Des idéaux principaux aux algorithmes

Jacques Faisant(*)

L'ensemble $\mathbb{Z}[i]$ des nombres complexes de la forme $a + i\cdot b$ où a et b appartiennent à $\mathbb{Z}$ est appelé l'ensemble des entiers de Gauss ; c'est un anneau intègre(1) dans lequel il est classique de « faire de l'arithmétique ». Pour débuter, voici quelques remarques.

I. Premières remarques concernant les entiers de Gauss

a. Les éléments inversibles de $\mathbb{Z}[i]$ sont 1, −1, i et −i. En effet, leurs inverses respectifs sont 1, −1, −i et i et il n'existe aucun autre élément inversible. On peut le voir en employant la norme : la norme d'un entier de Gauss z est $N(z) = z\cdot\overline{z}$. Ce n'est pas la norme de la géométrie mais on a quand même $N(z_1\cdot z_2) = N(z_1)\cdot N(z_2)$. Ainsi, si a, b, a' et b' sont des naturels et si $(a+i\cdot b)\cdot(a'+i\cdot b') = 1$, alors $N(a+i\cdot b)\cdot N(a'+i\cdot b') = (a^2+b^2)\cdot(a'^2+b'^2) = N(1) = 1$. Donc la norme d'un entier de Gauss inversible est nécessairement 1, qui est le seul naturel qui divise 1.

b. On dira que deux entiers de Gauss z et z' **sont associés** lorsque l'un d'eux est égal au produit de l'autre par un entier de Gauss **inversible**. On a alors $z' = u\cdot z$ et $z = u'\cdot z'$ où u et u' sont des éléments inverses l'un de l'autre de $\mathbb{Z}[i]$.

Il est utile de remarquer que tout entier de Gauss z a un élément associé situé dans le premier quadrant(2). Bien sûr, cela est vrai s'il se trouve que z appartient à ce premier quadrant. Si z est dans le troisième quadrant, alors son élément associé $-z$ convient. Si z est, respectivement, dans le deuxième quadrant ou le quatrième quadrant, alors les éléments $-i\cdot z$ et $i\cdot z$ conviennent respectivement. (Cela se justifie directement par l'étude des signes des parties réelles et imaginaires des éléments indiqués.)

c. Il est bien connu qu'on a $2 = (1 + i)\cdot(1 - i)$; on dit donc que $1 + i$ est un diviseur de 2 et que, par suite, 2 n'est pas premier ... dans $\mathbb{Z}[i]$! Pour éviter une confusion, on peut utiliser le mot **irréductible** plutôt que le mot premier pour les entiers de Gauss : un entier de Gauss z est irréductible si et seulement si toute égalité $z = z_1\cdot z_2$ où z_1 et z_2 sont des entiers de Gauss implique que z_1 ***ou bien*** z_2 est un entier de Gauss inversible. (Ni 0 ni les inversibles ne sont irréductibles.)

(*) jacques.faisant@math.unicaen.fr

(1) On dit qu'un anneau commutatif unitaire non nul est intègre lorsqu'il n'a aucun diviseur de 0.

(2) Il est nécessaire de définir précisément les frontières des quatre quadrants. Nous dirons ici, pour cela, que 0 et 1 sont dans le premier quadrant, i dans le deuxième, −1 dans le troisième et −i dans le quatrième.

Soit z un élément non nul de $\mathbb{Z}[i]$. Considérons l'ensemble E des normes des entiers de Gauss qui divisent z. Si z est irréductible, alors E = {1 ; N(z)}. Sinon E − {1} est une partie non vide de $\mathbb{N}$, non égale à {N(z)}. Elle a donc un plus petit élément et celui-ci est la norme d'un entier de Gauss qui divise z sans lui être associé et qui ne peut pas avoir de diviseurs autres que ses propres éléments associés et les entiers de Gauss inversibles : c'est un irréductible qui divise z.

On peut, de plus, remarquer qu'un entier de Gauss non irréductible z possède, parmi ses diviseurs non inversibles, au moins un entier de Gauss de norme inférieure ou égale à $\sqrt{N(z)}$: si $z = z_1 \cdot z_2$, d'où $N(z) = N(z_1) \cdot N(z_2)$, avec $N(z_1) > 1$ et $N(z_2) > 1$, il suffit de prendre celui des deux diviseurs z_1 et z_2 qui a la plus petite norme, ou n'importe lequel si les deux normes sont égales. Ainsi, on peut dire qu'il existe, parmi les diviseurs de z, au moins un entier de Gauss *irréductible*, de norme inférieure ou égale à $\sqrt{N(z)}$.

d. Quels sont les entiers de Gauss irréductibles ?

