Travaux pratiques n°11

Manipulation de structures de données en Python
Application au calcul matriciel
Résolution de l'équation matricielle $AX = B$

F. Cuvellier, J. Tanoh

M.P.S.I. 1, 2015–2016

1 Structures de données de type séquence

Une séquence est un objet

- itérable (c'est-à-dire qu'une variable peut parcourir),
- dont les éléments sont accessibles (par leurs indices, écrits entre crochets `[]`),
- auquel on peut associer une longueur (`len()`).

Python dispose de cinq structures de données de type séquence, la chaîne de caractères (`str`), le n-uplet (`tuple`), la liste (`list`) et deux structures de formats binaires qui ne seront pas abordées ici, l'octet `bytes` (non mutable) et le tableau d'octets `bytearray` (mutable)).
L'accès à un élément se fait par l'opérateur d'extraction (*slice* en anglais), dont la syntaxe connaît trois formes :

1. `seq[ind]`, pour donner l'élément de `seq` d'indice `ind` ;
2. `seq[start:end]`, pour donner la structure de même type que `seq`, formée des éléments d'indice compris entre `start` inclus et `end` exclu ;
3. `seq[start:end:step]`, pour la même action que précédemment, à ceci près que la différence entre un indice et le suivant est `step`.

Le premier élément a l'indice `0`, le suivant l'indice `1`, etc. Le dernier élément a l'indice `-1`, le précédent l'indice `-2`, etc.

On peut aussi répliquer une séquence à l'aide de l'opérateur `*`.

1.1 Chaînes de caractères

1.1.1 Description

Une chaîne de caractères est un objet non mutable formé d'une suite de caractères (codés par défaut en Unicode). Son type est `str`. La fonction `str()` permet aussi bien de créer une chaîne de caractères que de convertir un objet en chaîne de caractères.
Une chaîne de caractères peut être délimitée de trois façons, par des apostrophes (`'`), par des guillemets (`"`) ou par des triples guillemets (`"""`). Dans chaque cas, le signe employé doit être le même au début et à la fin de la chaîne. D'ordinaire, le caractère de nouvelle ligne sert à marquer la fin d'une expression, mais ce n'est pas le cas à l'intérieur d'un jeu de parenthèses, d'un de crochets, d'un d'accolades ou d'un de *triples guillemets*.

Les chaînes se concatènent à l'aide de l'opérateur `+`.

1.1.2 Comparaison de chaînes de caractères

On peut comparer bit à bit deux chaînes de caractères à l'aide des opérateurs `<`, `<=`, `==`, `!=`, `>` et `>=`. (Les surprises sont nombreuses du fait que le codage en Unicode n'est pas injectif.)

1.1.3 Méthodes sur les chaînes de caractères

De nombreuses méthodes agissent sur les chaînes de caractères. Plusieurs servent à l'édition de texte (conversion, codage, transformation de minuscules en majuscules, etc.). La commande `help(str)` les décrit.
Notons que pour une chaîne de caractères `s`,

- `s.count(t)` retourne le nombre d'occurrences de `t` ;
- `s.find(t)` retourne l'indice de la première occurrence de `t` s'il en existe une, `-1` sinon ;
- `s.index(t)` retourne l'indice de la première occurrence de `t` s'il en existe une, `ValueError` sinon ;
- `s.join(seq)` retourne à partir d'une séquence `seq` de chaînes de caractères la chaîne de caractères formée des éléments de `seq` séparés par la chaîne `s` ;
- `s.split(t)` retourne la liste des sous-chaînes de `s` séparées par la chaîne `t`. (Par défaut, `t` est une espace.)

(De nombreux arguments optionnels peuvent être passés.)

1.2 n-Uplets

1.2.1 Description

Le n-uplet (`tuple`) est une séquence ordonnée d'objets. Il est non mutable.

Les n-uplets se concatènent à l'aide de l'opérateur `+`.

