Projet de calcul Formel
Licence d'informatique

Polynômes multivariés : bases d'idéals

Le but de ce projet est d'implanter une version simplifiée d'un algorithme dû à Buchberger pour calculer une base réduite d'un idéal de polynômes.

Le projet sera rendu en `OCaml`, on rédigera également un rapport présentant fonctionnalités. Les algorithmes devront être expliqués, avec si possible une étude de complexité. Toutes les fonctions de ce projet devront être commentées. Il est également conseillé de garder au moins en commentaire les traces des tests effectués.

Le projet sera effectué seul ou en binôme. Le projet est à rendre pour le ? ? ? ? ? ? ? ? ?.

Introduction

Dans ce projet, on s'intéresse à la résolution des systèmes d'équations polynomiales en plusieurs variables, dans le cadre du calcul exact (sans approximations). Malheureusement, on sait depuis Gauss qu'il n'est pas possible, en général, de donner une formule close pour les solutions d'un polynôme de degré supérieur à 5. On va donc chercher à exprimer les solutions d'un système d'équations comme les racines d'un polynôme le plus simple possible. Par exemple, le système

$$\left\{\begin{array}{r} x^4 - 7\,x^3 - 7\,x^2 + 19\,x - 6 = 0 \\ x^3 - 6\,x^2 - 9\,x + 14 = 0 \end{array}\right. \iff \left\{\begin{array}{r} (x-1)(x+2)(x^2 - 8\,x + 3) = 0 \\ (x-1)(x+2)(x-7) = 0 \end{array}\right. \tag{1}$$

a pour seules solutions 1 et -2. Il a donc les mêmes solutions que le polynôme

$$x^2 + x - 2 = (x-1)(x+2)\,. \tag{2}$$

Ainsi, dans le cas des polynômes à une seule variable, pour résoudre un système d'équations, il suffit de savoir calculer le plus grand commun diviseur. En effet, si x_0 est une racine d'une famille de polynôme $B_1, \ldots, B_n$, alors le polynôme $X - x_0$ divise chacun des B_i, et donc il divise le PGCD de $B_1, \ldots, B_n$. Pour résoudre un système d'équations, on calcule donc le PGCD des polynômes que l'on résoud ensuite.

En fait le PGCD répond au problème plus général suivant :

Problème 1. ***(Racines Contenues – RC)***
Soit $S := (B_1, \ldots B_n)$ une famille de polynômes vu comme un système d'équations et A un autre polynôme. On cherche un algorithme rapide pour déterminer si les solutions du polynôme A contiennent toutes les solutions du système S.

En effet, les solutions du polynôme A contiennent toutes les solutions du système S si et seulement si A est divisible par le PGCD de S. Ainsi, pour résoudre le problème (RC), on calcule le PGCD P de S et on teste si le reste de la division de A par P est nul.

Malheureusement, en plusieurs variables, les choses ne se passent pas aussi bien. . .Il faut utiliser un algorithme plus sophistiqué dû à Buchberger. Le but de ce projet est d'en implanter une version légèrement simplifié.

1 Polynômes multivariés

Le but de cette première partie est d'implanter une bibliothèque de fonctions de calculs exacts sur des polynômes en plusieurs variables à coefficients rationnels.

Pour les coefficients, on utilisera soit les nombres rationnels écrit pendant les travaux dirigées soit la bibliothèque `Nums` de `OCaml`. Dans ce cas, on rappelle que pour pouvoir utiliser cette bibliothèque il faut d'abord l'avoir chargée dans `OCaml` par exemple avec la commande `#load "nums.cma";;`. On prendra également garde au fait que le type `Nums.nums` est un type **abstrait**. En conséquence, par défaut, `OCaml` ne sait pas afficher un `num`, il faut utiliser la commande `#install_printer` *prn*. De plus, **on ne peut pas comparer ou tester l'égalité** de deux `Nums.nums` avec les opérateurs usuels (`=`, `<`, ...), il faut utiliser ceux de la bibliothèque `Nums`. Enfin, lors de la phase de test, pour pouvoir taper plus facilement les nombres rationnels, on conseille d'utiliser la fonction `Nums.string_of_num` à laquelle on aura donné un nom court, par exemple `num`.

