Programmation

Introduction

Pour terminer cette étude du pgcd, il nous a été demandé de mettre en œuvre les algorithmes présentés dans ce document. Comme on l'a vu dans le paragraphe (XZXZ : PCA mettre un renvoi vers chapitre présentant XCAS), les logiciels de calculs formels savent très bien réalisés des calculs de pgcd, ils le font vite et bien sur un grand type de données : entiers, polynômes, entiers modulo n, …
Notre but n'était pas de concurrencer ces logiciels spécialisés mais de montrer :

- Comment ces algorithmes peuvent facilement être mis en œuvre dans des environnements différents.
- L'efficacité et la généricité de l'algorithme d'Euclide qui plus de 2000 ans après sa création, permet sans aucune modification notable, de calculer des pgcds dans des structures algébriques très éloignées de celles utilisées par l'auteur des éléments.

Nos allons commencer par utiliser le tableur de la suite bureautique LibreOffice. Ce dernier a un avantage, il est présent sur un grand nombre d'ordinateurs personnels, ses formule et son langage de macro permettent de mettre facilement en œuvre les algorithmes de calcul du pgcd. Cette simplicité de programmation a un inconvénient, il serait fastidieux de vouloir effectuer ces calculs sur autre chose que des entiers.

Pour aller plus loin et montrer que les algorithmes d'Euclide permettent d'effectuer des calculs de pgcds dans d'autres anneaux dont celui des polynômes, nous avons choisi d'implémenter ces derniers à l'aide du langage de programmation Python aidé de la bibliothèque de calcul formel sympy.

Utilisation d'un tableur

Décomposition en facteurs premiers

La partie théorique de cet algorithme est décrite au paragraphe (XZXZ : Pca mettre un lien vers paragraphe).
Cette décomposition est faite dans un but pédagogique, elle n'est pas exhaustive (elle teste seulement les nombres premiers compris entre 2 et 97) et le langage de macro de LibreOffice est très limité (il ne permet pas de réaliser une division euclidienne). Malgré ces limitations, la première feuille de notre tableur montre qu'il est effectivement possible de trouver le pcgd de deux nombres grâce à leur décomposition en facteurs premiers (comme décrit page XZXZ : Pca : Mettre renvoi vers la bonne page).
Le principe de cette macro est le suivant, la première partie est la décomposition de a et b en facteurs premiers :
Algorithme :

```
Pour chacun des nombres premiers de la liste faire
     Si le nombre premier divise a alors
          Puissance = 2
          Tant que a est divisible par le nombre premier exposant
               Puissance
               Puissance = Puissance + 1
          Fin tant que
          Mettre Puissance - 1 dans la cellule correspondant à la
```

```
                puissance du nombre premier traité
     Fin si
Fin pour
```

Explications :
Pour connaître l'exposant de chacun des nombres premiers dans la décomposition du nombre a. On commence par regarder si a est un multiple de ce nombre premier, si c'est le cas, on teste s'il est aussi un multiple du carré de ce nombre, puis du cube ainsi de suite jusqu'à trouver la puissance correspondante. Une fois cette puissance trouvée, on l'écrit dans la cellule correspondante.
On fait de même pour le nombre b.

C'est à cause de la boucle « tant que » que cette décomposition n'est pas faisable directement à l'aide de formules dans une feuille de calcul, mais nécessité une macro.
Seconde partie calcul du pgcd :
Algorithme

```
pgcd = 1
Pour chacun des nombres premiers de la liste faire
     Si (l'exposant du nombre premier dans la décomposition de a
          est différent de 0) et  (l'exposant du nombre premier
          dans la décomposition de b est différent de 0) alors
               puissance = min(exposant dans la décomposition de a,
                    exposant dans la décomposition de b)
               pgcg = pgcd * nombre premier exposant puissance
     Fin Si
Fin Pour
Mettre le pcgd dans la cellule qui lui est dédiée.
```

Explications :
On vient de calculer les exposants des nombres premiers, le calcul du pgcd est maintenant très simple. On initialise le pgcd à 1 et pour chaque nombre premier de la liste, on compare l'exposant de ce nombre dans la décomposition de a à celui dans la décomposition de b et on prend le plus petit des deux. On met à jour le pgcd en multipliant ce dernier par le nombre premier exposant la puissance trouvée.
Mise en œuvre :
Les algorithmes décrits ci-dessus sont des versions simplifiées des algorithmes implémentés, la mise en œuvre de ces derniers demande d'ajouter du code pour « pointer » vers les cellules de la feuilles de calculs, ainsi que pour effacer les cellules avant de les réutiliser lors d'un nouvel usage de la macro et d'autres actions sans relation directe avec l'algorithme. Ce code supplémentaire n'est pas détaillé dans ce document, il est décrit dans les commentaires de la macro reportée ci-dessous.
Code :
(XZXZ : Mettre le code de la macro).

