Algorithmes de calcul formel

B. Parisse
Institut Fourier
UMR 5582 du CNRS
Université de Grenoble I

Résumé

Ce document décrit une partie des algorithmes de calcul formel utilisés pour le logiciel de calcul formel Giac/Xcas, cf.
`http://www-fourier.ujf-grenoble.fr/~parisse/giac_fr.html`

Table des matières

1 Calculer sur ordinateur

1.1 Problèmes spécifiques au calcul formel

1.1.1 Calcul exact et approché, types, évaluation.

Dans les langages de programmation traditionnel (C, Pascal,...), il existe déjà des types permettant une représentation exacte des données (type entier) ou une représentation approchée (type flottant). Mais ces types de donnée de base occupent une taille fixe en mémoire, le type entier est donc limité à un intervalle d'entiers (par exemple $[0, 2^{32} - 1]$ pour un entier non signé sur une machine utilisant un processeur 32 bits) alors que le type flottant peut représenter des nombres réels, mais est limité à une précision en nombre de digits de la mantisse et de l'exposant (par exemple 12 chiffres significatifs et un exposant compris entre -499 et 499).

En calcul formel, on souhaite pouvoir calculer rigoureusement d'une part, et avec des paramètres dont la valeur n'est pas connue d'autre part ; il faut donc s'affranchir de ces limites :

- pour les entiers relatifs, on utilise des entiers de *précision arbitraire* dont la taille en mémoire est dynamique (déterminée pendant l'exécution et non à la compilation),
- pour les nombres complexes, on utilise un couple de nombres réels,
- pour les rationnels, on utilise un couple d'entiers relatifs,
- pour les irrationnels algébriques (par exemple $\sqrt{2}$), on utilise un polynôme irréductible dont ils sont racines,
- pour les paramètres ($x, y, z, t...$), on utilise un type structuré contenant un champ de type chaine de caractères pour représenter le nom du paramètre et un champ pour attribuer une valeur à (ou une hypothèse sur) ce paramètre,
- pour les nombres transcendants (par exemple π), on est obligé d'introduire un paramètre auquel on attribue une valeur numérique, qui ne sera utilisée qu'au moment où on veut une approximation numérique d'une expression contenant ce nombre transcendant, on parle de constante,
- lorsqu'on a besoin d'une approximation numérique d'un nombre, on peut utiliser des conversions de ces types en un type flottant. On peut aussi pour lutter contre les erreurs d'arrondi utiliser des nombres flottants étendus dont la précision est dynamique ou même des intervalles de flottants étendus,
- il faut aussi un nouveau type, appelé expression ou symbolique, permettant d'appliquer une fonction qu'on ne peut évaluer directement sur les objets précédents, par exemple $\sin(x)$. Il doit s'agir d'une opération de clôture, au sens où appliquer une fonction à un objet symbolique ne nécessite pas la création d'un nouveau type (en général on renvoie un objet symbolique).

Enfin, il faut pouvoir évaluer un objet (en particulier symbolique) : par exemple évaluer $\sin(x)$ lorsqu'on assigne une valeur à x. Dans cet exemple, on voit qu'il faut d'abord remplacer x par sa valeur avant de lui appliquer la fonction sinus. C'est le mécanisme général de l'évaluation, mais il y a quelques exceptions où on souhaite empêcher l'évaluation d'un ou plusieurs arguments d'une fonction avant l'évaluation de la fonction. Par exemple si on veut calculer la valeur numérique d'une intégrale par des méthodes de quadrature, on ne souhaitera pas rechercher une primitive de la fonction à intégrer. Dans le jargon, on parle alors de "quoter" un argument (l'origine du terme vient probablement de la notation `'` du langage

Lisp). Certaines fonctions doivent toujours quoter leurs arguments (par exemple la fonction qui permet de purger le contenu d'un paramètre), on parle parfois d'auto-quotation.

1.1.2 Forme normale et reconnaissance du 0.

Une fois défini ces types de base représentant les nombres d'un système de calcul formel, il faut pouvoir comparer ces nombres, en particulier décider si deux représentations distinctes correspondent au même nombre ou, ce qui revient au même, par soustraction décider quand un nombre est nul. Par exemple $4/2$ et 2 représentent le même nombre. Lorsqu'on dispose d'un algorithme permettant de représenter un nombre d'une manière unique, on parle de forme normale. C'est par exemple le cas pour les nombres rationnels, la forme normale usuelle est la fraction irréductible de dénominateur positif. C'est aussi le cas pour les fractions rationnelles de polynômes à coefficients entiers représentées par une fraction irréductible, avec au dénominateur un coefficient de plus haut degré positif. Malheureusement, il n'est pas toujours possible de trouver une forme normale pour diverses raisons théoriques ou pratiques :

- on ne connaît pas toujours le statut de certaines constantes (par exemple la constante d'Euler),
- il n'existe pas d'algorithmes permettant de déterminer s'il existe des relations algébriques entre constantes,
- il n'existe pas forcément une seule forme plus simple, par exemple :

$$\frac{(\sqrt{2}+1)x+1}{x+\sqrt{2}+1} = \frac{x+\sqrt{2}-1}{(\sqrt{2}-1)x+1}$$

  Ce cas se présente fréquemment avec les extensions algébriques.
- en pratique il peut être trop coûteux d'utiliser une forme normale, par exemple le polynôme $\frac{x^{1000}-1}{x-1}$ possède 1000 monômes

En résumé, au mieux on a une forme normale, au pire on risque de ne pas reconnaître un zéro, entre les deux on peut ne pas avoir de forme normale mais être capable de reconnaître à coup sûr une expression nulle (par contre, si le système de calcul formel détermine qu'une expression est nulle, alors elle l'est).

Il n'existe pas d'algorithme solution pour le problème de la reconnaissance du zéro pour une classe d'expressions "assez générale". Heureusement, dans la plupart des cas pratiques on sait résoudre ce problème, en se ramenant le plus souvent au cas des polynômes et fractions rationnelles. Par exemple, pour simplifier une expression trigonométrique, on remplace les fonctions trigonométriques $\sin(x), \cos(x), \tan(x)$ par leur expression en fonction de $t = \tan(x/2)$, on est ainsi ramené à une fraction rationnelle en t que l'on écrit sous forme normale.

Les polynômes ont un rôle central dans tout système de calcul formel puisque sauf dans les cas les plus simples (fractions d'entiers par exemple), la simplification d'expressions fait appel à un moment ou à un autre à des calculs de PGCD de polynômes. Le PGCD de polynômes est un algorithme très sollicité auquel nous consacrerons une section. En effet, l'application brutale de l'algorithme d'Euclide pose des problèmes d'efficacité ce qui a obligé à inventer des méthodes plus efficaces. Anticipons rapidement sur un exemple qui montre l'un des problèmes majeurs des algorithmes de calcul formel, l'explosion en taille (ici des coefficients des

restes successifs). Voici donc les restes successifs lorsqu'on applique l'algorithme d'Euclide pour calculer le PGCD de $P(x) = (x+1)^7 - (x-1)^6$ avec sa dérivée (les deux polynômes sont premiers entre eux) :

$$7(x+1)^6 - 6(x-1)^5$$
$$\frac{162}{49}x^5 + \frac{-390}{49}x^4 + \frac{1060}{49}x^3 + \frac{-780}{49}x^2 + \frac{474}{49}x + \frac{-78}{49}$$
$$\frac{157780}{729}x^4 + \frac{-507640}{2187}x^3 + \frac{290864}{729}x^2 + \frac{-101528}{729}x + \frac{28028}{729}$$
$$\frac{1}{49}(\frac{1400328}{2645}x^3 + \frac{-732888}{2645}x^2 + \frac{1133352}{3703}x + \frac{-732888}{18515})$$
$$\frac{1}{2187}(\frac{2161816376832}{4669921}x^2 + \frac{-555436846944}{4669921}x + \frac{301917024864}{4669921})$$
$$\frac{1}{907235}(\frac{469345063045455}{129411872}x + \frac{-47641670106615}{129411872})$$
$$\frac{5497465490623352995840}{209648836272383412129}$$

Le lecteur voulant tester d'autres exemples pourra utiliser le programme `Xcas` (cf. l'appendice) suivant :

```
pgcd(a):={
  local b,r,res;
  b:=diff(a,x);
  res:=NULL;
  for (;b!=0;){
    res:=res,b;
    r:=rem(a,b);
    a:=b;
    b:=r;
  }
  return(res);
}
```

1.1.3 Valeur générique des variables et hypothèses

Lorsqu'on utilise un symbole sans lui affecter de valeurs en mathématiques on s'attend à une discussion en fonction du paramètre représenté par ce symbole. Ce qui nécessiterait de créer un arborescence de calculs (on retrouve ici les problèmes d'explosion évoqués dans la section précédente). La plupart des systèmes de calcul formel contournent la difficulté en supposant que le paramètre possède une valeur générique (par exemple la solution de $(t^2-1)x = t-1$ sera $x = 1/(t+1)$) ou choisissent une branche pour les fonctions possédant un point de branchement (par exemple pour résoudre $x^2 = t$ en fonction de t). Certains systèmes demandent de manière interactive à l'utilisateur si la variable est par exemple positive ou différente de 1 mais cela s'oppose à un traitement automatique. On peut aussi anticiper ce type de décision en faisant des hypothèses sur une paramètre, la plupart des systèmes de calcul formel actuel proposent cette possibilité.

1.2 Structures de données

On a vu plus haut qu'on souhaitait manipuler des entiers de taille non fixe, des réels de précision fixe ou non, des fractions, des nombres complexes, des extensions algébriques, des paramètres, des expressions symboliques. La plupart des systèmes proposent un type générique qui recouvre ces divers types de scalaire. On peut par exemple utiliser un type structuré comportant un champ type et la donnée ou un pointeur sur la donnée (avec dans ce cas un pointeur sur un compteur de références de la donnée pour pouvoir la détruire dès qu'elle n'est plus référencée[1]). En programmation orientée objet, on utiliserait plutôt un type abstrait dont dérivent ces différents scalaires et le polymorphisme.

Il faut aussi un type pour les vecteurs, les matrices et les listes. Il faut prendre garde à la méthode utilisée par le système lorsqu'on modifie un élément d'un vecteur, matrice ou liste : soit on effectue une copie de tout l'objet en modifiant l'élément, soit on modifie l'élément de l'objet original. La première méthode (par valeur) est plus aisée à comprendre pour un débutant mais la seconde méthode (par référence) est bien plus efficace.

On peut se poser la question de savoir s'il faut inclure ces types dans le type générique ; en général la réponse est affirmative, une des raisons étant que les interpréteurs qui permettront de lire des données dans un fichier texte sont en général basé sur le couple de logiciels `lex(flex)/yacc(bison)` qui ne peut compiler qu'à destination d'un seul type. Ceci permet également d'unifier en un seul type symbolique les fonctions ayant un ou plusieurs arguments en voyant plusieurs arguments comme un vecteur d'arguments. Les fonctions sont le plus souvent elle-même incluses dans le type générique permettant ainsi à l'utilisateur de saisir des commandes ou programmes fonctionnels (on peut utiliser une fonction comme argument d'une commande).

Pour des raisons d'efficacité, les systèmes de calcul formel utilisent souvent des représentations particulières pour les polynômes dont on a dit qu'ils jouaient un rôle central. Pour les polynômes à une variable, on peut utiliser la liste des coefficients du polynôme, on parle alors de représentation dense. On peut aussi décider de ne stocker que les coefficients non nuls, on parle alors de représentation creuse (on stocke alors un couple formé par le coefficient et le degré du monôme correspondant). Pour les polynômes à plusieurs variables, on peut les considérer comme des polynômes à une variable à coefficients polynomiaux, on parle alors de représentation récursive. On peut aussi décider de ne pas briser la symétrie entre les variables (pas de variable principale), on parle alors de représentation distribuée, le plus souvent les représentation distribuées sont creuses car les représentations denses nécessitent très vite beaucoup de coefficients. Les méthodes de représentation creuses sont parfois aussi utilisées pour les matrices ayant beaucoup de coefficients nuls.

Voyons maintenant plus précisément sur quelques exemples de logiciels de calcul formel répandus quelles structures de données sont utilisées. Plusieurs éléments entrent en compte dans les choix faits :

[1] Certains systèmes de calcul formel (calculatrices par exemple) utilisent d'ailleurs des méthodes spécifiques pour gérer le problème de la fragmentation de la mémoire, appelés "garbage collector". Ce type de méthode est intégré dans des langages comme Lisp ou Java, en C/C++ on trouve des libraries pour cela, par exemple GC de Boehm, incluse dans la distribution de GCC.

– le(s) profil(s) d'utilisation (enseignement, ingéniérie, calcul intensif, recherche)
– les ressources disponibles (mémoire, puissance du processeur...)
– la facilité d'implémentation (choix du langage, outils disponibles en particulier débuggueurs, ...)
– l'histoire du système (un système conçu avec les outils disponibles aujourd'hui est forcément différent d'un système conçu il y a 20 ans)

Nous allons d'abord parler des calculatrices formelles HP et TI (le lecteur pourra facilement les tester grâce aux émulateurs gratuits pour PC). Ce sont des systèmes plutôt destinés à l'enseignement, soumis à de fortes contraintes en termes de taille mémoire, et destinés à traiter des petits problèmes. Puis nous présenterons des systèmes pour ordinateur où les ressources (par exemple mémoire) sont moins limitées ce qui permet d'utiliser des langages de programmation de plus haut niveau.

1.2.1 Calculatrices formelles HP

Les langages utilisés pour programmer ces calculateurs sont l'assembleur et le RPL (Reverse Polish Lisp) adapté à l'écriture de code en mémoire morte très compact.

Le type générique est implémenté avec un champ type appelé prologue (qui est en fait un pointeur sur la fonction chargée d'évaluer ce type d'objet) suivi de la donnée elle-même (et non d'un pointeur sur la donnée, on économise ainsi la place mémoire du compteur de référence).

Le type entier en précision arbitraire est codé par le nombre de digits (sur 5 quartets[2]) suivi du signe sur un quartet et de la représentation BCD (en base 10) de la valeur absolue de l'entier. Le choix de la représentation BCD a été fait pour optimiser les temps de conversion en chaîne de caractères pour l'affichage. La mémoire vive disponible est de 256K, c'est elle qui limite la taille des entiers et non le champ longueur de l'entier. Il n'y a pas de type spécifique pour les rationnels (on utilise un objet symbolique normal).

Les fonctions internes des HP49/50/40 utilisent le type programme pour représenter les entiers de Gauß (complexes dont la partie réelle et imaginaire est entière). Les nombres algébriques ne sont pas implémentés, sauf les racines carrées (représentée de manière interne par le type programme). Il y a un type spécifique prévu pour les flottants en précision arbitraire, mais l'implémentation des opérations sur ces types n'a pas été intégrée en ROM à ce jour.

Les types listes, programmes et objet symbolique sont composés du prologue (champ type) suivi par la succession d'objets situés en mémoire vive ou de pointeurs sur des objets situés en mémoire en lecture seule (ROM) et se terminent par un pointeur sur une adresse fixe (appelée `SEMI`). Ces types sont eux-mêmes des objets et peuvent donc être utilisés de manière récursive. La longueur des types listes, programmes, symboliques n'est stockée nulle part, c'est le délimiteur final qui permet de la connaître, ce qui est parfois source d'inefficacité. On utilise de manière interne les listes pour représenter les polynômes denses (avec représentation récursive pour les polynômes à plusieurs variables).

Les calculatrices HP4xG utilisent une pile[3], c'est-à-dire une liste de taille non

[2] un quartet=un demi octet

[3] Plus précisément deux piles, la pile de donnée et la pile gérant le flux d'exécution. Cette dernière n'est pas visible par l'utilisateur

fixée d'objets. On place les objets sur la pile, l'exécution d'une fonction prend ces arguments sur la pile et renvoie un ou plusieurs résultats sur la pile (ce qui est une souplesse du RPN comparé aux langages où on ne peut renvoyer qu'une valeur de retour). Il faut donc donner les arguments avant d'appeler la fonction correspondante. Par exemple pour calculer $a+b$ on tapera `a b +`. C'est la syntaxe dite polonaise inversée (RPN). Un avantage de cette syntaxe est que le codage d'un objet symbolique par cette syntaxe est évidente, il suffit de stocker la liste précédente `{a b +}`. Les objets symboliques sont donc représenté par une suite d'objets écrit en syntaxe polonaise inversée. L'évaluation d'un objet symbolique se fait dans l'ordre polonaise inversé : les arguments sont évalués puis les fonctions leur sont appliqués. Pour des raisons d'efficacité, on représente souvent les objets composites (listes, symboliques) par leurs composants placés sur la pile (appelé meta-objets).

Une rigidité de la syntaxe polonaise est que les fonctions ont toujours un nombre fixe d'arguments[4], par exemple l'addition a toujours 2 arguments, ainsi $a+b+c$ est obtenu par $(a+b)+c$ ou par $a+(b+c)$ c'est-à-dire respectivement `a b + c +` ou `a b c + +` ce qui brise parfois artificiellement la symétrie de certaines opérations. En polonaise inversée, le système doit de plus jongler avec l'autoquote puisque les arguments sont évalués avant l'opérateur qui éventuellement demanderait à ne pas évaluer ses arguments. À noter l'existence d'une commande `QUOTE` permettant à l'utilisateur de quoter une sous-expression.

Les hypothèses sur des variables réelles sont regroupées dans une liste stockée dans la variable globale `REALASSUME`, on peut supposer qu'une variable est dans un intervalle. Il n'y a pas à ce jour de possibilité de supposer qu'une variable est entière (ni à fortiori qu'une variable à une valeur modulo un entier fixé), bien qu'il ait été décidé de réserver la variable globale `INTEGERASSUME` à cet effet. Il n'y a pas de possibilité de faire des hypothèses ayant une portée locale.

1.2.2 Calculatrices formelles TI

Le langage utilisé pour programmer ces calculatrices est le langage C (on peut aussi écrire du code en assembleur pour ces calculatrices). On retrouve ici les différents types de données regroupé en un type générique qui est un tableau d'octets (aussi appelé quantum). Le champ type est appelé tag dans la documentation TI. Contrairement à ce qui précède, ce champ type est placé en mémoire à la fin de l'objet, ce qui est possible car la longueur d'un objet est toujours indiquée au début de l'objet. Ceci est fait afin de faciliter l'évaluation (cf. infra).

Les entiers en précision arbitraire sont codés par un tag parmi deux (pour différencier le signe), un octet pour la longueur, puis la valeur absolue de l'entier (en base 256). Ils sont donc limités par le champ longueur à 255 octets, le plus grand entier représentable est [5] $(256^{255}-1)$. Il existe un tag spécifique pour les rationnels, pour les constantes réelles et entières qui apparaissent par exemple en résolvant une équation. Il existe des tags utilisés de manière interne, par exemple pour les nombres complexes. Il n'y a pas de tag prévu pour les flottants en précision

[4] Sauf si on utilise comme dernier argument le nombre d'arguments de la fonction ou si on utilise (cf. infra) un tag de début de liste d'arguments

[5] Toutefois une adaptation du logiciel utilisant comme quantum de base par exemple 32 bits porterait cette limite à $65536^{65535}-1$

arbitraire. ni pour les nombres algébriques (racines carrées par exemple).

Les listes sont codées par la succession de leurs éléments. En principe elles ne peuvent pas contenir des listes (sauf pour représenter une matrice). Quelques fonctions utilisent les listes pour représenter des polynômes denses à une variable, mais probablement pas pour représenter de manière récursive des polynômes à plusieurs variables (puisque le type liste n'est en principe pas récursif).

Comme les HP, les TI utilisent une pile (non visible par l'utilisateur) appelée expression stack afin de traduire un expression mathématique sous forme d'un texte en un objet symbolique codé exactement comme ci-dessus en syntaxe polonaise. Toutefois, la présence du champ longueur permet d'évaluer un objet symbolique sans perdre en efficacité en partant de l'opérateur final et en redescendant ensuite sur ces arguments, c'est la stratégie adoptée. C'est pour cela que le tag d'identification se trouve à la fin de l'objet. L'utilisation de cette méthode facilite grandement l'autoquotation (on peut toutefois regretter que le système n'ait pas prévu d'instruction permettant à l'utilisateur d'empêcher l'évaluation d'une sous-expression).

On ne peut pas faire d'hypothèse globale sur un paramètre par contre on peut faire des hypothèses de type appartenance à un intervalle ayant une portée locale.

1.2.3 Maple, MuPAD, Mathematica, ...

Ces systèmes ont un noyau fermé, au sens où l'utilisateur n'a pas accès du tout, ou en tout cas pas facilement, aux structures de données de base. Je ne dispose donc pas d'information sur les structures de données utilisées par le noyau (pour MuPAD, on pourrait sans doute en savoir plus en achetant de la documentation sur la programmation des modules dynamiques).

L'interaction système-utilisateur se fait quasiment toujours en utilisant le langage de programmation propre au système, langage interprété par le noyau du système (ce qui ralentit l'exécution). Ces langages utilisateurs sont essentiellement non typés : on travaille avec des variables du type générique sans pouvoir accéder aux types sous-jacents. On ne bénéficie en général pas des vérifications faites lors de la compilation avec un langage typé, de plus ces systèmes ne sont pas toujours fourni avec de bon outils de mise au point. Enfin ces langages ne sont pas standardisés d'un système à l'autre et il est en général impossible d'utiliser ces systèmes comme des librairies depuis un langage de programmation traditionnel. Leur intérêt principal réside donc dans une utilisation interactive en profitant de la librairie de fonctions accessibles.

1.2.4 Giac/xcas

Il s'agit du système de calcul formel que j'implémente actuellement sous forme d'une bibliothèque C++ (ce qui permettra aux programmes tiers d'utiliser beaucoup plus facilement du calcul formel qu'avec les systèmes précédents). L'objectif est d'avoir un système facile à programmer directement en C++, proche du langage utilisateur, lui-même compatible avec Maple ou MuPAD, tout cela sans trop perdre en performances comparativement aux librairies spécialisées écrites en C/C++. Ce qui explique un choix de type générique (`gen`) non orienté objet, avec un champ type et soit une donnée immédiate (pour les nombres flottants par exemple), soit un pointeur vers un objet du type correspondant au champ type pour les données

de taille non fixe (on pourrait donc se contenter du langage C, mais le langage C++ permet de redéfinir les opérateurs sur des types utilisateurs ce qui améliore considérablement la lisibilité du code source). Les données dynamiques ne sont pas dupliquées, Giac utilise un pointeur sur un compteur de référence pour détruire ces données lorsqu'elles ne sont plus référencées.

Les entiers en précision arbitraire sont hérités de la bibliothque GMP (écrite en C) du projet GNU. Les flottants en précision arbitraire utiliseront aussi GMP (plus précisément MPFR). Il y a un type fraction, structure C composé d'un champ numérateur et d'un champ dénominateur, et un type nombre complexe.

Les listes, vecteurs, matrices utilisent le type paramétré `vector<>` de la librairie standard C++ (Standard Template Library). Les objets symboliques sont des structures composés d'un champ sommet qui est une fonction prenant un argument de type `gen` et renvoyant un résultat de type `gen`, et d'un champ feuille qui est de type `gen`. Lorsqu'une fonction possède plusieurs arguments, ils sont rassemblés en une liste formant le champ feuille de l'objet symbolique. Les programmes sont aussi des objets symboliques, dont le champ sommet est la fonction évaluation d'un programme. Les listes sont aussi utilisées pour représenter vecteurs, matrices et polynômes en une variable en représentation dense, on peut y accéder par valeur (`:=`) ou par référence (`=<`). Les polynômes en représentation creuse ou en plusieurs indéterminées sont également disponibles.

L'évaluation d'un objet symbolique se fait en regardant d'abord si la fonction au sommet doit évaluer ou non ses arguments (autoquote), on évalue les arguments si nécessaire puis on applique la fonction.

Une hypthèse sur un paramètre est une valeur spéciale affectée au paramètre, valeur ignorée par la routine d'évaluation.

1.3 Algorithmes et complexité.

On va présenter dans la suite quelques algorithmes que l'on peut considérer comme classiques dans le domaine du calcul formel. Avant d'implémenter ce type d'algorithmes, on a besoin des algorithmes de base en arithmétique. Le lecteur trouvera en appendice une brève présentation de certains de ces algorithmes, mes références en la matière sont le livre de Henri Cohen, et les livres de Donald Knuth (cf. appendice).

La plupart des problèmes posés en calcul formel nécessitent des calculs dont la taille croit de manière exponentielle voire doublement exponentielle en fonction de la taille des données et ce même si le résultat est lui aussi de taille petite. Un exemple est la réduction des systèmes de plusieurs équations polynomiales (bases de Groebner). Dans certains cas, l'application de théories mathématiques parfois sophistiquées permet de réduire la complexité (par exemple, M. Van Hoeij a découvert récemment qu'un algorithme très utilisé en théorie des nombres, l'algorithme LLL, permettait d'améliorer la complexité d'une des étapes de la factorisation des polynomes à coefficients entiers sur les entiers). Heureusement, dans de nombreux cas, on peut réduire la complexité (donc le temps de calcul) par des adaptations au problème d'une même idée à condition de faire des hypothèses sur les données (autrement dit en abandonnant la volonté d'implémenter un algorithme très générique, ou tout au moins en spécialisant des algorithmes génériques).

Par exemple lorsqu'on travaille avec des entiers (ou des polynômes à coeffi-

cients entiers, ou des matrices à coefficients entiers...) on utilise souvent des algorithmes modulaires et p-adiques. Comme le calcul exact nécessite presque toujours de calculer avec des entiers, ces méthodes ont un rôle central en calcul formel, nous les présentons donc maintenant brièvement. Dans les prochaines sections, nous utiliserons ce type de méthode, par exemple pour le calcul de PGCD ou la factorisation de polynômes à coefficients entiers.

Les méthodes modulaires consistent à réduire un problème dans $\mathbb{Z}$ à son équivalent dans $Z/n\mathbb{Z}$ pour une ou plusieurs valeurs de n, nombre premier. Le calcul dans $\mathbb{Z}/n\mathbb{Z}$ a l'avantage de se faire avec des entiers dont la taille est bornée. Ensuite à l'aide d'estimations à priori sur la taille des solutions éventuelles du problème initial, on reconstruit la solution au problème initial avec le théorème des restes chinois.

Par exemple, on peut calculer un déterminant d'une matrice à coefficients entiers en cherchant ce déterminant dans $\mathbb{Z}/n\mathbb{Z}$ pour plusieurs premiers n, dont le produit est plus grand qu'une estimation à priori de la taille du déterminant (donnée par exemple par l'inégalité d'Hadamard, cf. Cohen, p. 50).

Les méthodes p-adiques commencent de manière identique par un calcul dans $\mathbb{Z}/n\mathbb{Z}$, on augmente ensuite la précision de la solution en la « liftant »de $\mathbb{Z}/n^k\mathbb{Z}$ vers $\mathbb{Z}/n^{k+1}\mathbb{Z}$ ou vers $\mathbb{Z}/n^{2k}\mathbb{Z}$ (lift linéaire ou lift quadratique), on s'arrête lorsque k est assez grand (à l'aide d'estimations à priori) et on reconstruit alors la solution initiale. L'étape de « lift »est en général un lemme de Hensel dont on verra quelques exemples dans les prochains articles. L'algorithme commun au lemme de Hensel et au théorème des restes chinois est l'identité de Bézout, que l'on retrouve d'ailleurs un peu partout (par exemple pour le calcul de primitives).

Illustrons cette méthode sur un exemple simple, la recherche de racines rationnelles d'un polynôme $P(X) = a_d X^d + \cdots + a_0$ à coefficients entiers ou polynomiaux, avec a_d et a_0 non nuls. L'algorithme générique (assez connu) consiste à chercher les diviseurs de a_0 et de a_d et à tester toutes les fractions de ces diviseurs, on montre en effet aisément que si $X = p/q$ fraction irréductible est racine de P alors q divise a_d et p divise a_0. Cet algorithme est très inefficace si a_d ou a_0 est un grand entier (car on ne sait pas forcément le factoriser) ou s'il a beaucoup de facteurs premiers (la liste des diviseurs à tester est alors très grande).

Lorsque les coefficients de P sont entiers, la recherche précédente revient à trouver un facteur à coefficients entiers $qX - p$ de P, on peut donc réduire le problème modulo un entier premier n qui ne divise pas a_d : si un tel facteur existe dans $\mathbb{Z}$ alors ce facteur (réduit modulo n) est un facteur de P dans $\mathbb{Z}/n\mathbb{Z}$ donc P admet une racine dans $\mathbb{Z}/n\mathbb{Z}$ (puisque q est inversible modulo n car on a choisi n premier ne divisant pas a_d). On évalue maintenant P en les n éléments de $\mathbb{Z}/n\mathbb{Z}$. S'il n'y a pas de 0, alors P n'a pas de racine rationnelle. S'il y a des racines, on va les lifter de $\mathbb{Z}/n^k\mathbb{Z}$ dans $\mathbb{Z}/n^{2k}\mathbb{Z}$.

On suppose donc que pour $k \geq 1$, il existe un entier p_k tel que

$$P(p_k) = 0 \pmod{n^k}$$

Il s'agit de trouver un entier x tel que $p_{k+1} = p_k + n^k x$ vérifie

$$P(p_{k+1}) = 0 \pmod{n^{2k}}$$

On applique la formule de Taylor à l'ordre 1 pour P en p_k, le reste est nul modulo

n^{2k}, donc :

$$P(p_k) + n^k x P'(p_k) = 0 \pmod{n^{2k}}$$

soit finalement :

$$x = -\frac{P(p_k)}{n^k}(P'(p_k) \pmod{n^k})^{-1}$$

On reconnaît au passage la méthode de Newton, pour qu'elle fonctionne il suffit que $P'(p_k) \neq 0 \pmod{n}$ ce qui permet de l'inverser modulo n^k (et c'est ici qu'intervient l'identité de Bézout). En pratique quand on factorise un polynôme, on commence par retirer les multiplicités, on peut donc supposer que P est sans facteur multiple dans $\mathbb{Z}$. Ceci n'entraîne pas forcément qu'il le reste dans $\mathbb{Z}/n\mathbb{Z}$ ce qui crée une contrainte supplémentaire sur le choix de n, à savoir que P et P' restent premier entre eux dans $\mathbb{Z}/n\mathbb{Z}$ (il existe forcément de tels n, par exemple n premier plus grand que le plus grand entier intervenant dans le calcul du PGCD de P et P' dans $\mathbb{Z}$).

Reste donc à revenir dans $\mathbb{Z}$ à partir d'une racine p_k dans $\mathbb{Z}/(n^k\mathbb{Z})$ (où on peut choisir k). On va maintenant utiliser la représentation modulaire symétrique : on prend comme représentant modulaire d'un entier z dans $\mathbb{Z}/n^k\mathbb{Z}$ l'unique entier congru à z modulo n qui est strictement compris entre $-n^k/2$ et $n^k/2$ (si n est pair, la deuxième inégalité est choisie large).

Si $qX - p$ est un facteur de P, alors $a_dX - \frac{a_d}{q}p$ est encore un facteur de P (le quotient de P par $a_dX - \frac{a_d}{q}p$ est à coefficients rationnels mais le facteur est à coefficients entiers). Si on a choisi k tel que $n^k > 2|a_d a_0|$, l'écriture en représentation modulaire symétrique de $a_dX - \frac{a_d}{q}p$ est inchangée, en effet on a des estimations à priori sur les entiers p et q : $|q| \leq |a_d|$ et $|p| \leq |a_0|$ puisque q divise a_d et p divise a_0. Comme $a_dX - \frac{a_d}{q}p$ est égal à $a_d(X - p_k)$ dans $\mathbb{Z}/(n^k\mathbb{Z})$, il nous suffit d'écrire en représentation modulaire symétrique $a_d(X - p_k) = a_dX - p'$. Pour conclure, on sait que $a_dX - p'$ est un multiple entier de $qX - p$. On divise donc le facteur $a_dX - p'$ par le pgcd de a_d et p' et on teste la divisibilité de P par ce facteur réduit.

Exemple

Considérons le polynôme $2X^3 - X^2 - X - 3$ qui est sans facteur carré. On ne peut pas choisir $n = 2$ car on réduirait le degré, pour $n = 3$, on a $P' = X - 1$ qui est facteur de P, pour $n = 5$, $P' = 6X^2 - 2X - 1$, on vérifie que P et P' sont premiers entre eux (par exemple avec `GCDMOD` sur une HP49 où on aura fixé la variable `MODULO` à 5).

On teste ensuite les entiers de -2 à 2 sur P. Seul -1 est racine modulo 5 ($P(-1) = -5$), on va maintenant lifter $p_1 = -1$.

L'estimation à priori est $2|a_d||a_0| = 12$ donc $k = 2$ ($5^2 = 25 > 12$), une itération suffira. On a $P'(-1) = 7$, l'inverse de $P'(-1) \pmod{5}$ est -2 donc :

$$x = -\frac{P(-1)}{5}(-2) = -(-1)(-2) = -2$$

et $p_2 = -1 + 5 \times (-2) = -11$ est racine de P dans $\mathbb{Z}/25\mathbb{Z}$. On calcule ensuite $a_d(X - p_k) = 2(X + 11) = 2X + 22 = 2X - 3$ en représentation symétrique, le PGCD de 2 et -3 est 1 donc on teste le facteur $2X - 3$, ici il divise P donc P admet un unique facteur entier de degré 1 qui est $2X - 3$.

2 Quelques algorithmes d'arithmétique de base.

- Les algorithmes de multiplication et division dit rapides des entiers et polynômes (Karatsuba, FFT, ...). Cf. par exemple Knuth. ou pour les entiers la documentation de GMP.
- Au lieu de la division euclidienne, on utilise très souvent la pseudo-division pour les polynômes : étant donné deux polynômes A et B de degrés a et b à coefficients dans un anneau contenu dans un corps (par exemple $\mathbb{Z}$), on multiplie A par une puissance du coefficient dominant B_b de B, plus précisément par B_b^{a-b+1}, ce qui permet d'effectuer la division par B sans que les coefficients sortent de l'anneau.

  $$B_b^{a-b+1}A = BQ + R$$

  On utilise cette méthode lorsqu'on peut multiplier les polynômes par des constantes sans changer le problème (par exemple pour l'algorithme d'Euclide).
- L'algorithme d'Euclide est un algorithme « générique »de calcul de PGCD. Il n'est en général pas utilisé tel quel. Pour les entiers on utilise une variation adaptée à la représentation binaire des entiers (cf. Cohen ou le manuel de GMP version 4 pour plus de détails). Nous décrirons des algorithmes de PGCD plus efficaces pour les polynômes dans le prochain article.
- l'identité de Bézout, aussi appelée PGCD étendu. Étant donné deux entiers ou deux polynômes a et b on calcule u, v et d tels que $au + bv = d$. On écrit la matrice :

  $$\begin{pmatrix} a & 1 & 0 \\ b & 0 & 1 \end{pmatrix}$$

  où on remarque que pour chaque ligne le coefficient de la 1ère colonne est égal à a multiplié par le coefficient de la 2ème colonne additionné à b multiplié par le coefficient de la 3ème colonne. Ce qui reste vrai si on effectue des combinaisons linéaires de lignes (type réduction de Gauß). Comme on travaille dans les entiers ou les polynômes, on remplace la réduction de Gauß des matrices à coefficients réels par une combinaison linéaire utilisant le quotient *euclidien* q de a par b. On obtient alors le reste r en 1ère colonne :

  $$L_3 = L_1 - qL_2 \quad \begin{pmatrix} a & 1 & 0 \\ b & 0 & 1 \\ r & 1 & -q \end{pmatrix}$$

  et on recommence jusqu'à obtenir 0 en 1ère colonne. L'avant-dernière ligne obtenue est l'identité de Bézout (la dernière ligne donne le PPCM de a et b). Si l'on veut l'inverse de a modulo b on remarque qu'il n'est pas utile de calculer les coefficients appartenant à la 3ème colonne. Enfin, les lignes intermédiaires peuvent servir à reconstruire une fraction d'entier représentée par un entier de $\mathbb{Z}/n\mathbb{Z}$ lorsque le numérateur et le dénominateur sont de valeur absolue inférieure à $\sqrt{n/2}$.
- Le théorème des restes chinois. Si on connaît $x = a \pmod m$ et $x = b \pmod n$ avec m et n premiers entre eux, on détermine c tel que $x = c \pmod{m \times n}$ ($c = a + mu = b + nv$ et on applique Bézout pour trouver u et v, on en déduit c).

– Les tests de pseudo-primalité. Il est essentiel d'avoir une méthode rapide permettant de générer des nombres premiers pour appliquer des méthodes modulaires et p-adiques. On utilise par exemple le test de Miller-Rabin, qui prolonge le petit théorème de Fermat (si p est premier, alors $a^p = a \pmod p$).

2.1 Pour en savoir plus.

Sur des aspects plus théoriques :
- Knuth : TAOCP (The Art of Computer Programming), volumes 1 et suivants
- Henri Cohen : A Course in Computational Algebraic Number Theory
- Davenport, Siret, Tournier : Calcul formel : Systèmes et algorithmes de manipulations algébriques

Sur des aspects plus pratiques, quelques références en ligne, la plupart sont accessibles gratuitement :
- le code source de Giac disponible à l'URL :
  `http://www-fourier.ujf-grenoble.fr/~parisse/giac.html`
- le code source de GiNaC, cf. : `http://www.ginac.de`
- le site `http://www.hpcalc.org` pour les calculatrices HP, on y trouve tout, de la documentation, des émulateurs de calculatrices HP, des outils de développement pour Windows et Unix/Linux, ... Pour ce qui concerne cet article, je conseille de lire
  `http://www.hpcalc.org/hp48/docs/programming/rplman.zip`
- le site `http://www.ticalc.org`, on y trouve le portage tigcc du compilateur C de GNU, des émulateurs, etc. Des informations de cet article ont leur source dans le guide du développeur TI89/92
  `http://education.ti.com/`
- la librairie du système `MuPAD` (archivée dans le fichier `lib.tar` des distributions Unix, pour une installation par défaut, ce fichier se trouve dans le répertoire `/usr/local/MuPAD/share/lib`), cf. `www.sciface.com` pour obtenir une licence d'utilisation.
- en Maple, il est possible de décompiler une instruction `Maple` avec la commande
  `eval(instruction);`
  après avoir tapé
  `interface(verboseproc=2);`
- le source du plus ancien système de calcul formel `maxima` (devenu logiciel libre) pour les personnes familières du langage Lisp
  `http://sourceforge.net/projects/maxima`
  de même pour le système Axiom
- le source de librairies plus spécialisées (GMP, GP-PARI, Singular, NTL, Zen, ALP, GAP, CoCoA, ...), rechercher ces moms sur google.

3 Exercices sur types, calcul exact et approché, algorithmes de bases

Pour télécharger et installer Xcas sur votre ordinateur, suivre les instructions données sur
`http://www-fourier.ujf-grenoble.fr/~parisse/giac_fr.html`
Pour lancer `xcas` sous Unix, ouvrir un fenêtre terminal et taper la commande

```
xcas &
```

Lors de la première exécution, vous devrez choisir entre différents types de syntaxe (compatible C, maple ou TI89). Vous pouvez changer ce choix à tout moment en utilisant le menu Configuration->mode (syntaxe).

L'aide en ligne est accessible en tapant `?nom_de_commande`. Dans Xcas, vous pouvez aussi taper le début d'un nom de commande puis la touche de tabulation (à gauche du A sur un clavier francais), sélectionner la commande dans la boite de dialogues puis cliquer sur Details pour avoir une aide plus complète dans votre navigateur. Pour plus de détails sur l'interface de Xcas, consultez le manuel (Aide->Interface). Si vous n'avez jamais utilisé de logiciel de calcul formel, vous pouvez commencer par lire le tutoriel (menu Aide->Debuter en calcul formel->tutoriel) et faire certains des exercices proposés (des corrigés sous forme de sessions Xcas sont dans Aide->Debuter en calcul formel->solutions)

Il peut être interessant de tester ces exercices en parallèle avec Xcas et des calculatrices formelles....

1. Utiliser la commande `type` ou `whattype` ou équivalent pour déterminer la représentation utilisée par le logiciel pour représenter une fraction, un nombre complexe, un flottant en précision machine, un flottant avec 100 décimales, la variable x, l'expression $\sin(x)+2$, la fonction `x->sin(x)`, une liste, une séquence, un vecteur, une matrice. Essayez d'accéder aux parties de l'objet pour les objets composites (en utilisant `op` par exemple).
2. Comparer le type de l'objet `t` si on effectue la commande `t[2]:=0;` après avoir purgé `t` ou après avoir affecté `t:=[1,2,3]` ?
3. Comparer l'effet de l'affectation dans une liste et dans un vecteur ou une matrice sur votre logiciel (en Xcas, on peut utiliser `=<` au lieu de `:=` pour stocker par référence).
4. Voici un programme écrit en syntaxe compatible maple (menu Cfg->Mode->maple dans Xcas) qui calcule la base utilisée pour représenter les flottants.

   ```
   Base:=proc()
   local A,B;
   A:=1.0; B:=1.0;
   while evalf(evalf(A+1.0)-A)-1.0=0.0 do A:=2*A; od;
   while evalf(evalf(A+B)-A)-B<>0 do B:=B+1; od;
   B;
   end;
   ```

   Testez-le et expliquez.
5. Déterminer le plus grand réel positif x de la forme 2^{-n} (n entier) tel que $(1.0+x)-1.0$ renvoie 0 sur PC avec la précision par défaut puis avec `Digits:=30`.

6. Calculer la valeur de $a := \exp(\pi\sqrt{163})$ avec 30 chiffres significatifs, puis sa partie fractionnaire. Proposez une commande permettant de décider si a est un entier.
7. Déterminer la valeur et le signe de la fraction rationnelle
$$F(x,y) = \frac{1335}{4}y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + \frac{11}{2}y^8 + \frac{x}{2y}$$
en $x = 77617$ et $y = 33096$ en faisant deux calculs, l'un en mode approché et l'autre en mode exact. Que pensez-vous de ces résultats ? Combien de chiffres significatifs faut-il pour obtenir un résultat raisonnable en mode approché ?
8. À quelle vitesse votre logiciel multiplie-t-il des grands entiers (en fonction du nombre de chiffres) ? On pourra tester le temps de calcul du produit de $a(a+1)$ où $a = 10000!$, $a = 15000!$, etc.
9. Comparer le temps de calcul de $a^n \pmod m$ par la fonction `powmod` et la méthode prendre le reste modulo m après avoir calculé a^n.
Programmez la méthode rapide et la méthode lente.
Que se passe-t-il si on essaie d'appliquer l'algorithme de la puissance rapide pour calculer $(x+y+z+1)^{32}$? Calculer le nombre de termes dans le développement de $(x+y+z+1)^n$ et expliquez.
10. Déterminer un entier c tel que $c = 1 \pmod 3$, $c = 3 \pmod 5$, $c = 5 \pmod 7$ et $c = 2 \pmod 1$1.
11. Programmation de la méthode de Horner
Il s'agit d'évaluer efficacement un polynôme
$$P(X) = a_nX^n + ... + a_0$$
en un point. On pose $b_0 = P(\alpha)$ et on écrit :
$$P(X) - b_0 = (X - \alpha)Q(X)$$
où :
$$Q(X) = b_nX^{n-1} + ... + b_2X + b_1$$
On calcule alors par ordre décroissant b_n, b_{n-1}, ..., b_0.
   (a) Donner b_n en fonction de a_n puis pour $i \leq n-1$, b_i en fonction de a_i et b_{i+1}. Indiquez le détail des calculs pour $P(X) = X^3 - 2X + 5$ et une valeur de α non nulle.
   (b) Écrire un fonction `horn` effectuant ce calcul : on donnera en arguments le polynôme sous forme de la liste de ces coefficients (dans l'exemple `[1,0,-2,5]`) et la valeur de α et le programme renverra $P(\alpha)$. (On pourra aussi renvoyer les coefficients de Q).
   (c) En utilisant cette fonction, écrire une fonction qui calcule le développement de Taylor complet d'un polynôme en un point.
12. Algorithmes de base : écrire des programmes implémentant
   (a) le pgcd de 2 entiers
   (b) l'algorithme de Bézout
   (c) l'inverse modulaire en ne calculant que ce qui est nécessaire dans l'algorithme de Bézout
   (d) les restes chinois

4 Le PGCD

Comme on l'a remarqué dans le premier article, l'algorithme d'Euclide est inefficace pour calculer le pgcd de deux polynômes à coefficients entiers. On va présenter ici les algorithmes utilisés habituellement par les systèmes de calcul formel : sous-résultant (PRS), modulaire (GCDMOD), p-adique (EEZGD) et heuristique (GCDHEU). Le premier est une adaptation de l'algorithme d'Euclide et s'adapte à des coefficients assez génériques. Les trois autres ont en commun d'évaluer une ou plusieurs variables du polynôme (dans ce dernier cas il est nécessaire de bien distinguer le cas de polynômes à plusieurs variables) et de reconstruire le pgcd par des techniques distinctes, la plupart du temps ces algorithmes fonctionnent seulement si les coefficients sont entiers.

Soit donc P et Q deux polynômes à coefficients dans un corps. Le pgcd de P et Q n'est défini qu'à une constante près. Mais lorsque les coefficients de P et Q sont dans un anneau euclidien comme par exemple $\mathbb{Z}$ ou $\mathbb{Z}[i]$, on appelle pgcd de P et Q un polynôme D tel que P/D et Q/D soient encore à coefficients dans l'anneau, et que D soit optimal, c'est-à-dire que si un multiple μD de D vérifie $P/\mu D$ et $Q/\mu D$ sont à coefficients dans l'anneau, alors μ est inversible. La première étape d'un algorithme de calcul de pgcd consiste donc à diviser par le pgcd des coefficients entiers de chaque polynôme.

Exemple : $P = 4X^2 - 4$ et $Q = 6X^2 + 12X + 6$. Le polynôme $X + 1$ est un pgcd de P et Q puisqu'il est de degré maximal divisant P et Q mais le pgcd de P et Q est $2(X + 1)$. Remarquons qu'avec notre définition $-2(X + 1)$ convient aussi. Par convention on appelera pgcd le polynôme ayant un coefficient dominant positif.

Définition : On appelle contenu $c(P)$ d'un polynôme P le pgcd des coefficients de P. On définit alors la partie primitive de P : $\text{pp}(P) = P/c(P)$. Si $c(P) = 1$, on dit que P est primitif. On montre que :

$$D = \text{pgcd}(P, Q) = \text{pgcd}(c(P), c(Q))\text{pgcd}(\text{pp}(P), \text{pp}(Q))$$

4.1 Le sous-résultant.

La première idée qui vient à l'esprit pour améliorer l'efficacité de l'algorithme d'Euclide consiste à éviter les fractions qui sont créées par les divisions euclidiennes. On utilise à cet effet la pseudo-division : au lieu de prendre le reste R de la division euclidienne du polynôme P par Q, on prend le reste de la division de $Pq^{\delta+1}$ par Q, où q désigne le coefficient dominant de Q et δ la différence entre le degré de P et de Q.

Exercice : En utilisant votre système de calcul formel préféré, calculez les restes intermédiaires générés dans l'algorithme d'Euclide lorsqu'on utilise la pseudo-division par exemple pour les polynômes $P(x) = (x+1)^7 - (x-1)^6$ et sa dérivée.

