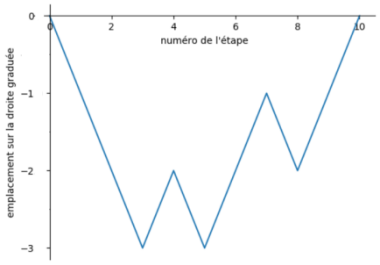


On va pour cela générer deux listes `abscisses` et `ordonnees` et utiliser `plot(abscisses,ordonnees)`.

1. La liste des abscisses est toujours la même quelle que soit la marche aléatoire. Elle correspond à `[0,1,2, … pas]` où « `pas` » est une variable qui représente le nombre de pas de la marche aléatoire.
   Comment créer cette liste en Python ?

Commentaire pour le prof :
Pour faire la liste des abscisses, au choix :
`abscisses = range(1,pas+1)` (si on veut absolument que ça soit de type `list`, on peut rajouter `list(range(1,pas+1))` mais après pour tracer avec plot, aucun problème si on laisse `range`).
ou en compréhension : `abscisses = [i for i in range(1,pas+1) ]`

Commentaire pour le prof : on peut rajouter des questions intermédiaires. Par exemple, que valent les listes `abscisses` et `ordonnees` dans la situation donnée par le schéma suivant ?



2. Compléter la fonction `simulation(pas)` suivante qui permet de simuler et représenter graphiquement une marche aléatoire dont le nombre de pas est passé en paramètre.

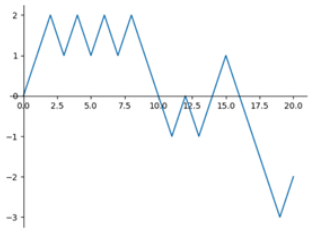
```
def simulation(pas):

    abscisses = ……
    ordonnees = [0]
    for k in range(1,pas+1):
        choix = deplacement_aleatoire()
        ordonnees.append(……)

    plot(……,……)
    axesnormaux() # pour que les axes se coupent en 0 (il faut importer le module où se trouve la fonction)
    xlabel("numéro de l'étape") # pas obligatoire
    ylabel("emplacement sur la droite graduée") # pas obligatoire
```

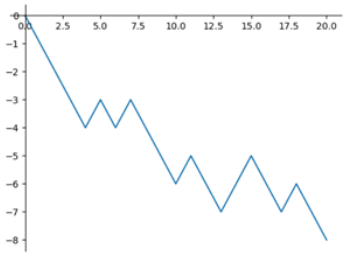
Commentaire pour le prof : je préfère ne pas mettre le `show()` à la fin à l'intérieur de la fonction mais plutôt le mettre dans la console. En effet, si on veut afficher sur le même graphique plusieurs simulations, on pourra le faire en appelant plusieurs fois la fonction simulation dans la console puis mettre un `show()` à la fin uniquement des appels.

3. Représenter graphiquement des simulations à 1000 pas.

[Pour le prof] Approfondissements possibles en probabilités

- On peut demander d'observer plusieurs fois une simulation avec un nombre de pas fixé (par exemple 20) et observer la fréquence de marches aléatoires où il y a retour en 0. Les élèves s'interrogent alors sur la probabilité de retour en 0.


Un exemple de marche aléatoire avec retour en 0


Un exemple de marche aléatoire sans retour en 0

On peut alors demander de créer une fonction `retour_en_0` qui simule une marche aléatoire et qui vérifie s'il y a retour en 0 ou pas (elle renvoie par exemple 1 s'il y a retour en 0 et 0 s'il n'y a pas). Il suffit en fait de faire comme plus haut et de vérifier si la liste `ordonnees` contient un « 0 » (en dehors du premier élément qui vaut forcément 0). Pour demander ça à Python, on peut le faire avec l'instruction `0 in ordonnees[1:]`. Le mot clé `in` permet de vérifier si un élément est dans une liste (ça renvoie True si c'est le cas et False sinon). La commande `ordonnees[1:]` permet de donner la liste des éléments de `ordonnees` à partir de l'élément d'indice 1.
(voir le fichier Python joint pour un exemple d'implémentation).
Une fois cette fonction créée, on peut classiquement simuler plein de fois l'expérience puis donner la fréquence empirique de retour en 0. Cette fréquence se rapproche de la probabilité théorique.