Algorithme des différences

La partie théorique de cet algorithme est décrite au paragraphe (XZXZ : Pca mettre un lien vers paragraphe).

Cet algorithme utilise uniquement des soustractions, il est donc possible de le programmer dans une feuille de calcul grâce à des formules LibreOffice sans utiliser de macros.
Le codage est très simple, il s'agit juste de s'assurer que toutes les différences seront faites sans jamais quitter l'ensemble des entiers naturels.

Pour faciliter la lecture, les données des opérations successives sont rangées dans trois colonnes, toujours de la même manière.

La première colonne accueille le plus grand des deux nombres en entrée de la ligne de calcul. Le plus petit sera placé dans la seconde colonne et la troisième servira à stocker le résultat de la soustraction. Comme les deux entrées ont été classées par ordre croissant, on effectue toujours la différence entre la première et la seconde colonne (sans avoir à se soucier du signe).

Pour initialiser l'algorithme, on place dans la première ligne, première colonne le max de a, b et dans la deuxième colonne le min de a, b. La formule insérée dans la troisième colonne calcule directement la différence.

Les autres lignes sont toutes construites sur le même principe, on regarde dans la ligne précédente le nombre en seconde colonne et le résultat de la soustraction. Le plus grand des deux est placé en première colonne, le plus petit en seconde colonne. La soustraction est calculée automatiquement, ce qui déclenche le traitement de la ligne suivante, ainsi les opérations s'enchaînent jusqu'au pgcd.

Mise en œuvre : les formules présentent dans les cellules de la feuille de calcul ont été améliorées pour permettre d'afficher seulement le contenu des cellules « utiles ».

XZXZ : Mettre copies écran avec formules.

Algorithme d'Euclide

La partie théorique de cet algorithme est décrite au paragraphe (XZXZ : Pca mettre un lien vers paragraphe).

Il s'agit maintenant de faire des divisions euclidienne, à la place des « soustraction ». Pour pouvoir facilement comparer l'efficacité de l'algorithme des différences et celui d'Euclide, ces deux algorithmes ont été codés sur la même feuille.

Pour plus de clarté, les données de chaque ligne sont placées par colonne de la manière suivante :
La première contient le nombre que l'on doit diviser
Dans la seconde, on place le diviseur
Dans la troisième, une formule calcule le quotient de la division.
Dans la quatrième, une formule calcule le reste de la division.

Le quotient et le reste sont d'une ligne sont respectivement recopiés dans la première et seconde colonne de la ligne suivante, ce qui déclenche le calcul de la division euclidienne et la recopie du nouveau quotient et reste ligne suivante, jusqu'à ce que le reste soit nul.

Le pgcd étant le dernier reste non nul, il se trouve en dernière colonne sur l'avant dernière ligne.

Cet algorithme a une particularité, on a pas besoin de l'initialiser avec a supérieur à b. En effet, si on effectue la division euclidienne de a par b avec b plus grand que a. On obtient alors : $a = b*0 + a$. A l'étape suivante on aura donc à diviser b par le reste de la première division soit a. Et tout sera rentrer dans l'ordre.

XZXZ : Mettre copies écran avec formules + copie écran pour faire ressortir efficacité algo euclide + copie dans cas entiers de fibonacci.

Algorithme d'Euclide étendue

Comme son nom l'indique, il s'agit d'une extension de l'algorithme précédent, décrite au paragraphe (XZXZ : Pca mettre un lien vers paragraphe). Pour mieux illustrer le fonctionnement de cet algorithme nous avons choisi de détailler sa présentation. Ainsi, notre feuille de calcul se divise en

trois grandes parties :
Celle de gauche reprend les divisions euclidienne à la base de l'algorithme d'Euclide
Celle de droite donne les coefficients de Bezout, calculés comme indiqué (XZXZ : Pca mettre un lien vers paragraphe partie théorique).
Au milieu, se trouvent des calculs intermédiaires permettant de mieux comprendre comment sont calculés les deux coefficients de Bezout.

