MATH-F-102 — Python — Projet 2020-21

Samuel Fiorini

29 mars 2021

Introduction

Ce projet de programmation est constitué de deux parties.

— La **première partie** a pour but de calculer l'ordre d'une permutation, et ce de deux manières.

— La **seconde partie** a pour but d'implémenter la procédure d'orthogonalisation de Gram-Schmidt.

Vous **devez** suivre les instructions suivantes pour la rédaction et la remise du projet.

1) Nous vous demandons de réaliser ce projet par

   groupes de 3 à 5 étudiants.

   Les groupes sont composés à votre initiative (ils ne sont pas imposés). Composez ceux-ci dès que possible !

2) Pour chaque partie, votre programme sera rendu sous la forme d'un fichier Python muni d'une extension '`.py`'. (Un fichier par partie.) Chaque programme comportera des commentaires explicatifs, signalés avec '`#`'.

3) Ces fichiers sont à télécharger sur l'UV dans l'activité devoir correspondante pour au plus tard le

   dimanche 2 mai à 23:59.

Notre évaluation dépendra en partie de la correction de vos programmes (vérifiable notamment via les tests standardisés auxquels nous procéderons), et en partie des commentaires inclus dans les deux fichiers '`.py`'.

Partie 1 (10 points)

Pour rappel, l'ordre d'une permutation α est le plus petit entier positif k tel que α^k est la permutation identique. Nous vous demandons d'écrire un programme définissant les fonctions suivantes :

1) `compose_perm(alpha,beta)` : calcule la composée `alpha` $\circ$ `beta` de deux permutations `alpha` et `beta` ;
2) `is_id(alpha)` : retourne `True` si `alpha` est la permutation identique, `False` sinon ;
3) `order1(alpha)` : calcule l'ordre de la permutation `alpha` via la définition (donc en calculant toutes les puissances α^k avec $k \geq 1$ jusqu'à ce que α^k soit l'identité) ;
4) `cycle_lengths(alpha)` : étant donné une permutation `alpha`, retourne toutes les longueurs des cycles (de la décomposition en cycles) de `alpha`, comme une liste d'entiers positifs ;
5) `Euclid(a,b)` : calcule le pgcd des entiers `a` et `b` ;
6) `lcm_of_list(l)` : calcule le ppcm des entiers d'une liste `l` donnée ;
7) `order2(alpha)` : calcule l'ordre de la permutation `alpha` en prenant le ppcm des longueurs des cycles de `alpha` (par un théorème vu au cours, ce ppcm est égal à l'ordre de la permutation).

Les permutations de degré n seront représentées comme des listes comportant tous les nombres de 0 à $n-1$, dans un certain ordre. Votre programme pourra donc déterminer le degré d'une permutation `alpha` via l'assignation `n = len(alpha)`.

Partie 2 (10 points)

Le but de cette partie est d'implémenter la procédure d'orthogonalisation de vecteurs de Gram-Schmidt dans $\mathbb{R}^n$. Nous vous demandons d'écrire un programme définissant les fonctions suivantes :

1) `scalar_prod(v,w)` : calcule le produit scalaire usuel des vecteurs `v` et `w` ;
2) `scalar_mult(mu,v)` : calcule la multiplication scalaire de `v` par `mu`, où `mu` est un scalaire et `v` est un vecteur ;
3) `proj(v,w)` : calcule la projection orthogonale du vecteur `v` sur le vecteur `w` ;
4) `subtract(v,w)` : retourne la différence `v - w` des deux vecteurs `v` et `w` ;
5) `Gram_Schmidt(E)` : applique la procédure d'orthogonalisation de Gram-Schmidt à la liste de vecteurs donnés dans `E`. Appelons `F` la liste de vecteurs résultante. Si `E` comporte k vecteurs, pour $i = 0, ..., k-1$, le i-ème vecteur de `F` est défini comme le i-ème vecteur de `E` auxquel on soustrait les projections orthogonale de ce vecteur sur chacun des vecteurs de `F` calculés précedemment (d'indices $j = 0, ..., i-1$).