ENSAE

Ecole Nationale de la Statistique et de l'Administration Economique

Initiation à la programmation

Enoncés de Travaux Dirigés

2011-2012

Xavier Dupré

http ://www.xavierdupre.fr/

Table des matières

Table des matières

Chapitre 1

Séances dirigées

1.1 Recherche dichotomique

Les lignes suivantes reprennent les points essentiels quant à l'organisation de cet enseignement.

1. Cours, TD, support de cours
   (a) Il y a 13 séances de TDs dont une notée, pas de cours magistraux
   (b) Le poly du cours est disponible sur Internet : http://www.xavierdupre.fr/mywiki/InitiationPython.
   (c) Sa version papier est disponible à la bibliothèque.
   (d) Les TDs sont accessibles ici : http://www.xavierdupre.fr/enseignement/td_python/python_td_minute/index.html.
2. évaluation :
   – un tutoriel obligatoire après les vacances de Toussaint
   – un TD noté (aux alentours de début décembre) (les annales des précédents examens : http://www.xavierdupre.fr/enseignement/td_python/python_td_simple/index.html)
   – un tutoriel facultatif plus facile avant Toussaint pour les élèves qui débutent et qui souhaitent avoir une note supplémentaire qui aura le même poids que les deux autres
3. Concernant le langage *Python*
   (a) le site officiel http://www.python.org/
   (b) Pourquoi *Python* ?
      – utilisable pour beaucoup d'usages (Web, calcul, interface graphique)
      – nombreuses librairies disponibles sur Internet (calcul scientifique, visualisation, internet, base de données, ...)
      – gratuit, facilement installable (librairie de même)
      – usage croissant

Plan prévisionnel :

Séance	Date	Notions informatiques	Notions algorithmiques ou sujet abordé
1		variables, boucles, tests	recherche dichotomique (programme à trou)
2		fonctions	tri
3		fichiers texte, listes	cryptage, décryptage (Vigenère)
4		dictionnaire	histogramme (déviner la langue d'un texte)
5		classes	carré magique
6		classes	début des graphes : parcours, successeur, ancêtres
7		modules, `numpy`	calcul matriciel, valeurs propres, inversion, utilisation des valeurs pour obtenir des variables normales non-corrélées
8		`numpy`	régression linéaire
9		`re`	discours des présidents [1]
10		séance de préparation aux TDs noté	
11		**TD noté**	**TD noté**
12		graphes, matrices, classes	plus court chemin dans un graphe, connexions avec l'algorithme de Viterbi (chaînes de Markov)
13		interfaces graphiques + `matplotlib`	visualisation d'un fichier texte X,Y,labels

Premier plan prévisionnel.

1.1.1 Enoncé

Abordé lors de cette séance	
programmation	variables, opérations, boucles, tests
algorithme	recherche dichotomique

Au cours de cette première séance, le chargé de TD vous fera découvrir les bases du langage *Python* et la façon d'écrire des programmes à l'ENSAE, notamment l'usage de l'éditeur de texte *SciTe* utilisé à l'ENSAE et de ses deux parties d'écran, l'une pour le programme, l'autre pour son exécution.

L'objectif de cette séance est aussi de programme la recherche dichotomique [2] qui est un algorithme simple mais qu'il est souvent utile de connaître pour comprendre pourquoi certaines façons de faire sont plus efficaces que d'autres.

Première demi-heure : premiers pas .

Pour ce premier TD et cette première demi-heure, l'objectif est de réussir à exécuter le simple programme suivant :

```
x = 3
y = x + 5
print x
print x,y
# commentaire
```

1. http://freakonometrics.blog.free.fr/index.php?post/2011/05/12/De-quoi-parle-un-president-francais-le-31-decembre, http://freakonometrics.blog.free.fr/public/data/voeuxpresidents.tar
2. http://fr.wikipedia.org/wiki/Dichotomie

Parvenir à la réalisation de cet objectif nécessite l'ouverture d'un éditeur de texte, l'écriture du programme, la conservation de ce programme sous forme d'un fichier texte, le nom de ce fichier texte devant se terminer par .py, l'interprétation de l'instruction `x = 3`, l'interprétation de l'instruction `print`, la correction des erreurs éventuels, l'exécution du programme, l'utilisation de commentaires...

Seconde demi-heure : variable, type .

On cherche à jouer avec les différents types de variables et certains opérations qu'on peut faire avec ou non.

```
i = 3                    # entier
r = 3.3                  # réel
s = "exemple"            # chaîne de caractères
c = (4,5)                # couple de valeurs (ou tuple)
l = [ 4, 5, 6.5]         # listes de valeurs ou tableaux
x = l [0]                # obtention du premier élément de la liste l
d = { "un":1, "deux":2 } # dictionnaire de valeurs
y = d ["un"]             # obtention de la valeur associée à l'élément "un"
couple = 4, 5            # collage de deux valeurs en un couple ou (tuple)
print type (l)           # on affiche le type de la variable l
mat = [ [0,1], [2,3] ]   # liste de listes
print mat [0][1]         # obtention du second élément de la première liste
n = None                 # None signifie que la variable existe mais qu'elle ne contient rien
                         # elle est souvent utilisé pour signifier qu'il n'y a pas de résultat
                         # car... une erreur s'est produite, une liste était vide...
```

Chaque lettre ou groupe de lettres désigne une variable, c'est une lettre qui permet de la manipuler quelque soit le contenu qui lui est affecté. Dans l'exemple précédent, `i` désigne 3 mais :

```
i = 3
i = i + 5
```

La variable `i` va d'abord correspondre à 3 puis 8 car on lui affecte une nouvelle valeur déduite de la première. L'ajout du mot-clé `print` permet de voir le résultat si celui-ci est syntaxiquement correct. Sans celui-ci, le programme peut marcher ou non mais rien n'apparaîtra dans la seconde partie d'écran de *SciTe*.

On cherche maintenant à voir comment se comporte les différents types de variables avec différentes opérations. `x` et `y` désigne des objets de même type.

```
print x + y          # addition
print x - y          # soustraction
print x / y          # division
print x * y          # multiplication
print x % y          # modulo
print x == y         # égalité
print x < y          # inférieur
print x <= y         # inférieur ou égal
print min (x,y)      # minimum
print max (x,y)      # maximum
print zip (x,y)      # zip ( [4,5], ["a", "b"] ) donne [ (4,"a"), (5,"b") ]
                     # de deux listes, on passe à une sorte de matrice
print True and False # et logique
print True or False  # ou logique
print (5 < 4) or (5 > 4) # condition
```

Ou encore de manipuler des variables :

```
print -x             # opposé
print len ( [ 4,5] ) # obtention du nombre d'éléments d'une liste
print not False      # négation
```

Les opérations marchent aussi parfois lorsque `x` et `y` ne sont pas de même nature, de type différent :

```
print [ 3,4 ] * 5
print "azerty" * 4
print 4 * "azerty"
print [ 3,4 ] + (3,4)
```

1) Le tableau suivant reprend tous les types standards du langage *Python*. Pourriez-vous remplir les deux cases correspondant à une opération entre les types `int` et `tuple` dans cet ordre puis dans l'autre. Est-ce la même liste ?

	None	bool	int	float	str	tuple	list	dict
None	==							
bool	==	and or						
int			+ - * / min max % == < <=	+ - * / min max % == < <=				
float								
str			*					
tuple								
list								
dict								

2) Le langage *Python* propose des conversions d'un type à un autre. Ainsi, il est possible de convertir un nombre en une chaîne de caractères. Quelques sont les lignes parmi les suivantes qui n'ont pas de sens selon le langage *Python* :

```
print int (3.4)
print list ( (4,5) )
print tuple ( [ 4,5] )
print dict ( [4,5] )
print str ( { "un":1, "deux":2 } )
```

3) Une chaîne de caractères (`str`) contient toujours du texte. Par exemple, si on veut afficher le message, quelle sont les lignes valides parmi les suivantes :

```
x = 1.2
y = 1.2 * 6.55
print "Le prix de la baguette est ", x, "francs."
print "Le prix de la baguette est " + x + "francs."
print "Le prix de la baguette est " + str(x) + "francs."
```

A chaque fois qu'on affiche un résultat numérique, il est implicitement converti en chaîne de caractères.

4) Que vaut l'expression $(0,1) <= (0,2)$? Une idée de comment ce résultat est construit ?

Troisième demi-heure : boucles

Le tri d'un tableau est une fonctionnalité indispensable et présente dans tous les langages de programmation. En *Python*, on écrira pour une liste :

```
l = [ 4, 6, 3, 4, 2, 9]  # n'importe quelle liste au hasard
l.sort ()
print l                  # le programme affiche la liste triée
```

5) En utilisant la documentation *Python* ou Internet ou un moteur de recherche, trouver comment classer des éléments en sens décroissant. L'instruction `help` dans le programme lui-même retourne l'aide associée à ce qu'il y a entre parenthèses comme `help(l.sort)`. Toute requête sur un moteur de recherche de type *python <élement recherché>* retourne souvent des résultats pertinents.

6) Le programme suivant affiche tous les éléments du tableau un par un grâce à une boucle `for` :

```
for i in xrange (0, len (l)) :
    print l [i]
```

Il utilise une boucle pour parcourir le tableau du début à la fin. Utilisez l'aide concernant la fonction[3] `xrange` pour parcourir le tableau en sens inverse.

7) On souhaite écrire un petit programme qui vérifie que le tableau est trié. Il suffit de compléter le programme suivant :

```
resultat = ...
for i in xrange (0, ...) :
    if ... > ... :
        resultat = False
        break
print "le tableau est-il trié : ", resultat
```

8) Il existe une autre boucle `while`. Complétez le programe suivant pour obtenir la même chose qu'à la question précédente :

```
resultat = ...
i = ...
while i < ...
    if ... > ... :
        resultat = False
        break
print "le tableau est-il trié : ", resultat
```

Quatrième demi-heure : recherche dichotomique

La recherche dichotomique[4] consiste à chercher un élément `e` dans un tableau trié `l`. On cherche sa position :

- On commence par comparer `e` à l'élément placé au milieu du tableau d'indice `m`, s'ils sont égaux, on a trouvé,
- s'il est inférieur, on sait qu'il se trouve entre les indices 0 et $\texttt{m} - 1$,
- s'il est supérieur, on sait qu'il se trouve entre les indices $\texttt{m} + 1$ et la fin du tableau.

3. Une fonction est différente d'une variable, on la reconnaît grâce aux parenthèses qui suivent un nom. `min`, `max`, `len`, `xrange` sont des fonctions. Entre parenthèses, on trouve les variables dont la fonction a besoin pour fonctionner. Chaque fonction effectue des traitements simples ou complexes mais surtout qu'on souhaite répéter plusieurs fois simplement.

4. http://fr.wikipedia.org/wiki/Dichotomie

Avec une comparaison, on a déjà éliminé une moitié de tableau dans laquelle on sait que `p` ne se trouve pas. On applique le même raisonnement à l'autre moitié pour réduire la partie du tableau dans laquelle on doit chercher.

9) Il ne reste plus qu'à écrire le programme qui effectue cette recherche. On cherche à déterminer la position de l'élément `e` dans la liste `l`. On utilise les indications suivantes :
– il y a une boucle, de préférence `while`
– il y a deux tests
– la liste des variables pourrait être `e`, `l`, `a`, `b`, `m`

10) Que se passe-t-il lorsqu'on cherche un élément qui ne se trouve pas dans le tableau ?

Pour aller plus loin ou pour ceux qui ont fini plus tôt .

11) Si deux joueurs de tennis ont autant de chance l'un que l'autre de gagner un point, combien de points a en moyenne le vainqueur d'un tie-break ? Ecrire un programme qui permette de répondre à la question. Le module `random` permet de générer des nombres aléatoires.

12) Que se passe-t-il si chaque joueur a 75% de chance de gagner un point sur son service ?

1.1.2 Correction

1)

fin correction TD 1.1.1 □

1.2 Tri

1.2.1 Enoncé

Abordé lors de cette séance	
programmation	fonctions, listes
algorithme	tri bulle

L'objectif de ce TD est de programmer un tri bulle. Ce tri sera décomposé en plusieurs petites *fonctions*. La première demi-heure sera utilisée pour définir les fonctions.

Première demi-heure : fonctions .

Une fonction est un moyen d'isoler une partie de programme qu'on répète souvent. Pour calculer une moyenne pondérée sur deux liste de notes, sans fonction :

```
# première moyenne
poids = [1, 2, 1, 2]
notes = [12, 14, 8, 9]
s = 0                              #
w = 0                              #
for p,n in zip (poids, notes) :    # calcule du résultat
    s += p*n                       #
    w += p                         #
resultat = s * 1.0 / w    # * 1.0 évite le problème dus aux divisions entières
```

```
                        # 1/2 --> 0,   1 * 1.0 / 2 --> 0.5

# seconde moyenne
poids = [1, 1, 2]
notes = [12, 14, 8]
s = 0
w = 0
for p,n in zip (poids, notes) :
    s += p*n
    w += p
resultat = s * 1.0 / w
```

Le programme est plutôt redondant. On utilise une fonction pour éviter cela :

```
def moyenne_ponderee (poids, notes) :
    s = 0
    w = 0
    for p,n in zip (poids, notes) :
        s += p*n
        w += p
    return s * 1.0 / w        # le mot-clé return désigne le résultat que la fonction calcule

poids = [1, 2, 1, 2]
notes = [12, 14, 8, 9]
resultat = moyenne_ponderee (poids, notes)


poids = [1, 1, 2]
notes = [12, 14, 8]
resultat = moyenne_ponderee (poids, notes)
```

Le second programme est plus court. Quelqus termes :

- **déclaration** : on déclare une fonction avec le mot-clé `def`, il ne faut oublier de décaler les lignes qui suivent, c'est la seule façon d'indiquer que ces lignes font partie de la fonction et non du reste du programme.
- **paramètres** : `poids`, `notes`, ce sont les entrées de la fonction, ils apparaissent entre parenthèses lors de la déclaration
- **résultat** : c'est l'objectif de la fonction, calculer ce résultat indiqué par le mot-clé `return`.

1) Que se passe-t-il si les tableaux `poids` et `notes` n'ont pas la même longueur ?

2) La fonction suivante prend comme entrée un tableau et le retourne trié.

```
def trie_tableau (tableau) :
    tableau.sort ()
    return tableau

tab = [ 3,5,4]
tri = trie_tableau (tab)
print tab
print tri
```

Qu'affiche les deux dernières lignes ? Cela vous paraît-il étrange ?

3) Le langage *Python* ne sait pas faire grand chose mais il dispose de nombreux modules ou extensions. Chaque ligne `import ...` signifie que le programme va utiliser une extension du langage. Par exemple, on voudrait que la fonction `trie_tableau` retourne un tableau trié mais qui ne modifie pas le tableau envoyé à la fonction. A l'aide de la documentation, utilisez la fonction `copy` du module `copy`.

4) Que fait le programme suivant :

```
def trie_tableau (tableau) :
    tableau.sort ()
    return tableau

tab = ( 3, 5, 4 )
tri = trie_tableau (tab)
print tab
print tri
```

En langage *Python*, les types `None`, `int`, `float`, `str`, `tuple` sont passés par valeur à une fonction : les modifier à l'intérieur de la fonction n'a pas d'impact à l'extérieur. Les types `list` et `dict` sont passés par adresse : les modifier à l'intérieur de la fonction a un impact à l'extérieur.

Seconde demi-heure : la fonction `sort`

5) On considère la liste suivante :

```
l = [ 4, 5, 3, 7, 7, 9, 10, 0 ]
```

L'instruction suivante permet de trier la liste :

```
l.sort ()
```

L'instuction `print l` permet d'afficher le résultat et de vérifier que la liste est bien triée. Rechercher comment trier en sens inverse sur Internet via un moteur de recherche ou en utilisant l'aide *Python*(avec l'instruction `help`).

6) Que donne la même instruction `sort` sur l'exemple suivant :

```
l = [ 4, 5, "ee", 7, "aa", 7, 9, 10, 0, "gg" ]
```

Le résultat est-il intuitif ?

7) Exécuter les deux instructions suivantes :

```
print l [1]
print l [ len (l)-1 ]
```

Le premier résultat correspond au premier ou au second élément. Quel est l'indice du premier élément du tableau ? Le second résultat correspond à un autre élément tableau `l`. Quel est l'indice du dernier élément ?

8) Il existe de nombreux algorithmes de tri. On les distingue selon deux critères :

1. Leur coût ou le nombre de comparaisons nécessaire pour trier un tableau de taille n.
2. Le nombre de copies du tableau initial.

Dans le cas général, on suppose qu'on sait ordonner deux éléments et rien de plus. Pour un tableau de taille n, il est prouvé qu'il n'est pas possible d'effectuer moins de $n \ln n$ comparaisons. Nous allons voir un tri plus lent ($\frac{n^2}{2}$ comparaisons) mais plus simple : le **tri bulle**. On commence par la séquence logique suivante :

1. On commence à $i = 0$

2. On compare les éléments d'indice i et $i+1$. Si le second est plus petit que le premier, on permute les deux nombres.
3. On se décale en suite d'un cran à $i+1$. Si $i+1 < n-1$, on retourne à l'étape 2.

On commence par écrire une fonction qui permute les deux éléments `i` et `j` d'un tableau :

```
def permutation (l, i, j) :
    ...
```

Troisième demi-heure : tri bulle

On cherche à décomposer le problème en des fonctions aussi petites que possibles puis à les assembler par la suite pour former la séquence décrite ci-dessus.

9) La fonction suivante doit comparer deux éléments consécutifs et les permuter s'ils sont dans le mauvais ordre.

```
def comparaison_permutation (l, i) :
    ...
```

Il faudra utiliser la fonction `permutation` implémentée à la question précédente.

10) Petite pause : exécuter les lignes suivantes. Quel est le dernier entier affiché ?

```
for i in xrange (0, 10) :
    print i
```

11) Etape suivante : on appelle la fonction `comparaison_permutation` pour tous les entiers de i allant de 0 à `len(l)` $-$ `1`.

```
def plusieurs_comparaison_permutation (l) :
    ...
```

Après l'exécution de la fonction `plusieurs_comparaison_permutation`, peut-on considérer le tableau trié ? Pourquoi ?

Quatrième demi-heure : fin du tri bulle

12) On écrit une troisième fonction pour appeler n fois la fonction `plusieurs_comparaison_permutation` :

```
def n_fois_plusieurs_comparaison_permutation (l) :
    ...
```

Peut-on considérez le tableau trié ? Pourquoi ?

13) Comptez le nombre de comparaisons de la précédente méthode ? Peut-on faire mieux et comment ?

14) Modifiez les fonctions précédentes pour diminuer le nombre de comparaisons ?

Pour aller plus loin ou pour ceux qui ont fini plus tôt

L'algorithme de tri est maintenant fonctionnel. On veut néanmoins l'altérer.

15) Comment modifier simplement l'algorithme pour trier dans l'autre sens (sens décroissant) ?

16) Comment modifier simplement l'algorithme pour effectuer une permutation aléatoire de la liste d'éléments ?

17) Comment modifier simplement l'algorithme pour placer les éléments pairs en premier et les éléments impairs en dernier ?

18) Au lieu de parcourir les éléments dans le sens des indices croissants, on les parcourt dans le sens des indices décroissants ? Que cela changerait-il ?

19) Et si on tirait au hasard les deux éléments consécitufs à comparer ? C'est-à-dire qu'au lieu d'utiliser des boucles de 0 à $n-1$, on tire au hasard un élément i entre 0 et $n-2$ pour comparer i et $i+1$ puis les inverser si nécessaire. Est-ce que cela marcherait ?

Remarques .

Un tutoriel vous permettra d'appréhender le tri par fusion qui effectue $n \ln n$ comparaisons.

1.2.2 Correction

1)

fin correction TD 1.2.1 □

1.3 Cryptage de Vigenère

1.3.1 Enoncé

Abordé lors de cette séance	
programmation	fonction, listes, fichiers, dictionnaire
algorithme	histogramme

César en son temps utilisait déjà le cryptage pour transmettre ses messages [5]. Il remplaçait chaque lettre par sa suivante. A devenait B, B devenait C, et ainsi de suite. Même sans connaître le décalage, décrypter un tel code est simple puisqu'il suffit de compter l'occurrence de chaque lettre du message. On peut raisonnablement affirmer que la lettre la plus fréquente est la lettre "e" ou, si l'auteur est Georges Pérec, une voyelle. On supposant que le décalage entre "e" et la lettre la plus fréquence dans le message crypté, il devient facile de deviner la correspondance entre lettres.

Certains cryptographes plus subtiles eurent tôt fait d'agrandir l'alphabet de chiffrement et de faire en sorte qu'une lettre aurait d'autant plus de représentations cryptées qu'elle serait fréquente. Toutefois, les cryptanalystes ripostèrent par l'analyse des fréquences non plus des lettres mais des couples de lettres.

Pour résister à ces méthodes de décryptage, Blaise de Vigenère inventa aux alentours de 1586 (date de la publication de son livre *Le traité des chiffres* [6]) un code qui ne se trahissait plus par l'étude des lettres ou des couples de lettres les plus fréquents. Cette méthode de chiffrement résista jusqu'en 1854, année où Babbage [7] élabora une méthode qui permettait de casser ce code.

5. http://fr.wikipedia.org/wiki/Chiffrement_par_d%C3%A9calage
6. http://fr.wikipedia.org/wiki/Chiffre_de_Vigen%C3%A8re
7. http://fr.wikipedia.org/wiki/Charles_Babbage

Première demi-heure : listes, boucles, fonctions .

Mais tout d'abord voyons en quoi consiste le code de Vigenère et cela commence par la description du **carré de Vigenère** : un alphabet recopié et décalé d'un cran vers la gauche à chaque ligne.

```
ABCDEFGHIJKLMNOPQRSTUVWXZ
BCDEFGHIJKLMNOPQRSTUVWXZA
CDEFGHIJKLMNOPQRSTUVWXZAB
DEFGHIJKLMNOPQRSTUVWXZABC
EFGHIJKLMNOPQRSTUVWXZABCD
FGHIJKLMNOPQRSTUVWXZABCDE
GHIJKLMNOPQRSTUVWXZABCDEF
HIJKLMNOPQRSTUVWXZABCDEFG
IJKLMNOPQRSTUVWXZABCDEFGH
JKLMNOPQRSTUVWXZABCDEFGHI
KLMNOPQRSTUVWXZABCDEFGHIJ
LMNOPQRSTUVWXZABCDEFGHIJK
MNOPQRSTUVWXZABCDEFGHIJKL
NOPQRSTUVWXZABCDEFGHIJKLM
OPQRSTUVWXZABCDEFGHIJKLMN
PQRSTUVWXZABCDEFGHIJKLMNO
QRSTUVWXZABCDEFGHIJKLMNOP
RSTUVWXZABCDEFGHIJKLMNOPQ
STUVWXZABCDEFGHIJKLMNOPQR
TUVWXZABCDEFGHIJKLMNOPQRS
UVWXZABCDEFGHIJKLMNOPQRST
VWXZABCDEFGHIJKLMNOPQRSTU
WXZABCDEFGHIJKLMNOPQRSTUV
XZABCDEFGHIJKLMNOPQRSTUVW
ZABCDEFGHIJKLMNOPQRSTUVWX
ABCDEFGHIJKLMNOPQRSTUVWXZ
```

1) La création de ce carré n'est pas nécessaire, il est possible passer directement à la question suivante sans créer ce carré sous forme d'une liste de 26 chaînes de caractères. On pourrait bien sûr copier/coller tel quel ce carré dans le programme ou le construire. Pour cela, on s'aidera de l'exemple suivant :

```
s = ""
for i in xrange (0,26) :
    s += chr (65+i)
print s
```

Il suffit de vous en inspirer pour créer une liste de 26 chaînes de caractères, toutes décalées les unes par rapport aux autres.

2) Pour chiffrer un message, il faut une clé qui est un mot (existant ou non) en majuscules, par exemple CODE. Il ne manque plus qu'un message à coder : *Les codes secrets ont joué un rôle discret mais important dans l'Histoire*. Tout d'abord, on va juxtaposer le messsage à coder et la clé maintes fois répétées :

A chaque lettre du message correspond ainsi une lettre de la clé et cette lettre va déterminer, à l'aide du carré de Vigenère, son équivalent crypté. En effet, la première lettre du message L est associée à la lettre C de la clé, cette première lettre est remplacée par celle se trouvant à l'intersection de la ligne commençant par C et de la colonne commençant par L, cela donne N. On continue comme cela jusqu'à la fin du message.

```
lescodessecretsontjoueunrolediscretmaisimportantdanslhistoire
CODECODECODECODECODECODECODECODECODECODECODECODECODECODECODEC
NSV...
```

L'objectif de cette question est d'écrire une fonction qui prend une lettre du message, une lettre de la clé et qui retourne la lettre codée. Il est possible que la fonction `ord` vous soit utile par la suite, c'est la fonction réciproque de la fonction `chr`.

```
def lettre_codee (lettre_message, lettre_cle, carre_vigenere) :
    ...
    return lettre_codee
```

Pour ceux qui ont choisi de passer la première question, il faudra utiliser les deux fonctions suivantes :

```
print chr(65)  # retourne la lettre correspondant au code 65 --> A
print ord("A") # retourne le code associé à la lettre "A" --> 65
```

Remarque 1.1 : Lettre minuscules et majuscules
Le language *Python* fait la différence entre les lettres minuscules et majuscules, elles ont des codes différents.

Seconde demi-heure : fonctions

3) Il suffit de parcourir toutes les lettres du texte à coder pour terminer le cryptage en utilisant la fonction précédente. La fonction `len` permet de retourner la longueur d'à peu près n'importe quoi, d'une liste, et aussi d'une chaîne de caractères.

```
def cryptage (message, cle, carre_vigenere) :
    ...
    return message_code
```

4) Ecrire la fonction réciproque de la précédente pour décrypter le texte.

Troisième demi-heure : fichiers

On souhaite appliquer le cryptage sur un texte plus grand en téléchargeant un texte depuis le site Gutenberg [8] comme celui-ci de Gide [9] ou un traité de géométrie en latex [10] ou pour les plus courageux une fable de la fontaine [11].

Deux options sont possibles, la première consiste d'abord à télécharger le fichier puis à l'enregistrer dans un fichier à côté du programme *Python* et enfin à le lire depuis le programme *Python* avec les quelques lignes suivantes :

```
f = open ("nom_de_fichier", "r")
text = f.read ()
f.close ()
```

Plus d'explication au chapitre 7 du support de cours [12]. L'autre solution est de télécharger directement le texte depuis le site :

8. http://www.gutenberg.org/
9. http://www.gutenberg.org/files/30696/30696-h/30696-h.htm
10. http://www.gutenberg.org/files/26400/26400-t/26400-t.tex
11. http://www.gutenberg.org/files/20971/mp3/20971-02.mp3
12. http://www.xavierdupre.fr/mywiki/InitiationPython

```
import urllib2
f = urllib2.urlopen ("http://www.gutenberg.org/files/30696/30696-h/30696-h.htm")
text = f.read ()
```

Si ce code ne fonctionne pas, il faudra utiliser le suivant :

```
import urllib2
req = urllib2.Request("http://www.gutenberg.org/files/30696/30696-h/30696-h.htm", headers={'User-Agent' : "Magic Brows
f = urllib2.urlopen (req)
text = f.read ()
```

5) Choisir l'une des deux solutions (chargement depuis un fichier ou depuis un url), puis appliquer le cryptage et décryptage. Est-ce le programme fonctionne et est-ce qu'il retourne le texte de départ ?

6) On souhaite supprimer tous les caractères indésirables. Que proposez-vous ?

Quatrième demi-heure : décryptage et dictionnaire

L'objectif de cette dernière partie est de deviner la longueur de la clé à partir d'un message crypté. On suppose seulement qu'on connaît la langue du texte chiffré.

7) Quel est la lettre la plus fréquente en français ? Expliquez pourquoi cette information est utile pour casser le code de César mais pas le code de Vigenère ?

8) On utilise un dictionnaire pour compter la fréquence de chaque lettre du texte chiffré. Dans ce cas, un dictionnaire nous permet d'associer un nombre à une lettre. Chaque nombre est *indicé* par une lettre.

```
h = { }
h ["A"] = 50
h ["B"] = 50
print h ["A"]
print h
```

On cherche donc à construire un histogramme.

```
def histogramme_lettre (texte) :
    h = { }
    ...
    return h
```

On vérifie que cette histogramme ne permet pas de deviner la longueur de la clé.

9) Babbage s'est dit qu'un groupe de trois lettres consécutives avaient toutes les chances, à chaque fois qu'il apparaissait dans le message chiffré, d'être la conséquence du chiffrement des mêmes lettres du message avec les mêmes lettres de la clé. Pour un groupe de quatre lettres, c'est encore plus probable. Par conséquent, l'espacement entre deux mêmes groupes de lettres chiffrées est un multiple de la longueur de la clé. Par exemple, si la répétition d'un groupe est espacée de 30 lettres, puis celle d'un autre de 25, le plus grand diviseur commun de 25 et 30 est 5. La clé possède donc dans ce cas 5 lettres.

Modifier la fonction précédente pour compter la fréquence de tous les triplets de lettres consécutifs dans le texte. Pourquoi un dictionnaire est-il plus indiqué dans ce cas plutôt qu'une liste ?

10) Le triplet le plus fréquent ne donne pas directement d'indication sur la longueur de la clé de chiffrement. Toutefois, si l'explication ci-dessus vous paraît claire, une fois qu'on a déterminé le triplet de lettres le

plus fréquent, comment en déduit-on une longueur possible pour la clé ? Il ne s'agit pas d'implémenter l'algorithme mais plus de le décrire.

Pour aller plus loin ou pour ceux qui ont fini plus tôt .

11) Implémenter en une fonction le raisonnement qui vous permet de proposer une longueur possible pour la clé ?

12) Une fois la longueur connue, comment déterminer la clé ?

1.3.2 Correction

fin correction TD 1.3.1 □

1.4 Deviner la langue d'un texte

1.4.1 Enoncé

Abordé lors de cette séance	
programmation	fonction, listes, fichiers, dictionnaire
algorithme	histogramme, score

L'objectif est de distinguer un texte anglais d'un texte français sans avoir à le lire. Le premier réflexe consisterait à chercher la présence de mots typiquement anglais ou français. Cette direction est sans doute un bon choix lorsque le texte considéré est une œuvre littéraire. Mais sur Internet, les contenus mélangent fréquemment les deux langues : la présence de tel mot anglais n'est plus aussi discriminante. Il n'est plus aussi évident d'étiqueter un document de langue anglaise lorsque les mots anglais sont présents partout.

