Decomposizione delle $\mathbb{F}_q$-algebre finite

Francesca Rizzo - Prof. Patrizia Gianni

Indice

---

1 Introduzione

Questo progetto ha come oggetto il problema della decomposizione delle $\mathbb{F}_q$-algebre finite su campi finiti, ovvero della ricerca di algebre locali $\{A_i\}$ tali che $A \simeq \prod_i A_i$. Si dimostra che per le algebre finite una decomposizione esiste sempre ed il problema di decomporre l'algebra è equivalente alla ricerca di un sistema di idempotenti ortogonali indecomponibili e a somma 1. La parte teorica su cui si basa l'algoritmo è presentata nella Sezione 2. Nella Sezione 3 sono descritti i dettagli dell'algoritmo per la decomposizione dell'algebra, in particolare la ricerca degli idempotenti è fatta seguendo il metodo proposto in [1]. Questo metodo è interessante dal punto di vista computazionale perchè permette di trovate, tramite una ricerca randomizzata, elementi idempotenti non banali con probabilità maggiore di $\frac{1}{2}$. Alla Sezione 4 è presente una documentazione del codice, scritto in Python usando la libreria SageMath ([2]). Infine la Sezione 5 mostra come utilizzare il codice e raccoglie diversi esempi di decomposizione di $\mathbb{F}_q$-algebre finite e applicazioni per la fattorizzazione degli ideali. Notiamo che algoritmi per la decomposizione delle $\mathbb{F}_q$-algebre sono interessanti perché possono essere usati per decomporre $\mathbb{Q}$-algebre tramite processi di lifting (si veda ad esempio [1]).

2 Decomposizione di algebre finite

Dato un campo $\Bbbk$, una $\Bbbk$-algebra è $A \simeq \Bbbk[x_1, \ldots, x_n]/I$: in particolare Ricordiamo le seguenti definizioni e i seguenti fatti (si veda ad esempio [3] §2.2 e [4] §8):

Definizione. Una $\Bbbk$-algebra A si dice *finita* se A è un $\Bbbk$-spazio vettoriale finito.

Definizione. R anello: un ideale $I \subset R$ si dice 0-*dimensionale* se $\dim_{\text{Krull}}(R/I) = 0$ (cioè se ogni primo che contiene I è massimale).

Proposizione 1. $A = \Bbbk[x_1, \ldots, x_n]/I$ è una $\Bbbk$-algebra finita $\iff$ I è un ideale 0-dimensionale

Proposizione 2. Ogni $\Bbbk$-algebra finita è un anello artiniano

Teorema 3 (Teorema di struttura degli anelli artiniani)**.** Ogni anello artiniano è isomorfo a un prodotto diretto di anelli artiniani locali

2.1 Decomposizione di anelli artiniani

Definizione. Un elemento $u \in A$ si dice *idempotente* se $u^2 = u$. Inoltre due idempotenti u, v si dicono *ortogonali* se $uv = 0$.

Osserviamo che se u_i è idempotente, anche $1 - u_i$ lo è ($(1-u_i)^2 = 1 - 2u_i + u_i^2 = 1 - u_i$), ed è ortogonale ad u_i: $(1 - u_i)u_i = u_i - u_i^2 = 0$.

Definiamo anche una relazione d'ordine fra gli idempotenti:

Definizione. Dati u, v idempotenti, diciamo che u contiene v, $u \supset v$, se $uv = v$. Un idempotente u si dice *primitivo* se non contiene propriamente nessun altro idempotente non nullo.

In particolare se $u \supset v$, allora $(u - v)^2 = u^2 - 2uv + v^2 = u - v$ è ancora idempotente e $(u) = (v, u - v)$.

Vale il seguente risultato che lega gli elementi idempotenti alla decomposizione degli anelli.

Proposizione 4 (cfr. [4] §2.4)**.** Sia A un anello, allora sono fatti equivalenti

(a) esistono $A_1, \ldots, A_n$ anelli, tali che $A \simeq \prod_{i=1}^n A_i$;

(b) esistono $u_1, \ldots, u_n$ idempotenti ortogonali tali che $\sum u_i = 1$

Dimostrazione. Vediamo le due implicazioni:

- $(a) \implies (b)$: Per ogni i sia e_i l'elemento di $\prod_i A_i$ che ha componente i-esima uguale a 1 e tutte le altre nulle. È facile verificare che $\sum_i e_i = 1$, $e_i e_j = 0 \ \forall i \neq j$ e $e_i^2 = 1$. Sia $\varphi : A \xrightarrow{\simeq} \prod_{i=1}^n A_i$. Allora $\{u_i = \varphi^{-1}(e_i)\}$ è un sistema di idempotenti che soddisfa (b).

- $(b) \implies (a)$: Consideriamo gli ideali $J_i = (u_j | j \neq i)$: poiché gli u_i sono ortogonali e a somma 1, vale $\sum_{j\neq i} u_j = 1 - u_i$ e $(1-u_i)\cdot u_j = \sum_{k\neq i} u_k u_j = u_j^2 = u_j$, e quindi $J_i = (1-u_i)$.
  Chiaramente $J_i + J_j = (1)$ e $\bigcap J_i = \prod J_i = (\prod(1-u_i)) = 0$, quindi per il teorema cinese del resto vale $A \simeq \prod A/J_i$. Osserviamo che in particolare $\bar{u_i} = \bar{1} \in A/J_i$.

□

Corollario 5. Data A una $\Bbbk$-algebra finita, sono fatti equivalenti:

(a) esistono $A_1, \dots, A_n$ $\Bbbk$-algebre finite, tali che $A \simeq \prod_{i=1}^n A_i$;

(b) esistono $u_1, \dots, u_n$ idempotenti ortogonali tali che $\sum u_i = 1$

Inoltre dati $\{u_1, \dots, u_n\}$ idempotenti ortogonali tali che $\sum_i u_i = 1$ questi sono linearmente indipendenti su R.

Dimostrazione. L'equivalenza dei fatti (a) e (b) segue dalla proposizione precedente osservando che quoziente di una $\Bbbk$-algebra finita è ancora una $\Bbbk$-algebra finita.
Per vedere che elementi idempotenti ortogonali a somma 1 sono linearmente indipendenti, basta osservare che $\{\varphi(u_i) = (0, \dots, 0, 1, 0, \dots, 0)\}$ sono linearmente indipendenti in $\prod A_i$. □

Osserviamo che in particolare la dimostrazione fornisce un modo per determinare una decomposizione di A una volta trovato un sistema di idempotenti ortogonali (non nulli) con somma 1.
In particolare segue anche il seguente corollario:

Corollario 6. Sia $A \simeq R/I$ e siano $u_1, \dots, u_n$ idempotenti di A ortogonali con $\sum u_i = 1$. Allora $I = (I, 1-u_1)\cdots(I, 1-u_n) = (I, 1-u_1) \cap \cdots \cap (I, 1-u_n)$.