Définissons deux conditions concernant un entier de Gauss z :

C_1 : la norme de z est un entier naturel premier.

C_2 : z **est associé** à un entier naturel premier congru à 3 modulo 4.

On peut montrer que z est irréductible si et seulement si il vérifie C_1 ou C_2 (Voir [2]).

e. Dans $\mathbb{Z}[i]$, on peut définir une adaptation de la **division euclidienne** de $\mathbb{Z}$: étant donnés des entiers de Gauss x et y avec $y \neq 0$, il existe des entiers de Gauss q et r tels que $x = y \cdot q + r$ et N(r) < N(y). On peut le voir en écrivant $\frac{x}{y} = t + i \cdot u$ où t et u appartiennent à $\mathbb{Q}$ et en notant a l'entier le plus proche de t (ou bien un des entiers les plus proches de t), b l'entier le plus proche ou l'un des plus proches de u, et $q = a + i \cdot b$. On a alors $x = y \cdot q + r$ avec $r = y \cdot (t + i \cdot u) - y \cdot (a + i \cdot b) = y \cdot (t - a + i \cdot (u - b))$ et $N(r) = N(y) \cdot ((t - a)^2 + (u - b)^2) < N(y)$ car $(t - a)^2 + (u - b)^2 \leq \frac{1}{2}$ et $N(y) > 0$.

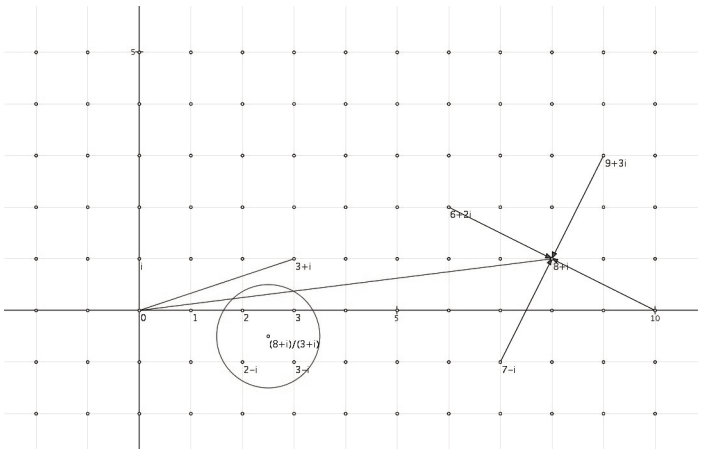
Exemple : considérons $x = 8 + i$ et $y = 3 + i$.

On a $N(y) = 10$ et $\frac{x}{y} = \frac{(8+i)(3-i)}{9+1} = \frac{5-i}{2}$. Nous disposons donc de quatre égalités de division euclidienne :

$$8 + i = (3 + i) \cdot (2 - i) + 1 + 2 \cdot i$$
$$= (3 + i) \cdot (3 - i) - 2 + i = (3 + i) \cdot 2 + 2 - i = (3 + i) \cdot 3 - 1 - 2 \cdot i.$$

Et pour chacune d'elles, nous constatons effectivement, par exemple sur la figure(3) qui suit, que la norme du reste, qui se trouve être toujours égale à 5, est strictement inférieure à 10.

(3) Figure réalisée à l'aide du logiciel CarMetal et de son langage JavaScript intégré. Voir http ://db-maths.nuxit.net/CaRMetal/



Or l'existence de cette division euclidienne dans $\mathbb{Z}[i]$ permet de démontrer que tous les idéaux[4] de $\mathbb{Z}[i]$ sont de la forme $g\cdot\mathbb{Z}[i]$ (idéal **principal**, engendré par g) où g est un élément de $\mathbb{Z}[i]$. En effet, si I est un idéal non nul de $\mathbb{Z}[i]$, l'ensemble E des normes des éléments **non nuls** de I est une partie non vide de $\mathbb{N}$ et possède donc un plus petit élément, qui est la norme d'au moins un élément non nul de I, j ; soit $x \in$ I ; nous savons qu'il existe des entiers de Gauss q et r vérifiant $x = j\cdot q + r$ et $N(r) < N(j)$. On a $r = x - j\cdot q$ avec $x \in$ I et $j \in$ I, ce qui implique $-j\cdot q \in$ I, d'où $r \in$ I ; comme $N(r) < N(j)$, on a, d'après la définition de j, $r = 0$, ce qui donne $x = j\cdot q$, égalité valable pour tout $x \in$ I dans laquelle, bien sûr, q dépend de x. Ainsi, $g = j$ convient. CQFD.

L'anneau des entiers de Gauss est donc un **anneau principal**[5].