On ne cherche plus à déterminer la langue d'un texte mais plutôt la langue majoritaire. Il serait encore possible de compter les mots de chacune des langues à l'aide d'un dictionnaire réduit de mots anglais et français. La langue majoritaire correspondrait à celle dont les mots sont les plus fréquents. Mais construire un dictionnaire est d'abord fastidieux. Ensuite, il faudrait que celui-ci contienne des mots présents dans la plupart des textes. Il faudrait aussi étudier le problème des mots communs aux deux langues. Pour ces raisons, il paraît préférable d'étudier d'abord une direction plus simple quitte à y revenir plus tard.

Cette idée plus simple consiste à compter la fréquence des lettres. On s'attend à ce que certaines lettres soient plus fréquentes dans un texte anglais que dans un texte français.

Première demi-heure : fichiers, histogramme .

1) On s'inspire de ce qui a été fait au TD précédent : télécharger un texte [13] et le lire depuis un programme *Python*. C'est-à-dire écrire une fonction qui prend comme argument un nom de fichier et qui retourne une chaîne de caractères.

2) Toujours en s'inspirant du TD précédent, construire un histogramme comptant les occurrences de chaque lettre dans ce texte. C'est-à-dire écrire une fonction qui prend comme argument une chaîne de caractères et qui retourne un dictionnaire dont vous choisirez ce que seront les clés et les valeurs.

13. via le site http://www.gutenberg.org/ par exemple

Seconde demi-heure : score

3) Un texte inconnu contient 10 lettres I. Que pouvez-vous en conclure ? Pensez-vous que les fréquences de la lettre I dans un texte long et dans un texte court soient comparables ?

4) Ecrire une fonction qui normalise toutes les valeurs du dictionnaire à un.

```
def normalise (dico) :
    # faire la somme en une ligne avec la fonction sum

    # diviser

    # fin
    return nouveau_dico
```

5) Appliquer votre fonction à un texte anglais et à un autre français, ... Que suggérez-vous comme indicateur pour distinguer un texte français d'un texte anglais ?

Troisième demi-heure : score et seuil

6) Choisir deux langues et calculer votre indicateur pour dix textes de chaque langue.

7) On suppose qu'on regroupe tous ces résultats en une matrice :

1. première colonne : la valeur de votre indicateur
2. seconde colonne : langue du texte

Que proposez-vous pour déterminer un seuil qui départage les deux langues selon votre indicateur ?

8) Comment proposez-vous d'implémenter la méthode que vous suggérez à la question précédente ? Une fonction... Ses paramètres... Son ou ses résultats...

Quatrième demi-heure : score et seuil

9) Implémenter la méthode suggérée. Si le résultat est constitué d'un unique nombre réel, on l'appelle le **score**.

10) Télécharger le fichier suivant : http://www.xavierdupre.fr/enseignement/tutoriels_data/articles.zip et le décompresser dans le même répertoire que votre programme. Ensuite, essayer le programme suivant.

```
import os
for fichier in os.listdir (".") :
    print fichier
```

Ecrire une fonction qui calcule votre score pour tous les textes décompressés ?

11) Est-ce que votre score marche dans tous les cas ?

Pour aller plus loin ou pour ceux qui ont fini plus tôt

12) Le score est ici un nombre unique généré à partir des documents. Admettons que nous disposons de deux scores, la fréquence de la lettre E et celle de la lettre W, comment les combiner pour obtenir un score meilleur que les deux pris séparément ?

Remarques

Ce problème s'inscrit dans un problème plus général de classification [14] [15] ou d'analyse discriminante [16] [17] [18]. Il s'agit de déterminer un score, un indicateur numérique capable de déterminer automatiquement la langue d'un texte sans avoir à le lire. Ces indicateurs ne sont pas infaillibles, il sera toujours possible de le duper particulièrement sur des petits textes mais cela ne veut pas dire que ce score ne pourrait pas être utilisé pour estimer de façon grossière la quantité de pages internet dans chaque langue.

1.4.2 Correction

fin correction TD 1.4.1 □

1.5 Carré Magique

1.5.1 Enoncé

Abordé lors de cette séance	
programmation	classe
algorithme	méthode de construction d'un carré magique

Un carré magique est un carré 3x3 dont chaque case contient un nombre entier et pour lequel les sommes sur chacune des lignes, chacune des colonnes, chacune des diagonales sont égales, soit huit sommes égales. Le plus simple des carrés magiques est celui dont tous les chiffres sont égaux même si un véritable carré magique contient des nombres a priori tous différents [19].

1	1	1
1	1	1
1	1	1

2	7	6
9	5	1
4	3	8

L'autre objectif de ce TD est de comprendre ce que sont les classes [20] au travers de l'exemple des carrés magiques. Le concept de classe est en pratique un moyen de créer ses propres types. Il existe des types prédéfinis en *Python* tels que les listes, les dictionnaires, les entiers... Par défaut, il n'existe pas de type matrice ni de type carré magique et mais les classes permettent de créer facilement ces types qui ne font pas partie du langage.

Première demi-heure : constructeur .

Une classe se crée invariablement de la même manière avec le mot-clé `class` et en ajoutant ce qu'on appelle un constructeur : c'est une fonction attachée à la classe et qui crée la classe.

```
class CarreMagique :
    def __init__ (self, ...) :
        ...
```

14. http://en.wikipedia.org/wiki/Statistical_classification
15. http://en.wikipedia.org/wiki/K-nearest_neighbor_algorithm
16. http://fr.wikipedia.org/wiki/Analyse_discriminante
17. http://fr.wikipedia.org/wiki/Analyse_discriminante_lin%C3%A9aire
18. http://cedric.cnam.fr/~saporta/discriminante.pdf
19. http://fr.wikipedia.org/wiki/Carr%C3%A9_magique_(math%C3%A9matiques)
20. http://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_objet

Dans le cas d'un carré magique cela pourrait donner :

```
class CarreMagique :
    def __init__ (self, a,b,c, d,e,f, g,h,i) :
        self.carre = [ [ a,b,c ], [ d,e,f ], [ g,h,i ] ]
```

Chaque fois que l'on écrit `self.` < `identifiant` >, c'est une façon d'attacher une variable à la classe. Les lignes précédentes définissent une classe, les lignes suivantes définissent une variable de type `CarreMagique`. On utilise aussi le terme d'*instance*.

```
c = CarreMagique ( 1,2,3, 4,5,6, 7,8,9 )
```

Pour afficher la variable ou *membre* `carre` de la classe `CarreMagique`, on écrit :

```
print c.carre
```

1) Modifier la classe de telle sorte que le constructeur accepte une liste plutôt que neuf paramètres. On souhaite, plutôt que de rentrer neuf paramètres à la suite, pouvoir les définir dans une liste. On souhaite par exemple pouvoir écrire cela :

```
l = range (1,10)
c = CarreMagique ( l )
```

2) Qu'affiche le programme suivant :

```
class CarreMagique :
    def __init__ (self, a,b,c, d,e,f, g,h,i) :
        self.carre = [ [ a,b,c ], [ d,e,f ], [ g,h,i ] ]

c = CarreMagique (2,7,6, 9,5,1, 4,3,8)
print c
```

Et celui-là :

```
class CarreMagique :
    def __init__ (self, a,b,c, d,e,f, g,h,i) :
        self.carre = [ [ a,b,c ], [ d,e,f ], [ g,h,i ] ]
    def __str__ (self) :
        return "carre magique : " + str (self.carre)

c = CarreMagique (2,7,6, 9,5,1, 4,3,8)
print c
```

Comment modifier le programme pour que l'instruction `print c` affiche le carré sur plusieurs lignes ?
Une indication avant de commencer :

```
s = "un deux"
print s
s = "un \n deux"  # ajout du caractère \n
print s
```

Seconde demi-heure : méthodes

Une classe inclut des variables ou *membres*, des fonctions ou *méthodes*. Ces fonctions ont toutes en commun de s'appliquer à la classe. Le constructeur est une méthode. L'exemple suivant reprend la classe `CarreMagique` et ajouter une méthode qui compte la somme des nombres sur une ligne :

```
class CarreMagique :
    def __init__ (self, a,b,c, d,e,f, g,h,i) :
        self.carre = [ [ a,b,c ], [ d,e,f ], [ g,h,i ] ]
    def __str__ (self) :
        return "carre magique : " + str (self.carre)

    def somme_ligne (self, i) :                # ajout
        return sum ( self.carre [i] )          # ajout

c = CarreMagique (2,7,6, 9,5,1, 4,3,8)
print c.somme_ligne (0)   # somme des nombres sur la première ligne
```

3) Ajouter une méthode qui fait la somme des nombres sur une colonne.

4) Ajouter une méthode qui fait la somme des nombres sur une diagonale.

5) Ajouter une méthode qui détermine si un carré est magique ou non en complétant la méthode suivante :

```
class CarreMagique :
    ...

    def carre_est_magique (self) :
        li   = [ self.somme_ligne (i) for i in xrange (0,3) ]
        lj   = [ self.somme_colonne (j) for j in xrange (0,3) ]
        diag = [ self.somme_diagonal (k) for k in xrange (0,2) ]
        tout = li + lj + diag
        tout.sort ()
        return ..................
```

Troisième demi-heure : le damier crénelé

Tout d'abord une astuce. Il n'existe pas d'élément d'indice $(1, -1)$ dans une matrice mais cela est tout-à-fait possible avec un dictionnaire :

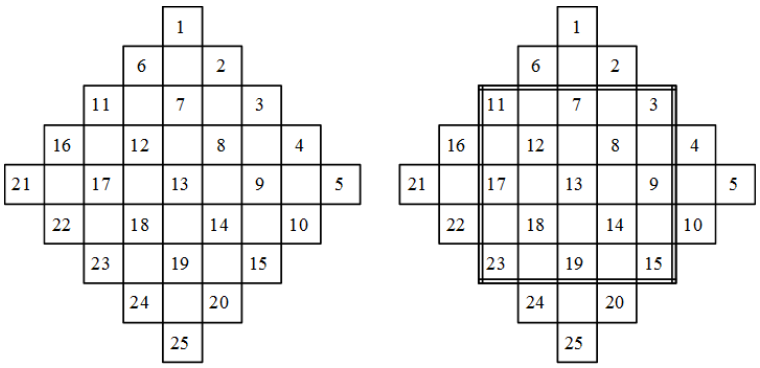
```
d = {}
d [ 1,-1 ] = 1
```

Seconde astuce, l'opérateur modulo ou % :

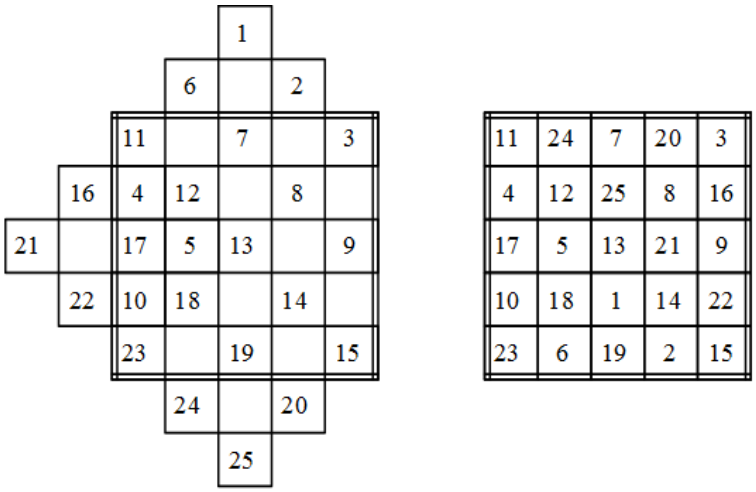
```
for i in xrange (0,9) :
    print i % 3           # affiche 0,1,2, 0,1,2, 0,1,2
```

On souhaite implémenter la méthode du damier crénelé [21]. Les illustrations suivantes sont extraites de Wikipédia :

21. http://fr.wikipedia.org/wiki/Carr%C3%A9_magique_(math%C3%A9matiques)#M.C3.A9thode_du_damier_cr.C3.A9nel.C3.A9



Premières étapes de construction d'un carré magique d'ordre 5. Chaque diagonale allant de gauche à droite comporte un entier unique en ordre croissant. Ensuite, le contour du carré magique final est esquissé.



Dernières étapes de construction d'un carré magique d'ordre 5. Chaque case qui se trouve en-dehors du contour est "glissé" de 5 places en direction du centre. Une fois les déplacements effectués, le carré magique final est complet.

6) Ecrire une méthode de la classe `CarreMagique` qui positionne les nombres de la première étape du damier crénelé dans un dictionnaire :

```
class CarreMagique :
    ...
    def etape_damier_1 (self, dimension) :
        d = { }
        ...
        return d
```

Quatrième demi-heure : le damier crénelé, seconde partie .

7) Ecrire une méthode de la classe `CarreMagique` qui implémente la seconde étape du damier crénelé.

```
class CarreMagique :
    ...
    def damier_crenele (self, dimension) :
```

```
        d = self.etape_damier_1 ( dimension )
        self.carre = [ [ 0 for i in xrange (0, ... ) ] for j in xrange (0, ... ) ]
        ...

c = CarreMagique ([])
c.damier_crenele ()
print c
print c.carre_est_magique ()
```

Si votre algorithme est correct, la dernière ligne devrait afficher `True`.

8) Créer un carré magique selon cette méthode pour toutes les dimensions de 3 à 10. Sont-ils tous magiques ?

Pour aller plus loin ou pour ceux qui ont fini plus tôt .

9) Ecrire une fonction qui permute la première et la dernière colonne du carré magique créé avec la méthode du damier crénelé. Ce carré est-il magique ?

10) Déterminer une condition nécessaire qui permette de conclure que le magique sera magique si on permute la première et la dernière colonne ?

11) Démontrez que cette condition est toujours vérifiée lorsque le carré est construit selon la méthode du damier crénelé ?

12) Est-ce que cela vous inspire une autre permutation sur les colonnes qui aboutit à un nouveau carré magique ? Vérifier le en l'implémentant.

1.5.2 Correction

fin correction TD 1.5.1 □

1.6 Graphes

1.6.1 Enoncé

Abordé lors de cette séance	
programmation	classes
algorithme	algorithme sur les graphes

Les graphes [22] sont un peu partout sans qu'on s'en rende compte car ils sont une façon simple et visuelle de représenter des relations. Un réseau social est un graphe, l'illustration 1.1 est un graphe.

Première demi-heure : construire un graphe .

On considère la liste des relations décrite comme suit [23] :

22. http://fr.wikipedia.org/wiki/Th%C3%A9orie_des_graphes
23. La liste complète est diponible à l'adresse suivante : http://www.xavierdupre.fr/enseignement/td_python/python_td_minute/data/lworld/pairs.txt.

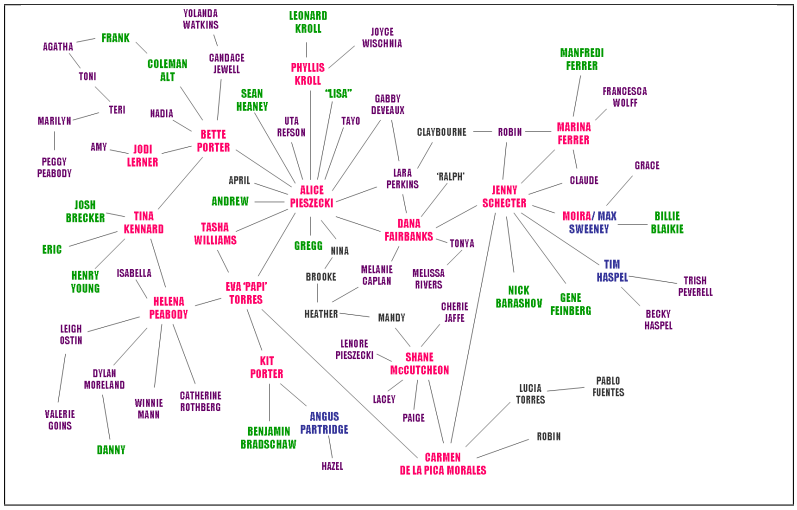


Figure 1.1 : *Image extraite de http://fr.wikipedia.org/wiki/The_L_Word.*

```
coleman,frank
dana,lara
dana,melanie
dana,ralph
danny,dylan
dona,jenny
...
```

Vous pouvez le récupérer directement en utilisant la fonction suivante :

```
def lit_donnees_depuis_un_fichier_ou_internet (fichier) :
    """
    cette fonction lit un fichier où le télécharge depuis un site internet,
    si le fichier n'existe pas, il est téléchargé puis enregistré sur disque
    """
    if not os.path.exists (fichier) :
        url = "http://www.xavierdupre.fr/enseignement/complements/" + fichier
        import urllib2
        f = urllib2.urlopen (url)
        t = f.read()
        f.close()
        f = open(fichier, "w")
        f.write(t)
        f.close()
```

1) Le programme suivant est à compléter (...), il permet de récupérer une matrice à partir du fichier téléchargé :

```
f = open (".......", "r")
lines = f.readlines ()
f.close ()

# la partie après le if sert à enlever les lignes vides
couples = [ l.strip(" \n\r").split ( ... ) for l in lines if ... ]
```

2) Comment compter le nombre de prénoms différents ?

3) On souhaite compter le nombre de relations directes pour chaque individu, que suggérez-vous en vous inspirant de la question précécente ?

Seconde demi-heure : les amis

Si on considère que chaque arc relie deux amis, on souhaite connaître pour chaque personne combien elle a d'amis d'amis. C'est l'objectif des questions suivantes.

4) Si vous avez deux amis qui ont chacun 50 amis, avez-vous nécessairement 100 amis d'amis ?

5) On considère la classe suivante :

```
class Personne :
    def __init__ (self, nom):
        self.nom = nom
        self.amis = []

    def ajout_ami (self, ami) :
        self.amis.append (ami)
```

Ajouter une méthode qui retourne le nombre d'amis.

6) On récupère la liste des prénoms distincts dans un dictionnaire :

```
graph = { }
for prenom in liste_prenoms :
    graph [ prenom ] = Personne ( prenom )
```

Complétez le programme suivant (`couples` est la liste créée à la première question) :

```
for a,b in couples :
    graph [ ... ].ajout_ami ( ... )
    graph [ ... ].ajout_ami ( ... )
```

Pourquoi faut-il ajouter la seconde ligne ? Que se passe-t-il si le fichier contient une relation dans un sens puis dans l'autre ?

Troisième demi-heure : les amis d'amis

7) Ecrire une méthode qui compte les amis d'amis :

```
class Personne :
    ...
    def nombre_amis_amis(self) :
        nb = 0
        ....
        return nb
```

8) Créer une seconde fonction pour calculer les amis d'amis d'amis ?

9) Pourquoi le terme d'amis d'amis d'amis n'est-il pas très exact ?

Quatrième demi-heure : composantes connexes

10) On propose de créer une fonction qui calcule la somme de tous les amis, amis d'amis, amis d'amis... La figure 1.2 [24] représente le graphe déduit des relations présentes dans le fichier de la première question.

24. Ce graphe a été réalisé grâce à l'outil *Graphviz* disponible à http://www.graphviz.org/.

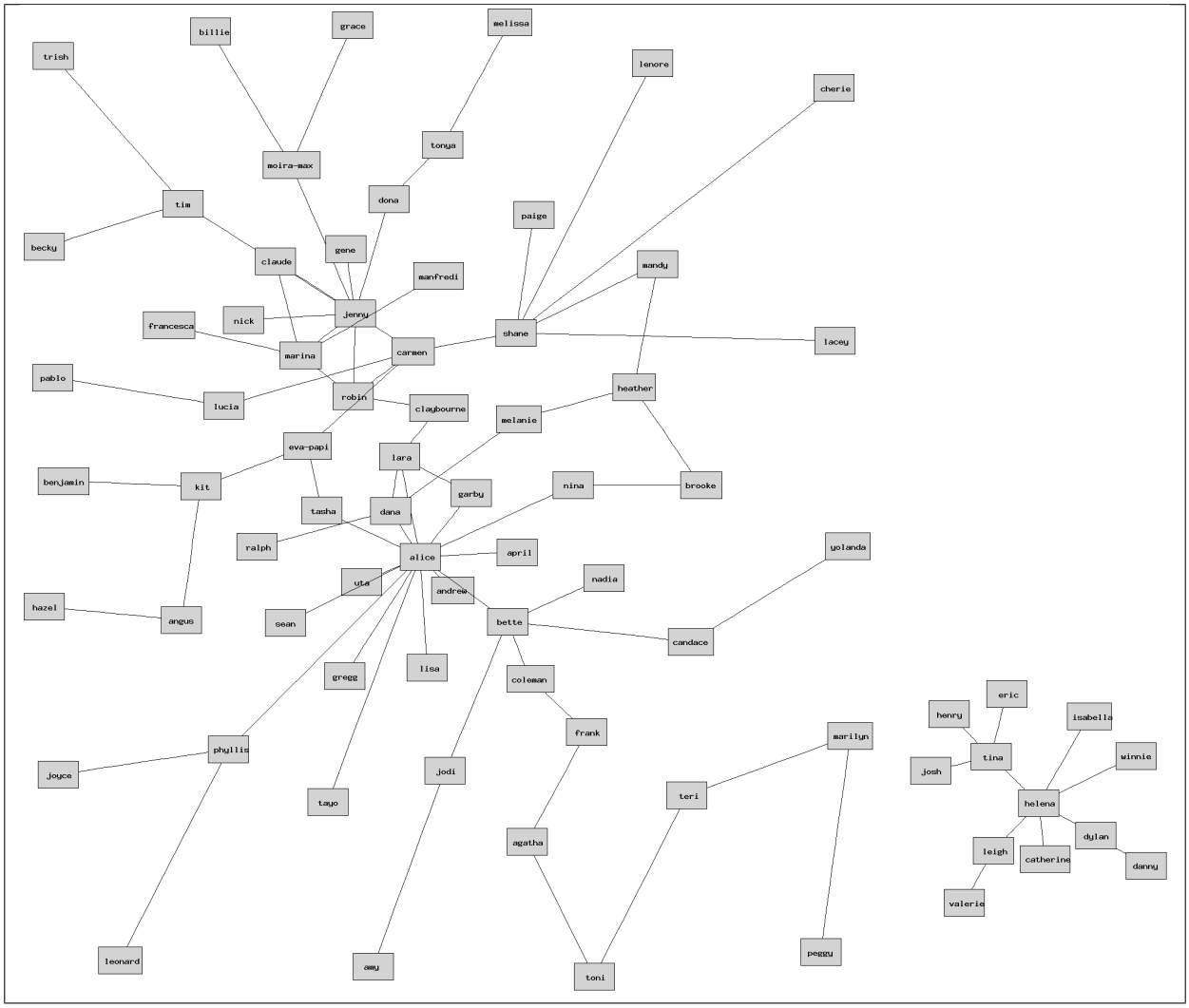


Figure 1.2 : *Image qui représente le graphe décrit par les relations présentes dans le fichier téléchargé à la question 1.*

Expliquer pourquoi, quelque soit la personne choisie, il n'y a que deux réponses possibles.

11) Implémenter la fonction qui retourne le nombre total d'amis ?

12) Vérifiez qu'il n'y a que deux réponses différentes quelque soit la personne.

Pour aller plus loin ou pour ceux qui ont fini plus tôt .

13) Comptez le nombre de triplets d'amis tous connectés entre eux ?

14) Une personne lance une rumeur. En un jour, il propage la rumeur à tous ses amis. En deux jours, tous ses amis d'amis la connaissent et ainsi de suite les jours suivant. Dans le pire des cas, combien de jours faut-il pour qu'une rumeur se propage dans la plus grande composante connexe ?

La réponse à cette dernière question n'est pas évidente sauf si on connaît déjà certains algorithmes comme celui de Bellman-Ford mais il est possible dans un premier temps de commencer à y réfléchir seul.

1.6.2 Correction

1)

fin correction TD 1.6.1 □

1.7 Régression linéaire

1.7.1 Enoncé

Abordé lors de cette séance	
programmation	`numpy` ou calcul matriciel
algorithme	régression linéaire

Le calcul matriciel est aujourd'hui très répandu et présent dans la plupart des logiciels mathématiques gratuits tels que *R*[25], *SciLab*[26], *Octave*[27] ou payants *Gauss*[28], *Matlab*[29], *S+*[30]. Le language *Python* propose un module qui reprend le calcul matriciel proposé par tous ces langages avec des notations similaires. C'est un module qu'il faut télécharger sur Internet à l'adresse suivante : http://numpy.scipy.org/[31]. Un tutoriel en anglais est aussi disponible à l'adresse suivante : http://www.scipy.org/Tentative_NumPy_Tutorial[32]. Ce TD appliquera le calcul matriciel à une régression linéaire[33].

Première demi-heure : données .

1) On applique le principe de la régression aux temps de parcours du marathon au fil des années[34]. Ce fichier contient quatre colonnes : villes, année, temps, temps en seconde. On peut le convertir en une matrice comme suit :

```
#coding:latin-1
import urllib, os, os.path
def charge_donnees () :
    if os.path.exists ("marathon.txt") :
        # si le fichier existe (il a déjà été téléchargé une fois)
        f = open ("marathon.txt", "r")
        text = f.read ()
        f.close ()
    else :
        # si le fichier n'existe pas
        link = "http://www.xavierdupre.fr/enseignement/td_python/" + \
                "python_td_minute/data/regression/marathon.txt"
```

25. http://www.r-project.org/, c'est le plus utilisé par les chercheurs dans des domaines à ceux que l'ENSAE aborde.
26. http://www.scilab.org/
27. http://www.gnu.org/software/octave/
28. http://www.aptech.com/
29. http://www.mathworks.com/
30. http://spotfire.tibco.com/products/s-plus/statistical-analysis-software.aspx
31. Il faut faire attention de bien choisir la version correspondant à votre système d'exploitation (Windows, Linux, Apple) et à la version de votre langage *Python*.
32. voir aussi http://www.scipy.org/NumPy_for_Matlab_Users
33. http://fr.wikipedia.org/wiki/R%C3%A9gression_lin%C3%A9aire, http://fr.wikipedia.org/wiki/R%C3%A9gression_lin%C3%A9aire_multiple ou http://en.wikipedia.org/wiki/Linear_regression
34. http://www.xavierdupre.fr/enseignement/td_python/python_td_minute/data/regression/marathon.txt, ces données proviennent du blog d'Arthur Charpentier http://freakonometrics.blog.free.fr/index.php?post/2011/09/27/agacant-cette-manie-de-battre-des-records.

```
        url = urllib.urlopen (link)
        text = url.read ()
        # on enregistre les données pour éviter de les télécharger une seconde fois
        f = open ("marathon.txt", "w")
        f.write (text)
        f.close ()

    lines = text.split ("\n")
    lines = [ l.split("\t") for l in lines if len(l) > 3 ]

    # conversion en réel des données numérique
    for l in lines :
        l [1] = float(l[1])
        l [3] = float(l[3])
    return lines

if __name__ == "__main__" :
    matrice = charge_donnees ()
    print "nombre de lignes ", len(matrice)
```

2) Le premier réflexe est ensuite de dessiner les données. On utilise le module *matplotlib*[35] Pour dessiner, il est assez simple d'aller à la gallerie pour choisir le graphique[36] qui vous convient et de cliquer dessus pour voir le programme qui permet de le tracer[37].

```
# coding:latin-1
import marathon
# cette première ligne suppose que le programme de la la première question
# a été enregistrée dans un fichier marathon.py
# il n'y a alors pas besoin de le recopier ici
import matplotlib
import matplotlib.pyplot as plt

def dessin (donnees, titre = "titre") :
    x = [ d[0] for d in donnees ]
    y = [ d[1] for d in donnees ]
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(x,y, 'o')
    ax.set_title(titre)
    plt.show()

if __name__ == "__main__" :
    matrice = marathon.charge_donnees()
    mat     = [ (m[1], m[3]) for m in matrice ]
    dessin (mat)
```

Seconde demi-heure : numpy

On crée une matrice numpy comme suit :

```
from marathon import *
import numpy as np

donnees = charge_donnees ()
mat = np.matrix(donnees)
```

35. http://matplotlib.sourceforge.net/, il faut préalablement télécharger et installer ce module.
36. http://matplotlib.sourceforge.net/gallery.html
37. http://matplotlib.sourceforge.net/examples/api/unicode_minus.html

3) Qu'affichent les différentes instructions suivantes :

```
print mat.ndim
print mat.shape
print mat.size
```

On peut manipuler les matrices facilement en utilisant le symbol ":". Que permettent de faire les instructions suivantes ?

```
a = mat[0,0]
b = mat[:,0]
c = mat[0,:]
d = mat[4:10,:]
e = mat.transpose()
```

4) On peut aussi extraire deux colonnes et les coller ensemble :

```
ms = np.column_stack ( (mat[:,1], mat[:,3]))
ms = mat[:,[1,3]]  # seconde écriture
```

Quelle erreur provoque l'instruction suivante et pourquoi ?

```
ms * 3.0
```

L'instruction `ms` $*$ `3.0` ne fait rien de visible, il faudrait pour cela utiliser `print ms` $*$ `3.0` ou encore conserver le résultat dans une variable : `y` $=$ `ms` $*$ `3.0`. Est-ce que la même erreur apparaît avec les deux instructions suivantes :

```
xy = np.matrix ( ms, dtype=float)
xy * 3.0
```

Complétez la ligne suivante pour calculer la moyenne des temps obtenus pour le marathon ?

```
print xy [ ... ].sum() / xy.shape [ ...]
```

5) On souhaite ajouter une colonne de zéros à la matrice `mat`, complétez le programme suivant :

```
z   = np.zeros ( mat.shape[ ... ] )
mat = np.column_stack ( (mat, z))
```

6) Que fait l'instruction suivante :

```
mat [:,4] = 1.0
```

Troisième demi-heure : régression linéaire .

La régression linéaire est une façon de relier une variable Y à une autre X. On écrit le modèle suivant :

$$Y = \alpha X + \beta + \epsilon \tag{1.1}$$

Pour estimer les coefficients α et β, on utilise plusieurs observations (ici 360) puis on minimise l'écart ϵ_i entre la valeur observée Y_i et la valeur estimée $\hat{Y_i} = \alpha X_i + \beta$ pour chaque observation i. Si les écarts sont indépendants et distribués selon une loi normale, cela revient à minimiser l'expression suivante :

$$E = \min_{\alpha,\beta} \left(\sum_i \epsilon_i^2 \right) \tag{1.2}$$

$$= \min_{\alpha,\beta} \left(\sum_i \left(Y_i - (\alpha X_i + \beta) \right)^2 \right) \tag{1.3}$$

Afin de simplifier l'écriture du problème, on suggère de considérer une seconde variable constante $X_2 = 1$ et d'écrire :

$$\alpha X + \beta = \alpha_1 X_1 + \alpha_2 X_2 = X \left(\alpha_1, \alpha_2 \right) = XA \tag{1.4}$$

Où X est la matrice composée des colonnes (X_1, X_2). Dans ce cas, on résoud ce problème en annulant la dérivée et A vaut :

$$A = \left(X'X \right)^{-1} X'Y \tag{1.5}$$

7) Calculer A.