Dimostrazione. Dal corollario precedente abbiamo

$$R/I = A \simeq A/(1-u_1) \times \cdots \times A/(1-u_n) \simeq \prod_{i=1}^n R/(I, 1-u_i) \overset{(*)}{\simeq} R \Big/ \prod_{i=1}^n (I, 1-u_i)$$

dove l'isomorfismo $(*)$ segue dal teorema cinese del resto.
Quindi per il teorema di corrispondenza $I = \prod_{i=1}^n (I, 1-u_i)$ □

Teorema 7. Sia A anello artiniano e sia $\{u_1, \cdots, u_n\}$ un sistema di idempotenti ortogonali. Allora la decomposizione $A \simeq \prod_{i=1}^n A/(1-u_i)$ è minimale (cioè fatta da anelli locali) se e soltanto se gli u_i sono tutti primitivi.

Vediamo prima il seguente lemma

Lemma 8. Sia A un anello artiniano. A è locale $\iff$ gli unici idempotenti sono 0 e 1

Dimostrazione. Chiaramente se A è locale, gli unici idempotenti sono 0 e 1: infatti in un anello locale per ogni $x \in A$ uno fra x e $1-x$ è invertibile. Quindi $u^2 = u \implies u(1-u) = 0 \implies u = 0$ o $1-u = 0$.
Supponiamo ora che gli unici idempotenti di A siano 0 e 1: poichè A è artiniano sappiamo che si decompone come prodotto di algebre locali. Se non fosse locale allora avrebbe una decomposizione non banale e quindi ammettere un sistema di idempotenti ortogonali non banali per la Proposizione 4 □

Dimostrazione(Teorema 7). Supponiamo che la decomposizione $A \simeq \prod_{i=1}^{n} A/(1-u_i)$ sia minimale e dimostriamo che u_i è primitivo per ogni i.
Sia v idempotente di A tale che $u \supset v$, cioè $u_i v = v$. Allora $\bar{v} \in A/(1-u_i)$ è ancora idempotente, e poiché $A/(1-u_i)$ è locale, necessariamente $\bar{v}$ è 0 o $\bar{1} = \bar{u_i}$. Ma $\bar{v} = 0 \implies v \in (1-u_i) \cap (u_i) = ((1-u_i)u_i) = 0$, mentre se $\bar{v} = \bar{u_i} \implies v - u_i = u_i(v-1) \in (1-u_i) \cap (u_i) = 0$. Quindi u_i è primitivo.

Supponiamo ora che tutti gli u_i siano primitivi e dimostriamo che $A/(1-u_i)$ è locale per ogni i.
Poiché A è artiniano (cioè noetheriano di dimensione 0) anche $A/(1-u_i)$ lo è perché entrambe le proprietà si conservano passando al quoziente. Quindi per vedere che è locale basta dimostrare che non ha idempotenti non banali. Sia $\bar{v}$ un idempotente di $A/(1-u_i)$: vale $v^2 - v \in (1-u_i)$. Ma allora $u_i v$ è un idempotente di A contenuto in u_i: infatti $(u_i v)^2 - u_i v = u_i(v^2 - v) \in (1-u_i) \cap (u_i) = 0$, e $u_i(u_i v) = u_i v$. Ma poiché u_i è primitivo, o $u_i v = 0$ e quindi $0 = \bar{u_i}\bar{v} = \bar{v}$, o $u_i v = u_i$ e quindi $1 = \bar{u_i}\bar{v} = \bar{v}$. □

Corollario 9. In particolare se $\{u_1, \ldots, u_n\}$ è un sistema di idempotenti ortogonali primitivi a somma 1, la fattorizzazione $I = \prod_{i=1}^{n}(I, 1-u_i)$ è fatta di ideali primari.

Abbiamo quindi dimostrato che il problema di decomporre un'algebra finitamente generata si riconduce a determinare un sistema di idempotenti primitivi ortogonali che sommano ad 1.
In particolare poiché le $\Bbbk$-algebre finite sono anelli artiniani sappiamo che una decomposizione minimale esiste, quindi:

Corollario 10. Ogni $\Bbbk$-algebra finita ammette un sistema di idempotenti primitivi ortogonali a somma 1, e questo ha sempre cardinalità uguale al numero di ideali massimali di A.

2.2 Ricerca di un sistema di idempotenti ortogonali primitivi a somma 1

Sappiamo che ogni algebra finita ammette un sistema di idempotenti ortogonali primitivi a somma 1. Vediamo come estrarne uno conoscendo gli idempotenti dell'algebra.

Definizione. Dati u, v idempotenti, chiamiamo *raffinamento di u con v* l'elemento $z = uv$. Diciamo inoltre che v *raffina* u se $uv \neq 0, u$.

È facile verificare che z è un idempotente contenuto in u ($uz = u^2 v = uv = z$), quindi per quanto visto $(u) = (z, u-z)$.

Osserviamo che u è primitivo se e solo se non esiste nessun idempotente v che raffina u.
Infatti, se u è primitivo, ogni raffinamento z è contenuto propriamente in u, quindi non esistono raffinamenti non banali. D'altra parte, supponiamo che u non sia primitivo: allora esiste $v \neq 0, u$ idempotente tale che $uv = v$, ma quindi v raffina u.
Inoltre per ogni altro idempotente w ortogonale a u, anche z e $u - z$ sono ortogonali a w, infatti $wz = wuv = 0$, e $w(u-z) = wu - wz = 0$.

Quindi dato $\{u_1, \dots, u_k\}$ sistema di idempotenti ortogonali con $\sum_{i=1}^{k} u_i = 1$, e dato un altro idempotente w che raffina u_1, anche $\{w, u_1 - w, u_2, \dots, u_k\}$ è un sistema di idempotenti ortogonali a somma 1, di cardinalità $k+1$.
In questo modo possiamo definire un algoritmo per la ricerca di un sistema di idempotenti primitivi ortogonali a somma 1.
Partendo da un idempotente u, costruiamo l'insieme $\{u, 1-u\}$: questo è un sistema di idempotenti ortogonali a somma 1. Supponiamo di aver costruito l'insieme $\{u_1, \dots, u_k\}$: se non esistono raffinamenti di u_i per nessun i, gli u_i sono tutti primitivi, ed il processo termina, altrimenti possiamo costruire $\{v_1, \dots, v_{k+1}\}$ sistema di idempotenti ortogonali a somma 1.
Osserviamo che in particolare sappiamo già che un sistema di idempotenti ortogonali primitivi a somma 1 ha cardinalità uguale al numero di ideali massimali di A, quindi necessariamente dopo $m-1$ passi la procedura termina.
Il punto delicato di questo algoritmo in realtà è la ricerca degli idempotenti, che in generale è non banale. Per questo ci limitiamo a considerare le algebre finite su campi finiti, caso in cui è più facile trovare gli idempotenti.

2.3 Ricerca degli idempotenti

Sia A una $\mathbb{F}_q$-algebra finita, cioè $A = \mathbb{F}_q[x_1, \dots, x_n]/I$ (con I ideale 0-dimensionale).
Consideriamo il Frobenius $\phi : A \to A$ definito da $\phi(a) = a^q$: poiché A ha caratteristica q, ϕ è un omomorfismo di algebre ed in particolare è lineare come mappa di $\mathbb{F}_q$ spazi vettoriali.