Une solution avec giac/xcas :

```
// -*- mode:C++ -*- a,b 2 polynomes -> pgcd de a et b
pgcd(a,b):={
 local P,p,Q,q,R,g,h,d;
 // convertit a et b en polynomes listes et extrait la partie primitive
 P:=symb2poly1(a);
```

```
 p:=lgcd(P); // pgcd des elements de la liste
 P:=P/p;
 Q:=symb2poly1(b);
 q:=lgcd(Q);
 Q:=Q/q;
 if (size(P)<size(Q)){ // echange P et Q
  R:=P; P:=Q; Q:=R;
 }
 // calcul du contenu du pgcd
 p:=gcd(p,q);
 g:=1;
 h:=1;
 while (size(Q)!=1){
  q:=Q[0]; // coefficient dominant
  d:=size(P)-size(Q);
  R:=rem(q^(d+1)*P,Q);
  if (size(R)==0) return(p*poly12symb(Q/lgcd(Q),x));
  P:=Q;
  Q:=R;
  // ligne suivante a decommenter pour prs
  // Q:=R/(g*h^d);
  print(Q);
  // ligne suivante a decommenter pour prs
  // g:=q; h:=q^d/h^(d-1);
 }
 return(p);
}
```

On s'aperçoit que les coefficients croissent de manière exponentielle. La deuxième idée qui vient naturellement est alors à chaque étape de rendre le reste primitif, donc de diviser R par le pgcd de ces coefficients. Cela donne un algorithme plus efficace, mais encore assez peu efficace car à chaque étape on doit calculer le pgcd de tous les coefficients, on peut imaginer le temps que cela prendra en dimension 1 et à fortiori en dimension supérieure. L'idéal serait de connaitre à l'avance une quantité suffisamment grande qui divise tous les coefficients du reste.

C'est ici qu'intervient l'algorithme du sous-résultant : après chaque pseudo-division euclidienne, on exhibe un coefficient "magique" qui divise les coefficients du reste. Ce coefficient n'est pas le pgcd mais il est suffisamment grand pour qu'on évite la croissance exponentielle des coefficients.

Algorithme du sous-résultant

Arguments : 2 polynômes P et Q primitifs. Valeur de retour : le pgcd de P et Q.

Pour calculer le coefficient "magique" on utilise 2 variables auxiliaires g et h initialisées a 1.

Boucle à effectuer tant que Q est non nul :

- on note $\delta =$degre(P)-degre(Q) et q le coefficient dominant de Q
- on effectue la division euclidienne (sans fraction) de $q^{\delta+1}P$ par Q, soit R le reste

– Si R est constant, on sort de l'algorithme en renvoyant 1 comme pgcd
– on recopie Q dans P puis $R/(gh^\delta)$ dans Q
– on recopie q dans g et $h^{1-\delta}q^\delta$ dans h.

Si on sort normalement de la boucle, Q est nul, on renvoie donc la partie primitive de P qui est le pgcd cherché.

Pour tester l'algorithme avec `xcas`, il suffit de décommenter les deux lignes `Q :=R/(g*h^d);` et `g :=q; h :=q^d/h^(d-1);` ci-dessus.

La preuve de l'algorithme est un peu longue et par ailleurs bien expliquée dans le 2ème tome de Knuth (The Art of Computer Programming, Semi-numerical Algorithms), on y renvoie donc le lecteur intéressé. L'idée générale (et l'origine du nom de l'algorithme) est de considérer la matrice de Sylvester des polynômes de départ P et Q (celle dont le déterminant est appelé résultant de P et Q) et de traduire les pseudo-divisions qui permettent de calculer les restes successifs du sous-résultant en opération de ligne sur ces matrices. On démontre alors que les coefficients de R divisés par gh^δ peuvent être interprétés comme des déterminants de sous-matrices de la matrice de Sylvester après réduction et c'est cela qui permet de conclure qu'ils sont entiers.

Par exemple, supposons que $P = R_0$, $Q = R_1$, R_2... diminuent de 1 en degré à chaque division (c'est le cas générique dans le déroulement de l'algorithme d'Euclide). Dans ce cas, $\delta = 1$, il s'agit par exemple de montrer que le reste R_3 de $Q = R_1$ par R_2 est divisible par le carré du coefficient dominant de $Q = R_1$. Voyons comment on obtient les coefficients de R_3 à partir de la matrice de Sylvester de P et Q. Prenons la sous-matrice constituée des 2 premières lignes de P et des 3 premières lignes de Q et réduisons-la sous forme échelonnée sans introduire de dénominateur.

$$\begin{pmatrix} p_n & p_{n-1} & p_{n-2} & p_{n-3} & \cdots \\ 0 & p_n & p_{n-1} & p_{n-2} & \cdots \\ q_{n-1} & q_{n-2} & q_{n-3} & q_{n-4} & \cdots \\ 0 & q_{n-1} & q_{n-2} & q_{n-3} & \cdots \\ 0 & 0 & q_{n-1} & q_{n-2} & \cdots \end{pmatrix}$$

On effectue $L_1 \leftarrow q_{n-1}L_1 - p_nL_3$ et $L_2 \leftarrow q_{n-1}L_2 - p_nL_4$, ce qui correspond à l'élimination du terme en x du quotient de P par Q

$$\begin{pmatrix} 0 & q_{n-1}p_{n-1} - p_nq_{n-2} & \cdots & \cdots & \cdots \\ 0 & 0 & q_{n-1}p_{n-1} - p_nq_{n-2} & \cdots & \cdots \\ q_{n-1} & q_{n-2} & q_{n-3} & q_{n-4} & \cdots \\ 0 & q_{n-1} & q_{n-2} & q_{n-3} & \cdots \\ 0 & 0 & q_{n-1} & q_{n-2} & \cdots \end{pmatrix}$$

on effectue ensuite

$$\begin{aligned} L_1 &\leftarrow q_{n-1}L_1 - (q_{n-1}p_{n-1} - p_nq_{n-2})L_4 \\ L_2 &\leftarrow q_{n-1}L_2 - (q_{n-1}p_{n-1} - p_nq_{n-2})L_5 \end{aligned}$$

ce qui correspond à l'élimination du terme constant du quotient de P par Q, on obtient

$$\begin{pmatrix} 0 & 0 & r_{2,n-2} & \cdots & \cdots \\ 0 & 0 & 0 & r_{2,n-2} & \cdots \\ q_{n-1} & q_{n-2} & q_{n-3} & q_{n-4} & \cdots \\ 0 & q_{n-1} & q_{n-2} & q_{n-3} & \cdots \\ 0 & 0 & q_{n-1} & q_{n-2} & \cdots \end{pmatrix}$$

si on enlève les lignes 3 et 4, et les colonnes 1 et 2, on obtient (après échanges de lignes) une sous-matrice de la matrice de Sylvester de Q et R_2

$$\begin{pmatrix} q_{n-1} & q_{n-2} & \cdots \\ r_{2,n-2} & \cdots & \cdots \\ 0 & r_{2,n-2} & \cdots \end{pmatrix}$$

On recommence les opérations de réduction de cette sous-matrice correspondant à la division euclidienne de Q par R_2, on obtient

$$\begin{pmatrix} 0 & 0 & r_{3,n-3} \\ r_{2,n-2} & \cdots & \cdots \\ 0 & r_{2,n-2} & \cdots \end{pmatrix}$$

puis après suppression des colonnes 1 et 2 et des lignes 2 et 3 la ligne des coefficients de R_3.

Supposons qu'on se limite dès le début de la réduction à ne garder que les colonnes 1 à 4 et une 5-ième colonne parmi les suivantes, on obtient à la fin de la réduction une matrice 1,1 qui contient un des coefficients de R_3 (selon le choix de la 5-ième colonne). Donc ce coefficient est égal au déterminant de la matrice 1,1 qui est égal, au signe près, au déterminant de la matrice 3,3 dont il est issu par notre réduction (en effet, dans la 2ième partie de la réduction, on a multiplié deux fois L_1 par $r_{2,n-2}$, mais on doit ensuite diviser le déterminant par $r_{2,n-2}^2$ pour éliminer les colonnes 1 et 2). Quant au déterminant de la matrice 3,3, il se déduit du déterminant de la matrice 5,5 par multiplication par q_{n-1}^4 (2 lignes ont été multipliées 2 fois par q_{n-1}) et division par q_{n-1}^2 (élimination des colonnes 1 et 2). Au final, tout coefficient de R_3 est égal au produit d'un déterminant 5,5 extrait de la matrice de Sylvester de P et Q par q_{n-1}^2, qui est justement le coefficient "magique" par lequel on divise le reste de $R_1 = Q$ par R_2 lors de l'algorithme du sous-résultant.

4.2 Le pgcd en une variable

.

4.2.1 Le pgcd heuristique.

On suppose ici que les coefficients sont entiers ou entiers de Gauss. **On peut donc se ramener au cas où les polynômes sont primitifs.**

L'idée consiste à évaluer P et Q en un entier z et à extraire des informations du pgcd g des entiers $P(z)$ et $Q(z)$. Il faut donc un moyen de remonter de l'entier g à un polynôme G tel que $G(z) = g$. La méthode consiste à écrire en base z l'entier g, avec une particularité dans les divisions euclidiennes successives on utilise le

reste symétrique (compris entre $-z/2$ et $z/2$). Cette écriture donne les coefficients d'un polynôme G unique. On extrait ensuite la partie primitive de ce polynôme G. Lorsque z est assez grand par rapport aux coefficients des polynômes P et Q, si $\text{pp}(G)$ divise P et Q, on va montrer que le pgcd de P et de Q est $D = \text{pp}(G)$.

On remarque tout d'abord que $d := D(z)$ divise g. En effet D divise P et Q donc pour tout entier (ou entier de Gauss) z, $D(z)$ divise $P(z)$ et $Q(z)$. Il existe donc une constante a telle que

$$g = ad$$

On a aussi $\text{pp}(G)$ divise D. Il existe donc un polynôme C tel que :

$$D = \text{pp}(G)C$$

Nous devons prouver que C est un polynôme constant. On suppose dans la suite que ce n'est pas le cas. Evaluons l'égalité précédente au point z, on obtient

$$d = \frac{g}{c(G)}C(z)$$

Finalement

$$1 = \frac{a}{c(G)}C(z)$$

La procédure de construction de G nous donne une majoration de ces coefficients par $|z|/2$, donc de $c(G)$ par $|z|/2$, donc $C(z)$ divise un entier de module plus petit que $|z|/2$, donc

$$|C(z)| \leqslant \frac{|z|}{2}$$

On considère maintenant les racines complexes $z_1, \ldots, z_n$ du polynôme C (il en existe au moins une puisqu'on a supposé C non constant). On a :

$$C(X) = c_n(X - z_1)\ldots.(X - z_n)$$

Donc, comme c_n est un entier (ou entier de Gauss) non nul, sa norme est supérieure ou égale à 1 et :

$$|C(z)| \geqslant \prod_{j=1}^{n}(|z| - |z_j|)$$

Il nous reste à majorer les racines de C pour minorer $|C(z)|$. Comme C divise D il divise P et Q donc les racines de C sont des racines communes à P et Q. On va appliquer le :

Lemme 1 *Soit x une racine complexe d'un polynôme* $P = a_nX^n + \ldots. + a_0$.
Alors

$$|x| < \frac{|P|}{|a_n|} + 1, |P| = \max_{0 \leqslant i \leqslant n}(|a_i|)$$

Application du lemme à $C(X)$: on a $1/|c_n| \leq 1$ donc si on a choisi z tel que $|z| \geqslant 2\min(|P|, |Q|) + 2$, alors pour tout j, $|z_j| < |z|/2$ donc

$$|C(z)| > \left(\frac{|z|}{2}\right)^n$$

qui contredit notre majoration de $|C(z)|$.

Théorème 1 *Soit P et Q deux polynômes à coefficients entiers. On choisit un entier z tel que $|z| \geqslant 2\min(|P|,|Q|)+2$, si la partie primitive du polynôme G reconstruit à partir du pgcd de $P(z)$etQ(z) par écriture en base z (avec comme reste euclidien le reste symétrique) divise P et Q alors c'est le pgcd de P et Q.*

Pour finir la démonstration du théorème, il nous faut encore montrer le lemme. On a

$$-a_n x^n = a_{n-1}x^{n-1} + \ldots . + a_0$$

Donc

$$|a_n||x|^n \leqslant |P|(1 + \ldots . + |x|^{n-1}) = |P|\frac{|x|^n - 1}{|x| - 1}$$

Ici on peut supposer que $|x| \geqslant 1$, sinon le lemme est démontré, donc $|x| - 1$ est positif et

$$|a_n|(|x| - 1) \leqslant |P|\frac{|x|^n - 1}{|x|^n} \Rightarrow |x| - 1 < \frac{|P|}{|a_n|}$$

Remarques

- Le théorème publié par Char, Geddes et Gonnet porte sur des coefficients entiers et c'est comme cela qu'il est utilisé par les systèmes de calcul formel (en commençant historiquement par Maple). Peu de systèmes l'utilisent pour les polynômes à coefficients entiers de Gauss. On peut d'ailleurs généraliser le théorème à d'autres types de coefficients, à condition d'avoir un anneau euclidien plongé dans $\mathbb{C}$ avec une minoration sur la valeur absolue des élements non nuls de l'anneau.
- Nous n'avons jusqu'à présent aucune certitude qu'il existe des entiers z tels que la partie primitive de G divise P et Q. Nous allons montrer en utilisant l'identité de Bézout que pour z assez grand c'est toujours le cas. Plus précisément, on sait qu'il existe deux polynômes U et V tels que

  $$PU + QV = D$$

  Attention toutefois, U et V sont à coefficients rationnels, pour avoir des coefficients entiers, on doit multiplier par une constante entière α, donc en évaluant en z on obtient l'existence d'une égalité à coefficients entiers

  $$P(z)u + Q(z)v = \alpha D(z)$$

  Donc le pgcd g de $P(z)$ et $Q(z)$ divise $\alpha D(z) = \alpha d$. Comme g est un multiple de d, on en déduit que $g = \beta d$, où β est un diviseur de α. Si on a choisi z tel que

  $$|z| > 2|D||\alpha|$$

  alors $|z| > 2|D||\beta|$ donc l'écriture symétrique en base z de g est $G = \beta D$. Donc la partie primitive de G est D, le pgcd de P et Q.

Exemple 1 *Si $P_0 = 6(X^2 - 1)$ et $Q_0 = 4(X^3 - 1)$.*

Le contenu de P_0 est 6, celui de Q_0 est 4.
On a donc pgcd des contenus = 2, $P = X^2 - 1, Q = X^3 - 1$. La valeur initiale de z est donc $2 * 1 + 2 = 4$. On trouve $P(4) = 15, Q(4) = 63$. Le pgcd entier de 15 et 63 est 3 que nous écrivons symétriquement en base 4 sous la forme $3 = 1 * 4 - 1$, donc $G = X - 1$, sa partie primitive est $X - 1$. On teste si $X - 1$ divise P et Q, c'est le cas, donc c'est le pgcd de P et Q et le pgcd de P_0 et Q_0 est $2(X - 1)$.

Algorithme gcdheu

En arguments deux polynômes P_0 et Q_0 à coefficients entiers ou entiers de Gauss. Retourne le pgcd de P_0 et Q_0 ou faux en cas d'échec.

1. Calculer le contenu de P_0 et Q_0. Vérifier que les coefficients sont entiers de Gauss sinon retourner faux.
2. Extraire la partie primitive P de P_0 et Q de Q_0, calculer le pgcd c des contenus de P_0 et Q_0
3. Déterminer $z = 2\min(|P|, |Q|) + 2$.
4. Début de boucle : initialisation du nombre d'essais à 1, test d'arrêt sur un nombre maximal d'essais, avec changement de z entre deux itérations (par exemple $z \leftarrow 2z$).
5. Calculer le pgcd g de $P(z)$ et $Q(z)$ puis son écriture symétrique en base z dont on extrait la partie primitive G.
6. Si Gnedivisepas$P$ passer à l'itération suivante. De même pour Q.
7. Retourner cG
8. Fin de la boucle
9. Retourner faux.

On remarque au passage qu'on a calculé le quotient de P par G et le quotient de Q par G lorsque la procédure réussit. On peut donc passer à la procédure gcdheu deux paramètres supplémentaires par référence, les deux polynômes que l'on affectera en cas de succès, ce qui optimise la simplification d'une fraction de 2 polynômes.

4.2.2 Le pgcd modulaire

On part du fait que si D est le pgcd de P et Q dans $\mathbb{Z}$ (ou $\mathbb{Z}[i]$) alors après réduction modulo un nombre premier n qui ne divise pas les coefficients dominants de P et Q, D divise le pgcd G de P et Q dans $\mathbb{Z}/n\mathbb{Z}$ (par convention, le pgcd dans $\mathbb{Z}/n\mathbb{Z}$ est normalisé pour que son coefficient dominant vaille 1). Comme on calcule G dans $\mathbb{Z}/n\mathbb{Z}$, les coefficients des restes intermédiaires de l'algorithme d'Euclide sont bornés, on évite ainsi la croissance exponentielle des coefficients. Il faudra ensuite reconstruire D à partir de G.

On remarque d'abord que si on trouve $G = 1$, alors P et Q sont premiers entre eux. En général, on peut seulement dire que le degré de G est supérieur ou égal au degré de D. En fait, le degré de G est égal au degré de D lorsque les restes de l'algorithme d'Euclide (calculé en effectuant des pseudo-divisions, cf. l'exercice 1) ont leur coefficient dominant non divisible par n. Donc plus n est grand, plus la probabilité est grande de trouver G du bon degré.

Dans la suite, nous allons déterminer une borne b à priori majorant les coefficients de D. On utilisera ensuite la même méthode que dans l'algorithme modulaire de recherche de racines évidentes : on multiplie G dans $\mathbb{Z}/n\mathbb{Z}$ par le pgcd dans $\mathbb{Z}$ des coefficients dominants p et q de P et Q. Soit $\tilde{D} = \text{pgcd}(p, q)G$ le résultat écrit en représentation symétrique. Si $n \geqslant b\text{pgcd}(p, q)$ et si G est du bon degré, on montre de la même manière que $D = \tilde{D}$. Comme on ne connait pas le degré de D, on est obligé de tester si $\tilde{D}$ divise P et Q. Si c'est le cas, alors $\tilde{D}$ divise D donc

$\tilde{D} = D$ puisque degre$(\tilde{D})$ = degre$(G) \geqslant$ degre(D). Sinon, n est un nombre premier malchanceux pour ce calcul de pgcd (degre$(G) \geqslant$ degre(D)), il faut essayer un autre premier.

Remarque : On serait tenté de dire que les coefficients de D sont bornés par le plus grand coefficient de P. C'est malheureusement faux, par exemple $(X+1)^2$ dont le plus grand coefficient est 2 divise $(X+1)^2(X-1)$ dont le plus grand coefficient (en valeur absolue) est 1.

Soit $P = \sum p_i X^i$ un polynôme à coefficients entiers. On utilise la norme euclidienne

$$|P|^2 = \sum |p_i|^2 \tag{1}$$

On établit d'abord une majoration du produit des racines de norme supérieure à 1 de P à l'aide de $|P|$. Ensuite si D est un diviseur de P, le coefficient dominant d de D divise le coefficient dominant p de P et les racines de D sont aussi des racines de P. On pourra donc déterminer une majoration des polynômes symétriques des racines de D et donc des coefficients de D.

Lemme 2 *Soit $A = \sum_{j=0}^{a} a_j X^j$ un polynôme et $\alpha \in \mathbb{C}$. Alors*

$$|(X-\alpha)A| = |(\overline{\alpha}X - 1)A|$$

Pour prouver le lemme 2, on développe les produits de polynômes. On pose $a_{-1} = a_{a+1} = 0$ et on note $\Re$ la partie réelle.

$$|(X-\alpha)A|^2 = \sum_{j=0}^{a+1} |a_{j-1} - \alpha a_j|^2 = \sum_{j=0}^{a+1} |a_{j-1}|^2 + |\alpha|^2 |a_j|^2 - 2\Re(a_{j-1}\overline{\alpha a_j})$$

$$|(\overline{\alpha}X - 1)A|^2 = \sum_{j=0}^{a+1} |\overline{\alpha} a_{j-1} - a_j|^2 = \sum_{j=0}^{a+1} |\alpha|^2 |a_{j-1}|^2 + |a_j|^2 - 2\Re(\overline{\alpha} a_{j-1}\overline{a_j})$$

Les deux donnent bien le même résultat.

Soit $P(X) = p\prod(X - \alpha_j)$ la factorisation de P sur $\mathbb{C}$. On introduit le polynôme

$$\tilde{P} = p \prod_{j/|\alpha_j| \geqslant 1} (X - \alpha_j) \prod_{j/|\alpha_j|<1} (\overline{\alpha_j} X - 1)$$

qui d'après le lemme a la même norme que P. La norme de P majore donc le coefficient constant de $\tilde{P}$ d'où :

$$\prod_{j/|\alpha_j| \geqslant 1} |\alpha_j| \leqslant \frac{|P|}{|p|} \tag{2}$$

On remarque que (2) reste vraie si on considère les racines δ_j de norme plus grande que 1 d'un diviseur D de P puisque le produit porte alors sur un sous-ensemble. On écrit maintenant l'expression des coefficients d_j de D à l'aide des racines δ_j de D :

$$|d_{m-j}| = |d| \left| \sum_{\text{choix de } j \text{ racines parmi les } m \text{ racines de } D} \prod_{\delta_k \in \text{racines choisies}} \delta_k \right|$$

Pour majorer $|d_{m-j}|$, on commence par majorer $|\delta_k|$ par $\beta_k = \max(1, |\delta_k|)$. On est donc ramené à majorer

$$\sigma_{j,m}(\beta) = \sum_{\text{choix de } j \text{ parmi } m \text{ valeurs } \beta_k} \prod_{\beta_k \in \text{choix}} \beta_k$$

avec pour hypothèse une majoration de $M = \prod_{k=1}^{m} \beta_k$ donnée par la relation (2). Pour cela, on cherche le maximum de $\sigma_{j,m}(\beta)$ sous les contraintes M fixé et $\beta_k \geqslant 1$.

On va montrer que le maximum ne peut être atteint que si l'un des $\beta_k = M$ (et tous les autres $\beta_k = 1$). Sinon, quitte à réordonner supposons que les β_k sont classés par ordre croissant. On a donc $\beta_{m-1} \neq 1$, on pose $\widetilde{\beta_k} = \beta_k$ pour $k \leqslant m-2$, $\tilde{\beta}_{m-1} = 1$ et $\tilde{\beta}_m = \beta_{m-1}\beta_m$. Comparons $\sigma_{j,m}(\beta)$ et $\sigma_{j,\text{nm}}(\tilde{\beta})$. Si le choix de j parmi m comporte $k = m-1$ et $k = m$, le produit est inchangé. Sinon on a la somme de deux produits, l'un contenant $k = m-1$ et l'autre $k = m$. On compare donc $B(\beta_{m-1} + \beta_m)$ et $B(1 + \beta_{m-1}\beta_m)$ avec $B = \prod_{\beta_k \in \text{resteduchoix}} \beta_k$. Comme

$$1 + \beta_{m-1}\beta_m \geqslant \beta_{m-1} + \beta_m$$

puisque la différence est le produit $(1 - \beta_m)(1 - \beta_{m-1})$ de deux nombres positifs, on arrive à la contradiction souhaitée.

Ensuite on décompose les choix de $\sigma_{m,j}$ en ceux contenant M et des 1 et ceux ne contenant que des 1, d'où la majoration

$$\sigma_{j,m}(\beta) \leqslant \begin{pmatrix} m-1 \\ j-1 \end{pmatrix} M + \begin{pmatrix} m-1 \\ j \end{pmatrix}$$

et finalement

$$|d_{m-j}| \leqslant |d| \left(\begin{pmatrix} m-1 \\ j-1 \end{pmatrix} \frac{|P|}{|p|} + \begin{pmatrix} m-1 \\ j \end{pmatrix} \right) \tag{3}$$

On peut en déduire une majoration indépendante de j sur les coefficients de D, en majorant $|d|$ par $|p|$ (puisque d divise p) et les coefficients binomiaux par 2^{m-1} (obtenue en développant $(1+1)^{m-1}$). D'où le

Théorème 2 *(Landau-Mignotte) Soit P un polynôme à coefficients entiers (ou entiers de Gauss) et D un diviseur de P de degré m. Si $|P|$ désigne la norme euclidienne du vecteur des coefficients de P et p le coefficient dominant de P alors les coefficients d_j de D satisfont l'inégalité*

$$|d_j| \leqslant 2^{m-1}(|P| + |p|) \tag{4}$$

Avec cette estimation, on en déduit que si n est un premier plus grand que

$$\min\left(2^{\text{degre}(P)-1}(|P| + |p|), 2^{\text{degre}(Q)-1}(|Q| + |q|)\right), \tag{5}$$

alors le pgcd trouvé dans $\mathbb{Z}/n\mathbb{Z}$ va se reconstruire en un pgcd dans $\mathbb{Z}$ si son degré est le bon.

Malheureusement la borne précédente est souvent très grande par rapport aux coefficients du pgcd et calculer dans $\mathbb{Z}/n\mathbb{Z}$ s'avèrera encore inefficace (surtout si

le pgcd est 1). Cela reste vrai même si on optimise un peu la majoration (5) en repartant de (3).

L'idée est donc de travailler modulo plusieurs nombres premiers plus petits et reconstruire le pgcd des 2 polynômes à coefficients entiers à partir des pgcd des polynômes dans $\mathbb{Z}/n\mathbb{Z}$ et du théorème des restes chinois. En pratique on prend des nombres premiers inférieurs à la racine carrée du plus grand entier hardware de la machine (donc plus petits que 2^{16} sur une machine 32 bits) ce qui permet d'utiliser l'arithmétique hardware du processeur sans risque de débordement.

Algorithme du PGCD modulaire en 1 variable :

En argument : 2 polynômes primitifs P et Q à coefficients entiers. Le résultat renvoyé sera le polynôme pgcd.

Variable auxiliaire : un entier N initialisé à 1 qui représente le produit des nombres premiers utilisés jusqu'ici et un polynôme H initialisé à 0 qui représente le pgcd dans $\mathbb{Z}/N\mathbb{Z}$.

Boucle infinie :

1. Chercher un nouveau nombre premier n qui ne divise pas les coefficients dominants p et q de P et Q
2. Calculer le pgcd G de P et Q dans $\mathbb{Z}/n\mathbb{Z}$. Si G=1, renvoyer 1.
3. Si $H = 0$ ou si le degré de G est plus petit que le degré de H, recopier G dans H et n dans N, passer à la 6ème étape
4. Si le degré de G est plus grand que celui de H passer à l'itération suivante
5. Si le degré de G est égal au degré de H, en utilisant le théorème des restes chinois, calculer un polynôme $\tilde{H}$ tel que $\tilde{H} = H$ modulo N et $\tilde{H} = G$ modulo n. Recopier $\tilde{H}$ dans H et nN dans N.
6. Ecrire pgcd$(p, q)H$ en représentation symétrique. Soit $\tilde{H}$ le résultat rendu primitif. Tester si $\tilde{H}$ divise P et Q. Si c'est le cas, renvoyer $\tilde{H}$, sinon passer à l'itération suivante.

Finalement on n'a pas utilisé b, la borne de Landau-Mignotte. On peut penser que l'étape 6 ne devrait être effectuée que lorsque N est plus grand que pgcd$(p, q)b$. En pratique, on effectue le test de l'étape 6 plus tôt parce que les coefficients du pgcd sont rarement aussi grand que b. Mais pour éviter de faire le test trop tôt, on introduit une variable auxiliaire H' qui contient la valeur de H de l'itération précédente et on ne fait le test que si $H' = H$ (ou bien sûr si on a dépassé la borne).

Remarque :

L'algorithme ci-dessus fonctionne également pour des polynômes à plusieurs variables.

Exemple 1 :

Calcul du pgcd de $(X + 1)^3(X - 1)^4$ et $(X^4 - 1)$. Prenons pour commencer $n = 2$. On trouve comme pgcd $X^4 + 1$ (en effet $-1 = 1$ donc on cherchait le pgcd de $(X + 1)^7$ et de $X^4 + 1 = (X + 1)^4$). On teste si $X^4 + 1$ divise P et Q, ce n'est pas le cas donc on passe au nombre premier suivant. Pour $n = 3$, on trouve $X^2 - 1$. Donc $n = 2$ n'était pas un bon nombre premier pour ce calcul de pgcd puisqu'on a trouvé un pgcd de degré plus petit. On teste si $X^2 - 1$ divise P et Q, c'est le cas ici donc on peut arrêter, le pgcd cherché est $X^2 - 1$.

Exemple 2 :

Calcul du pgcd de $(X+1)^3(X-1)^4$ et $(X^4-1)^3$. Pour $n=2$, on trouve un polynôme de degré 7. Pour $n=3$, on trouve X^6-1 donc $n=2$ était une mauvaise réduction. Comme X^6-1 ne divise pas P et Q, on passe à $n=5$. On trouve $X^6+2X^4-2X^2-1$. On applique le théorème des restes chinois qui va nous donner un polynôme dans $\mathbb{Z}/15\mathbb{Z}$. On cherche donc un entier congru à 2 modulo 5 et à 0 modulo 3, -3 est la solution (écrite en représentation symétrique), donc le polynôme modulo 15 est $X^6-3X^4+3X^2-1=(X^2-1)^3$. Ce polynôme divise P et Q, c'est donc le pgcd de P et de Q.

4.3 Le pgcd à plusieurs variables.

4.3.1 Le pgcd heuristique.

On suppose comme dans le cas à une variable que les polynômes sont primitifs, donc qu'on a simplifié les polynômes par le pgcd entier de leurs coefficients entiers.

Le principe est identique à celui du PGCD à 1 variable, on évalue les deux polynômes P et Q de k variables $X_1,\ldots,X_k$ en un $X_k=z$ et on calcule le pgcd g des 2 polynômes $P(z)$ et $Q(z)$ de $k-1$ variables. On remonte ensuite à un polynôme G par écriture symétrique en base z de g et on teste si pp(G) divise P et Q. Il s'agit à nouveau de montrer que si z est assez grand, alors pp(G) est le pgcd cherché. On sait que $d=D(z)$ divise g. Il existe donc un polynôme a de $k-1$ variables tel que $g=ad$. On sait aussi que pp(G) divise D, donc il existe un polynôme C de k variables tel que $D=C*\text{pp}(G)$. On évalue en z et on obtient $d=C(z)g/c(G)$, où $c(G)$ est un entier, donc

$$c(G)=a*C(z)$$

Comme $c(G)$ est un entier, a et $C(z)$ sont des polynômes constants. Comme précédemment, on a aussi $|C(z)|\leqslant|z|/2$ puisque $|c(G)|\leqslant|z|/2$.

- Premier cas : si C ne dépend que de la variable X_k. On continue le raisonnement comme dans le cas unidimensionnel.
- Deuxième cas : si C dépend d'une autre variable, par exemple X_1. On regarde le coefficient de plus haut degre de C par rapport a X_1. Ce coefficient divise le coefficient de plus haut degre de P et de Q par rapport a X_1. Comme $C(z)$ est constant, on en deduit que le coefficient de plus haut degre de P et Q par rapport a X_1 est divisible par X_k-z donc le coefficient de plus bas degre en X_k de ces coefficients de plus haut degre est divisible par z, ce qui contredit la majoration de ce coefficient.

En pratique, cet algorithme nécessite le calcul récursif de pgcd sans garantie de réussite. On l'évite donc s'il y a beaucoup de variables (la limite est par exemple de 5 pour MuPAD).

4.3.2 Le pgcd modulaire multivariables.

Ici, on travaille modulo $X_n-\alpha$, où $X_1,\ldots,X_n$ désignent les variables des polynômes. On considère donc deux polynômes P et Q comme polynômes de la variables X_n avec des coefficients dans $\mathbb{Z}[X_1,\ldots,X_{n-1}]$. On évalue en $X_n=\alpha$, on obtient deux polynômes en $n-1$ variables dont on calcule le pgcd (récursivement).

Il s'agit de reconstruire le pgcd par interpolation. Tout d'abord, on a une borne évidente sur le degré du pgcd par rapport à la variable X_n, c'est le minimum δ des degrés par rapport à X_n des polynômes P et Q. A première vue, il suffit donc d'évaluer les polynômes en $\delta + 1$ points α.

Il faut toutefois prendre garde aux mauvaises évaluations et à la normalisation des pgcd avant d'interpoler. En effet, si $D(X_1, \ldots., X_n)$ désigne le pgcd de P et Q et $G(X_1, \ldots., X_{n-1})$ le pgcd de $P(X_1, \ldots., X_{n-1}, \alpha)$ et de $Q(X_1, \ldots., X_{n-1}, \alpha)$, on peut seulement dire $D(X_1, \ldots., X_{n-1}, \alpha)$ divise G. Plusieurs cas sont donc possibles lorsqu'on évalue en un nouveau point α :

- l'un des degrés de G est plus petit que le degré du polynôme D' reconstruit par interpolation jusque là. Dans ce cas, toutes les évaluations qui ont conduit à reconstruire D' étaient mauvaises. Il faut recommencer l'interpolation à zéro ou à partir de G (si tous les degrés de G sont inférieurs ou égaux aux degrés du D' reconstruit).
- l'un des degrés de G est plus grand que le degré du D' reconstruit jusque là. Il faut alors ignorer α.
- Tous les degrés de G sont égaux aux degrés du D' reconstruit jusque là. Dans ce cas, G est un multiple entier du polynôme D' reconstruit jusque là et évalué en $X_n = \alpha$. Si on suppose qu'on a pu s'arranger pour que ce multiple soit 1, on ajoute le point α aux points d'évaluation précédents α_j en posant :

$$D' = D' + (G - D')\frac{\prod_{\alpha_j}(X_n - \alpha_j)}{\prod_{\alpha_j}(\alpha - \alpha_j)}$$

On voit que les mauvaises évaluations se détectent simplement par les degrés. Pour la normalisation, on utilise une petite astuce : au lieu de reconstruire lepgcd$D$, on va reconstruire un multiple du pgcd D (ce multiple appartiendra à $\mathbb{Z}[X_n]$). On voit maintenant P et Q comme des polynômes en $n-1$ variables $X_1, \ldots., X_{n-1}$ à coefficients dans $\mathbb{Z}[X_n]$. Alors lcoeff(D), le coefficient dominant de D (relativement à l'ordre lexicographique sur les variables $X_1, ..., X_{n-1}$), est un polynôme en X_n qui divise le coefficient dominant de P et de Q donc divise le coefficient dominant du pgcd des coefficients dominants de P et de Q. On va donc reconstruire le polynôme :

$$D' = D\frac{\Delta(X_n)}{\text{lcoeff}(D)(X_n)}, \Delta(X_n) = \text{pgcd}(\text{lcoeff}(P)(X_n), \text{lcoeff}(Q)(X_n))$$

c'est-à-dire D multiplié par un polynôme qui ne dépend que de X_n.

Revenons à G en un point α de bonne évaluation. C'est un multiple entier de $D(X_1, \ldots., X_{n-1}, \alpha)$:

$$G = \beta D(X_1, \ldots., X_{n-1}, \alpha)$$

Donc, comme polynômes de $X_1, ..., X_{n-1}$ à coefficients dans $\mathbb{Z}[X_n]$ ou dans $\mathbb{Z}$, lcoeff$(G) = \beta$lcoeff$(D)_{|X_n=\alpha}$. Comme lcoeff(D) divise $\Delta(X_n)$, il en est de même en $X_n = \alpha$ donc lcoeff(G) divise $\beta\Delta(\alpha)$. On en déduit que $\Delta(\alpha)G$ qui est divisible par $\Delta(\alpha)\beta$ est divisible par lcoeff(G). On va donc considérer le polynôme $\Delta(\alpha)G/$lcoeff(G) : ses coefficients sont entiers et son coefficient dominant est

$$\Delta(\alpha) = \text{lcoeff}(D'(X_1, \ldots., X_{n-1}, \alpha))$$

donc

$$\Delta(\alpha)G/\text{lcoeff}(G) = D'(X_1, \ldots., X_{n-1}, \alpha)$$

Algorithme du pgcd modulaire à plusieurs variables (interpolation dense) :

Arguments : 2 polynômes primitifs P et Q de n variables $X_1, \ldots., X_n$ à coefficients entiers. Renvoie le pgcd de P et Q.

1. Si $n = 1$, renvoyer le pgcd de P et Q en une variable.
2. Test rapide de pgcd trivial par rapport à X_n. On cherche des $n-1$-uplets α tels que $P(\alpha, X_n)$ et $Q(\alpha, X_n)$ soient de même degré que P et Q par rapport à la variable X_n. On calcule le pgcd G de ces 2 polynômes en une variable. Si le pgcd est constant, alors on retourne le pgcd des coefficients de P et Q.
3. On divise P et Q par leur contenu respectifs vu comme polynômes en $X_1, \ldots., X_{n-1}$ à coefficients dans $\mathbb{Z}[X_n]$, on note $C(X_n)$ le pgcd des contenus. On calcule aussi le pgcd $\Delta(X_n)$ des coefficients dominants de P et de Q.
4. On initialise D' le pgcd reconstruit à 0, $I(X_n)$ le polynôme d'interpolation à 1, $\delta = (\delta_1, ..., \delta_{n-1})$ la liste des degrés partiels du pgcd par rapport à $X_1, \ldots., X_{n-1}$ au minimum des degrés partiels de P et Q par rapport à $X_1, \ldots., X_{n-1}$, e le nombre d'évaluation à 0 et E l'ensemble des points d'interpolation à la liste vide.
5. Boucle infinie :
   - Faire α=entier aléatoire n'appartenant pas à E jusqu'à ce que

$$\text{degre}(P(X_1, \ldots., X_{n-1}, \alpha))=\text{degre}_{X_n}(P(X_1, \ldots., X_n)$$
$$\text{degre}(Q(X_1, \ldots., X_{n-1}, \alpha)) = \text{degre}_{X_n}(Q(X_1, \ldots., X_n))$$

   - Calculer le pgcd $G(X_1, \ldots., X_{n-1})$ en $n-1$ variables de $P(X_1, \ldots., X_{n-1}, \alpha)$ et $Q(X_1, \ldots., X_{n-1}, \alpha)$.
   - Si $\text{degre}(G)_i < \delta_i$ pour un indice au moins. Si $\text{degre}(G) \leqslant \delta$, on pose $\delta = \text{degre}(G)$, $D' = G\frac{\Delta(\alpha)}{\text{lcoeff}(G)}$, $I = X_n - \alpha$, $e = 1$ et $E = [\alpha]$, sinon on pose $\delta = \min(\delta, \text{degre}(G))$, $D' = 0, I = 1, e = 0, E = []$. On passe à l'itération suivante.
   - Si $\text{degre}(G) > \delta$, on passe à l'itération suivante.
   - Si $\text{degre}(G) = \delta$, on interpole :
     - $G := G\frac{\Delta(\alpha)}{\text{lcoeff}(G)}$
     - $D' := D' + \frac{I(X_n)}{\prod_{\alpha_j \in E}(\alpha - \alpha_j)}(G - D'(X_1, \ldots., X_{n-1}, \alpha))$
     - $I := I * (X_n - \alpha)$
     - $e := e + 1$ et ajouter α à E
     - Si e est strictement plus grand que le minimum des degrés partiels de P et Q par rapport à X_n, on pose $\tilde{D}$ la partie primitive de D'(vu comme polynôme à coefficients dans $\mathbb{Z}[X_n]$), on teste si P et Q sont divisibles par $\tilde{D}$, si c'est le cas, on renvoie $D = C(X_n)\tilde{D}$

On observe que dans cet algorithme, on fait le test de divisibilite de $\tilde{D}$ par P et Q. En effet, même après avoir évalué en suffisamment de points, rien n'indique que tous ces points sont des points de bonne évaluation. En pratique cela reste extrêmement improbable. En pratique, on teste la divisibilité plus tôt, dès que D' n'est pas modifié par l'ajout d'un nouveau point à la liste des α_j.

Il existe une variation de cet algorithme, appelé SPMOD (sparse modular), qui suppose que seuls les coefficients non nuls du pgcd en $n-1$ variables sont encore non nuls en n variables (ce qui a de fortes chances d'être le cas). L'étape d'interpolation est alors remplacée par la résolution d'un sous-système d'un système de Vandermonde. Cette variation est intéressante si le nombre de coefficients non nuls en $n-1$ variables est petit devant le degré. Si elle échoue, on revient à l'interpolation dense.

Notons enfin qu'on peut appliquer cette méthode lorsque les coefficients de P et Q sont dans $\mathbb{Z}/n\mathbb{Z}$ mais il faut alors vérifier qu'on dispose de suffisamment de points d'interpolation. Ce qui en combinant avec l'algorithme modulaire à une variable donne un algorithme doublement modulaire pour calculer le pgcd de 2 polynômes à coefficients entiers. C'est cette méthode qu'utilise par exemple MuPAD (en essayant d'abord SPMOD puis l'interpolation dense).

Exemple :

Dans cet exemple, on donne F et G sous forme factorisée, le but étant de faire comprendre l'algorithme. En utilisation normale, on n'exécuterait cet algorithme que si F et G étaient développés.

$P=((x+1)y+x^2+1)(y^2+xy+1), Q=((x+1)y+x^2+1)(y^2-xy-1)$.

Prenons x comme variable X_1 et y comme variable X_2. Les coefficients dominants de P et Q sont respectivement y et $-y$ donc $\Delta = y$.

En $y=0$, $P(x,0)=x^2+1$ n'est pas du bon degré.

En $y=1$, $P(x,1)=(x+x^2+2)(x+2)$ et $Q(x,1)=(x+x^2+2)(-x)$ sont du bon degré. Leur pgcd est $G=x^2+x+2$, $\Delta(1)=1$, donc $D'=x^2+x+1$. On teste la divisibilité de P par D', le teste échoue.

En $y=2$, $P(x,2)=(x^2+2x+3)(2x+5)$ et $Q(x,2)=(x^2+2x+3)(-2x+3)$ donc $G=x^2+2x+3$, $\Delta(2)=2$. On interpole :

$$D'=x^2+x+2+\frac{y-1}{2-1}(2(x^2+2x+3)-(x^2+x+2))=y(x^2+3x+4)-(2x+2)$$

On teste la divisibilité de P par D', le test échoue.

En $y=3$, $P(x,3)=(x^2+3x+4)(3x+10)$ et $Q(x,3)=(x^2+3x+4)(-3x+8)$ donc $G=x^2+3x+4$, $\Delta(3)=3$. On interpole :

$$\begin{aligned} D' &= y(x^2+3x+4)-(2x+2)+ \\ &\quad \frac{(y-2)(y-1)}{(3-2)(3-1)}\left(3(x^2+3x+4)-(3(x^2+3x+4)-(2x+2))\right) \end{aligned}$$

donc

$$D'=y(x^2+3x+4)-(2x+2)+\frac{(y-2)(y-1)}{2}(-2x-2)=x^2y+xy^2+y^2+y$$

On divise D' par son contenu et on trouve $x^2+xy+y+1$ qui est bien le pgcd de P et Q.

4.3.3 EZGCD.

Il s'agit d'une méthode p-adique. On évalue toutes les variables sauf une, on calcule le pgcd en une variable et on remonte au pgcd variable par variable

(EEZGCD) ou toutes les variables simultanément (EZGCD) par un lemme de Hensel. Il semble qu'il est plus efficace de remonter les variables séparément.

Soit donc F et G deux polynômes primitifs dépendant des variables $X_1, \ldots, X_n$ de pgcd D, on fixe une des variables qu'on appelera X_1 dans la suite. Soient lcoeff(F) et lcoeff(G) les coefficients dominants de F et G par rapport à X_1. On évalue F et G en un $n-1$ uplet b tel que le degré de F et G par rapport à X_1 soit conservé après evaluation en b. On suppose que $D_b(X_1) = \text{pgcd}(F(b), G(b))$ a le même degré que $D(b)$. On a donc l'égalité :

$$(F * \text{lcoeff}(F))(b) = \left(D_b \frac{\text{lcoeff}(F(b))}{\text{lcoeff}(D_b)}\right) * \left(\frac{F(b)}{D_b} \frac{\text{lcoeff}(F)(b)}{\text{lcoeff}(\frac{F(b)}{D_b})}\right)$$

et de même en remplaçant F par G.

Pour pouvoir lifter cette égalité (c'est-à-dire généraliser à plusieurs variables), il faut que D_b et $\frac{F(b)}{D_b}$ soient premiers entre eux. Sinon, on peut essayer de lifter l'égalité analogue avec G. En général, on montre qu'il existe un entier j tel que D_b et $\frac{F(b)+jG(b)}{D_b}$ soient premiers entre eux. En effet, sinon au moins un des facteurs irréductibles de D_b va diviser $\frac{F(b)+jG(b)}{D_b}$ pour deux valeurs distinctes de j et va donc diviser à la fois $\frac{F(b)}{D_b}$ et $\frac{G(b)}{D_b}$ en contradiction avec la définition de $D_b = \text{pgcd}(F(b), G(b))$. On lifte alors l'égalité obtenue en remplaçant F par $(F + kG)$ ci-dessus. Dans la suite, on suppose qu'on peut prendre $j = 0$ pour alléger les notations.

On va aussi supposer que $b = 0$. Sinon, on fait un changement d'origine sur les polynômes F et G pour que $b = 0$ convienne, on calcule le pgcd et on lui applique la translation d'origine opposée.

On adopte ensuite la notation suivante : si k est un entier, on dit qu'un polynôme P est un $O(k)$ si la valuation de P vu comme polynôme en $X_2, \ldots, X_n$ à coefficients dans $\mathbb{Z}[X_1]$ est supérieure ou égale à k, ou de manière équivalente si

$$P(X_1, hX_2, \ldots, hX_n) = O_{h\to 0}(h^k)$$

L'égalité à lifter se réécrit donc :

$$F\text{lcoeff}(F) = P_0 Q_0 + O(1)$$

où $P_0 = D_b \frac{\text{lcoeff}(F(b))}{\text{lcoeff}(D_b)}$ et $Q_0 = \frac{F(b)}{D_b} \frac{\text{lcoeff}(F)(b)}{\text{lcoeff}(\frac{F(b)}{D_b})}$ sont premiers entre eux et de degré 0 par rapport aux variables $X_2, \ldots, X_n$. Cherchons $P_1 = O(1)$ et $Q_1 = O(1)$ de degré 1 par rapport aux variables $X_2, \ldots, X_n$ tels que

$$F\text{lcoeff}(F) = (P_0 + P_1)(Q_0 + Q_1) + O(2)$$

Il faut donc résoudre

$$F\text{lcoeff}(F) - P_0 Q_0 = P_0 Q_1 + Q_0 P_1 + O(2)$$

On peut alors appliquer l'identité de Bézout qui permet de déterminer des polynômes P_1 et Q_1 satisfaisant l'égalité ci-dessus (avec comme reste $O(2)$ nul) puisque P_0 et Q_0 sont premiers entre eux. De plus, on choisit P_1 et Q_1 tels que $\text{degre}_{X_1} P_1 \leqslant \text{degre}_{X_1}(F) - \text{degre}(Q_0) = \text{degre}(P_0)$ et $\text{degre}_{X_1}(Q_1) \leqslant \text{degre}(Q_0)$ et $\text{lcoeff}_{X_1}(P_0 +$

$P_1) + O(2) = \text{lcoeff}_{X_1}(Q_0 + Q_1) + O(2) = \text{lcoeff}_{X_1}(F)$. On tronque ensuite P_1 et Q_1 en ne conservant que les termes de degré 1 par rapport à $X_2, \ldots, X_n$.

On trouve de la même manière par récurrence P_k et Q_k homogènes de degré k par rapport à $X_2, \ldots, X_k$, de degré par rapport à X_1 respectivement inférieur aux degrés de Q_0 et de P_0 et tels que

$$F\text{lcoeff}(F) = (P_0 + \ldots. + P_k)(Q_0 + \ldots. + Q_k) + O(k+1) \qquad (6)$$

et $\text{lcoeff}(F) = \text{lcoeff}_{X_1}(P_0 + \ldots. + P_k) + O(k+1) = \text{lcoeff}_{X_1}(Q_0 + \ldots. + Q_k) + O(k+1)$.

