UNIVERSITÉ DU QUÉBEC À MONTRÉAL

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN MATHÉMATIQUES

par

SIMON PLOUFFE

APPROXIMATIONS DE SÉRIES GÉNÉRATRICES ET QUELQUES CONJECTURES

AOÛT 1992

AVANT-PROPOS

Un livre très intéressant a été publié en 1973 par N.J.A. Sloane. Il portait le titre "*A Handbook of Integer Sequences* " et comporte plus de 2372 suites d'entiers prises dans tous les domaines des mathématiques et des sciences en général. Depuis sa publication, des milliers de suites nouvelles ont été trouvées, spécialement en combinatoire. L'auteur invitait ses lecteurs à lui communiquer toute correction ou information nouvelle concernant une suite. Il a reçu environ un mètre cube de lettres depuis.

J'entrepris de taper le livre au complet à la main dans un ordinateur au début de 1990; cela m'a demandé 6 mois de travail. Je n'étais pas au bout de mes peines, car une fois cette tâche terminée, j'envoyai une lettre à l'auteur lui indiquant les erreurs dans certaines d'entre elles et que j'avais commencé à constituer une banque de données avec ses suites, etc. Je reçus un coup de téléphone environ deux semaines plus tard. L'auteur était un peu surpris (et moi donc) que quelqu'un se soit donné la peine de taper tout le livre alors que lui avait un fichier sur ordinateur qui contenait toutes les suites. Après une heure de discussion, l'auteur disait qu'il était temps qu'il fasse la 2ème édition de ce livre. Moi je lui disais qu'il était temps que je complète mes études, etc. C'est là que tout a commencé. C'est en essayant de vérifier les suites d'entiers avec un programme que ce projet est né. Je voulais pouvoir vérifier les chiffres des suites pour qu'il n'y ait pas d'erreurs.

C'est également avec l'encouragement et la vision de mon directeur, Gilbert Labelle, que ce mémoire a vu le jour, à la confiance de Pierre Leroux, aux idées génératrices de mon co-directeur, François Bergeron. Tous les autres aussi, qui sont en France, à Bordeaux au LaBRI avec leurs chauds encouragements. Je pense à Xavier Viennot qui m'impressionnait tellement avec ses conférences en 1985, à Maylis Delest, Serge Dulucq, Jean-Guy Penaud, Jean-Marc Fedou, Mireille Bousquet-Mélou, etc. Ceux de Paris à l'INRIA qui m'ont invité à leur en parler et qui ont contribué grandement à faire que le

programme *gfun* soit une réalité. Je pense à Paul Zimmermann , “en possession tranquille de la vérité”, Bruno Salvy “le fou de Maple” à qui je dois de vraies belles formules trouvées grâce à ses méthodes (elles sont dans la table en appendice), Philippe Flajolet, “le bon maître”. Je leur dois des discussions fort enrichissantes.

A Neil Sloane évidemment, mon guide et mon maître à penser, qui m’a fait l’honneur de bien vouloir être mon “advisor” comme il se plait lui-même à le dire. Je lui dois de précieux conseils.

A ma mère, qui sera pas mal fière et contente que son garçon fasse une maîtrise en mathématiques.

A ma compagne Danièle, qui m’a beaucoup aidé au tout début pour la vérification des suites et qui m’a soutenu jusqu’à la fin. Je lui dois et lui dédie ce mémoire.

RÉSUMÉ

Le présent mémoire tente de répondre à une question simple : Étant donné une suite numérique, comment trouve t-on la fonction génératrice de cette suite? Il s'agit donc de prendre les termes d'une suite et de proposer une façon de les générer à l'aide d'une formule quelconque (simple si possible). Pour ce faire nous avons utilisé des programmes de calcul symbolique couramment disponibles, soit MapleV de l'Université de Waterloo et Pari-GP, un programme développé à l'Université Bordeaux I. Le jeu d'essai des suites est le livre bien connu de Neil J.A. Sloane, *A Handbook of Integer Sequences*[1] . L'exposé se compose de deux parties principales. La première explique les quatre méthodes qui ont permis de répondre à notre question initiale. La deuxième contient une table des formules trouvées à l'aide de ces méthodes. En tout, 1031 fonctions génératrices forment la table sur un total de 4568 suites que composait le jeu d'essai, soit à peu près 23% des suites.