È chiaro che gli idempotenti sono tutti punti fissi del Frobenius, $a^2 = a \implies a^n = a \ \forall n$.
Inoltre detta $A^\phi = \{a \in A | \phi(a) = a\}$ la sottoalgebra fissata da ϕ, vale la seguente proposizione (cfr. [1]):

Proposizione 11. Se A ha m ideali massimali, $\dim_{\mathbb{F}_q}(A^\phi) = m$, e un sistema di idempotenti primitivi ortogonali è una base di A^ϕ.

Dimostrazione. Vediamo per prima cosa che se A è locale $\dim_{\mathbb{F}_q}(A^\phi) = 1$.
Poiché A è locale, detto m il suo ideale massimale sappiamo che m è nilpotente, quindi $\mathrm{Div}_0(A) \subset m \subset \mathrm{Nil}(A)$. Sicuramente A^ϕ non contiene nilpotenti, quindi neanche divisori di 0: quindi A^ϕ è una $\mathbb{F}_q$-algebra finita che è anche un dominio, quindi è un campo. Dato che è un campo fissato fissato dal Frobenius, è proprio $\mathbb{F}_q$.

Il caso generale ora segue facilmente: $A \simeq \prod_{i=1}^{m} A_i$ con A_i algebre locali e per il punto precedente vale $A^\phi \simeq \prod_{i=1}^{m} A_i^\phi \simeq \prod_{i=1}^{m} \mathbb{F}_q$.
Per i Corollari 5 e 10 sappiamo che esistono $\{u_1, \dots, u_m\} \subset A^\phi$ idempotenti primitivi ortogonali a somma 1 e che questi sono linearmente indipendenti, quindi generano A^ϕ. □

Inoltre A^ϕ è facilmente calcolabile, nota una base di A (che si può determinare usando le basi di Groebner), possiamo scrivere la matrice M che rappresenta ϕ in quella base e calcolare $\ker M - I = A^\phi$.

3 Algoritmo per la decomposizione di algebre finite su $\mathbb{F}_q$

Usando i risultati presentati nella Sezione 2 , presentiamo un algoritmo per la decomposizione di algebre finite su campi finiti. Indicata con A l'algebra, sappiamo che $A = \mathbb{F}_q[x_1, \ldots, x_n]/I$, con I-ideale 0 dimensionale. L'algoritmo usato è il seguente:

1. Trova una base B dell'algebra
2. Costruisci la matrice M associata al Frobenius nella base B
3. Calcola $A^\phi = \ker(M - I)$ e $m = \dim(A^\phi)$
4. **if** $m = 1$:
   A è locale
   return
5. u=trova_idempotente e costruisci $V = \{u, 1 - u\}$
6. **while** $|V| < m$:
   w=trova_idempotente
   for $v \in V$:
   $y =$ raffina(v,w)
   if $y \neq 0, v$:
   rimuovi v da V
   aggiungi y e $v - y$ in V
   else: do nothing
7. **for** $i = 1, \ldots, m$: $J_i = (1 - u_i)$
8. **return**

Analizziamo i vari passi dell'algoritmo:

1. Se $A = \mathbb{F}_q[x]/I$, allora $I = (f)$ (perché $\mathbb{F}_q$ è un PID), quindi una base dell'algebra è data da $\{1, \ldots, x^{\deg(f)-1}\}$.
   Altrimenti possiamo usare le basi di Groebner per calcolare una base di A: detta G una base di Groebner dell'ideale I, si ha che una base di A è data da
   $$B = \{x^\alpha | \mathrm{lt}(g) \nmid x^\alpha \ \forall g \in G\}$$
4 Per la Proposizione 11, la dimensione di A^ϕ è uguale al numero di ideali massimali di A, quindi se $m = 1$, A è locale e non può essere decomposta ulteriormente
5 Altrimenti $m > 1$, quindi esiste almeno un idempotente non banale in A^ϕ. In questo passo cerchiamo allora un idempotente non banale di A e per farlo distinguiamo due casi:

- se $q^m = |A^\phi|$ è abbastanza piccola, possiamo elencare gli elementi di A^ϕ, e trovare così tutti gli elementi idempotenti di A;
- altrimenti la ricerca di idempotenti è più faticosa, e si usano costruzioni che a partire da elementi random di A^ϕ restituiscono elementi idempotenti.
  La costruzione usata è spiegata più nel dettaglio nel Paragrafo 3.1

6 Ad ogni passo V è un sistema di idempotenti ortogonali a somma 1. Usando l'algoritmo descritto al Paragrafo 2.2, ad ogni passo scegliamo un idempotente w e raffiniamo V con w. La procedura raffina(v, w) che raffina v con w restituisce $y = vw$.

7 Dato $V = \{u_1, \dots, u_n\}$ sistema ortogonale di idempotenti primitivi a somma 1 e $J_i = (1 - u_i)\ \forall i$, per quanto visto nel Paragrafo 2.1, si ha

$$A \simeq A/J_1 \times \cdots \times A/J_m$$

dove A/J_i sono algebre locali.

3.1 Costruzione randomizzata degli idempotenti

Nel caso in cui A è non locale, vogliamo trovare un elemento idempotente non banale di A e per farlo usiamo il metodo proposto in [1] per costruire un idempotente u a partire da un elemento $v \in A^\phi$, metodo che produce idempotenti non banali con probabilità maggiore di $1/2$. La costruzione è la seguente: scelto $v \in A^\phi$, $v \neq 0$:

- se $\mathrm{char}(A) = 2$ e $q = 2^k$ definiamo $u = \sum_{i=0}^{k-1} v^{2^i}$. Verifichiamo che in effetti u è idempotente:

$$u^2 = \left(\sum_{i=0}^{k-1} v^{2^i}\right)^2 \overset{\mathrm{char}=2}{=} \sum_{i=0}^{k-1} v^{2^{i+1}} = v^q + \sum_{i=1}^{k-1} v^{2^i} = v + \sum_{i=1}^{k-1} v^{2^i} = u$$

- altrimenti sia $t = v^{\frac{q-1}{2}}$, e definiamo $u_1 = \frac{t(t-1)}{2}$, $u_2 = \frac{t(t+1)}{2}$. Vale:

$$u_1^2 = \frac{t^2(t^2 - 2t + 1)}{4} = \frac{v^{q-1}(v^{q-1} - 2v^{\frac{q-1}{2}} + 1)}{4} = \frac{v^{\frac{q-1}{2}-1}(v^{q+\frac{q-1}{2}} - 2v^q + v^{1+\frac{q-1}{2}})}{4}$$
$$\overset{v^q=v}{=} \frac{v^{\frac{q-1}{2}-1}(v^{1+\frac{q-1}{2}} - 2v + v^{1+\frac{q-1}{2}})}{4} = \frac{v^{\frac{q-1}{2}}(2v^{\frac{q-1}{2}} - 2)}{4} = u_1$$

  quindi u_1 è idempotente, ed in maniera analoga si verifica che anche u_2 lo è.