Ceci permet de définir le pgcd de deux entiers de Gauss non nuls, z_1 et z_2 : c'est un générateur g de l'idéal non nul $z_1\cdot\mathbb{Z}[i] + z_2\cdot\mathbb{Z}[i]$ de $\mathbb{Z}[i]$ engendré par z_1 et z_2. Ce pgcd n'est donc pas défini de manière unique mais, comme deux de ces pgcd, g_1 et g_2, se divisent l'un l'autre du fait que, comme $g_1\cdot\mathbb{Z}[i] = g_2\cdot\mathbb{Z}[i]$, on a $g_1 = z_1\cdot g_2$ et $g_2 = z_2\cdot g_1$, d'où $g_1 = z_1\cdot z_2\cdot g_1$ avec $g_1 \neq 0$, donc z_1 et z_2 sont inversibles et g_1 et g_2 sont associés.

On peut donc décider arbitrairement qu'on prendra comme pgcd celui des quatre entiers de Gauss possibles qui est représenté géométriquement par un point situé dans le premier quadrant du plan. (Plus brièvement : le pgcd sera toujours situé dans le premier quadrant du plan complexe). La division euclidienne dans $\mathbb{Z}[i]$ évoquée plus haut permet de calculer « le » pgcd de deux entiers de Gauss en suivant les étapes de l'algorithme d'Euclide dans $\mathbb{Z}$.

Il s'en déduit aussi que, si z est un entier de Gauss irréductible qui divise le produit de deux entiers de Gauss z_1 et z_2, alors z divise z_1 ou z_2. En effet, si nous supposons que z ne divise pas z_1, l'idéal I de $\mathbb{Z}[i]$ engendré par z et z_1 est de la forme

(4) Une partie I d'un anneau commutatif A est appelée un **idéal** lorsque I est un sous-groupe additif de A et que, pour tout élément a de A et tout élément x de I, $a\cdot x \in$ I.

(5) On appelle anneau principal un anneau intègre dans lequel tout idéal est principal.

$g \cdot \mathbb{Z}[i]$ où g est un élément de $\mathbb{Z}[i]$ qui divise donc z_1 ainsi que l'entier de Gauss irréductible z. g, qui divise z_1, ne pouvant pas être associé à z est un inversible, donc $I = \mathbb{Z}[i]$, ce qui implique que 1 s'écrit sous la forme $v \cdot z + w \cdot z_1$, v et w étant des entiers de Gauss. On a alors $z_2 = z_2 \cdot 1 = z_2 \cdot v \cdot z + z_2 \cdot w \cdot z_1$. Comme z divise $z_1 \cdot z_2 \cdot w$ et aussi $z_2 \cdot v \cdot z$, z divise donc z_2. CQFD.

II. Algorithmes

Avec l'arrivée probable d'une spécialité informatique, il semble bon de se pencher sur l'algorithmique. Comme élément de réflexion, voici un texte de Jacques Arsac[6].

1. Présentation

1.1. Un extrait de l'article « Informatique et enseignement général » par Jacques Arsac

Cet article est paru en décembre 1993 dans la Revue de l'EPI n° 72 ; il est disponible en son entier par le lien

http ://www.epi.asso.fr/fic pdf/b72p117.pdf

« ... la nature véritable d'un programme. Toute instruction qui le compose décrit une action qui modifie des variables et change donc la **situation** qui existait avant son exécution. Un programme décrit ainsi la suite de transformations qui fait évoluer la **situation** depuis l'état **initial** (où les variables ont les valeurs des données) jusqu'à l'état **final** (où elles ont les valeurs des résultats cherchés). Un programme est difficile à lire parce que cette suite de transformations ne parle pas à l'esprit en elle-même ; on a trop vite perdu le fil des situations. C'est un peu comme avec ces manipulateurs de cartes …

… Au lieu d'aligner des instructions en s'en remettant aux essais pour savoir ce qui se passe, on suit l'évolution des situations, mettant à jour la situation au fur et à mesure de la rédaction des instructions, en tenant compte de leur signification. On sait ainsi ce que fait le programme et que, partant de la situation initiale, il produira la situation finale voulue. Ceci a valeur sinon de preuve formelle, du moins de validation sérieuse. Comme on connaît la situation réalisée à chaque pas, et le but cherché, on a un guide pour l'invention des transformations à faire. Il en résulte que bien des algorithmes s'en déduisent de façon assez naturelle… Très souvent, un résultat ne peut être obtenu directement. Il faut s'en rapprocher progressivement, par un chemin en spirale où l'on retrouve périodiquement la même situation générale, mais chaque fois un peu plus près de la situation finale. C'est le mécanisme de l'itération qui est une des grandes structures informatiques, et qui se rattache par cette méthode de raisonnement à la récurrence mathématique. »

Ainsi, les rappels et remarques précédents nous permettent de développer puis d'implémenter des algorithmes.

