

# Manipulation de suites P-récurrentes avec SageMath

Mathis CARISNTAN & Aurélien LAMOUREUX

sous la responsabilité de Marc MEZZAROBBA

09/03/2017

## Résumé

Ce rapport présente le travail que nous avons effectué au cours de ce projet. Nous présentons dans un premier temps ce que sont les suites P-récurrentes, ainsi que l'outil SageMath. Puis nous expliquons les motivations de ce projet. Enfin, nous détaillons les choix et détails de l'implémentation que nous avons réalisé, avant de discuter des limites de celle-ci et des possibles améliorations.

## 1 Introduction

...TODO...

### 1.1 Suites p-récurrentes & Algèbre d'Ore

Les suites sont beaucoup utilisées en mathématiques et dans différents domaines scientifiques, et on cherche, comme souvent en informatique, à en avoir une représentation exacte. De plus, il est généralement important que cette représentation soit également efficace pour la manipulation mathématique de ces suites.

On s'intéresse ici en particulier aux suites dites p-récurrentes. Une suite  $(u_n)_{n \in \mathbb{N}}$  sur un corps  $\mathbb{K}$  est dite p-récurrente si elle est solution d'une équation de la forme :

$$\sum_{i=0}^s p_i(n) u_{n+i} = 0 \quad (1)$$

où, les  $p_i$  sont des polynômes en  $n$ . Il est important de noter que contrairement à des suites arbitraires, les suites p-récurrentes, bien que comportant un nombre potentiellement infini de termes, peuvent être représentées exactement simplement avec la relation de récurrence, et les conditions initiales. Des exemples communs de suites p-récurrentes sont par exemple la suite de Fibonacci, ou la fonction factorielle.

$$\begin{aligned} \text{Fibonacci : } F_{n+2} - F_{n+1} - F_n &= 0, & F_0 &= 0, F_1 = 1 \\ \text{Factorielle : } (n+1)! - (n+1)u! &= 0, & 0! &= 1 \end{aligned}$$

De plus, les suites p-récurrentes forment un anneau (**Donner les détails ?**). Dès lors, il semble pertinent de réaliser une implémentation utilisant ces propriétés mathématiques afin de manipuler et utiliser les suites p-récurrentes.

TODO : ore\_algebra what & why

Intro ore... Pour le présent projet, nous n'avons besoin que des notions traitant des suites p-récurrentes.

## 1.2 Python & Sage

Sage est un outil de calcul formel libre. Il a été créé notamment pour proposer une alternative *opensource* aux logiciels existants comme Mathematica, Matlab, Maple ... Contrairement à ces logiciels, Sage s'appuie sur des outils et bibliothèques déjà existants comme NumPy, SciPy, matplotlib, FLINT et d'autres... L'utilisation de ces outils est unifiée et uniformisée au travers d'un langage basé sur Python. Ce langage présente une syntaxe qui diffère légèrement de celle de Python. Ainsi, Sage est doté d'un "pré-analyseur", qui transforme les idiomes Sage en pur Python. Ainsi, il est possible d'écrire des bibliothèques pour en Python pur ou en "langage sage". Bien qu'il existe également d'autres méthodes, on ne s'est intéressé qu'à celles-ci au cours du projet.

Comme évoqué plus haut, Sage est basé sur Python, et c'est donc naturellement que nous avons choisi ce langage pour le projet. En particulier, Python 2, puisque Sage n'est pas compatible avec Python 3 (bien que des efforts soient faits en ce sens).

Bien que Sage fournisse de nombreuses bibliothèques mathématiques, il n'inclut pas encore officiellement de bibliothèque pour l'algèbre d'Ore. Nous avons eu donc recours à une bibliothèque en cours de développement par la communauté qui implémente l'algèbre d'Ore. L'utilisation de cette bibliothèque est présentée dans [ref](#).

## 2 Méthodologie de travail, et progression

La première tâche à laquelle nous nous sommes attelés a été une recherche bibliographique, pour comprendre le sujet (les suites p-récurrentes), et nos outils (Sage et Python). Les résultats de cette démarche sont présentés dans la partie 1.

Puis nous avons commencé à discuter de l'implémentation. En accord avec notre encadrant, nous avons estimé qu'il était plus pertinent d'un point de vue pédagogique, de réaliser d'abord un module Python. Puis, une fois ce module éprouvé, si nous avions le temps, le réécrire avec la syntaxe de Sage. Cette manière de procéder devait permettre de se concentrer initialement sur le fond, et non la forme, puisque nous étions plus familiers avec Python. [Il s'est avéré par la suite que nous n'avons pas eu le temps d'aborder la réécriture.](#)

### 2.1 Module Python

La base du module a été d'écrire une classe Python ([init. n'étend aucune classes](#)). Cette classe devait notamment permettre d'utiliser la représentation basée sur la relation de récurrence, et des conditions initiales. Immédiatement après, nous avons surchargé l'opérateur `__getitem__` pour accéder au n-ième terme de la suite. Initialement, nous calculions tous les termes de la suite, jusqu'à celui voulu, que nous renvoyions, mais cette méthode est très inefficace. Nous avons donc résolu d'utiliser la fonction `forward_matrix` du module `ORE_ALGEBRA` à la place ([exemple et comparaison complexe avec Fibo ?](#))

[calculer tous les elts vs calculer que le bon élément, exemple de différence temps exécution sur 100000 !](#))

Par la suite, nous avons également surcharger les opérateurs d'addition, soustraction et multiplication, en accord avec les lois de l'algèbre d'Ore.

## Références

- [1] A. Bohr and B.R. Mottelson, Nuclear Structure, vol. 2, Benjamin, New York, 1975.
- [2] <http://ipnweb.in2p3.fr>
- [3] Nick Park, *A Grand Day Out*, 1989, [http://en.wikipedia.org/wiki/A\\_Grand\\_Day\\_Out](http://en.wikipedia.org/wiki/A_Grand_Day_Out)