Les colonnes de droite donnent les coefficients de Bezout intermédiaires, c'est à dire qu'à chaque étape, on écrit r_k le reste de la division euclidienne en fonction de a et b.
Si on se penche sur les colonnes centrales, on retrouve le mécanisme décrit dans la fabrication des suites $(u_k)_{k \le 0}$ et $(v_k)_{k \le 0}$ au paragraphe (XZXZ mettre renvoi). Suite à la division euclidienne on a, $r_k = q_{k-1} – q_k * r_{k-1}$ or $q_{k-1} = r_{k-2}$ et comme dans les lignes précédentes on a écrit r_{k-2} et $r_{k-1}en fonction de a et b. Il est très facile de calculer le nouveau reste en fonction de ces deux nombres.

L'initialisation de cet algorithme est légèrement différente du précédent car les deux premières lignes donnent la décomposition (triviale) de a et b en fonction de a et b. Les divisions euclidiennes commencent à partir de la troisième ligne, le calcul des coefficients de Bezout s'effectue de manière normale. C'est à dire en utilisant les données des deux lignes supérieures que l'on vient d'initialiser.

Comme d'habitude, l'insertion des nombres a et b dans le tableau déclenche un effet domino qui va aboutir au pgcd.

Utilisation d'un langage objet et d'une bibliothèque de calcul formel

Grâce à sa mise en œuvre sur tableur, on a déjà montré l'efficacité de l'algorithme d'Euclide. Il nous faut maintenant montré sa généricité en l'utilisant pour calculer des pgcds dans des anneaux autres qui celui des entiers.

D'un point de vue informatique, quand on pense à des structures abstraites pouvant prendre différentes formes, on ne peut s'empêcher de faire un parallèle avec la programmation objet et en particulier avec l'héritage. (Ces concepts de programmation sont expliquer dans le livre XZXZ : PCA : Trouver un livre expliquant ceci).

Les similitudes entre le concept d'anneau euclidienne et les possibilités de la programmation orientée objet sont trop fortes pour ne pas profiter de la seconde pour mettre en œuvre la première.

C'est ce que nous allons faire autant pour prouver la généricité de l'algorithme d'Euclide, que pour revenir à la base de ce qu'est un anneau euclidien. On va ainsi montrer comment on part des opérations possibles au sein de cette structure algébrique et on les assemble pour calculer un pgcd.

Comment allons nous procéder ?

En utilisant un langage objet comme on souhaite le faire, on va aboutir à un programme plus générique que celui sur tableur. En contrepartie, on va amener un niveau d'abstraction plus élevé que l'utilisation des formules LibreOffice.

Pour compenser cette difficulté, on va détailler de notre démarche. Voici les différentes étapes de la conception de notre programme :

-On va commencer par regarder les opération possibles dans un anneau euclidien. Comme on a déjà codé l'algorithme, on va directement compléter cette liste de manière à avoir tous les outils nécessaire au calcul du pgcd et des coefficients de Bezout. Pour ne sortir de la structure d'un anneau, on indiquera pour chaque l'opération rajoutée, pourquoi elle est réalisable dans un anneau euclidien.

-On va ensuite se concentrer sur les deux algorithmes et voir comment les mettre en place en utilisant seulement les opérations trouvées précédemment. C'est à dire sans tenir compte d'une quelconque contrainte sur les types de données en entrées de ces algorithmes. On va ainsi faire ressortir le lien très fort entre l'algorithme d'Euclide et les anneaux euclidiens.

-Arriver à ce point on disposera d'une structure d'anneau euclidien générique et d'algorithme de calcul du pgcd. Il nous faudra donc implémenter cette structure abstraite en quelque chose de connu et d'utilisable. On commencera par le plus simple l'anneau des entiers relatifs.

-On peut maintenant, commencer à montrer la généricité de l'algorithme d'Euclide en travaillant dans d'autres anneaux. On va monter progressivement en complexité, en travaillant dans l'anneau Z/nZ.