1.1 Monômes, termes et ordres

Définition 1. *Soit $x, y, z \dots$ des inconnues. On appelle* ***terme*** *toute expression de la forme $x^a y^b z^c \dots$ On appelle* ***monôme*** *le produit d'un terme par un coefficient. Un* ***polynôme*** *est une somme de monômes.*

Pour implanter ces définitions en `OCaml`, on procédera de la manière suivante :

- La liste des variables sera fixé un fois pour toutes sous la forme d'une liste de caractères (type `char`) nommée `variables`. On appellera `n_variables` le nombre de variables.
- Un terme sera représenté par une liste d'entiers de même longueur que la liste des variables : on notera la liste des exposants dans l'ordre des variables.
- Un monôme sera représenté par un couple (t, c) où t est un terme et c un coefficient.
- Enfin un polynôme sera représenté par une liste de monômes.

Par exemple, pour les variables x, y, z, les termes x^3yx^5 et x^2z seront respectivement représentés par les listes `[3; 1; 5]` et `[2; 0; 1]`, alors que le monôme $2x^3yx^5$ sera représenté par le couple `([3; 1; 5], 2)`.

▸ **Exercice 1.**

1. Déclarer les types `coeff_t`, `term_t`, `monom_t` et `poly_t` correspondant respectivement aux coefficients, termes, monômes et polynômes.

Dans la suite, pour avoir une représentation normalisée d'un polynôme, il faut fixer un ordre sur les termes. On utilisera l'ordre lexicographique ou LEX en abrégé. C'est-à-dire que pour comparer deux termes, on compare les exposants de x ; s'il sont égaux, on compare les exposants de y, et ainsi de suite. Nous donnons ici quelques termes triés dans l'ordre :

$$1 < z < z^2 < z^3 < y < yz < yz^2 < y^2 < y^2z < x < xz < xy < xyz < xyz^2 < xy^2 < x^2 \quad (3)$$

Les fonctions de manipulation de monômes supposent que les monômes sont des listes de même longueur que la liste `variables`. Toutefois, pour pouvoir plus aisément localiser les erreurs dans les programmes, on renverra une exception `Bad_monom` si ce n'est pas le cas.

2. Écrire une fonction `lex_compare` qui compare deux monômes pour l'ordre LEX en renvoyant -1, 0 ou 1 selon que le premier monôme passé en paramètre est inférieur, égal ou supérieur au deuxième.
3. Écrire une fonction `mult_term` qui multiplie deux termes.
4. Écrire une fonction `divide_term` qui divise un terme par un autre et lève une exception `Divide` si la division n'est pas possible. Par exemple, la division du terme $x^5y^2z^3$ par $x^2y^2z^2$ devra renvoyer le terme x^3z, alors que la division de $x^2y^2z^2$ par x^3 est impossible.
5. Écrire une fonction `print_term` qui affiche un terme.

1.2 Polynômes en plusieurs variables

Il est utile qu'un polynôme ait une et une seule représentation en mémoire possible. Cette représentation est dite **normalisée**. On utilisera la normalisation suivante :

- un polynôme est représenté par une liste de monômes triée dans l'ordre **lexicographique décroissant** des termes ;
- un terme ne peut apparaître qu'une fois au plus dans la liste ;
- aucuns des monômes apparaissant dans la liste n'a de coefficient nul.

Ainsi, le polynôme $x^2yz + 2/3\,x^3z^2$ sera représenté par la liste

```
[([3; 0; 2], num "2/3"); ([2; 1; 1], num "1") ]
```

où `num` "x" représente le rationnel x. Le polynôme constant égal à 1 sera donc représenté par la liste `[[0; 0; 0], num "1"]`.