1.2.2 Comparaison des n-uplets

Les n-uplets se comparent terme à terme à l'aide des opérateurs de comparaison `<`, `<=`, `==`, `!=`, `>` et `>=`.

1.2.3 Méthodes sur les n-uplets

Seules deux méthodes s'appliquent à cette structure de données. Pour un n-uplet `t`, `t.count(x)` retourne le nombre d'occurrences de `x`, et `t.index(x)` l'indice de la première occurrence de `x` s'il en existe une, `ValueError` sinon.

1.3 Listes

1.3.1 Description

Une liste est une séquence ordonnée d'objets. Elle est mutable. L'opérateur de concaténation est `+`. En particulier, on peut concaténer à une liste `L` une liste `M` à l'aide de l'opérateur `+=`.

1.3.2 Méthodes sur les listes

À une liste `L`, on peut appliquer les méthodes suivantes.

- `L.append(x)` met l'objet `x` au bout de la liste `L`.
- `L.count(x)` retourne le nombre d'occurrences de `x` dans la liste `L`.

- `L.extend(m)` met au bout de `L` les éléments de l'objet itérable `m`. L'opérateur `+=` produit le même résultat.
- `L.index(x)` retourne l'indice de la première occurrence de `x` s'il en existe une, `ValueError` sinon.
- `L.insert(i, x)` insère l'objet `x` à la place d'indice `int(i)`.
- `L.pop()` retourne le dernier élément de `L` et le supprime de `L`.
- `L.pop(i)` retourne l'élément d'indice `s` et le supprime de `L`.
- `L.remove(x)` supprime la première occurrence de l'objet `x` s'il en existe une, produit `ValueError` sinon.
- `L.reverse()` transforme `L` en la liste parcourue à partir de la fin.
- `L.sort(...)` transforme `L` en la liste ordonnée. Des arguments optionnels peuvent être passés.

1.3.3 Définition d'une liste par compréhension

On peut définir une liste par compréhension à l'aide de l'instruction `[ expr for x in obj if bool ]` où `x` est une variable qui parcourt l'objet itérable `obj`, l'expression de `x` `expr` est évaluée à condition que le booléen `bool` soit vrai.
Par exemple, pour obtenir les années bissextiles de 1850 à 2050, on peut se servir des instructions

```
leap_years = []
for year in range(1850, 2050):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        leap_years.append(year)
```

ou de l'instruction

```
leap_years = [year for year in range(1850, 2050)
              if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)]
```

On peut se servir d'autant de variables qu'on veut. Par exemple, l'instruction

```
L = [a + b for a in range(10) for b in range(10) if abs(a-b) < 2]
```

produit la liste formée des sommes de deux nombres entiers naturels strictement plus petits que 10 dont la valeur absolue de la différence est strictement plus petite que 2. On obtient la même liste `L` avec les instructions suivantes.

```
L = []
for a in range(10):
    for b in range(10):
        if abs(a-b) < 2:
            L.append(a+b)
```

1.4 Exercices

Exercice 1.

1. Transformer la chaîne de caractères `"abc"` en n-uplet et en liste.
2. Reformer la chaîne de caractères `"abc"` à partir de chaque résultat.

Exercice 2. On note `s` la chaîne de caractères `"J'aime les mathematiques."`.

1. Afficher `s` sans puis avec la fonction `print()`.

2. Extraire la sous-chaîne de `s` formée des caractères d'indice pair.

3. Scinder la chaîne `s` suivant les espaces, puis suivant la lettre `a`.

4. Écrire la chaîne `s` à l'envers. (La sous-chaîne formée des huit premiers caractères est `".seuqita"`.)

5. Répliquer la chaîne de caractères `s` dix fois, à raison d'une phrase par ligne.

Exercice 3. Former la liste *ordonnée* des nombres $a^2 + b^2$ pour tous les couples (a, b) de nombres entiers naturels non nuls plus petits que 100, premiers entre eux. Les éléments devront être distincts.

