Projet SFPN : Manipulation de suites P-récursives avec SageMath

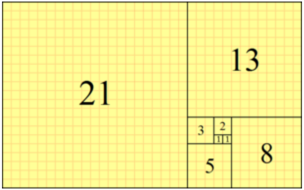
Mathis Caristan & Aurélien Lamoureux

Sous la responsabilité de Marc Mezzarobba

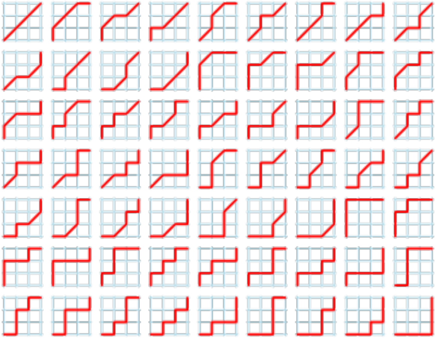
Université Pierre & Marie Curie

29/05/2017

Quelques objets mathématiques

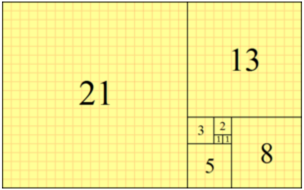


$$F_{n+2} - F_{n+1} - F_n = 0$$

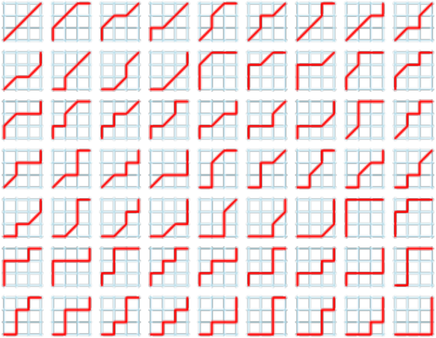


$$D(m,n) = D(m-1,n) + D(m-1,n-1) + D(m,n-1)$$

Quelques objets mathématiques



$$F_{n+2} - F_{n+1} - F_n = 0$$



$$D(m,n) = D(m-1,n) + D(m-1,n-1) + D(m,n-1)$$

mais aussi la fonction factorielle, le binôme de Newton, le nombre de partitions d'un nombre...

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

$$\begin{aligned} \text{Fibonacci : } F_{n+2} - F_{n+1} - F_n &= 0, & F_0 = 0, F_1 = 1 \\ \text{Factorielle : } (n+1)! - (n+1)(n!) &= 0, & 0! = 1 \\ \textit{Fonction anodine : } (n-2)u_{n+1} - u_n &= 0, & u_0 = 2 \end{aligned}$$

SageMath & Python

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

- Un logiciel de calcul formel
- Open source
- 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

Les algèbres d'Ore

Algèbres d'Ore

- Bien plus vaste que le cadre de ce projet
- Déterminée par un anneau, et un opérateur

Exemples

- L'anneau sera $\mathbb{R}[x]$
- L'opérateur de décalage $\mathrm{Sn}: n \mapsto n+1$
- On associe un opérateur d'annihilation à une suite :

$$\begin{aligned} \text{Fibonacci :} \quad & (\mathrm{Sn}^2 - \mathrm{Sn} - 1)F_n = 0 \\ \text{Factorielle :} \quad & (\mathrm{Sn} - (n+1))(n!) = 0 \\ (u_n)_{n\in\mathbb{N}} \ : \quad & ((n-2)\mathrm{Sn} - 1)u_n = 0 \end{aligned}$$

- Les annihilateurs sont des polynômes en n et Sn.
- On utilise la bibliothèque OreAlgebra comme boîte à outils.

Présentation du module

Problématique

Créer un module facilitant 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

Comportement par défaut

```
sage: fibo = PRecSequence ({0:0,1:1}, Sn**2 - Sn - 1)
sage: fact = PRecSequence ({0:1}, Sn - n - 1)
sage: u = PRecSequence ({0:2}, (n-2)*Sn - 1)
```

Comportements secondaires

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

- Création d'une suite constante :

  ```
  sage: u_const = PRecSequence (const=5)
  ```

- "Guessing" a partir d'une liste d'éléments :

  ```
  sage: fib_guess = PRecSequence ([0,1,1,2,3,5,8],R)
  sage: print fib_guess.annihilator
  -Sn^2 + Sn + 1
  ```

Les valeurs singulières

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

$$
\begin{aligned}
(n-2)u_{n+1} - u_n &= 0 & u_0 &= 2 \\
-2 * u_1 - u_0 &= 0 & \Rightarrow u_1 &= -1 \\
-u_2 - u_1 &= 0 & \Rightarrow u_2 &= 1 \\
0 * u_3 - u_2 &= 0 & \Rightarrow u_2 &= 0, u_3 = ???
\end{aligned}
$$

Constructeur - Problèmes

Les valeurs singulières

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

$$\begin{aligned}
(n-2)u_{n+1} - u_n &= 0 & u_0 &= 2 \\
-2 * u_1 - u_0 &= 0 & \Rightarrow u_1 &= -1 \\
-u_2 - u_1 &= 0 & \Rightarrow u_2 &= 1 \\
0 * u_3 - u_2 &= 0 & \Rightarrow u_2 &= 0, u_3 = ???
\end{aligned}$$

Les conditions initiales supplémentaires

Il est nécessaire de permettre de fixer des conditions supplémentaires.