Verifichiamo che l'idempotente trovato è non banale con probabilità maggiore di $\frac{1}{2}$.
Sappiamo dalla Proposizione 11 che $A^\phi =< e_1, \dots, e_m >_{\mathbb{F}_q}$ con $\{e_1, \dots, e_m\}$ sistema di idempotenti ortogonali primitivi a somma 1, quindi $v = \sum_{i=1}^m a_i e_i$ con $a_i \in \mathbb{F}_q$

- caso $\mathrm{char}(A) = 2$: osserviamo che $u = \sum_{i=0}^{k-1} v^{2^i} = \sum_{i=0}^{k-1} \sum_{j=1}^m a_j^{2^i} e_j = \sum_{i=0}^{k-1} \mathrm{tr}(a_j) e_j$, dove $\mathrm{tr} : \mathbb{F}_{2^k} \to \mathbb{F}_2$ è la traccia.
  Per unicità della scrittura in base sappiamo che $u = 0 \iff \mathrm{tr}(a_i) = 0\ \forall i$, $u = 1 \iff \mathrm{tr}(a_i) = 1\ \forall i$: poiché la traccia sulle estensioni separabili è surgettiva (si veda [5] §4.7 Proposizione 7) , i $\mathbb{F}_q$ ci sono $\frac{q}{2}$ elementi a traccia 0 e $\frac{q}{2}$ elementi a traccia 1. Quindi u sarà non banale con probabilità uguale a $1 - 2(\frac{q}{2})^m > \frac{1}{2}$.

- caso char$(A) \neq 2$: In questo caso In questo caso per $t \neq 0, 1, -1$ almeno uno fra $u_1 = \frac{t(t-1)}{2}$ e $u_2 = \frac{t(t+1)}{2}$ è diverso da 0 e da 1. Possiamo scrivere

$$t = v^{\frac{q-1}{2}} = \sum_{i=1}^{m} a_i^{\frac{q-1}{2}} e_i$$

troviamo un idempotente non banale se $t \neq 0, 1, -1$.

Osserviamo che $a_i \in \mathbb{F}_q \implies a_i = 0$ o $a_i^{q-1} = 1$: quindi per ogni $a_i \neq 0$ vale $a_i^{\frac{q-1}{2}} = \pm 1$.
Quindi, per unicità della scrittura in base abbiamo:

  - $t = 0$ se e solo se $a_i^{\frac{q-1}{2}} = 0 \ \forall i \iff a_i = 0 \ \forall i$, che non si verifica perché $v \neq 0$;
  - $t = 1$ se e solo se $a_i^{\frac{q-1}{2}} = 1 \ \forall i$ e $t = -1$ se e solo se $a_i^{\frac{q-1}{2}} = -1 \ \forall i$;

Sappiamo che $|\{a \in \mathbb{F}_q | a^{\frac{q-1}{2}} = 1\}| = |\{a \in \mathbb{F}_q | a^{\frac{q-1}{2}} = -1\}| = \frac{q-1}{2}$, quindi il numero di elementi v per cui $t = \pm 1$ è $2\left(\frac{q-1}{2}\right)^m$. Troviamo un idempotente non banale con probabilità maggiore di $1 - \frac{2\left(\frac{q-1}{2}\right)^m}{q^m} > 1 - \frac{1}{2^{m-1}} > \frac{1}{2}$.

Quindi la procedura trova_idempotente risulta essere nel caso random la seguente:

def trova_idempotente():

for i=1, ...

(i) Scegli un elemento random $0 \neq v \in A^\phi$

(ii) **if** char$(A) = 2$
$u = \sum_{i=0}^{k-1} v^{2^i}$
if $u \neq 0, 1$: **return** u
else: do nothing

(iii) **else**
$t = v^{q-1}/2$
if $t \neq 1, -1$:
if $t = 2$: **return** $u_2 = \frac{t(t+1)}{2}$
else: **return** $u_1 = \frac{t(t-1)}{2}$
else do nothing

return

4 Documentazione del codice implementato

Il codice è stato implementato in Python, utilizzando la libreria Sage [2].
Di seguito una breve documentazione delle funzioni presenti e del loro utilizzo.

```
class.algebra_dec(info = None, ipt = None)
```

Una classe per gestire algebre su campi finiti.
Questa classe permette la decomposizione di algebre finite su campi finiti, e prevedere la gestione di alcune operazioni su ideali 0-dimensionali (fattorizzazione) e algebre finite (ricerca degli idempotenti, costruzione del Frobenius)

Parametri:

`info`: `array_like`, `optional` sceglie se stampare o meno il file di output con le informazioni, `info[2]` può contenere il nome del file. Se non presente la scelta default è di non stampare le informazioni.

`ipt`: `array_like`, `optional` contiene i valori per costruire l'anello di polinomi $R = F_q[x_0, \dots, x_{n-1}]$, con `q = ipt[0]`, `n = ipt[1]`. Se non presente i parametri q e n vengono richiesti come input.

Attributi: `q`, `n`, `F`: campo $\mathbb{F}_q$, `R`: anello R, `v`: variabili $\{x_0, \dots, x_{n-1}\}$, `info`: 1 se stampa le informazioni, 0 altrimenti, `fileName`: stringa con il nome del file ouput se `info=1`

Metodi:

- `build_algebra(self, *ideal)` costruisce l'algebra $A = F_q[x_0, \dots, x_{n-1}]/I$ dato l'ideale I. L'ideale può essere passato come parametro o inserito da input. Nel caso di inserimento da input utilizza la funzione `self.convert_string(string)` per convertire l'input in un ideale di R, con cui definire il quoziente R/I.
  Attributi: `A`, `a`: generatori di A, `B = base_algebra()`: base dell'algebra

- `base_algebra(self)`: calcola la base B dell'algebra se l'ideale è 0 dimensionale. Se $n = 1$ una base è data da $1, x_0, \dots, x_0^{\deg f - 1}$ con f generatore di I. Altrimenti utilizza la funzione di Sage `normal_basis()`. Per potere usare le funzioni di Sage, B è un sollevamento di una base di A ad R.

- `Frobenius_matrix(self)`: calcola la matrice che rappresenta il Frobenius nella base B, utilizzando la funzione `self.in_base(w)`. Nel caso $n > 1$ utilizza la funzione di Sage `reduce()` per ridurre un elemento modulo l'ideale I.

- `in_base(self,w)`: dato $w \in R$ calcola le coordinate di w rispetto alla base B. Nel caso $n > 1$ utilizza la funzione di Sage `monomial_coefficient()`.

- `fixed_Frobenius(self)`: calcola e restituisce in output l'algebra dei punti fissi del Frobenius A^ϕ usando la funzione `self.Frobenius_matrix()`. Inoltre se la cardinalità di A^ϕ è più piccola di una certa costante, prepara la ricerca degli idempotenti per elencazione creando `self.index`.