Toutes les fonctions de manipulation de polynômes demandées dans l'énoncé renverront leurs résultats sous la forme de listes normalisées. Elle pourront également supposer que les polynômes passés comme paramètre le sont. Si ce n'est pas le cas, le comportement est non spécifié, il peut toutefois être utile, quand on peut facilement détecter une erreur, de renvoyer une exception pour afin de signaler les erreurs le plus tôt possible.

Définition 2. *Soit P un polynôme non nul.*

- *On appelle **terme dominant** de P le premier terme de P sous forme normalisée. C'est donc le terme le plus grand dans l'ordre LEX.*
- *Le coefficient de ce terme est appelé **coefficient dominant**.*
- *On dit qu'un polynôme est **monique** si son coefficient dominant est* 1.

▸ **Exercice 2.** On demande d'écrire les fonctions suivantes :

6. `of_coeff`, `of_char` qui créent un polynôme respectivement à partir d'un coefficient, d'un caractère qui représente une variable ;
7. `lterm`, `lcoeff` qui retournent le terme et le coefficient dominant (leading en anglais) d'un polynôme ;
8. `print_poly` qui affiche un polynôme ;
9. `is_zero_poly` et `equal_poly` qui testent la nullité et l'égalité de deux polynômes ;
10. `mult_coeff` et `mult_poly_term` qui multiplient un polynôme respectivement par un coefficient et par un terme ;
11. `add_poly`, `sub_poly`, `mult_poly` qui respectivement additionnent, soustraient et multiplient deux polynômes ;
12. Pour la suite, il peut être pratique d'utiliser des opérateurs pour ces fonctions. On pourra par exemple utiliser `=^`, `+^`, `-^` et `*^`.

On prendra bien soin de **tester exhaustivement** toutes les fonctions de la première partie avant de passer à la deuxième...

2 Idéaux et bases

L'un des algorithmes les plus important des la manipulation des polynômes en une variable est l'algorithme d'Euclide qui permet de calculer le plus grand diviseur commun d'une famille de polynômes. L'algorithme de Buchberger en est un analogue en plusieurs variables. La fonction principale du PGCD est de pouvoir tester par une division si un polynôme est une combinaison d'autres polynômes : soit $B_1, \ldots, B_n$ et A des polynômes. Alors le PGCD de $B_1, \ldots, B_n$ divise A si et seulement si il existe des polynômes $Q_1, \ldots, Q_n$ tels que

$$Q_1(X)B_1(X) + Q_2(X)B_2(X) + \cdots + Q_n(X)B_n(X) = A(X)\,. \tag{4}$$

Il est clair que, dans ce cas, si x_0 est une solution de tous les B_i alors il est une solution A. Dans le cas de plusieurs variables, on ne pourra pas toujours aboutir à un seul polynôme ; la solution sera donnée sous la forme d'un systèmes d'équations plus simple que celui de départ. En particulier, la réponse au problème "Racines Contenues" pourra être calculée au moyen d'un algorithme de division simple.

On commence par présenter l'analogue de la division euclidienne. On veut résoudre le problème suivant :

Problème 2. ***(Réduction (division) en plusieurs variables – RED)***

Soit la donnée d'une famille de polynômes $P = B_1, \ldots, B_n$ et d'un polynôme A. On cherche à calculer un polynôme R (l'équivalent du reste de la division), tel que

$$A = Q_1B_1 + Q_2B_2 + \cdots + Q_nB_n + R \tag{5}$$

et tel que aucun terme de R n'est divisible par le terme dominant de l'un des des B_i. On dit que R est un polynôme ***réduit*** *de A par les B_i.*

Dans la suite, on ne cherchera pas à calculer les quotients Q_i. Avant de décrire formellement l'algorithme, nous commençons par quelques exemples.