8) Construire une matrice contenant les trois colonnes suivantes : année, temps de course, temps de course estimé.

9) Adapter la fonction utilisée lors de la première demie-heure pour dessiner le temps de course, le temps de cours estimé en fonction des années.

Quatrième demi-heure : ajout d'une variable .

10) On souhaite ajouter la dénivellation maximum de chaque marathon. Voici les altitudes minimales et maximales observées sur chaque parcours [38] :

Paris	30-70
Berlin	25-77
Amsterdam	0-10
Londres	5-40
Stockholm	0-35
Fukuoka	0-8
Boston	0-120
Chicago	172-180

Il faut remplacer le zéro de la dernière colonne par la différence entre le minimum est le maximum. On s'aidera pour cela de l'instruction suivante :

38. source : http://www.marathonguide.com/races/races.cfm

```
mat[ mat[:,0] == "PARIS" , 4 ] = nouvelle valeur
```

11) Calculer les nouveaux coefficients en considérant le modèle suivant :

$$Y = \alpha_1 X_1 + \alpha_2 X_2 + \alpha_3 X_3 + \epsilon \tag{1.6}$$

Où Y est le temps de course, X_1 est l'année, X_2 est la dénivellation, X_3 est une constante (=1).

Pour aller plus loin ou pour ceux qui ont fini plus tôt .

12) Dessiner le temps de course, le premier temps de cours estimé, le second temps de course, en fonction des années.

13) En quelle année le temps de course estimé devient-il nul ? Qu'est-ce que cela vous suggère sur la pertinence du modèle linéaire dans ce cas ?

14) Il est possible de "tester" la nullité d'un coefficient pour vérifier si une variable est importante lors de la régression [39]. En particulier pour la dénivellation, on cherche à savoir si le coefficient est presque nul ou pas. On calcule pour cela la valeur :

$$t_{n-p-1} = \frac{a_j}{\sqrt{\frac{\|y-XA\|^2}{n-p-1}(X'X)^{-1}_{jj}}} \tag{1.7}$$

Où n est le nombre d'observations, p la dimension du problème, M_{jj} le j^{ieme} coefficient sur la diagonale. Ce coefficient suit une loi de Student [40]. Il suffit de comparer la valeur obtenue avec la table de la loi de Student.

1.7.2 Correction

fin correction TD 1.7.1 □

1.8 Décorrélation de variables normales

1.8.1 Enoncé

Abordé lors de cette séance	
programmation	`numpy` ou calcul matriciel
algorithme	décorrélation de variables normales

39. voir *Probabilités, Analyse des Données et Statistiques* de Gilles Saporta, Editions Technip.
40. http ://fr.wikipedia.org/wiki/Loi_de_Student

Le calcul matriciel est aujourd'hui très répandu et présent dans la plupart des logiciels mathématiques gratuits tels que *R*[41], *SciLab*[42], *Octave*[43] ou payants *Gauss*[44], *Matlab*[45], *S+*[46]. Le language *Python* propose un module qui reprend le calcul matriciel proposé par tous ces langages avec des notations similaires. C'est un module qu'il faut télécharger sur Internet à l'adresse suivante : http://numpy.scipy.org/[47]. Un tutoriel en anglais est aussi disponible à l'adresse suivante : http://www.scipy.org/Tentative_NumPy_Tutorial[48].

Ce TD appliquera le calcul matriciel aux vecteurs de variables normales corrélées[49].

Première demi-heure : création d'un jeu de données aléatoires

1) La première étape consiste à construire des variables aléatoires normales corrélées dans une matrice $N \times 3$. En vous inspirant du TD précédent, on cherche à construire cette matrice au format `numpy`. Le programme suivant est un moyen de construire un tel ensemble à l'aide de combinaisons linéaires. Complétez les lignes contenant des .....

```
import random
import numpy as np

def combinaison () :
    x = random.gauss(0,1) # génère un nombre aléatoire
    y = random.gauss(0,1) # selon une loi normale
    z = random.gauss(0,1) # de moyenne null et de variance 1
    x2 = x
    y2 = 3*x + y
    z2 = -2*x + y + 0.2*z
    return [x2, y2, z2]

mat = [ ............. ]
npm = np.matrix ( mat )
```

2) A partir de la matrice `npm`, on veut construire la matrice des corrélations.

```
npm = ...               # voir question précédente
t   = npm.transpose ()
a   = t * npm
a  /= npm.shape[0]
```

A quoi correspond la matrice `a` ?

Seconde demi-heure : matrice de corrélation

3) Construire la matrice des corrélations à partir de la matrice `a`. Si besoin, on pourra utiliser le module `copy`.

41. http://www.r-project.org/, c'est le plus utilisé par les chercheurs dans des domaines à ceux que l'ENSAE aborde.
42. http://www.scilab.org/
43. http://www.gnu.org/software/octave/
44. http://www.aptech.com/
45. http://www.mathworks.com/
46. http://spotfire.tibco.com/products/s-plus/statistical-analysis-software.aspx
47. Il faut faire attention de bien choisir la version correspondant à votre système d'exploitation (Windows, Linux, Apple) et à la version de votre langage *Python*.
48. voir aussi http://www.scipy.org/NumPy_for_Matlab_Users
49. voir http://fr.wikipedia.org/wiki/Covariance, ou aussi http://fr.wikipedia.org/wiki/D%C3%A9composition_en_valeurs_singuli%C3%A8res.

```
import copy
b = copy.copy (a)    # remplacer cette ligne par b = a
b [0,0] = 44444444
print b              # et comparer le résultat ici
```

4) Construire une fonction qui prend comme argument la matrice `npm` et qui retourne la matrice de corrélation. Cette fonction servira plus pour vérifier que nous avons bien réussi à décorréler.

```
def correlation ( npm ) :
    ..........
    return .....
```

Pour la suite, un peu de mathématique. On note M la matrice `npm`. $V = \frac{1}{n}M'M$ correspond à la matrice des **covariances** et elle est nécessairement symétrique. C'est une matrice diagonale si et seulement si les variables normales sont indépendantes. Comme toute matrice symétrique, elle est diagonalisable. On peut écrire :

$$\frac{1}{n}M'M = P\Lambda P' \tag{1.8}$$

P vérifie $P'P = PP' = I$. La matrice Λ est diagonale et on peut montrer que toutes les valeurs propres sont positives ($\Lambda = \frac{1}{n}P'M'MP = \frac{1}{n}(MP)'(MP)$).
On définit alors la racine carrée de la matrice Λ par :

$$\Lambda = diag(\lambda_1, \lambda_2, \lambda_3) \tag{1.9}$$
$$\Lambda^{\frac{1}{2}} = diag\left(\sqrt{\lambda_1}, \sqrt{\lambda_2}, \sqrt{\lambda_3}\right) \tag{1.10}$$

On définit ensuite la racine carrée de la matrice V :

$$V^{\frac{1}{2}} = P\Lambda^{\frac{1}{2}}P' \tag{1.11}$$

On vérifie que $\left(V^{\frac{1}{2}}\right)^2 = P\Lambda^{\frac{1}{2}}P'P\Lambda^{\frac{1}{2}}P' = P\Lambda^{\frac{1}{2}}\Lambda^{\frac{1}{2}}P' = V = P\Lambda P' = V$.

Troisième demi-heure : calcul de la racine carrée

5) Le module `numpy` propose une fonction qui retourne la matrice P et le vecteur des valeurs propres L :

```
L,P = np.linalg.eig(a)
```

Vérifier que $P'P = I$. Est-ce rigoureusement égal à la matrice identité ?
6) Que fait l'instruction suivante :

```
print np.diag(L)
```

7) Ecrire une fonction qui calcule la racine carrée de la matrice $\frac{1}{n}M'M$ (on rappelle que M est la matrice `npm`) [50].

Quatrième demi-heure : décorrélation .

La fonction suivante permet d'obtenir l'inverse de la matrice `a`.

```
np.linalg.inv(a)
```

8) Chaque ligne de la matrice M représente un vecteur de trois variables corrélées. La matrice de covariance est $V = \frac{1}{n}M'M$. Calculer la matrice de covariance de la matrice $N = MV^{-\frac{1}{2}}$ (mathématiquement).
9) Vérifier numériquement.

Pour aller plus loin ou pour ceux qui ont fini plus tôt .

10) A partir du résultat précédent, proposer une méthode pour simuler un vecteur de variables corrélées selon une matrice de covariance V à partir d'un vecteur de lois normales indépendantes.
11) Proposer une fonction qui crée cet échantillon :

```
def simultation (N, cov) :
    # simule un échantillon de variables corrélées
    # N : nombre de variables
    # cov : matrice de covariance
    ...
    return M
```

12) Vérifier que votre échantillon a une matrice de corrélations proche de celle choisie pour simuler l'échantillon.

1.8.2 Correction

fin correction TD 1.8.1 □

1.9 Expression réguligères et discours des présidents

1.9.1 Enoncé

Abordé lors de cette séance	
programmation	expression régulières
algorithme	comptage

Dès qu'on cherche à analyser du texte de façon intelligente, on utilise nécessairement les expressions régulières qui offrent un moyen simple et rapide de chercher des motifs dans un texte. Tous les langages

50. http://fr.wikipedia.org/wiki/Racine_carr%C3%A9e_d'une_matrice

proposent maintenant les mêmes outils, le langage *Python* propose quant à lui le module `re`[51]. Il s'agit d'explorer cette fonctionnalité lors de cette séance.

Première demi-heure : données

1) On récupère les voeux des présidents au soir du 31 décembre de quelques années et quelques présidents différents grâce au programme suivant.[52]

```
#coding:latin-1
import urllib, os, os.path
def charge_discours () :
    discours = { }
    for annee in [2001, 2005, 2006, 2007, 2008, 2009, 1974, 1975,
                  1979, 1983, 1987, 1989, 1990, 1994] :
        nom = "VOEUX%02d.txt" % (annee % 100)
        if os.path.exists (nom) :
            # si le fichier existe (il a déjà été téléchargé une fois)
            f = open (nom, "r")
            text = f.read ()
            f.close ()
        else :
            # si le fichier n'existe pas
            link = "http://www.xavierdupre.fr/enseignement/td_python/" + \
                    "python_td_minute/data/voeux_presidents/" + nom
            url = urllib.urlopen (link)
            text = url.read ()
            # on enregistre les données pour éviter de les télécharger une seconde fois
            f = open (nom, "w")
            f.write (text)
            f.close ()

        discours [annee] = text
    return discours

if __name__ == "__main__" :
    discours = charge_discours ()
    print "nombre de discours ", len(discours)
```

On pourra enregistrer ce programme sous le nom `discours.py` et s'y référer en utilisant l'instruction :

```
import discours
textes = discours.charge_discours()
```

2) On va dans un premier temps construire des statistiques sur l'ensemble des textes plutôt que sur chaque texte pris séparément. Ecrire une fonction qui fait la concaténation de tous les textes.

```
import discours
textes = discours.charge_discours()

def somme_texte (textes):
    ...
    return ...
```

51. http://docs.python.org/library/re.html
52. ttp://www.xavierdupre.fr/enseignement/td_python/python_td_minute/data/voeux_presidents/VOEUX*.txt, ces données proviennent du blog d'Arthur Charpentier http://freakonometrics.blog.free.fr/index.php?post/2011/05/12/De-quoi-parle-un-president-francais-le-31-decembre.

3) Les accents sont toujours un problème car une lettre majuscule n'a pas d'accent : l'ordinateur considère que "A", "à", "a" sont des mots différents. On écrit alors une fonction qui remplace les accents par des lettres non accentuées. La fonction fait aussi autre chose, qu'est-ce ?

```
accent = {     'à':'a', 'â':'a',  'ä':'a',
                    'é':'e', 'è':'e', 'ê':'e', 'ë':'e',
                    'î':'i', 'ï':'i',
                    'ù':'u', 'û':'u', 'ü':'u',
                    'ô':'o', 'ö':'o',
                }

def pas_daccent (texte) :
    res = ""
    for c in texte :
        c = c.lower ()
        res += accent.get (c,c)
    return res
```

Seconde demi-heure : compter les mots

4) On écrit une fonction à compléter qui découpe en mots avec la fonction `split` :

```
def decoupe_mot (texte) :
    return ....
```

5) On écrit une fonction à compléter qui compte la fréquence de chaque mot distinct :

```
def compte_mots (mots) :
    d = { }
    for mot in mots :
        ...
        ...
    return d
```

6) On écrit une fonction à compléter qui trie les mots par ordre de fréquence décroissant :

```
def mot_tries (d):
    l = [ (n,m) for ... in d.iteritems () ]
    l.sort (reverse = True)
    return l
```

7) On veut afficher les 25 mots les plus fréquents, comment utiliser les trois fonctions précédentes ? Que pensez-vous de ces mots ?

Troisième demi-heure : expression régulières

8) En vous aidant de la page http://docs.python.org/library/re.html, expliquez ce que fait la fonction suivante ?

```
def decoupe_mot2 (texte) :
    exp = re.compile ("\\w+", re.IGNORECASE)
    return exp.findall(texte)
```

9) Comparer les résultats avec la dernière question de la partie précédente. Pourquoi le mot `l` est-il présent ?

10) Comment modifier l'expression "\\w+" pour ne compter que les mots de plus de six lettres ?

Quatrième demi-heure : expressions plus complexes .

11) Toujours en vous aidant de la page http://docs.python.org/library/re.html, expliquez ce que fait la fonction suivante ?

```
def trouver_expression(texte) :
    exp = re.compile ("je .{1,60}", re.IGNORECASE)
    return exp.findall(texte)
```

12) Utiliser cette fonction sur le texte entier puis sur chaque texte pris séparément.

Pour aller plus loin ou pour ceux qui ont fini plus tôt .

13) Avec la même expression régulière, rechercher indifféremment le mot *securite* ou *insecurite*.

1.9.2 Correction

fin correction TD 1.9.1 □

1.10 Plus court chemin dans un graphe

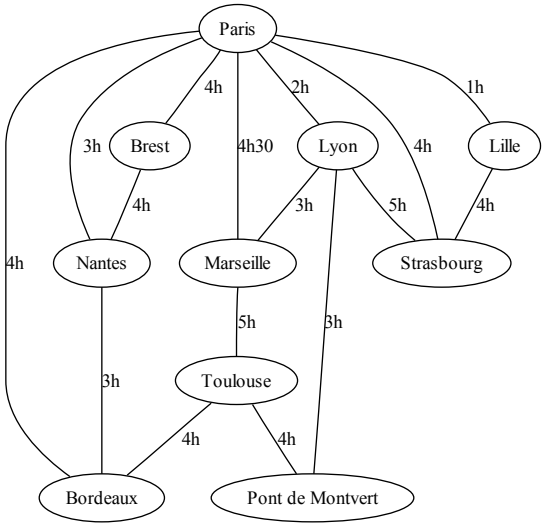
1.10.1 Enoncé

L'énoncé de cette séance est long, les deux dernières parties peuvent être traitées indépendamment l'une de l'autre.

Abordé lors de cette séance	
programmation	meilleur chemin dans un graphe
algorithme	graphe

L'algorithme du plus court chemin est celui qu'on évoque le plus souvent lorsqu'on parle de programmation dynamique [53]. Plusieurs variantes existent, chacune portant le nom de son inventeur Bellman, Ford [54], Djikstra [55]... Le graphe suivant représente quelques lignes ferroviaires et le temps approximatif que le train met pour relier deux villes.

53. http://en.wikipedia.org/wiki/Dynamic_programming
54. http://fr.wikipedia.org/wiki/Algorithme_de_Bellman-Ford
55. http://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra



Lorsque le graphe est aussi simple que celui-là, il est facile de chercher le chemin le plus court pour aller de Nantes à Pont de Montvert. Lorsqu'il ressemble à celui de la figure 1.3, il devient plus difficile de déterminer le chemin le plus court sans utiliser un algorithme.

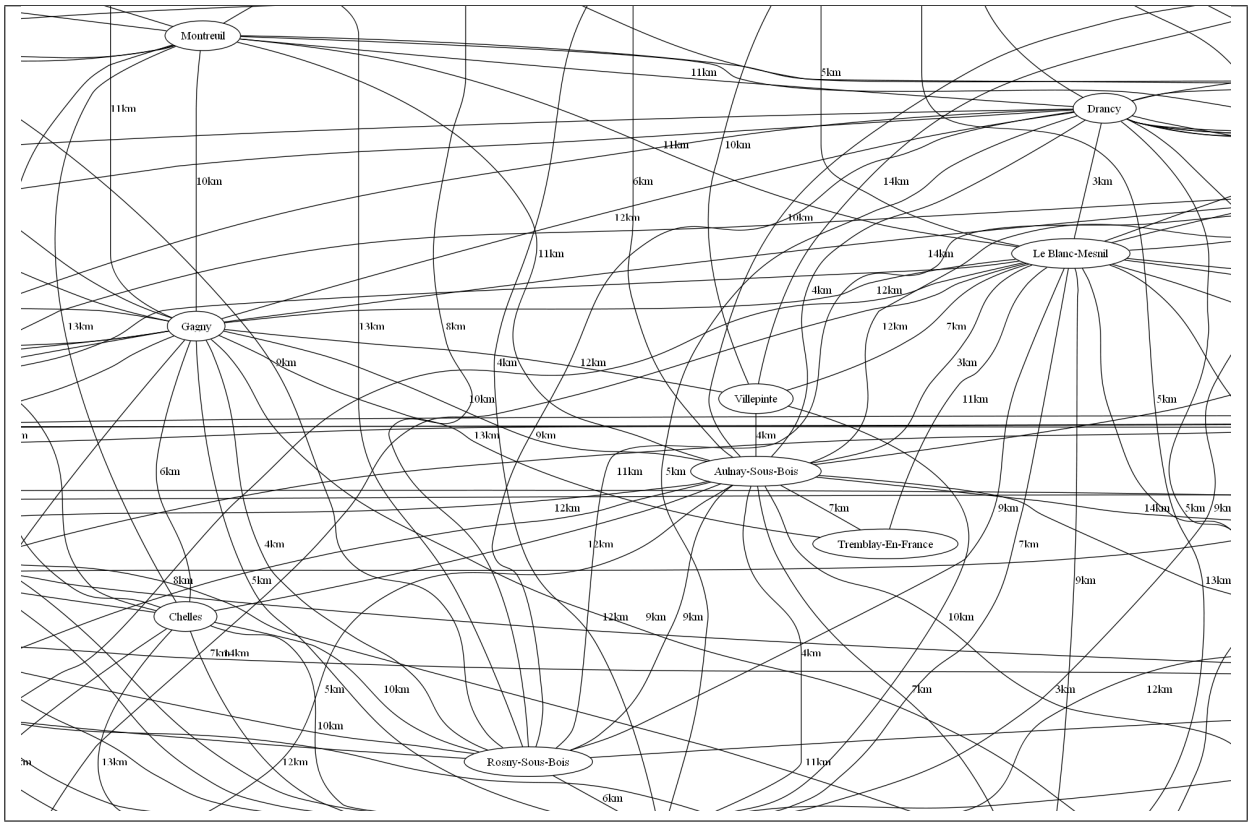


Figure 1.3 : *Extrait d'un graphe dont chaque arc correspond à une distance entre deux villes. Ces distances ne sont pas les distances à vol d'oiseau mais celles des itinéraires calculés par GoogleMaps.*

Première demi-heure : données

1) On utilise le programme suivant [56] pour récupérer les données qui représente le graphe dont une partie est représentée par la figure 1.3.

```
#coding:latin-1
import urllib, os, os.path
# coding:latin-1
def drawGraph (edges, script, image) :
    """
    dessine un graph en utilisant Graphviz (http://www.graphviz.org/
    edges = [ (1,2), (3,4), (1,3), ... ]  , liste d'arcs
    image = bom d'image (format png)
    """
    import os, urllib,struct
    files = [ "_graphviz_draw.exe"]
    if not os.path.exists (files[-1]) :
        # on télécharge les fichiers nécessaires d'abord
        for f in files :
            print "téléchargement de ", f
            url = "http://www.xavierdupre.fr/enseignement/tutoriel_python/graphviz/" + f
            u = urllib.urlopen (url, "rb")
            all = u.read ()
            if "404 Not Found" in all :
                raise Exception ("fichier introuvable")
            u.close ()
            u = open (f, "wb")
            u.write ( struct.pack ("c"*len(all), *all))
            u.close()
    if not os.path.exists (files[-1]) :
        raise Exception ("mauvais téléchargement")

    if script == None :
        li = [ "digraph{" ]
        for i,j in edges :
            li.append ( "%d -> %d ;" % (i,j) )
        li.append (";")
        f = open ("graph.gv", "w")
        f.write ( "\n".join(li) )
        f.close ()
    else :
        f = open ("graph.gv", "w")
        f.write ( script )
        f.close ()

    cmd = "%s . graph.gv %s png neato" % (files[-1], image)
    os.system (cmd)

def charge_donnees (file = "matrix_distance_7398.txt") :
    if os.path.exists (file) :
        # si le fichier existe (il a déjà été téléchargé une fois)
        f = open (file, "r")
        text = f.read ()
        f.close ()
    else :
        # si le fichier nŠexiste pas
        link = "http://www.xavierdupre.fr/enseignement/td_python/" + \
        "python_td_minute/data/court_chemin/" + file
        url = urllib.urlopen (link)
        text = url.read ()
        # on enregistre les données pour éviter de les télécharger une seconde fois
        f = open (file, "w")
        f.write (text)
```

56. http://www.xavierdupre.fr/enseignement/td_python/python_td_minute/data/court_chemin/load_distance_matrix.py

```
        f.close ()
    lines = text.split ("\n")
    lines = [ l.split("\t") for l in lines if len(l) > 1 ]
    return lines

def conversion_en_dictionnaire (lines) :
    res = { }
    for a,b,c in lines :
        c = int (c)
        res [a,b] = c
        res [b,a] = c
    return res

def graphviz_script (mat_dict) :
    script = ["graph {"]
    vertex = { }
    villes = [ _[0] for _ in mat_dict.keys() ]
    for v in villes :
        if v not in vertex :
            vertex [v] = len(vertex)
    for k,v in vertex.iteritems () :
        script.append ( "%d [label=\"%s\"];" % (v,k) )
    for k,v in mat_dict.iteritems () :
        i1 = vertex[k[0]]
        i2 = vertex[k[1]]
        if i1 < i2 and v < 50000 :
            #on coupe des arcs car le tracé est trop long sinon
            script.append ( "%d -- %d [label=\"%skm\"];" % (i1,i2,v/1000))
    script.append ("}")
    return "\n".join( script)

if __name__ == "__main__" :
    matrice_line = charge_donnees ()
    mat = conversion_en_dictionnaire (matrice_line)
    script = graphviz_script (mat)
    drawGraph([], script , "im.png")
```

2) En utilisant les résultats retournés par la fonction `conversion_en_dictionnaire` (ou `charge_donnees`), on souhaite évaluer la complexité du problème en calculant deux choses :

1. le nombre de villes du graphe
2. le nombre maximum d'arcs reliés à une ville

Ecrire deux fonctions qui calculent ces deux nombres.

3) Adaptez la première fonction pour qu'elle retourne une liste contenant la liste des villes, chaque ville n'apparaissant qu'une fois.

Seconde demi-heure : approche innocente .

Dans cette partie, on utilise le dictionnaire (d) retourné par la fonction `conversion_en_dictionnaire`.

4) Extraire les trois distances reliant les trois villes *Paris*, *Beauvais*, *Dieppe* entre elles. Vérifier sur *Google-Maps*[57] les trois itinéraires reliant ces trois villes deux à deux ? Par défaut, quel critère le site optimise-t-il ? Comparer les deux distances suivantes :

– distance(Paris,Dieppe)
– distance(Paris,Beauvais) + distance(Beauvais, Dieppe)

5) On cherche tous les triplets de villes (x, y, z) qui vérifient :

$$d(x, y) + d(y, z) < d(x, z) \tag{1.12}$$

57. http://maps.google.fr/

Combien sont-ils ? (le graphe n'inclut pas forcément un arc (x, z) mais il inclut les deux autres)

Troisième demi-heure : algorithme approchée de Bellman [58]

6) Ecrire une fonction `misajour` qui met à jour le dictionnaire d pour chaque triplet trouvé vérifiant la condition (1.12). La nouvelle valeur contient la distance la plus courte.

7) Modifier la fonction précédente pour qu'elle retourne le nombre de valeurs modifiées.

8) Appeler la fonction `misajour` une dizaine de fois en affichant le nombre de valeurs modifiées.

9) Après plusieurs itérations, on remarque que :

1. Le nombre de valeurs modifiées est croissant ou décroissant ?
2. La somme de chaque distance est croissante ou décroissante ?
3. On suppose que cette somme est constante après un grand nombre d'itérations, que vaut alors `d["Paris","Dieppe"]` ?
4. Est-il alors possible de connaître le chemin le plus court qui relie Paris à Dieppe par la seule connaissance de d ?

La figure 1.4 vous aidera à trouver quelques réponses.

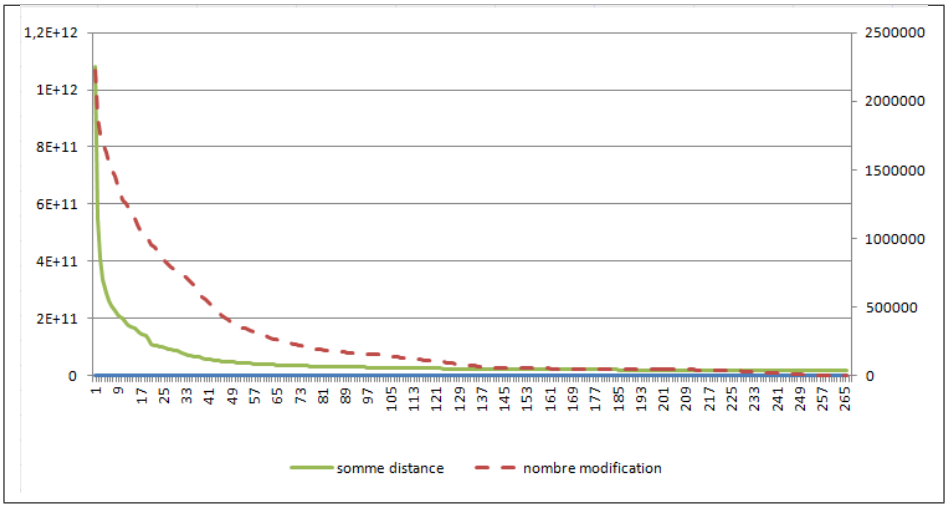


Figure 1.4 : *Evolution du nombre de modifications et de la somme des distances de chaque case de la matrice en fonction du nombre d'itérations.*

10) L'algorithme suggéré dans cette partie est différent de celui inventé par Bellman-Ford, pourquoi ?

Quatrième demi-heure : algorithme de Djikstra [59]

L'algorithme de Djikstra se concentre sur le chemin le plus court entre deux villes précises et permet de retourner le chemin le plus court.

11) Utiliser le pseudo-code de la page Wikipedia pour implémenter cet algorithme.

```
Fonction Dijkstra (nJuds, fils, distance, début, fin)
     Pour n parcourant nJuds
         n.parcouru = infini   // Peut être implémenté avec -1
```

58. http://fr.wikipedia.org/wiki/Algorithme_de_Bellman-Ford
59. http://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra

```
        n.précédent = 0
    Fin pour
    début.parcouru = 0
    pasEncoreVu = nJuds
    Tant que pasEncoreVu != liste vide
        n1 = minimum(pasEncoreVu)   // Le nJud dans pasEncoreVu avec parcouru le plus petit
        pasEncoreVu.enlever(n1)
        Pour n2 parcourant fils(n1)                          // Les nJuds reliés à n1 par un arc
            Si n2.parcouru > n1.parcouru + distance(n1, n2) // distance correspond au
                                                             // poids de l'arc reliant n1 et n2
                n2.parcouru = n1.parcouru + distance(n1, n2)
                n2.précédent = n1   // Dit que pour aller à n2, il faut passer par n1
            Fin si
        Fin pour
    Fin tant que
    chemin = liste vide
    n = fin
    Tant que n != début
        chemin.ajouterAvant(n)
        n = n.précédent
    Fin tant que
    chemin.ajouterAvant(debut)
    Retourner chemin
Fin fonction Dijkstra
```

12) Quel est le résultat entre Paris et Dieppe ?

1.10.2 Correction

fin correction TD 1.10.1 □

Chapitre 2

Compilation des précédentes séances notées

2.1 Année 2006

L'objectif de cet exercice est de programmer une recherche dans une liste triée.

1) Il faut d'abord récupérer un fichier texte disponible sur Intranet [1]. Ce fichier contient un mot par ligne. Il faut lire ce fichier et construire une liste avec tous ces mots. (4 points)

2) Construire une fonction qui vérifie que la liste chargée à la question précédente est triée. (4 points)

3) Construire une fonction qui recherche un mot X dans la liste et qui retourne sa position ou -1 si ce mot n'y est pas. Cette fonction prend deux paramètres : la liste et le mot à chercher. Elle retourne un entier. On précise que pour savoir si deux chaînes de caractères sont égales, il faut utiliser l'opérateur ==. (4 points)

4) Quels sont les positions des mots "UN" et "DEUX" ? La réponse doit figurer en commentaire dans le programme. Il faudra écrire aussi le nombre de comparaisons effectuées pour trouver ces deux positions. (2 points)

5) Lorsqu'une liste est triée, rechercher un élément est beaucoup plus rapide. Si on cherche le mot X dans la liste, il suffit de le comparer au mot du milieu pour savoir si ce mot est situé dans la partie basse (X inférieur au mot du milieu), la partie haute (X supérieur au mot du milieu). S'il est égal, le mot a été trouvé. Si le mot n'a pas été trouvé, on recommence avec la sous-liste inférieure ou supérieure selon les cas jusqu'à ce qu'on ait trouvé le mot ou qu'on soit sûr que le mot cherché n'y est pas.

Le résultat de la recherche est la position du mot dans la liste ou -1 si ce mot n'a pas été trouvé. Cette recherche s'appelle une recherche dichotomique.