2 Application au calcul matriciel

2.1 Représentation d'une matrice

On convient de représenter une matrice sous la forme d'une liste de listes. Plus précisément, chaque ligne d'une matrice à s lignes et t colonnes

$$M = \begin{pmatrix} m_{11} & m_{12} & \cdots & m_{1t} \\ m_{21} & m_{22} & \cdots & m_{2t} \\ \vdots & \vdots & & \vdots \\ m_{s1} & m_{s2} & \cdots & m_{st} \end{pmatrix}$$

est vue comme une liste de longueur t, et la matrice M comme la liste de ces s listes.
On notera au passage que, du fait que le premier indice d'une liste en Python est 0, l'élément d'indice (h, k) de M est `M[h-1][k-1]`.

Exercice 4. Avec la convention précédente, la matrice à éléments entiers

$$\begin{pmatrix} 5 & -6 & 7 & -2 \\ 11 & 2 & -13 & 1 \\ 4 & 101 & 3 & 8 \end{pmatrix}$$

est représentée par la liste `[[5, -6, 7, -2], [11, 2, -13, 1], [4, 101, 3, 8]]`. On voudrait la voir sous la forme d'une chaîne de caractères

```
(  5   -6    7  -2 )
( 11    2  -13   1 )
(  4  101    3   8 )
```

dont voici la description :

- Chaque ligne est délimitée par des parenthèses,
- Dans une colonne donnée de la matrice où le plus grand nombre de caractères qu'exige la représentation de ses nombres est c, le nombre n devient la chaîne de caractères de longueur $c + 2$ obtenue par la concaténation d'une chaîne d'au moins une espace, de la chaîne `str(n)` et de la chaîne d'une espace.

1. Écrire une fonction à deux paramètres, un nombre entier positif ou négatif n et un nombre entier c supérieur ou égal au nombre de caractères nécessaires à la représentation de n, qui retourne la chaîne de caractères de longueur $c+2$ formée (par concaténation) d'une ou de plusieurs espaces, de la chaîne `str(n)` et d'une espace.

   Par exemple, pour le couple $(-4, 3)$ elle devra retourner la chaîne de caractères `"__-4_"`, à ceci près qu'ici chaque espace a été remplacée par un souligné.

2. Donner une fonction à un paramètre, une matrice `M` de nombres entiers relatifs, qui retourne la liste dont l'élément d'indice `k` est la plus grande des longueurs des chaînes de caractères représentant les nombres de la colonne d'indice `k` de `M`.

3. Écrire une fonction qui transforme une matrice en la chaîne de caractères décrite précédemment.

4. Donner la fonction qui effectue la transformation réciproque.

2.2 Transposition

Exercice 5. À toute matrice $M = (m_{hk})$ à s lignes et t colonnes, on associe la matrice à t lignes et s colonnes dont l'élément d'indice (k, h) est l'élément d'indice (h, k) de M. On l'appelle la matrice *transposée* de M. Donner une fonction à un paramètre, une liste de listes représentant une matrice, qui retourne la liste de listes représentant la matrice transposée.

2.3 Opérations algébriques sur deux matrices

Exercice 6. Donner une fonction qui retourne la somme de deux matrices de même type, passées en arguments.

Exercice 7. Donner une fonction qui retourne le produit de deux matrices de types convenables, passées en arguments.

2.4 Déterminant d'une matrice

On peut définir de façon récursive le déterminant d'une matrice carrée d'ordre n de la façon suivante.

- Le déterminant d'une matrice carrée d'ordre 1 est son unique élément : $\det(m_{11}) = m_{11}$.
- Le déterminant d'une matrice $M = (m_{hk})$ carrée d'ordre $n > 1$ s'obtient par la formule

$$\det(M) = \sum_{h=1}^{n} (-1)^{h+n} m_{hn} \det(M^{(hn)})$$

où l'on note ici $m^{(hn)}$ la matrice obtenue à partir de M en supprimant la ligne d'indice h et la colonne d'indice n (développement suivant la dernière colonne).

