M1 Physique Appliquée — Université Paris-Sud XI
Année 2015-2016

Programmation et données numériques

TD3 : programmation orientée objet – Numpy

Une classe Vecteur

- ☞ 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)
```

Algèbre linéaire avec Numpy

Importer la totalité du module `numpy` par la commande

```
from numpy import *
set_printoptions(precision = 4, suppress=True)
```

La seconde instruction assure que l'affichage se fera avec 4 chiffres significatifs et remplace par 0 les nombres trop petits. Cela assure la lisibilité. On utilisera la classe `array` pour construire les vecteurs et les matrices. On cherchera les fonctions utiles de la librairie dans le memento et dans la documentation de `numpy`[1]. L est un paramètre global. On pourra faire les tests avec $L = 5$.

- ☞ Créer un vecteur `V` de taille L et d'éléments $V_j = \sqrt{\frac{2}{L+1}} \sin\left(\frac{\pi}{L+1} j\right)$ pour $j = 1, \ldots, L$.
- ☞ Créer la matrice carrée $L \times L$ avec des -2 sur la diagonale et des 1 autour selon

$$W = \begin{pmatrix} -2 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & \ddots & 0 \\ 0 & 1 & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & -2 & 1 \\ 0 & 0 & 0 & 1 & -2 \end{pmatrix} \tag{1}$$

- ☞ Calculer le produit WV et diviser le résultat par les composantes de V. Qu'en déduisez-vous ?
- ☞ Calculer les valeurs propres et les vecteurs propres de W. Qui est V ?
- ☞ Soit U la matrice des vecteurs propres et D la matrice diagonale contenant les vecteurs propres. Vérifier les deux relations $W = UDU^{-1}$ et $D = U^{-1}WU$.

Application à la dynamique d'une chaîne d'oscillateurs couplés

La matrice W rentre dans la solution des équations de la dynamique d'une chaîne d'oscillateur couplés. On considère L masses identiques m, couplées par des ressorts identiques de raideur k

1. Les opérations demandées tiennent en une ou deux lignes avec des fonctions bien choisies.

et alignées selon l'axe des x. On repère leurs écarts à leur positions d'équilibre par les variables X_j. Les extrémités sont fixées de sorte qu'on prend par convention $X_0 = X_{L+1} = 0$. En prenant $k/m = 1$, les équations de Newton prennent alors la forme

$$\ddot{X}_j = X_{j+1} + X_{j-1} - 2X_j \quad \text{pour } j = 1, \ldots, L \; . \tag{2}$$

En introduisant le vecteur X de dimension L et de composantes X_j, ce système d'équations se met sous la forme $\ddot{X} = WX$ avec W définie à l'équation (1). On résoud ce système en posant $Y = U^{-1}X$ qui donne $\ddot{Y} = DY$. Comme les valeurs propres de W sont négatives, on peut réécrire D sous la forme

$$D = \begin{pmatrix} -\omega_1^2 & 0 & \ldots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & -\omega_L^2 \end{pmatrix} \; . \tag{3}$$

où les ω_j sont les pulsations propres des modes. On choisit des conditions initiales avec vitesse nulle pour toutes les masses ce qui donne $\dot{X}^0 = \dot{X}(t=0) = \dot{Y}^0 = \dot{Y}(t=0) = 0$. Dès lors, les composantes de $Y(t)$ sont données par $Y_j(t) = Y_j^0 \cos(\omega_j t)$.

- ☞ On choisit comme condition initiale X^0 tel que $X_1^0 = 0.2$, $X_2^0 = 0.1$ et les autres $X_j^0 = 0$. Calculer la Y^0 en fonction de X^0.
- ☞ Pour un seul temps $t = 1.0$, écrire en une ligne le calcul de $X(t)$.
- ☞ Introduire un tableau `t` des temps de 0 à 10 avec $N = 101$ points, ainsi qu'une matrice `allX` de taille $L \times N$ pour stocker tous les $X(t)$. Remplir cette matrice.
- ☞ Tracer finalement l'évolution temporelle $X_j(t)$ de l'ensemble des positions pour $L = 5$ puis $L = 10$.

Utilisation de la librairie via l'environnement de développement `Spyder`

L'environnement de développement `Spyder` vous permet d'écrire des fichiers `.py` contenant des scripts ou des modules tout en ayant accès à une ligne de commande `IPython` pour les exécuter et les importer. Voici quelques commandes utiles pour l'interpréteur :

Pour afficher la liste des commandes `IPython` utiles

```
quickref
```

Fichiers manipuler et naviguer dans l'arborescence des dossiers et des fichiers :

```
pwd # affiche le dossier courant
cd  # remonte à la racine du nom d'utilisateur
cd MonDossier/ # se rend dans le dossier "MonDossier/"
               # (penser à utiliser la touche TAB pour la complétion automatique)
cd ..          # remonte d'un cran dans l'arborescence
mkdir MonDossier/ # crée le dossier "MonDossier/"
rmdir MonDossier/ # crée le dossier "MonDossier/"
ls    # affiche le contenu du dossier
copy fichier1 fichier2 # copie de fichiers
more fichier # affiche le contenu du fichier dans la console
```

Exécuter des scripts On peut faire `f5` qui exécute la feuille active de l'éditeur.

```
run script.py   # exécute script.py
rerun  # réexécute la dernière commande
timeit [math.sin(x) for x in range(5000)] # évalue le temps d'exécution d'une commande
timeit %run script.py # exécute plusieurs fois le script
                      # et évalue le temps d'exécution
```

Naviguer dans l'historique des commandes

- ☞ avec les 4 flèches.
- ☞ `Ctrl-a` et `Ctrl-e` pour atteindre les début et fin de ligne.
- ☞ `Ctrl-k` pour couper la fin de la ligne.
- ☞ `Ctrl-y` pour coller ce qu'il y a dans le presse-papier.
- ☞ utiliser `Tab` pour la complétion des noms.
- ☞ Créer un fichier `LinAlg.py` dans lequel vous copier votre classe `Vecteur`.
- ☞ Importer votre module `LinAlg.py` et faites quelques tests de fonctionnement.