-Pour terminer, on va coder l'anneau des polynômes à coefficients complexes. C'est là que la bibliothèque sympy intervient pour réaliser les calculs hors de portée du langage de programmation.

-Après avoir calculé des pgcds et coefficients de Bezout dans trois anneaux différents, on passera à la conclusion de cette section.

Opérations dans un anneau euclidien

Comme expliqué au paragraphe XZXZ, la structure d'anneau nous offre :

-Une addition et une multiplication : Ce sont les deux lois disponibles dans un anneau, elles sont à la base de toute opération et bien sûr indispensables au calcul du pgcd. Et comme il s'agit de lois internes, on est sûr de pouvoir les utiliser sans sortir de l'anneau.

-Une soustraction : Soustraire a revient à ajouter $-a$, le symétrique de a qui existe de par la structure de groupe formée par l'ensemble des éléments de l'anneau muni de l'addition. Pour faciliter le calcul du pgcd, on va directement introduire une fonction soustraction.

-La division euclidienne : D'un point de vue théorique, elle ne fait pas partie des opérations présentes dans un anneau générique mais seulement dans les anneaux euclidiens. Comme notre travail porte sur l'étude du pgcd, tous les anneaux dans lesquels on travaille sont euclidiens. D'un point de vue pratique, si on voulait être très rigoureux, il est possible de définir cette opération à l'aide d'une suite de soustractions (elles mêmes en lien avec l'addition). On démontre ainsi le lien entre cette opération et celles disponibles dans un anneau et on peut alors considérer que la division euclidienne fait partie des opérations possibles dans un anneau. Les langages de programmation et les bibliothèques mathématiques offrent des fonctions plus simples pour effectuer une division euclidienne qu'une suite de soustractions. On va donc utiliser les fonctionnalités qui nous sont offertes.

-Un élément neutre pour l'addition : Il est présent dans chaque anneau, il est par convention noté 0, mais le 0 de l'anneau des polynômes est différents du 0 de l'anneau des entiers. Pour aboutir à un programme générique une fonction sera chargée de renvoyer l'élément neutre pour l'addition dans l'anneau dans lequel on effectue nos calculs.

-Un élément neutre la multiplication : Idem que pour l'addition, une fonction nous retournera l'élément neutre pour la multiplication pour l'anneau en cours d'utilisation.

Les fonctions suivantes ne sont pas basées sur les opérations disponibles dans un anneau mais sont indispensables au calcul du pgcd :

-Une comparaison : L'algorithme de calcul du pgcd s'arrête quand le reste est égal à 0. Il nous faut donc pouvoir tester l'égalité du reste avec l'élément neutre de l'addition. On va donc introduire un test d'égalité et d'inégalité qu'il est toujours possible de définir dans un anneau.

-Simplifications des pgcds : dans le cas des polynômes, le pgcd est le polynôme unitaire décrit au paragraphe (XZXZ mettre renvoi). Il nous faut donc disposer d'une fonction de simplification qui sera appelée pour simplifier le pgcd en fin de calcul.

Avec ces fonctions on est capable de calculer un pgcd dans un anneau, on va donc maintenant voir comment les assembler pour arriver à ce résultat.

Calcul du pgcd par l'algorithme d'Euclide

On l'a déjà vu le principe de cet algorithme repose sur une suite de divisions euclidiennes.

Le coeur de cette fonction est une simple boucle « tant que » qui va effectuer les divisions successives. Le tout est d'implémenter celle-ci de façon à obtenir le bon résultat, tout en conservant la généricité de notre algorithme.

Le fonctionnement de cette boucle reprend le principe de l'algorithme d'Euclide. Voici son fonctionnement :

```
Tant que (reste différent de élément_neutre_addition)
     q, r = division_euclidienne(quotient, reste)
     quotient = reste
     reste = r
Fin tant que
```

On commence par faire un test d'égalité entre la variable « reste » et l'élément nul (i.e. l'élément neutre de la première loi). Si « reste » n'est pas nul, on lance la division euclidienne entre « quotient » et « reste ». Met à jour les variables « quotient » et « reste » en fonction du résultat de la division euclidienne. Et recommence le test d'égalité de « reste » avec l'élément nul et tant que reste n'est pas nul, on continue la série de divisions euclidiennes. Quand le sera nul, on sortira de la boucle.