(6) Jacques Arsac, né le 1er février 1929, est un informaticien français, professeur émérite à l'université Pierre et Marie Curie, qui a beaucoup œuvré pour l'enseignement de l'informatique en France.

Les trois langages envisagés ci-dessous sont *interprétés* (non pas compilés), ce qui, en particulier, impose, sur l'ordinateur utilisé, la *présence du langage*, plus précisément de la ***version convenable*** du langage.

1.2. Maple

Maple est incontournable ; c'est un logiciel propriétaire de calcul formel édité par la société canadienne Maplesoft. La dernière version est la version 14, disponible depuis le 29 avril 2010 ; en France, Maple est le logiciel de calcul formel le plus utilisé dans l'enseignement[7].

Jusqu'en 1999, le nom des versions successives était de la forme « *Maple V release x* » (*x* jusqu'à 5), mais, depuis, les éditeurs se contentent d'un seul nombre.

Récemment, il a été possible de télécharger gratuitement sur le Web la version étudiant (*allégée*) *Maple V release 4*.

Toutes ces précisions sont importantes du fait que les différentes versions sont relativement incompatibles ; un programme écrit pour Maple V.4, comme ceux qui sont présentés ici, ne va absolument pas pouvoir fonctionner directement avec Maple version 14, par exemple.

En résumé : un logiciel de qualité, mais très dispendieux si on a besoin de se maintenir en permanence au niveau.

La syntaxe à employer avec Maple paraît assez simple lorsqu'on a déjà utilisé un ou des langages de programmation.

1.3. Maxima

Maxima est un logiciel *libre* de calcul formel, descendant, sous licence GPL[8], de la version de 1982[9] de Macsyma, un logiciel de calcul formel développé à partir de 1967 au MIT sous contrat du Département de l'Énergie des États-Unis.

La syntaxe de Maxima (à cause du grand âge de ce logiciel ?) peut paraître baroque au premier abord, beaucoup plus surprenante que celle de Maple mais, d'une part aavec un peu d'habitude elle s'acquiert rapidement, d'autre part il existe des interfaces (wxMaxima[10] par exemple) qui simplifient l'emploi de Maxima, notamment pour les élèves.

Maxima étant un logiciel libre, il ne pose pas de problème de crédits pour les mises à jour, ou de piratage. De plus, les versions successives de Maxima sont syntaxiquement compatibles.

---

(7) Information trouvée sur http ://fr.wikipedia.org/wiki/Maple

(8) General Public Licence ; voir http ://www.gnu.org/licenses/gpl.html

(9) William Frederick Schelter (1947–2001), professeur de mathematiques à l'Université du Texas (Austin) et informaticien, spécialiste de la théorie des anneaux non commutatifs et des applications de l'algèbre, notamment la démonstration automatique de théorèmes en geometrie, fit évoluer, de 1982 jusqu'à sa mort, une version personnelle de Macsyma et, en 1998, obtint du Département de l'Énergie des États-Unis l'autorisation d'éditer cette version sous licence GPL et sous le nom GNU Maxima.
Dans l'été 2001, lors d'un voyage en Russie avec son épouse Olga, il décèda subitement d'une attaque cardiaque à l'âge de 54 ans.

(10) Lors des JN de 2010, dans l'atelier n° 35, Monique Gironce avait utilisé Maxima avec cette interface.

1.4. Javascript

Tout le monde utilise Javascript tous les jours et, souvent, sans en être conscient. C'est un langage de programmation qui a été créé en 1995 par la société Netscape qui l'a *intégré* dans son navigateur Internet, celui qui était le plus utilisé à cette époque. Ce langage est implémenté dans tous les navigateurs d'aujourd'hui ; les utilisateurs ont, toutefois, la possibilité d'interdire le fonctionnement de Javascript, mais cela est très rarement utilisé.

Javascript est un langage à objets, mais il ne connaît absolument pas le calcul formel : nativement, il ne connaît pas les nombres complexes, ne sait pas intégrer, dériver, résoudre des équations, … Il faudra donc lui définir explicitement tout ce qui sera utilisé.

Cependant, sa présence permanente sur les ordinateurs rend son emploi très attractif.

L'imbrication des langages Javascript et HTML[11] peut rebuter : il faut, en fait, connaître les deux langages[12] ; mais cet investissement supplémentaire peut conférer à l'utilisateur beaucoup de possibilités, par exemple pour l'affichage des résultats.