Ces 1031 formules ont toutes été obtenues expérimentalement. C'est donc dire qu'en fait ce sont autant de conjectures. Mais nous verrons que dans presque tous les cas les méthodes sont suffisamment sophistiquées pour pouvoir affirmer que les formules sont les bonnes.

[1] Le jeu d'essai est en fait la 2e édition de ce livre qui est en préparation.

TABLE DES MATIÈRES

INTRODUCTION

Dans toute cette étude nous procéderons selon une seule ligne directrice: il s'agira de prendre une suite numérique finie et à l'aide d'un programme informatique spécialisé, d'identifier un bon candidat pour la fonction génératrice. Une telle approche pourrait se limiter simplement à consulter un livre de table de suites. Dans [GKP] p.42 on note: "the best source for questions about sequences is an amazing little book called the Handbook of Integer Sequences, by Sloane, which lists thousands of sequences by their numerical values."; et aussi: "the look-up method is limited to problems that other people have decided". De plus, après avoir donné un exemple de suite numérique, les auteurs [GKP] p.327, ajoutent: "no closed form is evident, and this sequence isn't even listed in Sloane's Handbook".

Nous présentons ici une solution à ce problème: c'est-à-dire une méthode alternative aux méthodes standard connues dans ce domaine. Ces dernières partent des propriétés mathématiques d'une suite et de là en font l'analyse, le tout étant basé sur la connaisssance a priori des ces propriétés. Dans la présente étude nous proposons de procéder en sens inverse: c'est uniquement à partir de la suite numérique que les propriétés sont établies. A cette fin, nous décrirons quatre méthodes d'analyse d'une suite numérique.

Ces quatre méthodes s'appuient sur quatre modèles de fonctions génératrices. Le premier modèle suppose que les termes de la suite peuvent être générés avec le développement en série de Taylor d'un quotient de polynômes. Le deuxième modèle suppose que la suite satisfait à une récurrence linéaire à coefficients polynomiaux, appelée aussi une P-récurrence. Le troisième modèle suppose que la suite est donnée par le développement en série d'un produit infini, comme la suite des partages d'entiers ordinaires. Le quatrième modèle enfin suppose qu'une transformation simple de la suite permet de retrouver une fonction génératrice connue. Cette dernière est en fait une version améliorée de la "look-up method" de [GKP].

NOTES

Pour éviter les répétitions inutiles tout au long de cette étude, nous emploierons la notation Nxxxx pour désigner la suite numéro xxxx du livre de Sloane [Sl] cité en bibliographie. Par exemple, la suite des nombres de Catalan porte le numéro 577, on y fera référence en écrivant N0577. Les autres suites, celles apparues après 1973 dans la littérature, ont été recataloguées dans une deuxième édition que nous préparons avec Neil J.A. Sloane [PlSl]. Elles portent un numéro séquentiel "absolu" noté Axxxx. Donc quand nous parlerons du numéro de suite A3890, nous entendrons le numéro séquentiel de cette table. C'est cette même numérotation qui apparaît dans la table des résultats en appendice. Pour des raisons évidentes de consistence, il était nécessaire de conserver un numéro qui fasse référence toujours à la même suite sans ambiguité. En résumé :

• Nxxxx :Numéro séquentiel de la suite du livre de Sloane [Sl].

• Axxxx :Numéro séquentiel de la suite du livre [PlSl].

La plupart des algorithmes et méthodes décrites dans cette étude ont été regroupés dans un programme appelé "gfun" qui fait partie de la librairie publique de Maple de l'université de Waterloo. On peut avoir une copie de ce programme par transfert électronique via "ftp/anonymous". Le programme a été écrit en collaboration avec François Bergeron, professeur au département de Mathématiques/Informatique à l'Université du Québec à Montréal et également avec Paul Zimmermann et Bruno Salvy tous deux chercheurs à L'INRIA/Rocquencourt.

CHAPITRE 1

LA MÉTHODE DES APPROXIMANTS DE PADÉ

1.1 Les fractions rationnelles.

Une façon de donner les termes d'une suite est de les engendrer à l'aide d'une fraction rationnelle. Par exemple la suite de Fibonacci peut être générée à l'aide du développement en série de Taylor à l'origine de

(1.1) $1/(1- z - z^2) = 1 + z + 2\,z^2 + 3\,z^3 + 5\,z^4 + 8\,z^5 + 13\,z^6 + ... + a_n z^n + O(z^{n+1})$.