Cette boucle seule ne suffit pas à calculer le pgcd de deux éléments d'un anneau. Voici le l'algorithme complet de la fonction chargée de ce calcul :

```
fonction algorithmeEuclide(A, B)
quotient = A
reste = B
Tant que (reste différent de element_neutre_addition)
     q, r = division_euclidienne(quotient, reste)
     quotient = reste
     reste = r
Fin tant que
pgcd = simplifier_pgcd(quotient)
retourner pgcd
fin fonction
```

Deux lignes ont été introduites avant la boucle, elles vont initialiser les variables « reste » et « quotient » avec le entrées « A » et « B ». Ainsi la boucle peut commencer ses itérations jusqu'à atteindre un reste nul, qui provoquera la sortie de la boucle. La fonction « simplifier_pgcd » n'est utile que si l'on travaille avec des polynômes pour mettre le pgcd sous forme unitaire. Comme on souhaite aboutir à un programme générique cette fonction est toujours appelée, dans le cas des polynômes, elle mettra le pgcd sous forme unitaire. Dans les autres cas, elle renverra la variable qu'on lui passera en paramètre.

Algorithme d'Euclide étendu

Cet algorithme fonctionne de la manière suivante :

```
fonction algorithmeEuclideEtendu(A, B)
quotient = A
reste = B
- > oldCoefA =  element_neutre_multiplication
- > oldCoefB =  element_neutre_addition
- > coefA =  element_neutre_addition
- > coefB = element_neutre_multiplication
Tant que (reste différent de élément_neutre_addition)
     q, r = division_euclidienne(quotient, reste)
     - > tmpCoefA = oldCoefA - q * coefA
```

```
    - > tmpCoefB = oldCoefB - q * coefB

    - > oldCoefA = coefA

    - > oldCoefB = coefB

    - > coefA = tmpCoefA

    - > coefB = tmpCoefB

    quotient = reste
    reste = r
Fin tant que
- > pgcd, oldCoefA, oldCoefB =
simplifier_pgcd_and_coeffs(quotient, oldCoefA, oldCoefB)
retourner pgcd
fin fonction
```

Il y a peu de différences entre l'algorithme d'Euclide et l'algorithme d'Euclide étendu. Huit lignes ont été rajoutées et une a été modifiée (elles sont marquées par le signe `- >`) .

Les lignes rajoutées se chargent de calculer les coefficients de Bezout (en conformité avec le paragraphe XZXZ), on les initialise avant d'entrer dans la boucle (toujours par souci de généricité, on utilise pour cela, les éléments neutres des lois de l'anneau). Puis on les met à jour au fur et à mesure des divisions successives.

En sortie, une fonction se charge de simplifier le pgcd et d'adapter les coefficients de Bezout en cas de calcul dans l'anneau des polynômes.

Généricité des algorithmes d'Euclide

De la manière dont ils ont été conçus, les algorithmes de calculs du pgcd vont manipuler des éléments sans se soucier de leur nature. Ainsi pour arriver au résultat, il vont utiliser les différentes opérations disponibles dans d'un anneau euclidien (détaillées ci-dessus) sans savoir ce qu'elles font réellement.

Par exemple, pour exécuter la ligne « `tmpCoefA = oldCoefA - q * coefA` » ou la ligne « `q, r = division_euclidienne(quotient, reste)` », le programme se contente d'appeler l'opération correspondante dans l'anneau en cours d'utilisation. Comment s'effectue cette

opération sort du cadre de l'algorithme d'Euclide qui se contente d'utiliser le résultat retourné par l'opération. Le lien entre l'anneau en cours et la fonction a appelée sera fait par le langage de programmation, comment décrit au paragraphe suivant.

Ainsi en remplaçant toute référence à un anneau spécifique, les algorithmes d'Euclide deviennent complètement génériques. On va pouvoir les utiliser pour travailler dans différents type d'anneau.

Comment va faire Python ?