Exercice 8.

1. Donner une fonction à trois paramètres, une matrice M, un indice de ligne h et un indice de colonne k, qui retourne la matrice obtenue à partir de M en en supprimant la ligne d'indice h et la colonne d'indice k. (On pourra adopter l'indexation de Python, où le premier élément d'une liste a l'indice 0, mais on devra alors modifier la formule qui donne le déterminant de M par récurrence.)

2. Donner une fonction qui retourne le déterminant d'une matrice carrée.

On se propose de résoudre l'équation matricielle $AX = B$. D'une façon générale, une matrice M sera une liste de listes, toutes de même longueur, mais une matrice colonne pourra aussi être vue comme une simple liste de nombres.
On notera qu'en Python, l'indexation de chaque liste commence par 0, contrairement à l'usage courant en mathématiques. Par ailleurs, on n'oubliera pas que le résultat d'un calcul peut être nul en théorie et non nul en pratique (pour un logiciel comme Python).

3 Algorithme du pivot

3.1 Copies de collections

Dans les exercices suivants, on sera amené à modifier des listes. Du fait qu'en Python l'affectation se fonde sur la référence aux objets, si deux références `A` et `B` portent sur le même objet, toute modification de `B` concerne aussi l'objet `A`.
Par exemple, si `A` est la liste `[0, 1, 2]`, à la suite des opérations `B = A` et `B[2] = 3`, l'objet `A` est `[0, 1, 3]`.
Pour effectuer une copie d'une liste, on peut se servir de l'opérateur d'extraction (*slice* en anglais) : `B = A[:]`. Dans ce cas, `B` se réfère à un autre objet, et une modification sur lui ne concernera pas l'objet auquel se réfère `A`.
Pour effectuer une copie d'une collection, on peut se servir des fonctions `copy.copy()` et `copy.deepcopy()` (du module `copy`). La première effectue une copie élément par élément de la collection passée en argument, la seconde effectue une copie de la collection, mais aussi une de toute collection qui en serait un élément ou un élément d'élément, etc.

Exercice 9. Dans chacune des questions suivantes, on copie un objet `old` : `new = old` (par exemple), et on modifie sa copie. À la fin, on compare `old` et `new`. On doit obtenir des objets différents. (L'objet `old` ne doit pas être modifié.)

1. À partir de `old = [2, 3, 5, 7]`, effectuer la modification sur la copie `new[3] = 8`.
2. À partir de `old = [2, 3, 5, [7, 11]]`, effectuer la modification `new[3][0] = 8`.

3.2 Systèmes de Cramer triangulaires

Exercice 10.

1. Donner un algorithme pour la résolution de l'équation matricielle $TX = B$, où T est une matrice triangulaire supérieure dont les éléments diagonaux sont tous non nuls.
2. Le réaliser en Python. (La fonction obtenue ne s'assurera pas que la matrice T est triangulaire.)

3.3 Opérations élémentaires sur les lignes d'une matrice

Exercice 11. Donner des fonctions `typeI()`, `typeII()`, `typeIII()` qui effectuent les opérations élémentaires de type I ($L_h \leftarrow L_h + \alpha L_k$, où h et k sont des indices distincts), de type II ($L_h \leftarrow \alpha L_h$, où α désigne un nombre inversible) et de type III ($L_h \leftrightarrow L_k$, où h et k sont des indices distincts).

3.4 Recherche d'un pivot

Exercice 12. Donner une fonction `pivot()`, à deux paramètres `A` et `k`, qui recherche le pivot de plus grand module dans la colonne d'indice `k` de la matrice `A`. Elle devra aussi donner l'indice de ligne de ce pivot.

3.5 Systèmes de Cramer

Exercice 13. Donner une fonction qui réalise la méthode du pivot pour les systèmes de Cramer.

