Projet SFPN : Manipulation de suites P-récursives avec SageMath

Mathis Caristan & Aurélien Lamoureux

Sous la responsabilité de Marc Mezzarobba

Université Pierre & Marie Curie

29/05/2017

Contexte & problématique

Les suites P-récursives sont des objets couramment utilisés en mathématiques et en sciences.

Problématique

- **La question se pose de comment représenter et manipuler informatiquement ces objets.**
- Les suites sont infinies.

Contexte & problématique

Les suites P-récursives sont des objets couramment utilisés en mathématiques et en sciences.

Problématique

- **La question se pose de comment représenter et manipuler informatiquement ces objets.**
- Les suites sont infinies.

Solution

Il est nécessaire d'utiliser les propriétés mathématiques des suites P-récursives.

Suites P-récursives

Définition formelle

Une suite P-récursive sur un corps $\mathbb{K}$ vérifie la propriété suivante :

$$\sum_{i=0}^{k} P_i(n) u_{n+i} = 0$$

où les P_i sont des polynômes en n, et k est l'ordre de la récurrence.

Une suite P-récursive peut être représentée exactement avec sa relation de récurrence, et ses conditions initiales*

Exemples

Fibonacci : $F_{n+2} - F_{n+1} - F_n = 0$, $\quad F_0 = 0, F_1 = 1$

Factorielle : $(n+1)! - (n+1)(n!) = 0$, $\quad 0! = 1$

Algèbre d'Ore

TODO

SageMath & Python

SageMath, qu'est-ce que c'est ?

- Un logiciel de calcul formel
- Opensource
- Construit sur un ensmble d'outil pré-éxistant et Python
- Basé sur Python
- Doté d'une syntaxe spécifique pour la ligne de commande

Python ?

- C'est le langage sur lequel est basé Sage
- Python 2.7.9
- Les idiomes Sage sont transformés en Python pur
- Possibilité d'écrire des modules pour Sage en Python

La bibliothèque OreAlgebra

- Implémente l'algèbre d'Ore
- Non intégrée au projet Sage, et développée par la communauté
- Contient une partie des outils nécessaires à la réalisation du projet
  - Définir une algèbre dans laquelle travailler

    ```
    R.<n> = PolynomialRing(ZZ)
    A.<Sn> = OreAlgebra(R)
    ```
  - Les fonctions `lclm` et `symmetric_product` pour $+/\times$

    ```
    annihilSum = annihil1.lclm(annihil2)
    annihilProd = annihil1.symmetric_product(annihil2)
    ```
  - La fonction `forward_matrix_bsplit` pour le calcul d'un terme

Présentation du module

Problématique

Créer un module permettant la manipulation des suites p-récursives

Caractéristiques

- En Python
- Basé sur le modèle de programmation objet : une classe
- Surcharge d'opérateurs
- Des tests

Notre classe n'étend aucune classe pré-éxistante.

Objectifs du module

Objectifs principaux

- Un constructeur
- Les opérations $+$ et $\times$
- Une fonction pour calculer un élément

Objectifs du module

Objectifs principaux

- Un constructeur
- Les opérations $+$ et $\times$
- Une fonction pour calculer un élément

Objectifs importants

- Travailler dans différents anneaux
- Des suites constantes
- Une méthode qui teste si une suite est constante
- Les tests d'égalité/inégalité
- Un constructeur qui devine la récurrence

Objectifs du module

Objectifs principaux

- Un constructeur
- Les opérations $+$ et $\times$
- Une fonction pour calculer un élément

Objectifs importants

- Travailler dans différents anneaux
- Des suites constantes
- Une méthode qui teste si une suite est constante
- Les tests d'égalité/inégalité
- Un constructeur qui devine la récurrence

Objectifs secondaires

- Les opérateurs $<<$ et $>>$
- Un itérateur (infini)
- La division par une constante
- Un constructeur à partir d'une expression symbolique

Constructeur

C'est la méthode appelée par Python lors d'une instanciation de la classe.
C'est la première interaction de l'utilisateur avec le module.