Attributi: `dim`: dimensione di A^ϕ, `index`: intero, presente solo se la cardinalità di A è piccola

- `check_idempot(self, u)`: dati $u \in R$ controlla se è idempotente modulo I.

- `prossimo_idempot(self, V)`: usando `self.index` per scorrere gli elementi di $\mathbb{F}_q^d$, restituisce il primo elemento idempotente non banale di A non ancora trovato. Aggiorna `self.index`.

- `random_idempot_el(self, Aphi)`: estrae elementi random da A^ϕ per creare, seguendo la procedura indicata in [1], degli elementi u idempotenti che sono non banali con probabilità maggiore di $1/2$. Restituisce il primo elemento idempotente che trova.

- `find_idempot_el(self, Aphi)`: trova un idempotente di A: se la funzione `self.fixed_Frobenius()` ha preparato la ricerca per elencazione, trova il prossimo idempotente non ancora letto usando la funzione ,`self.prossimo_idempot(Aphi.basis())` altrimenti cerca un idempotente utilizzando la funzione `self.random_idempot_el(Aphi)`

- `sistem_idempot(self, Aphi)`: trova un sistema di idempotenti primitivi ortogonali a somma 1, utilizzando l'algoritmo descritto alla Sezione 2.2, usando la funzione `self.find_idempot_el()` per generare nuovi idempotenti.
  Attributi: `idem`: contiene il sistema di idempotenti ortogonali a somma 1,
  `dec` è -1 se A è locale, 1 o 0 se gli idempotenti trovati sono primitivi o meno.

- `decomponi(self, *id)`: se non è già stata costruita un'algebra A, o ha un ideale in input, costruisce l'algebra usando la funzione `self.build_algebra(*id)`. Se `self.idem` non è già presente cerca un sistema di idempotenti ortogonali primitivi a somma 1, tramite le funzioni `Aphi = self.fixed_Frobenius` e `self.sistem_idempot(Aphi)`. Costruisce `self.qzn` vettore che contiene i generatori degli ideali $J_1, \dots, J_m$ per cui $A \simeq A/J_1 \times \cdots \times A/J_m$: in particolare questi sono locali se `self.dec` è 1 (cfr. Sezione 2.1)
  Attributi: `qzn`

- `fattorizza(self, *id)`: se non è già stata costruita un'algebra A, o ha un ideale in input, costruisce l'algebra usando la funzione `self.build_algebra(*id)`. Se `self.idem` non è già presente cerca un sistema di idempotenti ortogonali primitivi a somma 1, tramite le funzioni `Aphi = self.fixed_Frobenius` e `self.sistem_idempot(Aphi)`.Costruisce `self.fact` vettore che contiene i generatori degli ideali $I_1, \dots, I_m$ per cui $I \simeq I_1 \cdots I_m$: n particolare questi sono primari se `self.dec` è 1 (cfr. Sezione 2.1)
  Attributi: `fact`

- `print_info(self)`: stampa le informazioni memorizzate su un file esterno con nome `self.fileName` se presente, o con un nome random altrimenti.

- `clear_ideal(self)`: cancella tutti gli attributi relativi all'algebra. Se `self.info=1` chiama il comando `self.print_info()`

5 Test

Per potere usare la classe `algebra_dec` basta caricarla sulla linea di comando del terminale, usando il comando `load("algebra_dec.spyx")`. Di seguito qualche esempio di utilizzo del codice per decomporre alcune algebre o fattorizzare polinomi.
Come specificato nella Sezione 4 è possibile passare le informazioni su $R = \mathbb{F}_q[x_0, \dots, x_{n-1}]$ sia da input, sia come argomenti. Per i test abbiamo preferito la seconda opzione. Inoltre per tutti i test la funzione `algebra_dec()` è stata chiamata con parametro `info = [1, string]` dove `string` è il nome desiderato per il file di output con le informazioni.

Esempio 1. Vediamo prima qualche esempio su $\mathbb{F}_q[x]/(f)$: in questo caso il processo di ricerca di una base e di scrittura in base è più semplice, e non usa le basi di Groebner. Osserviamo che in questo caso fattorizzare (f) in ideali primari corrisponde a fattorizzare il polinomio f come prodotto di potenze di polinomi irriducibili, in quando $\mathbb{F}_q[x]$ è un PID (e quindi gli ideali primari sono tutti potenze di ideali primi). Quindi se il sistema di idempotenti ortogonali a somma 1 trovato è fatto da idempotenti primitivi, `self.fattorizza((f))` restituisce una fattorizzazione del polinomio f in potenze di fattori irriducibili.

Per il primo esempio usando i comandi:

```
alg = algebra_dec(info = [1, 'F5N1'], ipt = [5,1])
#crea l'algebra A e la decompone
alg.decomponi(alg.convert_string('x0^4+x0^2+1'))
#fattorizza il polinomio x0^4+x0^2+1:
   #l'algebra è già decomposta quindi la procedura è veloce
alg.fattorizza()
#stampa le informazioni sul file di output
alg.print_info()
```

Il file di output restituito è:

```
Algebra A = F_5[x0]/(x0^4 + x0^2 + 1)
Dimensione del campo fissato dal Frobenius:2
Ricerca degli idempotenti: per elencazione
2 idempotenti trovati 2*a0^3 + 3, 3*a0^3 + 3
Decomposizione minimale
A =A/J_1*A/J_2
J_1 =(a0^3 + 1)
J_2 =(a0^3 + 4)
Fattorizzazione in ideali primari
Ideale I = I_1*I_2
I_1 =(x0^2 + 4*x0 + 1)
I_2 =(x0^2 + x0 + 1)
```

Osserviamo che in questo caso la funzione ha trovato e restituito: un sistema di idempotenti ortogonali primitivi e somma 1 di A, la decomposizione dell'algebra A e la fattorizzazione del polinomio $x^4 + x^2 + 1 = (x^2 + x + 1)(x^2 + 4x + 1) \equiv (x^2 + x + 1)(x^2 - x + 1) \pmod 5$. La ricerca degli idempotenti è avvenuta per elencazione (i valori da testare in

questo caso erano solo $5^{\dim(A^\phi)} = 5^m = 25$), ed in effetti la decomposizione ottenuta è minimale. Inoltre la fattorizzazione di $x^4 + x^2 + 1$ è fatta da fattori irriducibili.