2. Les algorithmes

2.1. Représentation des entiers de Gauss

Dans le cas des logiciels de calcul formel, nous utiliserons le type de données « nombre complexe » que possèdent ces logiciels. Pour JavaScript, on créera un objet ***Complexe*** qui se ramènera à la notion de couple de nombres muni de méthodes convenables.

(De toute manière, il serait prudent de toujours s'assurer que les parties entières et imaginaires utilisées sont des entiers relatifs.)

2.2. Description d'un algorithme

Un algorithme peut être généralement décomposé en trois parties : acquisition des données, traitement, sortie des résultats. Cette vision grossière suffit ici.

Autrefois, on décrivait les algorithmes à l'aide d'un organigramme, mais cela conduisait, dit-on, au style de programmation « go to » qui est abandonné[13].

On décrit maintenant un algorithme à l'aide d'un pseudo-langage de programmation, facile à comprendre, le seul symbole à connaître étant « := » qui représente l'affectation d'une valeur à une variable. Une variable à laquelle une valeur a été affectée conserve cette valeur tant que cette dernière n'a pas été remplacée par une autre.

(11) *HyperText Mark Up Language* ; c'est le langage de description de pages de la Toile.
(12) Voir l'annexe 4.
(13) E.W. Dijkstra. *GOTO statements considered as harmful*. Comm. ACM 11, 3, mars 1968, p. 147-148.

2.3. Algorithme du calcul de la norme de $z \in \mathbb{Z}[i]$

$u \coloneqq z$
$n \coloneqq u \cdot \overline{u}$
sortie de n

C'est simple[14] ; passons à plus compliqué !

2.4. Algorithme du calcul de celui des éléments associés à l'entier de Gauss z qui est dans le premier quadrant du plan complexe

On a vu précédemment que tout entier de Gauss, z, a un élément associé situé dans le premier quadrant. Comme la démonstration correspondante a été faite de manière constructive, l'algorithme recherché en découle. (C'est un argument en faveur de la connaissance des démonstrations constructives !)

Toutefois, le cas d'un réel ou d'un imaginaire pur est spécifique :

$u \coloneqq z$
si $Re(u) * Im(u) = 0$ alors
  $c \coloneqq |Re(u)| + |Im(u)|$
sinon
  si $Re(u) > 0$ alors
    si $Im(u) > 0$ alors
      $c \coloneqq u$
    sinon
      $c \coloneqq \mathrm{i} \cdot u$
  sinon
    si $Im(u) > 0$ alors
      $c \coloneqq -\mathrm{i} \cdot u$
    sinon
      $c \coloneqq -u$
sortie de c

L'imbrication des instructions « si … alors … sinon … » n'est clairement définie que grâce à l'indentation de certaines lignes par rapport aux autres ; le langage de programmation Python (langage à la mode !) est basé sur l'indentation, mais c'est le seul. Pour tout autre langage, une syntaxe spécifique doit être employée.

2.5. Algorithme du calcul « du » quotient et « du » reste de la division euclidienne d'un entier de Gauss x par un autre, y

De la définition vue précédemment découle l'algorithme figurant en annexe 1 ; avant de consulter cette annexe, vous pouvez, bien sûr, essayer d'écrire l'algorithme vous même.

2.6. Algorithme du calcul du pgcd normalisé de deux entiers de Gauss x et y

On écrit l'algorithme d'Euclide dans $\mathbb{Z}[i]$; le fait que ce soient, ici, non pas les

(14) On peut se demander à quoi sert la variable u ; sa présence évoque l'utilisation d'un programme (ou procédure, ou fonction) et le passage des données à celui-ci par l'intermédaire d'un ou plusieurs paramètres ou arguments.

restes de division mais les normes des restes de division qui sont de plus en plus petites, garantit quand même la validité de l'algorithme. Le dernier reste sera nul puisque sa norme sera nulle. Ici, nous ne rejetterons pas le cas où un des entiers de Gauss est nul[15].

```
a ≔ x
b ≔ y
si N(a) < N(b) alors
   échange de a et b
si b = 0 alors
   si a = 0 alors
      fin avec message d'erreur
   sinon
      fin et sortie de la valeur normalisée de a
sinon
   tant que b ≠ 0 faire
      c ≔ reste de la division euclidienne de a par b
      a ≔ b
      b ≔ c
fin et sortie de la valeur normalisée de a
```

2.7. Algorithme indiquant si un entier de Gauss *z* est irréductible ou non

Rappelons que z est irréductible si et seulement si il vérifie C_1 ou C_2 avec :
C_1 : la norme de z est un entier naturel premier
C_2 : z est associé à un entier naturel premier congru à 3 modulo 4.
De cela découle l'algorithme figurant en annexe 2 ; vous pouvez l'écrire vous même.

