M1 Physique Appliquée
Année 2016-2017

Université Paris-Sud XI

Programmation et données numériques
exercices complémentaires

Ces TDs sont plus difficiles et facultatifs, pour ceux qui ont envie d'aller plus loin.

Création d'un histogramme

On rappelle qu'un histogramme représente le nombre d'occurrences de valeurs réelles $\{x_i\}$ sur des intervalles élémentaires que l'on supposera de largeur uniforme Δx et qui découpe un intervalle plus grand noté $[x_{\min}, x_{\max}]$. On appelle en anglais `bin` chacune des boîtes ainsi créées et on notera N leur nombre total. On représente alors par un trait horizontal de largeur Δx le résultat dans chaque boîte. Pour un même ensemble de données $\{x_i\}$, l'allure de l'histogramme dépend de Δx et il est souhaitable de normaliser celui de sorte que l'intégrale sous l'histogramme soit égale à un. Dans la plupart des cas rencontrés en physique, la limite où $\Delta x \to 0$ pour un nombre de données $M \to \infty$ correspondra à la densité de probabilité $p(x)$ de la variable x.

- ☞ Se faire un dessin au brouillon représentant un histogramme typique et les différents paramètres pertinents introduits ci-dessus.
- ☞ Écrire une fonction `Histogram(Liste,(xmin, xmax),N=10)` qui prend en argument une liste de données réelles `Liste` et qui trace l'histogramme des résultats dans un intervalle $[x_{\min}, x_{\max}]$ en le découpant en N intervalles élémentaires. Dans cette première version, on supposera implicitement que tous les éléments de la liste sont dans l'intervalle $[x_{\min}, x_{\max}]$. On notera `X` les abscisses de l'histogramme et `Y` ses ordonnées et le tracé se fera à l'aide de la fonction `plot` de `pylab` :

```
plot(X,Y,'b')
```

  On souhaite enfin que la fonction retourne le couple `(bins,H)` où `bins` est une liste rassemblant les abscisses du début des boîtes et `H` l'histogramme normalisé.
  On pensera à utiliser l'opérateur division entière // pour obtenir l'indice de la boîte à remplir. D'autre part, pour créer des une courbe avec créneaux, vous allez devoir dédoubler les abscisses. Pour ce faire, on pourra utiliser une somme de deux listes ainsi que la fonction `sorted`*. Vous allez sûrement devoir écrire cette fonction en tâtonnant et le mieux pour vérifier est de comparer vos premiers tracés avec ceux de la question suivante.*
- ☞ Tracer alors les histogrammes de listes aléatoires de M échantillons obtenus avec les fonctions `uniform(-0.75,0.75)` de la bibliothèque `random` sur l'intervalle $[-1, 1]$. Observer qualitativement l'effet de la variation du nombre de boîtes et/ou du nombre d'échantillons. On tracera sur le même graphe les fonctions analytiques attendues.
- ☞ Si maintenant certains éléments de la liste n'appartiennent pas à l'intervalle $[x_{\min}, x_{\max}]$, la fonction précédente ne trouve pas les indices des boîtes de ces éléments et renvoie une erreur. Il faut donc filtrer la liste pour ne conserver que les éléments dans l'intervalle choisi puis faire attention à la normalisation qui doit tenir compte des éléments rejetés. Écrire une nouvelle version de la fonction `Histogram(Liste,(xmin, xmax),N=10)` qui tienne compte de ce filtrage. *On rappelle qu'il existe une fonction* `filter` *pour les listes.*

- ☞ Tracer alors les histogrammes sur l'intervalle $[-1, 1]$ de listes aléatoires de M échantillons obtenus avec la fonction `gauss(0.0,σ)` avec divers écart-types σ . Vérifier le bon comportement, en particulier la normalisation, en traçant sur le même graphe la fonction analytique attendue, à savoir :

$$g(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-x^2/2\sigma^2} \ .$$

- ☞ Vérifier que votre fonction remplit le même rôle que la fonction `hist` de la bibliothèque `pylab`.

Une classe Vecteur

Lire le chapitre Introduction à la programmation orientée objet sur le memento ainsi que le chapitre Classes et objets sur le site `www.courspython.com`. Demandez aussi des explications aux chargés de TDs si vous avez des questions puis essayer de construire une classe de vecteurs selon les étapes ci-dessous.

- ☞ Écrire le début de la classe `Vecteur` en faisant en sorte que le contenu du vecteur soit stocké dans une liste appelée `data` et sa taille dans l'attribut noté `size`.
- ☞ Écrire le constructeur `__init__(self,taille=0,valeur=0.0)` de sorte qu'il puisse prendre en arguments une taille `taille` et une valeur `valeur` par défaut. Faire en sorte que les données soient converties en nombres réels.
- ☞ Rajouter à la signature du constructeur une option `liste=[]` qui permet de créer le vecteur à partir de la liste `liste` dès que celle-ci a une taille non-nulle. On convertira les données de `liste` en réels.
- ☞ Définir la méthode `__str__(self)` pour faire afficher le vecteur sous la forme `[ a b c ]`.
- ☞ Effectuer les premiers tests par les commandes :

```
print Vecteur(), Vecteur(3), Vecteur(taille=3,valeur=1), Vecteur(taille=2,valeur='1.0')
print Vecteur(liste=range(3))
a = Vecteur(taille=3,valeur=1)
print type(Vecteur), type(a), a.__class__
```

- ☞ Écrire une méthode `apply(self,f)` qui applique la fonction f aux éléments du vecteurs. *Option : y créer un test qui vérifie que f est bien une fonction. Pour cela, on pourra utiliser la librairie* `types` *qui définit le type d'une fonction :*

```
from types import FunctionType
```

Tester le fonctionnement par la commande

```
v = Vecteur(liste=range(10))
from math import sin,pi
v.apply(lambda x: sin(pi*x/6.0))
print v
```

- ☞ Surcharger les opérateurs binaires `+,-` pour qu'ils ajoutent/retirent un autre vecteur après avoir vérifié que l'argument est de type `Vecteur` et que les dimensions coïncident. On pourra vérifier que le test de classe d'une instance peut se faire par l'instruction :

```
objet.__class__ is Vecteur
```

- ☞ Surcharger les opérateurs unaires `+=,-=` pour qu'ils ajoutent/retirent une constante à tous les éléments. On utilisera une conversion de l'argument vers le type `float`.
- ☞ Surcharger l'opérateur `*` pour qu'il renvoie le produit scalaire après avoir vérifié que l'argument est bien un vecteur et qu'il a la même taille.

- ☞ Écrire deux fonctions toutes deux appelées `norm` qui renvoient la norme du vecteur : une qui est une méthode de la classe et une externe à la classe.
- ☞ Faire afficher le résultat des tests suivants :

```
x, y = Vecteur(liste=range(8)), Vecteur(liste=range(0,16,2))
x += 0.5
print x, y, x - y
print x*y, x.norm(), norm(x)
```

- ☞ Surcharger les méthodes `__getitem__(self,key)`, `__setitem__(self,key,value)` et `__len__(self)` en renvoyant vers les méthodes définies pour les listes. Cela permet d'accéder aux éléments ainsi qu'utiliser les slices. Effectuer les tests suivants à partir du vecteur `x` précédent :

```
x[1] = 2.1
print x[:3]
```

- ☞ Si vous n'avez pas pensé à documenter vos méthodes, faites-le maintenant et observer comment Python met automatiquement en forme la documentation de votre classe en tapant :

```
help(Vecteur)
```