Si on est bien en un point de bonne évaluation et si k est plus grand que le degré total (par rapport aux variables $X_2, \ldots, X_n$) du polynôme $F\text{lcoeff}(F)$ on va vérifier que $P_0 + \ldots. + P_k = D\frac{\text{lcoeff}(F)}{\text{lcoeff}(D)}$. En effet, si on a deux suites de polynômes P et P' et Q et Q' satisfaisant (6) avec les même termes de degré zéro P_0 et Q_0, alors en prenant la différence, on obtient :

$$(P_0+P_1 \ldots +P_k)(Q_0+Q_1 \ldots +Q_k) = (P_0+P'_1 \ldots +P'_k)(Q_0+Q'_1 \ldots +Q'_k)+O(k+1)$$

On égale alors les termes homogènes de degré j, pour $j = 1$, on obtient $P_0(Q_1 - Q'_1) = Q_0(P_1 - P'_1)$, donc Q_0 divise $Q_1 - Q'_1$ qui est de degré strictement inférieur au degré de Q_0 par rapport à X_1 (car on a l'inégalité large et les termes de plus haut degré sont égaux), donc $Q_1 = Q'_1$ et $P_1 = P'_1$. On montre de la même manière que $Q_j = Q'_j$ et $P_j = P'_j$. L'écriture est donc unique, c'est donc l'écriture en polynôme homogène de degré croissant de $D\frac{\text{lcoeff}(F)}{\text{lcoeff}(D)}$ que l'on reconstruit.

Cet algorithme permet donc de reconstruire D, il suffit de tester à chaque étape si $P_0 + \ldots. + P_k$ divise $F\text{lcoeff}(F)$. On appelle cette méthode de remontée lemme de Hensel linéaire. Il existe une variante dite lemme de Hensel quadratique qui consiste à passer de $O(k)$ à $O(2k)$. Elle nécessite toutefois un calcul supplémentaire, celui de l'identité de Bézout à $O(2k)$ près pour les polynômes $P_0 + \ldots. + P_{k-1}$ et $Q_0 + \ldots. + Q_{k-1}$. Ce calcul se fait également par lifting.

Algorithme EZGCD (Hensel linéaire)

Arguments : 2 polynômes F et G à coefficients entiers et primitifs. Renvoie le pgcd de F et G ou false.

1. Evaluer F et G en $(X_2, \ldots, X_n) = (0, \ldots, 0)$, vérifier que les coefficients dominants de F et de G ne s'annulent pas. Calculer le pgcd D_b de $F(0)$ et de $G(0)$. Prendre un autre point d'évaluation au hasard qui n'annule pas les coefficients dominants de F et de G et vérifier que le pgcd a le même degré que D_b. Sinon, renvoyer false (on peut aussi faire une translation d'origine de F et de G en un autre point mais cela diminue l'efficacité de l'algorithme).
2. On note lcF et lcG les coefficients dominants de F et de G par rapport à X_1.
3. Si $\text{degre}(F) \leqslant \text{degre}(G)$ et $\text{degre}(D_b) = \text{degre}(G)$ et F divise G renvoyer F
4. Si $\text{degre}(G) < \text{degre}(F)$ et $\text{degre}(D_b) = \text{degre}(F)$ et G divise F renvoyer G
5. Si $\text{degre}(F) = \text{degre}(D_b)$ ou si $\text{degre}(G) = \text{degre}(D_b)$ renvoyer false
6. Boucle infinie sur j entier initialisé à 0, incrémenté de 1 à chaque itération : si $\text{pgcd}(D_b, \frac{F(0)+jG(0)}{D_b}) = C$ constant, alors arrêter la boucle

7. Lifter l'égalité $(F+jG)(\text{lcF}+j\text{lcG})(0) = \left(D_b \frac{(\text{lcF}+j\text{lcG})(0)}{\text{lcoeff}(D_b)}\right) * \ldots$. par remontée de Hensel linéaire ou quadratique. Si le résultat est false, renvoyer false. Sinon renvoyer le premier polynôme du résultat divisé par son contenu vu comme polynôme en X_1 à coefficients dans $\mathbb{Z}[X_2, \ldots, X_n]$.

Remontée de Hensel linéaire :

Arguments : F un polynôme, lcF=lcoeff(F) son coefficient dominant, P_0 un facteur de $F(0)$ ayant comme coefficient dominant lcF(0) et dont le cofacteur Q_0 est premier avec P_0.

Renvoie deux polynômes P et Q tels que FlcF $= PQ$ et $P(0) = P_0$ et lcoeff(P) = lcoeff(Q) = lcF.

1. Soit $G = F$lcF, , $Q_0 = G(0)/P_0$, $P = P_0$, $Q = Q_0$.
2. Déterminer les deux polynômes U et V de l'identité de Bézout (tels que $P_0U + Q_0V = d$ où d est un entier).
3. Boucle infinie avec un compteur k initialisé à 1, incrémenté de 1 à chaque itération
   - Si $k >$ degre$_{X_2,\ldots,X_n}(G)$, renvoyer false.
   - Si P divise G, renvoyer P et G/P.
   - Soit $H = G - PQ = O(k)$. Soit $u = U\frac{H}{d}$ et $v = V\frac{H}{d}$, on a $P_0u + Q_0v = H$
   - Remplacer v par le reste de la division euclidienne de v par P_0 et u par le reste de la division euclidienne de u par Q_0. La somme des deux quotients est égale au quotient euclidien de H par P_0Q_0, c'est-à-dire au coefficient dominant de H divisé par le produit des coefficients dominants de P_0 et Q_0 (qui sont égaux) donc on a l'égalité :

     $$P_0u + Q_0v = H - \frac{\text{lcoeff}(H)}{\text{lcoeff}(P_0)^2} P_0Q_0$$

   - Soit $\alpha = (\text{lcoeff}(F) - \text{lcoeff}(P))/\text{lcoeff}(P_0)$ et $\beta = (\text{lcoeff}(F) - \text{lcoeff}(Q))/\text{lcoeff}(P_0)$. On ajoute αP_0 à v, ainsi lcoeff$(P+v)$ = lcoeff$(F) + O(k+1)$ et βQ_0 à u, ainsi lcoeff$(Q+u)$ = lcoeff$(F) + O(k+1)$
     Remarque : on montre alors que $\alpha + \beta = \frac{\text{lcoeff}(H)}{\text{lcoeff}(P_0Q_0)} + O(k+1)$ donc $P_0u + Q_0v = H + O(k+1)$ en utilisant les propriétés :

     $$\text{lcoeff}(F) = \text{lcoeff}(P) + O(k) = \text{lcoeff}(Q) + O(k) = \text{lcoeff}(P_0) + O(1)$$

   - Réduire u et v en éliminant les termes de degré strictement supérieur à k par rapport à $X_2, \ldots, X_n$. S'il reste un coefficient non entier, renvoyer false
   - Remplacer P par $P + v$ et Q par $Q + u$, passer à l'itération suivante.

Exemple :

$F = ((x+1)y + x^2 + 1)(y^2 + xy + 1), G = ((x+1)y + x^2 + 1)(y^2 - xy - 1)$

On a $F(0, y) = (y+1)(y^2+1)$ et $G(0, y) = (y+1)(y^2-1)$, le pgcd est donc $D_b = (y+1)$. On remarque que D_b est premier avec le cofacteur de F mais pas avec le cofacteur de G. Si on évalue en un autre point, par exemple $x = 1$, on trouve un pgcd D_1 de même degré, donc 0 est vraissemblablement un bon point d'évaluation (ici on en est sûr puisque le pgcd de F et G se calcule à vue...). On a

lcoeff$(F) = x+1$, on va donc lifter $G = ((x+1)y+x^2+1)(y^2+xy+1)(x+1) = PQ$ où $P_0 = (y+1)$ et $Q_0 = (y^2+1)$.

On calcule les polynômes de l'identité de Bézout $U = (1-y)$ et $V = 1$ avec $d = 2$, puis à l'ordre $k = 1$:

$$H = G - P_0Q_0 = (2y^3 + 2y^2 + 3y + 1)x + O(2)$$

donc $u =$ reste$(UH/d, Q_0) = xy$ et $v =$ reste$(VH/d, P_0) = -x$.

Donc $Q_1 = xy+\alpha Q_0$ avec $\alpha = (x+1-1)/$lcoeff$(P_0) = x$ et $Q_0+Q_1 = (y^2+1)(x+1)+xy$. De même, $P_1 = -x+\beta P_0$, avec $\beta = (x+1-1)/$lcoeff$(P_0) = x$ donc $P_0 + P_1 = (y+1)(x+1) - x$. On remarque que $P_0 + P_1$ et $Q_0 + Q_1$ sont bien à $O(2)$ près les facteurs de Flcoeff(F) :

$$P = (x+1)y+x^2+1 = P_0+P_1+O(2),\ Q = (x+1)(y^2+xy+1) = Q_0+Q_1+O(2)$$

Une deuxième itération est nécessaire. On calcule

$$H = G - (P_0 + P_1)(Q_0 + Q_1) = (2y^2 + y + 1)x^2 + O(3)$$

puis reste$(UH/d, Q_0) = yx^2$ et reste$(VH/d, P_0) = x^2$. Ici les coefficients α et β sont nuls car lcoeff(F) n'a pas de partie homogène de degré 2. On trouve alors $P = P_0 + P_1 + P_2$ et $Q = Q_0 + Q_1 + Q_2$. Pour calculer le pgcd, il suffit de calculer la partie primitive de P vu comme polynôme en y, ici c'est encore P car le contenu de P est 1 (remarque : pour Q le contenu est $x + 1$).
On trouve donc P comme pgcd.

4.4 Quel algorithme choisir ?

Il est toujours judicieux de faire une évaluation en quelques $n-1$ uplets pour traquer les pgcd triviaux. (E)EZGCD sera efficace si (0,...,0) est un point de bonne évaluation et si le nombre de remontées nécessaires pour le lemme de Hensel est petit donc pour les pgcd de petit degré, GCDMOD est aussi efficace si le degré du pgcd est petit. Le sous-résultant est efficace pour les pgcd de grand degré car il y a alors peu de divisions euclidiennes à effectuer et les coefficients n'ont pas trop le temps de croitre. SPMOD est intéressant pour les polynômes creux de pgcd non trivial creux. GCDHEU est intéressant pour les problèmes relativement petits.

Avec des machines multiprocesseurs, on a probablement intérêt à lancer en parallèle plusieurs algorithmes et à s'arrêter dès que l'un deux recontre le succès.

4.5 Pour en savoir plus.

Parmi les références citées dans le premier article, ce sont les livres de Knuth, H. Cohen, et Davenport-Siret-Tournier qui traitent des algorithmes de pgcd. On peut bien sûr consulter le source de son système de calcul formel lorsqu'il est disponible :

- pour MuPAD sur un système Unix, depuis le répertoire d'installation de MuPAD (en général `/usr/local/MuPAD`) après avoir désarchivé le fichier `lib.tar` du répertoire `share/lib` par la commande
  `cd share/lib && tar xvf lib.tar`
  on trouve les algorithmes de calcul de PGCD dans le répertoire
  `share/lib/lib/POLYLIB/GCD`

– Pour l'algorithme EZGCD, je me suis inspiré de l'implémentation de Singular (logiciel libre disponible à `www.singular.uni-kl.de`)

Sur le web on trouve quelques articles en lignes sur le sujet en cherchant les mots clefs GCDHEU, EZGCD, SPMOD sur un moteur de recherche, il y a par exemple une description un peu différente du pgcd heuristique sur :
`www.inf.ethz.ch/personal/gonnet/CAII/HeuristicAlgorithms/node1.html`
et un article de comparaison de ces algorithmes par Fateman et Liao (dont la référence bibliographique est Evaluation of the heuristic polynomial GCD. in : ISSAC pages 240–247, 1995). Quelques autres références :

– K.O.Geddes et al "Alg. for Computer Algebra", Kluwer 1992.
– pour GCDHEU Char, Geddes, Gonnet, Gcdheu : Heuristic polynomial gcd algorithm based on integer gcd computation, in : Journal of Symbolic Computation, 7 :31–48, 1989.
– pour SPMOD "Probabilistic Algorithms for Sparse Polynomials", in : Symbolic & Algebraic Comp. (Ed E.W.Ng), Springer 1979, pp216,

5 Le résultant

Il s'agit d'un point de vue d'algèbre linéaire sur le PGCD. Considérons deux polynômes A et B de degrés p et q et de pgcd D et l'identité de Bézout correspondante :

$$AU + BV = D \tag{7}$$

avec degré$(U) < q$ et degré$(V) < p$. Imaginons qu'on cherche U et V en oubliant qu'il s'agit d'une identité de Bézout, en considérant simplement qu'il s'agit d'un problème d'algèbre linéaire de $p + q$ équations (obtenues en développant et en identifiant chaque puissance de X de 0 à $p + q - 1$) à $p + q$ inconnues (les p coefficients de V et les q coefficients de U) On sait que A et B sont premiers entre eux si et seulement si ce problème d'algèbre linéaire a une solution pour $D = 1$. Donc si le déterminant du système est non nul, alors A et B sont premiers entre eux. Réciproquement si A et B sont premiers entre eux, le système a une solution unique non seulement avec comme second membre 1 mais avec n'importe quel polynôme de degré inférieur $p + q$, donc le déterminant du système est non nul.

Définition :
On appelle résultant de A et B le déterminant de ce système (7). Il s'annule si et seulement si A et B ne sont pas premiers entre eux (ont au moins une racine commune). On appelle matrice de Sylvester la transposée de la matrice du système (les inconnues étant par ordre décroissant les coefficients de U et V)

$$M(A,B) = \begin{pmatrix} a_a & a_{a-1} & \dots & \dots & a_0 & 0 & \dots & 0 \\ 0 & a_a & \dots & \dots & a_1 & a_0 & \dots & 0 \\ \vdots & & & & & & & \vdots \\ 0 & 0 & \dots & & & & & a_0 \\ b_b & b_{b-1} & \dots & b_0 & 0 & 0 & \dots & 0 \\ \vdots & & & & & & & \vdots \\ 0 & 0 & \dots & & & & & b_0 \end{pmatrix}$$

(cette matrice contient b =degré(B) lignes de coefficients du polynôme A et a =degré(A) lignes de coefficients du polynôme B)

Lien avec l'algorithme du sous-résultant (calcul de PGCD)

On peut calculer le déterminant avec la suite des restes de divisions euclidiennes de la manière suivante, on part de la pseudo-division de A par B :

$$b_b^{a-b+1}A = BQ + R$$

on effectue alors sur chaque ligne contenant les coefficients de A la manipulation de ligne correspondante, c'est-à-dire multiplier la ligne par b_b^{a-b+1} et soustraire (q_0 fois la ligne de B terminant dans la même colonne+q_1 fois la ligne de B terminant une colonne avant+...). Toutes les lignes contenant les coefficients de A ont été remplacées par des lignes contenant les coefficients de R. Ces lignes contiennent k zéros initiaux avec $k \geq 1$, ce qui permet de réduire le déterminant à celui de la matrice de Sylvester de R et B (à un coefficient multiplicatif près qui vaut b_b^k par rapport au précédent donc $b_b^{k-b(a-b+1)}$ par rapport au déterminant de départ). On échange ensuite R et B ce qui change éventuellement le signe et on continue en faisant les divisions euclidiennes de l'algorithme du sous-résultant (cf. Knuth où on utilise la matrice de Sylvester pour prouver que l'algorithme du sous-résultant est correct). Rappelons que le sous-résultant définit les suites A_k ($A_0 = A, A_1 = B$), d_k le degré de A_k, $\delta_k = d_k - d_{k+1}$, g_k ($g_0 = 1$, si $k \neq 0$, g_k coefficient dominant de A_k) h_k ($h_0 = 1$, $h_{k+1} = h_k^{1-\delta_k} g_{k+1}^{\delta_k}$) et

$$g_k^{\delta_{k-1}+1} A_{k-1} = A_k Q_{k+1} + g_{k-1} h_{k-1}^{\delta_{k-1}} A_{k+1}$$

Théorème 3 *Le résultant est égal au signe près au coefficient h_k où k correspond au reste A_k constant (en supposant que le résultant soit non nul).*

Preuve

La transcription de l'égalité précédente sur les résultants donne par la méthode ci-dessus :

$$\begin{aligned} g_k^{(\delta_{k-1}+1)d_k} \text{Res}(A_{k-1}, A_k) &= g_k^{d_{k-1}-d_{k+1}} \text{Res}(g_{k-1} h_{k-1}^{\delta_{k-1}} A_{k+1}, A_k) \\ &= g_k^{d_{k-1}-d_{k+1}} (g_{k-1} h_{k-1}^{\delta_{k-1}})^{d_k} \text{Res}(A_{k+1}, A_k) \end{aligned}$$

On en déduit que :

$$\frac{\text{Res}(A_{k-1}, A_k)}{g_{k-1}^{d_k} h_{k-1}^{d_{k-1}-1}} = g_k^{d_{k-1}-d_{k+1}-(\delta_{k-1}+1)d_k} h_{k-1}^{\delta_{k-1}d_k+1-d_{k-1}} \text{Res}(A_{k+1}, A_k)$$

On observe que :

$$h_{k-1}^{\delta_{k-1}d_k+1-d_{k-1}} = h_{k-1}^{(\delta_{k-1}-1)(d_k-1)} = \left(h_{k-1}^{\delta_{k-1}-1}\right)^{d_k-1} = \left(\frac{g_k^{\delta_{k-1}}}{h_k}\right)^{d_k-1}$$

donc :

$$\begin{aligned} \frac{\text{Res}(A_{k-1}, A_k)}{g_{k-1}^{d_k} h_{k-1}^{d_{k-1}-1}} &= g_k^{d_{k-1}-d_{k+1}-(\delta_{k-1}+1)d_k} \left(\frac{g_k^{\delta_{k-1}}}{h_k}\right)^{d_k-1} \text{Res}(A_{k+1}, A_k) \\ &= g_k^{d_{k-1}-d_{k+1}-d_k-\delta_{k-1}} \left(\frac{1}{h_k}\right)^{d_k-1} \text{Res}(A_{k+1}, A_k) \\ &= \frac{\text{Res}(A_{k+1}, A_k)}{g_k^{d_{k+1}} h_k^{d_k-1}} \end{aligned}$$

Donc en valeur absolue

$$|\frac{\text{Res}(A_0, A_1)}{g_0^{d_1} h_0^{d_0-1}}| = |\frac{\text{Res}(A_{k-1}, A_k)}{g_{k-1}^{d_k} h_{k-1}^{d_{k-1}-1}}|$$

En prenant le rang k tel que A_k est constant, on a $d_k = 0$ et le résultant est égal à $g_k^{d_{k-1}}$, on obtient donc :

$$|\text{Res}(A_0, A_1)| = |\frac{g_k^{d_{k-1}}}{h_{k-1}^{d_{k-1}-1}}|$$

Comme ici $\delta_{k-1} = d_{k-1}$, le terme de droite est $|h_k|$.

Remarque

On peut calculer au fur et à mesure le signe du résultant en tenant compte des degrés de A_k pour inverser l'ordre de A_{k-1} et A_k dans le résultant.

Utilisation

La valeur du résultant est très utile pour savoir si 2 polynômes dépendant de paramètres sont premiers entre eux en fonction de la valeur des paramètres. En effet, la fonction `gcd` d'un logiciel de calcul formel calculera le PGCD par rapport à toutes les variables en incluant les paramètres. En cherchant quand le résultant s'annule en fonction des paramètres on obtient un autre type d'information.

Exemple :

Chercher quand le polynône $P = x^3 + px + q$ possède une racine multiple en fonction de p et q. On calcule le résultant de P et P' et on trouve $4p^3 + 27q^2$, donc P a une racine multiple si et seulement si $4p^3 + 27q^2 = 0$.

Remarque :

Comme le coefficient dominant de P' est un multiple du coefficient dominant de P, le résultant de P et P' est divisible par le coefficient dominant de P, on appelle le quotient discriminant.

6 Localisation des racines : les suites de Sturm.

L'algorithme du sous-résultant appliqué à un polynôme sans racine multiple P et à sa dérivée permet, à condition de changer les signes dans la suite des restes, de connaitre le nombre de racines réelles d'un polynôme à coefficients réels dans un intervalle. Ceci est trè utile pour par exemple simplifier des valeurs absolues de polynômes dans un intervalle.

On définit donc la suite de polynômes $A_0 = P, A_1 = P', ..., A_k, 0$ par :

$$A_i = A_{i+1} Q_{i+2} - A_{i+2} \tag{8}$$

avec A_k, le dernier reste non nul, un polynôme constant puisque P n'a pas de racine multiple. On utilise plutot l'algorithme du sous-résultant que l'algorithme d'Euclide, il faut alors s'assurer que les signes de A_i et A_{i+2} sont opposés lorsque A_{i+1} s'annule quitte à changer le signe de A_{i+2} en fonction du signe du coefficient dominant de A_{i+1}, de la parité de la différence des degrés et du signe du coefficient $gh^{1-\delta}$.

On définit $s(a)$ comme étant le nombre de changements de signes de la suite $A_i(a)$ en ignorant les 0. On a alors le

Théorème 4 *Le nombre de racines réelles de $A_0 = P$ sur l'intervalle $]a, b]$ est égal à $s(a) - s(b)$.*

Preuve

Par continuité de la suite des polynômes, s ne peut varier que si l'un des polynômes s'annule. On considére la suite des signes en un point : elle ne peut contenir deux 0 successifs (sinon toute la suite vaudrait 0 en ce point en appliquant (8), or A_k est constant non nul). Elle ne peut pas non plus contenir +,0,+ ni -,0,- à cause de la convention de signe sur les restes de (8). Donc une racine b de A_i pour $0 < i < k$, n'influe pas sur la valeur de s au voisinage de b (il y a toujours un changement de signe entre les positions $i-1$ et $i+1$). Comme A_k est constant, seules les racines de $A_0 = P$ sont susceptibles de faire varier s. Comme $A_1 = P'$, le sens de variations de A_0 au voisinage d'une racine de A_0 est déterminé par le signe de A_1, donc les possibilités sont -,+ vers +,+ ou +,- vers -,-, ce qui diminue s d'une unité.

Remarques

- On peut localiser les racines réelles par dichotomie : on sait que toutes les racines sont situées dans l'intervalle $[-C, C]$ avec $C = |P|_\infty / |\text{lcoeff}(P)|$. On coupe l'intervalle en deux, on calcule le nombre de racines dans chaque partie, et on continue en conservant uniquement les intervalles contenant au moins une racine. Lorsqu'un intervalle contient une seule racine, on passe à la dichotomie classique (changement de signe), ou à la méthode de Newton (avec évaluation exacte du polymôme et arrondi du dénominateur à une puissance de 2). C'est ce qui est utilisé par l'instruction `realroot` de Giac.
- Il existe un autre algorithme de localisation de racines réelles dû à Vincent, Collins et Akritas. Un étudiant d'Akritas travaille sur une implémentation C++ de cet algorithme dans Giac.
- Ces suites se généralisent dans le plan complexe, on peut déterminer le nombre de racines contenues dans un rectangle du plan complexe (cf. par exemple l'article de Mickael Eiserman sur `www-fourier.ujf-grenoble.fr/~eiserm`). Malheureusement, il faut calculer une nouvelle suite de Sturm pour chaque rectangle (alors que dans $\mathbb{R}$ on peut réutiliser la même suite de Sturm). Ce qui est donc beaucoup plus couteux, en pratique on ne peut guère aller au-delà du degré 10 avec l'instruction `complexroot` de Giac, analogue de `realroot`.
- Une autre méthode dans le cas complexe, peut-être plus prometteuse, consisterait à utiliser un hybride numérique-exact. Les racines d'un polynôme sont aussi les valeurs propres complexes de sa matrice companion M. On peut alors par une méthode itérative (on pose $A_0 = M$, puis on factorise $A_n = QR$ par la méthode de Hessenberg et on définit $A_{n+1} = RQ$), factoriser cette matrice sous forme de Schur : $M = P^{-1}SP$ où P est unitaire et S diagonale supérieure aux erreurs d'arrondis près. En approchant P par une matrice exacte P', on peut essayer de controler la localisation des valeurs propres de M à partir des coefficients diagonaux de S et de la taille des coefficients sous-diagonaux de $P'MP'^{-1}$.

7 Exercices (PGCD, résultant, ...)

7.1 Instructions

Les instructions arithmétiques sont en général dans la librairie standard. Elles sont dans les menus Cmds->Integer et Cmds->Polynomes de `Xcas`. Certaines de ces instructions sont dans la librairie `numtheory` (en maple) ou `numlib` (en MuPAD) (utilisez `?numtheory` ou `?numlib` pour avoir la liste des fonctions de ces librairies), pour éviter de taper `numlib::` ou `numtheory::` à chaque fois, on peut lancer en maple la commande `with(numtheory);` ou en MuPAD `export(numlib);`.

7.1.1 Entiers

- `chrem` (en MuPAD `numlib::ichrem`) : restes chinois (entier)
- `divisors` (en maple `numtheory::divisors`, en MuPAD `numlib::divisors`) : liste des diviseurs d'un entier
- `gcd, lcm` : PGCD et PPCM
- `igcdex` : Bézout pour des entiers
- `iquo` et `irem` quotient et reste de la division euclidienne de deux entiers
- `isprime` test de primalité. En maple et MuPAD, il s'agit d'un test de pseudo-primalité. En Xcas, utiliser `is_pseudoprime` pour effectuer un test plus rapide de pseudo-primalité.
- `mods` : reste euclidien symétrique
- `nextprime` et `prevprime` (en MuPAD `numlib::prevprime`) : nombre premier suivant ou précédent
- `powmod(a,b,n)` (Xcas), `a &^ b mod n` (Maple), `powermod(a,b,n)` (Mupad) : calcul de $a^b \pmod n$ par l'algorithme de la puissance rapide

7.1.2 Polynômes

On peut représenter les polynômes par leur écriture symbolique (par exemple `x^2+1`), ou par des listes (représentation dense ou creuse, récursive ou distribuée). Xcas, Maple et MuPAD acceptent la représentation symbolique. Xcas propose deux types de représentation, dense à une variable (`poly1[ ]`), ou distribuée (`%%%{ }%%%`) et des instructions de conversion (`poly2symb` et `symb2poly`) entre représentations. MuPAD propose également une représentation non symbolique, cf. la documentation `?poly`. L'intérêt d'une représentation non symbolique est l'efficacité des opérations polynomiales, (et la possibilité de chronométrer des opérations comme le produit de 2 polynômes).

Les instructions qui suivent utilisent la représentation symbolique, certaines acceptent les autres représentations.

- `coeff` coefficient(s) d'un polynôme,
- `coeffs` liste des coefficients d'un polynôme (à développer auparavant, en mupad on utilise `coeff`)
- `content` contenu (pgcd des coefficients)
- `degree` degré
- `divide` division euclidienne,
- `gcd, lcm` PGCD et PPCM

– `gcdex` Bézout,
– `genpoly` (en MuPAD `numlib::genpoly`) : crée un polynôme à partir de la représentation z-adique d'un entier (utile pour le PGCD heuristique)
– `icontent` : contenu entier pour un polynôme à plusieurs variables
– `indets` : liste des noms de variables d'une expression
– `lcoeff` : coefficient dominant d'un polynôme
– `ldegree` : valuation
– (MuPAD) `multcoeffs` multiplie les coefficients d'un polynôme
– (MuPAD) `pdivide` pseudo-division
– (MuPAD) `poly(expr,[var],coeff)` crée un polynôme à partir de l'-expression symbolique `expr` par rapport une variable ou à une liste de variables `var`, on peut indiquer dans quel anneau vivent les coefficients (par exemple dans $\mathbb{Z}/13\mathbb{Z}$ avec comme 3ème argument `IntMod(13)`)
– `primpart` : partie primitive d'un polynôme
– `quo`, `rem` (xcas et Maple) quotient et reste euclidien (en MuPAD utiliser les options Quo et Rem de `divide`)
– `tcoeff` : coefficient de plus bas degré d'un polynôme
– `interp` (MuPAD `interpolate`) : interpolation de Lagrange
– `convert(.,sqrfree)` (MuPAD `polylib::sqrfree`) : décomposition en facteurs n'ayant pas de racine multiples
– `convert(.,parfrac)` (MuPAD `polylib::partfrac`) : décomposition en éléments simples
– `resultant` (MuPAD `polylib::resultant`) : calcule le résultant de 2 polynômes par rapport à une variable.

Notez aussi que le menu `Exemples->poly->pgcd.xws` de Xcas contient des exemples de programmes de calcul de pgcd de type Euclide.

7.1.3 Calculs modulo n

Pour travailler dans $\mathbb{Z}/n\mathbb{Z}[X]$:
– avec `Xcas` on utilise la notation % comme en C, par exemple `gcd(P % 3, Q % 3)`. On peut aussi utiliser la notation Maple en mode "syntaxe Maple" (cf. ci-dessous)
– avec Maple, on utilise les formes inertes des instructions (qui renvoient l'instruction non évaluée), dont le nom est le même que le nom de commande habituel mais précédé par une majuscule, puis on indique `mod n`, par exemple `Gcd(P,Q) mod 11`.
– avec MuPAD, on désigne le type des coefficients par exemple par `IntMod(13)` puis on construit des objets ayant des coefficients de ce type (par exemple des polynômes, cf. infra). Par exemple `poly(x^2+1,[x],IntMod(13))`.

7.2 Exercices PGCD

1. Calculez le pgcd de $x^{202}+x^{101}+1$ et sa dérivée modulo 3 et modulo 5. Conclusion ?
2. $P=51x^3-35x^2+39x-115$ et $Q=17x^4-23x^3+34x^2+39x-115$. Calculez le pgcd de P et Q modulo 5, 7 et 11. En déduire le pgcd de P et Q

par le théorème des restes chinois. Pourquoi ne doit-on pas essayer modulo 17 ?

3. Écrire un programme qui détermine le degré probable du pgcd de 2 polynômes en une variable en utilisant le pgcd modulaire (on considère le degré probable déterminé lorsqu'on trouve deux nombres premiers réalisant le minimum des degrés trouvés)
4. Détaillez l'algorithme du PGCD heuristique pour les polynômes $P = (x+1)^7 - (x-1)^6$ et sa dérivée. Comparez avec l'algorithme d'Euclide naïf.
5. Écrire un programme mettant en oeuvre le pgcd heuristique pour des polynômes à une variable.
6. On veut comprendre comment un logiciel de calcul formel calcule

$$\int \frac{x^6+2}{(x^3+1)^2}\,dx$$

On se ramène d'abord à une fraction propre (numérateur N de degré inférieur au dénominateur), Soit $P = X^3 + 1$, calculez le PGCD de P et P', puis deux polynômes U et V tels que :

$$N = UP + VP'$$

On décompose alors l'intégrale en deux morceaux :

$$\int \frac{N}{P^2} = \int \frac{U}{P} + \int V\frac{P'}{P^2}$$

Faites une intégration par parties sur le deuxième terme et en déduire la valeur de l'intégrale du départ.

7. Écrire un programme mettant en oeuvre l'algorithme modulaire de calcul du PGCD.
8. Écrire un programme qui détermine le degré probable du PGCD par rapport à toutes les variables de 2 polynôme à plusieurs variables en utilisant l'évaluation en toutes les variables sauf une.
9. Calculer le pgcd par une méthode modulaire de $(xy - x + 1)(xy + x^2 + 1)$ et $(xy - x - y)(xy - x + 1)$

7.3 Exercices (résultant)

1. Pour quelles valeurs de p le polynôme $X^5 + X^3 - pX + 1$ admet-il une racine multiple ?
2. Résoudre le système en éliminant successivement les variables grâce au résultant :

$$\begin{cases} a^3 + b^3 + c^3 &= 8 \\ a^2 + b^2 + c^2 &= 6 \\ a + b + 2c &= 4 \end{cases}$$

3. Donner le détail des calculs avec Bézout de la décomposition en éléments simples de :

$$\frac{1}{(x^2-1)^2(x+2)}$$

puis calculer le coefficient de x^n du développement en séries entières de cette fraction en 0.

4. Calculer
$$\int \frac{1-x^2}{1+x^4}\,dx$$
en utilisant le résultant pour calculer les logarithmes.

5. En utilisant uniquement l'instruction de calcul de PGCD déterminer la multiplicité maximale d'un facteur irréductible de $x^{14}-x^{13}-14x^{12}+12x^{11}+78x^{10}-54x^9-224x^8+116x^7+361x^6-129x^5-330x^4+72x^3+160x^2-16x-32$

7.4 Exercice (Bézout modulaire)

Soit A et B deux polynômes à coefficients entiers et premiers entre eux. Soit $c \in \mathbb{Z}^*$ le résultant de A et B, on va calculer les polynômes U et V de l'identité de Bézout

$$AU + BV = c, \quad \deg(U) < \deg(B), \deg(V) < \deg(A) \tag{9}$$

par une méthode modulaire.

1. Montrer, en utilisant les formules de Cramer, que les coefficients de U et V sont des entiers de valeur absolue inférieure ou égale à la borne de Hadamard h de la matrice de Sylvester de A et B (dont le déterminant est c, le résultant de A et B). Calculer h en fonction de la norme euclidienne de A, B et de leurs degrés.
2. On calcule $c \in \mathbb{Z}^*$ puis on résoud (9) dans $\mathbb{Z}/p_iZ[X]$ pour plusieurs nombres premiers p_i (choisis si possible inférieurs à $\sqrt{2^{31}}$ pour des raisons d'efficacité), puis on calcule par le théorème des restes chinois (9) dans $\mathbb{Z}/\prod p_iZ[X]$. Donner une minoration de $\prod_i p_i$ faisant intervenir h qui permette de garantir que l'écriture en représentation symétrique de (9) dans $\mathbb{Z}/\prod p_iZ[X]$ est identique à (9) dans $\mathbb{Z}[X]$.
3. Application : résoudre de cette manière l'équation de Bézout pour
$$A = (X+1)^4(X-3), \quad B = (X-1)^4(X+2)$$
(vous pouvez utiliser sans justifications l'instruction de calcul de résultant, des coefficients de Bézout dans $\mathbb{Z}/p_iZ[X]$ et de reste chinois de votre logiciel).
4. Écrire une fonction mettant en oeuvre cet algorithme.
5. Que pensez-vous de l'intérêt de cet algorithme par rapport à l'algorithme d'Euclide étendu dans $\mathbb{Z}[X]$?

7.5 Exercice (Géométrie et résultants).

On cherche une relation algébrique entre les coordonnées de 4 points A, B, C, D qui traduise le fait que ces 4 points sont cocycliques. Cette condition étant invariante par translation, on cherche une relation entre les 6 coordonnées des 3 vecteurs $v_1 = (x_1, y_1)$, $v_2 = (x_2, y_2)$ et $v_3 = (x_3, y_3)$ d'origine A et d'extrémité B, C et D. On peut supposer quitte à translater que le centre du cercle est l'origine, on a

donc 5 paramètres : le rayon du cercle R et les 4 angles des points sur le cercle θ_0, θ_1, θ_2 et θ_3. La relation cherchée va s'obtenir en éliminant les 5 paramètres des expressions des 6 coordonnées en fonction de ces paramètres.

1. Exprimer les 6 coordonnées en fonction de R et $a = \tan(\theta_0/2)$, $b = \tan(\theta_1/2)$, $c = \tan(\theta_2/2)$ et $d = \tan(\theta_3/2)$. On obtient ainsi 6 équations, par exemple les deux premières sont de la forme

   $$x_1 - F(R, a, b) = 0, \quad y_1 - G(R, a, b) = 0$$

   où F et G sont deux fractions rationnelles.
2. En réduisant au même dénominateur, calculer 6 polynômes, fonction de $x_1, y_1, x_2, y_2, x_3, y_3, R, a, b, c, d$, qui doivent s'annuler pour que les points soient cocycliques (Vous pouvez utiliser l'instruction `numer` pour obtenir le numérateur d'une fraction rationnelle).
3. Éliminer b des polynômes contenant x_1 et y_1 et factoriser le polynôme obtenu, faire de même avec c, x_2 et y_2 et d, x_3 et y_3, en déduire (en supposant que les points sont tous distincts) 3 polynômes en $x_1, y_1, x_2, y_2, x_3, y_3, R, a$ qui s'annulent.
4. Éliminer R et a, en déduire la relation cherchée.
5. Vérifier que cette relation est équivalente à la nullité de la partie imaginaire du birapport des affixes $\alpha, \beta, \gamma, \delta$ des 4 points :

   $$\Im\left(\frac{\alpha-\beta}{\alpha-\gamma}\frac{\delta-\gamma}{\delta-\beta}\right) = 0$$

7.6 Décalage entier entre racines.

Soit P un polynôme à coefficients entiers sans racines multiples. On dira que P a la propriété $\mathcal{I}$ si deux des racines de P sont décalées d'un entier. En d'autres termes, si $r_1, ..., r_n$ désignent les racines complexes distinctes de P, P possède la propriété $\mathcal{I}$ s'il existe au moins un entier parmi les différences $r_i - r_j$ pour $i \neq j$.

1. Soit

   $$R(t) = \text{resultant}_x(P(x), P(x+t))$$

   Montrer que R est à coefficients entiers. Montrer que la propriété $\mathcal{I}$ est équivalente à la propriété "R possède une racine entière non nulle". On va maintenant construire un algorithme déterminant les racines entières du polynôme R.
2. Après division de R par une puissance de t, on peut supposer que R a un coefficient constant non nul. Après division de R par son contenu, on peut aussi supposer que le contenu de R est 1. En effectuant ensuite une factorisation square-free de R, on peut se ramener au cas où R et R' sont premiers entre eux. Soit a une racine de R.
   (a) Donner une majoration de $|a|$ en fonction du coefficient constant de R.
   (b) Soit p un nombre premier ne divisant pas le coefficient dominant de R et tel que R et R' soient premiers entre eux modulo p. On peut calculer a à partir d'une racine de R modulo p en la "remontant" modulo p^k pour k assez grand (algorithme p-adique). Pour quelle valeur de k peut-on reconstruire toutes les racines entières de R ?

(c) Comparer l'algorithme ci-dessus avec les algorithmes suivants : la factorisation de R sur $\mathbb{Z}$, la recherche numérique des racines complexes de R, la recherche des racines entières de R parmi les diviseurs entiers du coefficient constant de R et leurs opposés.

3. Une fois les racines entières de R connues, comment peut-on en déduire les facteurs de P dont les racines diffèrent de cet(ces) entier(s) ?

4. Soit
$$P(x) = x^6 + 9x^5 + 29x^4 + 41x^3 + 37x^2 + 59x + 31$$
Montrer que P a la propriété $\mathcal{I}$. Calculer la ou les racines entières de R et donner la factorisation correspondante de P.

5. Écrire un programme qui effectue cet algorithme sur un polynôme quelconque. On pourra utiliser la fonction `rationalroot` de Xcas pour déterminer les racines entières de R.

6. Application : on cherche à calculer
$$\sum_{k=1}^{n} \frac{-9x^2 - 27x - 30}{P(x)} \tag{10}$$
Décomposer cette fraction en éléments simples (donner le détail des calculs en utilisant la factorisation précédente et l'identité de Bezout `abcuv` en Xcas).

7. Calculer la somme précédente (10). On pourra remarquer que pour k entier strictement positif, $\frac{1}{f(x+k)} - \frac{1}{f(x)}$ s'exprime comme une somme de différences $\frac{1}{f(x+j+1)} - \frac{1}{f(x+j)}$.

8. Écrire un programme effectuant ce calcul avec une fraction quelconque, lorsque cela est possible.

8 Factorisation

On présente ici quelques algorithmes utilisés pour factoriser un polynôme à coefficients entiers. Pour un polynôme en une variable, cele se fait en plusieurs étapes : on commence par se ramener à un polynôme P dont tous les facteurs sont de multiplicité un, ensuite on factorise P dans $\mathbb{Z}/p\mathbb{Z}$ (par la méthode de Berlekamp ou Cantor-Zassenhauss), puis on remonte à $\mathbb{Z}/p^kZ$ pour k suffisamment grand (en fonction de la borne de Landau sur les facteurs de P), et on recombine enfin les facteurs modulaires pour trouver les facteurs de P. Lorsque P à plusieurs variables, on utilise une méthode analogue à celle permettant de trouver le pgcd de polynômes à plusieurs variables.

Rappel
Le pgcd des coefficients d'un polynôme est appelé contenu de ce polynôme. Un polynôme est dit primitif si son contenu est égal à 1.

8.1 Les facteurs multiples

Étant donné un polynôme P à coefficients entiers, on cherche à écrire :

$$P = \Pi_{k=1}^{n} P_k^k$$

où les P_k n'ont pas de facteurs multiples et sont premiers entre eux deux à deux. Comme on est en caractéristique 0, cela revient à dire que pgcd$(P_k, P_k') = 1$ et pgcd$(P_k, P_j) = 1$. Bien entendu on va utiliser la dérivée de P dans l'algorithme de recherche des P_k :

$$P' = \sum_{k=1}^{n} k P_k' P_k^{k-1} \Pi_{j \neq k} P_j^j$$

Soit G le pgcd de P et de P'. On a :

$$G = \Pi_{k=1}^{n} P_k^{k-1},$$

en effet G divise P et P' :

$$W_1 = \frac{P}{G} = \Pi_{k=1}^{n} P_k, \quad Z_1 = \frac{P'}{G} = \sum_{k=1}^{n} k P_k' \Pi_{j \neq k} P_j$$

il s'agit de vérifier que W_1 et Z_1 sont premiers entre eux. Soit F un facteur irréductible du pgcd de W_1 et Z_1, alors F divise l'un des P_k, appelons P_l ce facteur. Comme F divise $\Pi_{j \neq k} P_j$ si $k \neq l$, on en déduit que F divise le dernier terme de la somme de Z_1, c'est-à-dire que F divise $l P_l' \Pi_{j \neq l} P_j$ donc F divise P_l' puisque les P_k sont premiers entre eux. Donc P_l et P_l' ont un facteur en commun, ce qui est contraire aux hypothèses.

On pose alors :

$$Y_1 = Z_1 - W_1' = \sum_{k>1} (k-1) P_k' \Pi_{j \neq k} P_j$$

On définit alors par récurrence des suites de polynômes W_n, Y_n et G_m par :

- $G_m =$ pgcd(W_m, Y_m)
- $W_{m+1} = W_m / G_m$ et $Y_{m+1} = Y_m / G_m - W_m'$

On va montrer que $P_m = G_m$. Commençons au rang $n = 1$, on voit que P_1 divise Y_1 (puisqu'il est commun à tous les $\Pi_{j\neq k}P_j$ car $k > 1$) et divise W_1. Et c'est le seul facteur commun, car tout autre facteur irréductible serait un diviseur d'un P_l pour $l > 1$, donc diviserait $(l-1)P_l'\Pi_{j\neq l, j>1}P_j$, donc diviserait P_l'. Le raisonnement en un rang quelconque est identique, les polynômes sont donnés par :

$$G_m = P_m,\ W_m = \Pi_{k>=m}P_k,\ Y_m = \sum k > m(k-m)P_k'\Pi_{j\geq m, j\neq k}P_j$$

Lorsqu'on programme cet algorithme, le test d'arrêt est $G_m = 1$.

Square-free factorisation (Algorithme de Yun)

Argument : un polynôme primitif P à coefficients entiers (ou dans $\mathbb{Z}[i]$ ou dans un corps de caractéristique nulle).
Valeur renvoyée : une liste de polynômes P_m telle que $P = \Pi_{k=1}^{n}P_k^k$.

1. Initialiser la liste résultat à liste vide.
2. Initialiser W à P et Y à P'. Calculer le pgcd G de W et Y et simplifier W et Y par leur pgcd puis poser $Y = Y - W'$.
3. Boucle infinie.
4. Calculer le pgcd G de W et Y. Si $G = 1$, on renvoie la liste résultat sinon ajouter G à la liste résultat.
5. Simplifier W et Y par G, puis poser $Y = Y - W'$ et passer à l'itération suivante.

Remarque : lorsqu'on veut factoriser un polynôme à coefficients modulaires, il faut aussi se ramener à un polynôme sans facteurs multiples mais on ne peut pas utiliser cet algorithme tel quel car la caractéristique du corps n'est pas nulle.

Exemple :
Factorisation sans facteurs multiples de $P(X) = (X^3-1)(X+2)^2(X^2+3)^3$.
En mode interactif avec un logiciel de calcul formel, effectuons l'étape d'initialisation :

```
W:=normal((x^3-1)*(x+2)^2*(x^2+3)^3);
Y:=diff(W,x);
G:=gcd(W,Y);
        x^5+2*x^4+6*x^3+12*x^2+9*x+18
W:=normal(W/G);
        x^6+2*x^5+3*x^4+5*x^3+-2*x^2+-3*x-6
Y:=normal(Y/G);
Y:=normal(Y-diff(W,x));
        5*x^5+8*x^4+3*x^3+-5*x^2+-8*x-3
```

On vérifie bien que $W = (x+2)*(x^3-1)*(x^2+3)$ est le produit des facteurs P_i. On entame maintenant la boucle :

```
G:=gcd(W,Y);
        x^3-1   -> P1
Y:=normal(Y/G);
W:=normal(W/G);
Y:=normal(Y-diff(W,x));
```

```
        2*x^2+4*x
G:=gcd(W,Y);
        x+2     -> P2
Y:=normal(Y/G);
W:=normal(W/G);
Y:=normal(Y-diff(W,x));
        0
G:=gcd(W,Y);
        x^2+3   ->  P3
```

puis $W = 1$ et $Y = 0$ et le prochain G vaut 1, on a bien trouvé tous les facteurs P_i.

8.2 Factorisation en une variable

On suppose maintenant qu'on veut factoriser un polynôme P sans facteur multiple (et primitif). En général on commence par simplifier P par ses facteurs linéaires (détectés avec l'algorithme présenté dans le premier article de cette série). On commence par chercher un nombre premier p tel que P dans $\mathbb{Z}/p\mathbb{Z}$ conserve le même degré et reste sans facteur multiple (donc pgcd(P, P')=1 dans $\mathbb{Z}/p\mathbb{Z}$), ce qui est toujours possible (il suffit de prendre p plus grand que le plus grand entier apparaissant dans l'algorithme du sous-résultant pour calculer le pgcd de P et P' dans $\mathbb{Z}$).

Convention
Tous les polynômes ayant leurs coefficients dans un corps fini sont supposés avoir comme coefficient dominant 1 lorsque le choix existe (par exemple les facteurs d'un polynôme modulo p).

8.2.1 Factorisation dans $\mathbb{Z}/p\mathbb{Z}[X]$

On suppose qu'on a un polynôme P à coefficients dans $\mathbb{Z}/p\mathbb{Z}$ sans facteur multiple. Il s'agit de factoriser P dans $\mathbb{Z}/p\mathbb{Z}[X]$. Il existe essentiellement deux stratégies, l'une commence par factoriser par groupes de facteurs de même degré puis casse les facteurs et l'autre plus directe à base d'algèbre linéaire modulaire (méthode de Berlekamp). Dans les deux cas, on utilise le fait que si F est un polynôme, alors les polynômes à coefficients dans $\mathbb{Z}/p\mathbb{Z}$ modulo F forment un anneau A qui est aussi un espace vectoriel sur $\mathbb{Z}/p\mathbb{Z}$ de dimension le degré de F (si F est irréductible, alors A est un corps). On s'intéresse alors aux propriétés de l'application $\varphi : x \in A \mapsto x^p$. On observe d'abord que cette application est une application *linéaire*. Cela découle du petit théorème de Fermat pour $\varphi(\lambda x) = \lambda\varphi(x)$ et de la formule de Newton et de la primalité de p pour $\varphi(x + y) = \varphi(x) + \varphi(y)$.

Calcul de φ
Pour mettre en oeuvre ces algorithmes, on commence par déterminer la matrice de l'endomorphisme $\varphi : x \mapsto x^p$ dans $\mathbb{Z}/p\mathbb{Z}[X] \pmod{P(X)}$ muni de sa base canonique $\{1, X, ..., X^{\deg(P)-1}\}$.