De la même façon ces termes peuvent être calculés avec la récurrence $a_n = a_{n-1} + a_{n-2}$. Les deux représentations sont équivalentes. Il y a une correspondance assez simple entre la fraction rationnelle et la récurrence. Le dénominateur de la fraction rationnelle "est" la relation de récurrence. Le numérateur tient lieu de conditions initiales de cette même récurrence. Le lien se trouve en fait dans la réécriture de la récurrence en termes de z^n plutôt que n. Une procédure simple en quatre étapes, permettant de passer d'une récurrence linéaire à coefficients constants à la fraction rationnelle correspondante, est décrite dans [GKP]. On peut montrer que la réécriture se fait dans l'autre sens également. Cette mécanique est très connue mais suppose toujours que l'on connaisse au moins l'une des deux représentations.

Notre seul point de départ est la série S(z) tronquée à l'ordre k. Ce qu'on désire faire est de la représenter par une fraction rationnelle. Alors si on pose k=L+M et

(1.2) $$S(z) = \frac{u_0 + u_1 z^1 + ... + u_L z^L}{v_0 + v_1 z^1 + ... + v_m z^m} + O(z^{L+M+1})$$

il est toujours possible de trouver une solution à cette équation. La façon de faire est de fixer L et M

d'abord. Puis en multipliant le membre de gauche avec le dénominateur on obtient un système de M équations à M inconnues qui déterminent les constantes en v. Pour poser ces équations, on "identifie" les coefficients de z^i avec $L+1 \leq i \leq L+M$. Une fois trouvés les v_i, on peut faire de même avec le numérateur pour déterminer les constantes en u_j, en identifiant cette fois les coefficients de z^j, $0 \leq j \leq L$. Il est toutefois plus aisé de poser en partant que $v_0 = 1$. On donne le nom d'approximant de Padé [L/M] à l'expression trouvée pour un L et M donnés. Le calcul d'un approximant de Padé se fait en principe de façon mécanique. La plupart des programmes de calcul symbolique sur le marché aujourd'hui effectuent ce calcul automatiquement. On parle ici de la résolution du système d'équations linéaires pour un L et un M donnés. En théorie le problème est clos, mais dans la pratique il en est autrement.

Nous illustrerons les difficultés rencontrées en donnant deux exemples extrêmes d'approximants de Padé.

Exemple 1.1 La suite des parts de gâteaux en 3 dimensions.

Le premier est la suite N0419 qui porte le nom de: "Slicing a cake with n slices"; elle est plutôt simple et connue. C'est le nombre de parts de gâteaux différents avec n coupes en 3 dimensions. Nous nous en tiendrons uniquement aux termes numériques sans tenir compte du contexte. Considérant une quinzaine de termes, on pose les équations et les degrés des deux polynômes, en supposant que les degrés sont de taille égale, i.e. que L=M. La difficulté réside dans le fait que si le système *peut* se réduire, il faut prévoir un algorithme pour le simplifier d'une façon ou d'une autre. Justement, cette suite N0419 est une fraction rationnelle de degré [2/4]. Elle est complètement décrite par cette fraction rationnelle. C'est donc que, ayant pris notre quinzaine de termes et ayant supposé que L=M=7, on aurait été conduit à réduire le système à un nombre d'inconnues et d'équations plus petit. Donc à moins d'être chanceux, i.e. de prévoir exactement à l'avance le degré de la fraction rationnelle, on n'est pas assuré de trouver la *juste* fraction rationnelle.

La deuxième difficulté vient de la taille des calculs. Si la suite considérée EST une fraction rationnelle, comme la suite de Fibonacci (1.1), cela n'a rien de dramatique si on a fait un choix de L et M heureux. Si la suite N'EST PAS une fraction rationnelle, c'est là que les calculs deviennent énormes. Selon l'équation (1.2), il est quand même possible de trouver une fraction rationnelle qui se juxtaposera aux k premiers termes de toute suite, mais elle ne se simplifiera pas.

Exemple 1.2 La suite des nombres premiers : 2,3,5,7,11,13,....