Ecrire la fonction qui effectue la recherche dichotomique d'un mot dans une liste triée de mots. Vérifiez que les deux fonctions retournent bien les mêmes résultats. Cette fonction peut être récursive ou non. Elle prend au moins les deux mêmes paramètres que ceux de la question 3, si elle en a d'autres, il faudra leur donner une valeur par défaut. On précise que les comparaisons entre chaînes de caractères utilisent aussi les opérateurs <, ==, >. (4 points)

6) Normalement, les positions des mots "UN" et "DEUX" n'ont pas changé mais il faut de nouveau déterminer le nombre d'itérations effectuées pour trouver ces deux positions avec la recherche dichotomique. (2 points)

7) Quel est, au pire [2], le coût d'une recherche non dichotomique ? La réponse doit figurer en commentaire dans le programme. (1 point)

8) Quel est, au pire, le coût d'une recherche dichotomique ? La réponse doit figurer en commentaire dans le programme. (1 point)

Correction

```
# coding: latin-1
# question 1
def lit_fichier (file) :
```

1. http://www.xavierdupre.fr/enseignement/initiation/td_note_texte.txt
2. On cherche la meilleure majoration du coût de la recherche non dichotomique en fonction de la taille n de la liste.

```
    f = open (file, "r")
    mot = []
    for l in f :
        mot.append ( l.replace ("\n", "") )
    f.close ()
    return mot

mot = lit_fichier ("td_note_texte.txt")
print mot

# question 2
def est_trie (mot) :
    for i in range (1, len (mot)) :
        if mot [i-1] > mot [i] :
            return False
    return True

tri = est_trie (mot)
print "liste triée ", tri

# question 3
def cherche (mot, m) :
    for i in range (0, len (mot)) :
        if mot [i] == m :
            return i
    return -1

print "mot ACHATS ", cherche (mot, "ACHATS")
print "mot achats ", cherche (mot, "achats")

# question 4
un   = cherche (mot, "UN")
deux = cherche (mot, "DEUX")
print "recherche normale ", un, deux
print "nombre d'itérations", un + deux

# question 5, 6, nbun et nbdeux contiennent le nombre de comparaisons
def cherche_dicho (mot, m) :
    a = 0
    b = len (mot)-1
    nb = 0
    while a < b :
        nb += 1
        p = (a+b)/2
        if   mot [p] == m :  return p,nb
        elif mot [p] >  m :  b = p-1
        else :               a = p+1
    return -1,nb

un,nbun     = cherche_dicho (mot, "UN")
deux,nbdeux = cherche_dicho (mot, "DEUX")
print "recherche dichotomique ", un, deux
print "nombre d'itérations ", nbun + nbdeux

# question 7
"""
Lors d'une recherche simple, au pire, l'élément cherche sera
en dernière position, ce qui signifie n itérations pour le trouver.
Le coût de la recherche simple est en O(n).
"""

# question 8
"""
Lors de la recherche dichotomique, à chaque itération, on divise par deux
l'ensemble dans lequel la recherche s'effectue,
```

```
au départ n, puis n/2, puis n/4 jusqu'àă ce que n/2^k soit nul
c'est-à-dire k = partie entière de ln n / ln 2
il y a au plus k itérations donc le coût de l'algorithme est en O (ln n).
"""
```

fin exo 1 □

2.2 Année 2007

2.2.1 Avec des fonctions uniquement

Enoncé

Le gouvernement désire ajouter un jour férié mais il voudrait le faire à une date éloignée des jours fériés existant. On suppose également que ce jour ne sera pas inséré entre Noël et le jour de l'an. On va donc calculer le nombre de jours qui sépare deux jours fériés dont voici la liste pour l'année 2007 :

Jour de l'an	1er janvier 2007
Lundi de Pâques	9 avril 2007
Fête du travail	1er mai 2007
Victoire de 1945	8 mai 2007
Ascension	17 mai 2007
Lundi de Pentecôte	4 juin 2007
Fête nationale	14 juillet 2007
Assomption	15 août 2007
Toussaint	1er novembre 2007
Armistice de 1918	11 novembre 2007
Noël	25 décembre 2007

On rappelle que l'année 2007 n'est pas une année bissextile et qu'en conséquence, le mois de février ne comporte que 28 jours.

1) Afin de simplifier la tâche, on cherche à attribuer un numéro de jour à chaque jour férié : l'année a 365 jours, pour le numéro du lundi de Pâques, soit 31 (mois de janvier) + 28 (février) + 31 (mars) + 9 = 89. La première question consiste à construire une fonction qui calcule le numéro d'une date étant donné un jour et un mois. Cette fonction prend comme entrée : (4 points)

- un numéro de jour
- un numéro de mois
- une liste de 12 nombres correspondant au nombre de jours dans chacun des douze mois de l'année

2) Si on définit la liste des jours fériés comme étant une liste de couples (jour, mois) triée par ordre chronologique, il est facile de convertir cette liste en une liste de nombres correspondant à leur numéro dans l'année. La fonction à écrire ici appelle la précédente et prend une liste de couples en entrée et retourne comme résultat une liste d'entiers. (4 points)

3) A partir de cette liste d'entiers, il est facile de calculer l'écart ou le nombre de jours qui séparent deux jours fériés. Il ne reste plus qu'à écrire une fonction qui retourne l'écart maximal entre deux jours fériés, ceux-ci étant définis par la liste de numéros définie par la question précédente. Un affichage du résultat permet de déterminer les deux jours fériés les plus éloignés l'un de l'autre. Quels sont-ils ? (4 points)

2.2.2 Programme équivalent avec des classes

Enoncé

Le programme précédent n'utilise pas de classe. L'objectif de ce second exercice est de le réécrire avec une classe.

1) Une fonction du programme précédent effectue la conversion entre un couple jour-mois et un numéro de jour. Les calculs sont faits avec le numéro mais le résultat désiré est une date : les numéros ne sont que des intermédiaires de calculs qui ne devraient pas apparaître aussi explicitement. La première question consiste à créer une classe `Date` : (2 points)

```
class Date :
    def __init__ (self, jour, mois) :
        ...
```

2) A cette classe, on ajoute une méthode qui retourne la conversion du couple jour-mois en un numéro de jour de l'année. (2 points)

3) On ajoute maintenant une méthode calculant le nombre de jours séparant deux dates (ou objet de type `Date` et non pas numéros). Cette méthode pourra par exemple s'appeler `difference`. (2 points)

4) Il ne reste plus qu'à compléter le programme pour obtenir les mêmes résultats que le programme de l'exercice 1. (2 points)

5) Avec ce programme, lors du calcul des écarts entre tous les jours fériés consécutifs, combien de fois effectuez-vous la conversion du couple jour-mois en numéro pour le second jour férié de l'année ? Est-ce le même nombre que pour le programme précédent (en toute logique, la réponse pour le premier programme est 1) ? (2 points)

6) La réponse à la question précédente vous suggère-t-elle une modification de ce second programme ? (2 points)

Correction

```
# coding: latin-1
###################################
# exercice 1
###################################

# question 1
def numero (jour, mois, duree = [31, 28, 31,30,31,30,31,31,30,31,30,31] ) :
    s = 0
    for i in range (0,mois-1) :
        s += duree [i]
    s += jour - 1
    return s+1

# question 2
def conversion_liste (li) :
    res = []
    for jour,mois in s : res.append ( numero (jour, mois))
    # pareil que
    # for i in range (0, len (s)) : res.append ( numero (s [i][0], s [i][1]))
    return res

def ecart (num) :
    res = []
    for i in range (1, len (num)) :
        d = num [i] - num [i-1]
        res.append (d)
    return res
```

```
s = [ (1,1), (9,4), (1,5), (8,5), (17,5), (4,6), (14,7), \
        (15,8), (1,11), (11,11), (25,12) ]
r = conversion_liste (s)
ec = ecart (r)

# question 3
pos = ec.index ( max (ec) )
print "position de l'écart le plus grand ", pos
print "jour ", s [pos], " --> ", s [pos+1]

####################################
# exercice 2
####################################

# question 4
class Date :
    def __init__ (self, jour, mois) :
        self.jour = jour
        self.mois = mois
        self.duree = [31, 28, 31,30,31,30,31,31,30,31,30,31]

    # question 5
    def numero (self) :
        s = 0
        for i in range (0,self.mois-1) :
            s += self.duree [i]
        s += self.jour - 1
        return s+1

    # question 6
    def difference (self, autre) :
        return self.numero () - autre.numero ()

def conversion_date (s) :
    res = []
    for jour,mois in s :
        res.append ( Date (jour, mois) )
    return res

def ecart_date (date) :
    ec = []
    for i in range (1, len (date)) :
        ec.append ( date [i].difference ( date [i-1] ) )
    return ec


# question 7
s = [ (1,1), (9,4), (1,5), (8,5), (17,5), (4,6), \
            (14,7), (15,8), (1,11), (11,11), (25,12) ]

r = conversion_date (s)
ec = ecart_date (r)
pos = ec.index ( max (ec) )
print "position de l'ecart le plus grand ", pos
print "jour ", s [pos], " --> ", s [pos+1]

# question 8
"""
La conversion en Date est faite une fois pour les dates (1,1) et (25,12)
et 2 fois pour les autres en effet, la méthode difference effectue
la conversion en numéros des dates self et autre
la fonction ecart_date calcule date [i].difference ( date [i-1] ) et
                                    date [i+1].difference ( date [i] )
            --> la date [i] est convertie 2 fois
```

```
"""

# question 9
"""
On peut par exemple stocker la conversion en numéro
dans le constructeur comme suit :
"""

class Date :
    def __init__ (self, jour, mois) :
        self.jour = jour
        self.mois = mois
        self.duree = [31, 28, 31,30,31,30,31,31,30,31,30,31]
        self.num = self.numero ()

    # question 5
    def numero (self) :
        s = 0
        for i in range (0,self.mois-1) :
            s += self.duree [i]
        s += self.jour - 1
        return s+1

    # question 6
    def difference (self, autre) :
        return self.num - autre.num


r = conversion_date (s)
ec = ecart_date (r)
pos = ec.index ( max (ec) )
print "position de l'écart le plus grand ", pos
print "jour ", s [pos], " --> ", s [pos+1]
```

On pourrait encore améliorer la dernière classe `Date` en créant un attribut statique pour l'attribut `duree` qui est identique pour toutes les instances de la classe `Date`.

fin exo 2.2.1 □

2.3 Année 2008

Lorsqu'on se connecte à un site internet, celui-ci enregistre votre adresse IP, l'heure et la date de connexion ainsi que la page ou le fichier désiré. Ainsi le fichier *logpdf.txt*[3] recense ces quatre informations séparées par des espaces pour les fichiers d'extension *pdf* téléchargés sur le site http://www.xavierdupre.fr/.

1) La première étape consiste à charger les informations depuis le fichier texte dans une matrice de 4 colonnes (liste de listes, liste de t-uples). On utilisera pour cela les fonctions `open` et les méthodes `split`, `replace` associées aux chaînes de caractères. On pourra s'aider des corrections des TD précédents. (2 points)

2) On souhaite dans un premier temps faire des statistiques sur les dates : on veut compter le nombre de fichiers téléchargés pour chaque date présente dans les données. On pourra pour cela utiliser un dictionnaire dont la clé sera bien choisie. Le résultat devra être inséré dans une fonction prenant comme entrée une matrice (ou liste de listes, liste de t-uples) et retournant un dictionnaire comme résultat. (4 points)

3) On s'intéresse au programme suivant :

```
l = [ (1, "un"), (3, "deux",), (2, "deux"), (1, "hun"), (-1, "moinsun") ]
l.sort (reverse = True)
print l
```

3. Il faut télécharger ce fichier depuis l'adresse http://www.xavierdupre.fr/enseignement/initiation/logpdf.txt

Donc le résultat est :

```
[(3, 'deux'), (2, 'deux'), (1, 'un'), (1, 'hun'), (-1, 'moinsun')]
```

Que s'est-il passé ? (2 points)

4) On désire maintenant connaître les 10 dates pour lesquelles il y a eu le plus de téléchargements ces jours-là. L'inconvénient est que le dictionnaire élaboré à la question 2 ne retourne pas les réponses dans l'ordre souhaité : il faut classer les dates par nombre de téléchargements croissants. Il faut ici imaginer une fonction qui retourne ces dix meilleures dates et des dix fréquentations correspondantes en se servant de la remarque de la question 3. (4 points) A quels événements correspondent les quatre premières dates ? (Cette petite question ne rapporte pas de point.)

5) Effectuez le même travail pour déterminer les dix documents les plus téléchargés. (2 points)

6) Ecrire une fonction qui retourne l'heure sous forme d'entier à partir d'une date définie par une chaîne de caractères au format "`hh:mm:ss`". Par exemple, pour "`14:55:34`", la fonction doit retourner 14 sous forme d'entier. L'instruction `int("14")` convertit une chaîne de caractères en un entier. (2 points)

7) Calculer le nombre de documents téléchargés pour chaque heure de la journée. Le site est-il consulté plutôt le matin ou le soir ? Serait-il possible de conclure aussi rapidement pour un site d'audience internationale ? (4 points)

Correction

```
# coding: latin-1
# la première ligne autorise les accents dans un programme Python
# la langue anglaise est la langue de l'informatique,
# les mots-clés de tous les langages
# sont écrits dans cette langue.

####################################
# exercice 1
####################################
#

# question 1
def lit_fichier (file) :
    f = open (file, "r")
    li = f.readlines ()           # découpage sous forme de lignes
    f.close ()
    res = []
    for l in li :
        s = l.replace ("\n", "")
        s = s.split (" ")         # le séparateur des colonnes est l'espace
        res.append (s)
    return res


mat = lit_fichier ("logpdf.txt")
for m in mat [0:5] :       # on affiche les 5 premières lignes
    print m                # parce que sinon, c'est trop long

# question 2
def compte_date (mat) :
    d = { }
    for m in mat :
        date = m [1]   # clé
        if date in d : d [date] += 1
        else : d [date] = 1
    return d

dico_date = compte_date (mat)
print dico_date
```

```
# remarque générale : si le fichier logpdf.txt contient des lignes
# vides à la fin, il se produira une erreur à la ligne 34 (date = m [1])
# car la matrice mat contiendra des lignes avec une seule colonne et non quatre.
# Il suffit soit de supprimer les lignes vides du fichier logpdf.txt
# soit de ne pas les prendre en compte lors de la lecture de ce fichier.

# question 3
# La méthode sort trie la liste mais comment ?
# Il est facile de trier une liste de nombres mais une liste de couples de
# nombres ? L'exemple montre que la liste est triée selon le premier élément de
# chaque couple. Pour les cas où deux couples ont un premier élément en commun,
# les éléments semblent triés selon le second élément. L'exemple suivant le monte :

l = [(1, "un"), (3, "deux",), (2, "deux"), (1, "hun"), (1, "un"), (-1, "moinsun")]
l.sort (reverse = True)
print l  # affiche [(3, 'deux'), (2, 'deux'), (1, 'un'),
         #          (1, 'un'),   (1, 'hun'), (-1, 'moinsun')]

# question 4
def dix_meilleures (dico) :
    # dans cette fonction on crée une liste de couples (valeur,clé) ou
    # la clé représente une date et valeur le nombre de téléchargement
    # pour cette date
    li = []
    for d in dico :
        cle = d
        valeur = dico [cle]
        li.append ( ( valeur, cle ) )
    li.sort (reverse = True)
    return li [0:10]

dix = dix_meilleures (dico_date)
print dix # la première date est (283, '26/Sep/2007')

# les quatre premières dates correspondent aux quatre premiers TD en 2007 à l'ENSAE

# question 5
# la date est en colonne 1, le document en colonne 3
# on fait un copier-coller de la fonction compte_date en changeant un paramètre
def compte_document (mat) :
    d = { }
    for m in mat :
        doc = m [3]   # clé, 3 au lieu de 1 à la question 2
        if doc in d : d [doc] += 1
        else : d [doc] = 1
    return d

dix = dix_meilleures ( compte_document (mat) )
print dix   # le premier document est
            # (323, '/mywiki/Enseignements?.....target=python_cours.pdf'),

# question 6
def heure (s) :
    hs = s [0:2]           # on extrait la partie correspondant à l'heure
    return int (hs)        # on retourne la conversion sous forme d'entiers

# question 7
# on recommence avec un copier-coller
def compte_heure (mat) :
    d = { }
    for m in mat :
        h = m [2]   # clé, 2 au lieu de 1 à la question 2
        cle = heure (h)
        if cle in d : d [cle] += 1
```

```
        else : d [cle] = 1
    return d

h = compte_heure (mat)
dix = dix_meilleures ( h )
print dix   # la première heure est  (432, 17), ce qui correspond à l'heure des TD

for i in h :
    print i, "h  ", h [i]

# Il y a beaucoup plus de téléchargement entre 20h et 2h du matin
# que le matin avant 10h.
# Le site est plutôt consulté le soir.
# La conclusion ne serait pas aussi évidente avec un site consulté par des gens
# du monde entier puisque 6h du matin est une heure de l'après midi au Japon.
# Il faudrait croiser l'heure avec la position géographique de la personne
# qui consulte le site.
```

fin exo 4 □

2.4 Année 2009

Le président de la république souhaite créer une pièce de monnaie à son effigie à condition que celle-ci facilite la vie des citoyens. Il voudrait que celle-ci soit d'un montant compris entre 1 et 10 euros et qu'elle permette de construire la somme de 99 euros avec moins de pièces qu'actuellement. Les questions qui suivent ont pour but de décomposer en pièces n'importe quel montant avec un jeu de pièces donné.

1) On considère que `pieces` est une liste de pièces triées par ordre croissant. Compléter le programme suivant afin d'obtenir la liste triée par ordre décroissant [4]. (2 points)

```
pieces = [1,2,5,10,20,50]
...
```

2) On souhaite écrire une fonction qui prend comme argument un montant `m` et une liste `pieces` toujours triée en ordre décroissant. Cette fonction doit retourner la plus grande pièce inférieure ou égale au montant `m`. (3 points)

3) En utilisant la fonction précédente, on veut décomposer un montant `m` en une liste de pièces dont la somme est égale au montant. On construit pour cela une seconde fonction qui prend aussi comme argument un montant `m` et une liste de pièces `pieces`. Cette fonction retourne une liste de pièces dont la somme est égale au montant. Une même pièce peut apparaître plusieurs fois. L'algorithme proposé est le suivant :

1. La fonction cherche la plus grande pièce inférieure ou égale au montant.
2. Elle ajoute cette pièce au résultat puis la soustrait au montant.
3. Si la somme restante est toujours positive, on retourne à la première étape.

Cet algorithme fonctionne pour le jeu de pièces donné en exemple et décompose 49 euros en $49 = 20 + 20 + 5 + 2 + 2$. (5 points)

4) Prolongez votre programme pour déterminer avec cet algorithme le plus grand montant compris entre 1 et 99 euros inclus et qui nécessite le plus grand nombre de pièces ? (2 points)

5) On ajoute la pièce 4 à la liste des pièces :

```
pieces = [1,2,4,5,10,20,50]
```

Que retourne l'algorithme précédent comme réponse à la question 3 avec cette nouvelle liste et pour le montant 98 ? Est-ce la meilleure solution ? (2 points)

4. On peut s'aider de tout type de document. Il est possible d'utiliser les méthodes associées à la classe `list` qu'on peut aussi retrouver via un moteur de recherche internet avec la requête *python list*.

6) Comme l'algorithme précédent ne fournit pas la bonne solution pour tous les montants, il faut imaginer une solution qui puisse traiter tous les jeux de pièces. On procède par récurrence. Soit $P = (p_1, \ldots, p_n)$ l'ensemble des pièces. On définit la fonction $f(m, P)$ comme étant le nombre de pièces nécessaire pour décomposer le montant m. On vérifie tout d'abord que :

1. La fonction $f(m, P)$ doit retourner 1 si le montant m est déjà une pièce.
2. La fonction $f(m, P)$ vérifie la relation de récurrence suivante :

$$f(m, P) = \min \{f(m - p, P) + 1 \text{ pour tout } p \in P\}$$

Cet algorithme permet de construire la meilleure décomposition en pièces pour tout montant. La dernière expression signifie que si le montant m est décomposé de façon optimale avec les pièces (p_1, p_2, p_3, p_4) alors le montant $m - p_4$ est aussi décomposé de façon optimale avec les mêmes pièces (p_1, p_2, p_3).

Il ne reste plus qu'à implémenter la fonction [5] $f(m, P)$. (5 points)

7) Qu'obtient-on avec le jeu de pièces `pieces` = $[1, 2, 4, 5, 10, 20, 50]$? Prolongez le programme pour déterminer tous les choix possibles du président parmi les pièces [3,4,6,7,8,9] ? (1 point)

8) Quel est le coût de votre algorithme ? (facultatif)

Correction

```
# coding: latin-1

###################################
# question 1 : retourner un tableau
###################################

pieces = [1,2,5,10,20,50]
pieces.reverse ()
print pieces # affiche [50, 20, 10, 5, 2, 1]

# il existait d'autres solutions
pieces.sort (reverse = True)

# ou encore l'utilisation d'une fonction compare modifiée
# qui s'inspire de l'exemple
# http://www.xavierdupre.fr/enseignement/initiation/...
# ...initiation_via_python_ellipse/chap2_type_tex_-_tri-_3.html

# on encore un tri programmé
# http://www.xavierdupre.fr/enseignement/initiation/...
# ...initiation_via_python_ellipse/chap3_syntaxe_tex_-_tri-_27.html

#####################################################################
# question 2 : trouve la plus grande pièce inférieure à un montant
#####################################################################

def plus_grande_piece (montant, pieces) :
    # on suppose que les pièces sont triées par ordre décroissant

    for p in pieces :
        if p <= montant : return p

    # on peut ajouter la ligne
    return 0
    # qui correspond au fait qu'aucune pièce plus petite ou égale au montant
    # n'a été trouvé --> le montant est négatif ou nul

# on vérifie
```

5. Même si l'algorithme est présenté de façon récurrente, il peut être implémenté de façon récurrente ou non. La méthode récurrente est toutefois déconseillée car beaucoup plus lente.

```
print "plus_grande_piece (80, pieces) =", plus_grande_piece (80, pieces)
                  # affiche 50

####################################
# question 3 : décomposer un montant
####################################

def decomposer (montant, pieces) :
    # on suppose que les pièces sont triées par ordre décroissant
    res = [ ]  # contiendra la liste des pièces pour le montant
    while montant > 0 :
        p = plus_grande_piece (montant, pieces)  # on prend la plus grande pièce
                                                 # inférieure ou égale au montant
        res.append (p)                           # on l'ajoute à la solution
        montant -= p                             # on ôté p à montant :
                                      # c'est ce qu'il reste encore à décomposer
    return res

print "decomposer (98, pieces) =", decomposer (98, pieces)
                # affiche [50, 20, 20, 5, 2, 1]

print "decomposer (99, pieces) =", decomposer (99, pieces)
                # affiche [50, 20, 20, 5, 2, 2]

#####################################################
# question 4 : trouver la décomposition la plus grande
#####################################################

def maximum_piece (pieces) :
    # détermine le nombre maximum de pièces à utiliser
    maxi    = 0
    montant = 0
    for m in range (1, 100) :
        r = decomposer (m, pieces)
        if len (r) >= maxi :    # si on remplace cette ligne par if len (r) > maxi :
            maxi    = len (r)   # on trouve le plus petit montant
                                # avec la pire décomposition
            montant = m         # et non le plus grand montant
                                # avec la pire décomposation
    return maxi, montant

print "maximum_piece (pieces) =", maximum_piece (pieces) # affiche (6, 99)

#################################
# question 5 : décomposition de 98
#################################

pieces4 = [1,2,4,5,10,20,50]
pieces4.reverse ()

print "decomposer (98, pieces) = ", decomposer (98, pieces)    # [50, 20, 20, 5, 2, 1]
print "decomposer (98, pieces4) = ", decomposer (98, pieces4)  # [50, 20, 20, 5, 2, 1]

"""
Les deux décompositions sont identiques.
Or il existe une décomposition plus courte avec la pièce 4 :
98 = 50 + 20 + 20 + 4 + 4 = 5 pièces

L'algorithme fait la même erreur lorsqu'il décompose le montant 8.
Il cherche toujours la plus grande pièce inférieure au montant qui est 5.
Il lui est alors impossible d'utiliser la pièce 4  pour décomposer 8.

Cet algorithme ne fournit pas la bonne solution avec ce nouveau jeu de pièces.
"""
```

```
################################
# question 6 : algorithme optimal
################################

# version récursive : très longue
def decomposer_optimal (montant, pieces) :
    if montant in pieces :
        return [ montant ]
    else :
        r = [ 1 for m in range (0, montant) ]
        for p in pieces :
            if montant > p : # si ce test n'est pas fait, la récurrence peut être infinie
                             # car les montants négatifs ne sont pas pris en compte
                             # par le premier test
                dec = decomposer_optimal (montant - p, pieces) + [p]
                if len (dec) < len (r) :
                    r = dec

        return r

# print "decomposer_optimal (98, pieces4) =", decomposer_optimal (98, pieces4)
# trop long

# version non récursive
def decomposer_optimal (montant, pieces) :
    memo = [ [ 1 for l in range (0, m) ] for m in range (0, montant+1) ]
    # memo [i] contient la pire décomposition du montant i (que des pièces de un)

    # pour les pièces de pieces, on sait faire plus court
    for p in pieces :
        if p < len (memo) :
            memo [p] = [ p ]

    for m in range (1, montant+1) :
        for p in pieces :
            if m > p :
                # on calcule la nouvelle décomposition
                dec = [p] + memo [m-p]
                # si elle est plus courte, on la garde
                if len (dec) < len (memo [m] ) :
                    memo [m] = dec

    # on retourne la meilleur décomposition pour montant
    return memo [ montant ]

# beaucoup plus rapide
print "decomposer_optimal (98, pieces4) =",  decomposer_optimal (98, pieces4)
             # affiche [50, 20, 20, 4, 4]


########################
# question 7 : résultat
########################


# pour trouver la décomposition la plus longue avec n'importe quel jeu de pièces
# on reprend la fonction maximum_piece et on remplace decomposer par decomposer optimale

def maximum_piece (pieces) :
    # détermine le nombre maximum de pièces à utiliser
    maxi    = 0
    montant = 0
    for m in range (1, 100) :
        r = decomposer_optimal (m, pieces)
        if len (r) >= maxi :    # si on remplace cette ligne par if len (r) > maxi :
```

```
            maxi    = len (r)   # on trouve le plus petit montant
                                # avec la pire décomposation
            montant = m         # et non le plus grand montant
                                # avec la pire décomposation
    return maxi, montant

print "maximum_piece (pieces) =", maximum_piece (pieces) # affiche (6, 99)
print "maximum_piece (pieces4) =", maximum_piece (pieces4) # affiche (5, 99)


# on teste pour toutes les pièces [3,4,6,7,8,9]
# ajoutées au jeu de pièces standard [1,2,5,10,20,50]
ensemble = [3,4,6,7,8,9]

for ajout in [3,4,6,7,8,9] :
    pieces = [1,2,5,10,20,50] + [ ajout ]
    pieces.sort (reverse = True)
    print "maximum_piece (" + str (pieces) + ") = ", maximum_piece (pieces)

# résultat :
"""
maximum_piece ([50, 20, 10, 5, 3, 2, 1]) =  (6, 99)
maximum_piece ([50, 20, 10, 5, 4, 2, 1]) =  (5, 99)  # 4, ok
maximum_piece ([50, 20, 10, 6, 5, 2, 1]) =  (6, 99)
maximum_piece ([50, 20, 10, 7, 5, 2, 1]) =  (5, 99)  # 7, ok
maximum_piece ([50, 20, 10, 8, 5, 2, 1]) =  (5, 99)  # 8, ok
maximum_piece ([50, 20, 10, 9, 5, 2, 1]) =  (5, 98)  # 9, ok
"""


############################
# question 8 : décomposition
############################

"""
On cherche ici le coût de la fonction decomposer_optimal en fonction du montant.
Il n'y a qu'une seule boucle qui dépend du montant, le coût de la fonction est en O(n).

Dans la version récursive, le coût est le résultat d'une suite récurrente :
u(n) = u(n-1) + ... + u(n-d)
où d est le nombre de pièces.

Le coût est donc en O(a^n) où a est la plus grande des racines du polynôme :

P (x) = x^d - x^(d-1) - ... - 1

P(1) < 0 et lim P(x) = infini lorsque x tend vers infini,
donc la plus grande des racines est supérieure à 1.
Le coût de la fonction decomposer_optimal récursive est en O(a^n) avec 1,96 < a < 1,97.

Pour des explications plus conséquentes, voir la page
http://fr.wikipedia.org/wiki/Suite_r%C3%A9currente_lin%C3%A9aire
sur les suites récurrentes et l'exercice 12.3.3 du livre :
http://www.xavierdupre.fr/enseignement/initiation/...
 ...initiation_via_python_ellipse.pdf
(ou 13.3.3 http://www.xavierdupre.fr/enseignement/...
 ...initiation/initiation_via_python_small.pdf).
"""
```

L'algorithme qui décompose un montant de façon optimale quelque soient le montant et le jeu de pièces s'apparente à la programmation dynamique. C'est le même algorithme que celui qui permet de chercher le plus court chemin dans un graphe où les nœuds serait les montants de 0 à 99 et où il y aurait autant d'arcs que de pièces partant de chaque nœuds. Les arcs seraient tous de même poids : 1. Avec cette description, trouver la meilleure décomposition revient à trouver le chemin incluant le moins d'arcs possible.

fin exo 5 □

2.5 Année 2009, rattrapage

Cet exercice propose de fusionner des données provenant de deux sources. La première étape consiste à télécharger deux fichiers disponibles à l'adresse suivante : http://www.xavierdupre.fr/mywiki/InitiationPython :

td_note_2009_cluedo_1.txt et td_note_2009_cluedo_2.txt.

Il est préférable de placer ces pièces jointes dans le même répertoire que le progamme lui-même. Le premier fichier contient une matrice dont chaque ligne contient 4 informations séparées par des tabulations :

un nom un prénom une couleur une arme

Les premières lignes sont les suivantes [6] :

```
N10   P10   gris     corde
N15   P15   rouge    chandelier
N8    P8    marron   fer à cheval
```

Le second fichier contient une matrice dont chaque ligne contient 3 informations séparées par des tabulations :

un nom un prénom une pièce

Les premières lignes sont les suivantes [7] :

```
N3    P3    cuisine
N19   P19   bureau
N20   P20   salle à manger
```

Certaines élèves sont présents dans les deux fichiers, l'objectif de cet exercice et de construire un troisième fichier regroupant les informations concernant les élèves présents dans les deux fichiers. Ce fichier contiendra donc 5 colonnes.

1) Construire une fonction qui prend comme argument un nom de fichier et qui retourne le contenu du fichier dans une matrice (une liste de listes) [8].

```
def convertit_fichier_en_matrice (fichier) :
...
```

2) Construire une fonction qui prend comme entrée une matrice et qui construit un dictionnaire dont :
– la clé est un tuple composé des valeurs des deux premières colonnes (nom, prénom)
– la valeur est la liste des valeurs des autres colonnes

```
def convertit_matrice_en_dictionnaire (matrice) :
...
```

3) Construire une fonction qui effectue la fusion de deux dictionnaires. Le dictionnaire résultant vérifie :
– chaque clé est présente dans les deux dictionnaires à la fois

6. Ces données sont complètement aléatoires.
7. Ces données sont aussi complètement aléatoires.
8. On pourra s'inspirer du programme présent à l'adresse http://www.xavierdupre.fr/enseignement/initiation/initiation_via_python_all/chap7_fichiers_200_.html.