Notons pour tout entier n pair,

$$P(n) = 1 - \frac{\binom{n}{n/2}}{2^n}$$

Pour info, la probabilité de retour en 0 vaut $P(n)$ (où n désigne le nombre de pas) lorsque n est pair et $P(n-1)$ lorsque n est impair.

Cette probabilité tend vers 1 quand n (nombre de pas) tend vers $+\infty$. On peut tester avec la fonction `proba_retour_en_0(pas)` du fichier python joint. Quelques tests dans la figure ci-contre.

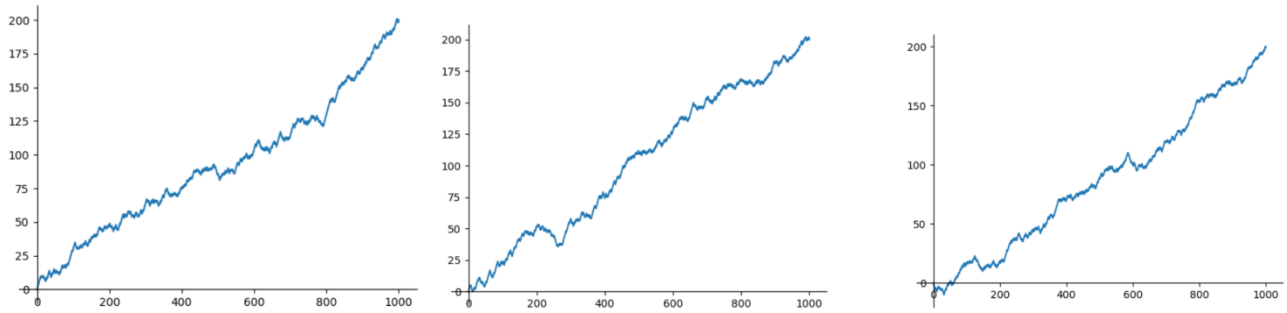
```
>>> proba_retour_0(10)
0.7574
>>> proba_retour_0(20)
0.8188
>>> proba_retour_0(30)
0.8575
>>> proba_retour_0(100)
0.9204
>>> proba_retour_0(1000)
0.9729
```

- On peut également s'interroger sur la probabilité que pour une marche aléatoire (infinie), il y ait retour en 0 à un instant donné m.
  Bien sûr, si m est impair, il n'y a aucune chance que le retour en 0 se fasse à l'étape m.
  Si m est pair, il y a des chances. Pour simuler cela en Python, c'est similaire à la probabilité du retour en 0. Il suffit de faire une marche aléatoire avec m pas seulement et vérifier s'il y a retour en 0 **à la fin**. Autrement dit, vérifier si le dernier élément de la liste `ordonnees` vaut 0. On peut accéder au dernier élément d'une liste car son indice est considéré comme -1.
  Donc avec l'instruction `0 in ordonnees[-1]`, on s'en sort pareil.
  Pour info, la probabilité exacte quand m est pair vaut $\frac{\binom{m}{m/2}}{2^m}$ et en utilisant la formule de Stirling, on a un équivalent quand m tend vers $+\infty$: $\frac{1}{\pi\sqrt{\frac{m}{2}}}$
  Une fonction `Python` est donnée dans le fichier joint permettant de simuler cela.

- **Théorème du scrutin**.
  Problème : Au cours d'un scrutin opposant deux candidats A et B, le candidat A obtient 600 voix et le candidat B, 400 voix. Quelle est la probabilité pour que A ait été majoritaire (*strictement, donc pas d'égalité*) tout au long du dépouillement ?

  On peut modéliser le dépouillement par une marche aléatoire à 1000 pas. On part de 0. Pendant le dépouillement, si le bulletin « candidat A » apparaît, on monte, sinon on descend.
  Dans cette marche aléatoire, on sait que l'on monte 600 fois et que l'on descend 400 fois. On arrive à la fin au point (1000,200).
  Le candidat A a été majoritaire si la marche aléatoire n'atteint jamais les négatifs (et ni 0).

Dans le fichier python joint, je propose une fonction `simu2(nb_haut,nb_bas)` qui affiche une marche aléatoire et prend en paramètres le nombres de coups vers le haut et le nombre de coups vers le bas. Cette marche aléatoire a forcément un nombre de pas qui vaut `nb_haut+nb_bas`.
Par exemple, si je veux afficher une marche aléatoire avec 600 coups vers le haut et 400 vers le bas, voici trois exemples.