8.2.2 Distinct degree factorization

Cette méthode consiste à détecter les groupes de facteurs ayant un degré donné (distinct degree factorization). Si nécessaire, on utilise ensuite un autre algorithme

pour casser ces groupes. On utilise ici les propriétés des itérées de l'application linéaire φ sur des espaces vectoriels de corps de base $\mathbb{Z}/p\mathbb{Z}$. On va déterminer le produit P_k de tous les facteurs de P de degré k en calculant le pgcd de P et de $X^{(p^k)} - X$ dans $\mathbb{Z}/p\mathbb{Z}[X]$.

Pour $k = 1$, $X^p - X$ est le produit des $X - k$ pour tout $k \in \mathbb{Z}/p\mathbb{Z}$ par le petit théorème de Fermat ($k^p = k \pmod p$), donc le pgcd de P et de $X^{(p^1)} - X$ dans $\mathbb{Z}/p\mathbb{Z}[X]$ est le produit des facteurs de P de degré 1.

Pour $k > 1$, le raisonnement se généralise de la manière suivante : on considère un facteur irréductible $F(X)$ de P de degré k et le corps $K = (\mathbb{Z}/p\mathbb{Z})[Y] \pmod{F(Y)}$. Le corps K est un corps fini, c'est aussi un espace vectoriel sur $\mathbb{Z}/p\mathbb{Z}$ de dimension k, donc K possède p^k éléments et K^* est un groupe multiplicatif à $p^k - 1$ éléments, donc tout élément de K^* vérifie l'équation $x^{p^k-1} = 1$ donc tout élément de K vérifie $x^{(p^k)} = x$. En particulier pour $x = Y \pmod{F(Y)}$ on trouve que $Y^{(p^k)} = Y \pmod{F(Y)}$ donc $F(X)$ divise $X^{(p^k)} - X$ dans $\mathbb{Z}/p\mathbb{Z}$.

Réciproquement, si on se donne un facteur irréductible F qui divise $X^{p^k} - X$, soit K le corps correspondant à F, alors le noyau de l'application linéaire

$$x \in K \mapsto x^{p^k} - x \in K$$

est K tout entier, car $Y = Y^{p^k} \pmod F$ entraine $(Y^2)^{(p^k)} = Y^{2p^k} = (Y^{p^k})^2 = Y^2 \pmod F$ et de même pour les autres puissances de Y qui, avec $Y^0 = 1$ également dans le noyau, forment une base de l'espace vectoriel K sur $\mathbb{Z}/p\mathbb{Z}$. Donc le nombre d'éléments de K est inférieur ou égal au degré du polynôme $X^{p^k} - X$ (puisque $X^{(p^k)} - X$ est divisible par $X - x$ pour tout $x \in K$), donc le degré de F est inférieur ou égal à k.

Donc P_k est égal au pgcd de $P/\Pi_{j<k}P_j$ avec $X^{p^k} - X$.

Algorithme distinct degree factorization

Argument : un polynôme P à coefficients entiers sans facteur multiple et primitif.
Valeur renvoyée : la liste L des produits des facteurs irréductibles et du degré correspondant de P (ordonné par ordre croissant de degré).
On commence par initialiser L à vide et un polynôme auxiliaire Q à X (il contiendra les valeurs de $X^{p^k} - X \pmod P$), on fait une boucle indéfinie sur k commençant à 1 et incrémenté de 1 à chaque itération

- Si k est strictement plus grand que le degré de P divisé par 2, on rajoute le couple (P,degre(P)) à L et on renvoie L
- On remplace Q par $Q^p \pmod P$ en utilisant le calcul de φ modulo P
- On calcule le pgcd G de $Q - X$ et de P.
- Si G vaut 1, on passe à l'itération suivante
- On rajoute le couple (G,k) à la liste L et on remplace P par le quotient de P par G.

Exemple :
Factorisation en degré distincts de $(X^3 + X + 1)(X^4 - X + 1)$ dans $\mathbb{Z}/5\mathbb{Z}$. On regarde d'abord si P reste sans facteur multiple après réduction modulo 5.

```
P:=normal((x^3+x+1)*(x^4-x+1) mod 5);
gcd(P,diff(P,x));
    1 mod 5  -> ok P est sans facteur multiple
P1:=gcd(P,(x^5-x)mod 5);
    (1 mod 5)*x -2 mod 5  -> P1
```

```
P:=normal(P/P1);
P2:=gcd(P,(x^(5^2)-x)mod 5);
    1 mod 5  -> pas de facteur de degre 2
P3:=gcd(P,(x^(5^3)-x)mod 5);
    (x^6+2*x^5+x^2+x+2) mod 5
```

Donc P admet 3 facteurs dans $\mathbb{Z}/5\mathbb{Z}$: un de degré 1 ($x-2$) et deux de degré 3 (dont le produit est $x^6+2x^5+x^2+x+2$).

Le même calcul dans $\mathbb{Z}/7\mathbb{Z}$ donne

```
P:=normal((x^3+x+1)*(x^4-x+1) mod 7);
gcd(P,diff(P,x));
    1 mod 7  -> ok P est sans facteur multiple
P1:=gcd(P,(x^7-x)mod 7);
    1 mod 7
P2:=gcd(P,(x^(7^2)-x)mod 7);
    1 mod 7
P3:=gcd(P,(x^(7^3)-x)mod 7);
    (x^3+x+1) mod 7
```

donc P possède un facteur de degré 3 modulo 7, donc le facteur restant de degré 4 est forcément irréductible.

On remarque sur cet exemple que 7 est plus intéressant que 5, car la factorisation modulo 7 donne moins de facteurs (à recombiner pour trouver la factorisation dans $\mathbb{Z}$) et la factorisation est complète modulo 7 alors que modulo 5 il faut casser le facteur de degré 6 en deux facteurs de degré 3. La plupart des algorithmes de factorisation effectuent la factorisation en degré distinct modulo plusieurs entiers (ce qui peut de plus être parallélisé) et choisissent le meilleur.

8.2.3 La méthode de Cantor-Zassenhaus

Cet algorithme sert à casser des groupes de facteurs de même degré, c'est une méthode probabiliste. On suppose donc qu'on a un produit P d'au moins deux facteurs irréductibles de degré d à casser. Soit D l'un des polynômes irréductibles de degré d à coefficients dans $\mathbb{Z}/p\mathbb{Z}$, et soit $K = \mathbb{Z}/p\mathbb{Z}[Y] \pmod{D(Y)}$, on a :

$$X^{p^d} - X = \Pi_{\alpha \in K}(X - \alpha)$$

puisque le corps K possède p^d éléments tous racines de l'équation $X^{p^d} = X$.

On considère un polynôme T non constant, et le polynôme $T^{p^d} - T$. En remplaçant X par T ci-dessus, on en déduit :

$$T^{p^d} - T = \Pi_{\alpha \in K}(T - \alpha)$$

Donc pour tout élément $\beta \in K = \mathbb{Z}/p\mathbb{Z}[Y] \pmod{D(Y)}$, on a

$$(T^{p^d} - T)(\beta) = \Pi_{\alpha \in K}(T(\beta) - \alpha) = 0$$

Donc $T^{p^d} - T$ est divisible par $X^{p^d} - X$ (puisque toutes les racines du second sont racines du premier), donc est divisible par tout polynôme irréductible de degré inférieur ou égal à d à coefficients dans $Z/p\mathbb{Z}$. Comme

$$T^{p^d} - T = T(T^{\frac{p^d-1}{2}} - 1)(T^{\frac{p^d+1}{2}} - 1) \tag{11}$$

et que ces trois facteurs sont premiers entre eux, on en déduit que tout polynôme irréductible de degré inférieur ou égal à d à coefficients dans $Z/p\mathbb{Z}$ divise l'un des trois facteurs ci-dessus. Pour casser P, l'idée consiste alors à calculer le pgcd de P et $T^{\frac{p^d-1}{2}}-1$ pour un polynôme pris au hasard. On sait que P divise le produit des 3 termes de (11), et on espère que les facteurs irréductibles de P ne diviseront pas tous le même terme.

On va montrer que si T est un polynôme de degré $\leq 2d-1$ choisi au hasard, la probabilité que deux facteurs irréductibles de P ne divisent pas $T^{p^d}-T$ est proche de 0.5. Soient donc A et B deux facteurs irréductibles de P de degré d. D'après l'identité de Bézout, tout polynôme T de degré $\leq 2d-1$ s'écrit de manière unique sous la forme :

$$T = AU + BV \tag{12}$$

avec degre($U \leq d-1$) et degre($V \leq d-1$) et réciproquement une combinaison linéaire de cette forme est un polynôme de degré $\leq 2d-1$. Choisir T au hasard revient donc à choisir un couple (U,V) de polynômes à coefficients dans $\mathbb{Z}/p\mathbb{Z}$ au hasard et de manière indépendante. D'autre part, A et B étant de degré d, on sait que dans $K = \mathbb{Z}/p\mathbb{Z}[Y] \pmod{D(Y)}$ ces polynômes admettent d racines. Soit donc α [respectivement β] une racine de A [resp. B] dans K. Alors A divise $T^{\frac{p^d-1}{2}}-1$ si et seulement si $T(\alpha)^{\frac{p^d-1}{2}} = 1$ (et de même pour B et β) car $T^{\frac{p^d-1}{2}}-1$ a ses coefficients dans $\mathbb{Z}/p\mathbb{Z}$ (et non dans K). En appliquant (12), A divise $T^{\frac{p^d-1}{2}}-1$ si et seulement si :

$$B(\alpha)^{\frac{p^d-1}{2}} V(\alpha)^{\frac{p^d-1}{2}} = 1$$

Le premier terme de cette égalité est une constante égale à 1 ou -1, le second a une probabilité proche de 0.5 (égale à $\frac{p^d-1}{2p^d}$) de valoir 1 ou -1 car, comme A est irréductible, $V(\alpha)$ décrit K lorsque V décrit les polynômes de degré $\leq d-1$. De même, B a une probabilité proche de 0.5 de diviser $T^{\frac{p^d-1}{2}}-1$, et ces 2 probabilités sont indépendantes puisque U et V le sont, donc la probabilité que soit A soit B divise divise $T^{\frac{p^d-1}{2}}-1$ est proche de 0.5.

Algorithme de Cantor-Zassenhaus

Argument : Un polynôme P à coefficients dans $\mathbb{Z}/p\mathbb{Z}$ de degré k dont tous les facteurs irréductibles sont de degré d.
Valeur renvoyée : la liste des facteurs irréductibles de P.

- Si $k = d$ renvoyer une liste contenant P.
- Déterminer un polynôme T aléatoire de degré inférieur ou égal à $2d-1$ et de coefficient dominant 1. Calculer le pgcd D de P et de $T^{(p^d-1)/2}-1$. Si le degré de T est égal à 0 ou à k recommencer cette étape.
- Appeler récursivement cet algorithme avec T et P/T et renvoyer la liste réunion des deux listes renvoyées.

Exemple :

Cassons le polynôme de degré 6 obtenu dans l'exemple précédent (modulo 5). Donc $P := (x^6+2*x^5+x^2+x+2) \pmod 5$ et $d = 3, 2d-1 = 5, (p^d-1)/2 = 62$. On choisit au hasard un polynôme de degré inférieur ou égal à 5, par exemple $T = x^4-x^3+x+1$, puis on calcule T^{62} modulo P ce qui donne $(x^5+x^3+x^2+1) \pmod 5$ puis le pgcd de $T^{62}-1$ et de P qui vaut $x^3+x+1 \pmod 5$, on a donc

cassé P en deux. En prenant $T := x^4 - x^3 + x + 2$, on trouve $T^{62} = 1 \pmod P$, donc ce T n'aurait pas permis de casser P.

8.2.4 La méthode de Berlekamp

Cette méthode permet de factoriser un polynôme sans facteurs multiples, elle peut aussi servir à casser des groupes de facteurs de même degré. Ici on travaille dans l'anneau des polynômes à coefficients dans $\mathbb{Z}/p\mathbb{Z}$ modulo le polynôme P et on s'intéresse au noyau de $\varphi - Id$ (où $\varphi : x \mapsto x^p$). On suppose que $P = \Pi_{j=1}^{n} F_j$ où les F_j sont irréductibles et premiers entre eux. On va montrer que le noyau de $\varphi - Id$ est composé des polynômes Q tels que $Q \pmod{F_j}$ est constant (dans $\mathbb{Z}/p\mathbb{Z}$) pour tout j.

Si $Q \pmod{F_j} = s_j \in \mathbb{Z}/p\mathbb{Z}$, alors $Q^p \pmod{F_j} = s_j^p = s_j$, donc par le théorème des restes chinois, $Q = Q^p \pmod P$.

Réciproquement, si $Q^p - Q = 0 \pmod P$, en utilisant la factorisation :

$$X^p - X = \Pi_{j \in \mathbb{Z}/p\mathbb{Z}}(X - j)$$

on en tire P divise $Q^p - Q = \Pi_{j \in \mathbb{Z}/p\mathbb{Z}}(Q(X) - j)$, donc F_j divise l'un des facteurs et $Q(X) \pmod{F_j} \in \mathbb{Z}/p\mathbb{Z}$. Le noyau de $\varphi - Id$ est donc un espace vectoriel de dimension n, le nombre de facteurs irréductibles de P et possède donc p^n éléments (en effet pour tout n uplet de s_j, on peut construire un polynôme Q du noyau par le théorème des restes chinois en posant $Q \pmod{F_j} = s_j$).

L'intérêt du noyau de $\varphi - Id$ est qu'on peut le calculer sans connaitre les F_j. Une fois ce calcul fait, voyons comment on peut remonter aux F_j. On connait déjà la dimension du noyau donc le nombre de facteurs irréductibles. De plus, on remarque que le polynome constant est un élément du noyau qu'on appellera T_1, on note alors $T_2, ..., T_n$ les autres polynômes du noyau. Ensuite, on calcule le pgcd de P avec $T_2 - jT_1$ pour $j \in \mathbb{Z}/p\mathbb{Z}$. On sait que $T_2 = s_{2,j} \pmod{F_j}$, donc ce pgcd est égal au produit des facteurs F_j tels que $s_{2,j} = jT_1$. L'un au moins des pgcd calculés est non trivial car sinon $T_2 = T_1 \pmod{F_j}$ pour tout j donc $T_2 = T_1$. Si on a de la chance tous les $s_{2,j}$ seront distincts et les pgcd non triviaux de P avec $T_2 - jT_1$ donneront les F_k. Sinon il faudra continuer avec $T_3 - jT_1$ etc.

Exemple :
Revenons sur la factorisation de $P := (x^6 + 2x^5 + x^2 + x + 2) \pmod 5$. Commençons par calculer la matrice de φ dans la base $\{1, x, x^2, ..., x^5\}$. On a évidemment $\varphi(1) = 1$ et $\varphi(x) = x^5$, puis $\varphi(x^2) = x^{10} = x^5 + x^4 - 2x^3 + x \pmod P$, puis en multipliant par x^5 et en divisant par P, $\varphi(x^3) = -x^4 + 2x^3$, de la même manière on obtient $\varphi(x^4) = -x^5 + 2x^4 + x^3 - x^2 - 2$ et $\varphi(x^5) = x^3 + x^2 - x$. La matrice de φ est donc :

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 & -2 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & -2 & 2 & 1 & 1 \\ 0 & 0 & 1 & -1 & 2 & 0 \\ 0 & 1 & 1 & 0 & -1 & 0 \end{pmatrix}$$

On calcule ensuite le noyau de $\varphi - Id$ (comme matrice à coefficients dans $\mathbb{Z}/5\mathbb{Z}$), on obtient une base du noyau en prenant par exemple les vecteurs $(-1, 0, 0, 0, 0, 0)$

et $(0, 0, -1, -1, 0, -1)$. Donc le polynôme P possède 2 facteurs dans $\mathbb{Z}/5\mathbb{Z}[X]$. Pour déterminer les facteurs, on calcule le pgcd de P avec le polynôme $T_2 - s$ où $T_2 = -x^5 - x^3 - x^2$ correspond au 2ème vecteur de la base du noyau. On obtient pour $s = 0$ un pcgd non trivial (x^3+x+1), ce qui permet de calculer les 2 facteurs. Si on avait essayé d'autres valeurs de s, pour $s = 1$ on obtient comme pgcd 1, pour $s = 2$ on trouve le 2ème facteur $x^3 + 2x^2 - x + 2$.

8.2.5 Remontée (Hensel)

Il s'agit de passer d'une factorisation de P dans $\mathbb{Z}/p\mathbb{Z}[X]$ à une factorisation de P dans $\mathbb{Z}/p^k Z[X]$, la méthode est analogue à celle de l'algorithme EZGCD de calcul de pgcd de polynômes.

On suppose donc que

$$P = \Pi_{j=1}^{n} P_j \pmod{p}$$

où les P_j sont premiers entre eux deux à deux dans $\mathbb{Z}/p\mathbb{Z}$. Il s'agit de trouver des polynômes $P_{j,k} = P_j \pmod{p}$ tels que

$$P = \Pi_{j=1}^{n} P_{j,k} \pmod{p^k}$$

Commençons par le cas $k = 2$. On pose

$$P_{j,2} = P_j + pQ_j = P_j \pmod{p}$$

On a alors :

$$\begin{aligned} P &= \Pi_{j=1}^{n} P_{j,2} \pmod{p^2} = \Pi_{j=1}^{n}(P_j + pQ_j) \pmod{p^2} \\ &= \Pi_{j=1}^{n} P_j + p\sum_{j=1}^{n} Q_j \Pi_{k\neq j} P_k \pmod{p^2} \end{aligned}$$

Donc :

$$\sum_{j=1}^{n} Q_j \Pi_{k\neq j} P_k = \frac{P - \Pi_{j=1}^{n} P_j}{p} \pmod{p}$$

On est ramené à résoudre une identité de Bézout généralisée. On montrera dans l'appendice le :

Théorème 5 *(Identité de Bézout généralisée) Soit P_1, ..., P_n ($n \geq 2$) des polynômes premiers entre eux deux à deux modulo p. Alors pour tout polynôme Q, il existe des polynômes Q_1, ..., Q_n tels que :*

$$\sum_{j=1}^{n} Q_j \Pi_{k\neq j} P_k = Q \pmod{p}$$

On a donc réussi à remonter l'égalité $P = \Pi P_j \pmod{p}$ à $P = \Pi P_{j,2} \pmod{p^2}$. Le passage de $P = \Pi P_{j,l} \pmod{p^l}$ à $P = \Pi P_{j,l+1} \pmod{p^{l+1}}$ est identique, on a :

$$P_{j,l+1} = P_{j,l} + p^l Q_j$$

où les Q_j sont les solutions de l'identité de Bézout généralisée avec :

$$Q = \frac{P - \Pi_{j=1}^{n} P_{j,l}}{p^l}$$

Lorsqu'on programme cet algorithme (cf. l'appendice), on calcule une fois pour toutes les solutions de l'identité de Bézout pour $Q = 1$, et on multiplie par Q.

Algorithme de remontée de Hensel linéaire

Arguments : Un polynôme P à coefficients entiers, la liste $L = \{P_j\}$ de ses facteurs dans $\mathbb{Z}/p\mathbb{Z}[X]$

Valeur renvoyée : la liste des facteurs de P dans $\mathbb{Z}/p^l\mathbb{Z}[X]$

On calcule la borne de Landau-Mignotte[6] pour les facteurs de P, on multiplie par le coefficient dominant de P et on calcule l tel que p^l est strictement plus grand que deux fois cette quantité. On calcule aussi les polynômes Q_j de l'identité de Bézout généralisée pour $Q = 1$

Puis on fait une boucle pour k variant de 2 à l :

- On détermine $P - \Pi_j P_j \pmod{p^k}$, on divise par p^{k-1} et on place le résultat dans Q
- On multiplie les polynômes Q_j de l'identité de Bézout généralisée (correspondants au polynôme 1) par Q et on détermine le reste de la division euclidienne de QQ_j par P_j, on multiplie par p^{k-1} et on ajoute le résultat à P_j.

Il existe une version quadratique de cette méthode. On passe alors de $P = \Pi P_{j,l} \pmod{p^l}$ à $P = \Pi P_{j,2l} \pmod{p^{2l}}$. Pour cela, il faut trouver les polynômes Q_j solutions de l'équation :

$$\sum_{j=1}^{n} Q_j \Pi_{k \neq j} P_{k,l} = Q \pmod{p^l}$$

Pour $l = 1$, c'est l'identité de Bézout généralisée, mais ce n'est plus le cas pour $l > 1$. En fait, on résout cette égalité en remontant l'identité de Bézout quadratiquement, plus précisément pour trouver les S_j solutions de

$$\sum_{j=1}^{n} S_j \Pi_{k \neq j} P_{k,2l} = Q \pmod{p^{2l}}$$

on pose $S_j = Q_j + p^l R_j$, il s'agit donc de trouver les R_j solutions de

$$\sum_{j=1}^{n} (Q_j + p^l R_j) \Pi_{k \neq j} P_{k,2l} = Q \pmod{p^{2l}}$$

soit :

$$\sum_{j=1}^{n} R_j \Pi_{k \neq j} P_{k,l} = \frac{Q - \sum_{j=1}^{n} Q_j \Pi_{k \neq j} P_{k,l}}{p^l} \pmod{p^l}$$

on en déduit les R_j.

Algorithme de remontée de Hensel quadratique

Arguments et valeur renvoyée identiques à l'algorithme de remontée de Hensel

[6] Rappelons qu'il s'agit d'une majoration sur la valeur absolue des coefficients des facteurs de P

linéaire ci-dessus.
On commence comme dans le cas linéaire par calculer les coefficients de l'identité de Bézout généralisée pour $Q = 1$ et la valeur de l telle que p^{2^l} soit supérieur à deux fois la borne de Landau des facteurs de P fois le coefficient dominant de P.
On fait une boucle sur k variant de 1 à l :

- On calcule $P - \Pi_j P_j \pmod{p^{2^k}}$, on divise par $p^{2^{k-1}}$ et on place le résultat dans Q
- On multiplie par Q les polynômes Q_j de l'identité de Bézout généralisée (avec comme second membre le polynôme 1), on calcule le reste euclidien du résultat par P_j (modulo $p^{2^{k-1}}$), on multiplie par $p^{2^{k-1}}$ et on ajoute à P_j (avec les notations précédentes, on passe ainsi des $P_{j,2^{k-1}}$ aux $P_{j,2^k}$)
- Si $k = l$ on renvoie la liste des P_j
- On calcule $1 - \sum_j Q_j \Pi_{k \neq j} P_k \pmod{p^{2^k}}$, on divise par $p^{2^{k-1}}$ et on place le résultat dans Q
- On multiplie par Q les polynômes Q_j de l'identité de Bézout, généralisée et on calcule le reste euclidien du résultat par P_j (modulo $p^{2^{k-1}}$), on multiplie par $p^{2^{k-1}}$ et on ajoute à Q_j (ce qui ajuste les polynômes Q_j qui vérifient maintenant l'identité de Bézout modulo p^{2^k})

Remarque
Pendant l'étape de remontée de Hensel, une optimisation classique consiste à tester la divisibilité dans $\mathbb{Z}$ du polynôme P par le facteur lifté P_j ([7]) lorsqu'il n'a pas subi de modification pendant 2 étapes successives (autrement dit lorsque $P_j \pmod{p^l} = P_j \pmod{p^{l+1}}$ (ou $\pmod{p^{2l}}$ pour le lift quadratique). Si la division est exacte, on obtient un facteur irréductible de P dans $\mathbb{Z}$. On recalcule alors la borne de Landau de P/P_j pour diminuer le nombre d'itérations à effectuer dans cette étape.

Exemple :
Reprenons le polynôme $P(X) = (X^3 + X + 1)(X^4 - X + 1)$ et supposons qu'on ait choisi de le factoriser modulo 5 puis de remonter. On a 3 facteurs $a = x - 2$, $b = x^3 + x + 1$ et $c = x^3 + 2x^2 - x + 2$. Si on développe P, on trouve 6 coefficients non nuls de valeur absolue 1, on peut calculer la borne de Landau-Mignotte correspondante sur les coefficients d'un facteur entier : $2^5(\sqrt{(6)} + 1)$ soit un peu plus de 110, il suffit donc d'effectuer 3 étapes de remontée linéaire ($5^4 = 625 > 111/2$). On commence par trouver 3 polynômes A, B, C tels que

$$A(x^3 + x + 1)(x^3 + 2x^2 - x + 2) + B(x - 2)(x^3 + 2x^2 - x + 2) + \\ + C(x - 2)(x^3 + x + 1) \quad = \quad 1 \pmod 5$$

On commence par résoudre $D(x^3 + 2x^2 - x + 2) + C(x - 2)(x^3 + x + 1) = 1 \pmod 5$, on trouve $C = 2x^2 - 2$ et $D = -2x^3 - 2x^2 + 2x + 1$. Puis on calcule A et B en résolvant $E(x^3 + x + 1) + F(x - 2) = 1$ qui donne $E = 1$ et $F = -x^2 - 2x$ qu'on multiplie par D, donc $A = D$ et $B = 2x^5 + x^4 + 2x^3 - 2x$. Ce qui donne l'identité de Bézout généralisée.

Passons aux calculs de remontée. On a $abc = x^7 - 4x^5 + 5x^4 + -9x^3 - x^2 - 4$ et $P = x^7 + x^5 + x^3 - x^2 + 1$, donc $Q = (P - abc)/5 = x^5 - x^4 + 2x^3 + 1$. On

[7] Plus exactement, on multiplie P_j par le coefficient dominant de P modulo p^l

pose alors

$$\begin{aligned} a_1 &= a + 5\,(QA \pmod a) \pmod{25}, \\ b_1 &= b + 5\,(QB \pmod b) \pmod{25}, \\ c_1 &= c + 5\,(QC \pmod c) \pmod{25} \end{aligned}$$

donc :

$$a_1 = a + 5 \times (-2), \quad b_1 = b + 5 \times 0, \quad c_1 = c + 5 \times (2x^2 - x)$$

En principe, on continue encore 2 itérations de la même manière. La 2ème itération donne :

$$Q = (P - a_1 b_1 c_1)/25 = 6x^5 - 3x^4 + 7x^3 + 3x^2 - 2x + 1$$

$$\begin{aligned} a_2 &= a_1 + 25\,(QA \pmod a) \pmod{125}, \\ b_2 &= b_1 + 25\,(QB \pmod b) \pmod{125}, \\ c_2 &= c_1 + 25\,(QC \pmod c) \pmod{125} \end{aligned}$$

donc :

$a_2 = a_1 + 25(-1) = x - 37$, $b_2 = b_1 = b$, $c_2 = c_1 + 25(x^2 + 1) = x^3 + 37x^2 - 6x + 27$

On peut aussi observer que $b_1 = b$, ceci laisse à penser que b est un facteur de P dans $\mathbb{Z}$ ce qu'on vérifie en effectuant la division euclidienne de P par $b = x^3 + x + 1$. Comme elle tombe juste, on est ramené à factoriser $x^4 - x + 1$ et donc à remonter la factorisation de ac. La borne de Landau diminue à $8(\sqrt{3} + 1)$ puisque le degré est 4 et la norme euclidienne du polynôme est $\sqrt{3}$. Il suffit alors de remonter dans $\mathbb{Z}/125\mathbb{Z}$ au lieu de $\mathbb{Z}/625\mathbb{Z}$ (on gagne ainsi une itération).

8.2.6 Combinaison de facteurs

Lorsqu'on a les facteurs de P dans $\mathbb{Z}/p^k\mathbb{Z}[X]$ avec p^k plus grand que le produit du coefficient dominant de P multiplié par la borne de Landau-Mignotte sur les coefficients de P, on commence par tester la divisibilité dans $\mathbb{Z}[X]$ de P par chaque facteur trouvé multiplié par le coefficient dominant de P. Si la division est exacte, on a un facteur irréductible, mais si elle n'est pas exacte il peut se produire qu'un facteur irréductible de P dans $\mathbb{Z}[X]$ soit un produit de deux, voir plusieurs, facteurs modulaires. Il faut donc tester la divisibilité de P dans $\mathbb{Z}[X]$ par toutes les combinaisons possibles de produits de facteurs modulaires (toujours multiplié par le coefficient dominant de P). Cette étape peut être exponentiellement longue si le nombre de facteurs modulaires est grand et si par exemple P est irréductible, bien que les cas soient très rares.

Algorithme de recombinaison

Arguments : un polynôme à coefficients entiers, primitif et sans facteur multiple P de coefficient dominant p_n, la liste L des facteurs de P dans $\mathbb{Z}/p^l Z[X]$ pour l assez grand et p^l

Valeur de retour : la liste F des facteurs de P dans $\mathbb{Z}$.

Initialiser F à vide, initialiser le nombre de facteurs à combine c à 1, entamer une boucle infinie :

– Si c est strictement supérieur au cardinal de L divisé par 2, ajouter le quotient de P par le produit des facteurs de F à F et retourner F
– Initialiser un vecteur $v = (v_1, ..., v_c)$ à c composantes à la valeur $(1, ..., c)$
– Boucle indéfinie intérieure :

1. Faire le produit des facteurs de F d'indice v, multiplier par p_n dans $\mathbb{Z}/p^l Z$, écrire le facteur en représentation symétrique, le rendre primitif et tester si c'est un facteur de P dans $\mathbb{Z}$.
2. Si on a trouvé un facteur, le rajouter à la liste F et supprimer les indices de v de la liste L, terminer cette boucle intérieure.
3. Sinon, incrémenter v de la manière suivante :
   On fait une boucle sur un index m initialisé à la taille de v, diminuant de 1 à chaque itération : on ajoute 1 à l'élement de v d'indice m, si l'élément obtenu est inférieur ou égal au cardinal de $L + m - n$, on arrête cette boucle, sinon on passe à l'itération suivante. Si $m = 0$ à la fin de la boucle, v ne peut pas être incrémenté.
4. Si v ne peut être incrémenté, on incrémente c et on termine la boucle intérieure.
5. Sinon on fait une boucle à nouveau sur m en partant de la valeur actuelle incrémentée de 1, et tant que $m \leq n$ on pose $v_m = v_{m-1} + 1$. Puis on passe à l'itération suivante de la boucle intérieure.

Il existe différentes méthodes qui améliorent la complexité de cette étape :
– La recherche des degré possibles de facteurs fondée sur la factorisation en degrés distincts pour différents nombres premiers permet d'éviter des tests de division si une combinaison de facteurs est d'un degré exclu par la factorisation pour d'autres nombres premiers.
– Le test de divisibilité du coefficient dominant ou du coefficient constant permet aussi d'éviter des divisions complètes de polynômes.

Mais ces astuces n'évitent pas l'énumération de toutes les combinaisons possibles de facteurs et donc la complexité exponentielle. Lorsque les combinaisons d'un petit nombre de facteurs (par exemple 3) échouent, les systèmes récents utilisent l'algorithme knapsack de Van Hoeij basé sur l'algorithme LLL (recherche de base d'un réseau ayant des vecteurs de petite norme) qui permet d'eliminer complètement cette complexité exponentielle.

Exemple :
Toujours le même exemple, il nous restait deux facteurs dans $\mathbb{Z}/125\mathbb{Z}$, le facteur $x^3 + x + 1$ ayant été détecté comme un facteur de $P = x^7 + x^5 + x^3 - x^2 + 1$ dans $\mathbb{Z}$. On teste chacun des facteurs $a_2 = x - 37$ et $c_2 = x^3 + 37x^2 - 6 * x + 27$ séparément, sans succès. On les multiplie alors modulo 125, ce qui donne $x^4 - x + 1$ en représentation symétrique qui est bien un facteur de P (donc un facteur irréductible).

8.3 Factorisation à plusieurs variables

Comme pour le PGCD en plusieurs variables, on se ramène d'abord en une variable, en général on évalue toutes les variables sauf celle correspondant au degré partiel le plus faible. On factorise ensuite en une variable puis on remonte. A chaque étape de remontée, il peut être à nouveau nécessaire de combiner plusieurs

facteurs. Différentes stratégies existent, comme pour le PGCD : factorisarion heuristique (avec reconstruction z-adique), remontée variable par variable ou toutes les variables en même temps comme dans EZGCD. On va présenter ici plus en détails l'algorithme de factorisation heuristique.

Soit P un polynôme en $X_1, ..., X_n$ à coefficients entiers avec $n > 1$, on choisit une des variables par exemple X_n, qu'on notera X dans la suite. On considère P comme un polynôme en $X_1, ..., X_{n-1}$ à coefficients dans $\mathbb{Z}[X]$. On suppose que P est primitif (quitte à extraire son contenu qui est dans $\mathbb{Z}[X]$). On calcule ensuite $P(z)$ pour un entier z tel que[8] $|z| \geq 2|P| + 2$. On factorise $P(z)$ dans $\mathbb{Z}[X_1, ..., X_{n-1}]$:

$$P(z)(X_1, ..., X_{n-1}) = c(z)\Pi_{j=1}^{k} p_j(X_1, ..., X_{n-1}) \tag{13}$$

où c est le contenu du polynôme $P(z)$ (comme polynôme en $n-1$ variables à coefficients entiers). Il s'agit de reconstruire les facteurs de P à partir des p_j et de c. Deux problèmes se posent alors, celui de la recombinaison possible de plusieurs facteurs p_j pour obtenir un facteur irréductible de P, et l'existence d'un facteur entier du contenu c à combiner avec un ou plusieurs p_j pour obtenir ce facteur irréductible. Plus précisément, si P_k est un facteur irréductible de P, on a :

$$P_k(z) = d(z)\Pi_{\text{certains } j} p_j, \quad \text{où } d(z) \text{ divise } c(z) \tag{14}$$

On a le :

Théorème 6 *Soit $P(X_1, ..., X_{n-1}, X)$ un polynôme à coefficients entiers ayant au moins 2 variables. On suppose que P est primitif vu comme polynôme en les variables $X_1, ..., X_{n-1}$ à coefficients dans $Z[X]$. Il existe une majoration C du contenu $|c(z)|$ de P évalué en $X = z$ (plus précisément on peut trouver un entier C tel que $c(z)$ divise C).*
Il existe un nombre fini de z tels que l'un des facteurs irréductibles P_k de P évalué en $X = z$ soit réductible (c'est-à-dire tels que (14) admette plusieurs facteurs p_j distincts)

Preuve
Pour déterminer C, on remarque que les facteurs du contenu de $P(z)$ sont des facteurs communs des coefficients de P évalués en z vu comme polynôme en $X_1, ..., X_{n-1}$ à coefficients dans $\mathbb{Z}[X]$. Donc $c(z)$ divise le générateur de l'idéal engendré par ces coefficients (ce générateur est un polynôme de $\mathbb{Z}[X]$ qui est constant car on a supposé P primitif), on peut aussi dire que deux au moins des coefficients dans $\mathbb{Z}[X]$ de P sont premiers entre eux, alors $c(z)$ divise le coefficient de l'identité de Bézout de ces 2 coefficients vu comme polynômes en X.

Considérons maintenant un facteur irréductible P_k de P de degré d par rapport à X. Pour $X_1, ..., X_{n-1}$ fixés, on factorise P_k sur $\mathbb{C}$:

$$P_k(X) = p_k\Pi_{j=1}^{d}(X - z_j)$$

On va maintenant se restreindre à un domaine des $X_1, ..., X_{n-1}$ sur lequel les z_j ont une dépendance analytique par rapport à $X_1, ..., X_{n-1}$. Pour cela on veut appliquer le théorème des fonctions implicites pour déterminer z_j au voisinage d'une solution

[8] Ici $|P|$ désigne le plus grand coefficient de P en valeur absolue

donnée. On calcule donc la dérivée P'_k de P_k par rapport à X. On sait que P n'a pas de facteurs multiples, donc P_k et P'_k sont premiers entre eux, donc d'après l'identité de Bézout, il existe un polynôme non nul D dépendant de $X_1, ..., X_{n-1}$ et deux polynômes U et V dépendant de $X_1, ..., X_{n-1}, X$ tels que :

$$UP_k + VP'_k = D$$

Si $D(X_1, ..., X_{n-1})$ ne s'annule pas, on va pouvoir appliquer le théorème des fonctions implicites. On se fixe $x_1, .., x_{n-1}$, on calcule dans $\mathbb{C}$ les racines z_j du polynôme $P(x_1, .., x_{n-1}, X)$ pour une solution z_j telle que $P(x_1, .., x_{n-1}, z_j) = 0$, comme D est non nul, on a $P'(x_1, ..., x_{n-1}, z_j) \neq 0$, donc on peut écrire au voisinage de $(x_1, .., x_{n-1})$

$$z_j = z_j(X_1, ..., X_{n-1}), \quad P(X_1, ..., X_{n-1}, z_j) = 0$$

avec des fonctions z_j analytiques. Si D est constant, D ne s'annule pas, sinon quitte à permuter les variables, on peut supposer que le degré de D par rapport à X_1 est non nul. On peut alors se restreindre à une zone $X_1 >> X_2 >> .. >> X_{n-1} >> 1$ où D sera non nul ce qui permet de suivre analytiquement les z_j.

Supposons maintenant qu'il existe un nombre infini de z tels $P_k(z)$ soit réductible. Alors il existe un ensemble infini Z de ces valeurs de z pour lesquels l'un des facteurs à coefficients entiers f_j de $P_k(z)$ correspond à un même sous-ensemble R des racines z_j de P_k et à un même contenu c (puisqu'il y a un nombre fini de combinaisons possibles des racines en facteur et un nombre fini de diviseurs possibles du contenu de P_k). Pour $z \in Z$, on a :

$$f_j(X_1, ..., X_n, z) = c\Pi_{l \in R}(z - z_j), \quad f_j \in \mathbb{Z}[X_1, ..., X_{n-1}]$$

Soit $L(X)$ le polynôme obtenu par interpolation de Lagrange en cardinal$(R) + 1$ points z de Z, égal à f_j en $X = z$. Pour des raisons de degré, on a :

$$L = c\Pi_{l \in R}(X - z_j)$$

donc L est un facteur de P. De plus L est un polynôme en $X_1, ..., X_{n-1}, X$ à coefficients rationnels (par construction). Ceci vient en contradiction avec l'hypothèse P_k irréductible, car on a construit un facteur de P_k à coefficients rationnels L de degré strictement inférieur.

Corollaire

Pour z assez grand, la reconstruction z-adique de $c(z)p_j(z)$ est un polynôme dont la partie primitive est un facteur irréductible de P.

Preuve du corollaire

On prend z assez grand pour que tous les facteurs irréductibles de P évalués en z aient un seul facteur polynomial (i.e. soient de la forme $d(z)p_j(z)$). Quitte à augmenter z, on peut supposer que $|z| > 2CL$ où C est la majoration de $|c(z)|$ et L est la borne de Landau sur les facteurs de P. Alors la reconstruction z-adique de $c(z)p_j(z)$ est $c(z)/d(z)P_j$, donc sa partie primitive est un facteur irréductible de P.

Algorithme de factorisation heuristique à plusieurs variables

Argument : un polynôme P primitif en au moins 2 variables.
Valeur renvoyée : les facteurs irréductibles de P

Choisir la variable X par rapport à laquelle P est de plus bas degré puis factoriser le contenu de P vu comme polynôme à coefficients dans $\mathbb{Z}[X]$. Initialiser un entier z à $2|P|+2$ (où $|P|$ est le plus grand coefficient entier de P en valeur absolue) et une liste L à la factorisation de du contenu de P.
Boucle indéfinie :

- Si $P=1$ renvoyer la liste L des facteurs de P.
- Tant que pgcd$(P(z),P'(z))=0$ incrémenter z de 1.
- Factoriser $P(z)=c(z)\Pi p_j$
- Pour tous les facteurs p_j, déterminer le polynôme P_j tel que $c(z)p_j=P_j(z)$ par remontée z-adique (avec les coefficients de P_j écrit en représentation symétrique, de valeur absolue plus petite que $|z|/2$). Tester si la partie primitive de P_j divise P. Si oui, rajouter un facteur irréductible à la liste L, et diviser P par ce facteur.
- Augmenter z, par exemple remplacer z par la partie entière de $\sqrt{2}z$.

8.4 Preuve de l'identité de Bézout généralisée

Elle se fait par récurrence. Pour $n=2$, c'est l'identité de Bézout usuelle. Pour passer du rang $n-1$ au rang n, on isole P_n dans l'identité à résoudre :

$$\left(\sum_{j=1}^{n-1} Q_j(\Pi_{1\leq k\leq n-1,k\neq j}P_k)\right)P_n+Q_n\Pi_{k\leq n-1}P_k=Q \pmod p$$

Comme P_n est premier avec $\Pi_{k\leq n-1}P_k$, en appliquant Bézout, on trouve deux polynômes Q_n et R_n tels que :

$$R_nP_n+Q_n\Pi_{k\leq n-1}P_k=Q \pmod p \tag{15}$$

Il reste à résoudre

$$\sum_{j=1}^{n-1} Q_j\Pi_{1\leq k\leq n-1,k\neq j}P_k=R_n \pmod p$$

ce que l'on peut faire par hypothèse de récurrence.

8.5 Algorithme de Bézout généralisé

Arguments : une liste $P_1,...,P_n$ de polynômes premiers entre eux 2 à 2 et un polynôme Q à coefficients dans $\mathbb{Z}/p\mathbb{Z}$
Valeur renvoyée : la liste de polynômes $Q_1,...,Q_n$ tels que

$$\sum_{j=1}^{n} Q_j\Pi_{k\neq j}P_k=Q \pmod p$$

On peut commencer par calculer le produit de tous les P_k puis faire une boucle sur j pour calculer les produits des P_k pour $k\neq j$ en divisant le produit complet par P_j (on fait ainsi $n-1$ multiplications et n divisions au lieu de $n(n-1)$ multiplications).
Boucle indéfinie sur n décrémenté de 1 par itération :

– Si $n = 2$, on rajoute à la liste résultat les polynômes Q_1 et Q_2 de l'algorithme de Bézout usuel et on renvoie la liste
– Sinon, on calcule les polynômes R_n et Q_n vérifiant (15), on rajoute Q_n en début de liste, on remplace Q par R_n.

Remarquons que lorsque nous utiliserons cet algorithme, Q sera la différence entre deux polynômes de même degré (le degré de P) et de même coefficient dominant 1, on peut donc remplacer les Q_i par le reste euclidien de Q_i par P_i sans changer l'égalité.

8.6 Pour en savoir plus

Pour factoriser des polynômes ayant des coefficients dans des extensions algébriques, il existe un algorithme assez simple, l'algorithme de Trager, qui n'est pas forcément le plus performant (la recherche est encore active dans ce domaine), cf. le livre de Henri Cohen pp. 142-144.

Pour factoriser sur des corps finis, on peut consulter la thèse de Bernardin disponible sur le web (`http://www.bernardin.lu`).

On peut aussi consulter le code source de Mupad, les routines de factorisation se trouvent dans le répertoire `lib/POLYLIB/FACLIB` après avoir désarchivé la `lib.tar`. Le point d'entrée pour factoriser des polynômes à plusieurs variables sur $\mathbb{Z}$ est le fichier `mfactor.mu`, on observera que l'algorithme utilisé par Mupad est assez différent de celui qu'on a détaillé dans la section précédente.

8.7 Exercices (factorisation des polynômes)

1. Déterminer le nombre de racines de $-x^7 + x^4 + 12x - 5$ comprises entre 0 et 6 (en utilisant les suites de Sturm, on donnera les détails des calculs).
2. Écrire un programme calculant la suite de Sturm d'un polynôme supposé squarefree (on peut tester avec `sqrfree`), en utilisant l'algorithme d'Euclide.
3. Trouver les facteurs de degré 1 s'ils existent de $3x^5 + 25x^4 + 67x^3 + 77x^2 + 55x + 13$ en remontant ses racines dans $\mathbb{Z}/pZ[X]$ pour p premier bien choisi.
4. Factoriser le polynôme $x^5 + x + 1$ par la méthode de Berlekamp.
5. Calculer avec un logiciel les valeurs numériques des racines complexes de $P(x) = x^5 + x + 1$. Trouver les combinaisons de racines dont la somme est entière (aux arrondis près). En déduire la factorisation en facteurs irréductibles sur $\mathbb{Z}$ de P.
6. Factorisation numérique sur $\mathbb{C}$. Écrire un programme qui calcule une racine d'un polynôme à coefficients complexes en utilisant une méthode itérative de type méthode de Newton (avec éventuellement un préfacteur lorsqu'on débute la recherche). Les polynômes seront représentés par la liste de leurs coefficients et l'évaluation faite par la méthode de Horner. Trouver ensuite toutes les racines du polynôme en éliminant la racine trouvée (toujours avec Horner). Trouver les combinaisons de racines correspondant à un facteur à coefficients entiers.
7. Même question pour les facteurs de degré 2 d'un polynôme à coefficients réels sans racines réelles en utilisant la méthode de Bairstow décrite ci-dessous.
   On cherche un facteur $F = x^2 + sx + p$ de P, on calcule le quotient et le reste de la division $P = FQ + R$ par une méthode de type Horner, il s'agit de rendre R (vu comme un vecteur à 2 composantes) nul. On calcule donc $\partial_{s,p}R$ (en cherchant le quotient et le reste de xQ et Q par F, pourquoi ?) et on pose :
   $$(s,p)_{n+1} = (s,p)_n - \lambda(\partial_{s,p}R)^{-1}R(s,p)_n$$
   où λ est un préfacteur compris entre 0 et 1 et ajusté à 1 lorsqu'on est proche du facteur.
8. Soit p un entier premier et P un polynôme à coefficients dans $\mathbb{Z}/p\mathbb{Z}$. On a la relation
   $$gcd(X^{p^k} - X, P) = \prod_{f|P, \deg(f)|k} f, \quad f \text{ irréductible}$$
   En utilisant cette relation, déterminer les degrés des facteurs de
   $$(x^3 + x + 1)(x^4 + x + 1)$$
   modulo 5 et 7 (sans utiliser la commande factor). Peut-on en déduire que $x^3 + x + 1$ et $x^4 + x + 1$ sont irréductibles sur $\mathbb{Z}$?
9. Utiliser les options "verbose" de votre logiciel de calcul formel pour factoriser $x^{202} + x^{101} + 1$ et vérifiez que vous avez compris la méthode utilisée.

10. Montrer que $2x+x^2y+x^3+2x^4+y^3+x^5$ est irréductible sur $\mathbb{Z}$ sans utiliser l'instruction factor à 2 variables (on pourra factoriser pour quelques valeurs de x ou de y)
11. Que se passe-t-il lorsqu'on exécute l'algorithme de Yun dans $\mathbb{Z}/n\mathbb{Z}$?
12. Déterminer les degrés des facteurs de $(x^3+x+1)(x^4+x+1)$ modulo 5 et 7 (sans utiliser la commande factor). Peut-on en déduire que x^3+x+1 et x^4+x+1 sont irréductibles sur $\mathbb{Z}$?
13. Utiliser les options "verbose" de votre logiciel de calcul formel pour factoriser $x^{202}+x^{101}+1$ et vérifiez que vous avez compris la méthode utilisée.
14. Montrer que $2x+x^2y+x^3+2x^4+y^3+x^5$ est irréductible sur $\mathbb{Z}$ sans utiliser directement l'instruction factor (on pourra factoriser pour quelques valeurs de x ou de y)

9 Intégration

9.1 Introduction

Que peut-on espérer d'un système de calcul formel lorsqu'il s'agit de calculer une primitive ? Tout d'abord, on peut espérer qu'il sache résoudre ce que l'on donne en exercice à nos étudiants ! Ceci suppose donc de connaitre quelques méthodes classiques, par exemple : intégration de polynômes (!), polynômes multipliés par exponentielle ou/et fonctions trigonométriques, de polynômes trigonométriques par linéarisation, de fractions rationnelles, de fractions trigonométriques, de fractions de racines carrées de polynômes du second ordre, de fonctions s'y ramenant par une ou plusieurs intégrations par parties ou par changement de fonction (par exemple reconnaissance de formes $F(u)u'$) ou par changement de variables, etc.