2.8. Algorithme déterminant tous les entiers de Gauss qui sont irréductibles et dont la norme est inférieure ou égale à un naturel *borne* donné

Cette fois, l'algorithme ne nous est pas donné directement par la connaissance mathématique des objets à obtenir.

Mettons en œuvre la méthode de la situation favorable[16] : au départ de l'algorithme, nous dirons que nous sommes dans une situation favorable, puisque nous disposerons des données qui nous seront nécessaires. Nous allons ensuite imaginer un enchaînement d'opérations nous conduisant au travers de diverses situations, toutes analogues, toutes favorables pour nous, en nombre fini, et dont la dernière correspondra exactement à notre objectif.

Il vient l'idée d'utiliser une liste pour représenter le résultat à obtenir ; au départ, cette liste sera vide ; les situations favorables successives correspondront au remplissage progressif de cette liste par les éléments recherchés, et ceci dans un ordre déterminé, lexicographique ici par exemple, afin de n'en oublier aucun !

On utilise deux variables prenant des valeurs entières, s et t, et une variable ayant une liste d'entiers de Gauss comme valeur, l. (Les logiciels de calcul formel possèdent le type de données « liste ».)

(15) Un seul !
(16) On peut penser que c'est *Jacques Arsac* qui a popularisé la méthode de la *situation favorable*.

```
a ≔ borne
l ≔ liste vide
pour s variant, par pas de 1, de 0 jusqu'à la partie entière de la racine carrée de a
faire
   t ≔ 0
   tant que la norme de l'entier de Gauss s + i·t est inférieure ou égale à a faire
      si s + i·t est irréductible, alors
         ajouter s + i·t à la fin de la liste l
      augmenter de 1 la valeur de t
sortie de l.
```

Il est très important de remarquer que l'avant-dernière ligne de l'algorithme est indentée de deux espaces, alors que la dernière ligne ne l'est pas du tout.

Par souci de concision, on peut réécrire cet algorithme en utilisant davantage de symboles :

```
a ≔ borne
l ≔ [ ]
pour s ≔ 0 jusqu'à partie entière (√a) faire
   t ≔ 0
   tant que norme(s + i·t) ≤ a faire
      si s + i·t est irréductible, alors
         l ≔ append(l, s + i·t)
      t ≔ t + 1
sortie de l
```

Appelons *irréductibles* cet algorithme.

2.9. Algorithme donnant un diviseur irréductible du complexe z

Il vient l'idée d'utiliser la liste obtenue ci-dessus pour y rechercher le résultat à obtenir.

```
t ≔ z
l ≔ irréductibles(norme(t))
j ≔ 1
tant que le reste de la division euclidienne de t par l[j] est non nul faire
   j ≔ j + 1
sortie de l[j]
```

Appelons *diviseur* l'algorithme constitué des quatre lignes précédentes, dans lequel $l[j]$ représente le contenu de la place d'indice j dans la liste l.

2.10. Algorithme déterminant la factorisation d'un entier de Gauss z en éléments irréductibles

De même, nous utilisons la liste des irréductibles de norme inférieure ou égale à N(z) car z est peut-être irréductible ; voir [2].

III. Conclusions

1. En plus de la question de l'écriture d'un naturel sous forme de la somme des carrés de deux naturels, il reste à étudier ***l'écriture des algorithmes utiles dans un des langages de programmation envisagés***. À titre d'exemple, voir l'annexe 3 et voir [2].

2. Ce qui peut être intéressant et qui ne se voit pas automatiquement, c'est le rôle des mathématiques dans l'efficacité d'un algorithme ; en effet, on pourrait écrire :

3. **Algorithme *peu efficace* indiquant si un entier de Gauss z de norme supérieure ou égale à 2 est irréductible ou non.**

$a \coloneqq z$
irred $\coloneqq$ *vrai*
borne $\coloneqq \sqrt{\text{norme}(a)}$
$s \coloneqq 0$
tant que *irred* est vrai et que $s^2 \leq$ *borne* faire
  $t \coloneqq 0$
  tant que *irred* est vrai et que norme$(s + \text{i}\cdot t) \leq$ *borne* faire
    si norme$(s + \text{i}\cdot t) > 1$ alors
      si le reste de la division eclidienne de a par $s + \text{i}\cdot t$ est nul alors
        irred $\coloneqq$ *faux*
    augmenter t de 1
  augmenter s de 1
sortie de *irred*

Cet algorithme va être très lent si la norme de z est grande (environ une seconde pour tester l'irréductibilité de 599 + 30·i), contrairement à celui qui est basé sur les Conditions C_1 ou C_2 (moins d'un centième de seconde).