On a préparé nos algorithmes pour travailler avec des données de différentes sortes et ensachant que suivant le type de données, le langage de programmation se débrouillera pour appeler la bonne fonction. Voici comment cela va se passer :

On va coder nos algorithmes sous Python, nos fichiers sources ne seront pas compilés mais interprétés lors du lancement du programme. On va créer trois fichiers sources, un par anneau où l'on souhaite travailler, on aura ainsi un fichier « anneauEntier.py », « anneauPolynome.py » et « anneauZNZ.py » (pour l'anneau Z/nZ). Dans chacun de ces fichiers, on va coder les différentes opérations disponibles dans un anneau, en conformité avec l'anneau qui donné son nom au fichier.

Lors du lancement du programme, Python va associer à chaque variable un type, dans notre cas une variable représentant un élément de l'anneau des entier sera du type « anneauEntier ». C'est dire que le type de la variable est le nom du fichier (sans le .py) où sont fonctionnement est décrit. Ainsi quand on voudra additionner deux polynôme, Python regardera le type de variable en cours de traitement, verra qu'il s'agit de variables de type « anneauPolynome » et utilisera la fonction d'addition du fichier « anneauPolynome.py ». De même, la division euclidienne de deux entiers déclenchera l'utilisation de la fonction division euclidienne du fichier « anneauEntier.py ».

Autre point important, dans les algorithmes on a désigné l'addition par le signe « + », la soustraction par le signe « - » et la multiplication par « * ». On maintenant définir trois fonction appelée « __add__ », « __sub__ » et « __mul__ » que Python va associer respectivement aux trois signes « + », « - » et « * ».

L'anneau des entiers

Pour faciliter la lecture de ce document, les fonctions sont décrites à l'aide « pseudo code », le code Python de ces fonctions est consultable dans les fichiers Python reportés au paragraphe (XZXZ : mettre renvoi).

Voici les fonctions chargées d'effectuer les opérations utilisées dans les algorithmes d'Euclide pour l'anneau des entiers :

```
fonction division_euclidienne(self, B) :
     retouner anneauEntier(quotient(self.value, B.valeur)),
reste(self.valeur, B.valeur)

fonction element_neutre_addition(self) :
     retouner anneauEntier(0)
```

```
fonction element_neutre_multiplication(self) :
     retouner anneauEntier(1)


fonction est_nul(self) :
     if self.valeur == 0 :
          retouner True
     else :
          retouner False

fonction simplifier_pgcd(self) :
     retouner self

fonction simplifier_pgcd_and_coeffs(self, coefA, coefB) :
     retouner self, coefA, coefB

fonction __add__(self, B) :
     retouner anneauEntier(self.valeur + B.valeur)

fonction __sub__(self, B) :
     retouner anneauEntier(self.valeur - B.valeur)

fonction __mul__(self, B) :
     retouner anneauEntier(self.valeur * B.valeur)
```

Les variable de type « anneauEntier » représente un élément de l'anneau des entiers, c'est à dire que ces variables représentent un entier. La valeur de cet entier est stockée dans le champs valeur, pour garantir la généricité des algorithmes de calcul du pgcd ce champs de la variable n'est pas accessible directement. Les algorithmes doivent passer par les fonctions décrites ci-dessus pour manipuler cette donnée.

Ainsi l'appel à la fonction « anneauEntier(val) » crée une variable de type « anneauEntier » avec le champs valeur initialisé avec « val ».

La fonction « `division_euclidienne` » calcule la division euclidienne de deux variables de type « anneauEntier » et retourne le quotient et le reste de cette division au travers deux variables de type « anneauEntier ». Ce dernier point est très important, pour indiquer à Python que les fonctions associées à cette variable se trouvent dans le fichier « anneauEntier.py ».

Les fonctions « `element_neutre_addition` » et « `element_neutre_multiplication` » retournent respectivement l'élément neutre de l'addition et de la multiplication, toujours sous la forme de variable de type « anneauEntier ».

La fonction « `est_nul` » est le fameux test d'égalité avec l'élément neutre de l'addition. C'est cette fonction qui va permettre à la boucle « tant que » des algorithmes de calcul du pgcd de s'arrêter.

Comme indiqué précédemment les fonctions « `simplifier_pgcd` » et « `simplifier_pgcd_and_coeffs` » ne font rien dans le cas des entiers.

Les fonctions «`__add__`» , «`__sub__`» et «`__mul__`» se chargent respectivement de l'addition, de la soustraction et de la multiplication et retourne un résultat sous forme d'une variable de type « anneauEntier ».