Mais au-delà de ces méthodes (qui ont l'avantage de la rapidité mais tiennent parfois plus de la recette de cuisine que de l'algorithme...), on peut se demander si la primitive d'une fonction donnée peut ou non s'exprimer en terme des fonctions "élémentaires". Par exemple, tout le monde "sait" que la fonction e^{x^2} n'admet pas de primitive "simple", encore faut-il donner un sens mathématique précis à cette affirmation. Ceci nécessite de donner une définition rigoureuse du terme fonction élémentaire. On peut alors appliquer un algorithme développé par Risch (pour les extensions dites transcendantes, obtenue par ajout des fonctions exponentielle et logarithme) qui permet de répondre à la question : il s'agit vu de très loin d'une extension de l'algorithme d'intégration des fractions rationnelles.

Cet article se décompose en deux parties principales :

- la section 9.2 présente les définitions de fonctions élémentaires, de tour de variables, et donne deux théorèmes, le théorème de structure de Risch qui permet d'écrire une fonction contenant des exponentielles et des logarithmes comme une fonction élémentaire par rapport à une tour de variable, et le théorème de Liouville qui donne la forme que peut prendre une primitive d'une fonction élémentaire lorsqu'elle est aussi élémentaire.
- la section 9.3 décrit l'algorithme d'intégration de Risch permettant de décider si une fonction élémentaire donnée possède ou non une primitive élémentaire et de la calculer dans le premier cas. Nous ne présentons ici l'algorithme de Risch que pour les extensions transcendantes pures (ln et exp).

Le lecteur intéressé par le cas des extensions algébriques pourra consulter la thèse de Trager. Pour les extensions plus générales (incluant en particulier les fonctions tangente, arctangente), la référence est le livre de Bronstein donnée en section 9.4.

9.2 Fonctions élémentaires

9.2.1 Extensions transcendantes, tour de variables

On se donne une expression $f(x)$ dépendant de la variable x que l'on souhaite intégrer par rapport à x. L'algorithme de Risch s'applique à cette expression si on peut l'écrire comme une fraction rationnelle à plusieurs variables algébriquement indépendantes

$$x, f_1(x), f_2(x, f_1(x)), ..., f_n(x, f_1(x), f_2(x, f_1(x)), ..., f_{n-1}(x, f_1(x), ..., f_{n-2}(x)))$$

où les f_i sont soit l'exponentielle soit le logarithme d'une fraction rationnelle (le corps de base appelé aussi corps de constantes ici est soit $\mathbb{C}$, soit une extension algébrique de $\mathbb{Q}$ ou une extension algébrique d'un corps de fractions rationnelles s'il y a des paramètres). On appelle tour de variables la suite des $x, f_1, ..., f_n$ (chaque étage est donc une exponentielle d'une fraction rationnelle ou le logarithme d'une fraction rationnelle dépendant des étages précédents) et on dira que f est une fonction élémentaire par rapport à cette tour de variables.

L'intérêt de l'écriture d'une expression sous forme de tour est qu'elle est stable par dérivation : si on dérive par rapport à x une fonction élémentaire dépendant d'une tour de variables, on obtient encore une fonction élémentaire dépendant de la même tour de variables. Autrement dit, l'ensemble des fonctions élémentaires pour une tour fixée est un corps différentiel.

Exemples :

- e^{x^2} est bien dans ce cas, pour $n = 1$, f_1 est l'exponentielle de x^2 qui est algébriquement indépendant de x. Les fonctions $(2x^2-1)e^{x^2}$ ou $x/(e^{x^2}-1)$ sont aussi élémentaires par rapport à la tour de variables $\{x, e^{x^2}\}$.
- $x\ln(x)\exp(x)$ est élémentaire par rapport à la tour $\{x, \ln(x), \exp(x)\}$, mais aussi par rapport à la tour $\{x, \exp(x), \ln(x)\}$.
- $xe^{x\ln(x)}$ est élémentaire, en prenant $n = 2$, $f_1 = \ln(x)$ et $f_2 = e^{xf_1}$.
- $x^n = e^{n\ln(x)}$, où n est un paramètre, convient avec comme tour $\{x, \ln(x), e^{n\ln(x)}$
- $e^{\ln(x)}$ ne convient pas car il n'est pas algébriquement indépendant de $x, \ln(x)$ mais on peut le réécrire sous une forme acceptable puisque $e^{\ln(x)} = x$.
- $e^{\ln(x)/2}$ ne convient pas non plus car son carré est égal à x. Une réécriture ne suffit pas, cet exemple est bien sûr une extension algébrique et non transcendante.

Dans la suite, on va s'intéresser aux tours de variables dans lesquelles on a effectué des simplifications évidentes. On élimine les $\ln \circ \exp$ de la manière suivante : si $f_k = \ln(g_k)$, on regarde si g_k vu comme fraction en $f_1, ..., f_{k-1}$ possède un facteur f_j^m (avec $m \in \mathbb{Z}$) lorsque $f_j = \exp(g_j)$ est une exponentielle. Si c'est le cas, on a $f_k = mg_j + \ln(g_k/g_j^m)$. On change alors de tour en remplaçant f_k par $\tilde{f_k} = \ln(g_k/g_j^m) = f_k - mg_j$. On élimine aussi les $\exp \circ \ln$, si $f_k = \exp(g_k)$, pour $j < k$ si f_j est un logarithme, on regarde si $c_j = \partial_{f_j} g_k|_{f_j=0}$ est un entier, si c'est le cas on remplace f_k par $\tilde{f_k} = f_k/g_k^{c_j}$.

Exemples :

$$\begin{aligned} \ln(\frac{e^{x^2}+1}{e^{x^2}}) &\rightarrow -x^2 + \ln(e^{x^2}+1) \\ e^{3\ln(x)+\ln(x)^2+5} &\rightarrow x^3 e^{\ln(x)^2+5} \end{aligned}$$

9.2.2 Théorème de structure de Risch

On voit donc qu'il est nécessaire de disposer d'un algorithme pour décider si des exponentielles et logarithmes sont algébriquement indépendants. Cet algorithme est basé sur un théorème de structure dû à Risch :

Théorème 7 *Soit $f = \ln(g(x))$ le logarithme d'une fonction élémentaire g par rapport à une tour de variables T, alors soit f est algébriquement indépendant des variables de T, soit f est élémentaire et plus précisément combinaison linéaire rationnelle des logarithmes et des arguments des exponentielles de la tour T.*

Soit $f = \exp(g)$ l'exponentielle d'une fonction élémentaire g par rapport à une tour de variables T, alors soit f est algébriquement indépendante des variables de T, soit il existe n tel que f^n soit élémentaire par rapport à T (on peut alors appliquer le cas précédent à $ng = \ln(f^n)$)

Démonstration :
Commençons par le cas de l'exponentielle. On considère le polynôme minimal de $f = \exp(g)$:

$$a_n f^n + ... + a_0 = 0, \quad a_n \neq 0, a_0 \neq 0$$

où les a_i sont des fractions rationnelles en T. On dérive et on applique $f' = g'f$:

$$(a'_n + na_n g')f^n + ... + (a'_k + ka_k g')f^k + ... = 0$$

c'est un multiple du polynôme minimal donc il existe une fraction rationnelle C (par rapport à la tour de variables) telle que :

$$\forall k, \quad (a'_k + ka_k g') = Ca_k$$

Comme $a_n \neq 0$, cela entraine $a'_n/a_n + ng' = C$. Le coefficient constant a_0 est aussi non nul, donc $a'_0/a_0 = C$ et

$$ng' = a'_0/a_0 - a'_n/a_n \Rightarrow ng = \ln(\frac{a_0}{a_n}) + k$$

où k est constant, donc $f^n = \exp(ng) = e^k a_0/a_n$ est élémentaire.

Passons au cas du logarithme, supposons que $f = \ln(g)$ dépende algébriquement de la tour T, on va commencer par montrer que f est élémentaire. On écrit :

$$a_n f^n + ... + a_0 = 0$$

où les a_i sont des fractions rationnelles en T. On dérive en appliquant $f' = g'/g$:

$$a'_n f^n + (na_n f' + a'_{n-1})f^{n-1} ... + a_1 f' + a'_0$$

Comme f' est une fraction rationnelle en T, le polynôme $a'_n X^n + (na_n f' + a'_{n-1})X^{n-1} + ... + a_1 f' + a'_0$ qui annule f doit être un multiple du polynôme minimal de f, il existe donc une fraction rationnelle C par rapport à T telle que :

$$a'_n = Ca_n \quad (na_n f' + a'_{n-1}) = Ca_{n-1} \quad ...$$

On en déduit f' :

$$f' = \frac{\frac{a'_n}{a_n}a_{n-1} - a'_{n-1}}{na_n} = \left(\frac{-a_{n-1}}{na_n}\right)'$$

donc il existe une constante c telle que :

$$f = \frac{-a_{n-1}}{na_n} + c$$

donc f est élémentaire par rapport à la même tour T que g.

Montrons maintenant qu'un logarithme $f = \ln(g)$ qui est élémentaire par rapport à une tour de variable T est combinaison linéaire à coefficients rationnelles

des logarithmes et des arguments des exponentielles de T[9]. Soit X la dernière variable de la tour T. On factorise maintenant le numérateur et le dénominateur de g en $\prod_j P_j^j$ où les P_j sont sans facteurs multiples et premiers entre eux 2 à 2 (par rapport à X), il existe C indépendant de X tel que :

$$g = C\prod_{j\in\mathbb{Z}} P_j^j \Rightarrow \ln(g) = \ln(C) + \sum_{j\in\mathbb{Z}} j\ln(P_j) \tag{16}$$

Alors $f' = \ln(C)' + \sum_j jP_j'/P_j$ donc $\prod P_j f'$ est un polynôme en X. Soit N/D la fraction irréductible représentant f, on a :

$$f' = \frac{N'D - ND'}{D^2}$$

on vient donc de montrer que :

$$\left(\prod_j P_j\right)\frac{N'D - ND'}{D^2} \text{ est un polynôme en } X \tag{17}$$

Soit P un facteur irréductible de D de multiplicité k tel que $D = P^kQ$ (donc P premier avec Q, mais P est aussi premier avec N car $f = N/D$ est irréductible). Alors en simplifiant numérateur et dénominateur par P^{k-1}, on a :

$$\left(\prod_j P_j\right)\frac{N'PQ - N(kP'Q + PQ')}{P^{k+1}Q^2} \text{ est un polynôme en } X. \tag{18}$$

On en déduit, après simplification d'au plus un facteur P au dénominateur avec l'un des P_j, que P^k divise $N'PQ - N(kP'Q + PQ')$ donc P divise P'. Ceci n'est possible que si $P = 1$ (et donc le dénominateur de f est égal à 1) ou si la variable X est une exponentielle et $P = X$.

Montrons que ce deuxième cas est en fait exclus : en effet si $P = X = \exp(Y)$ est une exponentielle, on a alors $D = X^k$ et $Q = 1$. Comme $P' = Y'X$, (18) devient :

$$\left(\prod_j P_j\right)\frac{X(N' - kNY')}{X^{k+1}} \text{ est un polynôme en } X$$

Comme X ne divise pas N, N possède donc un coefficient constant a_0 non nul. Le coefficient constant de $N' - kNY'$ est $a_0' - ka_0Y'$. Si ce terme était nul alors $a_0' = ka_0Y'$ donc $a_0 = c\exp(kY) = cX^k$ or a_0 ne dépend pas de X donc $c = 0$ donc $a_0 = 0$, absurde. Donc X ne divise pas $N' - kNY'$. Comme X^{k+1} divise $\prod P_j X(N' - kNY')$, on en déduit que X^k divise un des P_j. Donc $k = 1$ et $P_j = XQ_j$. Revenons maintenant à (16), on a :

$$f = \ln(g) = \ln(C) + j\ln(XQ_j) + \sum_{l\neq j} l\ln(P_l)$$

on dérive :

$$f' = \ln(C)' + jY' + j\frac{Q_j'}{Q_j} + \sum_{l\neq j} l\frac{P_l'}{P_l}$$

[9] cette preuve peut être sautée en première lecture

on voit qu'il n'est plus nécessaire de multiplier f' par P_j pour avoir un polynôme, multiplier par Q_j suffit, plus précisément

$$\left(\prod_{l\neq j} P_l\right) Q_j \frac{N'D - ND'}{D^2} \text{ est un polynôme en } X.$$

donc X^{k+1} divise $\left(\prod_{l\neq j} P_l\right) Q_j X(N' - kNY')$ ce qui est impossible.

Donc $D = 1$ dans tous les cas et on a $f = N$. Donc

$$f' = N' = \ln(C)' + \sum_j jP_j'/P_j \text{ est un polynôme par rapport à } X$$

On en déduit que les P_j ne dépendent pas de X sauf si X est une exponentielle et $P_j = X$. Dans les deux cas N' ne dépend pas de X donc le polynôme N est de degré 0 ou 1 en X (si X est une exponentielle, N est forcément de degré 0)

- Si $X = \exp(Y)$ est une exponentielle (avec Y élémentaire ne dépendant pas de X), alors $f = N$ est indépendant de X. On retire jY à f et on divise g par X^j (en posant $j = 0$ si aucun des P_j n'est égal à X), qui devient indépendant de X, on conserve ainsi l'égalité $f = \ln(g)$ mais avec une variable de moins dans la tour de variables par rapport à laquelle f et g sont élémentaires.
- Si X n'est pas une exponentielle, $N = cX + d$ avec c dans le corps de constantes, et d indépendant de X. Si $X = x$, on a $g = \exp(cx+d)$ qui n'est rationnel que si $c = 0$. On a alors d donc f et g constants. Si $X = \ln(Y)$ est un logarithme (avec Y élémentaire ne dépendant pas de X), alors $\forall j, P_j = 1$ donc g est élémentaire indépendante de X. On a alors :

  $$f = N = c\ln(Y) + d = \ln(g)$$

  avec c dans le corps des constantes, d et g élémentaires indépendants de X. On cherche maintenant la fonction élémentaire d. Cette fonction n'est pas le logarithme d'une fonction élémentaire en général car c n'est pas forcément entier, mais d' a les mêmes propriétés que la dérivée du logarithme d'une fonction élémentaire. On peut donc reprendre le même raisonnement mais avec une variable de moins dans la tour de variables. Si la tour qu'on a choisie est normalisée, alors Y ne contient au numérateur et au dénominateur aucune puissance d'une exponentielle d'une variable de la tour donc le polynôme P_j du cas précédent ne peut provenir de Y ce qui entraine que j est bien entier dans le cas précédent (bien que c ne le soit pas forcément).

Après avoir fait une récurrence sur le nombre de variables de la tour, on a donc f qui s'exprime comme combinaison linéaire à coefficients entiers des arguments g_k des variables exponentielles $f_k = \exp(g_k)$ de la tour et à coefficients a priori quelconque des variables logarithmes $f_l = \ln(g_l)$ de la tour :

$$f = \sum_k j_k g_k + \sum_l x_l \ln(g_l) = \ln(g)$$

Comme g est élémentaire, $h = g/\prod_k \exp(g_k)^{j_k}$ est élémentaire de logarithme $\sum_l x_l \ln(g_l)$. Montrons que si les arguments des $\ln$ sont des polynômes sans facteurs multiples, alors les x_l sont entiers. Rappelons que les $\ln(g_l)$ sont algébriquement indépendants, on peut donc construire des polynômes irréductibles I_l par

rapport aux variables de la tour tels que I_l divise une fois g_l mais ne divise pas les g_k précédents. Soit $h = \prod_{j\in\mathbb{Z}} P_j^j$ la factorisation sans facteurs multiples de h. On dérive alors $\ln(h)$ ce qui donne :

$$\sum_l x_l g_l'/g_l = \sum_j j P_j'/P_j$$

où $\prod_j P_j^j$ est la décomposition sans facteurs multiples de h. Comme I_l divise un et un seul des P_j on en déduit que x_l est égal au j correspondant et est donc entier. (Remarque : si on n'impose pas aux arguments des logarithmes d'être des polynômes sans facteurs carrés, on obtiendrait ici des coefficients rationnels).

En pratique :

On peut effecter l'algorithme de la manière suivante :

- on cherche les variables généralisées de l'expression qui dépendent de x.
- On ajoute les variables généralisées en commençant par la moins longue
- Si c'est un logarithme, on extrait les puissances des exponentielles précédentes dont il dépend. On cherche des relations entre fonctions $\ln$ en les réécrivant comme combinaison linéaire de $\ln$ indépendants. Pour avoir des $\ln$ indépendants, on se ramène d'abord à des polynômes sans facteurs multiples en utilisant la relation $\ln(a/b) = \ln(a) - \ln(b)$ et en écrivant la factorisation sans facteurs multiples de chaque polynôme argument, puis on extrait le PGCD 2 à 2 des arguments de logarithmes jusqu'à obtenir des arguments de $\ln$ premiers entre eux.
- Si c'est une exponentielle, on teste si son argument est combinaison linéaire à coefficients rationnels :
  - des arguments des exponentielles précédentes,
  - des $\ln$ des logarithmes précédents,
  - de $\ln(x)$ et de $i * \pi$.

  Pour cela on substitue les $\ln$ par des indéterminées, et on dérive une fois par rapport à cette indéterminée, le résultat doit être un rationnel, pour les variables exponentielles, il faut réduire au même dénominateur et résoudre le système linéaire obtenu en identifiant les coefficients du numérateur. Si l'exponentielle est indépendante des précédentes, on extrait de l'exponentielle à rajouter la partie linéaire de la dépendance en les $\ln$ précédents si le coefficient correspondant est entier. Par exemple, on réécrit :

$$xe^{2\ln(x)+\ln(x)^2} = x^3 e^{\ln(x)^2}$$

Remarque

On n'est pas obligé de se limiter aux seules fonctions logarithmes et exponentielles, l'essentiel est de pouvoir tester l'indépendance algébrique des expressions créées. Pour éviter d'avoir à introduire des exponentielles et logarithmes complexes dans une expression réelle, on peut autoriser par exemple des extensions en tangente ou en arctangente.

9.2.3 Théorème de Liouville

On a vu que la dérivée d'une fonction élémentaire dépendant d'une tour de variables est une fonction élémentaire dépendant de la même tour de variables.

Réciproquement, supposons qu'une fonction élémentaire admette une primitive qui soit élémentaire, c'est-à-dire qu'elle doit être une fraction rationelle par rapport à une tour de variables mais pas forcément identique à celle de départ. Alors, si une telle écriture existe, à des termes logarithmiques près, elle ne peut dépendre que de la même tour de variables, plus précisément on a le théorème de Liouville :

Théorème 8 *Soit f une fonction élémentaire par rapport à une tour de variables T et un corps de constantes K admettant une primitive élémentaire F. Alors il existe un nombre fini de constantes $c_1, ..., c_n$ et de fonctions élémentaires $v_1, ..., v_n$ par rapport à T avec comme corps de constantes une extension algébrique K' de K tel que $F - \sum_k c_k \ln(v_k)$ soit élémentaire par rapport à T et K.*

Preuve :[10]
Soit f élémentaire de tour T_1 (corps K) et F sa primitive supposée élémentaire de tour T_2 et de corps K' une extension algébrique de K. On commence par rajouter après les élements de T_1 les élements nécessaires de T_2 pour obtenir une tour T par rapport à laquelle f et F sont élémentaires (plus précisément F sera élémentaire quitte à autoriser des puissances fractionnaires des variables exponentielles de T_1). Le théorème de structure de Risch permet de faire cela, en effet on regarde pour chaque élément de T_2 s'il est algébriquement indépendant des éléments de T_1 ou non. S'il l'est, on le rajoute à la tour T, s'il ne l'est pas alors dans le cas d'un logarithme il est élémentaire et dans le cas d'une exponentielle, une de ses puissances est élémentaire. Donc F est bien une fraction rationnelle par rapport aux éléments logarithmiques de T_1, aux racines n-ième des éléments exponentiels de T_1 et à des éléments de T_2 dans cet ordre (le corps des constantes étant K').

Première étape :
Commençons par les éléments restant de T_2. Soit X_k l'élément au sommet de la tour T. La dérivée f de F par rapport à X_k ne dépend pas de X_k. Donc soit F ne dépend pas de X_k et on passe à la variable suivante, soit $X_k = \ln(v_k)$ est un logarithme et $F = c_k \ln(v_k) + d_k$ avec $c_k \in K'$ et v_k et d_k indépendants de X_k. S'il n'y a pas d'autres éléments restants de T_2, on passe à la 2ème étape. Sinon soit X_{k-1} la variable suivante (juste en-dessous de X_k dans la tour). En dérivant, on a :

$$F' = c_k \frac{v_k'}{v_k} + d_k' = f$$

Supposons que v_k dépende de X_{k-1}, on fait alors un raisonnement analogue à celui de la preuve du théorème de structure de Risch, en décomposant v_k en produit/quotient de facteurs sans multiplicités $v_k = \prod P_j^j$ et en écrivant $d_k = N/D$ on a :

$$\left(\prod_j P_j\right) \frac{N'D - ND'}{D^2}$$

est un polynôme en X_{k-1}. On en déduit comme précédemment que $D = 1$, $N' = d_k'$ est indépendant de X_{k-1}. Comme on a supposé que v_k dépend de X_{k-1}, $X_{k-1} = \exp(Y_{k-1})$ est alors une exponentielle, $N = d_k$ ne dépend pas de X_{k-1} et l'un des $P_j = X_{k-1}$ (sinon tous les P_j seraient constants en X_{k-1} donc v_k aussi).

[10] Peut être omise en première lecture

On élimine alors la variable X_{k-1} en écrivant $\ln(v_k) = jY_{k-1} + \ln(w_k)$, avec Y_{k-1} et w_k élémentaires et indépendants de X_{k-1}.

Si v_k est indépendant de X_{k-1}, alors d'_k aussi donc soit d_k est indépendant de X_{k-1} et on passe à la variable suivante, soit X_{k-1} est un logarithme et $d_k = c_{k-1}\ln(v_{k-1}) + d_{k-1}$. En continuant pour toutes les variables restantes de T_2, on obtient

$$F = \sum_k c_k \ln v_k + d$$

avec d et v_k élémentaires pour T_1 (avec exponentielles modifiées en en prenant une racine n-ième) et K'.

Deuxième étape Il s'agit de montrer que pour les exponentielles, il n'est en fait pas nécessaire de prendre de racines n-ième. La compréhension de cette étape demande un peu de familiarité avec l'algorithme de Risch (cf. infra). On va faire la preuve pour la variable au sommet de la tour T_1 si c'est une exponentielle. On verra dans le déroulement de l'algorithme de Risch que pour les autres variables, il y a appel récursif de l'algorithme d'intégration, donc traiter la variable au sommet suffira. Soit donc $\exp(Y)$ la variable au sommet de la tour T_1, on note $X = \exp(Y/n)$ la racine n-ième de cette variable qui est utilisée pour exprimer $F = \sum c_k \ln v_k + N/D$ comme une fraction rationnelle en X alors que $f = F'$ est une fraction rationnelle en X^n. On a donc :

$$\sum c_k \frac{v'_k}{v_k} + \frac{N'}{D} = f = \text{fraction rationnelle en } (X^n)$$

Notons que le fait que X soit une exponentielle est essentiel, car par exemple l'intégrale d'une fraction rationnelle dépendant de x^n comme x^3 ou $1/(x^3-1)$ ne s'exprime pas en fonction de x^3. On traite d'abord la partie polynomiale généralisée de f en X^n :

$$\sum_{j\in\mathbb{Z}} a_j (X^n)^j$$

Son intégrale est un polynôme généralisé, éventuellement dépendant de X, soit $\sum_{j\in\mathbb{Z}} A_j X^j$. On dérive, et on obtient pour k non multiple de n, $A_k Y/n + A'_k = 0$ dont $A_k = 0$ est solution. La partie polynôme généralisé ne dépend donc que de X^n. On effectue aussi les intégrations par parties pour réduire le dénominateur de f à un polynôme sans facteurs multiples (réduction de Hermite), ce qui se fait en introduisant des fractions rationnelles en X^n uniquement. Reste la partie logarithmique. On utilise le critère du résultant, les coefficients des logarithmes sont les racines c_k du polynôme en t

$$\mathrm{Res}_X(D, N - tD')$$

où ces racines doivent être indépendantes de x (puisque F existe) et les v_k correspondants sont égaux à

$$\gcd(D, N - c_k D')$$

Or comme X est une exponentielle, D' est un polynôme en X^n, de même que D et N, donc v_k est un polynôme en X^n.

Troisième étape Il reste enfin à montrer que seuls les c_k et v_k nécessitent une extension algébrique de K. Ceci est encore une conséquence de l'algorithme de Risch, la construction de la partie polynomiale (éventuellement généralisée) et de

la partie fractionnaire ne font en effet intervenir que des coefficients dans le corps K.

9.3 L'algorithme de Risch

On suppose dans la suite qu'on s'est ramené à une fraction rationnelle par rapport à une tour de variables (où on a effectué les simplifications évidentes $\ln \circ \exp$, ainsi que $\exp \circ \ln$, dans le premier cas en extrayant les facteurs évidents en les variables précédentes exponentielles, dans le deuxième cas en extrayant la partie linéaire à coefficient entier en les variables logarithmes précédentes). On note X la variable au sommet de la tour et N_0/D_0 l'écriture de la fonction élémentaire comme fraction irréductible avec N_0 et D_0 polynômes en X.

Exemples

$$\int (2x^2+1)e^{x^2} \quad X = e^{x^2} \quad N_0 = (2x^2+1)X, D_0 = 1$$

$$\int \frac{x\ln(x)}{x+\ln(x)} \quad X = \ln(x) \quad N_0 = xX, D_0 = x + X$$

La première étape va consister à se ramener à un dénominateur sans facteurs multiples. Elle est analogue au cas des fractions rationnelles de x et est basée sur l'identité de Bézout entre P et P' vu comme polynômes en la variable du haut de la tour. Il apparait toutefois une difficulté pour les extensions exponentielles, à savoir que $X = e^f$ et $X' = f'X$ ne sont pas premiers entre eux comme polynômes en X, on devra traiter le pôle 0 d'une fraction rationnelle en une exponentielle X comme on traite l'intégration d'un polynôme en x. Si P est sans facteurs multiples et premier avec X, alors $P(X)$ et $P(X)' = f'XP'(X)$ vu comme polynômes en X n'ont pas de facteurs en commun.

On commence donc, si X est une exponentielle et D_0 un multiple de X, par appliquer Bézout pour décomposer la fraction N_0/D_0 en :

$$\frac{N_0}{D_0} = \frac{N_1}{D_1} + \frac{P}{X^k}, \quad \gcd(X, D_1) = 1, D_0 = X^k D_1$$

On isole aussi la partie polynômiale en effectuant la division euclidienne de N_0 par D_0 (ou de N_1 par D_1 si X est une exponentielle), on obtient alors une écriture sous la forme :

$$\frac{N}{D} + \sum_j a_j X^j$$

où la somme sur j est finie et porte sur des entiers positifs ou nul si X n'est pas une exponentielle, ou sur des entiers relatifs si X est une exponentielle.

On effectue la même écriture sur la partie fractionnaire de F, et en identifiant les parties polynomiales et éventuellement la partie polaire en 0 si X est une exponentielle, on peut séparer l'intégration en 2 parties : intégration de la partie polynomiale (généralisée) et intégration de la partie fractionnaire propre.

Exemples

– $(2x^2+1)e^{x^2} = 0 + (2x^2+1)X$ est un polynôme,

–
$$\frac{x\ln(x)}{x+\ln(x)} = \frac{xX}{x+X} = -\frac{x^2}{x+X} + x$$
la partie polynomiale est x (de degré 0 en X), la partie fractionnaire est $-x^2/(x+X)$

–
$$\frac{x(e^{2x}+1)}{e^x(e^x+1)^2} = \frac{x(X^2+1)}{X(X+1)^2} = -\frac{2x}{(X+1)^2} + xX^{-1}$$
la partie polynôme généralisé est xX^{-1}

9.3.1 Intégration d'une fraction propre

9.3.2 Réduction sans facteurs multiples

On factorise D en $\prod_i P_i^i$ avec P_i sans facteurs multiples (et les P_i premiers entre eux 2 à 2) et on décompose en éléments simples relativement à cette factorisation (en appliquant Bézout) :

$$\frac{N}{D} = \sum_{i>0} \frac{N_i}{P_i^i}$$

Pour chaque polynome P_i, on applique Bézout à P_i et P_i' :

$$N_i = A_i P_i + B_i P_i' \Rightarrow \frac{N_i}{P_i^i} = \frac{A_i}{P_i^{i-1}} + \frac{B_i P_i'}{P_i^i}$$

on intègre par parties le second terme

$$\int \frac{N_i}{P_i^i} = \int \frac{A_i}{P_i^{i-1}} - \frac{B_i}{(i-1)P_i^{i-1}} + \int \frac{B_i'}{(i-1)P_i^{i-1}}$$

on rassemble les deux intégrales ayant P_i^{i-1} au dénominateur et on recommence jusqu'à avoir une puissance 1 au dénominateur. Il reste alors à intégrer une somme de fractions du type N/D avec D et D' premiers entre eux.

Exemple

On reprend le dernier exemple de la section précédente pour éliminer la puissance 2 au dénominateur : $N_2 = 2x$ et $P_2 = (X+1)$ avec $X = e^x$. On a $P_2' = X$, donc $A_2 = 2x$ et $B_2 = -2x$:

$$\int \frac{2x}{(X+1)^2} = \int \frac{2x}{P_2} + \int \frac{-2xP_2'}{P_2^2} = \int \frac{2x}{P_2} + \frac{2x}{P_2} - \frac{2}{P_2}$$

il reste donc à intégrer $(2x-2)/(e^x+1)$.

9.3.3 La partie logarithmique

Comme on l'a vu lors de la preuve du théorème de structure de Risch, si on dérive une fraction en X, le dénominateur de la dérivée ne peut se décomposer qu'en produit de facteurs de multiplicité supérieure ou égale à 2. Il en résulte que

la fraction à intégrer résiduelle (encore notée $f = N/D$) après l'étape de réduction ci-dessus ne peut provenir que de la dérivation de $F = \sum_k c_k \ln(v_k)$:

$$f = \frac{N}{D} = F' = (\sum_k c_k \ln(v_k))' = \sum_k c_k \frac{v_k'}{v_k}$$

En identifiant les décompositions en éléments simples de F' et f, on montre également que les v_k divisent D, plus précisément on peut imposer aux v_k d'être premiers entre eux 2 à 2 et dans ce cas $D = \prod v_k$. Donc :

$$\sum_k c_k \frac{v_k'}{v_k} = \frac{N}{\prod_k v_k} = \frac{N}{D}$$

et :

$$N = \sum_k c_k v_k' \prod_{j \neq k} v_j$$

Soit t un paramètre, formons le polynôme $N - tD'$:

$$N - tD' = \sum_k \left((c_k - t) v_k' \prod_{j \neq k} v_j \right)$$

donc le pgcd en X des polynômes $N - tD'$ et D est :

- si t n'est égal à aucun des c_k, $N - tD'$ est premier avec v_k pour tout k car v_k divise $\sum_{l \neq k} (c_l - t) v_l' \prod_{j \neq l} v_j$ et $v_k' \prod_{j \neq k} v_j$ est premier avec v_k. Donc le pgcd est 1.
- si t est égal à l'un des c_k, alors le pgcd est le produit des v_k tels que $c_k = t$ (notons que dans ce cas on peut rassembler ces v_k à l'intérieur d'un même logarithme)

Considérons le polynôme R de la variable t égal au résultant par rapport à X des polynômes D et $N - tD'$ (rappelons qu'il s'agit du déterminant du système linéaire $AD + B(N - tD') = 1$ où les inconnues sont les coefficients des polynômes A et B, ce déterminant est nul si et seulement si le système n'a pas de solution donc si et seulement si D et $N - tD'$ ne sont pas premiers entre eux), alors ce polynôme en t s'annule si et seulement si $t = c_k$. On cherche les racines c_k en t de ce polynôme, elles doivent être indépendantes de x si F est élémentaire, et dans ce cas la primitive F de $f = N/D$ vaut

$$F = \sum_{c_k \text{ racine de } R} c_k \ln(\text{gcd}(N - c_k D', D))$$

Exemples

–

$$\frac{2x - 2}{e^x + 1}, \quad D = X + 1, D' = e^x = X, \quad N - tD' = 2x - 2 - tX$$

On calcule $R = -2 * x - t + 2$, l'unique racine est $t = 2 - 2x$ qui n'est pas constante donc cette fonction n'admet pas de primitive élémentaire.

–

$$\frac{(2x^2-x-2)X-1}{X^2+(x+1)X+x}, \quad X=\exp(x^2+x)$$

On a $D'=2(2x+1)X^2+(1+(2x+1)(x+1))X+1$

$$R=-(2x-1)(x+1)(2x+1)(x-1)^2(t+1)(t-1)$$

les racines en t sont constantes et égales à 1 et -1, donc $c_1=1$ et v_1 =gcd$(N-D',D)=X+1$ et $c_2=-1$, v_2 =gcd$(N+D',D)=x+X$ donc :

$$\int \frac{(2x^2-x-2)X-1}{X^2+(x+1)X+x}=\ln(X+1)-\ln(x+X)$$

Remarque importante
Pour les extensions exponentielles ou logarithmiques, la dérivée de la partie logarithmique calculée comme ci-dessus contiendra en général une partie entière constante par rapport à X, il faut donc retirer cette partie entière à la partie polynomiale.

9.3.4 La partie polynomiale (généralisée)

On doit résoudre :

$$(\sum_j A_jX^j)'=\sum_j a_jX^j$$

avec une somme sur $j\in\mathbb{Z}$ si X est une exponentielle et $j\in\mathbb{N}$ sinon.

Si $X=x$, $j\geq 0$ et la résolution est immédiate : on prend $A_0=0$ et $A_{j+1}=a_j/(j+1)$.

9.3.5 Extension logarithmique

Si $X=\ln(Y)$ est un logarithme, $j\geq 0$ et on doit résoudre :

$$\sum_{j\geq 0}(A_j'+(j+1)A_{j+1}\frac{Y'}{Y})X^j=\sum_j a_jX^j$$

Soit k la plus grande puissance non nulle de f ($a_j=0$ si $j>k$ et $a_k\neq 0$). Pour $j>k$, on a :

$$A_j'+(j+1)A_{j+1}\frac{Y'}{Y}=0$$

On résout pour des valeurs de j décroissante, pour j suffisamment grand, on a $A_{j+1}=0$ car la somme sur j est finie, donc A_j est constant. Si $A_j\neq 0$, alors au rang $j-1$, on a $A_{j-1}'=-jA_jY'/Y$ qui n'admet pas de solutions car A_{j-1} ne peut pas dépendre de $X=\ln(Y)$. On en déduit que pour $j>k+1$, on a $A_j=0$ et A_{k+1} est constant. En fait la valeur constante de A_{k+1} sera déterminée par une condition de compatibilité en résolvant l'équation au rang du dessous. On continue la résolution de

$$A_j'+(j+1)A_{j+1}\ln(Y)'=a_j$$

par valeur décroissante de j, à chaque rang on va déterminer A_j à une constante près en résolvant un problème d'intégration (par appel récursif de l'algorithme de

Risch, mais si $j \neq 0$ sans autoriser l'ajout de nouveaux logarithmes sauf $\ln(Y)$) et la valeur de la constante de A_{j+1} (on fait varier A_{j+1} de la constante nécessaire pour absorber le terme en $\ln(Y)$ qui apparait lors de l'appel récursif de Risch). Au rang 0, on est ramené à un problème d'intégration avec une variable de moins (la constante indéterminée dans A_1 peut par exemple être choisie comme le coefficient constant de $\ln(Y)$ s'il en apparait un en intégrant).

Exemple
$X = \ln(x^2+1)$ et on cherche l'intégrale de X^2. On a donc A_3 est constant,

$$A_2' + 3A_3 \ln(x^2+1)' = 1$$

La primitive de 1 est élémentaire et ne fait pas intervenir de $\ln$ donc $A_3 = 0$ et $A_2 = x + C_2$. Au rang 1, on a :

$$A_1' + 3x\frac{2x}{x^2+1} + C_2 \ln(x^2+1)' = 0$$

On calcule la primitive de $6x^2/(x^2+1)$ qui doit être une fraction rationnelle à un $C\ln(x^2+1)$ près, on voit que ce n'est pas le cas donc X^2 n'admet pas de primitive élémentaire. Remarque : si on avait voulu intégrer X au lieu de X^2, la même méthode montre que la primitive existe, car au rang 0 il n'y a plus de contraintes sur les $\ln$ qu'on peut rajouter.

9.3.6 Extension exponentielle

Si $X = \exp(Y)$ est une exponentielle, on doit résoudre :

$$\sum_j (A_j' + jY'A_j)X^j = \sum_j a_j X^j$$

Ceci va se faire degré par degré :

$$A_j' + jY'A_j = a_j \tag{19}$$

Exemple
Pour calculer $\int a(x)\exp(x^2)$, on a $j = 1$, et on doit résoudre l'équation différentielle :

$$A_1' + 2xA_1 = a(x)$$

Pour $j = 0$, il suffit de faire un appel récursif à l'algorithme de Risch, mais pour $j \neq 0$, la situation se complique ! Notons Z la variable située juste en-dessous de X dans la tour de variables (dans l'exemple ci-dessus $Z = x$), il s'agit de résoudre :

$$y' + fy = g \tag{20}$$

avec f, g élémentaires par rapport à une tour dont le variable au sommet est Z, on cherche y élémentaire par rapport à cette tour (ici $f = jY'$ est une dérivée mais dans certains cas nous devrons résoudre par appel récursif des équations du type ci-dessus où f ne sera pas une dérivée).

Élimination des dénominateurs
Soit P un facteur irréductible du dénominateur de y, notons $\alpha < 0$ la valuation de y par rapport à P, β celle de f, γ celle de g. Si P n'est pas une exponentielle,

la valuation de y' est $\alpha - 1$, celle de fy est $\alpha + \beta$. Si $\beta \neq -1$, il n'y a pas de simplification possible dans le membre de gauche donc $\alpha + \min(\beta, -1) = \gamma$. Autrement dit, si $\beta \geq 0$ alors $\alpha = \gamma + 1$ et si $\beta < -1$ alors $\alpha = \gamma - \beta$. On observe que $\gamma < 0$ donc P est un facteur du dénominateur g_d de g. De plus, on va montrer que la valuation α de P dans y est l'opposé de celle de P dans :

$$D = \frac{\gcd(g_d, \partial_Z g_d)}{\gcd(c, \partial_Z c)}, \quad c = \gcd(f_d, g_d) \tag{21}$$

En effet, si $\beta \geq 0$, P ne divise pas f_d donc ne divise pas c, donc la valuation de P dans D est $-\gamma - 1$. Si $\beta < -1$, alors $\alpha = \gamma - \beta < 0$ entraine $-\gamma > -\beta$ donc la valuation de P dans c est $-\beta$ et la valuation de P dans D est $-\gamma - 1 - (-\beta - 1)$.

Si $\beta = -1$, s'il n'y a pas de simplifications dans le membre de gauche pour les termes de plus petite puissance en P, alors $\alpha = \gamma + 1$. S'il y a simplification, on décompose en éléments simples (avec Bézout) puis on ordonne par puissances croissantes de P :

$$y = N_1 P^\alpha + ..., f = N_2 P^{-1} + ...,$$

avec N_1, N_2 de degré plus petit que P, puis on remplace dans (20). On cherche les termes de valuation $\alpha - 1$ en P qui doivent se simplifier :

$$\alpha N_1 P' P^{\alpha-1} + N_2 P^{-1} N_1 P^\alpha = 0$$

donc :

$$N_2 = -\alpha P'$$

ce qui détermine α.

Récapitulons

Si f est une dérivée, alors $\beta = -1$ est exclus et on peut appliquer (21) pour déterminer D. Si f n'est pas une dérivée, on calcule les facteurs de degré 1 de f_d :

$$f_1 = \frac{f_d}{\gcd(f_d, \partial_Z f_d)}$$

on décompose f par Bézout en isolant la partie N/f_1 les α possibles sont alors les racines entières (en t) du résultant en Z de $N - tf_1'$ et f_1, ils correspondent aux facteurs $\gcd(N - \alpha f_1', f_1)$ que l'on retire de f_d pour appliquer (21).

Exemple

Reprenons $y' + 2xy = a(x)$. Si $a(x) = 1$ (résolution de $\int \exp(x^2)$), ou plus généralement si $a(x)$ est un polynôme, alors $D = 1$. Si $a(x) = 1/x^2$, on trouve $D = x$ et on pose $y = xz$, donc $x^2(xz' + z) + 2x^4 z = 1$ soit $x^3 z' + (2x^4 + 1)z = 1$.

Reste le cas où Z est une exponentielle et $P = \exp(z)$. On reprend le même raisonnement, y' a pour valuation $-\alpha < 0$, fy a pour valuation $-\beta - \alpha$, donc si $\beta > 0$, $\alpha = \gamma$ et si $\beta < 0$, $\alpha = \gamma - \beta$. Si $\beta = 0$, s'il n'y a pas de simplifications du terme de plus bas degré, on est ramené au cas précédent. Si $\beta = 0$ et s'il y a simplification des termes de plus bas degré en Z, notons f_0 le coefficient constant de f par rapport à Z et y_α le coefficient de Z^α dans y, on a

$$y_\alpha' + (\alpha z' + f_0) y_\alpha = 0$$

donc :

$$y_\alpha = \exp(-\alpha z - \int f_0)$$

Comme y_α est élémentaire et indépendant de Z on en déduit par le théorème de structure de Risch que $-\alpha z - \int f_0$ est combinaison linéaire à coefficients rationnels des logarithmes et des arguments des exponentielles de la tour, de plus le coefficient de z doit être nul pour que y_α soit indépendant de Z, ce qui impose la valeur de α (après avoir résolu récursivement le problème d'intégration pour f_0)

Majoration du degré du numérateur de y

En multipliant y par $DZ^{-\alpha}$, puis en réduisant au même dénominateur, on se ramène alors à une équation différentielle à coefficients polynomiaux par rapport à la variable Z dont l'inconnue est un polynôme N :

$$RN' + SN = T \tag{22}$$

On va chercher une majoration sur le degré possible de N puis utiliser l'identité de Bézout pour simplifier cette équation.

On écrit maintenant $N = \sum_{k=0}^{n} N_k Z^k$ et on remplace, il y a à nouveau trois cas selon le type de Z.

Si $Z = x$: cas exponentielle rationnelle

Donc $Z' = 1$, le degré de RN' est $r + n - 1$ (si N est non constant c'est-à-dire si T n'est pas un multiple de S), le degré de SN est $s + n$. Si $r - 1 \neq s$, on en déduit que :

$$n = t - \max(r - 1, s)$$

Si $r - 1 = s$, on peut avoir une simplification du terme de plus haut degré $s + n$ (sinon on est dans le cas précédent) si $nR_r = S_s$ d'où on déduit le degré n de N.

Par exemple, pour $y' + 2xy = T$ ou pour $x^3 z' + (2x^4 + 1)z = 1$ on a $r = s - 1$ donc $n + s = t$, donc pas de solution dans le deuxième cas, dans le premier cas il ne peut y avoir de solutions que si $t \geq s$, en particulier il n'y a pas de solution pour $t = 1$, on a donc démontré que $\int \exp(x^2)$ n'admet pas de primitive élémentaire.

Si $Z = \exp(z)$: cas exponentielle d'exponentielle

Ici les N_k peuvent ne pas être constants, on a :

$$N' = \sum_{k=0}^{n} (N_k' + kN_k z') Z^k$$

Comme on l'a déjà observé, $N_n' + nN_n z' \neq 0$, donc le degré de N' est égal au degré de N. On a donc trois cas :

- si $r \neq s$, alors $n = t - \max(r, s)$
- si $r = s$ et les termes de plus haut degré du membre de gauche ne se simplifient pas, alors, $n = t - r = t - s$.
- si $r = s$ et s'il y a simplification, alors :

  $$R_r(N_n' + nN_n z') + S_s N_n = 0$$

  donc :

  $$N_n' + (\frac{S_s}{R_r} + nz')N_n = 0$$

  et :

  $$N_n = C \exp(-nz - \int \frac{S_s}{R_r})$$

  On appelle alors l'algorithme de Risch avec une variable de moins (S_s et R_r ne dépendent plus de Z) pour calculer $I = \int S_s/R_r$. Il s'agit alors de

trouver n tel que l'exponentielle précédente soit élémentaire et indépendante de la variable Z. Le théorème de structure de Risch implique que $-nz - \int S_s/R_r$ est combinaison linéaire à coefficients rationnels des logarithmes et des arguments des exponentielles de autres variables de la tour (jusqu'à z non compris). Ceci permet de déterminer n de manière unique (c'est le coefficient rationnel de $\int S_s/R_r$ en z).

Si $Z = \ln(z)$: exponentielle de logarithme

Ici aussi, les N_k peuvent ne pas être constants, on a :

$$N' = \sum_{k=0}^{n} (N_k' Z^k + k N_k \frac{z'}{z} Z^{k-1})$$

Si N_n n'est pas constant, le terme de plus haut degré de RN' est $N_n' R_r Z^{n+r}$, si N_n est constant, le terme de plus haut degré de RN' est $R_r(nN_n z'/z + N_{n-1}') Z^{r-1}$ qui est non nul (sinon $z'/z = CN_{n-1}'$ et $z = \exp(CN_{n-1})$ serait une exponentielle). Le terme de plus haut degré de SN est $N_n S_s Z^{n+s}$.

- Si $r < s$ ou si $r = s$ sans simplifications, alors $n = t - s$.
- Si $r > s + 1$ ou si $r = s + 1$ sans simplifications, alors $\deg(N') = t - r$ donc $n = t - r$ ou $n = t - r + 1$.
- Si $r = s + 1$, et s'il y a simplifications, alors N_n est constant et :

  $$R_r(nN_n z'/z + N_{n-1}') + S_s N_n = 0$$

  alors $N_{n-1} = C(-\int N_n S_s/R_r - nN_n \ln(z))$ doit être élémentaire et indépendante de Z donc $\int S_s/R_r$ est élémentaire, on détermine n en éliminant le coefficient de $Z = \ln(z)$ provenant de $\int S_s/R_r$.
- Si $r = s$, et s'il y a simplification des termes de plus haut degré du membre de gauche, alors $N_n' R_r + N_n S_s = 0$ donc $N_n = \exp(-\int S_s/R_r)$ est élémentaire et indépendante de Z. On peut donc changer d'inconnue $N = N_n M$ sans changer le fait que M est un polynôme de même degré que N. On se ramène alors à une équation du même type

  $$RM' + (S - R\frac{S_s}{R_r})M = \frac{T}{N_n}$$

  mais avec s diminué de 1 au moins.

Réduction (algorithme SPDE de Rothstein)

On observe d'abord que si R et S ont un facteur en commun, alors ce facteur divise T car N' et N sont des polynômes en Z. On peut donc quitte à simplifier par $\gcd(R, S)$ se ramener au cas où R et S sont premiers entre eux, il existe donc deux polynômes U et V tels que :

$$RU + SV = T, \quad \deg(V) < \deg(R) \tag{23}$$

En soustrayant (23) de (22), on montre que R divise $N - V$. Soit $H = (N - V)/R$. Alors $N = RH + V$ donc

$$R(RH' + R'H + V') + SRH + SV = T = RU + SV$$

donc après simplification par SV et division par R, H vérifie l'équation :

$$RH' + (S + R')H = U - V'$$

C'est une équation du même type mais avec deg(H)=deg(N)-deg(R) ou $H = 0$ (si $N = V$). Donc si deg(R) > 0, au bout d'un nombre fini d'étapes on doit tomber sur un second membre nul ou des simplifications de R avec $S + R'$ telles que R simplifié soit constant en Z.