– la valeur associée à une clé est la somme des deux listes associées à cette même clé dans chacun des deux dictionnaires.

```
def fusion_dictionnaire (dico1, dico2) :
```

4) Construire une fonction qui convertit un dictionnaire en matrice. C'est la fonction réciproque de la fonction de la question 2. Les deux premières colonnes sont celles de la clé, les suivantes sont celles des valeurs.

```
def convertit_dictionnaire_en_matrice (dico) :
```

5) Ecrire une fonction qui écrit une matrice dans un fichier texte.

```
def convertit_matrice_en_fichier (mat, fichier) :
```

6) En utilisant toutes les fonctions précédentes, écrire une fonction qui effectue la fusion de deux fichiers et qui écrit le résultat dans un autre fichier.

```
def fusion_fichier (fichier1, fichier2, fichier_resultat) :
```

7) En vous inspirant du schéma proposé dans les précédentes questions, il s'agit ici de produire un troisième fichier qui contient tous les couples (nom, prénom) qui sont présents dans un fichier mais pas dans les deux à la fois.

```
def union_moins_intersection_fichier (fichier1, fichier2, fichier_resultat) :
```

8) Serait-il compliqué d'écrire une fonction qui permette de fusionner 3 fichiers ? n fichiers ? Justifier.

A quoi ça sert : cette opération de fusion peut être effectuée sous Excel lorsque la taille des fichiers n'excède pas 65000 lignes. Au delà, cet énoncé propose un moyen de fusionner des fichiers de plusieurs millions de lignes.

Correction

```
# coding: latin-1
# Question 1

def convertit_fichier_en_matrice (file):
    f = open (file, "r")
    l = f.readlines ()
    f.close ()

    mat = list()
    for s in l :
        l = s.strip ("\n\r").split ("\t")
        mat.append (l)
    return mat

# Question 2

def convertit_matrice_en_dictionnaire (matrice) :
    d={}
    for line in matrice :
        d[line[0],line[1]] = line [2:]
    return d

# Question 3
```

```
def fusion_dictionnaire (dico1, dico2) :
    dico={}
    for k in dico1 :
        if k in dico2 : dico[k] = dico1[k] + dico2[k]
    return dico

# Question 4

def convertit_dictionnaire_en_matrice (dico) :
    m =[]
    for k,v in dico.iteritems () :
        line = list (k) + v
        m.append (line)
    return m

# Question 5

def convertit_matrice_en_fichier (mat,nomfichier):
    f = open (nomfichier, "w")
    for line in mat :
        f.write ( "\t".join ( [ str (x) for x in line ] ) + "\n")
    f.close ()

# Question 6

def fusion_fichier (fichier1, fichier2, fichier_resultat) :
    matrice1    = convertit_fichier_en_matrice(fichier1)
    matrice2    = convertit_fichier_en_matrice(fichier2)
    dico1       = convertit_matrice_en_dictionnaire(matrice1)
    dico2       = convertit_matrice_en_dictionnaire(matrice2)
    dico        = fusion_dictionnaire (dico1,dico2)
    matrice     = convertit_dictionnaire_en_matrice(dico)
    convertit_matrice_en_fichier (matrice,fichier_resultat)

fusion_fichier (    "td_note_2009_cluedo_1.txt",
                    "td_note_2009_cluedo_2.txt",
                    "cluedo.txt")

# Question 7

def fusion_dictionnairebis (dico1,dico2) :
    l1=dico1.keys()
    l2=dico2.keys()
    dico={}
    for k in l1 :
        if k not in l2 : dico[k]=[]
    for k in l2 :
        if k not in l1 : dico[k]=[]
    return dico

def union_moins_intersection_fichier (fichier1, fichier2, fichier_resultat):
    matrice1    = convertit_fichier_en_matrice(fichier1)
    matrice2    = convertit_fichier_en_matrice(fichier2)
    dico1       = convertit_matrice_en_dictionnaire(matrice1)
    dico2       = convertit_matrice_en_dictionnaire(matrice2)

    dico        = fusion_dictionnairebis (dico1,dico2)

    matrice     = convertit_dictionnaire_en_matrice(dico)
    convertit_matrice_en_fichier (matrice,fichier_resultat)

union_moins_intersection_fichier (  "td_note_2009_cluedo_1.txt",
                                    "td_note_2009_cluedo_2.txt",
                                    "cluedo2.txt")
```

```
# Question 8
"""
Il suffit de fusionner les fichiers deux par deux
en procédant par récurrence. On fusionne d'abord les
deux premiers, puis on fusionne le troisième
au résultat de la première fusion...
"""
```

fin exo 6 □

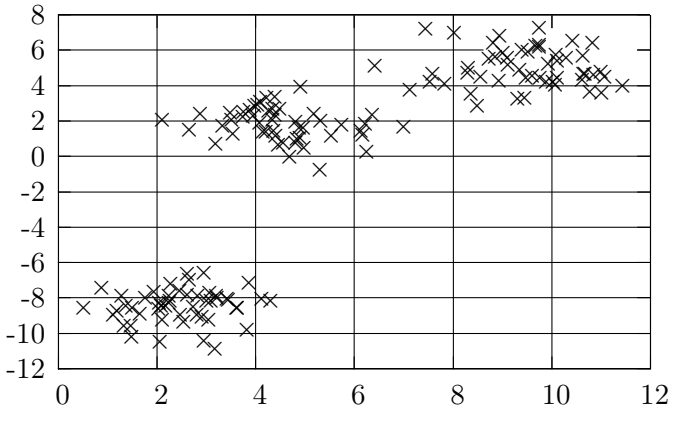
2.6 Année 2010

Glossaire :

barycentre Un barycentre est le centre d'un ensemble de points $(x_i, y_i)_{1 \leqslant i \leqslant n}$. Dans le plan, il a deux coordonnées (X, Y) égales à $X = \frac{1}{n}\sum_i x_i$ et $Y = \frac{1}{n}\sum_i y_i$. Ses coordonnées le placent au milieu des points dans il est le barycentre : il est situé dans l'enveloppe convexe formée par l'ensemble des points.

centree En dimension deux, même si c'est une expression employée dans la suite de l'énoncé, une loi normale de centre (x, y) n'est pas une expression correcte. On devrait dire une loi normale de moyenne (x, y). De même, cette loi n'a pas une variance $\sigma \in \mathbb{R}$, on devrait dire une variance $\begin{pmatrix} \sigma & 0 \\ 0 & \sigma \end{pmatrix}$.

Les nuées dynamiques servent à construire automatiquement des classes dans un ensemble d'observations. C'est une façon de regrouper entre elles des observations qui sont proches les unes des autres. Prenons par exemple le nuage de points suivant qui inclut trois sous-nuages.



Le nuage de points contient trois sous-ensembles de points. Chacun est un ensemble de points simulés selon une loi normale de variance 1 et de moyenne identique à l'intérieur d'un sous-ensemble.

1) La fonction `gauss` du module `random` permet de générer un nombre selon une loi normale. Le premier objectif est de créer une fonction qui retourne un ensemble de points simulés selon une loi normale de variance v et de centre (x, y). (2 points)

```
def sous_nuage (nb, x, y, v) :  # retourne une liste de 2-uples
```

2) On cherche à créer un nuage regroupant n sous-nuages de même variance 1 avec la fonction précédente. Chaque sous-nuage est centré autour d'une moyenne choisie aléatoirement selon une loi de votre choix. La fonction dépend de deux paramètres : le nombre de points dans chaque classe et le nombre de classes. (2 points)

```
def n_sous_nuages (n, nb) :     # retourne une liste de 2-uples
```

3) Dessiner le nuage avec le module `matplotlib` pour vérifier que le résultat correspond à vos attentes. On pourra s'appuyer sur l'extrait qui suit. (1 point)

```
import matplotlib.pyplot as plt
x   = [ ... ]
y   = [ ... ]
fig = plt.figure()
ax  = fig.add_subplot(111)
ax.plot (x,y, 'o')
plt.savefig ("im1.png") # ou plt.show () pour afficher le graphe
```

4) L'algorithme des nuées dynamiques commence par affecter chaque point à une classe choisie au hasard. Pour cette tâche, on pourra utiliser la fonction `randint` du module `random`. On veut créer une fonction qui retourne une classe aléatoire pour chaque point du nuage. Elle doit prendre comme entrée le nombre de classes souhaité. (2 points)

```
def random_class (points, n) :  # retourne une liste d'entiers
```

5) L'algorithme des nuées dynamiques répète ensuite alternativement deux étapes :

Etape 1 On calcule le barycentre de chaque classe.
Etape 2 On associe à chaque point la classe dont le barycentre est le plus proche (au sens de la distance euclidienne).

On propose de commencer par écrire une fonction qui retourne pour un point donné le barycentre le plus proche. (2 points)

```
def proche_barycentre (point, barycentres) :   # retourne un entier
```

6) La fonction suivante retourne le barycentre le plus proche pour chaque point. (2 points)

```
def association_barycentre (points, barycentres) :  # retourne une liste d'entiers
```

7) On découpe la première étape de la même façon :

1. Première fonction : calcule le barycentre d'une classe.
2. Seconde fonction : calcule le barycentre de toutes les classes.

Il faut implémenter ces deux fonctions. (3 points sans utiliser `numpy`, 4 points avec `numpy` et une fonction).

```
def barycentre_classe (points, classes, numero_class) :   # retourne un 2-uple
def tous_barycentres  (points, classes) :       # retourne une liste de 2-uples
```

8) L'algorithme commence par la création des classes (fonction `n_sous_nuages`) et l'attribution d'une classe au hasard (fonction `random_class`). Il faut ensuite répéter les fonctions `tous_barycentres` et `association_barycentre`. L'enchaînement de ces opérations est effectué par la fonction `nuees_dynamiques`. (2 points)

```
def nuees_dynamiques (points, nombre_classes) : # retourne une liste d'entiers
```

9) Dessiner le résultat permettra de vérifier que tout s'est bien passé, toujours avec un code similaire à celui-ci. (2 points)

```
import matplotlib.pyplot as plt
x1  = [ ... ]
y1  = [ ... ]
x2  = [ ... ]
y2  = [ ... ]
fig = plt.figure()
ax  = fig.add_subplot(111)
ax.plot (x1,y1, 'o')
ax.plot (x2,y2, 'x')     # ligne ajoutée, 'x', 'bo', ...
plt.savefig ("im2.png")  # 'rx', 'go', 'gs', 'bs', ...
```

10) Question facultative : comment savoir quand s'arrêter ? (0 point)

Correction

Le corrigé final apparaît après les commentaires qui suivent. Ils sont inspirés des réponses des élèves.

1) 2) Le centre de chaque sous-nuages a été généré selon diverses lois aléatoires, des lois normales, uniformes réelles ou discrètes.

```
def n_sous_nuages (n, nb):
    m = []
    for i in range (0,n):
        x = 5*random.random()
        y = 5*random.random()
        d = sous_nuage(nb,x,y,1)
        m += d
    return m
```

```
def n_sous_nuages (n, nb):
    m = []
    for i in range (0,n):
        x = random.randint(0,20)
        y = random.randint(0,20)
        d = sous_nuage(nb,x,y,1)
        m += d
    return m
```

L'exemple de droite utilise la même loi pour générer aléatoirement à la fois le centre de chaque nuage et les points qu'ils incluent. Il sera alors difficile de distinguer visuellement plusieurs sous-nuages avec le graphe dessiné à la question suivante.

```
def n_sous_nuages (n, nb):
    m = []
    for i in range (0,n):
        x = random.gauss(0,1)
        y = random.gauss(0,1)
        d = sous_nuage(nb,x,y,1)
        m += d
    return m
```

Quels que soient les points simulés, les réponses aux questions suivantes n'en dépendaient pas. L'algorithme des centres mobiles s'appliquent à n'importe quel ensemble de points bien que le résultat ne soit pas toujours pertinent.

Certains élèves ont ajouté [d] au lieu de `d` seul. Au lieu d'obtenir comme résultat une liste de 2 coordonnées (une matrice de deux colonnes), le résultat est alors une liste de matrices de deux colonnes : c'est un tableau à trois dimensions. Ce n'est pas faux mais cela complique inutilement l'écriture des fonctions qui suivent en ajoutant une boucle à chaque fois qu'on parcourt l'ensemble des points.

```
def n_sous_nuages (n, nb):
    m = []
    for i in range (0,n):
        x = random.gauss(0,1)
        y = random.gauss(0,1)
        d = sous_nuage(nb,x,y,1)
        m +=  [ d ]
           # le résultat n'est
           # plus une liste
    return m
```

3) Utiliser l'exemple de l'énoncé n'a pas posé de problème excepté un cas particulier :

```
import matplotlib.pyplot as plt
x = [ p [0] for p in n_sous_nuages (3,50) ]
y = [ p [1] for p in n_sous_nuages (3,50) ]
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot (x,y, 'o' )
plt.savefig ("im1.png")
```

Dans cet exemple, la fonction `n_sous_nuages` est appelée une fois pour extraire les abscisses, une seconde fois pour extraire les ordonnées. Etant donné que cette fonction retourne un résultat aléatoire, il est très peu probable qu'elle retourne deux fois le même résultat. Par conséquence, les abscisses et les ordonnées ne proviennent pas du même nuage : le graphique résultant ne montrera pas trois nuages séparés.

4) La fonction `randint(a, b)` retourne un nombre entier aléatoire compris entre `a` et `b` **inclus**. Il fallait donc bien choisir `a` et `b`. Le meilleur choix était $a = 0$ et $b = n - 1$. Un autre choix assez fréquent était $a = 1$ et $b = n$ comme dans l'exemple suivant :

```
def random_class(l,n):
    l = []
    for i in range(0,len(l)):
        l += [ random.randint (1,n) ]
    return l
```

Les deux réponses sont correctes. Toutefois, la solution ci-dessus implique de faire un peu plus attention par la suite car elle complique la correspondance entre les barycentres et le numéro de la classe qu'il représente. En effet, qu'en est-il de la classe 0 dans ce cas. Dans cet exemple, la fonction `random_class` n'associe aucun point à la classe 0. On peut alors se demander à quoi correspond le premier élément du tableau `barycentre` utilisé dans les fonctions des questions suivantes. Quoi qu'il en soit, la fonction `proche_barycentre` retourne l'indice du barycentre le plus proche, pas le numéro de la classe à laquelle il correspond. Selon les programmes, avec un peu de chance, les numéros des classes ont commencé à 0 après le premier appel à la fonction `proche_barycentre`. Le calcul du barycentre de la première classe amène une division par zéro à moins que ce cas ait été pris en compte.

Dans l'exemple suivant, on tire une classe aléatoire parmi $n + 1$ numéros. Il y a donc une classe de plus qu'attendu mais là encore, cette erreur peut être compensée par une autre plus loin.

```
def random_class(l,n):
    l = []
    for i in range(0,len(l)):
        l += [ random.randint (0,n) ]
    return l
```

Un élève a fait une allusion à la probabilité qu'une classe soit vide : un numéro entre 0 et $n - 1$ n'est jamais attribué. On peut déjà se pencher sur la probabilité que la classe 0 soit vide. Chaque point a un probabilité $\frac{1}{n}$ d'être associé à la classe 0. La probabilité cherchée est donc : $\left(\frac{n-1}{n}\right)^N$ où N est le nombre de points. On peut ainsi majorer la probabilité qu'une classe soit vide par : $n\left(\frac{n-1}{n}\right)^N$.

5) La fonction `proche_barycentre` a été plutôt bien traitée malgré deux erreurs fréquentes. La première concerne la fonction puissance lors du calcul de la distance euclidienne :

```
d= (  (p[0]-f[0])**2+(p[1]-f[1])**2 ) ** (1/2)
```

Dans l'exemple précédente, `1/2` est une division entière et son résultat est nul. Comme $\forall x$, $x^0 = 1$ (pour *Python* du moins), toutes les distances calculées sont donc égales à 1. Il faut noter que la racine carrée

n'était pas indispensable puisqu'on cherchait le barycentre le plus proche : seule la plus petite valeur comptait et non la valeur elle-même.

L'autre erreur fréquente est celle-ci :

```
def proche_barycentre (point,barycentres):
    d=distance_euclidienne(point,barycentres[0])
    for i in range (0,len(barycentres)):
        if distance_euclidienne(point,barycentres[i])<=d:
            d=distance_euclidienne(point,barycentres[i])
    return d
```

On retourne non pas l'indice du barycentre le plus proche mais la distance de ce barycentre au point considéré.

6) Cette question a été bien traitée. Les erreurs introduites dans la fonction précédentes se sont propagées sans provoquer d'erreur d'exécution.

7) Dans la fonction suivante, si la plupart ont pensé à ne prendre que les points de la classe `numero_class`, ils ont parfois oublié de diviser par le bon nombre d'éléments. Ici, c'est la variable `n` qui n'est définie nulle part. Si le programme ne provoque pas d'erreurs, c'est donc une variable globale déclarée avant.

```
def barycentre_classe (points, classes, numero_class):
    x=0
    y=0
    for i in range (0,len(classes)):
        if classes[i]==numero_class:  # ligne importante
            l=point[i]
            x=x+l[0]
            y=y+l[1]
    c=[x/n,y/n]                       # ligne importante
    return c
```

La variable `n` a parfois été remplacée par `len(classes)` qui est aussi faux puisque cela correspond au nombre total de points et non celui de la classe `numero_class`.

Il arrive que la fonction provoque une division par zéro lorsqu'une classe est vide. C'est un cas à prendre en compte. L'algorithme peut alors évoluer dans deux directions. La première consiste à supprimer la classe. Le second choix évite la disparition d'une classe en affectant un ou plusieurs points désignés aléatoirement à la classe disparue. L'énoncé ne demandait à ce qu'il en soit tenu compte même si cela serait souhaitable.

La fonction `tous_barycentre` a été victime de deux erreurs. La première est la suivante où on construit autant de barycentres qu'il y a de points alors qu'on souhaite autant de barycentres qu'il y a de classes :

```
def tous_barycentres (points,classes):
    c=[]
    for i in classes :   # or on a len(classes) == len(points)
        c+=[barycentre_classe (points,classes,i)]
    return c
```

La seconde erreur intervient lors du numéro de classes et fait écho à la fonction `randon_class` et ses erreurs. Dans l'exemple suivant, la classe dont le numéro est le plus grand a pour numéro `max(classes)`. Or dans la boucle `for i in range(0,mx)` :, elle est oubliée car la fonction `range` va jusqu'à `mx - 1` inclus. Il aurait fallu écrire `mx = max(classes) + 1`. Dans le cas contraire, on perd une classe à chaque fois qu'on appelle la fonction `tous_barycentres`.

```
def tous_barycentres (points,classes):
    c=[]
```

```
    mx=max(classes)        # il faut ajouter +1
    for i in range(0,mx) :
        c+=[barycentre_classe (points,classes,i)]
    return c
```

Il faut noter également que la classe 0 reçoit un barycentre après la fonction `tous_barycentres` même si la fonction `random_class` ne lui en donnait pas lorsqu'elle utilise l'instruction `random.randint(1,n)`.

Peu d'élèves ont utilisé le module `numpy`. Son usage avait pour but d'éviter une boucle sur les points : elle ne disparaît pas mais est prise en charge par les fonctions de calcul matriciel proposées par le module `numpy`. Cette boucle a persisté dans la grande majorité des solutions envisagées. La difficulté réside dans la construction d'une ou deux matrices qui mènent au calcul des barycentre par quelques manipulations matricielles. La correction qui suit présente deux implémentations dont les seules boucles portent sur le nombre de classes.

8) La dernière fonction de l'algorithme de classification a connu trois gros défauts. Le premier est l'oubli de la boucle qui permet de répéter les opérations plus d'une fois. Le second défaut apparaît lorsque le résultat de la fonction `association_barycentre` n'est pas utilisé. Dans l'exemple suivant, le calcul des barycentres a toujours lieu avec la liste `l` qui n'est jamais modifiée : le résultat `a` est toujours celui de l'algorithme après la première itération qui est répétée ici 10 fois exactement de la même manière.

```
def nuees_dynamiques (points,nombre_classes):
    l = random_class (points,nombre_classes)
    for j in range (0,10):
        c = tous_barycentres (points, l)
        a = association_barycentre (points,c)
                    # il faut ajouter ici l = a pour corriger la fonction
    return a
```

La dernière erreur suit la même logique : l'instruction `l = a` est bien présente mais son effet est annulé par le fait de générer aléatoirement un numéro de classe à chaque itération.

```
def nuees_dynamiques (points,nombre_classes):
    for j in range (0,10):
        l = random_class (points,nombre_classes)
        c = tous_barycentres (points, l)
        a = association_barycentre (points,c)
        l = a
    return a
```

Enfin, une erreur grossière est parfois survenue : l'exemple suivant change les données du problème à chaque itération. Le résultat a peu de chance de signifier quoi que ce soit.

```
def nuees_dynamiques (n,nb):
    for j in range (0,10):
        points = n_sous_nuage (n,nb)
        l = random_class (points,nombre_classes)
        c = tous_barycentres (points, l)
        a = association_barycentre (points,c)
        l = a
    return a
```

9) La dernière question fut plus ou moins bien implémentée, souvent très bien pour le cas particulier de deux classes. Le cas général où le nombre de classes est variable n'a pas été souvent traité. La correction complète suit.

```
# coding: latin-1
import random
import numpy

def dessin (nuage, image = None) :
    """dessine un nuage de points
    @param      nuage       le nuage de points
    @param      image       si None, on affiche le nuage au travers d'une fenêtre,
                            sinon, image correspond à un nom d'image
                            sur disque dur qui contiendra le graphique final"""
    import matplotlib.pyplot as plt
    x = [ p[0] for p in nuage ]
    y = [ p[1] for p in nuage ]
    plt.clf ()
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot (x,y, 'o')
    if image == None :  plt.show ()
    else :              plt.savefig (image)

def dessin_classes (nuage, classes, image = None) :
    """dessine un nuage, donne des couleurs différentes
    selon que le point appartient à telle ou telle classes
    @param      nuage           nuage[i], c'est le point i
    @param      classes         classes [i] est la classe associée au point i
    @param      image           voir la fonction précédente
    """
    import matplotlib.pyplot as plt
    x = {}
    y = {}
    for i in range (0, len (nuage)) :
        cl = classes [i]
        if cl not in x :
            x [cl] = []
            y [cl] = []
        x [cl].append ( nuage [i][0] )
        y [cl].append ( nuage [i][1] )
    plt.clf ()
    fig = plt.figure()
    ax = fig.add_subplot(111)
    for cl in x :
        ax.plot (x [cl], y [cl], "+")
    if image == None :  plt.show ()
    else :              plt.savefig (image)

def sous_nuage (nb, x, y) :
    """retourne un ensemble de points tirés aléatoirement selon
    une loi normale centrée autour du point x,y
    @param      nb              nombre de points
    @param      x               abscisse du centre
    @param      y               ordonnée du centre
    @return                     une liste de points ou matrice de deux colonnes
                                - la première correspond aux abscisses,
                                - la seconde aux ordonnées
    """
    res = []
    for i in xrange (0, nb) :
        xx = random.gauss (0,1)
        yy = random.gauss (0,1)
        res.append ( [x+xx, y+yy] )
    return res

def n_sous_nuages (nb_class, nb_point) :
    """crée un nuage de points aléatoires
```

```
    @param      nb_class        nombre de sous nuages
    @param      nb_point        nombre de points dans chaque sous nuage
    @return                     une liste de points ou matrice de deux colonnes
                                - la première correspond aux abscisses,
                                - la seconde aux ordonnées"""
    res = []
    for c in xrange (0, nb_class) :
        x = random.gauss (0,1) * 5
        y = random.gauss (0,1) * 5
        res += sous_nuage (nb_point, x,y)
    return res

def random_class ( nuage, n) :
    """choisis aléatoirement un entier pour chaque point du nuage
    @param      nuage           un nuage de points (matrice de deux colonnes)
    @param      n               nombre de classes
    @return                     une liste d'entiers
    """
    res = [ ]
    for p in nuage :
        c = random.randint (0, n-1)
        res.append (c)
    return res

def proche_barycentre (point, barycentres) :
    """détermine le barycentre le plus d'un point
    @param      point           liste de 2 réels : [x,y]
    @param      barycentres     liste de n points = matrice de deux colonnes,
                                chaque ligne correspond à un barycentre
    @return                     un entier qui correspond à l'index
                                du barycentre le plus proche"""
    dmax = 1e6
    for i in range (0, len (barycentres)) :
        b  = barycentres [i]
        dx = point [0] - b [0]
        dy = point [1] - b [1]
        d  = (dx**2 + dy**2) ** 0.5
        if d < dmax :
            dmax = d
            m    = i
    return m

def association_barycentre (points, barycentres) :
    """détermine pour chaque point le barycentre le plus proche
    @param      points          nuage (matrice de deux colonnes)
    @param      barycentres     c'est aussi une matrice de deux colonnes mais
                                avec moins de lignes
    @return                     liste d'entiers, chaque entier
                                correspond à la classe du point points[i],
                                c'est-à-dire l'index du barycentre le plus proche
                                ici:
                                point:      points [i]
                                classe:     res[i]
                                barycentre: barycentres[ res[i] ]
    """
    res = []
    for p in nuage :
        m = proche_barycentre (p, barycentres)
        res.append (m)
    return res

def barycentre_classe (points, classes, numero_class) :
    """calcule le barycentre d'une classe
    @param      points          ensemble de points (matrice de deux colonnes)
    @param      classes         liste d'entiers de même longueur,
```

```
                                chaque élément classes[i] est la classe de point[i]
    @param      numero_class    classe pour laquelle on doit calculer le barycentre
    @return                     résultat barycentre x,y

    dans cette fonction, on doit calculer le barycentre d'une classe
    c'est-à-dire le barycentre des points points[i]
    pour lesquelles classes[i] == numero_class
    """
    mx,my = 0.0,0.0
    nb    = 0
    for i in range (0, len (points)) :
        p = points [i]
        c = classes [i]
        if c != numero_class : continue
        nb += 1
        mx += p [0]
        my += p [1]
    return mx/nb, my/nb

def tous_barycentres (points, classes) :
    """calcule les barycentres pour toutes les classes
    @param      points      points, nuage, matrice de deux colonnes
    @param      classes     liste d'entiers
    @return                 liste de barycentre = matrice de deux colonnes
    """
    mx          = max (classes)+1
    barycentre  = []
    for m in range (0,mx) :
        b = barycentre_classe (points, classes, m)
        barycentre.append (b)
    return barycentre

def numpy_tous_barycentres (points, classes) :
    """écriture de barycentre_classe et tous_barycentres
    en une seule fonction avec numpy
    """
    nbcl = max (classes)+1
    mat  = numpy.matrix (points)
    vec  = numpy.array ( classes )
    clas = numpy.zeros ( (len (points), nbcl) )
    for i in range (0, nbcl) :
        clas [ vec == i, i ] = 1.0
    nb   = clas.sum (axis = 0)
    for i in range (0, nbcl) :
        clas [ vec == i, i ] = 1.0 / nb [i]
    ba = mat.transpose () * clas
    ba = ba.transpose ()
    ba = ba.tolist ()
    barycentre = [ b for b in ba ]
    return barycentre

def numpy_tous_barycentres2 (points, classes) :
    """écriture de barycentre_classe et tous_barycentres
    en une seule fonction avec numpy
    """
    nbcl        = max (classes)+1
    mat         = numpy.matrix (points)
    matt        = mat.transpose ()
    matcl       = numpy.matrix (classes).transpose ()
    barycentre  = []
    for c in xrange (0, nbcl) :
        w  = numpy.matrix (matcl)
        w [matcl==c] = 1
        w [matcl!=c] = 0
        wt = w.transpose ()
```

```
        r  = matt * w
        n  = wt * w
        r /= n [0,0]
        barycentre += [ [ r [0,0], r [1,0] ] ]

    return barycentre

def nuees_dynamiques (points, nbcl) :
    """algorithme des nuées dynamiques
    @param      points          ensemble points = matrice de deux colonnes
    @param      nbcl            nombre de classes demandées
    @return                     un tableau incluant la liste d'entiers
    """
    classes = random_class (points, nbcl)

    # on a le choix entre la version sans numpy
    for i in range (0,10) :
        print "iteration",i, max (classes)+1
        barycentres = tous_barycentres (points, classes)        # ou l'un
        classes     = association_barycentre (points, barycentres)
    cl1 = classes

    # ou la première version avec numpy
    for i in range (0,10) :
        print "iteration",i, max (classes)+1
        barycentres = numpy_tous_barycentres (points, classes)  # ou l'autre
        classes     = association_barycentre (points, barycentres)
    cl2 = classes

    # ou la seconde version avec numpy
    for i in range (0,10) :
        print "iteration",i, max (classes)+1
        barycentres = numpy_tous_barycentres2 (points, classes)  # ou l'autre
        classes     = association_barycentre (points, barycentres)
    cl3 = classes

    # on doit trouver cl1 == cl2 == cl3
    if cl1 != cl2 or cl1 != cl3 :
        print "erreur de calculs dans l'une des trois fonctions"
    return classes

# début du programme : on construit un nuage de points
nuage = n_sous_nuages (3, 50)
# on appelle l'algorithme
classes = nuees_dynamiques (nuage, 3)
# on dessine le résultat
dessin_classes (nuage, classes)
```

fin exo 7 □

2.7 Année 2010, rattrapage

On souhaite au cours de cette séance traverser quelques villes de France le plus rapidement possible. Il est plutôt évident que le chemin illustré par le graphique 2.1 n'est pas le plus rapide. On cherchera à implémenter quelques astuces qui permettront de construire un chemin "acceptable".