Constructeur - Problèmes

Les valeurs singulières

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

$$\begin{aligned}
(n-2)u_{n+1} - u_n &= 0 & \quad u_0 &= 2 \\
-2 * u_1 - u_0 &= 0 & \quad &\Rightarrow u_1 = -1 \\
-u_2 - u_1 &= 0 & \quad &\Rightarrow u_2 = 1 \\
0 * u_3 - u_2 &= 0 & \quad &\Rightarrow u_2 = 0, u_3 = ???
\end{aligned}$$

Les conditions initiales supplémentaires

Il est nécessaire de permettre de fixer des conditions supplémentaires.
1re idée : Obliger l'utilisateur à renseigner les valeurs singulières. Obliger l'utilisateur à saisir *toutes* les valeurs jusqu'à la dernière racine.

Constructeur - Problèmes

Les valeurs singulières

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

$$\begin{aligned} (n-2)u_{n+1} - u_n &= 0 & u_0 &= 2 \\ -2 * u_1 - u_0 &= 0 & \Rightarrow u_1 &= -1 \\ -u_2 - u_1 &= 0 & \Rightarrow u_2 &= 1 \\ 0 * u_3 - u_2 &= 0 & \Rightarrow u_2 &= 0, u_3 = ??? \end{aligned}$$

Les conditions initiales supplémentaires

Il est nécessaire de permettre de fixer des conditions supplémentaires.
1re idée : Obliger l'utilisateur à renseigner les valeurs singulières. 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

```
sage: PRecSequence ({0:0,1:1,5:6}, Sn**2 - Sn - 1)
```

Les valeurs de la suite sont $(0, 1, 1, 2, 3, 6, 8, 13, 21 \ldots)$

Les valeurs influent sur les termes suivants

```
sage: PRecSequence ({0:0,1:1,5:6}, Sn**2 - Sn - 1)
```

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.

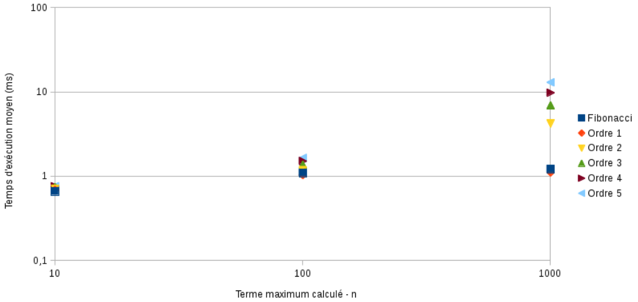
- Facile à implémenter
- Mais lent

Deuxième méthode : `forward_matrix_bsplit`

Utiliser l'algèbre linéaire pour calculer directement le terme n

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

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

Principe

- Trouver un opérateur qui annule les deux termes
- Trouver de nouvelles conditions initiales

Addition et multiplication

Principe

- Trouver un opérateur qui annule les deux termes
- Trouver de nouvelles conditions initiales

Exemple

```
sage: C = PRecSequence ({0:0}, Sn - 1)
sage: S = fibo + C ; print S.annihilator
Sn^3 - 2*Sn^2 + Sn
```

L'annihilateur de S annule fibo et C.

Addition et multiplication

Principe

- Trouver un opérateur qui annule les deux termes
- Trouver de nouvelles conditions initiales

Exemple

```
sage: C = PRecSequence ({0:0}, Sn - 1)
sage: S = fibo + C ; print S.annihilator
Sn^3 - 2*Sn^2 + Sn
```

L'annihilateur de S annule fibo et C.

Mathématiquement

```
sage: op1 = fibo.annihilator ; op2 = C.annihilator
sage: op1.lclm(op2)
Sn^3 - 2*Sn^2 + Sn
```

Addition et multiplication - Procurer des conditions supplémentaires

Valeurs singulières dans les sommes/produits

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

Utiliser les facteurs

```
sage: E = PRecSequence ({0:0}, n*Sn - n - 1)
sage: S = E + fibo ; print S.annihilator
(n-1)*Sn^3 + (-2*n+1)*Sn^2 + Sn + n
```

$$S_0 = E_0 + F_0 = 0, \quad S_1 = E_1 + F_1 = 2, \quad S_2 = E_2 + F_2 = 3$$

D_4 est singulier, mais E_4 et F_4 existent : $D_4 = E_4 + F_4 = 7$

Autres fonctions

TODO