Résolution

Si R est constant par rapport à Z, on simplifie par R et on doit résoudre

$$N' + SN = T$$

Si $S = 0$, c'est un problème d'intégration. Supposons donc que $S \neq 0$. Si S est non constant par rapport à Z ou si $Z = x$, le degré de N' est strictement inférieur au degré de SN, on peut donc facilement résoudre. Reste le cas où $S = b$ est constant non nul par rapport à Z et Z est une exponentielle ou un logarithme.

Si $Z = \exp(z)$

On a alors doit alors résoudre

$$N'_k + kN_k z' + bN_k = T_k$$

c'est une équation différentielle de Risch mais avec une variable de moins.

Si $Z = \ln(z)$

On doit alors résoudre

$$N'_k + (k+1)N_{k+1}\frac{z'}{z} + bN_k = T_k$$

c'est aussi une équation différentielle de Risch avec une variable de moins.

Exemple

Voyons comment on intègre x^n avec n un paramètre par l'algorithme de Risch (cela illustre les possibilités couvertes par l'algorithme mais aussi l'efficacité des méthodes traditionnelles d'intégration lorsqu'elles s'appliquent). On écrit d'abord $x^n = e^{n\ln(x)}$, donc la tour de variables est $\{x, Z = \ln(x), X = e^{n\ln(x)}\}$, il s'agit donc d'intégrer X qui est un polynôme généralisé. On cherche donc A_1 solution de l'équation différentielle de Risch

$$A'_1 + n/xA_1 = 1$$

Par rapport à $Z = \ln(x)$ la fonction $f = n/x$ est un polynôme, donc on applique le dernier cas ci-dessus, A_1 est aussi indépendant de $\ln(x)$ et on se ramène à résoudre la même équation mais avec comme variable principale x et non Z. Cette fois, il y a un dénominateur x en f. Si A_1 possède un dénominateur, il faut qu'il y ait annulation du terme de plus bas degré en x car le second membre n'a pas de dénominateur, on obtient $n + \alpha = 0$ qui n'a pas de solution, donc A_1 est un polynôme en x et l'équation se réécrit en :

$$xA'_1 + nA_1 = x$$

On majore alors le degré en x de A_1 par 1, car il ne peut pas y avoir d'annulation de terme de plus grand degré. Ensuite, on peut appliquer l'algorithme SPDE de Rothstein pour réduire le degré, ou ici conclure à la main, x divise nA_1 donc $A_1 = Cx$ qu'on remplace et $C = 1/(n+1)$. Finalement, $A_1 = x/(n+1)$ et $\int x^n = x/(n+1)x^n$.

9.4 Quelques références

- M. Bronstein :
  Symbolic Integration I, Transcendental functions, Springer
- M. Bronstein :
  Integration tutorial,
  `http://www-sop.inria.fr/cafe/Manuel.Bronstein/publications/mb_papers.`
- J.H. Davenport, Y. Siret, E. Tournier :
  Calcul formel : Systèmes et algorithmes de manipulations algébriques
- R. Risch :
  les références des articles originaux de Risch sont dans le "Integration tutorial" de Bronstein.
- B. Trager :
  PHD thesis MIT, 1984
- On peut lire en clair le code source de l'implémentation en MuPAD (sous Unix, désarchiver le fichier `lib.tar` du répertoire `/usr/local/MuPAD/share/lib` et regarder dans le sous-répertoire `lib/INTLIB`)

10 Intégration numérique

Les fractions rationnelles admettent une primitive que l'on calcule en décomposant la fraction avec Bézout comme expliqué précédemment. Mais elles font figure d'exceptions, la plupart des fonctions n'admettent pas de primitives qui s'expriment à l'aide des fonctions usuelles. Pour calculer une intégrale,on revient donc à la définition d'aire sous la courbe, aire que l'on approche, en utilisant par exemple un polynome de Lagrange.

Le principe est donc le suivant : on découpe l'intervalle d'intégration en subdivisions $[a,b] = [a,a+h] + [a+h,a+2h] + ...[a+(n-1)h, a+nh = b$, où $h = (b-a)/n$ est le pas de la subdivision, et sur chaque subdivision, on approche l'aire sous la courbe.

10.1 Les rectangles et les trapèzes

Sur une subdivision $[\alpha,\beta]$, on approche la fonction par un segment. Pour les rectangles, il s'agit d'une horizontale : on peut prendre $f(\alpha)$, $f(\beta)$ (rectangle à droite et gauche) ou $f((\alpha+\beta)/2)$ (point milieu), pour les trapèzes on utilise le segment reliant $[\alpha, f(\alpha)]$ à $[\beta, f(\beta)]$.

Exemple : calcul de la valeur approchée de $\int_0^1 t^3 dt$ (on en connait la valeur exacte $1/4 = 0.25$) par ces méthodes en subdivisant $[0,1]$ en 10 subdivisions (pas $h = 1/10$), donc $\alpha = j/10$ et $\beta = (j+1)/10$ pour j variant de 0 à 9. Pour les rectangles à gauche, on obtient sur une subdivision $f(\alpha) = (j/10)^3$ que l'on multiplie par la longueur de la subdivision soit $h = 1/10$:

$$\frac{1}{10}\sum_{j=0}^{9}(\frac{j}{10})^3 = \frac{81}{400} = 0.2025$$

Pour les rectangles à droite, on obtient

$$\frac{1}{10}\sum_{j=1}^{10}(\frac{j}{10})^3 = \frac{121}{400} = 0.3025$$

Pour le point milieu $f((\alpha+\beta)/2) = f((j/10+(j+1)/10)/2) = f(j/10+1/20)$

$$\frac{1}{10}\sum_{j=0}^{9}(\frac{j}{10}+\frac{1}{20})^3 = 199/800 = 0.24875$$

Enfin pour les trapèzes, l'aire du trapèze délimité par l'axe des x, les verticales $y=\alpha$, $y=\beta$ et les points sur ces verticales d'ordonnées respectives $f(\alpha)$ et $f(\beta)$ vaut

$$h\frac{f(\alpha)+f(\beta)}{2}$$

donc

$$\frac{1}{10}\sum_{j=0}^{9}\left((\frac{j}{10})^3+(\frac{j+1}{10})^3\right) = \frac{101}{400} = 0.2525$$

Dans la somme des trapèzes, on voit que chaque terme apparait deux fois sauf le premier et le dernier.

Plus généralement, les formules sont donc les suivantes :

$$\text{rectangle gauche} = h\sum_{j=0}^{n-1} f(a+jh) \tag{24}$$

$$\text{rectangle droit} = h\sum_{j=1}^{n} f(a+jh) \tag{25}$$

$$\text{point milieu} = h\sum_{j=0}^{n-1} f(a+jh+\frac{h}{2}) \tag{26}$$

$$\text{trapezes} = h\left(\frac{f(a)+f(b)}{2}+\sum_{j=1}^{n-1} f(a+jh)\right) \tag{27}$$

où $h=(b-a)/n$ est le pas de la subdivision, n le nombre de subdivisions.

On observe sur l'exemple que le point milieu et les trapèzes donnent une bien meilleure précision que les rectangles. Plus généralement, la précision de l'approximation n'est pas la même selon le choix de méthode. Ainsi pour les rectangles à gauche (le résultat est le même à droite), si f est continument dérivable, de dérivée majorée par une constante M_1 sur $[a,b]$, en faisant un développement de Taylor de f en α, on obtient

$$|\int_\alpha^\beta f(t)dt - \int_\alpha^\beta f(\alpha)dt| = |\int_\alpha^\beta f'(\theta_t)(t-\alpha)dt| \leq M_1\int_\alpha^\beta (t-\alpha)dt = M_1\frac{(\beta-\alpha)^2}{2}$$

Ainsi dans l'exemple, on a $M_1=3$, l'erreur est donc majorée par 0.015 sur une subdivision, donc par 0.15 sur les 10 subdivisions.

Pour le point milieu, on fait le développement en $(\alpha+\beta)/2$ à l'ordre 2, en supposant que f est deux fois continument dérivable :

$$\begin{aligned}
|\int_\alpha^\beta f(t) - \int_\alpha^\beta f(\frac{\alpha+\beta}{2})| &= |\int_\alpha^\beta f'(\frac{\alpha+\beta}{2})(t-\frac{\alpha+\beta}{2})dt \\
&\quad + \int_\alpha^\beta \frac{f''(\theta_t)}{2}(t-\frac{\alpha+\beta}{2})^2| \\
&\leq \frac{M_2}{2} 2 \int_{\frac{\alpha+\beta}{2}}^\beta (t-\frac{\alpha+\beta}{2})^2 dt \\
&\leq M_2 \frac{(\beta-\alpha)^3}{24}
\end{aligned}$$

Dans l'exemple, on a $M_2 = 6$, donc l'erreur sur une subdivision est majorée par $0.25e-3$, donc sur 10 subdivisions par $0.25e-2 = 0.0025$.

Pour les trapèzes, la fonction g dont le graphe est le segment reliant $[\alpha, f(\alpha)]$ à $[\beta, f(\beta)]$ est $f(\alpha)+(t-\alpha)/(\beta-\alpha)f(\beta)$, c'est en fait un polynome de Lagrange, si f est deux fois continument dérivable, on peut donc majorer la différence entre f et g en utilisant (41), on intègre la valeur absolue ce qui donne

$$|\int_\alpha^\beta f(t)dt - \int_\alpha^\beta g(t)dt| \leq \int_\alpha^\beta |\frac{f''(\xi_x)}{2}(x-\alpha)(x-\beta)| \leq M_2 \frac{(\beta-\alpha)^3}{12}$$

où M_2 est un majorant de $|f''|$ sur $[a,b]$.

Lorsqu'on calcule l'intégrale sur $[a,b]$ par une de ces méthodes, on fait la somme sur $n = (b-a)/h$ subdivisions de longueur $\beta - \alpha = h$, on obtient donc une majoration de l'erreur commise sur l'intégrale :

- pour les rectangles à droite ou gauche $nM_1h^2/2 = M_1h(b-a)/2$
- pour le point milieu $M_2h^2(b-a)/24$
- pour les trapèzes $M_2h^2(b-a)/12$.

Lorsque h tend vers 0, l'erreur tend vers 0, mais pas à la même vitesse, plus rapidement pour les trapèzes et le point milieu que pour les rectangles. Plus on approche précisément la fonction sur une subdivision, plus la puissance de h va être grande, plus la convergence sera rapide lorsque h sera petit, avec toutefois une contrainte fixée par la valeur de M_k, borne sur la dérivée k-ième de f (plus k est grand, plus M_k est grand en général). Nous allons voir dans la suite comment se comporte cette puissance de h en fonction de la facon dont on approche f.

10.2 Ordre d'une méthode

On appelle méthode d'intégration l'écriture d'une approximation de l'intégrale sur une subdivision sous la forme

$$\int_\alpha^\beta f(t)dt \approx I(f) = \sum_{j=1}^k w_j f(y_j)$$

où les y_j sont dans l'intervalle $[\alpha,\beta]$, par exemple équirépartis sur $[\alpha,\beta]$. On utilise aussi la définition :

$$\int_\alpha^\beta f(t)dt \approx I(f) = (\beta-\alpha)\sum_{j=1}^k \tilde{w}_j f(y_j)$$

On prend toujours $\sum_j w_j = \beta - \alpha$ (ou $\sum_j \tilde{w}_j = 1$) pour que la méthode donne le résultat exact si la fonction est constante.

On dit qu'une méthode d'intégration est d'ordre n si il y a égalité ci-dessus pour tous les polynômes de degré inférieur ou égal à n et non égalité pour un polynôme de degré $n+1$. Par exemple, les rectangles à droite et gauche sont d'ordre 0, le point milieu et les trapèzes sont d'ordre 1. Plus généralement, si on approche f par son polynôme d'interpolation de Lagrange en $n+1$ points (donc par un polynôme de degré inférieur ou égal à n), on obtient une méthode d'intégration d'ordre au moins n.

Si une méthode est d'ordre n avec des $w_j \geq 0$ et si f est $n+1$ fois continument dérivable, alors sur une subdivision, on a :

$$|\int_\alpha^\beta f - I(f)| \leq M_{n+1}\frac{(\beta-\alpha)^{n+2}}{(n+1)!}(\frac{1}{n+2}+1) \tag{28}$$

En effet, on fait le développement de Taylor de f par exemple en α à l'ordre n

$$\begin{aligned} f(t) &= T_n(f) + \frac{(t-\alpha)^{n+1}}{(n+1)!}f^{[n+1]}(\theta_t), \\ T_n(f) &= f(\alpha) + (t-\alpha)f'(\alpha) + ... + \frac{(t-\alpha)^n}{n!}f^{[n]}(\alpha) \end{aligned}$$

Donc

$$|\int_\alpha^\beta f - \int_\alpha^\beta T_n(f)| \leq \int_\alpha^\beta \frac{(t-\alpha)^{n+1}}{(n+1)!}|f^{[n+1]}(\theta_t)| \leq \left[M_{n+1}\frac{(t-\alpha)^{n+2}}{(n+2)!}\right]_\alpha^\beta$$

De plus,

$$\begin{aligned} |I(f) - I(T_n(f))| = |I\left(f^{[n+1]}(\theta_t)\frac{(t-\alpha)^{n+1}}{(n+1)!}\right)| &\leq \sum_{j=1}^k |w_j| M_{n+1}\frac{(y_j-\alpha)^{n+1}}{(n+1)!} \\ &\leq \sum_{j=1}^k |w_j| M_{n+1}\frac{(\beta-\alpha)^{n+1}}{(n+1)!} \end{aligned}$$

Donc comme la méthode est exacte pour $T_n(f)$, on en déduit que

$$\begin{aligned} |\int_\alpha^\beta f - I(f)| &= |\int_\alpha^\beta f - \int_\alpha^\beta T_n(f) + I(T_n(f)) - I(f)| \\ &\leq |\int_\alpha^\beta f - \int_\alpha^\beta T_n(f)| + |I(T_n(f)) - I(f)| \\ &\leq M_{n+1}\frac{(\beta-\alpha)^{n+2}}{(n+2)!} + \sum_{j=1}^k |w_j| M_{n+1}\frac{(\beta-\alpha)^{n+1}}{(n+1)!} \end{aligned}$$

Si les $w_j \geq 0$, alors $\sum_{j=1}^k |w_j| = \sum_{j=1}^k w_j = \beta - \alpha$ et on obtient finalement (28)

On remarque qu'on peut améliorer la valeur de la constante en faisant tous les développement de Taylor en $(\alpha+\beta)/2$ au lieu de α, Après sommation sur les n subdivisions, on obtient que :

Théorème 9 *Pour une méthode d'ordre n à coefficients positifs et une fonction f $n+1$ fois continument dérivable*

$$|\int_a^b f - I(f)| \leq M_{n+1}\frac{h^{n+1}}{2^{n+1}(n+1)!}(b-a)(\frac{1}{(n+2)}+1)$$

On observe que cette majoration a la bonne puissance de h sur les exemples déja traités, mais pas forcément le meilleur coefficient possible, parce que nous avons traité le cas général d'une méthode d'ordre n.

10.3 Simpson

Il s'agit de la méthode obtenue en approchant la fonction sur la subdivision $[\alpha, \beta]$ par son polynome de Lagrange aux points $\alpha, (\alpha+\beta)/2, \beta$. On calcule l'intégrale par exemple avec un logiciel de calcul formel, avec Xcas :

```
factor(int(lagrange([a,(a+b)/2,b],[fa,fm,fb]),x=a..b))
```

qui donne la formule sur une subdivision

$$I(f) = \frac{h}{6}(f(\alpha) + 4f(\frac{\alpha+\beta}{2}) + f(\beta))$$

et sur $[a, b]$:

$$I(f) = \frac{h}{6}\left(f(a) + f(b) + 4\sum_{j=0}^{n-1} f(a+jh+\frac{h}{2}) + 2\sum_{j=1}^{n-1} f(a+jh)\right) \quad (29)$$

Si on intègre t^3 sur $[0, 1]$ en 1 subdivision par cette méthode, on obtient

$$\frac{1}{6}(0 + 4\frac{1}{2^3} + 1) = \frac{1}{4}$$

c'est-à-dire le résultat exact, ceci est aussi vérifié pour f polynome de degré inférieur ou égal à 2 puisque l'approximation de Lagrange de f est alors égale à f. On en déduit que la méthode de Simpson est d'ordre 3 (pas plus car la méthode de Simpson appliquée à l'intégrale de t^4 sur $[0, 1]$ n'est pas exacte). On peut même améliorer (cf. par exemple Demailly) la constante générale de la section précédente pour la majoration de l'erreur en :

$$|\int_a^b f - I(f)| \leq \frac{h^4}{2880}(b-a)M_4$$

Cette méthode nécessite $2n+1$ évaluations de f (le calcul de f est un point étant presque toujours l'opération la plus couteuse en temps d'une méthode de quadrature), au lieu de n pour les rectangles et le point milieu et $n+1$ pour les trapèzes. Mais on a une majoration en h^4 au lieu de h^2 donc le "rapport qualité-prix" de la méthode de Simpson est meilleur, on l'utilise donc plutot que les méthodes précédentes sauf si f n'a pas la régularité suffisante (ou si M_4 est trop grand).

10.4 Newton-Cotes

On peut généraliser l'idée précédente, découper la subdivision $[\alpha, \beta]$ en n parts égales et utiliser le polynôme d'interpolation en ces $n+1$ points $x_0 = \alpha, x_1, ..., x_n = \beta$. Ce sont les méthodes de Newton-Cotes, qui sont d'ordre n au moins. Comme le polynôme d'interpolation dépend linéairement des ordonnées, cette méthode est bien de la forme :

$$I(f) = (\beta - \alpha) \sum_{j=0}^{n} \tilde{w}_j f(x_j)$$

De plus les $\tilde{w}_j$ sont universels (ils ne dépendent pas de la subdivision), parce qu'on peut faire le changement de variables $x = \alpha + t(\beta - \alpha)$ dans l'intégrale et le polynôme d'interpolation et donc se ramener à $[0, 1]$.

Exemple : on prend le polynôme d'interpolation en 5 points équidistribués sur une subdivision $[a, b]$ (méthode de Boole). Pour calculer les $\tilde{w}_j$, on se ramène à $[0, 1]$, puis on tape

```
int(lagrange(seq(j/4,j,0,4),[f0,f1,f2,f3,f4]),x=0..1)
```

et on lit les coefficients de `f0` à `f4` qui sont les $\tilde{w}_0$ à $\tilde{w}_4$: 7/90, 32/90, 12/90, 32/90, 7/90. La méthode est d'ordre au moins 4 par construction, mais on vérifie qu'elle est en fait d'ordre 5 (exercice), la majoration de l'erreur d'une méthode d'ordre 5 est

$$|\int_a^b f - I(f)| \leq \frac{M_6}{2^6 6!}(1 + \frac{1}{7})h^6(b-a)$$

elle peut être améliorée pour cette méthode précise en

$$|\int_a^b f - I(f)| \leq \frac{M_6}{1935360}h^6(b-a)$$

En pratique, on ne les utilise pas très souvent, car d'une part pour $n \geq 8$, les w_j ne sont pas tous positifs, et d'autre part, parce que la constante M_n devient trop grande. On préfère utiliser la méthode de Simpson en utilisant un pas plus petit.

Il existe aussi d'autres méthodes, par exemple les quadratures de Gauss (on choisit d'interpoler en utilisant des points non équirépartis tels que l'ordre de la méthode soit le plus grand possible) ou la méthode de Romberg qui est une méthode d'accélération de convergence basée sur la méthode des trapèzes (on prend la méthode des trapèzes en 1 subdivision de $[a, b]$, puis 2, puis 2^2, ..., et on élimine les puissances de h du reste $\int f - I(f)$ en utilisant un théorème d'Euler-Mac Laurin qui montre que le développement asymptotique de l'erreur en fonction de h ne contient que des puissances paires de h). De plus, on peut être amené à faire varier le pas h en fonction de la plus ou moins grande régularité de la fonction.

10.5 En résumé

Intégration sur $[a, b]$, h pas d'une subdivision, M_k majorant de la dérivée k-ième de la fonction sur $[a, b]$

	formule	Lagrange degré	ordre	erreur
rectangles	(24), (25)	0	0	$M_1 h(b-a)/2$
point milieu	(26)	0	1	$M_2 h^2(b-a)/24$
trapèzes	(27)	1	1	$M_2 h^2(b-a)/12$
Simpson	(29)	2	3	$M_4 h^4(b-a)/2880$

11 Algèbre linéaire

On présente ici des algorithmes autour de la résolution exacte de systèmes (réduction des matrices sous forme échelonnée) et la recherche de valeurs propres et de vecteurs propres (diagonalisation et jordanisation des matrices).

11.1 Résolution de systèmes, calcul de déterminant.

11.1.1 La méthode du pivot de Gauß.

- Le pivot : on détermine à partir d'une ligne i la ligne j où apparait le premier coefficient non nul p dans la colonne à réduire. On échange les lignes i et j. Puis pour $j > i$ (réduction sous-diagonale) ou $j \neq i$ (réduction complète), on effectue l'opération $L_j \leftarrow L_j - \frac{p_j}{p} L_i$.
  Inconvénient : avec des données exactes de taille non bornée, la complexité des coefficients augmente plus vite qu'en choisissant le pivot le plus simple possible, (remarque, lorsque les données sont approchées, on n'utilise pas non plus cette méthode pour des raisons de stabilité numérique). Le domaine d'utilisation naturel concerne donc les coefficients dans un corps fini (par exemple $\mathbb{Z}/n\mathbb{Z}$).
- Le pivot partiel. On choisit le meilleur coefficient non nul de la colonne, où meilleur dépend du type de coefficient : avec des données exactes, on choisirait le coefficient de taille la plus petite possible, avec des données approximatives, on choisit le coefficient de plus grande norme dans la colonne. Le domaine d'utilisation naturel concerne les coefficients approchés. Pour les coefficients exacts, on remplacerait la réduction par $L_j \leftarrow pL_j - p_j L_i$ pour ne pas effectuer de division. Mais avec cette méthode, la taille des coefficients augmente de manière exponentielle. On peut améliorer la taille des coefficients intermédiaires en divisant chaque ligne par le PGCD de ses coefficients, mais comme pour le calcul du PGCD par l'algorithme du sous-résultant, il existe une méthode plus efficace présentée ci-dessous.
- La méthode de Bareiss : on initialise un coefficient b à 1. On remplace l'étape de réduction ci-dessus par $L_j \leftarrow (pL_j - p_j L_i)/b$. À la fin de l'étape de réduction, on met le coefficient b à la valeur du pivot p. L'intérêt de la méthode est que la division se fait sans introduire de fraction (la preuve pour les deux premières étapes se fait facilement à la main ou avec un système de calcul formel (cf. infra), pour le cas général, on vérifie que le déterminant de la matrice de départ est égal au dernier coefficient sur la diagonale obtenu par cette méthode de réduction, ce dernier est donc entier, le même raisonnement fait sur des sous-matrices dont on prend les k premières lignes et colonnes et une autre ligne et une autre colonne montre que tous les coefficients des matrices intermédiaires sont entiers). On peut utiliser cette méthode aussi bien pour la réduction sous-diagonale que pour la réduction complète (les lignes intervenant dans la combinaison linéaire subissent des modifications identiques dans les deux cas).

Montrons avec MuPAD ou xcas en mode mupad (commande `maple_mode(2)`) qu'en effet, on n'introduit pas de dénominateur dans la méthode de Bareiss. Sans restreindre la généralité, il suffit de le montrer avec une matrice 3x3 à coefficients symboliques génériques.

```
pivot:=proc (M,n,m,r) // n ligne du pivot, m colonne, r ligne a modifier
      local col,i,a,b;
       begin
         col:=ncols(M);
         a:=M[n,m];
         b:=M[r,m];
         for i from 1 to col do
           // print(i,a,b,n,m,r);
           M[r,i]:=a*M[r,i]-b*M[n,i];
         end_for;
         return(M);
       end_proc; /* End of pivot */
A:=matrix(3,3,[[a,b,c],[d,e,f],[g,h,j]]);
A:=pivot(A,1,1,2); A:=pivot(A,1,1,3); /* reduction 1ere colonne */
A:=pivot(A,2,2,3); A:=pivot(A,2,2,1); /* reduction 2eme colonne */
factor(A[3,3]);
```

Ce qui met bien en évidence le facteur a dans $A_{3,3}$.

11.1.2 Le déterminant.

On peut bien sûr appliquer les méthodes ci-dessus en tenant compte des pivots utilisés et du produit des coefficients diagonaux. Dans le cas de la méthode de Bareiss, si on effectue la réduction sous-diagonale uniquement, il n'est pas nécessaire de garder une trace des pivots et de calculer le produit des coefficients diagonaux, montrons que la valeur du déterminant est égal au dernier coefficient diagonal : en effet si R désigne la matrice réduite et que l'on pose $R_{0,0} = 1$, alors la réduction par la méthode de Bareiss de la colonne i a pour effet de multiplier le déterminant de la matrice initiale M par $(R_{i,i}/(R_{i-1,i-1})^{n-i}$. Donc :

$$\begin{aligned} \det(R) &= \det(M) \prod_{i=1}^{n-1} (R_{i,i}/(R_{i-1,i-1})^{n-i} \\ \prod_{i=1}^{n} R_{i,i} &= \det(M) \prod_{i=1}^{n-1} R_{i,i} \\ R_{n,n} &= \det(M) \end{aligned}$$

Pour les matrices à coefficients entiers, on peut aussi utiliser une méthode modulaire : on calcule une borne à priori sur le déterminant et on calcule le déterminant modulo suffisamment de petits nombres premiers pour le reconstruire par les restes chinois. L'avantage de cet algorithme est qu'il est facile à paralléliser.

On utilise souvent la borne d'Hadamard sur le déterminant :

$$|\det(M)| \leq \prod_{1 \leq i \leq n} \sqrt{\sum_{1 \leq j \leq n} |m_{i,j}|^2}$$

Preuve de la borne : on majore le déterminant par le produit des normes des vecteurs colonnes de M.

Remarque :
Si on veut juste prouver l'inversibilité d'une matrice à coefficients entiers, il suffit

de trouver un nombre premier p tel que le déterminant de cette matrice modulo p soit non nul.

Développement par rapport à une ligne ou une colonne
On a tendance à oublier ce type de méthode car le développement complet du déterminant (faisant intervenir une somme sur toutes les permutations du groupe symétrique) nécessite d'effectuer $n!$ produits de n coefficients et $n!$ additions ce qui est gigantesque. Or on peut "factoriser" une partie des calculs et se ramener à $n.2^n$ opérations élémentaires au lieu de $n.n!$. Remarquons aussi que le nombre d'opérations élémentaires n'a guère de sens si on ne tient pas compte de la complexité des expressions, l'avantage principal de la méthode de développement étant d'éviter d'effectuer des divisions.

Calcul du déterminant par développement de Laplace
On calcule d'abord tous les mineurs 2x2 des colonnes 1 et 2 que l'on place dans une table de mineurs, puis on calcule les mineurs 3x3 des colonnes 1 à 3 en développant par rapport à la colonne 3 et en utilisant les mineurs précédents, puis les mineurs 4x4 avec les mineurs 3x3, etc.. On évite ainsi de recalculer plusieurs fois les mêmes mineurs. Cf. par exemple l'implémentation en C++ dans giac/xcas (`www-fourier.ujf-grenoble.fr/~parisse/giac.html`) qui utilise le type générique `map<>` de la librairie standard C++ (STL) pour stocker les tables de mineurs (fonction `det_minor` du fichier `vecteur.cc`).
Nombre d'opérations élémentaires : il y a $\binom{n}{2}$ mineurs d'ordre 2 à calculer nécessitant chacun 2 multiplications (et 1 addition), puis $\binom{n}{3}$ mineurs d'ordre 3 nécessitant 3 multiplications et 2 additions, etc. donc le nombre de multiplications est de $2\binom{n}{2}+3\binom{n}{3}+...+n\binom{n}{n}$, celui d'additions est $\binom{n}{2}+2\binom{n}{3}+...+(n-1)\binom{n}{n}$ soit un nombre d'opérations élémentaires majoré par $n.2^n$.

On observe "expérimentalement" que cet algorithme est intéressant lorsque le nombre de paramètres dans le déterminant est grand et que la matrice est plutôt creuse (majorité de coefficients nuls). Il existe des heuristiques de permutation des lignes ou des colonnes visant à optimiser la position des zéros (par exemple, les auteurs de GiNaC (`www.ginac.de`) suite à des expérimentations privilégient la simplification des petits mineurs en mettant les colonnes contenant le maximum de zéros à gauche selon la description faite ici).

Pour se convaincre de l'intérêt de cet algorithme, on peut effectuer le test O1 de Lewis-Wester
`http://www.bway.net/~lewis/calatex.html`
il s'agit de calculer un déterminant de taille 15 avec 18 paramètres.

11.1.3 Systèmes linéaires

On peut appliquer la méthode du pivot de Gauß ou les règles de Cramer. Pour les systèmes à coefficients entiers non singuliers, on peut aussi utiliser une méthode p-adique asymptotiquement plus efficace. On calcule d'abord une borne sur les coefficients des fractions solutions de l'équation $Ax = b$ en utilisant les règles de Cramer et la borne d'Hadamard. On calcule ensuite C, l'inverse de A modulo p (en changeant de p si A n'est pas inversible modulo p), puis, si

$$x = \sum_i x_i p^i, \quad A(\sum_{i<k} x_i p^i) = b \pmod{p^k}$$

on ajoute $x_k p^k$ et on obtient l'équation :

$$Ax_k = \frac{b - \sum_{i<k} x_i p^i}{p^k} \pmod{p}$$

qui détermine x_k. On s'arrête lorsque k est suffisamment grand pour pouvoir reconstruire les fractions à l'aide de l'identité de Bézout (cf. l'appendice), ce qui est le cas si p^k est supérieur à 4 fois la borne de Hadamard de A au carré. Pour éviter de recalculer plusieurs fois $b - \sum_{i<k} x_i p^i$, on utilise la récurrence suivante

$$y_0 = b, \quad x_k = Cy_k \pmod{p}, \quad y_{k+1} = \frac{y_k - Ax_k}{p}$$

Pour une matrice de taille n, il faut $O(n^3)$ opérations pour calculer C, puis $kn^2 \ln(n)$ opérations pour calculer x_k (le terme $\ln(n)$ vient de la taille des coefficients de y_k dans le produit Cy_k), donc pour pouvoir reconstruire x, il faut prendre k de l'ordre de $n \ln(n)$, ce qui nécessite finalement $O(n^3 \ln(n)^2)$ opérations.

Application au calcul de déterminant de matrices à coefficient entiers
Cette méthode p-adique peut servir à accélérer le calcul du déterminant d'une matrice à coefficients entiers de grande taille. En effet, le PPCM f des dénominateurs des composantes de x est un diviseur du déterminant, et si b est choisi avec des coefficients aléatoires, on a une forte probabilité d'obtenir le dernier facteur invariant de la matrice A. Comme le déterminant de A a une très faible probabilité de contenir un gros facteur carré, ce dernier facteur invariant est très proche du déterminant. Ce dernier est pour une matrice A aléatoire lui-même à un facteur de l'ordre de $(2/\pi)^n$ proche de la borne de Hadamard. Il suffit donc de très peu de nombres premiers pour déterminer $\det(A)/f$ par le théorème des restes chinois. En pratique pour des n de l'ordre de 100 à 1000, cet algorithme est plus rapide que le calcul uniquement par les restes chinois. Pour des n plus grands, il faut se rabattre sur des algorithmes probabilistes avec arrêt prématuré pour être plus rapide (on s'arrête lorsque le déterminant n'évolue plus par reconstruction par les restes chinois pour plusieurs nombres premiers successifs), et également utiliser des méthodes d'inversion ou de réduction de type Strassen.

11.1.4 Bézout et les p-adiques.

Soit n et a/b une fraction irréductible d'entiers tels que b est premier avec n et $|a| < \sqrt{n}/2$ et $0 \leq b \leq \sqrt{n}/2$. Il s'agit de reconstruire a et b connaissant $x = a \times (b^{-1}) \pmod{n}$ avec $x \in [0, n[$.

Unicité
S'il existe une solution (a, b) vérifiant $|a| < \sqrt{n}/2$ et $0 \leq b \leq \sqrt{n}/2$, soit (a', b') une solution de $x = a \times (b^{-1}) \pmod{n}$ et vérifiant $|a'| < \sqrt{n}$ et $0 \leq b' \leq \sqrt{n}$, alors :

$$ab' = a'b \pmod{n}$$

Comme $|ab'| < n/2$, $|a'b| < n/2$, on en déduit que $ab' = a'b$. Donc $a/b = a'/b'$ donc $a = a'$ et $b = b'$ car a/b et a'/b' sont supposées irréductibles.

Reconstruction lorsqu'on sait qu'il y a une solution
On suit l'algorithme de calcul des coefficients de Bézout pour les entiers n et x. On pose :

$$\alpha_k n + \beta_k x = r_k$$

où les r_k sont les restes successifs de l'algorithme d'Euclide, avec la condition initiale :

$$\alpha_0 = 1, \beta_0 = 0, \alpha_1 = 0, \beta_1 = 1, r_0 = n, r_1 = x$$

et la relation de récurrence :

$$\beta_{k+2} = \beta_k - q_{k+2}\beta_{k+1}, \quad q_{k+2} = \frac{r_k - r_{k+2}}{r_{k+1}}$$

On a $\beta_k x = r_k \pmod{n}$ pour tout rang mais il faut vérifier les conditions de taille sur β_k et r_k pour trouver le couple (a, b). Montrons par récurrence que :

$$\beta_{k+1}r_k - r_{k+1}\beta_k = (-1)^k n \tag{30}$$

Au rang $k = 0$, on vérifie l'égalité, on l'admet au rang k, alors au rang $k + 1$, on a :

$$\begin{aligned}
\beta_{k+2}r_{k+1} - r_{k+2}\beta_{k+1} &= \beta_k r_{k+1} - q_{k+2}r_{k+1}\beta_{k+1} - r_{k+2}\beta_{k+1} \\
&= \beta_k r_{k+1} - (r_k - r_{k+2})\beta_{k+1} - r_{k+2}\beta_{k+1} \\
&= \beta_k r_{k+1} - r_k\beta_{k+1} \\
&= -(-1)^k n
\end{aligned}$$

On vérifie aussi que le signe de β_k est positif si k est impair et négatif si k est pair, on déduit donc de (30) :

$$|\beta_{k+1}|r_k < n$$

(avec égalité si $r_{k+1} = 0$)

Considérons la taille des restes successifs, il existe un rang k tel que $r_k \geq \sqrt{n}$ et $r_{k+1} < \sqrt{n}$. On a alors $|\beta_{k+1}| < n/r_k \leq \sqrt{n}$.

Donc l'algorithme de Bézout permet de reconstruire l'unique couple solution s'il existe.

Exemple

On prend $n = 101$, $a = 2$, $b = 3$, $a/b = 68 \pmod{101}$. Puis on effectue Bézout pour 68 et 101 en affichant les étapes intermédiaires (par exemple avec `IEGCD` sur une HP49 ou exercice avec votre système de calcul formel) :

```
   = alpha*101+beta*68
101    1        0
 68    0        1  L1 - 1*L2
 33    1       -1  L2 - 2*L3
  2   -2        3  ...
```

On s'arrête à la première ligne telle que le coefficient de la 1ère colonne est inférieur à $\sqrt{101}$, on retrouve bien 2 et 3. Quand on programme l'algorithme de reconstruction, on ne calcule bien sûr pas la colonne des α, ce qui donne par exemple le programme xcas ou mupad suivant :

```
// Renvoie a/b tel que a/b=x mod n et |a|,|b|<sqrt(n)
padictofrac:=proc (n,x)
  local r0,beta0,r1,beta1,r2,q2,beta2;
begin
  r0:=n;
```

```
  beta0:=0;
  r1:=x;
  beta1:=1;
  sqrtn:=float(sqrt(n));
  while r1>sqrtn do
    r2:= irem(r0,r1);
    q2:=(r0-r2)/r1;
    beta2:=beta0-q2*beta1;
    beta0:=beta1; r0:=r1; beta1:=beta2; r1:=r2;
  end_while;
  return(r1/beta1);
end_proc;
```

11.1.5 Base du noyau

On présente ici deux méthodes, la première se généralise au cas des systèmes à coefficients entiers, la deuxième utilise un peu moins de mémoire (elle travaille sur une matrice 2 fois plus petite).

Première méthode Soir M la matrice dont on cherche le noyau. On ajoute à droite de la matrice transposée de M une matrice identité ayant le même nombre de lignes que M^t. On effectue une réduction sous-diagonale qui nous amène à une matrice composée de deux blocs

$$(M^t I_n) \ \rightarrow \ (U\tilde{L})$$

Attention, $\tilde{L}$ n'est pas la matrice L de la décomposition LU de M^t, on a en fait

$$\tilde{L}M^t = U$$

donc

$$M\tilde{L}^t = U^t$$

Les colonnes de $\tilde{L}^t$ correspondant aux colonnes nulles de U^t (ou si on préfère les lignes de $\tilde{L}$ correspondant aux lignes nulles de U) sont donc dans le noyau de M et réciproquement si $Mv = 0$ alors

$$U^t(\tilde{L}^t)^{-1}v = 0$$

donc, comme U est réduite, $(\tilde{L}^t)^{-1}v$ est une combinaison linéaire des vecteurs de base d'indice les lignes nulles de U. Finalement, les lignes de $\tilde{L}$ correspondant aux lignes nulles de U forment une base du noyau de M.

On peut faire le raisonnement ci-dessus à l'identique si M est une matrice à coefficients entiers, en effectuant des manipulations élémentaires réversibles dans $\mathbb{Z}$, grâce à l'idendité de Bézout. Si a est le pivot en ligne i, b le coefficient en ligne j à annuler, et u, v, d les coefficients de l'identité de Bézout $au + bv = d$ on fait les changements :

$$L_i \leftarrow uL_i + vL_j, \quad L_j \leftarrow -\frac{b}{d}L_i + \frac{a}{d}L_j$$

qui est réversible dans $\mathbb{Z}$ car le déterminant de la sous-matrice élémentaire correspondante est

$$\begin{vmatrix} u & v \\ -\frac{b}{d} & \frac{a}{d} \end{vmatrix} = 1$$

Cette réduction (dite de Hermite) permet de trouver une base du noyau à coefficients entiers et telle que tout élément du noyau à coefficient entier s'écrit comme combinaison linéaire à coefficients entiers des éléments de la base.

Deuxième méthode On commence bien sûr par réduire la matrice (réduction complète en-dehors de la diagonale), et on divise chaque ligne par son premier coefficient non nul (appelé pivot). On insère alors des lignes de 0 pour que les pivots (non nuls) se trouvent sur la diagonale. Puis en fin de matrice, on ajoute ou on supprime des lignes de 0 pour avoir une matrice carrée de dimension le nombre de colonnes de la matrice de départ. On parcourt alors la matrice en diagonale. Si le i-ième coefficient est non nul, on passe au suivant. S'il est nul, alors tous les coefficients d'indice supérieur ou égal à i du i-ième vecteur colonne v_i sont nuls (mais pas forcément pour les indices inférieurs à i). Si on remplace le i-ième coefficient de v_i par -1, il est facile de se convaincre que c'est un vecteur du noyau, on le rajoute donc à la base du noyau. On voit facilement que tous les vecteurs de ce type forment une famille libre de la bonne taille, c'est donc bien une base du noyau.

11.2 Réduction des endomorphismes

11.2.1 Le polynôme minimal

On prend un vecteur v au hasard et on calcule la relation linéaire de degré minimal entre v, Av, ..., $A^n v$ en cherchant le premier vecteur w du noyau de la matrice obtenue en écrivant les vecteurs v, Av, etc. en colonne dans cet ordre. Les coordonnées de w donnent alors par ordre de degré croissant un polynôme P de degré minimal tel que $P(A)v = 0$ donc P divise le polynôme minimal M. Donc si P est de degré n, $P = M$. Sinon, il faut vérifier que le polynôme obtenu annule la matrice A. On peut aussi calculer en parallèle le polynôme P précédent pour quelques vecteurs aléatoires et prendre le PPCM des polynômes obtenus.

Exemple 1

Polynôme minimal de $\begin{pmatrix} 1 & -1 \\ 2 & 4 \end{pmatrix}$. On prend $v = (1, 0)$, la matrice à réduire est alors :

$$\begin{pmatrix} 1 & -1 & -11 \\ 2 & 10 & 38 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & -6 \\ 0 & 1 & 5 \end{pmatrix}$$

Le noyau est engendré par $(-6, 5, -1)$ donc $P = -x^2 + 5x - 6$.

Exemple 2

$$A = \begin{pmatrix} 3 & 2 & -2 \\ -1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

en prenant $v = (1, 0, 0)$ on obtient la matrice :

$$A = \begin{pmatrix} 1 & 3 & 5 & 7 \\ 0 & -1 & -2 & -3 \\ 0 & 1 & 2 & 3 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & -1 & -2 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

le permier vecteur du noyau est $(-1, 2, -1)$ d'où un polynôme divisant le polynôme minimal $-x^2 + 2x - 1$.

11.2.2 Le polynôme caractéristique

Pour une matrice générique, le polynôme caractéristique est égal au polynôme minimal, il est donc intéressant de chercher si le polynôme annulateur de A sur un vecteur aléatoire est de degré n, car le temps de calcul du polynôme caractéristique est alors en $O(n^3)$. Si cette méthode probabiliste échoue, on se rabat sur une des méthode déterministe ci-dessous :

- on utilise la formule $\det(\lambda I - A)$ déterminé par une des méthodes de calcul de déterminant ci-dessus. Cela nécessite $O(n^3)$ opérations mais avec des coefficients polynômes en λ.
- on fait une interpolation de Lagrange en donnant $n+1$ valeurs distinctes à λ. Ce qui nécessite $O(n^4)$ opérations mais avec des coefficients indépendants de λ, de plus cette méthode est facile à programmer de manière parallèle.
- si la matrice est à coefficients entiers on peut utiliser la méthode de Hessenberg (voir ci-dessous), on calcule une borne à priori sur les coefficients du polynôme caractéristique (cf. Cohen p.58-59) :

  $$|P_k| \leq \begin{pmatrix} n \\ n-k \end{pmatrix} (n-k)^{(n-k)/2} |M|^{n-k} ,$$

  on calcule le polynôme caractéristique modulo suffisamment de petits entiers puis on remonte par les restes chinois.

11.2.3 La méthode de Hessenberg

Pour les matrices à coefficients de taille bornée (modulaires par exemple) on préfère la méthode de Hessenberg qui est plus efficace, car elle nécessite de l'ordre de n^3 opérations sur les coefficients.

On se raméne d'abord à une matrice triangulaire supérieure à une diagonale près qui est semblable à la matrice de départ puis on applique une formule de récurrence pour calculer les coefficients du polynôme caractéristique.

Algorithme de réduction de Hessenberg :
Dans une colonne m donnée de la matrice H, on cherche à partir de la ligne $m+1$ un coefficient non nul. S'il n'y en a pas on passe à la colonne suivante. S'il y en a un en ligne i, on échange les lignes $m+1$ et i et les colonnes $m+1$ et i. Ensuite pour tout $i \geq m+2$, soit $u = H_{i,m}/H_{m+1,m}$, on remplace alors la ligne L_i de H par $L_i - uL_{m+1}$ et la colonne C_{m+1} par $C_{m+1} + uC_i$ ce qui revient "à remplacer le vecteur e_{m+1} de la base par le vecteur $e_{m+1} + ue_i$" ou plus précisément à multiplier à gauche par $\begin{pmatrix} 1 & 0 \\ -u & 1 \end{pmatrix}$ et à droite par la matrice inverse $\begin{pmatrix} 1 & 0 \\ u & 1 \end{pmatrix}$ (en utilisant

les lignes et colonnes $m+1$ et i au lieu de 1 et 2 pour ces matrices). Ceci a pour effet d'annuler le coefficient $H_{i,m}$ dans la nouvelle matrice.

On obtient ainsi en $O(n^3)$ opérations une matrice H' semblable à H de la forme :

$$\begin{pmatrix} H'_{1,1} & H'_{1,2} & ... & H'_{1,n-2} & H'_{1,n-1} & H'_{1,n} \\ H'_{2,1} & H'_{2,2} & ... & H'_{2,n-2} & H'_{2,n-1} & H'_{2,n} \\ 0 & H'_{3,2} & ... & H'_{3,n-2} & H'_{3,n-1} & H'_{3,n} \\ 0 & 0 & ... & H'_{4,n-2} & H'_{4,n-1} & H'_{4,n} \\ \vdots & \vdots & ... & \vdots & \vdots & \vdots \\ 0 & 0 & ... & 0 & H'_{n,n-1} & H'_{n,n} \end{pmatrix}$$

On calcule alors le polynôme caractéristique de H' par une récurrence qui s'obtient en développant le déterminant par rapport à la dernière colonne :

$$\begin{aligned} h_n(\lambda) = \det(\lambda I_n - H) &= (\lambda - H'_{n,n})h_{n-1}(\lambda) - (-H'_{n-1,n})(-H'_{n,n-1})h_{n-2}(\lambda) + \\ &\quad +(-H'_{n-2,n})(-H'_{n,n-1})(-H'_{n-1,n-2})h_{n-3}(\lambda) - ... \end{aligned}$$

où les h_i s'entendent en gardant les i premières lignes/colonnes de H'. On peut écrire cette formule pour $m \leq n$:

$$h_m(\lambda) = (\lambda - H'_{m,m})h_{m-1}(\lambda) - \sum_{i=1}^{m-1} H'_{m-i,m} \prod_{j=1}^{i-1} H'_{m-j+1,m-j} h_{i-1}(\lambda)$$

Pour effectuer cette récurrence de manière efficace, on conserve les $h_m(\lambda)$ dans un tableau de polynômes et on utilise une variable produit contenant successivement les $\prod H'_{m-j+1,m-j}$.

11.2.4 La méthode de Leverrier-Faddeev-Souriau

Cette méthode permet le calcul simultané des coefficients p_i $(i = 0..n)$ du polynôme caractéristique $P(\lambda) = \det(\lambda I - A)$ et des coefficients matriciels B_i $(i = 0..n-1)$ du polynôme en λ donnant la matrice adjointe (ou comatrice) $B(\lambda)$ de $\lambda I - A$:

$$(\lambda I - A)B(\lambda) = (\lambda I - A) \sum_{k \leq n-1} B_k \lambda^k = (\sum_{k \leq n} p_k \lambda^k) I = P(\lambda) I \qquad (31)$$

Remarquons que cette équation donne une démonstration assez simple de Cayley-Hamilton puisque le reste de la division euclidienne du polynôme $P(\lambda)I$ par $\lambda I - A$ est $P(A)$.