Cet exemple est directement lié à notre problème de scrutin. Ca modélise à chaque fois un dépouillement. Dans la première marche, le candidat A a été majoritaire pendant l'intégralité du dépouillement. Dans la deuxième marche, en zoomant bien, on voit que l'axe des abscisses a été touché donc il y a eu un cas d'égalité à un moment du dépouillement. Le candidat A n'a donc pas été tout le temps majoritaire. Dans la dernière figure, le candidat B a été majoritaire au tout début avant.

Dans le fichier, vous trouverez ensuite deux fonctions dont une `scrutin(nb_voix_A, nb_voix_B)` qui renvoie la probabilité que A (que l'on suppose toujours gagnant, dans ce sens là) ait été majoritaire tout au long du dépouillement.
Je trouve la probabilité suivante :

```
>>> scrutin(600,400)
0.2033
```

Remarque : le théorème du scrutin dit justement que cette probabilité vaut $\frac{\text{nb\_voix\_A - nb\_voix\_B}}{\text{nb\_voix\_A + nb\_voix\_B}}$. Ici, ça nous donne une probabilité théorique de $\frac{600-400}{600+400} = 0{,}2$. L'approximation est plutôt pas mal !

Autre remarque : on a aussi un théorème du scrutin en considérant « A majoritaire au sens large », c'est-à-dire autorisant les cas d'égalité. Dans le programme Python, il y a juste un « $<$ » à mettre à la place d'un « $\leq$ ». Théoriquement, le résultat devient $\frac{\text{nb\_voix\_A - nb\_voix\_B+1}}{\text{nb\_voix\_A}}$, soit dans notre situation $\frac{600-400+1}{600} \approx 0{,}335$.
Avec Python, on trouve :

```
>>> scrutin_sens_large(600,400)
0.3318
```

Proposition d'exo 4 : marche aléatoire sur Z^2

Thèmes concernés : Probas, Vecteurs.

On cherche dans cet exercice à simuler une marche aléatoire en deux dimensions. On ne marche plus sur une droite mais dans un plan. Au départ, un objet se trouve en position (0,0) et peut avancer dans une des quatre directions (nord, sud, est, ouest).
On représente ces quatre directions par des vecteurs. Le nord est représenté par exemple par le vecteur de coordonnées $(1;0)$.

1. Par quels vecteurs représenter les directions sud,est,ouest ?

2. Ecrire une fonction Python, `donne_direction()` qui retourne aléatoirement une des quatre directions nord/sud/est/ouest sous la forme d'une liste à deux éléments[1] représentant les coordonnées du vecteur associé à la direction.
3. On veut représenter graphiquement une marche aléatoire à deux dimensions. On représente les différents points, chaque point étant relié au précédent. Représenter graphiquement la marche aléatoire suivante :
   On part de **(0,0)** puis on avance selon les directions nord,nord,nord,sud,nord,est,nord,ouest,sud.

4. On créé une fonction `marche_aleatoire(nb_points)` qui prend en pramètre le nombre des points de la marche aléatoire et qui la représente graphiquement.
   Compléter cette fonction.

```
def marche_aleatoire(nb_point):

    abscisses = [0]
    ordonnees = [0]
    # au départ, le point est (0,0)

    for k in range(1,nb_point):
        direction = donne_direction() #
        abscisses.append(.....)
        ordonnees.append(.....)

    plot(.....,.....)
    axesnormaux()
```

5. Visualiser quelques marches aléatoires en prenant 10 000 pas.

Remarque : ici aussi, la probabilité de retour en 0 tend vers 1 lorsque le nb de points tend vers l'infini.
A partir de la dimension 3, la probabilité de retour en 0 d'une marche aléatoire ne tend plus vers 0 (voir Théorème de Pólya)

Mes sources pour écrire cette fiche :
http://alain.camanes.free.fr/universite/vulgarisation/expose_recurrence_clemenceau.pdf
http://culturemath.ens.fr/maths/pdf/proba/marchesZ.pdf
Manuel scolaire Déclic 2nde MATHS.
Cours de maths de proba de la fac.

[1] *un tuple c'est mieux mais au lycée, on voit surtout les listes.*