Exemple 1. Soit $B_1 = xy + 1$, $B_2 = y + 1$ et $A = xy^2 + 1$.

$$\begin{array}{r|cc}
 & xy+1 & y+1 \\ \hline
xy^2+1 & y & \\
-(xy^2+y) & & \\ \hline
-y+1 & & -1 \\
-(-y-1) & & \\ \hline
2 & &
\end{array}
\qquad
\begin{array}{r|cc}
 & xy+1 & y+1 \\ \hline
xy^2+1 & & xy \\
-(xy^2+xy) & & \\ \hline
-xy+1 & & -x \\
-(xy-x) & & \\ \hline
x+1 & &
\end{array} \tag{6}$$

$$xy^2 + 1 = y \cdot B_1 - 1 \cdot B_2 + 2 \qquad xy^2 + 1 = 0 \cdot B_1 + (xy - x) \cdot B_2 + (x + 1) \tag{7}$$

Dans le tableau de gauche, on fait une division comme dans le cas univarié, d'abord par rapport à B_1 puis par rapport à B_2. Les quotients des termes dominants sont écrits en dessous des diviseurs respectifs. Dans la dernière ligne le reste 2 n'est divisible, ni par B_1 ni par B_2. On remarque qu'il y a du choix, on aurait pu commencer comme dans le tableau de droite par diviser par B_2, et aucun des termes du reste $x + 1$ n'est divisible ni par B_1 ni par B_2.

Exemple 2. Soit maintenant $A = x^2y + xy^2 + y^2$, $B_1 = xy - 1$ et $B_2 = y^2 - 1$.

$$\begin{array}{r|ccc}
 & xy+1 & y+1 & \text{reste} \\
\hline
x^2y + xy^2 + y^2 & x & & \\
-(x^2y - x) & & & \\
\hline
xy^2 + x + y^2 & y & & \\
-(xy^2 - y) & & & \\
\hline
x + y^2 + y & & & x \\
-x & & & \\
\hline
y^2 + y & & 1 & \\
-(y^2 - 1) & & & \\
\hline
y + 1 & & &
\end{array} \tag{8}$$

On voit ici un phénomène qui n'a pas lieu dans le cas univarié. Lors de la troisième étape, le terme dominant x n'est divisible ni par B_1, ni par B_2. On le déplace alors dans la colonne reste. La division continue ensuite avec les termes restants. On trouve finalement

$$x^2y + xy^2 + y^2 = (x + y) \cdot B_1 + 1 \cdot B_2 + (x + y + 1)\,. \tag{9}$$

L'algorithme utilisé est le suivant

Algorithme 1. ***(Division avec reste en plusieurs variable)***
Entré : A et $B_1, \dots B_n$ des polynômes.
Sortie : R tels qu'il existe $Q_1, \dots, Q_n$ vérifiant

$$A = Q_1B_1 + Q_2B_2 + \cdots + Q_nB_n + R \tag{10}$$

et tel que le terme dominant du polynôme R soit plus petit pour l'ordre lexicographique que tous les termes dominants des B_i.

1. $r \leftarrow 0$
2. tant que $A \neq 0$ faire
3. si il existe un i tel que $\mathtt{lterm}(B_i)$ divise $\mathtt{lterm}(A)$
   alors chosir un tel i et faire $A \leftarrow A - \frac{\mathtt{lterm}(A)}{\mathtt{lterm}(B_i)} B_i$
   sinon $r \leftarrow r + \mathtt{lterm}(A)$ et $A \leftarrow A + \mathtt{lterm}(A)$
4. retourner R.

Une telle division n'est pas unique, il faut choisir un i à l'étape 3. Dans la suite, à chaque étape on choisira le plus petit i possible. On notera $A \ \mathrm{rem}\ (B_1, \dots B_n)$ le reste de la division.