Pour déterminer simultanément les p_k et B_k, on a les relations de récurrence :

$$B_{n-1} = p_n I = I, \quad B_k - AB_{k+1} = p_{k+1} I \qquad (32)$$

Il nous manque une relation entre les p_k et B_k pour pouvoir faire le calcul par valeurs décroissantes de k, on va montrer le :

Théorème 10 *La dérivée du polynôme caractéristique $P'(\lambda)$, est égale à la trace de la matrice adjointe de $\lambda I - A$*

$$tr(B) = P'(\lambda)$$

Le théorème nous donne $\text{tr}(B_k) = (k+1)p_{k+1}$. Si on prend la trace de (32), on a :

$$\text{tr}(B_{n-1}) = np_n, \quad (k+1)p_{k+1} - \text{tr}(AB_{k+1}) = np_{k+1}$$

donc on calcule p_{k+1} en fonction de B_{k+1} puis B_k :

$$p_{k+1} = \frac{\text{tr}(AB_{k+1})}{k+1-n}, \quad B_k = AB_{k+1} + p_{k+1}I$$

Démonstration du théorème :
Soient $V_1(\lambda), ...V_n(\lambda)$ les vecteurs colonnes de $\lambda I - A$ et $b_{i,j}(\lambda)$ les coefficients de B, on a :

$$\begin{aligned} P'(\lambda_0) &= \det(V_1(\lambda), V_2(\lambda), ..., V_n(\lambda))'_{|\lambda=\lambda_0} \\ &= \det(V_1'(\lambda_0), V_2(\lambda_0), ..., V_n(\lambda_0)) + \det(V_1(\lambda_0), V_2'(\lambda_0), ..., V_n(\lambda_0)) + \\ &\quad +... + \det(V_1(\lambda_0), V_2(\lambda_0), ..., V_n'(\lambda_0)) \end{aligned}$$

Il suffit alors de remarquer que $V_i'(\lambda_0)$ est le i-ième vecteur de la base canonique donc :

$$\det(V_1(\lambda_0), V_2(\lambda_0), ..., V_i'(\lambda_0), ..., V_n(\lambda_0)) = b_{i,i}(\lambda_0)$$

Finalement :

$$P'(\lambda_0) = \sum_{i=1}^{n} b_{i,i}(\lambda_0) = \text{tr}\left(B(\lambda_0)\right)$$

Remarque :
En réindexant les coefficients de P et B de la manière suivante :

$$\begin{aligned} P(\lambda) &= \lambda^n + p_1\lambda^{n-1} + p_2\lambda^{n-2} ... + p_n \\ B(\lambda) &= \lambda^{n-1}I + \lambda^{n-2}B_1 + ... + B_{n-1} \end{aligned}$$

on a montré que :

$$\begin{cases} A_1 = A, & p_1 = -\text{tr}(A), & B_1 = A_1 + p_1 I \\ A_2 = AB_1, & p_2 = -\frac{1}{2}\text{tr}(A_2), & B_2 = A_2 + p_2 I \\ \vdots & \vdots & \vdots \\ A_k = AB_{k-1}, & p_k = -\frac{1}{k}\text{tr}(A_k), & B_k = A_k + p_k I \end{cases}$$

On peut alors vérifier que $B_n = A_n + p_n I = 0$. D'où ce petit programme à utiliser avec xcas en mode mupad (`maple_mode(2);`), ou avec MuPAD, ou à adapter avec un autre système :

```
iequalj:=(j,k)->if j=k then return(1); else return(0); end_if;
faddeev:=proc(A) // renvoie la liste des matrices B et le polynome P
local Aj,AAj,Id,coef,n,pcara,lmat;
begin
 n:=ncols(A);
 Id:=matrix(n,n,iequalj);     // matrice identite
 Aj:=Id;
 lmat:=[];                    // B initialise a liste vide
 pcara:=[1];                  // coefficient de plus grand degre de P
```

```
 for j from 1 to n do
  lmat:=append(lmat,Aj);      // rajoute Aj a la liste de matrices
  AAj:=Aj*A;
  coef:=-trace(AAj)/j;        // mupad linalg::tr
  pcara:=append(pcara,coef);  // rajoute coef au polynome caracteristique
  Aj:=AAj+coef*Id;
 end_for;
 lmat,pcara;                  // resultat
end_proc;
```

11.2.5 Les vecteurs propres simples.

On suppose ici qu'on peut factoriser le polynôme caractéristique (ou calculer dans une extension algébrique d'un corps). Lorsqu'on a une valeur propre simple λ_0, en écrivant la relation $(A-\lambda_0 I)B(\lambda_0) = P(\lambda_0)I = 0$, on voit que les vecteurs colonnes de la matrice $B(\lambda_0)$ sont vecteurs propres. Remarquer que $B(\lambda_0) \neq 0$ sinon on pourrait factoriser $\lambda - \lambda_0$ dans $B(\lambda)$ et apreѕ simplifications on aurait :

$$(A - \lambda_0 I)\frac{B}{\lambda - \lambda_0}(\lambda_0) = \frac{P}{\lambda - \lambda_0}(\lambda_0)I$$

or le 2ème membre est inversible en λ_0 ce qui n'est pas le cas du premier. Pour avoir une base des vecteurs propres associés à λ_0, on calcule $B(\lambda_0)$ par la méthode de Horner appliquée au polynôme $B(\lambda)$ en $\lambda = \lambda_0$, et on réduit en colonnes la matrice obtenue.

11.2.6 La forme normale de Jordan

Pour les valeurs propres de multiplicité plus grande que 1, on souhaiterait généraliser la méthode ci-dessus pour obtenir une base de l'espace caractéristique, sous forme de cycles de Jordan. Soit λ_i, n_i les valeurs propres comptées avec leur multiplicité. On fait un développement de Taylor en λ_i :

$$\begin{aligned} -P(\lambda)I &= (A-\lambda I)\left(B(\lambda_i) + B'(\lambda_i)(\lambda-\lambda_i) + ... + \frac{B^{(n-1)}(\lambda_i)}{(n-1)!}(\lambda-\lambda_i)^{n-1}\right) \\ &= -(\lambda-\lambda_i)^{n_i}\prod_{j\neq i}(\lambda-\lambda_j)^{n_j} I \end{aligned}$$

Comme $A-\lambda I = A-\lambda_i I-(\lambda-\lambda_i)I$, on obtient pour les n_i premières puissances de $\lambda - \lambda_i$:

$$(A-\lambda_i I)B(\lambda_i) = 0 \quad (33)$$

$$(A-\lambda_i I)B'(\lambda_i) = B(\lambda_i) \quad (34)$$

$$... \quad (35)$$

$$(A-\lambda_i I)\frac{B^{(n_i-1)}(\lambda_i)}{(n_i-1)!} = \frac{B^{(n_i-2)}(\lambda_i)}{(n_i-2)!} \quad (36)$$

$$(A-\lambda_i I)\frac{B^{(n_i)}(\lambda_i)}{n_i!} - \frac{B^{(n_i-1)}(\lambda_i)}{(n_i-1)!} = -\prod_{j\neq i}(\lambda_i-\lambda_j)^{n_j} I \quad (37)$$

Le calcul des matrices $B^{(n)}(\lambda_i)/n!$ pour $n < n_i$ se fait en appliquant n_i fois l'algorithme de Horner (avec reste).

Théorème 11 *L'espace caractéristique de λ_i est égal à l'image de $B^{(n_i-1)}(\lambda_i)/(n_i-1)!$.*

Preuve :
On montre d'abord que Im$B^{(n_i-1)}(\lambda_i)/(n_i-1)!$ est inclus dans l'espace caractéristique correspondant à λ_i en appliquant l'équation (36) et les équations précédentes. Réciproquement on veut prouver que tout vecteur caractéristique v est dans l'image de $B^{(n_i-1)}(\lambda_i)/(n_i-1)!$. Prouvons le par récurrence sur le plus petit entier m tel que $(A-\lambda_i)^m v = 0$. Le cas $m = 0$ est clair puisque $v = 0$. Supposons le cas m vrai, prouvons le cas $m+1$. On applique l'équation (37) à v, il suffit alors de prouver que

$$w = (A-\lambda_i)\frac{B^{(n_i)}(\lambda_i)}{n_i!}v$$

appartient à l'image de $B^{(n_i-1)}(\lambda_i)/(n_i-1)!$. Comme $B^{(n_i)}(\lambda_i)$ commute avec A (car c'est un polynôme en A ou en appliquant le fait que $B(\lambda)$ inverse de $A-\lambda I$) :

$$(A-\lambda_i)^m w = \frac{B^{(n_i)}(\lambda_i)}{n_i!}(A-\lambda_i)^{m+1}v = 0$$

et on applique l'hypothèse de récurrence à w.

Pour calculer les cycles de Jordan, nous allons effectuer une réduction par le pivot de Gauß simultanément sur les colonnes des matrices $B^{(k)}(\lambda_i)/k!$ où $k < n_i$. La simultanéité a pour but de conserver les relations (33) à (36) pour les matrices réduites. Pour visualiser l'algorithme, on se représente les matrices les unes au-dessus des autres, colonnes alignées. On commence par réduire la matrice $B(\lambda_i)$ jusqu'à ce que l'on obtienne une matrice réduite **en recopiant** les opérations élémentaires de colonnes faites sur $B(\lambda_i)$ sur toutes les matrices $B^{(k)}(\lambda_i)/k!$. On va continuer avec la liste des matrices réduites issues de $B'(\lambda_i)$, ..., $B^{(n_i-1)}(\lambda_i)/(n_i-1)!$, mais en déplacant les colonnes non nulles de $B(\lambda_i)$ d'une matrice vers le bas (pour une colonne non nulle de la matrice réduite $B(\lambda)$ les colonnes correspondantes de $B^{(k)}(\lambda_i)$ réduite sont remplacées par les colonnes correspondantes de $B^{(k-1)}(\lambda_i)$ réduite pour k décroissant de n_i-1 vers 1). À chaque étape, on obtient une famille (éventuellement vide) de cycles de Jordan, ce sont les vecteurs colonnes correspondants aux colonnes non nulles de la matrice réduite du haut de la colonne. On élimine bien sûr les colonnes correspondant aux fins de cycles déjà trouvés.

Par exemple, si $B(\lambda_i) \neq 0$, son rang est 1 et on a une colonne non nulle, et un cycle de Jordan de longueur n_i fait des n_i vecteurs colonnes des matrices $B^{(k)}(\lambda_i)/k!$ réduites. Plus généralement, on obtiendra plus qu'un cycle de Jordan (et dans ce cas $B(\lambda_i) = 0$).

11.2.7 Exemple 1

$$A = \begin{pmatrix} 3 & -1 & 1 \\ 2 & 0 & 1 \\ 1 & -1 & 2 \end{pmatrix}$$

$\lambda = 2$ est valeur propre de multiplicité 2, on obtient :

$$B(\lambda) = \lambda^2 I + \lambda \begin{pmatrix} -2 & -1 & 1 \\ 2 & -5 & 1 \\ 1 & -1 & -3 \end{pmatrix} + \begin{pmatrix} 1 & 1 & -1 \\ -3 & 5 & -1 \\ -2 & 2 & 2 \end{pmatrix}$$

on applique l'algorithme de Horner :

$$\begin{aligned} B(2) &= \begin{pmatrix} 1 & -1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \\ B'(2) &= \begin{pmatrix} 2 & -1 & 1 \\ 2 & -1 & 1 \\ 1 & -1 & 1 \end{pmatrix} \end{aligned}$$

Comme $B(2) \neq 0$, on pourrait arrêter les calculs en utilisant une colonne non nulle et le cycle de Jordan associé $(2,2,1) \rightarrow (1,1,0) \rightarrow (0,0,0)$. Expliquons tout de même l'algorithme général sur cet exemple. La réduction de $B(2)$ s'obtient en effectuant les manipulations de colonnes $C_2 + C_1 \rightarrow C_2$ et $C_3 - C_1 \rightarrow C_3$. On effectue les mêmes opérations sur $B'(2)$ et on obtient :

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$
$$\begin{pmatrix} 2 & 1 & -1 \\ 2 & 1 & -1 \\ 1 & 0 & 0 \end{pmatrix}$$

L'étape suivante consiste à déplacer vers le bas d'une matrice les colonnes non nulles de la matrice du haut, on obtient :

$$\begin{pmatrix} 1 & 1 & -1 \\ 1 & 1 & -1 \\ 0 & 0 & 0 \end{pmatrix}$$

qui se réduit en :

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

on chercherait alors dans les colonnes 2 et 3 de nouveaux cycles (puisque la colonne 1 a déja été utilisée pour fournir un cycle).

11.2.8 Exemple 2

$$A = \begin{pmatrix} 3 & 2 & -2 \\ -1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

$\lambda = 1$ est valeur propre de multiplicité 3. On trouve :

$$\begin{aligned} B(1) &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \\ B'(1) &= \begin{pmatrix} 2 & 2 & -2 \\ -1 & -1 & 1 \\ 1 & 1 & -1 \end{pmatrix}, \\ \frac{B''(1)}{2} &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

Le processus de réduction commence avec $B'(1)$ en haut de la liste de matrices, on effectue les opérations élémentaires de colonne $C_2 - C_1 \to C_2$ et $C_3 + C_1 \to C_3$ et on obtient :

$$\begin{pmatrix} 2 & 0 & 0 \\ -1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix},$$
$$\begin{pmatrix} 1 & -1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

La première colonne donne le premier cycle de Jordan $(1, 0, 0) \to (2, -1, 1)$. On déplace les premières colonnes d'une matrice vers le bas :

$$\begin{pmatrix} 2 & -1 & 1 \\ -1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

qu'on réduit par les opérations $2C_2 + C_1 \to C_2$ et $2C_3 - C_1 \to C_3$ en :

$$\begin{pmatrix} 2 & 0 & 0 \\ -1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Puis on effectue $C_3 - C_2 \to C_3$ et la deuxième colonne nous donne le deuxième cycle de Jordan, réduit ici à un seul vecteur propre $(0, 1, 1)$.

11.2.9 Le polynôme minimal par Faddeev

On vérifie aisément que le degré du facteur $(\lambda - \lambda_i)$ dans le polynôme minimal de A est égal à $n_i - k$ où k est le plus grand entier tel que :

$$\forall j < k, \quad B^{(j)}(\lambda_i) = 0$$

11.2.10 Formes normales rationnelles

On se place ici dans une problématique différente : trouver une matrice semblable la plus simple possible sans avoir à introduire d'extension algébrique pour

factoriser le polynôme caractéristique. Quitte à "compléter" plus tard la factorisation et la jordanisation à partir de la forme simplifiée. Il existe diverses formes associées à une matrice et plusieurs algorithmes permettant de les relier entre elles, forme de Smith, de Frobenius, forme normale de Jordan rationnelle.

On va présenter une méthode directe de calcul d'une forme normale contenant le maximum de zéros (dont la forme dite normale de Jordan rationnelle peut se déduire) en utilisant le même algorithme que pour la forme normale de Jordan. Soit $Q(\lambda) = q_0 + ... + q_d\lambda^d$ un facteur irréductible de degré d et de multiplicité q du polynôme caractéristique P. Il s'agit de construire un sous-espace de dimension dq formé de "cycles de Jordan rationnels". On part toujours de la relation $(\lambda I - A)\sum_{k\leq n-1} B_k\lambda^k = P(\lambda)I$. On observe que $Q(\lambda)I - Q(A)$ est divisible par $(\lambda I - A)$ donc il existe une matrice $M(\lambda)$ telle que :

$$(Q(\lambda)I - Q(A))(\sum_{k\leq n-1} B_k\lambda^k) = Q(\lambda)^q M(\lambda)$$

On observe aussi que Q a pour coefficient dominant 1 puisqu'il divise P, on peut donc effectuer des divisions euclidiennes de polynômes donc de polynômes à coefficients matriciels par Q sans avoir à diviser des coefficients. Ce qui nous permet de décomposer $B(\lambda) = \sum_{k\leq n-1} B_k\lambda^k$ en puissances croissantes de Q :

$$B(\lambda) = \sum_k C_k(\lambda)Q(\lambda)^k, \quad \deg(C_k) < q$$

On remplace et on écrit que les coefficients des puissances inférieures à q de Q sont nulles (la k-ième étant non nulle car $M(\lambda)$ n'est pas divisible par Q pour les mêmes raisons que pour la forme normale de Jordan). On a donc les relations :

$$Q(A)C_0 = 0, \quad C_k = Q(A)C_{k+1}$$

ce qui donne une colonne de matrice $C_{q-1} \to C_{q-2}... \to C_0 \to 0$ qui sont images l'une de l'autre en appliquant $Q(A)$. On peut alors faire l'algorithme de réduction simultanée sur les colonnes des C_j. On observe ensuite que le nombre de cycles de Jordan de $Q(A)$ de longueur donnée est un multiple de d, en effet il suffit de multiplier un cycle par A, ..., A^{d-1} pour créer un autre cycle, de plus ces cycles forment des familles libres car on a supposé Q irréductible. On peut donc choisir pour un cycle de longueur k des bases de la forme $(v_{k-1}, Av_{k-1}..., A^{d-1}v_{k-1}) \to ... \to (v_0, Av_0..., A^{d-1}v_0) \to (0, ..., 0)$ où la flèche $\to$ désigne l'image par $Q(A)$. Si on écrit la matrice de A dans la base $v_0, Av_0..., A^{d-1}v_0, ..., v_{k-1}, Av_{k-1}..., A^{d-1}v_{k-1}$ on obtient un "quasi-bloc de Jordan rationnel" de taille kd multiple de d :

$$\begin{pmatrix}
0 & 0 & ... & -q_0 & 0 & 0 & ... & 1 & ... \\
1 & 0 & ... & -q_1 & 0 & 0 & ... & 0 & ... \\
0 & 1 & ... & -q_2 & 0 & 0 & ... & 0 & ... \\
\vdots & \vdots & ... & \vdots & \vdots & \vdots & ... & \vdots & ... \\
0 & 0 & ... & -q_{d-1} & 0 & 0 & ... & 0 & ... \\
 & & & & & & & & \\
0 & 0 & ... & 0 & 0 & 0 & ... & -q_0 & ... \\
0 & 0 & ... & 0 & 1 & 0 & ... & -q_1 & ... \\
\vdots & \vdots & ... & \vdots & \vdots & \vdots & ... & \vdots & ...
\end{pmatrix}$$

Exemple

Soit la matrice

$$A=\begin{pmatrix} 1 & -2 & 4 & -2 & 5 & -4 \\ 0 & 1 & \frac{5}{2} & \frac{-7}{2} & 2 & \frac{-5}{2} \\ 1 & \frac{-5}{2} & 2 & \frac{-1}{2} & \frac{5}{2} & -3 \\ 0 & -1 & \frac{9}{2} & \frac{-7}{2} & 3 & \frac{-7}{2} \\ 0 & 0 & 2 & -2 & 3 & -1 \\ 1 & \frac{-3}{2} & \frac{-1}{2} & 1 & \frac{3}{2} & \frac{1}{2} \end{pmatrix}$$

Son polynôme caractéristique est $(x-2)^2(x^2-2)^2$ et on va déterminer la partie bloc de Jordan rationnel correspondant au facteur irréductible sur les entiers $Q(x) = (x^2-2)$ de multiplicité $q=2$. On calcule $B(x)$ et l'écriture de B comme somme de puissances de Q (ici avec `xcas` en mode `xcas`) :

```
A:=[[1,-2,4,-2,5,-4],[0,1,5/2,(-7)/2,2,(-5)/2],[1,(-5)/2,2,1/(-2),5/2,-3]
    [0,-1,9/2,(-7)/2,3,(-7)/2],[0,0,2,-2,3,-1],[1,(-3)/2,1/(-2),1,3/2,1/2
P:=det(A-x*idn(6));
B:=normal(P*inv(A-x*idn(6))); // preferer un appel a faddeev bien sur!
ecriture(B,Q,q):={
  local j,k,l,n,C,D,E;
  C:=B;
  D:=B;
  E:=NULL;
  n:=coldim(B);
  for (j:=0;j<q;j++){
    for (k:=0;k<n;k++){
      for (l:=0;l<n;l++){
        D[k,l]:=rem(C[k,l],Q,x);
        C[k,l]:=quo(C[k,l],Q,x);
      }
    }
    E:=E,D;
  }
  return E;
};
E:=ecriture(B,x^2-2,2);
QA:=A*A-2*idn(6);
```

On vérifie bien que `normal(QA*E(0))` et `normal(QA*E(1))-E(0))` sont nuls. On sait qu'on a un bloc de taille 2 de cycles de Jordan de longueur 2, donc il n'est pas nécessaire de faire des réductions ici, il suffit de prendre une colonne non nulle de $E(0)$, par exemple la première colonne en $x=0$ et la colonne correspondante de $E(1)$ et leurs images par A, ici cela donne $(4, 24, 12, 32, 8, -4)$ correspondant à $(0, 4, -4, 8, 4, -4)$, on calcule les images par A, la matrice de l'endomorphisme restreint à ce sous-espace est alors le bloc de taille 4 :

$$\begin{pmatrix} 0 & 2 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Cette forme normale minimise le nombre de coefficients non nuls, mais présente un inconvénient, la partie nilpotente ne commute pas avec la partie bloc-diagonale, contrairement à la forme normale rationnelle de Jordan qui contient des blocs identités au-dessus de la diagonale de blocs. Pour créer la forme normale rationnelle de Jordan, on doit donc remplacer les blocs $\begin{pmatrix} \dots & 0 & 1 \\ \dots & 0 & 0 \\ \dots & & \end{pmatrix}$ par des matrices identités. Supposons constitués les j premiers blocs de taille d numérotés de 0 à $j-1$ avec comme base de vecteurs $(v_{0,0}, ..., v_{0,d-1}, ..., v_{j-1,d-1})$. Il s'agit de trouver un vecteur $v_{j,0}$ pour commencer le bloc suivant. On définit alors $v_{j,l}$ en fonction de $v_{j,l-1}$ en appliquant la relation $Av_{j,l-1} = v_{j,l} + v_{j-1,l-1}$. Il faut donc chercher $v_{j,0}$ tel que

$$Av_{j,d-1} = -q_0 v_{j,0} - ... - q_{d-1} v_{j,d-1} + v_{j-1,d-1} \tag{38}$$

En utilisant les relations de récurrence précédentes, on voit que cela revient à fixer $Q(A)v_{j,0}$ en fonction des $v_{j',l}$ avec $j' < j$ (l quelconque). Ce qui est toujours possible en utilisant la colonne de matrices $C_{j'}$ qui s'obtiennent en fonction des $C_{j'+1}$ en appliquant $Q(A)$.

Plus précisément, calculons les $v_{j,l}$ en fonction de $v_{j,0}$ et des $v_{j',l'}$ ($j' < j$). On utilise les coefficients binomiaux $\binom{l}{m}$ calculés par la règle du triangle de Pascal et on montre par récurrence que :

$$v_{j,l} = A^l v_{j,0} - \sum_{m=1}^{\inf(l,j)} \binom{l}{m} v_{j-m,l-m} \tag{39}$$

On remplace dans (38) d'où :

$$A^d v_{j,0} - \sum_{m=1}^{\inf(d,j)} \binom{d}{m} v_{j-m,l-m} + \sum_{l=0}^{d} q_l (A^l v_{j,0} - \sum_{m=1}^{\inf(l,j)} \binom{l}{m} v_{j-m,l-m}) = 0$$

finalement :

$$Q(A)v_{j,0} = \sum_{l=1}^{d} q_l \sum_{m=1}^{\inf(l,j)} \binom{l}{m} v_{j-m,l-m} \tag{40}$$

Application à l'exemple :
Ici $v_{0,0} = (4, 24, 12, 32, 8, -4)$ et $v_{0,1} = Av_{j,0}$ dont une préimage par $Q(A)$ est $w_{1,0} = (0, 4, -4, 8, 4, -4)$ et $w_{1,1} = Aw_{1,0}$. On applique (40), comme $q_1 = 0$ et $q_2 = 1$ on doit avoir :

$$Q(A)v_{1,0} = \sum_{l=1}^{2} q_l \sum_{m=1}^{\inf(l,1)} \binom{l}{m} v_{1-m,l-m} = 2v_{0,1}$$

donc :

$$\begin{array}{rcccc} v_{1,0} & = & 2A(0,4,-4,8,4,-4) & = & (-8,-32,0,-48,-16,16) \\ v_{1,1} & = & Av_{1,0} - v_{0,0} & = & (4,40,-4,64,24,-20) \end{array}$$

On vérifie bien que $Av_{1,1} = 2v_{1,0} + v_{0,1}$.

11.2.11 Fonctions analytiques

Soit f une fonction analytique et M une matrice. Pour calculer $f(M)$, on calcule la forme normale de Jordan de $M = P(D+N)P^{-1}$ où $D =$diag$(d_1, ..., d_m)$ est diagonale et N nilpotente d'ordre n. On calcule aussi le développement de Taylor formel de f en x à l'ordre $n-1$, on a alors :

$$f(N) = P\left(\sum_{j=0}^{n-1} \frac{\text{diag}(f^{(j)}(d_1), ..., f^{(j)}(d_m))}{j!} N^j\right) P^{-1}$$

11.3 Quelques autres algorithmes utiles

11.3.1 Complexité asymptotique

Pour calculer le produit de matrices, on peut utiliser l'algorithme de Strassen, on présente ici la variante de Winograd. Soit à calculer :

$$\begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix} = \begin{pmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{pmatrix}$$

On calcule :

$$\begin{gathered} s_1 = a_{2,1} + a_{2,2}, \quad s_2 = s_1 - a_{1,1}, \quad s_3 = a_{1,1} - a_{2,1}, \quad s_4 = a_{1,2} - s_2 \\ t_1 = b_{1,2} - b_{1,1}, \quad t_2 = b_{2,2} - t_1, \quad t_3 = b_{2,2} - b_{1,2}, \quad t_4 = b_{2,1} - t_2 \end{gathered}$$

puis :

$$\begin{aligned} p_1 = a_{1,1}b_{1,1}, \quad p_2 = a_{1,2}b_{2,1}, \quad p_3 = s_1t_1, \quad p_4 = s_2t_2 \\ p_5 = s_3t_3, \quad p_6 = s_4b_{2,2}, \quad p_7 = a_{2,2}t_4 \\ u_1 = p_1 + p_2 \quad u_2 = p_1 + p_4, \quad u_3 = u_2 + p_5, \quad u_4 = u_3 + p_7 \\ u_5 = u_3 + p_3, \quad u_6 = u_2 + p_3, \quad u_7 = u_6 + p_6 \end{aligned}$$

Alors $c_{1,1} = u_1, c_{1,2} = u_7, c_{2,1} = u_4, c_{2,2} = u_5$.
Cet algorithme utilise 7 multiplications et 15 additions ce qui économise 1 multiplication et permet en appliquant récursivement cet algorithme pour des matrices blocs de réduire la complexité d'un produit de grandes matrices normalement en $O(n^3 = n^{\ln(8)/\ln(2)})$ à $O(n^{\ln(7)/\ln(2)})$ (la preuve est analogue à celle de la multiplication des polynômes par l'algorithme de Karatsuba).

En utilisant une factorisation LU par blocs, on peut montrer que cette complexité asymptotique se généralise au calcul de l'inverse. On peut d'ailleurs améliorer l'exposant, mais la constante non explicitée dans le O augmente aussi. En pratique, Strassen n'est pas utilisée pour des matrices de taille plus petites que plusieurs centaines de lignes et colonnes.

De même on peut gagner sur le calcul du polynôme minimal en faisant des opérations de multiplication par bloc.

11.3.2 Numériques

La plupart des algorithmes d'algèbre linéaire "numérique" ont une utilité en calcul exact : par exemple la factorisation LU (avec les variations décrites dans

la section réduction de Gauß), la factorisation QR (et donc la méthode de Gram-Schmidt, ici pour des raisons d'efficacité on orthogonalise d'abord la base de départ et on la normalise à la fin seulement), Cholesky,....

Certains algorithmes numériques peuvent s'utiliser conjointement à des algorithmes exacts. Par exemple, pour une matrice M à coefficients rationnels, on peut localiser par des méthodes exactes les racines du polynôme caractéristique, on trouve une valeur approchée r d'une valeur propre à une précision fixée, puis calculer par la méthode de la puissance appliquée à $(M - rI)^{-1}$ un vecteur propre approché. Inversement, des méthodes de diagonalisation numériques couplées à du calcul exact peuvent permettre de localiser des valeurs propres complexes.

11.3.3 Décomposition de Schur

Il s'agit d'une factorisation de matrice sous la forme

$$A = PSP^{-1}$$

où P est unitaire et S diagonale supérieure. Existence (théorique) : on prend une valeur propre et un vecteur propre correspondant, puis on projette sur l'orthogonal de ce vecteur propre et on s'y restreint, on prend à nouveau une valeur propre et un vecteur propre correspondant, etc.

On peut approcher cette factorisation par un algorithme itératif qui utilise la factorisation QR d'une matrice quelconque comme produit d'une matrice unitaire par une matrice triangulaire supérieure à coefficients positifs sur la diagonale. On fait l'hypothèse que les valeurs propres de S sur la diagonale sont classées par ordre de module strictement décroissant $|\lambda_1| > |\lambda_2| > ... > |\lambda_n|$ (développement inspiré par Peter J. Olver dans le cas symétrique `http://www.math.umn.edu/~olver/aims_/qr.pdf`). On peut toujours s'y ramener quitte à remplacer A par $A - \alpha I$. Posons $A_1 = A$, et par récurrence $A_n = Q_n R_n$ (avec Q_n unitaire et R triangulaire supérieure à coefficients diagonaux positifs), $A_{n+1} = R_n Q_n$. On a alors

$$\begin{aligned} A^k &= (Q_1R_1)(Q_1R_1)(Q_1R_1)...(Q_1R_1)(Q_1R_1) \\ &= Q_1(R_1Q_1)(R_1Q_1)(R_1...Q_1)(R_1Q_1)R_1 \\ &= Q_1(Q_2R_2)(Q_2R_2)..(Q_2R_2)R_1 \\ &= Q_1Q_2(R_2Q_2)R_2..Q_2R_2R_1 \\ &= Q_1Q_2(Q_3R_3)...Q_3R_3R_2R_1 \\ &= Q_1...Q_kR_k...R_1 \end{aligned}$$

D'autre part $A = PSP^{-1}$ donc $A^k = PS^kP^{-1}$. Soit D la forme diagonale de S et U la matrice de passage $S = UDU^{-1}$, où U est triangulaire supérieure et où on choisit la normalisation des coefficients sur la diagonale de U valant 1. On a donc

$$A^k = PUD^kU^{-1}P^{-1}$$

Ensuite, on suppose qu'on peut factoriser $U^{-1}P^{-1} = L\tilde{U}$ sans permutations, donc qu'on ne rencontre pas de pivot nul, et quitte à multiplier les vecteurs unitaires de P^{-1} par une constante complexe de module 1 on peut supposer que les pivots sont positifs donc que $\tilde{U}$ a des coefficients positifs sur la diagonale, on a alors

$$A^k = PUD^kL\tilde{U} = Q_1...Q_kR_k...R_1$$

puis en multipliant par $U^{-1}|D|^{-k}$

$$PUD^kL|D|^{-k} = Q_1...Q_kR_k...R_1\tilde{U}^{-1}|D-^{-k}$$

où $R_k...R_1\tilde{U}^{-1}|D|^{-k}$ est triangulaire supérieure à coefficients positifs sur la diagonale et $Q_1...Q_k$ est unitaire. On regarde ensuite les entrées de la matrice $D^kL|D|^{-k}$, sous la diagonale elles convergent (géométriquement) vers 0, donc $UD^kL|D|^{-k}$ tend vers une matrice triangulaire supérieure dont les coefficients diagonaux valent $e^{ik\arg(\lambda_j)}$. On montre que cela entraine que $Q_1...Q_k$ est équivalent à $P(D/|D|)^k$

$$Q_1...Q_k \approx P(D/|D|)^k, \quad R_k...R_1\tilde{U}^{-1}|D|^{-k} \approx (D/|D|)^{-k}UD^kL|D|^{-k}$$

Donc, Q_k tend à devenir diagonale, et $R_kQ_k = A_{k+1}$ triangulaire supérieure. De plus

$$A = Q_1A_2Q_1^{-1} = ... = Q_1...Q_kA_{k+1}(Q_1...Q_k)^{-1}$$

la matrice A_{k+1} est donc semblable à A.

En pratique, on n'impose pas la positivité des coefficients diagonaux de R dans la factorisation QR, ce qui ne change évidemment pas le fait que Q_k s'approche d'une matrice diagonale et A_k d'une matrice triangulaire supérieure (avec convergence à vitesse géométrique). On utilise aussi des "shifts" pour accélerer la convergence, c'est-à-dire qu'au lieu de faire QR et RQ sur la matrice A_k on le fait sur $A_k - \alpha_kI$ où λ_k est choisi pour accélerer la convergence vers 0 du coefficient d'indice ligne n colonne $n-1$ (idéalement il faut prendre α_k proche de λ_n la valeur propre de module minimal, afin de minimiser $|\lambda_n - \alpha_k|/|\lambda_{n-1} - \alpha_k|$). En effet, si $A_k - \lambda_kI = Q_kR_k$ et $A_{k+1} = R_kQ_k + \lambda_kI$ alors :

$$\begin{aligned}
(A-\alpha_1I)...(A-\alpha_kI) &= Q_1R_1(Q_1R_1 - (\alpha_2-\alpha_1)I)...(Q_1R_1 - (\alpha_2-\alpha_1)I) \\
&= Q_1(R_1Q_1 - (\alpha_2-\alpha_1)I)R_1(Q_1R_1 - (\alpha_3-\alpha_1)I)...(Q_1R_1 - (\alpha_2-\alpha_1)I) \\
&= Q_1(A_2 - \alpha_1I - (\alpha_2-\alpha_1)I)R_1Q_1(R_1Q_1 - (\alpha_3-\alpha_1)I)R_1...(Q_1R_1 - (\alpha_2 - \\
&= Q_1(A_2-\alpha_2I)(A_2-\alpha_3I)...(A_2-\alpha_{k-1}I)R_1 \\
&= ... \\
&= Q_1...Q_kR_k...R_1
\end{aligned}$$

On peut aussi éliminer la dernière ligne et la dernière colonne de la matrice pour accélerer les calculs dès que le coefficient en ligne n colonne $n-1$ est suffisamment petit.

On remarque que pour une matrice réelle si on choisit des shifts conjugués, alors $Q_1...Q_kR_k...R_1$ est réel. Or si $QR = \overline{QR}$ et si R est inversible

$$\overline{Q}^{-1}Q = \overline{R}R^{-1}$$

On a donc une matrice symétrique (car $\overline{Q}^{-1} = Q^t$) et triangulaire supérieure. On en déduit que $\overline{Q}^{-1}Q = D$ est diagonale, donc $Q = \overline{Q}D$. On peut donc rendre Q réelle en divisant chaque colonne par un $e^{i\theta}$, et rendre R réelle en conjuguant par la matrice D. Mais ce procédé de retour au réel après élimination de 2 valeurs propres complexes conjuguées d'une matrice réelle se heurte à un problème de conditionnement parce que le choix d'un shift intéressant pour la convergence va rendre

la matrice R proche d'une matrice non inversible (les deux derniers coefficients diagonaux de R sont proches de 0). On a alors seulement

$$\overline{Q}^{-1}QR = \overline{R}$$

Si on décompose $\overline{Q}^{-1}Q$, R, $\overline{R}$ par blocs $n-2, n-2$, $n-2, 2$, $2, n-2$ et $2, 2$, on a

$$\begin{pmatrix} QQ_{11} & QQ_{12} \\ QQ_{21} & QQ_{22} \end{pmatrix}\begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} = \begin{pmatrix} QQ_{11}R_{11} & QQ_{21}R_{11} \\ QQ_{11}R_{12}+QQ_{12}R_{22} & QQ_{21}R_{12}+QQ_{22}R_{22} \end{pmatrix}$$
$$= \begin{pmatrix} \overline{R}_{11} & \overline{R}_{12} \\ 0 & \overline{R}_{22} \end{pmatrix}$$

Donc on a $QQ_{11} = \overline{R}_{11}R_{11}^{-1}$. Comme Q est unitaire, $QQ = \overline{Q}^{-1}Q = Q^t\,Q$ est symétrique, donc QQ_{11} est diagonale puisque symétrique et triangulaire supérieure. On peut donc ramener Q_{11} et R_{11} en des matrices réelles.

Revenons à la localisation des valeurs propres. On suppose qu'on a maintenant une matrice unitaire P et une matrice triangulaire supérieure S (aux erreurs d'arrondi près) telles que

$$P^{-1}AP = S$$

Que peut-on en déduire ? On va d'abord arrondir P en une matrice exacte à coefficients rationnels, dont les dénominateurs sont une puissance de 2 (en fait c'est exactement ce que donne l'écriture d'un flottant en base 2, une fois ramené tous les exposants à la même valeur). On a donc une matrice P_e presque unitaire exacte et telle que

$$S_e = P_e^{-1}AP_e$$

est semblable à A, et presque triangulaire supérieure. (comme P_e est presque unitaire, sa norme et la norme de son inverse sont proches de 1 donc S_e est proche de S, les coefficients de S_e sont de la même taille que les coefficients de A : le changement de base est bien conditionné et c'est la raison pour laquelle on a choisi d'effectuer des transformations unitaires).

Notons $\mu_1, ..., \mu_n$ les coefficients diagonaux de S_e, soit ε un majorant de la norme des coefficients sous-diagonaux de S_e, et soit δ un minorant de l'écart entre 2 μ_j distincts. On a donc $S_e = U + E$ où U est triangulaire supérieure, E est triangulaire inférieure avec des 0 sous la diagonale et des coefficients de module majorés par ε. Si ε est suffisamment petit devant δ, on va montrer qu'on peut localiser les valeurs propres de S_e (qui sont celles de A) au moyen des μ_j.

En effet, fixons j et soit C un cercle de centre $\mu = \mu_j$ et de rayon $\alpha \leq \delta/2$. Si A est une matrice diagonalisable, on sait que

$$\text{nombre de valeurs propres} \in C = \frac{1}{2i\pi}\text{trace}\int_C (A - zI)^{-1}$$

En prenant $A = S_e$, et en écrivant

$$(S_e - zI)^{-1} = (U - zI + E)^{-1} = (I + (U - zI)^{-1}E)^{-1}(U - zI)^{-1}$$

on développe le second terme si la norme de $(U-zI)^{-1}E$ est strictement inférieure à 1

$$(S_e-zI)^{-1} = (U-zI)^{-1}-(U-zI)^{-1}E(U-zI)^{-1}+(U-zI)^{-1}E(U-zI)^{-1}E(U-zI)^{-1}+...$$

puis on calcule la trace

$$\text{trace}(S_e - zI)^{-1} = \sum_j (\mu_j - z)^{-1} + \eta$$

avec

$$|\eta| \leq 2\pi\alpha ||(U - zI)^{-1}|| \frac{||(U - zI)^{-1}E||}{1 - ||(U - zI)^{-1}E||}$$

Au final, le nombre de valeurs propres dans C est donné par

$$1 + \tilde{\eta}, \quad |\tilde{\eta}| \leq \alpha \text{max}_{z \in C} ||(U - zI)^{-1}|| \frac{||(U - zI)^{-1}E||}{1 - ||(U - zI)^{-1}E||}$$

Il suffit donc que le max soit plus petit que 1 pour avoir l'existence d'une valeur propre et une seule de S_e dans le cercle C (à distance au plus α de μ). Ce sera le cas si

$$\varepsilon \leq \frac{1}{2} \left(\frac{\delta}{2||S_e||} \right)^{n-1} \frac{\alpha}{\sqrt{n-1}}$$

on choisit donc α pour réaliser l'égalité ci-dessus, sous réserve que δ ne soit pas trop petit, rappelons que α doit être plus petit ou égal à $\delta/2$. Si δ est petit, il peut être nécessaire d'utiliser une précision plus grande pour les calculs de la décomposition de Schur en arithmétique flottante.

Typiquement, on peut espérer (pour un écart δ pas trop petit) pouvoir localiser les racines d'un polynôme de degré n par cette méthode avec précision b bits en $O(n^3b^2 + n^2b^3)$ opérations pour le calcul de la décomposition de Schur en flottant (n^3b^2 pour Hessenberg initial puis n^2b^2 par itération et un nombre d'itérations proportionnel à b). Pour le calcul exact de S_e, il faut inverser une matrice de taille n avec des coefficients de taille proportionnelle à b donc $O(n^4 b \ln(n))$ opérations (en modulaire, la taille des coefficients de l'inverse est $O(nb \ln(n))$) puis calculer un produit avec une matrice n, n de coefficients de taille proportionnelle à b, soit $O(n^4 b^2 \ln(nb))$ opérations. Asymptotiquement, on peut faire mieux avec des méthodes de multiplication et d'opérations matricielles par blocs. Pour éviter la perte d'un facteur n, on peut aussi ne pas faire de calculs en mode exact et controler les erreurs sur la matrice S. On peut regrouper les valeurs propres par "clusters" si elles sont trop proches à la précision de b bits. Pour la recherche des racines d'un polynôme P, on peut montrer, en calculant le résultant de P et de P' qui est en module plus grand ou égal à 1, et en l'écrivant comme produit des différences des racines, et en majorant toutes les différences de racine sauf une à l'aide de la norme infinie de P, qu'il faut $b = O(n)$ bits pour séparer les racines).

11.3.4 Autres

On peut aussi facilement programmer la recherche de la décomposition ${}^tPDP$ d'une matrice symétrique et en déduire la signature d'une forme quadratique. Citons enfin l'algorithme LLL (cf. Cohen) qui est utile dans de nombreux domaines (il permet de trouver des vecteurs assez courts dans un réseau, ce ne sont pas les plus courts, mais en contrepartie on les trouve très vite).

11.4 Quelques références

- Comme toujours on renvoie à l'excellent livre de Henri Cohen : A Course in Computational Algebraic Number Theory
- Gantmacher : Théorie des matrices
- Pour une implémentation des algorithmes de forme normale de Smith ou de Frobenius, cf. le source de MuPAD ou
  `http://www.mapleapps.com/maplelinks/share/normform.html`
- Ferrard, Lemberg : Mathématiques Concrètes, Illustrées par la TI 92 et la TI 89
  Présente aussi des algorithmes plus numériques, et le lien avec la diagonalisation numérique de matrices.
- Press et al. : Numerical recipies in Fortran/C/Pascal.
  Pour des algorithmes numériques (sur les matrices et autres).

11.5 Exercices (algèbre linéaire)

11.5.1 Instructions

- Les commandes d'algèbre linéaire de Xcas sont regroupées dans le menu `Math->Alglin`. En maple et mupad, la commande `?linalg` affiche la liste des commandes d'algèbre linéaire.
- En maple il est conseillé d'exécuter `with(linalg);`, en mupad `export(linalg);`, sinon il faut précéder chaque commande de `linalg ::`.
- En maple, attention il faut utiliser le caractère & avant la multiplication et il faut souvent utiliser `evalm` dans les programmes utilisant des matrices et vecteurs.
- Pour travailler avec des coefficients modulaires, en Xcas on fait suivre les coefficients ou matrices de `% n`, en maple on utilise les noms de commandes avec une majuscule (forme inerte) suivi de `mod n`, en mupad on définit les coefficients dans l'anneau, par exemple

```
Z19:=Dom::IntegerMod(19): MatZ19 := Dom::Matrix(Z19):
A:=MatZ19([[1, 2], [2]]); Z19(5)*A;
```

11.5.2 Exercices

1. En utilisant un logiciel de calcul formel, comparez le temps de calcul d'un déterminant de matrice aléatoire à coefficients entiers de tailles 50 et 100, d'une matrice de taille 6 et 12 avec comme coefficients symboliques ligne j colonne k, x_{j+k} lorsque $j+k$ est pair et 0 sinon. Peut-on en déduire une indication sur l'algorithme utilisé ?
2. Écrire un programme calculant la borne de Hadamard d'un déterminant à coefficients réels (rappel : c'est la borne obtenue en faisant le produit des normes euclidiennes des vecteurs colonnes).
3. Créez une matrice 4x4 aléatoire avec des coefficients entiers compris entre -100 et 100, calculer la borne de Hadamard de son déterminant avec le programme précédent, calculer ce déterminant modulo quelques nombres premiers choisis en fonction de la borne de Hadamard et vérifiez le résultat de la reconstruction modulaire du déterminant.

4. Créez une matrice 100x100 aléatoire à coefficients entiers et calculez son déterminant modulo quelques nombres premiers. Dans quels cas peut-on conclure que la matrice est inversible dans $\mathbb{R}$? dans $\mathbb{Z}$?
5. Écrire un programme calculant par interpolation de Lagrange le polynôme caractéristique d'une matrice (en donnant à λ de $\det(\lambda I - A)$, $n+1$ valeurs distinctes).
6. (Long) Écrire un programme qui calcule un déterminant de matrice en calculant les mineurs 2x2 puis 3x3 etc. (méthode de Laplace)
7. Recherche du polynôme minimal. On prend un vecteur aléatoire à coefficients entiers et on calcule v, Av, ..., $A^n v$ puis on cherche une relation linéaire minimale entre ces vecteurs, en calculant le noyau de la matrice ayant ces vecteurs colonnes. Si le noyau est de dimension 1, alors le polynôme minimal est égal au polynome caractéristique et correspond à un vecteur de la base du noyau. Sinon, il faut choisir un vecteur du noyau correspondant au degré le plus petit possible puis faire le PPCM avec les polynomes obtenurs avec d'autres vecteurs pour obtenir le polynôme minimal avec une grande probabilité. Essayez avec la matrice A de taille 3 ayant des 0 sur la diagonale et des 1 ailleurs. Écrire un programme mettant en oeuvre cette recherche, testez-le avec une matrice aléatoire de taille 30.
8. Testez l'algorithme méthode de Fadeev pour la matrice A ci-dessus. Même question pour
$$A = \begin{pmatrix} 3 & -1 & 1 \\ 2 & 0 & 1 \\ 1 & -1 & 2 \end{pmatrix}, \quad A = \begin{pmatrix} 3 & 2 & -2 \\ -1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$
9. Écrire un programme calculant par une méthode itérative la valeur propre de module maximal d'une matrice à coefficients complexes. Dans le cas réel, modifier le programme pour pouvoir traiter le cas d'un couple de complexes conjugués de module maximal. Dans le cas hermitien ou réel symétrique, éliminer le couple valeur propre/vecteur propre et continuer la diagonalisation numérique.
10. Soient $|a|, |b| < \sqrt{n/2}$ Écrire une fonction ayant comme arguments $a/b \pmod n$ qui calcule a et b.
Utiliser ce programme pour résoudre un système 4,4 à coefficients entiers par une méthode p-adique.

12 Interpolation

Étant donné la facilité de manipulation qu'apportent les polynomes, on peut chercher à approcher une fonction par un polynôme. De plus l'interpolation est un outil très utilisé pour calculer des polynômes en calcul formel.

12.1 Interpolation de Lagrange

La méthode la plus naturelle consiste à chercher un polynôme de degré le plus petit possible égal à la fonction en certains points $x_0, ..., x_n$ et à trouver une majo-

ration de la différence entre la fonction et le polynôme. Le polynome interpolateur de Lagrange répond à cette question.

12.1.1 Existence et contrôle de l'erreur.

Soit donc $x_0, ..., x_n$ des réels distincts et $y_0, ..., y_n$ les valeurs de la fonction à approcher en ces points (on posera $y_j = f(x_j)$ pour approcher la fonction f). On cherche donc P tel que $P(x_j) = y_i$ pour $j \in [0, n]$.

Commencons par voir s'il y a beaucoup de solutions. Soit P et Q deux solutions distinctes du problème, alors $P - Q$ est non nul et va s'annuler en $x_0, ..., x_n$ donc possède $n + 1$ racines donc est de degré $n + 1$ au moins. Réciproquement, si on ajoute à P un multiple du polynome $A = \prod_{j=0}^{n}(X - x_j)$, on obtient une autre solution. Toutes les solutions se déduisent donc d'une solution particulière en y ajoutant un polynome de degré au moins $n + 1$ multiple de A.

Nous allons maintenant construire une solution particulière de degré au plus n. Si $n = 0$, on prend $P = x_0$ constant. On procède ensuite par récurrence. Pour construire le polynôme correspondant à $x_0, ..., x_{n+1}$ on part du polynoôme P_n correspondant à $x_0, ..., x_n$ et on lui ajoute un multiple réel de A

$$P_{n+1} = P_n + a\prod_{j=0}^{n}(X - x_j)$$

Ainsi on a toujours $P_{n+1}(x_j) = y_j$ pour $j = 0, ..n$, on calcule maintenant a pour que $P_{n+1}(x_{n+1}) = y_{n+1}$. En remplacant avec l'expression de P_{n+1} ci-dessus, on obtient

$$P_n(x_{n+1}) + a\prod_{j=0}^{n}(x_{n+1} - x_j) = y_{n+1}$$

Comme tous les x_j sont distincts, il existe une solution unique a :

$$a = \frac{y_{n+1} - P_n(x_{n+1})}{\prod_{j=0}^{n}(x_{n+1} - x_j)}$$

On a donc prouvé le :

Théorème 12 *Soit $n + 1$ réels distincts $x_0, ..., x_n$ et $n + 1$ réels quelconques $y_0, ..., y_n$. Il existe un unique polynôme P de degré inférieur ou égal à n, appelé polynome de Lagrange, tel que :*

$$P(x_i) = y_i$$

Exemple : déterminons le polynome de degré inférieur ou égal à 2 tel que $P(0) = 1, P(1) = 2, P(2) = 1$. On commence par $P_0 = 1$. Puis on pose $P_1 = P_0 + aX = 1 + aX$. Comme $P(1) = 2 = 1 + a$ on en tire $a = 1$ donc $P_1 = 1 + X$. Puis on pose $P_2 = P_1 + aX(X - 1)$, on a $P_2(2) = 3 + 2a = 1$ donc $a = -1$, finalement $P_2 = 1 + X - X(X - 1)$.