Comportement par défaut

```
u = PRecSequence (conditions,annihilateur)
```

`conditions` est un dictionnaire

`annihilateur` est un objet du module OreAlgebra

Comportements secondaires

D'autres comportements sont possibles, grâce aux arguments optionnels de Python

- Création d'une suite constante
- "Guessing" a partir d'une liste d'éléments

Faut-il remplacer l'utilisation des arguments mots-clefs par le décorateur `@classmethod` ?

Constructeur - Problèmes

Les valeurs dégénérées

Lorsque le polynôme dominant a des racines dans $\mathbb{Z}$:

$(n-1)u_{n+1} - u_n = 0, u_0 = 1$ $\qquad$ $u_0 = 1, u_1 = (-1)u_0 = -1, u_2 = ???$

Constructeur - Problèmes

Les valeurs dégénérées

Lorsque le polynôme dominant a des racines dans $\mathbb{Z}$:

$(n-1)u_{n+1} - u_n = 0, u_0 = 1$ $\qquad$ $u_0 = 1, u_1 = (-1)u_0 = -1, u_2 = ???$

Les conditions initiales supplémentaires

Il est nécessaire de permettre de fixer des conditions supplémentaires.

Constructeur - Problèmes

Les valeurs dégénérées

Lorsque le polynôme dominant a des racines dans $\mathbb{Z}$:

$(n-1)u_{n+1} - u_n = 0, u_0 = 1$ $\qquad$ $u_0 = 1, u_1 = (-1)u_0 = -1, u_2 = ???$

Les conditions initiales supplémentaires

Il est nécessaire de permettre de fixer des conditions supplémentaires.

1re idée : Obliger l'utilisateur à renseigner les valeurs dégénérées. Obliger l'utilisateur à saisir *toutes* les valeurs jusqu'à la dernière racine.

Constructeur - Problèmes

Les valeurs dégénérées

Lorsque le polynôme dominant a des racines dans $\mathbb{Z}$:

$(n-1)u_{n+1} - u_n = 0, u_0 = 1$ $\qquad$ $u_0 = 1, u_1 = (-1)u_0 = -1, u_2 = ???$

Les conditions initiales supplémentaires

Il est nécessaire de permettre de fixer des conditions supplémentaires.

1re idée : Obliger l'utilisateur à renseigner les valeurs dégénérées. Obliger l'utilisateur à saisir *toutes* les valeurs jusqu'à la dernière racine.

2e idée : Lever des exceptions

Constructeur - Conditions initiales

Comment traiter les conditions initiales supplémentaires ?

Les valeurs n'influent pas le calcul

$u_{n+2} - u_{n+1} - u_n = 0, u_0 = 0, u_1 = 1, u_5 = 6$
Les valeurs de la suite sont $(0, 1, 1, 2, 3, 6, 8, 13, 21 \ldots)$

Les valeurs influent sur les termes suivants

$u_{n+2} - u_{n+1} - u_n = 0, u_0 = 0, u_1 = 1, u_5 = 6$
Les valeurs de la suite sont $(0, 1, 1, 2, 3, 6, 9, 15, 24 \ldots)$

Accéder à un élément de la suite

Il faut surcharger l'opérateur `__getitem__`, qui permet en Python d'accéder à l'élément n : `fibo[`n`]` ou `fibo[`n_1 : n_2`]`.

Première méthode : `to_list`

La classe de notre annihilateur propose `to_list`, qui calcule récursivement tous les termes jusqu'à n.