3.6 Systèmes quelconques

Exercice 14. Donner une fonction qui réalise la méthode du pivot pour les équations matricielles $AX = B$ quelconques. Elle retournera une solution particulière et une base du sous-espace vectoriel des solutions de l'équation homogène associée, ou l'ensemble vide, selon que le système est compatible ou non.

3.7 Application au calcul de l'inverse d'une matrice carrée

Exercice 15. Donner une fonction qui résolve l'équation $AX = I_n$.

Exercice 16. Pour les nombres entiers naturels n plus petits que 9, inverser la matrice de Hilbert d'ordre n, c'est-à-dire la matrice carrée d'ordre n dont l'élément d'indice (h, k) est $1/(h + k - 1)$ $(1 \leqslant h, k \leqslant n)$. Vérifier le résultat.

3.8 Usage du module NumPy

Le module `numpy.linalg` dispose de fonctions pour traiter les questions d'Algèbre linéaire. Par exemple, la fonction `solve` permet de résoudre les équations matricielles.

4 Autres méthodes

4.1 Méthode de point fixe

L'inverse d'une matrice diagonale dont les éléments diagonaux sont tous non nuls se calcule aisément. On peut généraliser cette situation.
Une matrice carrée d'ordre n dont le module de chaque élément diagonal est strictement plus grand que la somme des modules des autres éléments de la ligne est inversible.
Considérons le cas d'une telle matrice A. Soient D la matrice diagonale dont les éléments diagonaux sont ceux de A et H la matrice $A - D$. Pour résoudre l'équation $AX = B$, on se sert du fait que toute suite $(X_n)_{n\in\mathbf{N}}$ telle que, pour tout entier naturel n,

$$X_{n+1} = D^{-1}(-HX_n + B)$$

est convergente vers la solution X_∞ de l'équation $AX = B$. Plus précisément, en notant de façon générale $\|M\|$ le plus grand des modules des éléments de M, on peut établir la relation

$$\|X_\infty - X_n\| \leqslant \frac{R^n}{1 - R}\|X_1 - X_0\|,$$

où R désigne le nombre strictement plus petit que 1

$$\max_{1\leqslant k\leqslant n} \frac{1}{a_{kk}} \sum_{h\neq k} |a_{hk}|.$$

Exercice 17.

1. Écrire un algorithme fondé sur ces formules.
2. Quelle est sa complexité ?
3. Le réaliser en Python.

4.2 Calcul du polynôme minimal d'une matrice carré

Étant donné un endomorphisme u d'un espace vectoriel de dimension finie d, pour tout élément $\vec{x}$ de E, la suite $(u^n(\vec{x}))_{n\in\mathbf{N}}$ d'éléments de E est liée. Il existe donc un plus petit entier naturel n tel que $u^n(\vec{x})$ soit une combinaison linéaire des vecteurs $u^k(\vec{x})$ $(0 \leqslant k \leqslant n-1)$. Le polynôme unitaire f de plus bas degré tel que $f(u)(\vec{x})$ s'appelle le *polynôme minimal* de $\vec{x}$. On le note $p_{\vec{x}}$.
On peut démontrer que toute base de E possède un vecteur $\vec{a}$ tel que $p_{\vec{a}}(u) = 0$. Comme le polynôme minimal de tout vecteur de E divise tout polynôme annulateur de l'endomorphisme u, on obtient ainsi un procédé de calcul du *polynôme minimal* de u, c'est-à-dire du polynôme unitaire de plus bas degré qu'annule u.

Exercice 18.

1. Donner un algorithme de calcul du polynôme minimal d'un vecteur colonne pour une matrice carrée donnée.
2. Le réaliser en Python.
3. Donner un algorithme de calcul du polynôme minimal d'une matrice carrée. Évaluer sa complexité.
4. Le réaliser en Python.
5. En déduire un algorithme de calcul de l'inverse d'une matrice carrée inversible.