Nous prendrons ici la suite des nombres premiers N0241. On le sait, il n'existe pas de fonction rationnelle qui permette de les obtenir successivement. Si on prend les 20 premiers termes , de 2 à 71 et que l'on cherche une expression rationnelle qui se juxtapose à cette suite, l'expression que l'on trouvera sera une fraction rationnelle d'une taille appréciable. La taille, disons en nombre de caractères, dépassera largement celle de la suite. Il ne faut pas oublier que l'on cherche une solution rationnelle, donc exacte à l'ordre d'approximation de la série de départ; ce ne sont pas des calculs en "virgule flottante". Ainsi, avec les 48 premiers termes de la suite des nombres premiers on obtient une fraction rationnelle d'une taille de l'ordre de 10,000 caractères, chaque coefficient étant de l'ordre de 120 chiffres. La taille de la suite de départ avec ses 48 termes, pour sa part, ne dépasse pas 200 caractères.

Il existe une procédure en Maple qui permet de convertir une série (tronquée) en une fraction rationnelle. Elle porte le nom de "ratpoly" pour "rational polynomial". Cette procédure est une véritable perle de programmation (elle a plus de 500 lignes). Non seulement elle fait le calcul exactement à l'ordre d'approximation de la série, mais en plus elle le fait bien. On le sait, Maple est en mesure d'effectuer des calculs symboliquement et en principe avec une précision infinie. Le résultat en est que les deux difficultés rencontrées plus tôt sont complètement tranparentes à l'utilisateur.

Donc avec cet outil presque "magique" qu'est "ratpoly", il est possible assez facilement d'effectuer le calcul fastidieux de représentation d'une suite sous forme de série avec une fraction rationnelle. En fait, deux critères simples nous permettrons de détecter une *bonne* fraction rationnelle. Le premier est le degré de l'expression trouvée: si le degré total (L + M) retourné par le programme est plus petit que le nombre de termes, on est alors potentiellement en présence d'une bonne représentation. Le deuxième critère est la taille (en nombre de caractères) de l'expression: si la taille de l'expression est plus grande que la taille de la suite testée, on rejette alors l'expression rationnelle candidate. En combinant ces deux critères, il est possible de détecter avec une assez grande certitude une suite qui EST une fraction rationnelle simple.

En soumettant toute notre table de 4568 suites à cette simple procédure qu'est "ratpoly" , nous avons détecté 614 fractions rationnelles. De ce nombre, 580 nous semblent bonnes: elles sont

répertoriées dans la table en appendice. On peut consulter [BP] à ce sujet également.

1.2 La dérivée logarithmique et l'inverse fonctionnel.

Malgré le succès remporté (en nombre de fonctions génératrices trouvées) avec notre méthode des approximants de Padé, une partie du problème demeure. Si 580 suites sur 4568 sont des fractions rationnelles, quelle est alors la nature des quelque 4000 qui restent ? La réponse à cette question est inconnue. Ce que l'on sait, c' est que notre méthode permet de détecter la fraction rationnelle d'une suite comme celle de Fibonacci. Elle permet également de détecter des variantes de celle-ci. Il se trouve que la plupart des opérations simples et connues que l'on peut effectuer sur une suite sont en fait des *transformations rationnelles* . Une TR en plus court. Si S(z) est notre suite sous forme de série tronquée, une TR conservera le caractère rationnel de la fonction génératrice. Par exemple, la différence terme à terme de la suite est une TR puisqu'il suffit d'effectuer S(z)(1-z). La somme de deux termes successifs est également une TR : il suffit de faire S(z)(1+z). La suite des sommes partielles s'obtient en prenant S(z)/(1-z), etc. Il en est de même de l'inverse de ces transformations. Ce point est essentiel.

Donc les suites qui ont une fonction génératrice qui est une fraction rationnelle et toutes les variations usuelles de celles-ci sont détectées avec notre méthode.

L'idée fort simple est alors d'utiliser notre méthode et une transformation qui ne soit pas rationnelle dans les deux sens, dans le but de détecter d'autres types de fonctions génératrices. Par exemple, bien que la dérivée soit une transformation qui conserve le caractère rationnel d'une expression, il suffit de prendre une fraction rationnelle quelconque pour se rendre compte que l'intégrale n'est pas une fraction rationnelle en général. En effectuant une dérivation et en appliquant ensuite notre méthode de détection, on pourra obtenir des fonctions génératrices qui sont en fait des intégrales de fractions rationnelles. En poussant le même raisonnement plus loin, on pourrait effectuer d'autres transformations de ce type comme la dérivée du logarithme ou l'inverse fonctionnel. Si nous pouvons toujours retourner sur nos pas à chaque fois, cela nous donne une façon de détecter des expressions qui font partie d'une classe plus vaste que les fractions rationnelles. Avec la dérivée, il est facile de revenir en arrière: une fois le test effectué, si c'est rationnel, il suffit de faire l'intégrale de l'expression. La dérivée du logarithme est aussi "réversible": il suffit de faire l'exponentielle de l'intégrale de l'expression trouvée. L'inverse fonctionnel d'une série est également "réversible" à condition que la suite débute par 0,1,…:

en effet, l'inverse d'une série à coefficients entiers est aussi à coefficients entiers, si la série s'annule en zéro et son premier terme non nul est 1.