Con un codice simile a quello procedente è stato effettuato anche il secondo test, questa volta su $F_4[x]$: in questo caso vediamo che il polinomio $x^6 + x^3 + x + 1$ viene fattorizzato come $(x^3 + x^2 + 1)(x^3 + x^2 + x + 1) = (x^3 + x^2 + 1)(x + 1)^3$, dove il primo fattore è irriducibile e il secondo potenza di un irriducibile. In output abbiamo:

```
Algebra A = F_4[x0]/(x0^6 + x0^3 + x0 + 1)
Dimensione del campo fissato dal Frobenius:2
Ricerca degli idempotenti: per elencazione
2 idempotenti trovati a0^5 + a0, a0^5 + a0 + 1
Decomposizione minimale
A =A/J_1*A/J_2
J_1 =(a0^5 + a0 + 1)
J_2 =(a0^5 + a0)
Fattorizzazione in ideali primari
Ideale I = I_1*I_2
I_1 =(x0^3 + x0^2 + 1)
I_2 =(x0^3 + x0^2 + x0 + 1)
```

È possibile anche utilizzare solo la funzione `fattorizza()` per fattorizzare un polinomio su $\mathbb{F}_q[x]$ come nel seguente esempio ricavato usando il codice seguente:

```
alg = algebra_dec(info = [1, 'F7N1'], ipt = [7, 1])
#il comando fattorizza crea l'algebra A
   #trova un sistema di idempotenti
   #e fattorizza il polinomio
alg.fattorizza(alg.convert_string('x0^6 + 4*x0^5 + 2*x0^4 + 6*x0^2 + 2*x0 + 4'))
#stampa le informazioni sul file di output
alg.print_info()
```

E il file di output restituito:

```
Algebra A = F_7[x0]/(x0^6 + 4*x0^5 + 2*x0^4 + 6*x0^2 + 2*x0 + 4)
Dimensione del campo fissato dal Frobenius:3
Ricerca degli idempotenti: per elencazione
3 idempotenti trovati
        4*a0^4 + a0^3 + 2*a0 + 6,
        2*a0^5 + 2*a0^4 + 4*a0^3 + 2*a0^2 + a0 + 3,
        5*a0^5 + a0^4 + 2*a0^3 + 5*a0^2 + 4*a0 + 6
Fattorizzazione in ideali primari
Ideale I = I_1* ... *I_3
I_1 =(x0^2 + 2*x0 + 5)
I_2 =(x0^3 + x0^2 + 6*x0 + 5)
I_3 =(x0 + 1)
```

Esempio 2. Qualche esempio su $A = F_q[x_0, \ldots, x_{n-1}]/I$ con I ideale 0-dimensionale.

Il primo esempio è su una $\mathbb{F}_5$-algebra e il metodo di ricerca random usato è quello del caso char $\neq 2$. Vediamo in questo caso come ottenere una decomposizione dell'algebra: il codice usato è il seguente.

```
alg = algebra_dec(info = [1, 'F5N3'], ipt = [5, 3])
#Ideale di R
I=alg.convert_string('(x0^2 - 2, x2^2 - 2, x1^2 + 2)')
alg.print_info()
#il comando decompone l'algebra A = R/I
alg.decomponi(I)
#stampa le informazioni sul file di output
alg.print_info()
```

E il file di output restituito:

```
Algebra A = F_5[x0, x1, x2]/(x0^2 - 2, x2^2 - 2, x1^2 + 2)
Dimensione del campo fissato dal Frobenius:4
Ricerca degli idempotenti: per elencazione
4 idempotenti trovati
        -a0*a1 + 2*a0*a2 + a1^2*a2^2 - a1*a2 - 2,
        a0*a1 + 2*a0*a2 - a1^3*a2^3 - 2*a1^2*a2^2 + 2*a1*a2 + 1,
        a0*a1 - 2*a0*a2 + a1^3*a2^3 + a1^2*a2^2 - 2*a1*a2 - 2,
        -a0*a1 - 2*a0*a2 + a1*a2 - 1
Decomposizione minimale
A =A/J_1* ... *A/J_4
J_1 =(a0*a1 - 2*a0*a2 - a1^2*a2^2 + a1*a2 - 2)
J_2 =(a0*a1 + 2*a0*a2 - a1^3*a2^3 - 2*a1^2*a2^2 + 2*a1*a2)
J_3 =(a0*a1 - 2*a0*a2 + a1^3*a2^3 + a1^2*a2^2 - 2*a1*a2 + 2)
J_4 =(a0*a1 + 2*a0*a2 - a1*a2 + 2)
```

Anche in questo caso possiamo usare solo la funzione `self.fattorizza(I)` per fattorizzare un ideale I in ideali primari: vediamo questa funzione su una $\mathbb{F}_8$-algebra (in questo caso la ricerca degli idempotenti utilizza il metodo random nel caso con char $= 2$).

```
alg = algebra_dec(info = [1, 'F8N3'], ipt = [8, 3])
I=alg.convert_string('x0^4+x0^2+1, x1^4+x1^2+1, x2^4+x2^2+1')
alg.fattorizza(I)
alg.print_info()
```

```
Algebra A = F_8[x0, x1, x2]/(x0^4 + x0^2 + 1, x1^4 + x1^2 + 1, x2^4 + x2^2 + 1)
Dimensione del campo fissato dal Frobenius:4
4 idempotenti trovati
Ricerca degli idempotenti: random
Fattorizzazione in ideali primari
Ideale I = I_1* ... *I_4
I_1 =(x0^2 + x2^2, x1^2 + x2^2, x2^4 + x2^2 + 1)
I_2 =(x0^2 + x2^2 + 1, x1^2 + x2^2 + 1, x2^4 + x2^2 + 1)
I_3 =(x0^2 + x2^2 + 1, x1^2 + x2^2, x2^4 + x2^2 + 1)
I_4 =(x0^2 + x2^2, x1^2 + x2^2 + 1, x2^4 + x2^2 + 1)
```

In questo caso la fattorizzazione dell'ideale, che è fatta da 4 ideali primari, ci garantisce che $\sqrt{I}$ è intersezione di 4 ideali massimali, che in particolare sono i radicali degli ideali I_j.

L'ultimo esempio mostra come il comando `decomponi()` (anche `fattorizza()`) possono essere usati più volte per decomporre algebre diverse (o fattorizzare ideali definirsi) definite sullo stesso anello $R = \mathbb{F}_q[x_0, \ldots, x_{n-1}]$.

In particolare in questo caso, nella seconda chiamata della funzione `alg.decomponi()` l'ideale da decomporre è stato ricavato tramite manipolazioni algebriche: questa funzionalità può essere utile per fare verifiche (spesso infatti non sono noti dei generatori di un ideale, o comunque non è facile calcolarli in modo da passarli come input).

```
alg = algebra_dec(info = [1,'F9N2'], ipt = [ 9, 2])
alg.build_algebra(alg.convert_string('(x0^4 + x1^2, x1^5)'))
alg.decomponi()
x = alg.v
I = ideal(x[0]**4, x[1]**5)
J = ideal(x[0]**3 + x[0] + 1, x[1]**3 + x[1] + 1)
L = I.intersection(J)
alg.decomponi(L)
alg.fattorizza()
alg.print_info()
```