- Facile à implémenter
- Mais lent

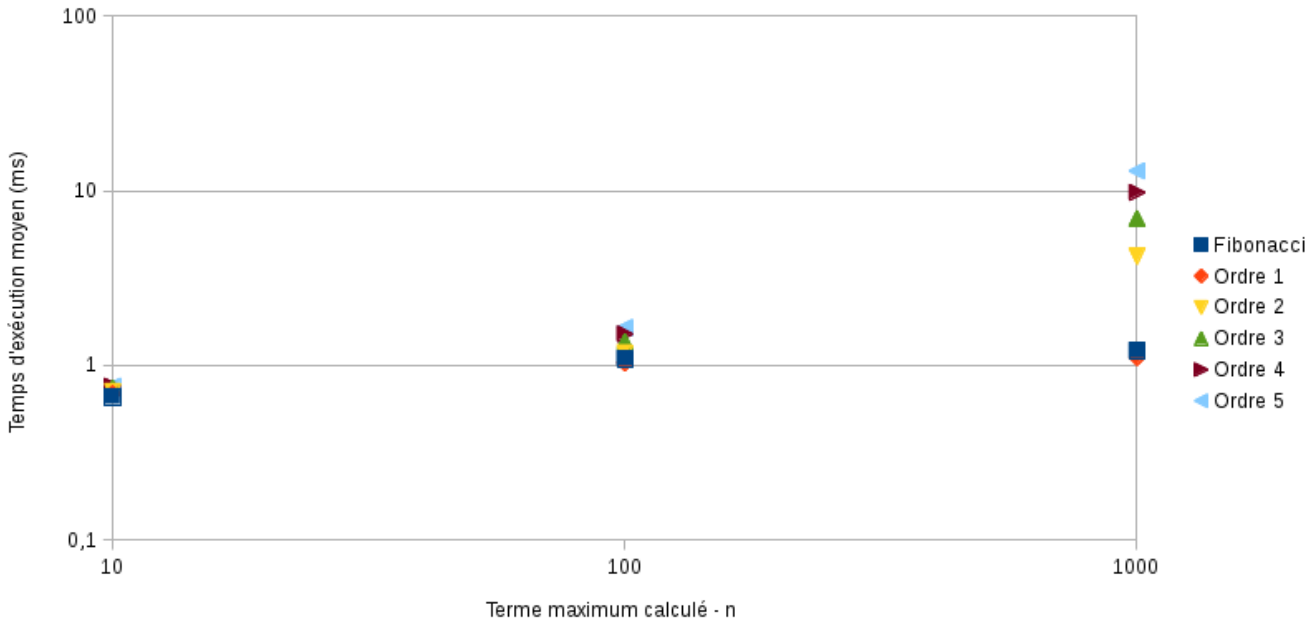
Deuxième méthode : `forward_matrix_bsplit`

Utiliser l'algèbre linéaire pour calculer directement le terme n

- Théoriquement plus rapide
- Souffre d'un temps d'amorce

Optimiser _getitem _

Nous avons comparé les temps d'exécution pour les deux méthodes.



Rapport du temps avec `to_list` sur le temps avec `forward_matrix_bsplit`.

La seconde méthode est déjà plus efficace pour des valeurs de l'ordre de 100.

Addition et multiplication

Surcharger l'addition

- L'addition $+$ correspond à l'opérateur `__add__` en Python
- Utilisation de `lclm`

Surcharger la multiplication

- La multiplication $\times$ correspond à l'opérateur `__mul__` en Python
- Utilisation de `symmetric_product`

Addition et multiplication - Procurer des conditions supplémentaires

Termes dégénérés dans les sommes/produits

Le coefficient dominant de certains produits ou sommes ont des racines dans $\mathbb{Z}$.

Utiliser les facteurs

$$\begin{aligned}
\text{Ent. Consec. : } & nE_{n+1} - (n+1)E_n = 0, \qquad E_0 = 0, N_1 = 1 \\
\text{Fibonacci : } & F_{n+2} - F_{n+1} - F_n = 0, \qquad F_0 = 0, F_1 = 1 \\
\text{Leur Sommme : } & (n-1)D_{n+3} + (-2n+1)D_{n+2} + D_{n+1} + nD_n = 0, \\
& D_0 = 0, D_1 = 2, D_2 = 3
\end{aligned}$$

D_4 est dégénéré, mais E_4 et F_4 existent.

Autres fonctions

TODO