4. Question qu'on peut alors se poser : sur quel résultat mathématique un algorithme efficace de décomposition d'un entier naturel comme somme des carrés de deux entiers naturels, lorsque celle-ci est possible, pourrait-il reposer [2] ?

IV. Annexes

Annexe 1 : algorithme du calcul « du » quotient et « du » reste de la division euclidienne d'un entier de Gauss x par un autre, y

$a \coloneqq x$
$b \coloneqq y$
si $b = 0$ alors
  fin avec message d'erreur
$q \coloneqq \dfrac{a}{b}$
$q \coloneqq arrondi\big(Re(q)\big) + \text{i}\cdot arrondi\big(Im(q)\big)$
$r \coloneqq a - b.q$
sortie de la liste composée de q puis de r

(La fonction « arrondi » est censée donner l'entier « le » plus proche d'un rationnel donné, ou d'un réel flottant donné).

Annexe 2 : algorithme indiquant si un entier de Gauss z est irréductible ou non

$a \coloneqq$ entier de Gauss associé à z mais qui est dans le premier quadrant
si la norme de est un entier premier alors
  sortie de *vrai*
sinon
  si a est un réel, donc un entier, est premier et est congru à 3 modulo 4 alors
    sortie de *vrai*
  sinon
    sortie de *faux*

Annexe 3 : écriture des deux premiers de ces algorithmes pour Maple et Maxima

A. Calcul de la norme de $z \in$ Z[i]

Programme Maple	Commentaire pour Maple
znorm := proc (u)	znorm : *nom d'une procédure (ou fonction) qui a un argument*
u*conjugate(u)	*calcul de la norme de l'argument ; le résultat reste « pendant »*
end ;	*Procédure terminée ; le résultat « pendant » sera renvoyé à l'appelant*

On voit que, pour Maple, le symbole d'affectation est tout simplement « := ». On voit aussi qu'**une variable peut contenir une fonction** (c'est le cas ici de la variable *znorm*), et que la définition d'une fonction peut contenir plusieurs instructions, la dernière instruction (« *end* ») étant donc nécessaire ; elle renvoie au programme appelant, qui est peut-être Maple (ou « *top-level* ») lui-même, le résultat de l'avant-dernière instruction, c'est-à-dire la norme qui était à calculer.

Exemple :

> znorm(2 - 3*I) ;

13

En effet, pour Maple, le symbole du complexe i est I. Le point virgule est nécessaire pour obtenir l'affichage du résultat ; si on ne souhaite pas d'affichage, on tape « : » à la place de « ; »

Programme Maxima
znorm(z) := expand(z*conjugate(z))
;

« := » est le symbole de Maxima *pour la définition* ***des fonctions***, le symbole correspondant à l'affectation ordinaire étant « : ». Une fonction de Maxima comprend **une seule instruction** dont le résultat est envoyé à l'appelant. Cela peut surprendre mais n'est pas gênant car Maxima utilise la notion d'instruction composée : pour en créer une, il suffit de mettre les instructions voulues bout-à-bout, entre parenthèses et séparées par des virgules. Le résultat d'une instruction composée est celui de la dernière instruction exécutée parmi celles qui la composent.

Maxima est moins « doué » avec les nombres complexes que Maple ; il est nécessaire de lui demander explicitement de poursuive le calcul jusqu'au bout en

utilisant la fonction *expand*.

Exemple :

(%i2) znorm(2-3*%i) ;

(%o2)

13

Maxima numérote ses lignes : (%i2) indique la deuxième ligne en entrée, (%o2) indique la deuxième ligne en sortie.

Pour Maxima, le symbole du complexe i est %i. Le point virgule est nécessaire pour obtenir l'affichage du résultat ; si on ne souhaite pas d'affichage, on tape « $ » à la place de « ; ».

B. Algorithme du calcul de celui des éléments associés à l'entier de Gauss z qui est dans le premier quadrant du plan complexe

Pour cet algorithme, nous allons utiliser des instructions de contrôle « si … alors … sinon … » imbriquées et nous allons voir la syntaxe qui doit être spécifiquement employée.