1) La première étape consiste à représenter la liste des villes et de leurs coordonnées via des listes *Python* :

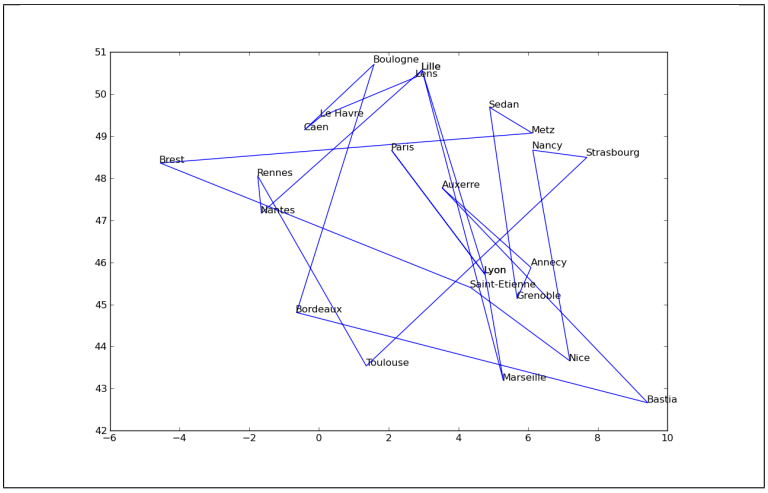


Figure 2.1 : *Le tour de France. On veut trouver le plus court chemin passant par toutes les villes. Ce problème est aussi connu sous le nom du problème du voyageur de commerce.*

Auxerre	3,537	47,767
Bastia	9,434	42,662
Bordeaux	-0,643	44,808
Boulogne	1,580	50,709
...	...	...
Grenoble	5,684	45,139
Annecy	6,082	45,878

Elles sont accessibles depuis l'adresse http://www.xavierdupre.fr/enseignement/examen_python/villes.txt. Il s'agit de créer une fonction qui récupère ces informations, soit depuis un fichier texte, soit elles peuvent être directement insérées dans le programme sous la forme d'une seule chaîne de caractères.

L'objectif est ensuite d'obtenir une matrice avec le nom de chaque ville en première colonne, l'abscisse et l'ordonnée en seconde et troisième colonnes. Les fonctions `strip`, `replace`, `split` pourraient vous être utiles.

```
[['Auxerre', 3.537309885, 47.767200469999999],
 ['Bastia', 9.4343004229999998, 42.661758419999998],
 ...
```

L'abscisse et l'ordonnée doivent être des réels (`float`) afin d'être facilement manipulées par la suite. (3 points) *Insérer directement dans le programme la matrice dans sa forme finale ne rapporte pas de point.*

```
def get_tour () :
    stour = """Auxerre        3,537309885        47,76720047
Bastia        9,434300423        42,66175842
Bordeaux        -0,643329978        44,80820084"""
    ...
    return tour
```

2) Ecrire une fonction `distance` qui calcule la distance euclidienne entre deux villes. On supposera que la distance à vol d'oiseau est une approximation acceptable de la distance entre deux villes. (2 points)

```
def distance (tour, i,j) :
    ...
    return d
```

3) Ecrire une fonction `longueur_tour` qui retourne la longueur d'un circuit. Un circuit est décrit par la matrice de la première question : on parcourt les villes les unes à la suite des autres dans l'ordre où elles apparaissent dans la matrice. On commence à Auxerre, on va à Bastia puis Bordeaux pour terminer à Annecy et revenir à Auxerre. (2 points)

```
def longueur_tour (tour) :
    ...
    return d
```

4) Il est facile de vérifier visuellement si un chemin est absurde comme celui de la figure 2.1. La fonction suivante vous aidera à tracer ce chemin. Il faut la compléter. (2 points)

```
import pylab
def graph (tour) :
    x = [ t[1] for t in tour ]
    y = [ t[2] for t in tour ]
    ....
    ....
    pylab.plot (x,y)
    for ville,x,y in tour :
        pylab.text (x,y,ville)
    pylab.show ()
```

5) La première idée pour construire un chemin plus court est de partir du chemin initial. On échange deux villes choisies aléatoirement puis on calcule la distance du nouveau chemin. Si elle est plus courte, on conserve la modification. Si elle est plus longue, on annule cette modification. On continue tant qu'il n'est plus possible d'améliorer la distance. (4 points)

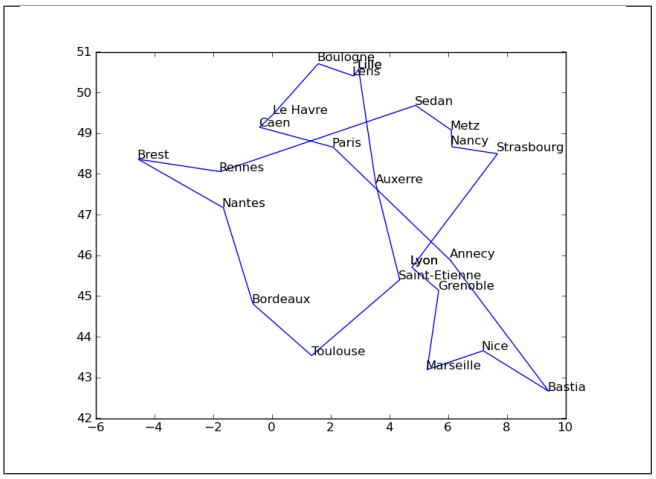


Figure 2.2 : *Le tour de France lorsque des chemins se croisent. Ce chemin n'est pas optimal de manière évidente.*

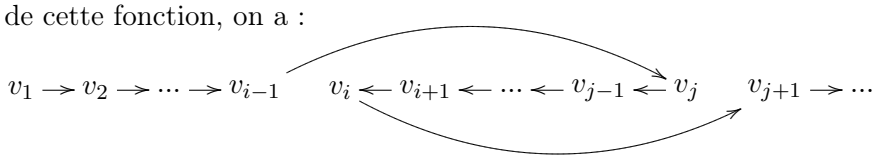
```
def permutation (tour) :
```

6) Le résultat n'est pas parfait. Parfois, les chemins se croisent comme sur la figure 2.2. Pour cela on va essayer de retourner une partie du chemin. Il s'agit ici de construire une fonction `retourne` qui retourne un chemin entre deux villes i et j. (3 points)

Avant l'exécution de cette fonction, on a :

$$v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_{i-1} \rightarrow v_i \rightarrow v_{i+1} \rightarrow \cdots \rightarrow v_{j-1} \rightarrow v_j \rightarrow v_{j+1} \rightarrow \cdots$$

Après l'exécution de cette fonction, on a :



Ou encore :

$$v_1 \to v_2 \to \cdots \to v_{i-1} \to v_j \to v_{j-1} \to \cdots \to v_{i+1} \to v_i \to v_{j+1} \to \cdots$$

```
def retourne (tour, i,j) :
```

7) De la même manière qu'à la question 5, on choisit deux villes au hasard. On applique la fonction précédente entre ces deux villes. Si la distance est plus courte, on garde ce changement, sinon, on revient à la configuration précédente. On répète cette opération tant que la distance du chemin total diminue. (2 points)

```
def croisement (tour) :
```

8) On termine par l'exécution des fonctions `permutation`, `croisement`, `graph`. On vérifie que le chemin obtenu est vraisemblable même s'il n'est pas optimal.

```
def resoud_et_dessine (tour) :
```

Les deux transformations proposées pour modifier le chemin sont décrites par les fonctions `permutation` et `croisement`. A aucun moment, on ne s'est soucié du fait que le chemin est circulaire. Est-ce nécessaire ? Justifier. (2 points)

Correction

La dernière question suggère que l'aspect circulaire du circuit n'a pas été pris en compte par les fonctions `croisement` et `permutation`. Pour échanger deux villes, il n'est nul besoin de tenir compte du fait que la dernière ville est reliée à la première. En ce qui concerne la fonction `croisement`, il est vrai qu'on ne considère aucune portion incluant le segment reliant la première et la dernière ville. Toutefois, lorsqu'on retourne toutes les villes dans l'intervalle $[i]\,j$, on aboutit au même résultat que si on retournait toutes les villes qui n'y sont pas.

```
# coding: latin-1
import random, numpy, math, pylab, copy

###
### réponse à la question 1
###

def get_tour () :
    tour = """Auxerre        3,537309885        47,76720047
Bastia        9,434300423        42,66175842
Bordeaux        -0,643329978        44,80820084
Boulogne        1,579570055        50,70875168
Caen        -0,418989986        49,14748001
Le Havre        0,037500001        49,45898819
Lens        2,786649942        50,40549088
Lille        2,957109928        50,57350159
Lyon        4,768929958        45,70447922
Paris        2,086790085        48,65829086
```

```
Lyon        4,768929958        45,70447922
Marseille        5,290060043        43,1927681
Lille        2,957109928        50,57350159
Nantes        -1,650889993        47,16867065
Rennes        -1,759150028        48,05683136
Toulouse        1,356109977        43,5388298
Strasbourg        7,687339783        48,49562836
Nancy        6,134119987        48,66695023
Nice        7,19904995        43,6578598
Saint-Etienne        4,355700016        45,39992905
Brest        -4,552110195        48,36014938
Metz        6,11729002        49,0734787
Sedan        4,896070004        49,68407059
Grenoble        5,684440136        45,13940048
Annecy        6,082499981        45,8782196""".replace (",", ".").split ("\n")
    # ligne d'avant : on découpe l'unique chaîne de caractères

    # ligne suivant : on découpe chaque ligne en colonne
    tour = [ t.strip ("\r\n ").split ("\t") for t in tour ]
    # puis on convertit les deux dernières colonnes
    tour = [ t [:1] + [ float (x) for x in t [1:] ] for t in tour ]
    return tour

###
### réponse à la question 2
###

def distance (tour, i,j) :
    dx = tour [i][1] - tour [j][1]
    dy = tour [i][2] - tour [j][2]
    return (dx**2 + dy**2) ** 0.5

###
### réponse à la question 3
###

def longueur_tour (tour) :
    # n villes = n segments
    d = 0
    for i in xrange (0,len(tour)-1) :
        d += distance (tour, i,i+1)
    # il ne faut pas oublier de boucler pour le dernier segment
    d += distance (tour, 0,-1)
    return d

###
### réponse à la question 4
###

def graph (tour) :
    x = [ t[1] for t in tour ]
    y = [ t[2] for t in tour ]
    x += [ x [0] ]   # on ajoute la dernière ville pour boucler
    y += [ y [0] ]   #
    pylab.plot (x,y)
    for ville,x,y in tour :
        pylab.text (x,y,ville)
    pylab.show ()

###
### réponse à la question 5
###

def permutation (tour) :
```

```
    # on calcule la longueur du tour actuelle
    best  = longueur_tour (tour)

    # variable fix : dit combien d'échanges ont eu lieu depuis la
    # dernière amélioration
    fix   = 0
    while True :
        # on tire deux villes au hasard
        i = random.randint (0, len(tour)-1)
        j = random.randint (0, len(tour)-1)
        if i == j : continue

        # on les échanges si i != j
        e = tour [i]
        tour [i] = tour [j]
        tour [j] = e

        # on calcule la nouvelle longueur
        d = longueur_tour (tour)

        if d >= best :
            # si le résultat est plus long --> retour en arrière
            # ce qui consiste à échanger à nouveau les deux villes
            fix += 1
            e = tour [i]
            tour [i] = tour [j]
            tour [j] = e
        else :
            # sinon, on garde le tableau tel quel
            best = d
            # et on met fix à 0 pour signifier qu'une modification a eu lieu
            fix = 0

        # si aucune modification n'a eu lieu durant les dernières 10000 itérations,
        # on s'arrête
        if fix > 10000 : break

###
### réponse à la question 6
###

def retourne (tour, i,j) :
    """
    on échange les éléments i et j
    puis i+1 et j-1
    puis i+2 et j-2
    tant que i+k < j-k
    """
    while i <= j :
        e = tour [i]
        tour [i] = tour [j]
        tour [j] = e
        i += 1
        j -= 1

###
### réponse à la question 7
###

def croisement (tour) :
    """
    cette fonction reprend le même schéma que la fonction permutation
    on annule une modification en appelant à nouveau la fonction retourne
    """
    best  = longueur_tour (tour)
```

```
    fix   = 0
    while True :
        i = random.randint (0, len(tour)-2)
        j = random.randint (i+1, len(tour)-1)
        retourne (tour, i,j)
        d = longueur_tour (tour)
        if d >= best :
            # retour en arrière
            fix += 1
            retourne (tour, i,j)
        else :
            fix = 0
            best = d
        if fix > 10000 : break

###
### réponse à la question 8
###

def enchaine (tour) :
    """
    cette fonction est plus complexe que le résultat demandé pour cette question
    on enchaîne les deux fonctions (croisement, permutation) tant que
    la longueur du circuit diminue

    et si jamais cette longueur ne diminue plus, on perturbe le circuit
    au plus deux fois
    en échangeant trois couples de villes choisies au hasard,
    cette dernière partie n'était pas prévue dans l'énoncé
    """
    best = longueur_tour (tour)
    tttt = copy.deepcopy (tour)
    print "debut", best
    nom  = 0
    while True :

        croisement (tour)
        d = longueur_tour (tour)
        print "croisement", d, best

        permutation (tour)
        d = longueur_tour (tour)
        print "permutation", d, best

        if d < best :
            best = d
            tttt = copy.deepcopy (tour)
            nom  = 0
        elif nom > 2 :
            break
        else :
            nom += 1
            for k in range (0,3) :
                i = random.randint (0, len(tour)-2)
                j = random.randint (i+1, len(tour)-1)
                e = tour [i]
                tour [i] = tour [j]
                tour [j] = e

    return tttt

if __name__ == "__main__" :
    tour = get_tour ()
    tour = enchaine (tour)
    graph (tour)
```

fin exo 8 □

2.8 Année 2011

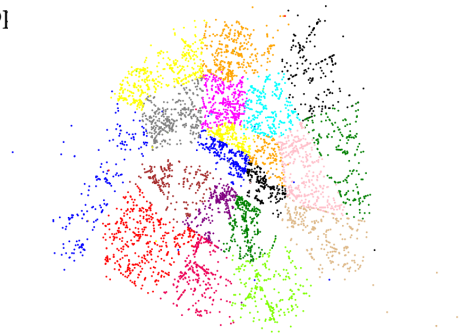
Lors de cette séance, on s'intéresse aux restaurants de Paris et à leur répartition selon les arrondissements. Ils sont décrits dans le fichier accessible à l'adresse : http://www.xavierdupre.fr/enseignement/examen_python/restaurant_paris.txt. Ce fichier contient trois colonnes : l'arrondissement, la longitude, la latitude. On supposera que Paris est suffisamment petit pour que la longitude et la latitude soient considérées comme des coordonnées cartésiennes.

1) La première question consiste à télécharger le fichier puis à récupérer ses informations via la fonction `def lit_fichier`. Cette fonction retourne une matrice (une liste de listes). (2 points)

2) On souhaite ensuite compter le nombre de restaurants par arrondissement pour déterminer celui qui en contient le plus et celui qui en contient le moins. On écrit une seule fonction `def compte_restaurant(mat)` qui retourne un dictionnaire contenant le décompte des restaurants par arrondissement. (3 points)

3) Certains arrondissements sont plus grands que d'autres et on aimerait mesurer la densité des restaurants par arrondissement plutôt que leur nombre. Cependant il n'est pas évident de deviner la superficie d'un arrondissement à partir des seuls restaurants. On construit donc l'apl

1. Chaque arrondissement est un cercle.
2. Le centre de ce cercle est le barycentre de tous les restaurants qu'il contient.
3. Le rayon est la distance séparant le barycentre du premier restaurant en dehors de l'arrondissement.
4. La superficie de l'arrondissement est alors πR^2.



On cherche d'abord à calculer le barycentre de chaque arrondissement. On constuit la fonction `def barycentre(mat)` qui retourne un dictionnaire associant un couple de points à chaque arrondissement. (4 points) **Cette question vaut 5 points si le coût de l'algorithme est optimal.**

4) On cherche maintenant à connaître le plus proche restaurant en dehors de l'arrondissement. Ecrire une fonction qui calcule la distance euclidienne entre deux points : `def distance(x1,y1,x2,y2)`. (2 points)

5) Pour un arrondissement, écrire une fonction `def plus_proche_restaurant(x,y,arr,mat)` qui retourne le restaurant le plus proche du point `(x,y)` en dehors de l'arrondissement `arr`. (4 points)

6) Ecrire une dernière fonction qui retourne sous forme de dictionnaire la densité approchée pour chaque arrondissement : `def densite_approchee(mat)`. Quelles sont les arrondissements le plus et le moins dense ? Commentez les résultats et proposez des améliorations ? (4 points)

```
75002    2.407478    48.930141        75010    2.405963    48.921033
75002    2.400573    48.913487        75018    2.391144    48.934509
```

Correction

```
# coding: latin-1
import urllib2, math

# question 1
def lit_fichier () :
    # le principe est le même que pour le chargement d'un fichier
    # le programme lit directement les informations depuis Internet
    f = urllib2.urlopen ("http://www.xavierdupre.fr/enseignement"\
                         "/examen_python/restaurant_paris.txt")
```

```
    s = f.read ()
    f.close ()
    lines = s.split ("\n")  # on découpe en lignes
    # on découpe en colonnes
    lines = [ _.strip ("\n\r ").split ("\t") for _ in lines if len (_) > 0 ]
    lines = [ _ for _ in lines if len (_) == 3 ]  # on supprime les lignes vides
    # on convertit les coordonnées en réel
    lines = [ (a [3:], float (b), float (c)) for a,b,c in lines ]
    return lines

# question 2
def compte_restaurant (mat) :
    # simple comptage, voir le chapitre 3...
    compte = { }
    for cp,x,y in mat :
        if cp not in compte : compte [cp] = 0
        compte [cp] += 1
    return compte

# question 3
def barycentre (mat) :
    # un barycentre est un point (X,Y)
    # où X et Y sont respectivement la moyenne des X et des Y
    barycentre = { }
    # boucle sur la matrice
    for cp,x,y in mat :
        if cp not in barycentre : barycentre [cp] = [ 0, 0.0, 0.0 ]
        a,b,c           = barycentre [cp]
        barycentre [cp] = [a+1, b+x, c+y]
    # boucle sur les barycentres
    for cp in barycentre :
        a,b,c = barycentre [cp]
        barycentre [cp] = [b/a, c/a]

    # le coût de cette fonction est en O (n log k)
    # où k est le nombre de barycentre
    # de nombreux élèves ont deux boucles imbriquées,
    # d'abord sur la matrice, ensuite sur les barycentres
    # ce qui donne un coût en O (nk), beaucoup plus grand
    return barycentre

# question 4
def distance (x1, y1, x2, y2) :
    return ((x1-x2)**2 + (y1-y2)**2)**0.5

# question 5
def plus_proche_restaurant (x,y, arr, mat) :
    m,mx,my = None, None, None
    for cp,a,b in mat :
        if cp != arr and (m == None or distance (a,b,x,y) < m) :
            mx,my = a,b
            m     = distance (a,b,x,y)
    return mx,my

# question 6
def densite_approchee (mat) :
    g      = barycentre (mat)
    compte = compte_restaurant (mat)
    res    = { }

    for cp in g :
        out  = plus_proche_restaurant (g [cp][0], g [cp][1], cp, mat)
        r    = distance (g [cp][0], g [cp][1], out [0], out [1])
        aire = math.pi * r ** 2
        res [cp] = compte [cp] / aire
```

```
    return res

if __name__ == "__main__" :

    if False :  #mettre à vrai pour remplacer la fonction plus_proche_restaurant
        # ajout par rapport à l'énoncé
        # en réponse à la dernière question
        # plutôt que de prendre le premier point à hors de l'arrondissement
        # on considère celui correspondant à un quantile (5%)
        # ce qui évite les quelques restaurants dont les données
        #sont erronées
        def plus_proche_restaurant_avec_amelioration (x,y, arr, mat) :
            all     = []
            for cp,a,b in mat :
                if cp != arr :
                    m = distance (a,b,x,y)
                    all.append ( (m,a,b))
            all.sort ()
            a,b = all [len(all)/20][1:]
            return a,b

        # ajout par rapport à l'énoncé
        plus_proche_restaurant = plus_proche_restaurant_avec_amelioration

    mat = lit_fichier ()
    com = densite_approchee (mat)
    ret = [ (v,k) for k,v in com.iteritems () ]
    ret.sort ()
    for a,b in ret : print "%d\t%s" % (a,b)

    # ajout par rapport à l'énoncé
    # permet de dessiner les restaurants, une couleur par arrondissement
    # on observe que certains points sont aberrants, ce qui réduit d'autant
    # l'estimation du rayon d'un arrondissement (il suffit qu'un restaurant
    # étiquetés dans le 15ème soit situé près du barycentre du 14ème.)
    import matplotlib
    import numpy as np
    import matplotlib.pyplot as plt
    import matplotlib.mlab as mlab
    import matplotlib.cbook as cboo

    fig = plt.figure()
    ax = fig.add_subplot(111)
    colors = [  'red', 'blue', 'yellow', 'orange', 'black', 'green',
                'purple', 'brown', 'gray', 'magenta', 'cyan', 'pink', 'burlywood',
                'chartreuse', '#ee0055']
    for cp in barycentre (mat) :
        lx = [ m[1] for m in mat if m [0] == cp ]
        ly = [ m[2] for m in mat if m [0] == cp ]
        c  = colors [ int(cp) % len (colors) ]
        #if cp not in ["02", "20"] : continue
        ax.scatter(lx,ly, s = 5., c=c,edgecolors = 'none'  )
    plt.show ()
```

fin exo 9 □

2.9 Année 2012

2.9.1

Enoncé

Cet exercice doit être réalisé avec le module `numpy`.

Cet exercice a pour but de revenir sur le calcul matriciel proposé par le module `numpy`. Il s'appuie sur une enquête de l'INSEE réalisée en 2003 sur un échantillon du 8000 personnes [9] dont on a extrait une sous-partie que récupère le programme suivant [10]. Ce programme fournit également quelques exemples de manipulation de la classe `array` du module `numpy`.

```
#coding:latin-1
import urllib, os, os.path, numpy
def charge_donnees (nom = "donnees_enquete_2003_television.txt") :
    if os.path.exists (nom) :
        # si le fichier existe (il a déjà été téléchargé une fois)
        f = open (nom, "r")
        text = f.read ()
        f.close ()
    else :
        # si le fichier n'existe pas
        link = "http://www.xavierdupre.fr/enseignement/td_python/" + \
                "python_td_minute/data/examen/" + nom
        url = urllib.urlopen (link)
        text = url.read ()
        # on enregistre les données pour éviter de les télécharger une seconde fois
        f = open (nom, "w")
        f.write (text)
        f.close ()

    lines = text.split ("\n")
    lines = [ l.split("\t") for l in lines if len(l) > 3 ]
    lines = [ [ "0" if s.strip() == "" else s for s in l ] for l in lines ]
    return lines

donnees = charge_donnees ()

colonne = donnees [0]
matrice = numpy.array (donnees [1:], dtype=float)

# quelques exemples d'utilisation du module numpy
petite_matrice = matrice [0:5,2:4]
print petite_matrice

# dimension d'une matrice
print petite_matrice.shape

# multiplication terme à terme
vecteur = petite_matrice [:,0] * petite_matrice [:,1]
print vecteur

# sum
print vecteur.sum ()

# changer une valeur selon une condition
petite_matrice [ petite_matrice [:,1] == 1, 1] = 5
print petite_matrice
```

9. disponible à l'adresse http://www.insee.fr/fr/themes/detail.asp?ref_id=fd-hdv03\&page=fichiers_detail/HDV03/telechargement.htm

10. téléchargeable à l'adresse http://www.xavierdupre.fr/enseignement/examen_python/python_examen_2011_2012.py

```
# ne conserver que certaines lignes de la matrice
m = petite_matrice [ petite_matrice[:,1] == 5,: ]
print m

# créer une matrice 10x2 avec des zéros
m = numpy.zeros( (10,2) )

# trier les lignes selon la première colonne
tr = numpy.sort (petite_matrice, 0)


# dessiner deux courbes
def dessin_temperature (temperature) :
    import pylab
    u = [ t[3] for t in temperature ]
    v = [ t[4] for t in temperature ]
    pylab.plot (u)
    pylab.plot (v)
    pylab.show()
```

Le fichier téléchargé [11] est stocké dans la variable `matrice`, il contient quatre colonnes :

colonne	nom	description
0	POIDSLOG	Pondération individuelle relative
1	POIDSF	Variable de pondération individuelle
2	cLT1FREQ	Nombre d'heures en moyenne passées à regarder la télévision
3	cLT2FREQ	Unité de temps utilisée pour compter le nombre d'heures passées à regarder la télévision, cette unité est représentée par les quatre valeurs suivantes : 0 non concerné 1 jour 2 semaine 3 mois

Le fichier contient des lignes comme celles qui suivent. Sur la première ligne contenant des chiffres, un individu a déclaré passer deux heures par jour devant la télévision. Sur la suivante, un autre individu a déclaré passer six heures par semaine.

```
POIDLOG       POIDSF         cLT1FREQ  cLT2FREQ
0.8894218317  4766.8652013   2         1
2.740069772   14685.431344   6         2
```

Après la première exécution, le fichier est présent sur votre disque dur à côté du programme, il est alors facile de l'ouvrir de voir ce qu'il contient.

1) Toutes les lignes ne sont pas renseignées. Modifier la matrice pour enlever les lignes pour lesquelles l'unité de temps (cLT2FREQ) n'est pas renseignée ou égale à zéro. (1 point)

2) Remplacer les valeurs de la quatrième colonne par une durée en heures. (1 point)

3) Calculer le nombre d'heures par jour moyen passées devant la télévision et écrire la réponse en commentaire de la fonction. (1 point)

4) Calculer ce même nombre mais pondéré par la colonne POIDSF et écrire la réponse en commentaire de

11. http://www.xavierdupre.fr/enseignement/td_python/python_td_minute/data/examen/donnees_enquete_2003_television.txt

la fonction. (2 points)

5) Combien de gens dans l'échantillon regardent-ils la télévision plus de 24h par jour ? Quelle est la raison la plus probable ? Ecrire la réponse en commentaire de la fonction. (1 point)

6) Calculer la médiane non pondérée. Ecrire la réponse en commentaire de la fonction. (2 points)

2.9.2

Enoncé

Cet exercice peut-être réalisé indifféremment avec le module `numpy` ou non.

Mes grands-parents passent chaque année l'été à Charleville-Mézières (dans le Nord-Est de la France) et l'hiver à Cannes (dans le Sud). Les températures mesurées quotidiennement sur l'année 2010 sont représentées par la figure 2.3 [12]. Mes grands-parents cherchent à se rapprocher d'une température idéale de 20 degrés. Ils voudraient savoir quand ils auraient dû partir à Cannes et quand en revenir pour atteindre le plus possible cet objectif en 2010.

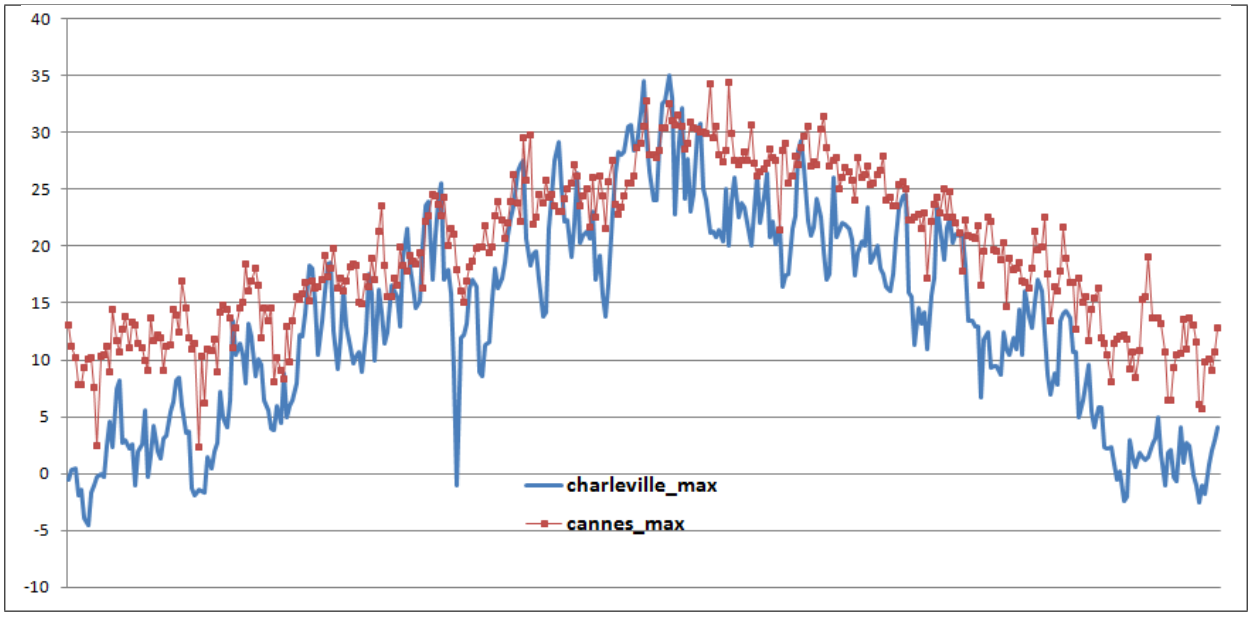


Figure 2.3 : *Températures maximales mesurées quotidiennement à Cannes et Charleville-Mézières en 2010.*

L'exercice précédent utilise une fonction `charge_donnees` qu'on utilise également pour récupérer les données de températures [13].

```
temperature = charge_donnees("cannes_charleville_2010_max_t.txt")
```

Après la première exécution, le fichier est présent sur votre disque dur à côté du programme, il est alors facile de l'ouvrir de voir ce qu'il contient. L'ensemble peut être aussi copié/collé sous Excel.