▸ **Exercice 3.**

13. Écrire une fonction `reduce` qui implante l'algorithme 1. Pour accélérer les divisions, on pourra commencer par rendre les B_i moniques en les divisant par leurs coefficients dominants.

Pour la suite du projet, il n'est pas nécessaire de calculer les quotients Q_i. Cela dit, pour vérifier les calculs, il peut être utile d'écrire une deuxième fonction qui renvoie les quotients.

2.1 S-paires et S-polynômes

Malheureusement l'algorithme précédent ne résoud pas toujours correctement le problème RC comme on peut le voir dans l'exemple suivant :

Exemple 3. On divise $A = xy^2 - x$ par $B_1 = xy + 1$ et $B_2 = y^2 - 1$.

$$\begin{array}{r|ll} & xy+1 & y^2+1 \\ \hline xy^2 - x & y & \\ -(xy^2+y) & & \\ \hline -x-y & & \end{array} \qquad (11)$$

et donc $A = y \cdot B_1 + 0 \cdot B_2 + (-x - y)$. On trouve donc un reste non nul. Pourtant, si l'on avait commencé par diviser par B_2 on aurait trouvé $A = 0 \cdot B_1 + x \cdot B_2 + 0$. Ainsi toute solution commune à B_1 et B_2 est une solution de A. Pourtant, si, comme c'est le cas ici, on fait le mauvais choix, la seule division par (B_1, B_2) donne un reste non nul.

Définition 3. *Un ensemble de polynômes est appelé* ***base d'idéal*** *si, quelque soit l'ordre dans lequel on fait les divisions, le reste est toujours le même.*

Dans l'exemple précédent le problème vient du fait que

$$y \cdot B_1 - x \cdot B_2 = xy^2 + y - xy^2 + x = x + y \qquad (12)$$

mais que aucun des termes de $x + y$ n'est divisible ni par $\texttt{lterm}(B_1) = xy$ ni par $\texttt{lterm}(B_2)$. Une telle paire (B_1, B_2) de polynômes induit un choix dans l'algorithme des divisions.

Définition 4. *Soit A et B deux polynômes non nuls tel que*

$$\texttt{lterm}(A) = x^{a_1} y^{a_2} z^{a_3} \dots \qquad \textit{et} \qquad \texttt{lterm}(B) = x^{b_1} y^{b_2} z^{b_2} \dots . \qquad (13)$$

On pose $c_i = \max(a_i, b_i)$ pour tout i et $C := x^{c_1} y^{c_2} z^{c_3} \dots$. Le ***S-polynôme*** *du couple $\{A, B\}$ est défini par*

$$S(A,B) := \frac{C}{\texttt{lterm}(A)} A - \frac{C}{\texttt{lterm}(B)} B \qquad (14)$$

On a clairement $S(A, B) = -S(B, A)$. Par exemple, $S(xy+1, y^2-1) = x+y$. Le problème de la division vient du fait que le S-polynôme ne se réduit pas à zéro.

Le théorème suivant affirme que c'est effectivement là le seul problème :

Théorème 1. *Une famille de polynômes $B := B_1, \dots, B_n$ est une base si et seulement si pour toute paire $1 \leq i, j \leq n$, le reste de la division de $S(B_i, B_j)$ par B est nul, c'est-à-dire si*

$$S(B_i, B_j) \ \text{rem} \ (B_1, \dots, B_n) = 0 \qquad (15)$$

On calcule une base en ajoutant successivement à la liste les S-polynômes tant qu'il existe une paire i, j pour laquelle l'équation 15 n'est pas vérifiée.