Programme Maple	Programme Maxima	Commentaire
znormalize := proc(u)	znormalize(u) :=	
if Re(u)*Im(u) = 0 then	if realpart(u)*imagpart(u) = 0 then	*premier niveau d'imbrication*
abs(Re(u)) + abs(Im(u))	abs(realpart(u)) + abs(imagpart(u))	*le résultat reste « pendant »*
else	else	
if Re(u) > 0 then	(if realpart(u) > 0 then	*deuxièmeniveau d'imbrication*
if Im(u) >= 0 then	(if imagpart(u) >= 0 then	*troisième niveau d'imbrication*
u	u	*le résultat reste « pendant »*
else	else	
I * u	expand(%i*u)	*de même*
fi	)	*fin de l'instruction if numéro 3*
else else		
if Im(u) > 0 then	(if 0 < imagpart(u) then	*troisième niveau d'imbrication*
–I u	expand(-%i*u)	
else	else	
–u	expand(–u)	
fi	)	*fin de l'instruction if numéro 4*
fi	)	*fin de l'instruction if numéro 2*
fi	)	*fin de l'instruction if numéro 1*
end ;	;	*le résultat est envoyé à l'appelant*

Dans la version V.4 de Maple, *fi* indique la fin de l'instruction *if* ; dans les versions plus récentes, cela a été remplacé par *end if*.

Exemple :

> znormalize(2 - 3*I) ;

3 + 2 I

Maxima reconnaît les instructions composées **grâce aux parenthèses**, mais on peut aussi employer celles-ci autour d'une instruction simple. Et c'est ce que nous faisons ici afin de lever l'ambiguïté causée, sinon, par l'imbrication des instructions *if*.

Exemple :
(%i6) znormalize(2-3*%i) ;
(%o6)

$$2\ \%i + 3$$

Annexe 4 : écriture des deux premiers de ces algorithmes pour JavaScript

HTML et JavaScript	*Commentaire*
<html>	*Début de la page* **HTML**
<head>	*Dans l'entête de cette page*
<title>Entiers de Gauss</title>	*Titre de cette page*
<script type="text/javascript">	*Dans l'entête figurent, à partir d'ici , des fonctions* **javascript.**
function Complexe(reel, imag) { this.x = reel ; this.y = imag ; } ;	*Pour Javascript, le symbole d'affectation est « = » ...* *Le fait d'écrire cette fonction ainsi crée la* ***classe*** **Complexe** *dont les éléments sont des couples du genre (z.x , z.y)*
Complexe.prototype.Re = function() { return this.x ; } ;	*Le mot prototype fait que cette fonction est, pour chaque nombre complexe, une* ***méthode*** *et doit donc être appellée ainsi :* **z.Re()**
Complexe.prototype.Im = function() { return this.y ;} ;	… *De même !*
Complexe.i = new Complex(0, 1) .	*Le complexe i.*
Complexe.prototype.toString = function() { return "(" + this.x + " + i.(" + this.y + "))";} ;	*Il est nécessaire qu'il existe une méthode toString car JavaScript l'utilisera lorsque l'affichage de l'objet sera demandé.*
Complexe.prototype.znorm = function() { return this.x*this.x + this.y*this.y ;} ; var z ;	*La norme de z s'obtient en écrivant* **z.znorm()** *Une variable globale utile pour la suite.*
Affichage = function() {	*Une fonction utile pour la suite.*
z = new Complexe(document.Calcul.ree.value, document.Calcul.ima.value) ; document.Calcul.affi.value = z ;} ;	*Voir plus loin ...*
</script></head><body>	*HTML : fin du script et de l'entête, début du corps de la page*
<H3> ENTIERS DE GAUSS</CENTER> </H3>	*Titre de paragraphe (paragraphe unique, en fait).*
<form name="Calcul"> Partie réelle <input type="text" name="ree" size="20" value=""> Partie imaginaire <input type="text" name="ima" size="20" value=""> <input type="button" name="bouton1" value= "Entrée et affichage" onClick="Affichage()"> <input type="text" name="affi" size="20" value="">) <input type="button" name="bouton2" value="Norme" onClick= "document.Calcul.norm.value = z.znorm() ;"> <input type="text" name="norm" size="20" value=""> 	*Début d'un formulaire* *Ici, libellé et zone de saisie,* *suivie d'un saut de ligne provoqué par * *Idem* *Ligne commençant par un bouton, suivi d'une zone de saisie qui recevra un résultat. Cliquer sur le bouton provoque l'exécution de la fonction* **affichage**. *(La variable globale z contient ensuite le complexe désiré.)* *Idem mais, ici, cliquer sur le bouton provoque* *le calcul de la norme de z et son affichage dans cette ligne.*
</P></form></body></html>	*Ici, tout se termine. Ouf !*

V. Bibliographie

[1] D. Guin et T. Hausberger : Algèbre I, groupes, corps et théorie de Galois, EDP Sciences, 2008.

[2] Pour trouver des compléments à cet article, on peut aller visiter le site

http ://pagesperso-orange.fr/jacques.faisant/Entiers de Gauss/