1) Ecrire une fonction qui récupère le tableau matrice et convertit tous les éléments en réels (faire attention à la première ligne). (1 point)

2) Certaines valeurs sont manquantes et sont remplacées par la valeur -1000 choisie par convention (voir la date du 4 mai). Ecrire une fonction qui remplace ces valeurs par la moyenne des deux valeurs adjacentes. (3 points)

On pourra s'aider de la fonction suivante pour vérifier que les manquantes ne sont plus présentes :

```
def dessin_temperature (temperature) :
    import pylab
    u = [ t[3] for t in temperature ]
```

12. Ces données sont extraites du site http://www.meteociel.fr/climatologie/villes.php.
13. http://www.xavierdupre.fr/enseignement/td_python/python_td_minute/data/examen/cannes_charleville_2010_max_t.txt

```
    v = [ t[4] for t in temperature ]
    pylab.plot (u)
    pylab.plot (v)
    pylab.show()
```

3) Après avoir discuté avec mes grands-parents, nous avons décidé d'utiliser une fonction de coût égale au carré de l'écart entre deux températures : la température qu'ils souhaitent (20 degré) et la température qu'ils ont (celle de Charleville ou Cannes). Ecrire une fonction `distance` qui calcule la fonction suivante : (1 point)

$$d(T_1, T_2) = (T_1 - T_2)^2 \tag{2.1}$$

4) Supposons que nous connaissons les deux dates d'aller t_1 et de retour t_2, mes grands-parents seraient donc à :

hiver	t_1	été	t_2	hiver
Cannes		Charleville		Cannes

Ecrire une fonction qui calcule la somme des écarts de température au carré entre la température T souhaitée et la température où ils sont : (3 points)

$$E = \sum_{t=1}^{t=t_1} d\left(T(Cannes, t), T\right) + \sum_{t=t_1}^{t=t_2} d\left(T(Charleville, t), T\right) + \sum_{t=t_2}^{t=365} d\left(T(Cannes, t), T\right) \tag{2.2}$$

5) Ecrire une fonction qui détermine les meilleures valeurs t_1 et t_2. Ecrire en commentaire de votre programme le raisonnement suivi et la réponse trouvée. (3 points)

6) Quel est le coût de votre algorithme en fonction du nombre de jours ? Si, on remplace la série de températures quotidienne par une série hebdomadaire, l'algorithme est combien de fois plus rapide ? (1 point)

Correction

```
# coding: latin-1
# ce fichier contient le programme fournit au début de l'examen
# http://www.xavierdupre.fr/enseignement/examen_python/python_examen_2011_2012.py
from td_note_2012_enonce import *
import numpy, pylab

########################################################################
# exercice 1
########################################################################

# question 1 (+1=1p)
donnees = charge_donnees ()
colonne = donnees [0]
matrice = numpy.array (donnees [1:], dtype=float)

mat     = matrice
mat     = mat [ mat [:,3] > 0, : ]

# question 2 (+1=2p)
mat [ mat[:,3] == 1, 3 ] = 24
mat [ mat[:,3] == 2, 3 ] = 24*7
mat [ mat[:,3] == 3, 3 ] = 24*30

# question 3 (+1=3p)
res = mat [:,2] / mat [:,3]
print res.sum() / res.shape [0]       # 0.111 ~ 11,1% du temps passé devant la télévision
print res.sum() / res.shape [0] * 24  # soit 2h40min
```

```
# question 4 (+2=5p)
m = mat[:,1] * mat[:,2] / mat[:,3]
print m.sum() / mat[:,1].sum()        # 0.108 ~ 10,8%

# question 5 (+1=6p)
m = mat[ mat[:,2] > mat[:,3], : ]
print m  # il y a deux personnes et la raison la plus probable est une erreur dans l'unité de temps

# question 6 (+2=8p)
res = numpy.sort (res, 0)
print res[res.shape[0]/2]   # 0.083 ~ 8.3% = 2h

# question 7 (+2=10p)
pr = numpy.zeros ((mat.shape[0],4))
pr [:,0] = mat[:,2] / mat[:,3]
pr [:,1] = mat[:,1]
pr [:,2] = pr[:,0] * pr[:,1]
pr = numpy.sort (pr, 0)
total = pr[:,2].sum()
pr[0,3] = pr [0,2]
for i in xrange (1, pr.shape[0]) :
    pr[i,3] = pr[i-1,3] + pr[i,2]
    if pr[i,3]/total > 0.5 :
        fin = i
        break
print pr[fin,3] / pr[:fin+1,1].sum()  # 0.0895 ~ 8.95%

#############################################################################
# exercice 2
#############################################################################

# question 1 (+1=1p)
temperature = charge_donnees("cannes_charleville_2010_max_t.txt")

def conversion_reel (temperature) :
    return [ [ float (x) for x in l ] for l in temperature [1:] ]

temperature = conversion_reel(temperature)

#question 2  (+2=3p)
def valeur_manquante (temperature, c) :
    for i in xrange (1, len (temperature)-1) :
        if temperature [i][c] == -1000 :
            temperature [i][c] = (temperature [i-1][c] + temperature [i+1][c]) / 2

valeur_manquante(temperature, 3)
valeur_manquante(temperature, 4)

def dessin_temperature (temperature) :
    import pylab
    u = [ t[3] for t in temperature ]
    v = [ t[4] for t in temperature ]
    pylab.plot (u)
    pylab.plot (v)
    pylab.show()

# on met en commentaire pour éviter de l'exécuter à chaque fois
# dessin_temperature(temperature)

# question 3 (+1=4p)

def distance (u,t) :
    return (u-t)**2

# question 4 (+3=7p)
```

```
def somme_ecart (temperature, t1, t2, T) :
    s = 0
    for i in xrange (0, len(temperature)) :
        if t1 < i < t2 :
            s += distance (temperature[i][3], T) # charleville
        else :
            s += distance (temperature[i][4], T) # cannes
    return s

# question 5 (+3=10p)

def minimisation (temperature, T) :
    best = 1e10
    t1t2 = None
    for t1 in xrange (0,len(temperature)) :
        for t2 in xrange (t1+1,len(temperature)) :
            d = somme_ecart(temperature,t1,t2,T)
            if best == None or d < best :
                best = d
                t1t2 = t1,t2
    return t1t2, best

#for i in range (300,363) : print "*",somme_ecart (temperature, i, i+2, 20)
print temperature [191]
print temperature [266]
for T in range (15, 25) :
    print T, "**",minimisation (temperature,T) # (191 = 11/7, 266 = 24/9)  (attendre 2 minutes)

# question 6
# Le coût de l'algorithme est on O(n^2) car l'optimisation est une double boucle sur les températures.
# Passer des jours aux semaines, c'est utiliser des séries 7 fois plus courtes,
# l'optimisation sera 7^2 fois plus rapide.
```

fin exo 2.9.2 □

2.10 Année 2013, examen court

ENSAE mardi 23 octobre 2012

Cette interrogation écrite compte pour 5 points ajoutés à l'ensemble des notes de la matière. Tous les documents sont autorisés. La durée est d'une demi-heure. Vous devrez imprimer le résultat en n'omettant pas d'y ajouter votre nom et le numéro de l'exercice qui vous aura été distribué (1, 2 ou 3).

2.10.1

Enoncé

1) Ecrire une fonction qui retourne la fréquence de chaque lettre d'un mot. Le résultat sera un dictionnaire dont les clés seront les lettres et les valeurs seront les fréquences. La fréquence désigne le nombre d'occurence d'une lettre. (2 points)

2) Ecrire une fonction qui vérifie que deux mots sont des anagrammes : deux mots sont anagrammes l'un de l'autre s'ils sont composés des mêmes lettres. Cette fonction devra inclure au moins une boucle. (3 points)

Correction

```
# coding: latin-1
# question 1
```

```
def frequence_lettre (mot) :
    res = { }
    for c in mot :
        if c in res : res[c] += 1
        else : res [c] = 1
    return res

print frequence_lettre ("aviateur")
# affiche {'a': 2, 'e': 1, 'i': 1, 'r': 1, 'u': 1, 't': 1, 'v': 1}

# Deux autres écritures de la fonction
def frequence_lettre (mot) :
    res = { c:0 for c in mot }
    for c in mot : res[c] += 1
    return res

def frequence_lettre (mot) :
    res = { }
    for c in mot :
        # la méthode get retourne res[c] si cette valeur existe, 0 sinon
        res[c] = res.get( c, 0 ) + 1
    return res

# question 2

def anagramme (mot1, mot2) :
    h1 = frequence_lettre(mot1)
    h2 = frequence_lettre(mot2)

    # il fallait éviter d'écrire la ligne suivante bien qu'elle retourne le résultat cherché :
    # return h1 == h2

    for c in h1 :
        if c not in h2 or h1[c] != h2[c] : return False
    # il ne faut pas oublier cette seconde partie
    for c in h2 :
        if c not in h1 : return False
    return True

a,b = "anagramme", "agrammane"
print anagramme (a,b), anagramme (b,a) # affiche True, True

# on vérifie que la fonctionne marche aussi dans l'autre cas
a,b = "anagramme", "agrummane"
print anagramme (a,b), anagramme (b,a) # affiche False, False

# on pouvait faire plus rapide en éliminant les cas évidents
def anagramme (mot1, mot2) :
    if len(mot1) != len(mot2) : return False
    h1 = frequence_lettre(mot1)
    h2 = frequence_lettre(mot2)
    if len(h1) != len(h2) : return False

    for c in h1 :
        if h1[c] != h2.get(c, 0) : return False
    return True
```

fin exo 2.10.1 □

2.10.2

Enoncé

1) Ecrire une fonction qui calcule la factorielle de n : $f(n) = n! = 2 * 3 * ... * n$. Le calcul ne doit pas être récursif. (2 points)

2) Ecrire une fonction qui $f(a,b) = f(a-1,b) + f(a,b-1)$ pour $a,b \geqslant 1$. On suppose que $f(0,n) = f(n,0) = n$. Le calcul ne doit pas être récursif. (3 points)

Correction

```
# coding: latin-1

# question 1

def factorielle (n) :
    res = 1
    while n > 1 :
        res *= n
        n -= 1
    return res

print factorielle (3)

# voici la version récursive qui n'était pas demandée :
def factorielle_recursive (n) :
    return n*factorielle_recursive (n-1) if n > 1 else 1

# question 2

def f(a,b) :

    # f(a,b) = f(a-1,b) + f(a,b-1)
    # la formule implique de calculer toutes les valeurs f(i,j)
    # avec (i,j) dans [[0,a]] x [[0,b]]
    # un moyen afin d'éviter de trop nombreux calculs est de
    # stocker les valeurs intermédiaires de f dans une matrice
    # il faut ensuite s'assurer que f(a-1,b) et f(a,b-1)
    # ont déjà été calculées avant de calculer f(a,b)
    # pour cela, on parcourt la matrice dans le sens des i et j croissants
    # il ne faut pas oublier les a+1,b+1 car range (a) est égal à [ 0,1, ..., a-1 ]

    mat = [ [ 0 for i in range (b+1) ] for j in range (a+1) ]
    for i in range (a+1) :
        for j in range (b+1) :
                if i == 0 or j == 0 :
                    mat [i][j] = max (i,j)
                else :
                    mat [i][j] = mat [i-1][j] + mat[i][j-1]
    return mat [a][b]

# on teste pour des valeurs simples
print f(0,5) # affiche 5
print f(1,1) # affiche 2
# on vérifie en suite que cela marche pour a < b et b > a
print f (4,5)  # affiche 210
print f (5,4)  # affiche 210

# autres variantes

# la version récursive ci-dessous est juste beaucoup plus longue, elle appelle
# la fonction f_recursive autant de fois que la valeur retournée
# par la fonction cout_recursive alors que la fonction précédente
# effectue de l'ordre de O(a*b) calculs
```

```
def f_recursive(a,b) :
    return f_recursive(a-1,b) + f_recursive(a,b-1) if a > 0 and b > 0 else max(a,b)

def cout_recursive(a,b) :
    return cout_recursive(a-1,b) + cout_recursive(a,b-1) if a > 0 and b > 0 else 1

# une autre version non récursive
def f(a,b) :
    mat = { }
    for i in range (a+1) :
        for j in range (b+1) :
                mat [i,j] = mat [i-1,j] + mat[i,j-1] if i > 0 and j > 0 else max (i,j)
    return mat [a,b]
```

La fonction `cout_recursive` retourne le nombre d'appels à la fonction `f_recursive` fait pour calculer $f(a, b)$. Pour avoir une idée plus précise de cette valeur, on cherche à déterminer la valeur de la fonction `cout_recursive` en fonction de (a, b) et non de façon récursive. On note la fonction g = `cout_recursive` :

$$g(a,b) = \begin{cases} g(a-1,b) + g(a,b-1) & \text{si } a > 0 \text{ et } b > 0 \\ 1 & \text{sinon} \end{cases} \tag{2.3}$$

Cette définition est identique à celle du triangle de Pascal sur les combinaisons. On peut donc en déduire que :

$$g(a,b) = C_{a+b}^{a} \tag{2.4}$$

La version récursive sous cette forme est simple mais très coûteuse. Même pour des petites valeurs, le calcul devient vite très lent. Cela ne veut pas dire qu'il ne faut pas l'envisager à condition de stocker les valeurs intermédiaires :

```
def f_recursive(a,b, memoire) :
    if (a,b) in memoire : return memoire [a,b]
    else :
        res = f_recursive(a-1,b, memoire) + f_recursive(a,b-1, memoire) \
                    if a > 0 and b > 0 else max(a,b)
        memoire [a,b] = res
        return res
```

fin exo 2.10.2 □

2.10.3

Enoncé

1) Ecrire une fonction qui retourne la somme des éléments de cette liste entre les positions `i` et `j` exclu sans utiliser la fonction `sum`. Testez la fonction sur une liste de votre choix contenant des nombres positifs et négatifs. (2 points)

2) Utiliser la fonction précédente pour trouver la sous-liste `li[i : j]` dont la somme des éléments est la plus grande. Elle inclura deux boucles. (3 points)

Correction

L'une des réponses était suffisante pour obtenir les trois points.

```
# coding: latin-1

# question 1

def somme_partielle (li, i, j) :
    r = 0
    for a in range (i,j) :
        r += li [a]
    return r

# question 2

def plus_grande_sous_liste (li) :
    meilleur = min(li)              # ligne A
    im,jm = -1,-1
    for i in range (0,len(li)) :
        for j in range (i+1, len(li)+1) :   # ne pas oublier +1 car sinon
                        # le dernier élément n'est jamais pris en compte
            s = somme_partielle(li, i,j)
            if s > meilleur :
                meilleur = s
                im,jm = i,j
    return li [im:jm]

# si li ne contient que des valeurs positives, la solution est évidemment la liste entière
# c'est pourquoi il faut tester cette fonction avec des valeurs négatives
li = [ 4,-6,7,-1,8,-50,3]
m  = plus_grande_sous_liste(li)
print m   # affiche [7, -1, 8]

li = [1,2,3,4,5,-98,78,9,7,7]
m  = plus_grande_sous_liste(li)
print m   # affiche [79, 9, 7, 7]

# autre version plus courte

def plus_grande_sous_liste (li) :
    solution = [ (somme_partielle(li,i,j), i, j) \
                    for i in range (0,len(li)) \
                    for j in range (i+1, len(li)+1) ]
    m = max(solution)
    return li [m[1]:m[2]]
```

La ligne A contient l'instruction `meilleur = min(li)`. Pour une liste où tous les nombres sont négatifs, la meilleure sous-liste est constitué du plus petit élément de la liste. Remplacer cette instruction par `meilleur = 0` a pour conséquence de retourner une liste vide dans ce cas précis. En ce qui concerne le coût de la fonction, les deux solutions proposées sont équivalentes car elles effectuent une double boucle sur la liste. Le coût de l'algorithme est en $O(n^3)$:

$$cout(n) = \sum_{i=0}^{n} \sum_{j=i+1}^{n} j - i \tag{2.5}$$

$$= \sum_{i=0}^{n} \sum_{j=0}^{i} j \tag{2.6}$$

$$= \sum_{i=0}^{n} \frac{i(i+1)}{2} \tag{2.7}$$

Il est possible de modifier cette fonction de telle sorte que le coût soit en $O(n^2)$ car on évite la répétition de certains calculs lors du calcul de la somme des sous-séquences.

```
def plus_grande_sous_liste_n2 (li) :
    meilleur = 0
    im,jm = -1,-1
    for i in range (0,len(li)) :
        s = 0
        for j in range (i, len(li)) :
            s += li[j]
            if s > meilleur :
                meilleur = s
                im,jm = i,j+1
    return li [im:jm]

li = [ 4,-6,7,-1,8,-50,3]
m  = plus_grande_sous_liste_n2(li)
print m   # affiche [7, -1, 8]

li = [1,2,3,4,5,-98,78,9,7,7]
m  = plus_grande_sous_liste_n2(li)
print m   # affiche [79, 9, 7, 7]
```

Enfin, il existe une dernière version plus rapide encore. Si on considère la liste $l = (l_1, ..., l_n)$ dont on cherche à extraire la sous-séquence de somme maximale. Supposons que l_a appartienne à cette sous-séquence. On construit la fonction suivante :

$$f(a,k) = \left\{ \begin{array}{ll} \sum_{i=a}^{k} l_i & \text{si } k > a \\ \sum_{i=k}^{a} l_i & \text{si } k < a \end{array} \right. \tag{2.8}$$

On cherche alors les valeurs k_1 et k_2 telles que :

$$f(a,k_1) = \max_{k<a} f(a,k) \tag{2.9}$$

$$f(a,k_2) = \max_{k>a} f(a,k) \tag{2.10}$$

La sous-séquence de somme maximale cherchée est $[k_1, k_2]$ avec a dans cet intervalle et le coût de cette recherche est en $O(n)$. Mais ceci n'est vrai que si on sait que l_a appartient à la sous-séquence de somme maximale.

Autre considération : pour deux listes l_1 et l_2, la séquence maximale de la réunion des deux appartient soit à l'une, soit à l'autre, soit elle inclut le point de jonction.

Ces deux idées mises bout à bout donne l'algorithme suivant construit de façon récursive. On coupe la liste en deux parties de longueur égale :
– On calcule la meilleure séquence sur la première sous-séquence.
– On calcule la meilleure séquence sur la seconde sous-séquence.
– On calcule la meilleure séquence en suppose que l'élément du milieu appartient à cette séquence.
La meilleure séquence est nécessairement l'une des trois.

```
#coding:latin-1
def plus_grande_sous_liste_rapide_r (li, i,j) :
    if i == j : return 0
    elif i+1 == j : return li[i],i,i+1

    milieu = (i+j)/2
```

```
    # on coupe le problème deux
    ma,ia,ja = plus_grande_sous_liste_rapide_r (li, i, milieu)
    mb,ib,jb = plus_grande_sous_liste_rapide_r (li, milieu, j)

    # pour aller encore plus vite dans un cas précis
    if ja == ib :
        total = ma+mb
        im,jm = ia,jb
    else :
        # on étudie la jonction
        im,jm = milieu,milieu+1
        meilleur = li[milieu]
        s = meilleur
        for k in range (milieu+1, j) :
            s += li[k]
            if s > meilleur :
                meilleur = s
                jm = k+1

        total = meilleur
        meilleur = li[milieu]
        s = meilleur
        for k in range (milieu-1, i-1, -1) :
            s += li[k]
            if s > meilleur :
                meilleur = s
                im = k

        total += meilleur - li[milieu]

    if   ma >= max(mb,total) : return ma,ia,ja
    elif mb >= max(ma,total) : return mb,ib,jb
    else : return total,im,jm

def plus_grande_sous_liste_rapide (li) :
    m,i,j = plus_grande_sous_liste_rapide_r (li,0,len(li))
    return li[i:j]

li = [ 4,-6,7,-1,8,-50,3]
m  = plus_grande_sous_liste_rapide(li)
print m   # affiche [7, -1, 8]

li = [1,2,3,4,5,-98,78,9,7,7]
m  = plus_grande_sous_liste_rapide(li)
print m   # affiche [79, 9, 7, 7]
```

Le coût de cette fonction est au pire en $O(n \ln n)$ et c'est la réponse optimale à cette seconde question.

fin exo 2.10.3 □

2.11 Année 2013, préparation

2.11.1

Enoncé

On construit une séquence selon le procédé suivant :

1. On tire un nombre entier entre 0 et 2.
2. On l'ajoute à la séquence.

3. Si le nombre tiré est 0, on s'arrête.
4. Si c'est 1, on tire à nouveau une fois et on répète le même processus depuis l'étape 2.
5. Si c'est 2, on tire deux nombres qu'on ajoute à la séquence et on tire encore autant de fois que la somme des deux nombres tirés. Pour chacun d'entre eux, on répète le processus depuis l'étape 3.

Rappel : voici deux lignes de code permettant de générer un nombre aléatoire entier entre 0 et 5 inclus

```
import random
i = random.randint(0,5)
```

1) Construire une fonction qui construit une séquence telle que celle définie plus haut.

2) Construire une fonction qui calcule la moyenne des longueurs des séquences obtenues (sur 1000 séquences par exemple) ?

Correction

L'énoncé peut suggérer qu'il faut agir différemment selon que le nombre entier aléatoire est 0, 1 ou 2. Cependant, il est utile de remarquer que le nombre de tirages restant à faire dépend de la longueur de la séquence de nombres et de la somme des nombres qu'elle contient. Si on note `s` la séquence de nombres aléatoires dans un état intermédiaire, le nombre de tirages aléatoires restant à effectuer est égal à $\texttt{sum(s)} - \texttt{len(s)} + 1$. On vérifie que cela fonctionne lorsque `s` contient juste un nombre.

La seconde question ne pose pas de problème puisqu'il s'agit de faire une moyenne des longueurs d'un grand nombre de séquences, 100 dans le programme qui suit.

```
#coding:latin-1
import random

# question 1
def sequence () :
    res = [ ]
    nb = 1
    while nb > 0 :
        i = random.randint(0,2)
        nb += i - 1
        res.append (i)
    return res

# question 2
def moyenne (nb_tirage) :
    somme = 0.0
    for i in range(nb_tirage) :
        s = sequence()
        somme += len(s)
    return somme / nb_tirage

s = sequence ()
print len(s),s
m = moyenne (100)
print m
```

Après quelques exécutions, on remarque que cette moyenne n'est pas jamais la même ou tout simplement que le programme ne se termine pas. Il y a deux explications possibles :

1. Soit 100 tirages ne suffisent pas pour faire converger la moyenne des longueurs,
2. Soit la moyenne n'existe pas.

Le programme informatique ne permet pas de répondre de manière certaine à cette question, il permet juste d'avoir une intuition qui serait dans ce cas que la moyenne n'existe pas. La preuve doit être mathématique.

Aparté

On note $S_n = (N_1, N_2, ..., N_n)$ une séquence de nombres aléatoires tirés dans l'ensemble $\{0, 1, 2\}$ avec les probabilités (a, b, c). Cette séquence correspond à celle de l'énoncé dans le cas où $a = b = c = \frac{1}{3}$. L'intuition est que si $a > c$, la moyenne des longueurs des séquences est finie et si $a \leqslant c$ alors elle est infinie.

La démonstration qui suit repose sur l'indépendance des tirages aléatoires. On note la variable aléatoire $\#S$ qui désigne la longueur d'une séquence générée selon le processus décrit au paragraphe précédent. On décompose cette variable comme ceci :

$$\#S = k + \#S_k \tag{2.11}$$

Où $\#S_k$ est la longueur de la séquence après le $k^{\text{ième}}$ élément exclu. En tenant compte de l'indépendance des tirages et en supposant que l'espérance de $\#S$ existe, on peut dire que :

$$\mathbb{E}\left(\#S_k | \sum_{j=0}^{k} N_j \leqslant k+1\right) = 0 \tag{2.12}$$

$$\mathbb{E}\left(\#S_k | \sum_{j=0}^{k} N_j = k+2\right) = \mathbb{E}\left(\#S\right) \tag{2.13}$$

$$\mathbb{E}\left(\#S_k | \sum_{j=0}^{k} N_j = k+3\right) = 2\mathbb{E}\left(\#S\right) \tag{2.14}$$

La première assertion est évidente. Si à partir d'un certain rang, la somme des nombres tirés est inférieure à $k+1$, la séquence s'arrête. La deuxième assertion l'est également, à partir du rang k, il reste un nombre à tirer, l'espérance de la longueur de la séquence qui commence à partir de cette position est identique à celle au début de celle-ci.

Pour la dernière assertion, imaginons que nous devions tirer 2 nombres aléatoires. On peut le faire aux positions $k+1$ et $k+2$ ou on peut tirer le premier nombre, continuer la séquence, attendre qu'il n'y ait plus de nombres à tirer puis tirer le second. Dans ce cas, on comprend que l'espérance de la longueur est bien 2 fois celle d'une séquence. On en déduit que :

$$\mathbb{E}\left(\#S_k\right) = \sum_{m=0}^{\infty} \mathbb{E}\left(\#S_k | \sum_{j=0}^{k} N_j = k+m+1\right) \mathbb{P}\left(\sum_{j=0}^{k} N_j = k+m+1\right) \tag{2.15}$$

$$= \sum_{m=0}^{\infty} m\mathbb{E}\left(\#S\right) \mathbb{P}\left(\sum_{j=0}^{k} N_j = k+m+1\right) \tag{2.16}$$

Pour utiliser ce raisonnement, on isole le premier nombre s_0 de la séquence aléatoire S en $S = (N_0, S\ N_0)$. $\#S_1$ est la séquence S privée de N_0. On applique le résultat précédent :

$$\mathbb{E}\left(\#S\right) = \begin{cases} 1 & \text{si } N_0 = 0 \\ 1 + \mathbb{E}\left(\#S\right) & \text{si } N_0 = 1 \\ 1 + 2\mathbb{E}\left(\#S\right) & \text{si } N_0 = 2 \end{cases} \tag{2.17}$$

On en déduit que :

$$\begin{aligned}\mathbb{E}(\#S) &= a + b[1 + \mathbb{E}(\#S)] + c[1 + 2\mathbb{E}(\#S)] \\ &= a + b + c + \mathbb{E}(\#S)(b + 2c) \\ &= 1 + \mathbb{E}(\#S)(1 - a + c) \end{aligned} \tag{2.18}$$

On en déduit que :

$$\mathbb{E}(\#S) = \frac{1}{a - c} \tag{2.19}$$

Sachant que cette quantité est forcément positive, elle n'est définie que si $a > c$. Le programme suivant permet de vérifier qu'en simulant des séquences pour plusieurs valeurs a, b, c, on retrouve bien une espérance proche de celle donnée par cette formule.

```
#coding:latin-1
import random

def randomint (a,b,c) :
    x = random.random()
    if x <= a : return 0
    elif x <= a+b : return 1
    else : return 2

def sequence (a,b,c) :
    res = [ ]
    nb = 1
    while nb > 0 :
        i = randomint(a,b,c)
        nb += i - 1
        res.append (i)
    return res

def moyenne (nb_tirage,a,b,c) :
    somme = 0.0
    for i in range(nb_tirage) :
        s = sequence(a,b,c)
        somme += len(s)
    return somme / nb_tirage

a,c = 0.3, 0.2
b = 1-a-c

moy = 1.0 / (a-c)
print "calcul",moy

m1 = moyenne (100000, a,b,c)
print "simulée", m1
```

Second aparté

$S_n = (N_1, N_2, ..., N_n)$ est toujours une séquence de nombres aléatoires tirés dans l'ensemble $\{0, 1, 2\}$ avec des probabilités équiprobables (a, b, c), le nombre de tirages T_n restant à effectuer après n tirages est :

$$T_n = \sum_{i=1}^{n} N_i - n + 1 \tag{2.20}$$

Si définit la séquence $S'_n = (N'_1, ..., N'_n) = (N_1 - 1, ..., N_n - 1)$ où N'_i est une variable aléatoire à valeur dans l'ensemble $\{-1, 0, +1\}$:

$$T_n = \sum_{i=1}^{n} N'_i + 1 = T'_n + 1 \tag{2.21}$$

La séquence S'_n est de longueur finie s'il existe n tel que $T'_n = 0$. T'_n est en quelque sorte une marche aléatoire dont on peut définir l'espérance et la variance :

$$\mathbb{E}\left(T'_n\right) = \sum_{i=1}^{n} \mathbb{E}\left(N'_i\right) = n\mathbb{E}\left(N'_1\right) = 0 \tag{2.22}$$

$$\mathbb{V}\left(T'_n\right) = \sum_{i=1}^{n} \mathbb{V}\left(N'_i\right) = n\mathbb{V}\left(N'_1\right) = n\left[\mathbb{E}\left((N'_1)^2\right) - \left(\mathbb{E}\left(N'_1\right)\right)^2\right] = n(a + c - (c - a)^2) \tag{2.23}$$

Dans la suite, on pose $e = -1$. On définit le nombre U tel que : $U = \inf\{u | T'_u = e\}$. U est la longueur de la séquence S. C'est aussi une variable aléatoire qui vérifie :

$$\begin{cases} T'_U = e \\ \forall u < U,\ T'_u \neq e \end{cases} \tag{2.24}$$

U est un temps d'arrêt pour le processus aléatoire $(S'n)_n$. On s'intéresse maintenant à la séquence S'_n. Etant donné que chaque élément peut prendre trois valeurs, il existe 3^n séquences différentes. On va chercher à calculer la probabilité de chaque séquence S'_n vérifiant :

$$\forall u < n,\ T'_u = \sum_{i=1}^{u} N'_i \neq e \tag{2.25}$$

Donc notre cas, on suppose $e = -1$ ce qui implique pour la marche aléatoire de rester positive. Le raisonnement serait le même pour $e > 0$. La probabilité $\mathbb{P}(U = u)$ revient à énumérer toutes les marches aléatoires (ou le nombre de chemins) qui terminent à e tout en restant supérieures à e entre 1 et u exclu. On définit p_{ui} le nombre de chemins terminant par $T'_u = e$ et ne passant jamais par e ($\forall k < u$, $T'_k \neq e$). On définit :

$$p_{ue} = f_e(u) = \mathbb{P}\left(T'_u = e,\ T'_k \neq e\ \forall k < u\right) \tag{2.26}$$

Si $e < 0$, on peut construire p_{ui} par récurrence :

$$p_{1k} = \begin{cases} 0 & \text{si } k \neq 1 \\ 1 & \text{si } k = 0 \end{cases} \tag{2.27}$$

$$p_{uk} = \begin{cases} c\,p_{u-1,k-1} & +b\,p_{u-1,k} & +a\,p_{u-1,k+1} & \text{si } k > e+1 \\ & b\,p_{u-1,k} & +a\,p_{u-1,k+1} & \text{si } k = e+1 \\ & & a\,p_{u-1,k+1} & \text{si } k = e \\ 0 & & & \text{sinon} \end{cases} \tag{2.28}$$

A partir de cette définition, on peut désormais écrire que si l'espérance de U existe, alors :

$$\mathbb{E}(U) = \sum_{u=1}^{\infty} u\mathbb{P}(U = u) = \lim_{n\to\infty} \sum_{u=1}^{n} u\mathbb{P}(U = u) \tag{2.29}$$

$$= \lim_{n\to\infty} \sum_{u=1}^{n} u\mathbb{P}(T_u = e, T_k \neq e\,\forall k < u) \tag{2.30}$$

$$= \lim_{n\to\infty} \sum_{u=1}^{n} up_{ue} = \lim_{n\to\infty} r_n \tag{2.31}$$

On repasse à l'informatique pour avoir l'intuition mathématique de la limite de $(r_u)_u$ lorsque u tend vers l'infini.

```
#coding:latin-1
import random, math

# question 1
def calcul_suite_a (n, e, a,b,c) :
    p = {}
    p [0,0]  = 1
    for u in range (1,n+1) :
        for k in range (e,u+2) :
            if   k == e   : p [u,k] = a * p.get ( (u-1,k+1), 0 )
            elif k == e+1 : p [u,k] = a * p.get ( (u-1,k+1), 0 ) + \
                                      b * p.get ( (u-1,k  ), 0 )
            elif k >  e+1 : p [u,k] = a * p.get ( (u-1,k+1), 0 ) + \
                                      b * p.get ( (u-1,k  ), 0 ) + \
                                      c * p.get ( (u-1,k-1), 0 )
    return p

def affiche_proba (ps, e) :
    n     = max ( [ k[1] for k,z in ps.iteritems () ] )
    moy = 0.0
    logru = []
    logu  = []
    for u in range (1, n+1) :
        p = ps.get((u,e),0)*1.0
        moy += p * u
        mes = "u % 3d P(U=u) %1.6g r_u %1.6g" % (u, p, moy)
        if u < 3 or u %50 == 0 : print mes
        logru.append(math.log(moy))
        logu.append(math.log(u))

    import pylab
    pylab.plot ( logu, logru, "o")
    pylab.show()

a,c = 1.0/3, 1.0/3
b = 1-a-c
e = -1

su = calcul_suite_a(600,e,a,b,c)
affiche_proba(su, e)
```

Pour $a > c$, on vérifie que la suite $(r_u)_u$ converge vers l'expression (2.19). Pour $a = b = c = \frac{1}{3}$, cela donne :

```
u    1 P(U=u) 0.333333    r_u  0.333333
u    2 P(U=u) 0.111111    r_u  0.555556
u   50 P(U=u) 0.0013566   r_u  5.57241
u  100 P(U=u) 0.00048407  r_u  8.38753
u  150 P(U=u) 0.000264311 r_u 10.5627
u  200 P(U=u) 0.000171942 r_u 12.4016
u  250 P(U=u) 0.000123146 r_u 14.0242
u  300 P(U=u) 9.37388e-05 r_u 15.4926
u  350 P(U=u) 7.44204e-05 r_u 16.8438
u  400 P(U=u) 6.09325e-05 r_u 18.1021
u  450 P(U=u) 5.10779e-05 r_u 19.2843
u  500 P(U=u) 4.36202e-05 r_u 20.4028
u  550 P(U=u) 3.78157e-05 r_u 21.4669
u  600 P(U=u) 3.31933e-05 r_u 22.4839
```

Il en ressort que la suite $P(U = u)$ semble tendre vers 0 à l'infini. Ceci signifierait que la probabilité de construire une séquence S_n infinie est nulle. Mais la suite $(r_u)_u$ semble tendre vers l'infini ce qui signifirait

que la moyenne des longueurs des séquences initiales S_n n'existe pas. On vérifie en traçant le graphe $(\log u, \log r_u)_u$ (voir figure 2.4). Il suggère que $r_n \sim cn^\alpha$ avec $0 < \alpha < 1$.