La seconda chiamata della funzione `alg.decomponi(L)` chiama la funzione `alg.clear_ideal()` che stampa le informazioni del primo test sul file di output e cancella tutti gli attributi, tranne `F`, `q`, `n`, `v` e quelli riguardanti i file di output. Quindi viene chiamata la funzione `alg.build_algebra(L)` che permette di costruire l'algebra R/L e di decomporla.

```
Algebra A = F_9[x0, x1]/(x0^4 + x1^2, x1^5)
Dimensione del campo fissato dal Frobenius:1
Ricerca degli idempotenti: per elencazione
A= R/I: algebra locale
------------------------------------------------------------
------------------------------------------------------------

Algebra A = F_9[x0, x1]/intersection(I,J)
        I = (x0^4, x1^5) ,
        J = (x0^3 + x0 + 1, x1^3 + x1 + 1)
Dimensione del campo fissato dal Frobenius:10
10 idempotenti trovati
Ricerca degli idempotenti: random
Decomposizione minimale
A =A/J_1* ... *A/J_10
Fattorizzazione in ideali primari
Ideale I = I_1* ... *I_10
I_1 =(x0 - 1, x1 + (-z2 + 1))
I_2 =(x0 + (z2), x1 + (-z2 + 1))
I_3 =(x0 + (-z2 + 1), x1 + (-z2 + 1))
I_4 =(x0 - 1, x1 - 1)
I_5 =(x0 + (-z2 + 1), x1 + (z2))
I_6 =(x0 - 1, x1 + (z2))
I_7 =(x0^4, x1^5)
I_8 =(x0 + (-z2 + 1), x1 - 1)
I_9 =(x0 + (z2), x1 - 1)
I_10 =(x0 + (z2), x1 + (z2))
```

Tutte le altre funzioni presentate in 4, possono essere usate singolarmente per ottenere informazioni parziali: ad esempio con il seguente codice possiamo trovare il numero di ideali massimali in un'algebra A

```
alg = algebra_dec(info = [1,'F5N3dim'], ipt = [ 5, 3])
x = alg.v
K = ideal(x[0]**2+x[0]+1, x[1]**2+x[1] +1, x[2]**2+x[2]+1)
alg.build_algebra(K)
Aphi = alg.fixed_Frobenius()
alg.print_info()
```

ottenendo il seguente outuput:

```
Algebra A = F_5[x0, x1, x2]/(x0^2 + x0 + 1, x1^2 + x1 + 1, x2^2 + x2 + 1)
Dimensione del campo fissato dal Frobenius:4
Ricerca degli idempotenti: per elencazione
```

In fase di scrittura del codice la correttezza della decomposizione è stata verificata utilizzando la seguente procedura

- La prima verifica effettuata è $I = I_1 \cdots I_m$: questo garantisce, per il teorema cinese del resto che $A \simeq A/I_1 \times \cdots \times A/I_m$, per costruzione infatti gli idempotenti costruiti sono ortogonali e a somma 1.

```
    if len(alg.idem)!=0:
        f = alg.fact
        J = alg.R.ideal(f[0])
        for fatt in f[1:len(f)]:
            K = alg.R.ideal(fatt)
            J = J*K
        print(J==alg.I)
```

- Per verificare che la decomposizione sia effettivmente minimale, ho usato la funzione `primary_decomposition()` già presente in Sage, verificando `alg.dim == len((alg.I.radical()).primary_decomposition())`. Infatti nel caso di ideali 0-dimensionali, la decomposizione primaria del radicale di I è data dagli ideali massimali sopra I.

Riferimenti bibliografici

[1] P. Gianni, V. Miller, and B. Trager, "Decomposition of algebras."

[2] Documentazione sagemath. [Online]. Available: https://doc.sagemath.org/html/en/constructions/index.html

[3] D. Cox, J. Little, and D. O. Shea, *Using Algebraic Geometry*. Springer, 1998.

[4] I. M.F.Atiyah, *Introduction to Commutative Algebra*. Addison-Weasley, 1969.

[5] S. Bosch, *Algebra: From the Viewpoint of Galois Theory*. Birkhäuser, 2018.