Exemple 4. Soit $B_1 = xy + yz$ et $B_2 = xz + y$. On a alors

$$S(B_1, B_2) = z\, B_1 - y\, B_2 = yz^2 - y^2 = -y^2 + yz^2 \, . \qquad (16)$$

Ce polynôme ne peut se réduire ni par B_1 ni par B_2. On appelle donc B_3 le polynôme monique associé : $B_3 := y^2 - yz^2$. Alors, d'après le théorème précédent (B_1, B_2, B_3) est une base si et seulement si

$$S(B_1, B_3) \ \text{rem} \ (B_1, B_2, B_3) = S(B_2, B_3) \ \text{rem} \ (B_1, B_2, B_3) = 0 \, . \qquad (17)$$

On a alors $S(B_1, B_3) = xyz^2 + y^2 z$ et $S(B_2, B_3) = xyz^3 + y^3$. On réduit alors ces deux polynômes par rapport à (B_1, B_2, B_3). On trouve

$$xyz^2 + y^2 z = z^2 \cdot B_1 + 0 \cdot B_2 + z \cdot B^3 + 0 \qquad (18)$$

et donc $S(B_1, B_3) \ \text{rem} \ (B_1, B_2, B_3) = 0$. De même pour $S(B_2, B_3)$, on trouve

$$xyz^3 + y^3 = z^3 \cdot B_1 + 0 \cdot B_2 + (y - z^2) \cdot B^3 + 0 \qquad (19)$$

et donc $S(B_2, B_3) \ \text{rem} \ (B_1, B_2, B_3) = 0$. Ainsi (B_1, B_2, B_3) est la base cherché.

Voici un autre exemple :

Exemple 5. On part de $B_1 = x^2y^2 + yz$, et $B_2 = x^2z + xy$. On a alors $S(B_1, B_2) = -xy^3 + yz^2$. On pose $B_3 := xy^3 - yz^2$ le polynôme monique associé. On calcule alors

$$S(B_1, B_3) \text{ rem } (B_1, B_2, B_3) = xyz^2 + y^2z := B_4\,. \tag{20}$$

On appelle ce polynôme B_4. On aurait pu aussi calculer

$$S(B_2, B_3) \text{ rem } (B_1, B_2, B_3) = xyz^3 + y^2z^2 \tag{21}$$

et l'ajouter à la base, mais ce n'est pas utile car c'est $z \cdot B_4$ et donc

$$S(B_2, B_3) \text{ rem } (B_1, B_2, B_3, B_4) = 0 \tag{22}$$

Il faut maintenant calculer toutes les S-polynômes avec B_4. On trouve

$$S(B_1, B_4) \text{ rem } (B_1, B_2, B_3, B_4) = S(B_1, B_4) \text{ rem } (B_1, B_2, B_3, B_4) = 0 \tag{23}$$

En revanche

$$S(B_3, B_4) \text{ rem } (B_1, B_2, B_3, B_4) = y^4z + yz^4 =: B_5\,. \tag{24}$$

On vérifie maintenant que pour $1 \leq i, j \leq 5$

$$S(B_i, B_j) \text{ rem } (B_1, B_2, B_3, B_4, B_5) = 0\,. \tag{25}$$

De sorte que $(B_1, B_2, B_3, B_4, B_5)$ est une base.

Quelques remarques :

1. Il n'est pas évident que cet algorithme s'arrête. On montre que c'est effectivement le cas, entre autre en utilisant le fait que l'ordre LEX est bien fondé.
2. Selon la stratégie employée pour fabriquer les S-polynômes, on peut obtenir des bases différentes. Par exemple, on aurait pu rajouter $S(B_2, B_3)$ dans l'exemple précédent. Pour obtenir un résultat unique on réduit la base par rapport à elle-même comme suit.

Définition 5. *Une base (B_i) est dite réduite si et seulement si aucun des termes d'un des B_i n'est divisible par le terme dominant d'un autre B_i.*

En partant d'une base non réduite, on calcule la base réduite associée en réduisant tour à tour chaque B_i par rapport aux autres, et en supprimant les polynômes nuls ainsi obtenus.