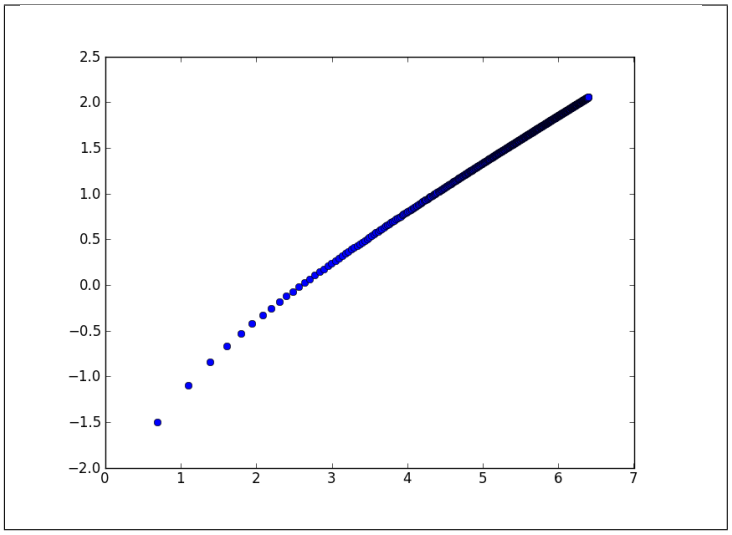


Figure 2.4 : *Graphe* $(\log u, \log r_u)_u$ *défini par (2.31).*

Il reste à démontrer formellement que la suite r_n tend vers l'infini. Pour cela, on définit :

$$f_e(n) = \mathbb{P}(U = n) = \mathbb{P}\left(T'_n = e, T'_i \neq e \text{ si } i < n\right) \tag{2.32}$$

$f_e(n)$ est différent de la suite p_{n0}. La marche aléatoire pour atteindre -1 au temps n doit nécessaire commencer par 0 ou 1. Si on note, k le premier temps auquel elle passe par 0, on peut décompser $f_{-1}(n)$:

$$f_{-1}(1) = c \tag{2.33}$$

$$f_{-1}(n) = bf_{-1}(n-1) + c\left[\sum_{k=2}^{n-1} \mathbb{P}\left(\sum_{i=2}^{k} S'_i = -1\right) \mathbb{P}\left(\sum_{i=k+1}^{n} S'_i = -1\right)\right] \tag{2.34}$$

On en déduit que :

$$f_{-1}(1) = a \tag{2.35}$$

$$f_{-1}(n) = bf_{-1}(n-1) + c\left[\sum_{k=2}^{n-1} f_{-1}(k-1) f_{-1}(n-k)\right] \tag{2.36}$$

On pose :

$$F_e(s) = \sum_{n=1}^{\infty} f_e(n) s^n \tag{2.37}$$

$F_e(1) = \sum_{n=1}^{\infty} f_e(n)$ correspond à la probabilité que la marché aléatoire atteint e en un temps fini. La moyenne des longueurs des séquences $S'n$ est défini comme le temps moyen d'arrivée en e : $\sum_{n=1}^{\infty} nf_e(n)$. On utilise le théorème de la convergence monotone [14] pour montrer que :

14. http://fr.wikipedia.org/wiki/Th%C3%A9or%C3%A8me_de_convergence_monotone

$$\sum_{n=1}^{\infty} n f_e(n) s^n = F_e'(s)$$
$$\Longrightarrow \lim_{s\to 1} \sum_{n=1}^{\infty} n f_e(n) s^n = \sum_{n=1}^{\infty} n f_e(n) = F_e'(1) \tag{2.38}$$

On cherche une relation fonctionnelle du type $x[F_{-1}(s)]^2 + yF_{-1}(s) + z = 0$ en utilisant (2.36).

$$\begin{aligned}
[F_{-1}(s)]^2 &= \left[\sum_{n=1}^{\infty} f_{-1}(n) s^n\right]^2 \\
&= \sum_{n=3}^{\infty} s^{n-1} \left[\sum_{k=2}^{n-1} f_{-1}(k-1) f_{-1}(n-k)\right] \\
&= \sum_{n=3}^{\infty} s^{n-1} \frac{1}{c} \left[f_{-1}(n) - b f_{-1}(n-1)\right] \\
&= \frac{1}{cs} \sum_{n=3}^{\infty} s^n f_{-1}(n) - \frac{b}{cs} \sum_{n=3}^{\infty} s^n f_{-1}(n-1) \\
&= \frac{1}{cs} \left[F_{-1}(s) - f_{-1}(1)s\right] - \frac{b}{c} F_{-1}(s) \\
&= F_{-1}(s) \left[\frac{1}{cs} - \frac{b}{c}\right] - \frac{f_{-1}(1)}{c} \\
\Longrightarrow \quad & cs[F_{-1}(s)]^2 - F_{-1}(s)(1-bs) - s f_{-1}(1) = 0
\end{aligned} \tag{2.39}$$

En résolvant le polynôme $csx^2 - (1-bs)x - sf_{-1}(1) = 0$, il est possible de déterminer l'expression de $F_{-1}(s)$. On rappelle que $f_{-1}(1) = a$. Une seule des solutions du polynôme du second degré est positive [15].

$$F_{-1}(s) = \frac{(1-bs) + \sqrt{(1-bs)^2 + 4acs^2}}{2cs} \tag{2.40}$$

Il ne reste plus qu'à dériver et à trouver la limite lorsque $s \to 1$.
A suivre.

fin exo 2.11.1 □

2.11.2

Enoncé

On considère une matrice 10×10 remplie de 0 et de 1 aléatoirement avec la probabilité d'avoir 1 égale à 0,2.

1) Construire une telle matrice.

2) Compter le nombre de points m_{ij} de la matrice vérifiant les conditions suivantes :

1. $m_{ij} = 0$
2. ~~$m_{i-1,j} = 1$ ou $m_{i+1,j} = 1$ ou $m_{i,j-1} = 1$~~ ou $m_{i,j+1} = 1$

15. Les solutions d'un polynôme du second degré de type $ax^2 + bx + c = 0$ sont de type : $x = \frac{-b+/-\sqrt{b^2-4ac}}{2a}$.

Correction

Pour obtenir 1 avec une probabilité de 0,2, il suffit de tirer un nombre aléatoire N entre 1 et 5 inclus et de ne considérer que le cas où $N = 1$.

```
import random
N = 10
M = [ [  1 if random.randint(1,5) == 1 else 0 for i in range (N) ] for j in range(N) ]
for l in M : print l

nb = 0
for i in range(N) :
    for j in range (N) :
        if   i > 0   and M[i-1][j] == 1 : nb += 1
        elif i < N-1 and M[i+1][j] == 1 : nb += 1
        elif j > 0   and M[i][j-1] == 1 : nb += 1
        elif j < N-1 and M[i][j+1] == 1 : nb += 1
print nb
```

Cela donne pour un exemple :

```
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 1, 1]
[0, 0, 0, 0, 1, 0, 1, 1, 0, 1]
[0, 0, 1, 0, 1, 0, 0, 1, 1, 1]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
[0, 1, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
46
```

Ce n'est pas évident de vérifier qu'on ne s'est pas trompé. Un moyen simple consiste à prendre une valeur de N plus petite.

```
[0, 1, 0]
[0, 0, 1]
[0, 0, 0]
4
```

fin exo 2.11.2 □

2.11.3

Enoncé

On considère une matrice 10×10 remplie de nombres entiers aléatoires tirés entre 0 et 100. On apelle M cette matrice.

1) Créer une autre matrice N qui vérifie : $N_{ij} = \frac{M_{ij}}{\sum_{i=1}^{10} M_{ij}}$. Le module `numpy` simplifie l'écriture du programme.

Correction

```
import random, numpy
N  = 10
M  = [ [  1.0 if random.randint(1,5) == 1 else 0.0 for i in range (N) ] for j in range(N) ]
M  = numpy.matrix(M)
MM = numpy.matrix(M)
for i in range (N) :
    s = numpy.sum(M[i,:])  # ou M[i,:].sum()
    if s > 0 : MM [i,:] = M [i,:] / s
print MM
```

Le dernier exercice cache deux pièges. Le premier est le problème des divisions entières. Si on remplace `1.0` et `0.0` par 1 et 0 sur la troisième ligne, tous les nombres manipulés deviennent entiers. La matrice `MM` est alors peuplée de 0 et de 1 uniquement. Le second piège intervient quand on pense avoir résolu le premier en forçant une division réelle en multipliant d'un côté par `1.0` :

```
import random, numpy
N  = 5
M  = [ [  1 if random.randint(1,5) == 1 else 0 for i in range (N) ] for j in range(N) ]
M  = numpy.matrix (M)
MM = numpy.matrix(M)
for i in range (N) :
    s = numpy.sum(M[i,:])
    if s > 0 : MM [i,:] = M [i,:]*1.0 / s   # multiplication par 1.0
print MM
```

Comme initialement, on a créé la matrice `M` avec des entiers, le module `numpy` refuse l'ajout ultérieur de nombres réels. On peut regretter que le module `numpy` soit aussi strict ou ne jette une erreur indiquant au programmeur qu'il s'est trompé. Le fait même de vérifier les types des objets contenus dans la matrice et ceux qu'on lui injecte aurait le désavantage de ralentir les calculs.

fin exo 2.11.3 □

2.12 Année 2013

2.12.1

Enoncé

L'objectif de cet exercice est de colorier l'espace situé entre les deux spirales de la figure 2.5.

Le code qui a servi à construire ces deux spirales vous est fourni en pièce jointe [16]. Ce code vous est donné, aucune question ne vous sera posée dessus. Il est présent à la fin de l'énoncé de l'exercice. **Afin d'éviter son inclusion (et son impression), votre programme devra impérativement commencer par :**

```
#coding:latin-1
import exoS
matrice = exoS.construit_matrice(100)
```

Pour visualiser la matrice et obtenir la figure 2.5, il faut écrire :

```
exoS.dessin_matrice(matrice)   # cette fonction place un point bleu pour une case contenant 1,
                               # rouge pour une case contenant 2,
                               # rien si elle contient 0
```

16. lien http://www.xavierdupre.fr/enseignement/complements_site_web/td_note_2013_novembre_2012_exoM.py

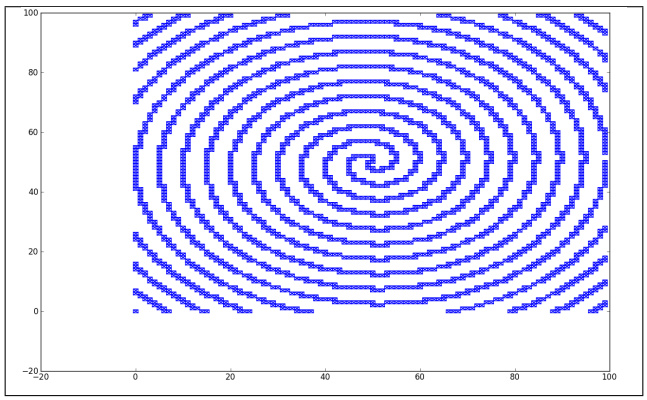


Figure 2.5 : *Double spirale*

Au départ la matrice retournée par la fonction `construit_matrice` contient soit 0 si la case est vide, soit 1 si cette case fait partie du tracé d'une spirale. L'objectif de cet exercice est de colorier une zone de la matrice.

1) Ecrire une fonction qui prend comme entrée une matrice, deux entiers et qui retourne la liste des voisins parmi les quatre possibles pour lesquels la matrice contient une valeur nulle. On fera attention aux bords du quadrillage. (2 points)

```
def voisins_a_valeur_nulle ( matrice, i, j ) :
    resultat = [ ]
    # ...
    return resultat
```

2) En utilisant la fonction précédente, écrire une fonction qui reçoit une liste de points de la matrice contenant une valeur nulle et qui retourne tous les voisins possibles contenant une valeur nulle pour tous les points de la liste. (2 points)

```
def tous_voisins_a_valeur_nulle ( matrice, liste_points ) :
    resultat = [ ]
    # ...
    return resultat
```

3) On a tous les éléments pour écrire l'algorithme de coloriage :

1. On part d'un point $(i0, j0)$ qui fait partie de la zone à colorier, qu'on insère dans une liste `acolorier` = [(`i0`, `j0`)].
2. Pour tous les points (i, j) de la liste `acolorier`, on change la valeur de la case (i, j) de 0 à 2.
3. Pour tous les points (i, j) de la liste `acolorier`, on regarde les quatre voisins $(i, j+1), (i, j-1), (i+1, j), (i-1, j)$. Si la matrice contient 0 pour un de ces voisins, on l'ajoute à la liste et on retourne l'étape précédente tant que cette liste n'est pas vide.

En utilisant la fonction précédente, écrire une fonction qui colorie la matrice à partir d'un point $(i0, j0)$. (3 points)

```
def fonction_coloriage( matrice, i0, j0) :
    # ...
```

4) Tester la fonction précédente avec le point de coordonnées $(53, 53)$ puis afficher le résultat avec la fonction `dessin_matrice`. (1 point)

Il y avait deux questions différentes selon les énonces.

5) Enoncé 1 : Ecrire une fonction qui retourne la surface coloriée. (2 points)

```
def surface_coloriee ( matrice ) :
    # ...
    return surface
```

6) Enoncé 4 : Créer une autre fonction identique à celle de la question 3 à ceci près qu'elle doit s'arrêter après que environ 1000 points distincts ont été coloriés. (2 points)

```
def fonction_coloriage_environ_1000 ( matrice, i0, j0 ) :
    # ...
    return surface
```

Programme générant la spirale

```
#coding:latin-1
import math

# cette fonction construit deux spirales imbriquées dans une matrice nb x nb
# le résultat est retourné sous forme de liste de listes
def construit_matrice (nb) :
    mat = [ [ 0 for x in range (0,nb) ] for y in range(0,nb) ]

    def pointij (nb,r,th,mat,c,phase) :
        i,j = r*th * math.cos(th+phase), r*th*math.sin(th+phase)
        i,j = int(i*100/nb), int(j*100/nb)
        i,j = (i+nb)/2, (j+nb)/2
        if 0 <= i < nb and 0 <= j < nb :
            mat[i][j] = c
        return i,j

    r = 3.5
    t = 0
    for tinc in range (nb*100000) :
        t  += 1.0 * nb / 100000
        th  = t * math.pi * 2
        i,j = pointij (nb,r,th,mat,1, 0)
        i,j = pointij (nb,r,th,mat,1, math.pi)
        if i >= nb and j >= nb : break

    return mat

# cette fonction reçoit une matrice sous forme de liste de listes contenant des entiers : 0,1,2
# à chaque valeur est associée une couleur :
# 0 pour blanc, 1 pour bleu, 2 pour rouge
def dessin_matrice (matrice) :
    import pylab
    colors = { 1: "blue", 2:"red"  }
    for i in range(0,len(matrice)) :
        for j in range (0, len(matrice[i])) :
            if matrice [i][j] in colors :
                pylab.plot ([i-0.5,i-0.5,i+0.5,i+0.5,i-0.5,i+0.5,i-0.5,i+0.5],
                            [j-0.5,j+0.5,j+0.5,j-0.5,j-0.5,j+0.5,j+0.5,j-0.5],
                        colors [ matrice[i][j] ])
    pylab.show()

if __name__ == "__main__" :
    matrice = construit_matrice(100)
    dessin_matrice(matrice)
```

Correction

```
#coding:latin-1
import td_note_2013_novembre_2012_exoS as exoS
```

```
# question 1, exo S (1 ou 4)
def voisins_a_valeurs_nulle (matrice,i,j) :
    res = []
    if i > 0                 and matrice[i-1][j] == 0 : res.append ( (i-1,j) )
    if i < len(matrice)-1    and matrice[i+1][j] == 0 : res.append ( (i+1,j) )
    if j > 0                 and matrice[i][j-1] == 0 : res.append ( (i,  j-1) )
    if j < len(matrice[i])-1 and matrice[i][j+1] == 0 : res.append ( (i,  j+1) )
    return res

# question 2, exo S (1 ou 4)
def tous_voisins_a_valeurs_nulle (matrice, liste_points) :
    res = []
    for i,j in liste_points :
        res += voisins_a_valeurs_nulle (matrice, i,j)
    return res

# question 3, exo S (1 ou 4)
def fonction_coloriage ( matrice, i0, j0) :
    # étage 1
    acolorier = [ ( i0, j0 ) ]
    while len (acolorier) > 0 :
        # étape 2
        for i,j in acolorier :
            matrice [i][j] = 2
        # étape 3
        acolorier = tous_voisins_a_valeurs_nulle ( matrice, acolorier )
        # on enlève les doublons car sinon cela prend trop de temps
        d = { }
        for i,j in acolorier : d [i,j] = 0
        acolorier = [ (i,j) for i,j in d ]

# question 5, exo S (version 1)
def surface_coloriee (matrice) :
    surface = 0
    for line in matrice :
        for c in line :
            if c == 2 : surface += 1
    return surface

# question 5, exo S (version 4)
def fonction_coloriage_1000 ( matrice, i0, j0) :
    acolorier = [ ( i0, j0 ) ]
    nb = 0                                # ligne ajoutée
    while len (acolorier) > 0 :
        for i,j in acolorier :
            matrice [i][j] = 2
            nb += 1                       # ligne ajoutée
        if nb > 1000 : break              # ligne ajoutée
        acolorier = tous_voisins_a_valeurs_nulle ( matrice, acolorier )
        d = { }
        for i,j in acolorier : d [i,j] = 0
        acolorier = [ (i,j) for i,j in d ]

# question 4, exo S (1 ou 4)
matrice = exoS.construit_matrice(100)
fonction_coloriage  (matrice, 53, 53)
exoS.dessin_matrice(matrice)
print surface_coloriee (matrice) # retourne 3258

# question 5, exo S (version 4) vérification
matrice = exoS.construit_matrice(100)
fonction_coloriage_1000  (matrice, 53, 53)
exoS.dessin_matrice(matrice)
print surface_coloriee (matrice) # retourne 1002
```

fin exo ?? □

2.12.2

Enoncé

Certaines questions suggèrent l'utilisation du module `numpy`, ne pas le faire ne sera pas pénalisé si les réponses proposées produisent des résultats équivalents.

Cet exercice utilise des données [17] fournies en pièce jointe. Elles comptabilisent le nombre d'équipements sportifs par canton. Il faudra utiliser le programme également fourni en pièce jointe [18] pour les récupérer sous forme de tableau. Ce code ne sera pas modifié durant l'examen excepté le dernier paramètre transmis à la fonction `construit_matrice` qui permet de ne considérer qu'une partie des données pour tester plus rapidement ses idées sur les premières lignes. On souhaite mesurer si les Français ont accès aux mêmes équipements sportifs sur tout le territoire français. **Afin d'éviter son inclusion (et son impression), votre programme devra impérativement commencer par :**

```
#coding:latin-1
import exoM
fichier_zip   = exoM.import_module_or_file_from_web_site ("equipements_sportif_2011.zip")
fichier_texte = exoM.unzip_fichier (fichier_zip)
# enlever le dernier paramètre 500 pour avoir le tableau complet
colonne, intitule, variables = exoM.construit_matrice (fichier_texte, 500)
    # colonne   : contient le nom des colonnes
    # intitule  : contient les deux premières colonnes du fichier textes avec du texte
    # variables : contient les autres colonnes avec des valeurs numériques
```

Les lignes suivantes permettent de convertir les informations extraites en un tableau `numpy` [19].

```
import numpy
intitule  = numpy.array(intitule)  # array et non matrix
variables = numpy.array(variables) # array et non matrix

# utilisation de numpy pour sélectionner des lignes spécifiques
print intitule [ intitule[:,1] == "Chevroux", : ]  # affiche [['01102' 'Chevroux']]
print variables[ intitule[:,1] == "Chevroux", : ]  # affiche [[  82.    1.   12 ...
```

1) Le tableau `intitule` a deux colonnes : le code postal et la ville. On veut créer un tableau `intitule3` qui contient trois colonnes : celles de `intitule` et le département déduit du code postal. Quelques fonctions utiles : (2 points)

```
print tab.shape                     # si tab est une matrice ou un tableau numpy à deux dimensions,
                                    # tab.shape est un couple (nb_lignes, nb_colonnes)
a = numpy.column_stack ( ( m, e ) ) # coller deux matrices, tableaux ayant le même nombre de lignes
```

Au final, la ligne ['01008','Ambutrix'] deviendra ['01008','Ambutrix','01'].

2) Construire la liste des départements, cette liste contient une unique instance du code du département. Elle doit être triée par ordre croissant. Il n'est pas recommandé d'utiliser `numpy` pour cette question. En anglais, tri se dit *sort*. (2 points)

3) Construire un tableau D de dimension $d \times v$ où d est le nombre de départements distincts, v est le nombre de variables (normalement 105). Le coefficient D_{ij} est la somme des valeurs pour la variable j et

17. lien http://www.xavierdupre.fr/enseignement/complements_site_web/equipements_sportif_2011.zip
18. lien http://www.xavierdupre.fr/enseignement/complements_site_web/td_note_2013_novembre_2012_exoM.py
19. Documentation : http://docs.scipy.org/doc/numpy/reference/

le département i. Si A désigne le tableau `variables`, B le tableau à trois colonnes de la question 1, C la liste des départements distincts (question 2) :

$$D_{ij} = \sum_{k|B_{k,3}=C_i} A_{kj}$$

L'objectif de cette question est d'agréger des données par départements alors qu'elles sont disponibles par canton. (3 points)

Remarque : l'instruction suivante pourrait être utile.

```
# crée une matrice de dimension nb_lignes x nb_colonnes initialisés à zéro
mvide = numpy.zeros ( ( nb_lignes, nb_colonnes) )
```

4) La colonne 5 du tableau D (la première colonne a l'indice 0) contient la population. Créer un autre tableau E qui vérifie : $E_{ij} = D_{ij}/D_{i5}$. (1 point)

5) Le programme fourni en pièce jointe contient une fonction `coefficient_gini` qui calcule le coefficient de Gini [20]. On l'utilise pour comparer le nombre d'équipements par habitants. Il vaut 0 si ce ratio est constant quelque soit le département, il vaut 1 si un seul département propose un certain type d'équipement. Entre 0 et 1, il indique l'inégalité de la distribution. Quel est l'équipement sportif le plus inégalitairement réparti sur tout le territoire ? (2 points)

Remarque : les lignes suivantes pourront vous aider.

```
li = list ( mat [:,i] )           # convertit une colonne d'un tableau numpy en une liste
print colonne[0][i+2]             # affiche le label de la colonne i
gini = exoM.coefficient_gini (li) # retourne le coefficient de Gini
                                  # pour la liste li
```

Programme préparant les données :

```
#coding:latin-1
import math, sys

# extrait les données depuis un site internet, puis les écrit à côté du programme
# ne fait rien si le fichier a déjà été téléchargé
def import_module_or_file_from_web_site (module) :
    import os
    if os.path.exists ("data\\equipement_sportifs_2011\\" + module) :
        return "data\\equipement_sportifs_2011\\" + module
    if not os.path.exists (module) :
        url = "http://www.xavierdupre.fr/enseignement/complements/" + module
        import urllib2
        if module.lower().endswith("zip") :
            f = urllib2.urlopen (url, "rb")
            t = f.read()
            f.close()
            f = open(module, "wb")
            f.write(t)
            f.close()
        else :
            f = urllib2.urlopen (url)
            t = f.read()
            f.close()
            f = open(module, "w")
            f.write(t)
```

20. http://fr.wikipedia.org/wiki/Coefficient_de_Gini

```
            f.close()
    return module

# extrait le fichier texte contenu dans le fichier zip
# et l'enregistre à côté du programme
# ne fait rien si cela est déjà fait
def unzip_fichier (fichier_zip) :
    import zipfile, os
    file = zipfile.ZipFile (fichier_zip, "r")
    res = None
    for info in file.infolist () :
        filename = info.filename
        res = filename
        if not os.path.exists (filename) :
            data = file.read(filename)
            f = open (filename,"w")
            if sys.version.startswith("3.") :
                data = str (data, encoding="iso-8859-1")
                data = data.replace("\r","").split("\n")
                data = [ _ for _ in data if len (_) > 1 ]
                data = "\n".join(data)
            f.write (data)
            f.close()
    file.close ()
    return res

# construit le tableau extrait du fichier précédent
# les deux premières lignes contiennent la description des colonnes
# les autres lignes contiennent les données elles-même
# pour aller plus vite à chaque exécution, on peut limiter le nombre de lignes
# il faudra toutes les utiliser pour l'exécution final
def construit_matrice (fichier, stop_apres = -1) :
    def float_except(x) :
        try : return float(x)
        except : return -1
    f = open (fichier, "r")
    lines = [ line.replace("\n","").split("\t")[:107] \
            for line in f.readlines()[:stop_apres] ]
    f.close ()
    colonne = lines [:2]
    lines = lines [2: ]
    lines = [ line [:2] + [ float_except(x) for x in line [2:] ] \
            for line in lines if len(line)>5 ]
    intitule = [ line[:2] for line in lines ]
    lines = [ line[2:] for line in lines ]
    return colonne, intitule, lines

def coefficient_gini (valeurs) :
    #voir http://fr.wikipedia.org/wiki/Coefficient_de_Gini
    valeurs.sort()
    gini = 0
    s = 0
    for (i,v) in enumerate (valeurs) :
        gini += (i+1)*v
        s += v
    gini = 2*gini / (len(valeurs)*s) - (len(valeurs)+1.0)/len(valeurs)
    return gini

if __name__ == "__main__" :
    fichier_zip   = import_module_or_file_from_web_site ("equipements_sportif_2011.zip")
    fichier_texte = unzip_fichier (fichier_zip)

    # enlever le dernier paramètre 500 pour avoir le tableau complet
    colonne, intitule, variables = construit_matrice (fichier_texte, 500)
```

```
    import numpy
    intitule  = numpy.array(intitule)
    variables = numpy.array(variables)

    # affichage des colonnes
    for i in range (len(colonne[0])) : print (i,colonne[0][i], " --- ", colonne[1][i])

    # utilisation de numpy pour sélectionner des lignes spécifiques
    print (intitule  [ intitule[:,1] == "Chevroux", : ])
    print (variables [ intitule[:,1] == "Chevroux", : ])
```

Correction

```
#coding:latin-1
import numpy
import td_note_2013_novembre_2012_exoM as exoM

fichier_zip   = exoM.import_module_or_file_from_web_site ("equipements_sportif_2011.zip")
fichier_texte = exoM.unzip_fichier (fichier_zip)

# enlever le dernier paramètre 500 pour avoir le tableau complet
colonne, intitule, variables = exoM.construit_matrice (fichier_texte)

import numpy
intitule  = numpy.array(intitule)
variables = numpy.array(variables)

# question 1, exo M (2 ou 3)
code_postaux = [ intitule[i,0] [:2] for i in range (intitule.shape[0] ) ]
intitule3    = numpy.column_stack ( (intitule, code_postaux) )

# question 2, exo M (2 ou 3)
comptage = {}
for i in range (intitule3.shape[0]) :
    comptage [intitule3 [i,2] ] = 0
departements = [ k for k in comptage ]
departements.sort()

# question 3, exo M (2 ou 3)
D = numpy.zeros ( (len(departements), variables.shape[1] ) )
for i in range (len (departements)) :
    d = departements [i]
    for j in range (variables.shape[1]) :
        D [i,j] = variables [ intitule3 [:,2] == d, j ].sum()

# question 4, exo M (2 ou 3)
E = numpy.zeros ( D.shape )
for i in range (E.shape[0]) :
    E [i,:] = D[i,:] / D[i,5]

# question 5, exo M (2 ou 3)
ginis = []
for j in range (E.shape[1]) :
    li = list ( E [:,j] )
    gini = exoM.coefficient_gini (li)
    ginis.append ( (gini, colonne[0][j+2]) )
ginis.sort ()
for line in ginis : print line

# les dernières lignes du tableau sont :
#(0.86910090569180598, 'Domaine skiable')
#(0.88139092467853186, 'Sports nautiques avec au moins une aire de pratique couverte')
#(0.89326137963164931, 'Domaine skiable - nombre de pistes')
#(0.9348918282098031,  'Parcours sportif avec au moins un parcours couvert')
```

```
#(0.93902978850018792, 'Domaine skiable avec au moins une piste \xe9clair\xe9e')
#(0.94625459043715754, '\xc9quipement de cyclisme avec au moins une piste couverte')
#(0.95743849241598267, 'Sports nautiques - nombre de places en tribune')
#(0.97248425032547758, 'Domaine skiable avec au moins une piste couverte')
#(0.97718065858676906, 'Parcours sportif - nombre de places en tribune')
#(0.98637386313881081, 'Terrain de golf - nombre de places en tribune')
#(0.98969072164948457, 'Domaine skiable - nombre de places en tribune')
```

fin exo ?? □

Index

Table des matières