6 Codice Python

```
#!/usr/bin/env sage

import sys
import numpy as np
from sage.all import *
from datetime import datetime
import time

def refine(z, u):
    #raffina l'idempotente z con u
    y = z*u
    if (y==0) or (z-y == 0): return []
    return [y, z-y]

class algebra_dec():
    def __init__(self, info = None, ipt = None):
        self.info = 0
        if info != None: self.info = info[0]
        if self.info == 1:
            if len(info) >0: self.fileName = info[1]+'.txt'
            else: self.fileName = str(datetime.now().isoformat())+'.txt'
            self.file = open(self.fileName, 'w')
            (self.file).close()
        if len(ipt)>0:
            self.q = ipt[0]
            self.n = ipt[1]
        else:
            self.q = int(input("Dimensione del campo\n"))
            self.n = int(input("Numero di variabili\n"))
        self.F = GF(self.q)
        self.R = PolynomialRing(self.F, ['x%s' %i for i in range(0,self.n)], order = 'lex')
        self.v = [(self.R).gen(i) for i in range(0,self.n)]

    def build_algebra(self, *id):
        if len(id) == 0:
            self.I = self.convert_string(input("Inserire un ideale di F_%s[x0, _, x%s]\n" %(self.q,
            ↪  (self.n)-1)))
        else:
            self.I = id[0]
        self.A = (self.R).quotient(self.I, ['a%s' %i for i in range(0,self.n)])
        self.a = (self.A).gens()
        self.B = self.base_algebra()
        return

    def base_algebra(self):
        if (self.n==1) and (self.I == self.A(0)):
            print("Errore, ideale non zero-dimensionale")
            return
        if (self.n>1) and (self.I.dimension()>0):
            print("Errore, ideale non zero-dimensionale")
            return
        G = self.I.groebner_basis();
        if len(self.v)==1:
```

```
            d = G[0].degree();
            B = [self.v[0]**i for i in range(0, d)]
            return B
        return self.I.normal_basis()

    def in_base(self, w):
        if self.n == 1: return vector(self.A(w))
        wcoefs = []
        for i in range(0, len(self.B)):
            wcoefs.append(w.monomial_coefficient((self.B)[i]))
        return vector(wcoefs)

    def Frobenius_matrix(self):
        m = len(self.B)
        S = MatrixSpace(self.F, m, m);
        M = S()
        for i in range(0,m):
            #per ogni elemento della base calcolo la potenza q-esima
            el = (self.B)[i]
            w = el**(self.q)
            if self.n > 1: w = w.reduce(self.I.groebner_basis())
            #scrivo w in base
            M[i,:] = self.in_base(w)
        return M #trasposto della matrice di Frobenius

    def fixed_Frobenius(self):
        m = len(self.B)
        S = MatrixSpace(self.F, m, m)
        M = self.Frobenius_matrix()
        Aphi = (M-S(1)).kernel() #B.kernel computes the left kernel wB = 0
        V = Aphi.basis()
        self.dim = len(V)
        #prepara la ricerca degli idempotenti
        if (self.q)**self.dim < 700:
            self.tipoRic = 'Ricerca degli idempotenti: per elencazione\n'
            self.index = 0
        return Aphi

    def check_idempot(self, u):
        if self.n == 1: return ((self.A)(u*u-u) == 0)
        return ((u*u - u).reduce(self.I.groebner_basis()) == 0)

    def prossimo_idempot(self,V):
        d = self.dim
        S = (self.F)**d
        while 1:
            self.index = self.index +1
            i = S[self.index]
            ucoef = sum(i[j]*V[j] for j in range(0,d))
            #scrivo date le coordinate dell'elemento di R corrispondente
            u = sum(ucoef[j]*(self.B)[j] for j in range(0,len(self.B)))
            #print(u)
            if ((self.A)(u)!=0) and ((self.A)(u)!= 1):
                if (self.check_idempot(u) == 1): return self.A(u)

    def random_idempot_el(self, Aphi):
```

```
        d = self.dim
        p = list(factor(self.q))[0][0]; G = GF(p)
        n = list(factor(self.q))[0][1]
        start = time.time()
        now = time.time()
        while now-start <1:
            vcoef = Aphi.random_element()
            v = sum(vcoef[j]*(self.B)[j] for j in range(0,len(self.B)))
            #crea il candidato idempotente
            if p%2 == 0: #caso char 2
                u = sum(v**(p**i) for i in range(0, n))
            else: #caso char diversa da 2
                t = v**((self.q-1)/2)
                u = t*(t-1)/2
                if (u==0) or (u==1): u = t*(t+1)/2
            if ((self.A)(u)!=0) & ((self.A)(u)!= 1):
                if (self.check_idempot(u) == 1):
                    return self.A(u)
            now = time.time()
        print("Riprovare")
        return

    def find_idempot_el(self, Aphi):
        if hasattr(self, 'index'):
            return self.prossimo_idempot(Aphi.basis())
        else:
            self.tipoRic = 'Ricerca degli idempotenti: random\n'
            return self.random_idempot_el(Aphi)

    def sistem_idempot(self, Aphi):
        d = self.dim
            start = time.time()
            now = time.time()
            self.idem = []
            if d == 1:
                self.dec = -1
                return
        if hasattr(self, 'index'): self.index = 0
            while now-start<20:
                u = self.find_idempot_el(Aphi)
                if self.idem == []: self.idem = [u, 1-u]
                else:
                    new = []
                    for el in self.idem:
                        l = refine(el, u)
                        if (len(l)!=0):
                            self.idem.remove(el)
                            new.append(l[0])
                            new.append(l[1])
                    self.idem = self.idem + new
                    self.idem = list(set(self.idem))
                if len(self.idem) == d:
                    self.dec = 1
                        return
                now = time.time()
        self.dec = 0
```

```
165         print("Non sono stati trovati abbastanza idempotenti")
166         return
167 
168     def clear_ideal(self):
169         if not hasattr(self, 'I'): return
170         #cancella tutti gli attributi relativi all'algebra
171         if self.info == 1:
172             #se richeisto stampa le informazioni trovate sul file di output
173             self.print_info()
174         for el in list((self.__dict__).keys()):
175             if el not in ['q', 'n', 'F', 'R', 'v', 'info', 'fileName']:
176                 delattr(self, el)
177         return
178 
179     def decomponi(self, *id):
180         if not(hasattr(self, 'A')) or len(id)>0:
181             self.clear_ideal()
182             self.build_algebra(*id)
183         if not(hasattr(self, 'dec')):
184             Aphi = self.fixed_Frobenius()
185             self.sistem_idempot(Aphi)
186         if self.dec == -1:
187             print("Aphi is local")
188             return
189         self.qzn = []       #decomposizione minimale sse self.dec == 1
190         for el in self.idem:
191             I = (self.R).ideal((self.A(1)-el).lift())
192             gen = I.groebner_basis()
193             self.qzn.append([self.A(gen[i]) for i in range(0, len(gen))])
194         return
195 
196     def fattorizza(self, *id):
197         if not(hasattr(self, 'A')) or len(id)>0:
198             self.clear_ideal()
199             self.build_algebra(*id)
200             print('costructed')
201         if not(hasattr(self, 'dec')):
202             Aphi = self.fixed_Frobenius()
203             print('Aphi')
204             self.sistem_idempot(Aphi)
205             print('find-idem')
206         if self.dec == -1:
207             print("Potenza di un ideale massimale")
208             return
209         self.fact = []
210         for el in self.idem:
211             J = self.I + ideal((self.A(1)-el).lift())
212             gen = J.groebner_basis()
213             (self.fact).append(gen)
214         return
215 
216     def print_info(self):
217         if not hasattr(self, 'fileName'):
218             self.fileName = str(datetime.now().isoformat())+'.txt'
219         self.file = open(self.fileName, 'a')
220         q =str(self.q)
```

```
        v =str(self.v)
        I = str((self.I).gens())
        if self.n==1: I = '('+I[1: len(I)-2]+')'
        else: I = '('+I[1: len(I)-1]+')'
        (self.file).write('Algebra A = F_'+q+v+'/'+I+'\n')
        for el in self.__dict__.keys():
            if el == 'dim':
                (self.file).write('Dimensione del campo fissato dal Frobenius:' +
                ↪  str(self.dim)+'\n')
            if el == 'tipoRic':
                if self.tipoRic!= 1: (self.file).write(self.tipoRic)
            if el == 'idem':
                (self.file).write(str(len(self.idem)) + ' idempotenti trovati ')
                if (len(self.idem)!= 0) and (len(self.idem)<5):
                    if len(str(self.idem[0]))< 50:
                        (self.file).write(str(self.idem[0]))
                        for i in range(1, len(self.idem)): (self.file).write(','+str(self.idem[i]))
                (self.file).write('\n')
                if len(self.idem) == 2: string_dots = '*'
                if (len(self.idem)>2): string_dots ='* ... *'
            if el == 'dec':
                if self.dec == 0: decType ='non minimale\n'; idType = 'non necessariamente
                ↪  primari\n'
                if self.dec == 1: decType ='minimale\n'; idType = 'primari\n'
                if self.dec == -1:
                    (self.file).write('A= R/I: algebra locale\n')
                    (self.file).close()
                    return
            if el == 'qzn':
                (self.file).write('Decomposizione '+decType)
                (self.file).write('A =A/J_1'+ string_dots +'A/J_'+str(len(self.qzn))+'\n')
                for i in range(0,len(self.qzn)):
                    id = str(self.qzn[i])
                    if len(id)< 100: (self.file).write('J_'+str(i+1)+' ='+ '('+id[1:len(id)-1]+')'
                    ↪  +'\n')
            if el == 'fact':
                (self.file).write('Fattorizzazione in ideali '+idType)
                (self.file).write('Ideale I = I_1'+string_dots +'I_'+str(len(self.idem)) + '\n')
                for i in range(0,len(self.fact)):
                    id = str(self.fact[i])
                    (self.file).write('I_'+str(i+1)+' ='+ '('+id[1:len(id)-1]+')' +'\n')
        (self.file).close()
        return


    def convert_string(self,string):
        if string[0] == '(': string = string[1:len(string)]
        if string[len(string)-1] == ')': string = string[0:len(string)-1]
        I = []
        while ',' in string:
            i = string.index(',')
            I.append(self.R(string[0:i]))
            string = string[i+ 1: len(string)]
        I.append(self.R(string))
        return ideal(I)
```