Théorème 2. *Soit (P_i) une famille de polynômes. Il existe une unique base réduite (B_i) telle que les solutions du problème RS pour (P_i) soient les mêmes que pour (B_i). On peut calculer cette base par ajout des S-polynômes puis réduction.*

On écrira les fonctions suivantes :

14. `s_poly` qui calcule le S-polynôme d'une paire de polynômes ;
15. `basis` qui calcule une base par ajout des S-polynômes ; On pourra utiliser deux stratégies :
    - Soit on calcule tous les S-polynômes, on les ajoute à la base et on calcule les nouveaux S-polynômes ainsi obtenus, jusqu'à ce qu'il se réduise tous à zéro.
    - Soit on traite un seul S-polynôme à la fois, en gardant dans une liste les paires de polynômes (B_i, B_j) pour lesquels on doit encore calculer puis réduire le S-polynôme. Cette deuxième méthode est beaucoup plus rapide, car il est très coûteux de réduire un polynôme.
16. `reduce_basis` qui réduit une base par rapport à elle-même.

3 Quelques applications

Les applications suivantes permettront de tester le bon fonctionnement du programme. L'une des difficultés avec cet algorithme est qu'il est, dans certains cas, exponentiel. On conseille donc d'afficher les progrès lors du calcul pour savoir si le programme est parti dans une boucle.

- Dans le cas des polynômes en une seule variable, on doit retrouver le PGCD. Par exemple

  ```
  # pl1,pl2;;
    - : poly_t * poly_t =
  (x^6 - x^5 + x^4 - x^3 + x^2 - 1, x^4 + x^3 + 2x^2 - 3x - 1)
  #  reduce_basis (basis [pl1;pl2]);;
  - : poly_t list = [x - 1]
  ```

  Le pgcd est $x - 1$.

- Si l'on donne un système qui n'a pas de solutions, la base retournée est alors formée du seul polynôme constant égal à 1.

  ```
  # pl1,pl2,pl3;;
  - : poly_t * poly_t * poly_t = (x - 1, y - 3, xy - 1)
  #  reduce_basis (basis [pl1;pl2;pl3]);;
  - : poly_t list = [1]
  ```

- Si l'on ne donne que des polynômes linéaires (pas de variable au carré ni de produits de variables), l'algorithme est alors le même que celui du pivot de Gauss. On peut donc résoudre les systèmes linéaires.

  ```
  # pl1,pl2,pl3;;
    - : poly_t * poly_t * poly_t =
  (x - y - 3, x - 2z + 4, x + 2y - 3z + 3)
  #  reduce_basis (basis [pl1;pl2;pl3]);;
  - : poly_t list = [x - 6; y - 3; z - 5]
  ```

  Il y a une solution $x = 6, y = 3, z = 5$.

- On peut aussi utiliser cet algorithme pour faire des calculs géométriques. Par exemple la sphère de centre $(2, 0, 0)$ et de rayon 2 a comme équation $(x - 2)^2 + y^2 + z^2 = 4$, soit $x^2 - 4x + y^2 + z^2 = 0$. Celle de centre $(0, 0, 0)$ et de rayon $\sqrt{6}$ a comme équation $x^2 + y^2 + z^2 - 6 = 0$. Si l'on donne ces deux équations à l'algorithme

  ```
  # pl1,pl2;;
  - : poly_t * poly_t = (x^2 + -4x + y^2 + z^2, x^2 + y^2 + z^2 + -6)
  #  reduce_basis (basis [pl1;pl2]);;
  - : poly_t list = [x - 3/2; y^2 + z^2 - 15/4]
  ```

  L'intersection est donc contenue dans le plan $x = 3/2$ et c'est le cercle de centre $(3/2, 0, 0)$ et de rayon $\sqrt{15}/2$.

- Par définition si $B = (B_i)$ est la base associée à un système d'équations (P_i) alors un polynôme A se réduit à zéro modulo B si et seulement si il est une combinaison $A_1P_1 + A_2P_2 + \cdots$ où les A_i sont des polynômes. On peut utiliser cette caractérisation pour vérifier la base.