On peut calculer le polynome de Lagrange comme indiqué ci-dessus, la méthode dite des différences divisées permettant de le faire de la manière la plus efficace possible (cf. par exemple Demailly).

Reste à estimer l'écart entre une fonction et son polynome interpolateur, on a le :

Théorème 13 *Soit f une fonction $n+1$ fois dérivable sur un intervalle $I=[a,b]$ de $\mathbb{R}$, $x_0,...,x_n$ des réels distincts de I. Soit P le polynome de Lagrange donné par les x_j et $y_j=f(x_j)$. Pour tout réel $x\in I$, il existe un réel $\xi_x\in[a,b]$ (qui dépend de x) tel que :*

$$f(x)-P(x)=\frac{f^{[n+1]}(\xi_x)}{(n+1)!}\prod_{j=0}^{n}(x-x_j) \tag{41}$$

Ainsi l'erreur commise dépend d'une majoration de la taille de la dérivée $n+1$-ième sur l'intervalle, mais aussi de la disposition des points x_j par rapport à x. Par exemple si les points x_j sont équidistribués, le terme $|\prod_{j=0}^{n}(x-x_j)|$ sera plus grand près du bord de I qu'au centre de I.

Preuve du théorème : Si x est l'un des x_j l'égalité est vraie. Soit

$$C=(f(x)-P(x))/\prod_{j=0}^{n}(x-x_j)$$

on considère maintenant la fonction :

$$g(t)=f(t)-P(t)-C\prod_{j=0}^{n}(t-x_j)$$

elle s'annule en x_j pour j variant de 0 à n ainsi qu'en x suite au choix de la constante C, donc g s'annule au moins $n+2$ fois sur l'intervalle contenant les x_j et x, donc g' s'annule au moins $n+1$ fois sur ce même intervalle, donc g'' s'annule au moins n fois, etc. et finalement $g^{[n+1]}$ s'annule une fois au moins sur cet intervalle. Or

$$g^{[n+1]}=f^{[n+1]}-C(n+1)!$$

car P est de degré inférieur ou égal à n et $\prod_{j=0}^{n}(x-x_j)-x^{n+1}$ est de degré inférieur ou égal à n. Donc il existe bien un réel ξ_x dans l'intervalle contenant les x_j et x tel que

$$C=\frac{f^{[n+1]}(\xi_x)}{(n+1)!}$$

12.1.2 Différences divisées

Calcul efficace du polynôme de Lagrange.
Avec la méthode de calcul précédent, on remarque que le polynôme de Lagrange peut s'écrire à la Horner sous la forme :

$$\begin{aligned}P(x) &= \alpha_0+\alpha_1(x-x_0)+...+\alpha_n(x-x_0)...(x-x_{n-1})\\ &= \alpha_0+(x-x_0)(\alpha_1+(x-x_1)(\alpha_2+...+(x-x_{n-2})(\alpha_{n-1}+(x-x_{n-1})\alpha_n)...))\end{aligned}$$

ce qui permet de le calculer rapidement une fois les α_i connus. On observe que

$$\alpha_0=f(x_0),\quad \alpha_1=\frac{f(x_1)-f(x_0)}{x_1-x_0}$$

On va voir que les α_k peuvent aussi se mettre sous forme d'une différence. On définit les différences divisées d'ordre n par récurrence

$$f[x_i]=f(x_i),\quad f[x_i,...,x_{k+i+1}]=\frac{f[x_{i+1},...,x_{k+i+1}]-f[x_i,...,x_{k+i}]}{x_{k+i+1}-x_i}$$

On va montrer que $\alpha_k = f[x_0, ..., x_k]$. C'est vrai au rang 0, il suffit donc de le montrer au rang $k+1$ en l'admettant au rang k. Pour cela on observe qu'on peut construire le polynôme d'interpolation en $x_0, ..., x_{k+1}$ à partir des polynômes d'interpolation P_k en $x_0, ..., x_k$ et Q_k en $x_1, ..., x_{k+1}$ par la formule :

$$P_{k+1}(x) = \frac{(x_{k+1} - x)P_k + (x - x_0)Q_k}{x_{k+1} - x_0}$$

en effet on vérifie que $P_{k+1}(x_i) = f(x_i)$ pour $i \in [1, k]$ car $P_k(x_i) = f(x_i) = Q_k(x_i)$, et pour $i = 0$ et $i = k+1$, on a aussi $P_{k+1}(x_0) = f(x_0)$ et $P_{k+1}(x_{k+1}) = f(x_{k+1})$. Or α_{k+1} est le coefficient dominant de P_{k+1} donc c'est la différence du coefficient dominant de Q_k et de P_k divisée par $x_{k+1} - x_0$, c'est-à-dire la définition de $f[x_0, ..., x_{k+1}]$ en fonction de $f[x_1, ..., x_{k+1}]$ et $f[x_0, ..., x_k]$.

Exemple : on reprend $P(0) = 1, P(1) = 2, P(2) = 1$. On a

x_i	$f[x_i]$	$f[x_i, x_{i+1}]$	$f[x_0, x_1, x_2]$
0	$\boxed{1}$		
		$(2-1)/(1-0) = \boxed{1}$	
1	2		$(-1-1)/(2-0) = \boxed{\text{-1}}$
		$(1-2)/(2-1) = -1$	
2	1		

donc $P(x) = \boxed{1} + (x-0)(\boxed{1} + (x-1)(\boxed{\text{-1}})) = 1 + x(2-x)$.

On peut naturellement utiliser l'ordre que l'on souhaite pour les x_i, en observant que le coefficient dominant de P ne dépend pas de cet ordre, on en déduit que $f[x_0, ..., x_k]$ est indépendant de l'ordre des x_i, on peut donc à partir du tableau ci-dessus écrire P par exemple avec l'ordre 2,1,0, sous la forme

$$P(x) = 1 + (x-2)(-1 + (x-1)(-1)) = 1 + (x-2)(-x)$$

12.2 Les splines

Il s'agit de fonctions définies par des polynomes de degré borné sur des intervalles, dont on fixe la valeur aux extrémités des intervalles (comme pour le polynome de Lagrange) ce qui rend la fonction continue, de plus on exige un degré de régularité plus grand, par exemple etre de classe C^2. Enfin, on fixe des conditions aux bornes de la réunion des intervalles, par exemple avoir certaines dérivées nulles.

Par exemple supposons qu'on se donne n intervalles, donc $n+1$ points $x_0, ..., x_n$, on se fixe une régularité C^{d-1}. Ceci entraine $(n-1)d$ conditions de recollement, on y ajoute $n+1$ conditions de valeur en $x_0, ..., x_n$, on a donc $nd+1$ conditions, la borne sur le degré des polynomes doit donc etre d (ou plus, mais d suffit) ce qui donne $n(d+1)$ degrés de liberté, on peut donc ajouter $d-1$ conditions, par exemple pour les splines naturelles, on impose que les dérivées d'ordre $d/2$ à $d-1$ soient nulles en x_0 et x_n (si d est pair, on commence à la dérivée $d/2+1$-ième nulle en x_n).

Pour trouver les polynomes, on doit donc résoudre un grand système linéaire. Une méthode permettant de diminuer la taille du système linéaire à résoudre dans le cas des splines naturelles consiste à se fixer n inconnues $z_0, .., z_{n-1}$ représentant les

dérivées d-ième de la spline f en x_0 sur $[x_0, x_1]$ à x_{n-1} sur $[x_{n-1}, x_n]$, et $(d-1)/2$ inconnues f_j, représentant la valeur de la dérivée de f en x_0 pour j variant de 1 à $(d-1)/2$. On peut alors écrire le polynome sur l'intervalle $[x_0, x_1]$ car on connait son développement de Taylor en x_0. On effectue un changement d'origine (par application répétée de Horner) en x_1. On obtient alors le polynome sur $[x_1, x_2]$ en remplaçant uniquement la dérivée d-ième par z_1. On continue ainsi jusqu'en x_{n-1}. Le système s'obtient en calculant la valeur du polynome en $x_0, ..., x_n$ et la nullité des dérivées d'ordre $(d-1)/2$ à $d/2$ en x_n. On résoud le système et on remplace pour avoir les valeurs numériques des coefficients du polynome.

13 La moyenne arithmético-géométrique.

La moyenne arithmético-géométrique est un processus itératif qui converge très rapidement et est très utile pour calculer les fonctions transcendantes réciproques en multi-précision. On peut alors trouver les fonctions transcendantes directes par application de la méthode de Newton.

13.1 Définition et convergence

Soient a et b deux réels positifs, on définit les 2 suites

$$u_0 = a, v_0 = b, \quad u_{n+1} = \frac{u_n + v_n}{2}, v_{n+1} = \sqrt{u_n v_n} \tag{42}$$

On va montrer que ces 2 suites sont adjacentes et convergent donc vers une limite commune notée $M(a, b)$ et il se trouve que la convergence est très rapide, en raison de l'identité :

$$u_{n+1} - v_{n+1} = \frac{1}{2}(\sqrt{u_n} - \sqrt{v_n})^2 = \frac{1}{2(\sqrt{u_n} + \sqrt{v_n})^2}(u_n - v_n)^2 \tag{43}$$

la convergence est quadratique.

On suppose dans la suite que $a \geq b$ sans changer la généralité puisque échanger a et b ne change pas la valeur de u_n et v_n pour $n > 0$. On a alors $u_n \geq v_n$ (d'après (43) pour $n > 0$) et $u_{n+1} \leq u_n$ car

$$u_{n+1} - u_n = \frac{1}{2}(v_n - u_n) \leq 0$$

et $v_{n+1} = \sqrt{u_n v_n} \geq \sqrt{v_n v_n} = v_n$. Donc (u_n) est décroissante minorée (par v_0), (v_n) est croissante majorée (par u_0), ces 2 suites sont convergentes et comme $u_{n+1} = \frac{u_n + v_n}{2}$, elles convergent vers la même limite l qui dépend de a et b et que l'on note $M(a, b)$. On remarque aussi que $M(a, b) = bM(a/b, 1) = aM(1, b/a)$.

Précisons maintenant la vitesse de convergence lorsque $a \geq b > 0$. On va commencer par estimer le nombre d'itérations nécessaires pour que u_n et v_n soient du même ordre de grandeur. Pour cela, on utilise la majoration

$$\ln(u_{n+1}) - \ln(v_{n+1}) \leq \ln(u_n) - \ln(v_{n+1}) = \frac{1}{2}(\ln(u_n) - \ln(v_n))$$

donc

$$\ln \frac{u_n}{v_n} = \ln(u_n) - \ln(v_n) \leq \frac{1}{2^n}(\ln(a) - \ln(b)) = \frac{1}{2^n} \ln \frac{a}{b}$$

Donc si $n \geq \frac{\ln(\ln(a/b)/m)}{\ln(2)}$ alors $\ln \frac{u_n}{v_n} \leq m$ (par exemple, on peut prendre $m = 0.1$ pour avoir $u_n/v_n \in [1, e^{0.1}]$). Le nombre minimum d'itérations n_0 est proportionnel au log du log du rapport a/b. Ensuite on est ramené à étudier la convergence de la suite arithmético-géométrique de premiers termes $a = u_{n_0}$ et $b = v_{n_0}$ et même en tenant compte de $M(a, b) = aM(1, b/a)$ à $a = 1$ et $b = v_n/u_n$ donc $0 \leq a - b \leq 1 - e^{-0.1}$. Alors l'équation (43) entraine

$$u_{n+1} - v_{n+1} \leq \frac{1}{8}(u_n - v_n)^2$$

puis (par récurrence)

$$0 \leq u_n - v_n \leq \frac{1}{8^{2^n-1}}(a - b)^{2^n}$$

Donc comme $M(a, b)$ est compris entre v_n et u_n, l'erreur relative sur la limite commune est inférieure à une précision donnée ϵ au bout d'un nombre d'itérations proportionnel au $\ln(\ln(1/\epsilon))$.

Typiquement dans la suite, on souhaitera calculer $M(1, b)$ avec b de l'ordre de 2^{-n} en déterminant n chiffres significatifs, il faudra alors $O(\ln(n))$ itérations pour se ramener à $M(1, b)$ avec $b \in [e^{-0.1}, 1]$ puis $O(\ln(n))$ itérations pour avoir la limite avec n chiffres significatifs.

Le cas complexe

On suppose maintenant que $a, b \in \mathbb{C}$ avec $\Re(a) > 0, \Re(b) > 0$. On va voir que la suite arithmético-géométrique converge encore.

Étude de l'argument

On voit aisément (par récurrence) que $\Re(u_n) > 0$; de plus $\Re(v_n) > 0$ car par définition de la racine carrée $\Re(v_n) \geq 0$ et est de plus non nul car le produit de deux complexes d'arguments dans $]-\pi/2, \pi/2[$ ne peut pas être un réel négatif. On en déduit que $\arg(u_{n+1}) = \arg(u_n + v_n)$ se trouve dans l'intervalle de bornes $\arg(u_n)$ et $\arg(v_n)$ et que $\arg(v_{n+1}) = \frac{1}{2}(\arg(u_n) + \arg(v_n))$ donc

$$|\arg(u_{n+1} - \arg(v_{n+1})| \leq \frac{1}{2}|\arg(u_n) - \arg(v_n)|$$

Après n itérations, on a

$$|\arg(u_n) - \arg(v_n)| \leq \frac{\pi}{2^n}$$

Après quelques itérations, u_n et v_n seront donc presque alignés. Faisons 4 itérations. On peut factoriser par exemple v_n et on est ramené à l'étude de la suite de termes initiaux $a = u_n/v_n$ d'argument $\arg(u_n) - \arg(v_n)$ petit (inférieur en valeur absolue à $\pi/16$) et $b = 1$. On suppose donc dans la suite que

$$|\arg(\frac{u_n}{v_n})| \leq \frac{\pi/16}{2^n}$$

Étude du module

On a :

$$\frac{u_{n+1}}{v_{n+1}} = \frac{1}{2}\left(\sqrt{\frac{u_n}{v_n}} + \frac{1}{\sqrt{\frac{u_n}{v_n}}}\right)$$

Posons $\frac{u_n}{v_n} = \rho_n e^{i\theta_n}$, on a :

$$\begin{aligned}
|\frac{u_{n+1}}{v_{n+1}}| &= \frac{1}{2}\left|\sqrt{\rho_n}e^{i\theta_n/2} + \frac{1}{\sqrt{\rho_n}}e^{-i\theta_n/2}\right| \\
&= \frac{1}{2}\left|(\sqrt{\rho_n} + \frac{1}{\sqrt{\rho_n}})\cos\frac{\theta_n}{2} + i(\sqrt{\rho_n} - \frac{1}{\sqrt{\rho_n}})\sin\frac{\theta_n}{2}\right| \\
&= \frac{1}{2}\sqrt{(\sqrt{\rho_n} + \frac{1}{\sqrt{\rho_n}})^2\cos^2\frac{\theta_n}{2} + (\sqrt{\rho_n} - \frac{1}{\sqrt{\rho_n}})^2\sin^2\frac{\theta_n}{2}} \\
&= \frac{1}{2}\sqrt{\rho_n + \frac{1}{\rho_n} + 2\cos\theta_n}
\end{aligned}$$

Si ρ désigne le max de ρ_n et $1/\rho_n$, on a alors la majoration

$$|\frac{u_{n+1}}{v_{n+1}}| \leq \frac{1}{2}\sqrt{\rho + \rho + 2\rho} = \sqrt{\rho}$$

donc en prenant les logarithmes

$$\ln\rho_{n+1} \leq \frac{1}{2}\ln\rho = \frac{1}{2}|\ln\rho_n| \tag{44}$$

On rappelle qu'on a la majoration

$$|\arg(\frac{u_n}{v_n})| = |\theta_n| \leq \frac{\pi/16}{2^n} \leq \frac{1}{2^{n+1}}$$

qui va nous donner la minoration de ρ_{n+1}

$$\begin{aligned}
\rho_{n+1} = |\frac{u_{n+1}}{v_{n+1}}| &= \frac{1}{2}\sqrt{\rho_n + \frac{1}{\rho_n} + 2 - 2(1-\cos\theta_n)} \\
&= \frac{1}{2}\sqrt{\rho_n + \frac{1}{\rho_n} + 2 - 4\sin^2(\frac{\theta_n}{2})} \\
&\geq \frac{1}{2}\sqrt{\rho_n + \frac{1}{\rho_n} + 2 - \theta_n^2} \\
&\geq \frac{1}{2}\sqrt{\rho_n + \frac{1}{\rho_n} + 2} \times \sqrt{1 - \frac{\theta_n^2}{\rho_n + \frac{1}{\rho_n} + 2}} \\
&\geq \frac{1}{2}\sqrt{\frac{1}{\rho} + \frac{1}{\rho} + 2\frac{1}{\rho}} \times \sqrt{1 - \frac{\theta_n^2}{4}} \\
&\geq \frac{1}{\sqrt{\rho}}\sqrt{1 - \frac{\theta_n^2}{4}} \\
&\geq \frac{1}{\sqrt{\rho}}\sqrt{1 - \frac{1}{4\times 2^{2n+2}}}
\end{aligned}$$

en prenant les log et en minorant $\ln(1-x)$ par $-2x$

$$\ln\rho_{n+1} \geq \frac{1}{2}(-|\ln\rho_n| + \ln(1 - \frac{1}{4\times 2^{2n+2}})) \geq -\frac{1}{2}(|\ln\rho_n| + \frac{1}{2^{2n+3}})$$

Finalement avec (44)

$$|\ln \rho_{n+1}| \leq \frac{1}{2}(|\ln \rho_n| + \frac{1}{2^{2n+3}})$$

On en déduit

$$|\ln \rho_n| \leq \frac{1}{2^n}\ln \rho_0 + \frac{1}{2^{n+3}} + ... + \frac{1}{2^{2n+1}} + \frac{1}{2^{2n+2}} = \frac{1}{2^n}\ln \rho_0 + \frac{1}{2^{n+2}}$$

La convergence du $\ln(u_n/v_n)$ vers 0 est donc géométrique, donc u_n et v_n convergent quadratiquement.

13.2 Lien avec les intégrales elliptiques

Le calcul de la limite commune des suites u_n et v_n en fonction de a et b n'est pas trivial au premier abord. Il est relié aux intégrales elliptiques, plus précisément on peut construire une intégrale dépendant de deux paramètres a et b et qui est invariante par la transformation $u_n, v_n \to u_{n+1}, v_{n+1}$ (42)

$$I(a,b) = \int_{-\infty}^{+\infty} \frac{dt}{\sqrt{(a^2+t^2)(b^2+t^2)}}$$

On a en effet

$$I(\frac{a+b}{2}, \sqrt{ab}) = \int_{-\infty}^{+\infty} \frac{du}{\sqrt{((\frac{a+b}{2})^2+u^2)(ab+u^2)}}$$

On pose alors

$$u = \frac{1}{2}(t - \frac{ab}{t}), \quad t > 0$$

où $t \to u$ est une bijection croissante de $t \in]0, +\infty[$ vers $u \in]-\infty, +\infty[$, donc

$$\begin{aligned} I(\frac{a+b}{2}, \sqrt{ab}) &= \int_0^{+\infty} \frac{dt/2(1+ab/t^2)}{\sqrt{((\frac{a+b}{2})^2 + 1/4(t-ab/t)^2)(ab+1/4(t-ab/t)^2)}} \\ &= 2\int_0^{+\infty} \frac{dt}{\sqrt{(a^2+t^2)(b^2+t^2)}} = I(a,b) \end{aligned}$$

On note au passage que I est définie si $a, b \in \mathbb{C}$ vérifient $\Re(a) > 0, \Re(b) > 0$, on peut montrer que la relation ci-dessus s'étend (par holomorphie).

Lorsque $a = b = l$ (par exemple lorsqu'on est à la limite), le calcul de $I(l,l)$ est explicite

$$I(l,l) = \int_{-\infty}^{+\infty} \frac{dt}{(l^2+t^2)} = \frac{\pi}{l}$$

donc

$$I(a,b) = I(M(a,b), M(a,b)) = \frac{\pi}{M(a,b)}$$

On peut transformer $I(a,b)$ en posant $t = bu$

$$I(a,b) = 2\int_0^{+\infty} \frac{du}{\sqrt{(a^2+b^2u^2)(1+u^2)}} = \frac{2}{a}\int_0^{+\infty} \frac{du}{\sqrt{(1+(b/a)^2u^2)(1+u^2)}}$$

Puis en posant $u = \tan(x)$ ($du = (1+u^2)dx$)

$$I(a,b) = \frac{2}{a}\int_0^{\frac{\pi}{2}} \sqrt{\frac{1+\tan(x)^2}{1+(b/a)^2\tan(x)^2}}\, dx$$

et enfin en posant $\tan^2(x) = \frac{\sin(x)^2}{1-\sin(x)^2}$

$$I(a,b) = \frac{2}{a}\int_0^{\frac{\pi}{2}} \sqrt{\frac{1}{1-(1-\frac{b^2}{a^2})\sin(x)^2}}\, dx$$

Si on définit pour $m < 1$

$$K(m) = \int_0^{\frac{\pi}{2}} \frac{dx}{\sqrt{1-m\sin(x)^2}}$$

alors on peut calculer K en fonction de I, en posant $m = 1 - b^2/a^2$ soit $b^2/a^2 = 1-m$

$$K(m) = \frac{a}{2}I(a, a\sqrt{1-m}) = \frac{a}{2}\frac{\pi}{M(a, a\sqrt{1-m})} = \frac{\pi}{2M(1,\sqrt{1-m})}$$

d'où l'on déduit la valeur de l'intégrale elliptique en fonction de la moyenne arithmético-géométrique :

$$K(m) = \int_0^{\frac{\pi}{2}} \frac{dx}{\sqrt{1-m\sin(x)^2}} = \frac{\pi}{2M(1,\sqrt{1-m})} \qquad (45)$$

Dans l'autre sens, pour x et y positifs

$$K((\frac{x-y}{x+y})^2) = \frac{\pi}{2M(1,\sqrt{1-(\frac{x-y}{x+y})^2})} = \frac{\pi}{2M(1,\frac{2}{x+y}\sqrt{xy})} = \frac{\pi}{2\frac{2}{x+y}M(\frac{x+y}{2},\sqrt{xy})} = \frac{\pi}{4}\frac{x+y}{M(x,y)}$$

et finalement

$$M(x,y) = \frac{\pi}{4}\frac{x+y}{K\left((\frac{x-y}{x+y}\right)^2)}$$

13.3 Application : calcul efficace du logarithme.

On peut utiliser la moyenne arithmético-géométrique pour calculer le logarithme efficacement, pour cela on cherche le développement asymptotique de $K(m)$ lorsque m tend vers 1. Plus précisément, on va poser $1-m = k^2$ avec $k \in]0,1]$, donc

$$K(m) = \int_0^{\frac{\pi}{2}} \frac{dx}{\sqrt{1-(1-k^2)\sin(x)^2}} = \int_0^{\frac{\pi}{2}} \frac{dy}{\sqrt{1-(1-k^2)\cos(y)^2}}$$

en posant $y = \pi/2 - x$, et

$$K(m) = \int_0^{\frac{\pi}{2}} \frac{dy}{\sqrt{\sin(y)^2 + k^2\cos(y)^2}}$$

la singularité de l'intégrale pour k proche de 0 apparait lorsque y est proche de 0. Si on effectue un développement de Taylor en $y=0$, on trouve

$$\sin(y)^2+k^2\cos(y)^2=k^2+(1-k^2)y^2+O(y^4)$$

Il est donc naturel de comparer $K(m)$ à l'intégrale

$$J=\int_0^{\frac{\pi}{2}}\frac{dy}{\sqrt{k^2+(1-k^2)y^2}}$$

qui se calcule en faisant par exemple le changement de variables

$$y=\frac{k}{\sqrt{1-k^2}}\sinh(t)$$

ou directement avec Xcas,

```
supposons(k>0 && k<1);
J:=int(1/sqrt(k^2+(1-k^2)*y^2),y,0,pi/2)
```

qui donne après réécriture :

$$J=\frac{1}{\sqrt{1-k^2}}\left(\ln\left(\frac{\pi}{k}\right)+\ln\left(\frac{1}{2}\left(\sqrt{1-k^2+4\frac{k^2}{\pi^2}}+\sqrt{1-k^2}\right)\right)\right) \quad (46)$$

et on peut calculer le développement asymptotique de J en 0

```
series(J,k=0,5,1)
```

qui renvoie :

$$J=\ln\left(\frac{\pi}{k}\right)+O(\left(\frac{-1}{\ln(k)}\right)^5)$$

on peut alors préciser ce développement par

```
series(J+ln(k)-ln(pi),k=0,5,1)
```

qui renvoie (après simplifications et où la notation $\tilde{O}$ peut contenir des logarithmes)

$$\left(\frac{1}{\pi^2}+\frac{\ln(\pi)-\ln(k)-1}{2}\right)k^2+\tilde{O}(k^4)$$

donc

$$J=-\ln(k)+\ln(\pi)+\left(\frac{1}{\pi^2}+\frac{\ln(\pi)-\ln(k)-1}{2}\right)k^2+\tilde{O}(k^4) \quad (47)$$

Examinons maintenant $K-J$, il n'a plus de singularité en $y=0$, et il admet une limite lorsque $k\to 0$, obtenue en remplacant k par 0

$$(K-J)_{|k=0}=\int_0^{\frac{\pi}{2}}\left(\frac{1}{\sin(y)}-\frac{1}{y}\right)\,dy=\left[\ln\left(\tan\left(\frac{y}{2}\right)\right)-\ln(y)\right]_0^{\frac{\pi}{2}}=\ln(\frac{4}{\pi})$$

D'où pour K

$$K_{k\to 0}=\ln\left(\frac{4}{k}\right)+O(\left(\frac{-1}{\ln(k)}\right)^5)$$

Pour préciser la partie du développement de K en puissances de k, nous allons majorer $K - J - \ln(4/\pi)$, puis $J - \ln(\pi/k)$. Posons

$$A = \sin(y)^2 + k^2\cos(y)^2, \quad B = y^2 + (1-y^2)k^2$$

Majoration de $K - J - \ln(4/\pi)$
L'intégrand de la différence $K - J - \ln(\frac{4}{\pi})$ est

$$\begin{aligned}
\frac{1}{\sqrt{A}} - \frac{1}{\sqrt{B}} - \left(\frac{1}{\sin(y)} - \frac{1}{y}\right) &= \frac{\sqrt{B}-\sqrt{A}}{\sqrt{A}\sqrt{B}} - \frac{y-\sin(y)}{y\sin(y)} && (48)\\
&= \frac{B-A}{\sqrt{A}\sqrt{B}(\sqrt{A}+\sqrt{B})} - \frac{y-\sin(y)}{y\sin(y)} && (49)\\
&= \frac{(y^2-\sin(y)^2)(1-k^2)}{\sqrt{A}\sqrt{B}(\sqrt{A}+\sqrt{B})} - \frac{y-\sin(y)}{y\sin(y)} && (50)
\end{aligned}$$

Soit

$$K-J-\ln(\frac{4}{\pi}) = \int_0^{\frac{\pi}{2}} \frac{(y-\sin(y))[(1-k^2)y\sin(y)(y+\sin(y)) - \sqrt{AB}(\sqrt{A}+\sqrt{B})]}{\sqrt{A}\sqrt{B}(\sqrt{A}+\sqrt{B})y\sin(y)} \quad (51)$$

On décompose l'intégrale en 2 parties $[0,k]$ et $[k,\pi/2]$. Sur $[0,k]$ on utilise (49), on majore chaque terme séparément et on minore A et B par

$$A = k^2 + (1-k^2)\sin(y)^2 \geq k^2, \quad B = k^2 + (1-k^2)y^2 \geq k^2$$

Donc

$$\begin{aligned}
|\int_0^k| &\leq \int_0^k \frac{|B-A|}{2k^3}\,dy + \int_0^k (\frac{1}{\sin(y)} - \frac{1}{y})\,dy\\
&\leq \int_0^k \frac{y^2-\sin(y)^2}{2k^3}\,dy + \ln(\tan(\frac{k}{2})) - \ln(\frac{k}{2})\\
&\leq \frac{\frac{1}{3}k^3 + \frac{-1}{2}k + \frac{1}{4}\sin(2k)}{2k^3} + \ln(\sin(\frac{k}{2})) - \ln(\frac{k}{2}) - \ln(\cos(\frac{k}{2}))\\
&\leq \frac{\frac{1}{3}k^3 + \frac{-1}{2}k + \frac{1}{4}(2k - \frac{8k^3}{6} + \frac{32k^5}{5!}}{2k^3} - \ln(\cos(\frac{k}{2}))\\
&\leq \frac{k^2}{30} - \ln(1 - \frac{1}{2!}\left(\frac{k}{2}\right)^2)\\
&\leq \frac{k^2}{30} + \frac{k^2}{4}
\end{aligned}$$

Sur $[k,\pi/2]$, on utilise (51) et on minore A et B par

$$A = \sin(y)^2 + k^2\cos(y)^2 \geq \sin(y)^2, \quad B = y^2 + (1-y^2)k^2 \geq y^2$$

on obtient

$$|\int_k^{\frac{\pi}{2}}| \leq \int_k^{\frac{\pi}{2}} \frac{(y-\sin(y))|C|}{y\sin(y)(y+\sin(y))},$$

où :

$$\begin{aligned}
C &= (1-k^2)y\sin(y)(y+\sin(y)) - A\sqrt{B} + B\sqrt{A}\\
&= -A(\sqrt{B}-y) - B(\sqrt{A}-\sin(y)) - Ay - B\sin(y) + (1-k^2)y\sin(y)(y+\sin(y))\\
&= -A(\sqrt{B}-y) - B(\sqrt{A}-\sin(y)) - k^2(y+\sin(y))
\end{aligned}$$

Donc

$$\begin{aligned} |C| &\leq A(\sqrt{B}-y)+B(\sqrt{A}-\sin(y))+k^2(y+\sin(y)) \\ &\leq A\frac{B-y^2}{\sqrt{B}+y}+B\frac{A-\sin(y)^2}{\sqrt{A}+\sin(y)}+k^2(y+\sin(y)) \\ &\leq A\frac{k^2}{2y}+B\frac{k^2}{2\sin(y)}+k^2(y+\sin(y)) \end{aligned}$$

et

$$|\int_k^{\frac{\pi}{2}}| \leq \int_k^{\frac{\pi}{2}} \frac{(y-\sin(y))k^2(\frac{A}{2y}+\frac{B}{2\sin(y)}+(y+\sin(y)))}{y\sin(y)(y+\sin(y))}$$

On peut majorer $y-\sin(y)\leq y^3/6$, donc

$$|\int_k^{\frac{\pi}{2}}| \leq \frac{k^2}{6}\int_k^{\frac{\pi}{2}} \frac{Ay}{2\sin(y)(\sin(y)+y)}+\frac{By^2}{\sin(y)^2(\sin(y)+y)}+\frac{y^2}{\sin(y)}$$

On majore enfin A et B par 1,

$$|\int_k^{\frac{\pi}{2}}| \leq \frac{k^2}{6}\int_k^{\frac{\pi}{2}} \frac{y}{2\sin(y)^2}+\frac{y^2}{\sin(y)}$$

Le premier morceau se calcule par intégration par parties

$$\begin{aligned} \frac{k^2}{6}\int_k^{\frac{\pi}{2}} \frac{y}{2\sin(y)^2} &= \frac{k^2}{6}\left([-\frac{y}{\tan(y)}]_k^{\pi/2}+\int_k^{\frac{\pi}{2}}\frac{1}{\tan(y)}\right) \\ &= \frac{k^2}{6}\left(\frac{k}{\tan(k)}+[\ln(\sin(y))]_k^{\frac{\pi}{2}}\right) \\ &= \frac{k^2}{6}\left(\frac{k}{\tan(k)}-\ln(\sin(k))\right) \\ &\leq \frac{k^2}{6}(1-\ln(k)) \end{aligned}$$

Le deuxième morceau se majore en minorant $\sin(y)\geq(2y)/\pi$

$$\frac{k^2}{6}\int_k^{\frac{\pi}{2}}\frac{y^2}{\sin(y)} \leq \frac{k^2}{6}\int_0^{\frac{\pi}{2}}\frac{\pi}{2}y = \frac{k^2\pi^3}{96}$$

Finalement

$$|K-J-\ln(\frac{4}{\pi})| \leq k^2\left(-\frac{1}{6}\ln(k)+\frac{\pi^3}{96}+\frac{1}{6}+\frac{1}{30}+\frac{1}{4}\right)$$

où J est donné en (46).

Majoration de $J-ln(\pi/k)$

On a

$$|J-\ln\left(\frac{\pi}{k}\right)| = \left|(\frac{1}{\sqrt{1-k^2}}-1)\ln\left(\frac{\pi}{k}\right)+\frac{1}{\sqrt{1-k^2}}\ln\left(\frac{1}{2}\left(\sqrt{1-k^2+4\frac{k^2}{\pi^2}}+\sqrt{1-k^2}\right)\right)\right|$$

et on va majorer la valeur absolue de chaque terme de la somme. Pour $k \leq 1/2$, on a

$$\frac{1}{\sqrt{1-k^2}} - 1 = \frac{k^2}{\sqrt{1-k^2}+1-k^2} \leq \frac{k^2}{3/4+\sqrt{3}/2}$$

Pour le second terme, on majore le facteur $\frac{1}{\sqrt{1-k^2}}$ par $\frac{2}{\sqrt{3}}$, l'argument du logarithme est inférieur à 1 et supérieur à

$$\frac{1}{2}(1-\frac{k^2}{2}+1-\frac{k^2(1-\frac{4}{\pi^2})}{2}) = 1-k^2(1-\frac{1}{\pi^2}) > 1-k^2$$

donc le logarithme en valeur absolue est inférieur à

$$2k^2$$

donc, pour $k \leq 1/2$,

$$|J - \ln\left(\frac{\pi}{k}\right)| \leq \frac{k^2}{3/4+\sqrt{3}/2}\ln\left(\frac{\pi}{k}\right) + k^2\frac{4}{\sqrt{3}}$$

Finalement, pour $k < 1/2$

$$|K - \ln\left(\frac{4}{k}\right)| \leq k^2\left(\frac{\ln\pi}{3/4+\sqrt{3}/2} + \frac{4}{\sqrt{3}} + \frac{\pi^3}{96} + \frac{9}{20} - (\frac{1}{3/4+\sqrt{3}/2}+\frac{1}{6})\ln(k)\right) \tag{52}$$

que l'on peut réécrire

$$|\frac{\pi}{2M(1,k)} - \ln\left(\frac{4}{k}\right)| \leq k^2(3.8 - 0.8\ln(k)) \tag{53}$$

La formule (53) permet de calculer le logarithme d'un réel positif avec (presque) n bits lorsque $k \leq 2^{-n/2}$ (ce à quoi on peut toujours se ramener en calculant le logarithme d'une puissance 2^m-ième de x ou le logarithme de $2^m x$, en calculant au préalable $\ln(2)$). Par exemple, prenons $k = 2^{-27}$, on trouve (en 8 itérations) $M(1, 2^-27) = M_1 = 0.0781441403763$. On a, avec une erreur inférieure à $19 \times 2^{-54} = 1.1 \times 10^{-15}$

$$M(1,2^-27) = M_1 = \frac{\pi}{2\ln(2^{29})} = \frac{\pi}{58\ln(2)},$$

On peut donc déduire une valeur approchée de π si on connait la valeur approchée de $\ln(2)$ et réciproquement. Si on veut calculer les deux simultanément, comme les relations entre $\ln$ et π seront des équations homogènes, on est obligé d'introduire une autre relation. Par exemple pour calculer une valeur approchée de π on calcule la différence $\ln(2^{29}+1) - \ln(2^{29})$ dont on connait le développement au premier ordre, et on applique la formule de la moyenne arithmético-géométrique. Il faut faire attention à la perte de précision lorsqu'on fait la différence des deux logarithmes qui sont très proches, ainsi on va perdre une trentaine de bits, il faut grosso modo calculer les moyennes arithmético-géométrique avec 2 fois plus de chiffres significatifs.

L'intérêt de cet algorithme apparait lorsqu'on veut calculer le logarithme avec beaucoup de précision, en raison de la convergence quadratique de la moyenne

arithmético-géométrique (qui est nettement meilleure que la convergence linéaire pour les développements en série, ou logarithmiquement meilleure pour l'exponentielle), par contre elle n'est pas performante si on ne veut qu'une dizaine de chiffres significatifs. On peut alors calculer les autres fonctions transcendantes usuelles, telle l'exponentielle, à partir du logarithme, ou les fonctions trigonométriques inverses (en utilisant des complexes) et directes.

On trouvera dans Brent-Zimmermann quelques considérations permettant d'améliorer les constantes dans les temps de calcul par rapport à cette méthode (cela nécessite d'introduire des fonctions spéciales θ) et d'autres formules pour calculer π.

On peut ensuite à partir du logarithme, calculer l'exponentielle en utilisant la méthode de Newton, rappelée ci-dessous.

13.4 La méthode de Newton.

La méthode de Newton est une méthode de résolution de l'équation $f(x) = 0$, attention à la différence avec le théorème du point fixe qui permet de résoudre numériquement $f(x) = x$. Si x_0 est proche de la racine r on peut faire un développement de Taylor à l'ordre 1 de la fonction f en x_0 :

$$f(x) = f(x_0) + (x - x_0)f'(x_0) + O((x - x_0)^2)$$

Pour trouver une valeur approchée de r, on ne garde que la partie linéaire du développement, on résout :

$$f(r) = 0 \approx f(x_0) + (r - x_0)f'(x_0)$$

donc (si $f'(x_0) \neq 0$) :

$$r \approx x_0 - \frac{f(x_0)}{f'(x_0)}$$

Graphiquement, cela revient à tracer la tangente à la courbe représentative de f et à chercher où elle coupe l'axe des x. On considère donc la suite récurrente définie par une valeur u_0 proche de la racine et par la relation :

$$u_{n+1} = u_n - \frac{f(u_n)}{f'(u_n)}$$

Il y a deux théorèmes importants, l'un d'eux prouve que si u_0 est "assez proche" de r alors la suite u_n converge vers r, malheureusement il est difficile de savoir en pratique si on est "assez proche" de u_0 pour que ce théorème s'applique. Le second théorème donne un critère pratique facile à vérifier qui assure la convergence, il utilise les propriétés de convexité de la fonction.

Théorème 14 *Soit f une fonction de classe C^2 (2 fois continument dérivable) sur un intervalle fermé I. Soit r une racine simple de f située à l'intérieur de I (telle que $f(r) = 0$ et $f'(r) \neq 0$). Alors il existe $\varepsilon > 0$ tel que la suite définie par*

$$u_{n+1} = u_n - \frac{f(u_n)}{f'(u_n)}, \quad |u_0 - r| \leq \varepsilon$$

converge vers r.

Si on a $|f''| \leq M$ et $|1/f'| \leq m$ sur un intervalle $[r - \eta, r + \eta]$ contenu dans I, alors on peut prendre tout réel $\varepsilon > 0$ tel que $\varepsilon < 2/(mM)$ et $\varepsilon \leq \eta$.

Démonstration : on a

$$u_{n+1} - r = u_n - r - \frac{f(u_n)}{f'(u_n)} = \frac{(u_n - r)f'(u_n) - f(u_n)}{f'(u_n)}$$

En appliquant un développement de Taylor de f en u_n à l'ordre 2, on obtient pour un réel θ situé entre r et u_n :

$$0 = f(r) = f(u_n) + (r - u_n)f'(u_n) + (r - u_n)^2 \frac{f''(\theta)}{2}$$

donc :

$$(u_n - r)f'(u_n) - f(u_n) = (u_n - r)^2 \frac{f''(\theta)}{2}$$

d'où :

$$|u_{n+1} - r| \leq |u_n - r|^2 \frac{1}{|f'(u_n)|} \frac{|f''(\theta)|}{2}$$

On commence par choisir un intervalle $[r - \varepsilon, r + \varepsilon]$ contenant strictement r et tel que $|f''| < M$ et $|1/f'| < m$ sur $[r - \varepsilon, r + \varepsilon]$ (c'est toujours possible car f'' et $1/f'$ sont continues au voisinage de r puisque $f'(r) \neq 0$). Si u_n est dans cet intervalle, alors θ aussi donc

$$|u_{n+1} - r| \leq |u_n - r|^2 \frac{Mm}{2} \leq \frac{|u_n - r|Mm}{2} |u_n - r|,$$

On a $|u_n - r| \leq \varepsilon$, on diminue si nécessaire ε pour avoir $\varepsilon < 2/(Mm)$, on a alors :

$$|u_{n+1} - r| \leq k|u_n - r|, \quad k = \frac{\varepsilon Mm}{2} < 1$$

donc d'une part u_{n+1} est encore dans l'intervalle $[r - \varepsilon, r + \varepsilon]$ ce qui permettra de refaire le même raisonnement au rang suivant, et d'autre part on a une convergence au moins géométrique vers r. En fait la convergence est bien meilleure lorsqu'on est proche de r grace au carré dans $|u_n - r|^2$, plus précisément, on montre par récurrence que

$$|u_n - r| \leq |u_0 - r|^{2^n} \left(\frac{Mm}{2}\right)^{2^n - 1}$$

il faut donc un nombre d'itérations proportionnel à $\ln(n)$ pour atteindre une précision donnée.

Remarque : ce théorème se généralise sur $\mathbb{C}$ et même sur $\mathbb{R}^n$.

Exemple : pour calculer $\sqrt{2}$, on écrit l'équation $x^2 - 2 = 0$ qui a $\sqrt{2}$ comme racine simple sur $I = [1/2, 2]$, on obtient la suite récurrente

$$u_{n+1} = u_n - \frac{u_n^2 - 2}{2u_n}$$

Si on prend $\eta = 1/2$, on a $f' = 2x$ et $f'' = 2$ donc on peut prendre $M = 2$ et $m = 1$ car $|1/f'| \leq 1$ sur $[\sqrt{2} - 1/2, \sqrt{2} + 1/2]$. On a $2/(mM) = 1$, on peut donc prendre $\varepsilon = 1/2$, la suite convergera pour tout $u_0 \in [\sqrt{2} - 1/2, \sqrt{2} + 1/2]$.

Plus générallement, on peut calculer une racine k-ième d'un réel a en résolvant $f(x) = x^k - a$ par la méthode de Newton.

L'inconvénient de ce théorème est qu'il est difficile de savoir si la valeur de départ qu'on a choisie se trouve suffisamment près d'une racine pour que la suite converge. Pour illustrer le phénomène, on peut par exemple colorer les points du plan complexe en $n+1$ couleurs selon que la suite définie par la méthode de Newton converge vers l'une des n racines d'un polynôme de degré n fixé au bout de par exemple 50 itérations (la $n+1$-ième couleur servant aux origines de suite qui ne semblent pas converger).

Passons maintenant à un critère très utile en pratique :

Définition 1 *(convexité)*
Une fonction f continument dérivable sur un intervalle I de $\mathbb{R}$ est dite convexe si son graphe est au-dessus de la tangente en tout point de I.

Il existe un critère simple permettant de savoir si une fonction de classe C^2 est convexe :

Théorème 15 *Si f est C^2 et $f'' \geq 0$ sur I alors f est convexe.*

Démonstration :
L'équation de la tangente au graphe en x_0 est

$$y = f(x_0) + f'(x_0)(x - x_0)$$

Soit

$$g(x) = f(x) - (f(x_0) + f'(x_0)(x - x_0))$$

on a :

$$g(x_0) = 0, \quad g'(x) = f'(x) - f'(x_0), \quad g'(x_0) = 0, \quad g'' = f'' \geq 0$$

donc g' est croissante, comme $g'(x_0) = 0$, g' est négative pour $x < x_0$ et positive pour $x > x_0$, donc g est décroissante pour $x < x_0$ et croissante pour $x > x_0$. On conclut alors que $g \geq 0$ puisque $g(x_0) = 0$. Donc f est bien au-dessus de sa tangente.

On arrive au deuxième théorème sur la méthode de Newton

Théorème 16 *Si $f(r) = 0, f'(r) > 0$ et si $f'' \geq 0$ sur $[r, b]$ alors pour tout $u_0 \in [r, b]$ la suite de la méthode de Newton*

$$u_{n+1} = u_n - \frac{f(u_n)}{f'(u_n)},$$

est définie, décroissante, minorée par r et converge vers r. De plus

$$0 \leq u_n - r \leq \frac{f(u_n)}{f'(r)}$$

Démonstration :
On a $f'' \geq 0$ donc si $f'(r) > 0$ alors $f' > 0$ sur $[r, b]$, f est donc strictement croissante sur $[r, b]$ on en déduit que $f > 0$ sur $]r, b]$ donc $u_{n+1} \leq u_n$. Comme la courbe représentative de f est au-dessus de la tangente, on a $u_{n+1} \geq r$ (car u_{n+1} est l'abscisse du point d'intersection de la tangente avec l'axe des x). La suite u_n

est donc décroissante minorée par r, donc convergente vers une limite $l \geq r$. À la limite, on a

$$l = l - \frac{f(l)}{f'(l)} \Rightarrow f(l) = 0$$

donc $l = r$ car $f > 0$ sur $]r, b]$.

Comme (u_n) est décroissante, on a bien $0 \leq u_n - r$, pour montrer l'autre inégalité, on applique le théorème des accroissements finis, il existe $\theta \in [r, u_n]$ tel que

$$f(u_n) - f(r) = (u_n - r)f'(\theta)$$

comme $f(r) = 0$, on a

$$u_n - r = \frac{f(u_n)}{f'(\theta)}$$

et la deuxième inégalité du théorème en découle parce que f' est croissante.

Variantes :

Il existe des variantes, par exemple si $f'(r) < 0$ et $f'' \geq 0$ sur $[a, r]$. Si $f'' \leq 0$, on considère $g = -f$.

Application :

On peut calculer la valeur approchée de la racine k-ième d'un réel $a > 0$ en appliquant ce deuxième théorème. En effet si $a > 0$, alors $x^k - a$ est 2 fois continument dérivable et de dérivée première kx^{k-1} et seconde $k(k-1)x^{k-2}$ strictement positives sur $\mathbb{R}^{+*}$ (car $k \geq 2$). Il suffit donc de prendre une valeur de départ u_0 plus grande que la racine k-ième, par exemple $1 + a/k$ (en effet $(1 + a/k)^k \geq 1 + ka/k = 1 + a$). En appliquant l'inégalité du théorème, on a :

$$0 \leq u_n - \sqrt[k]{a} \leq \frac{u_n^k - a}{k\sqrt[k]{a}^{k-1}} \leq \frac{u_n^k - a}{ka}\sqrt[k]{a} \leq \frac{u_n^k - a}{ka}(1 + \frac{a}{k})$$

Pour avoir une valeur approchée de $\sqrt[k]{a}$ à ε près, on peut donc choisir comme test d'arrêt

$$u_n^k - a \leq \frac{ka}{1 + \frac{a}{k}}\varepsilon$$

Par exemple pour $\sqrt{2}$, le test d'arrêt serait $u_n^2 - 2 \leq 2\varepsilon$.