C'est l'expérience qui a orienté le choix des transformations judicieuses à effectuer. Le succès d'une transformation plutôt que d'une autre étant guidé simplement par le nombre de fonctions génératrices trouvées une fois la table complète traitée par le programme. Le rejet ou l'acceptation d'une expression est donné par les deux critères énoncés plus haut. Il y a aussi le fait que plus on tranforme une suite avec de telles opérations, plus précises et strictes sont les conditions imposées à la suite de départ. Par exemple, l'inverse fonctionnel de la dérivée du logarithme d'une suite sous forme de série tronquée doit se faire seulement si les coefficients sont restés entiers et débutent par 0,1, ... , une fois que la première transformation a été effectuée.

Notre choix s'est arrêté sur la dérivée, la dérivée logarithmique et l'inverse fonctionnel. Ce sont ces opérations qui ont remporté le plus de succès. Exactement 120 fonctions génératrices qui ne sont pas des fractions rationnelles ont été isolées de cette façon. En tout 700 fonctions génératrices (incluant les fractions rationnelles) ont été trouvées grâce à la procédure "ratpoly". Les résultats sont présentés en appendice.

CHAPITRE 2

LA MÉTHODE DES P-RÉCURRENCES

2.1 Les suites P-récurrentes.

L'hypothèse de travail que nous posons ici sur la suite a_n consiste à dire que chaque terme de celle-ci peut être calculé à partir des termes précédents. Dans [Sta80] on introduit ce genre de dépendance sur les autres termes en disant que la suite a_n est une suite P-récurrente, si elle satisfait l'équation suivante

(2.1) $$a_n P_0(n) = a_{n-1} P_1(n) + a_{n-2} P_2(n) + \ldots, + a_{n-k} P_k(n)$$

où les $P_i(n)$, $0 \leq i \leq k$, sont des polynômes à coefficients rationnels. Ce type de relation est une classe plus vaste que les relations de récurrences linéaires ordinaires à coefficients constants rencontrées au chapitre précédent. En effet, il y a équivalence entre les fonctions génératrices rationnelles et les relations de récurrence à coefficients constants. Il n'y a cependant pas d'équivalent en termes de fonctions génératrices pour les P-récurrences en général. A l'heure actuelle, il n'existe pas de méthode pour trouver la fonction génératrice correspondant à une P-récurrence quelconque; seuls certains types de P-récurrences peuvent être résolus. Ce qui peut être fait, par contre, est de vérifier si la suite satisfait *numériquement* une P-récurrence. On ne peut donner qu'une P-récurrence *vraisemblable*.

Il faut donc procéder pas-à-pas en augmentant le degré et le nombre de termes. Nous posons d'abord les équations et, en supposant que la suite satisfasse l'équation (2.1) où les $P_i(n)$ sont des polynômes de degré d, il y aura (d+1)(k+1) équations (il faut tenir compte du terme de rang 0). On dira alors qu'elle satisfait une P-récurrence de type (d,k). On remarque que le système admet toujours une solution nulle. S'il y a solution, il y en aura une infinité, ce qui découle du fait que le système d'équations est non-homogène. Ceci est évident, puisque l'on peut multiplier par une constante C arbitraire de chaque côté sans changer l'équation. On prendra donc soin de garder la solution la plus simple. La

résolution d'un système d'équations linéaires est une chose que les programmes de calcul symbolique comme Maple font couramment. Un programme a donc été écrit pour permettre de résoudre le système à (d+1)(k+1) inconnues. Le voici, en entrée il accepte une suite et en sortie il donne soit 0 soit une ou plusieurs constantes, quand le nombre de constantes est 1 on pose la solution comme étant la plus simple en substituant la constante à 1.

```
1) read suite : listesuite:=":
2) nbrdetermes:=nops(listesuite):
3)    rec:=proc(w,n,t) local ff,c,d,i,j,k,ii;
4)    option remember;
5)    termes:=(n+1)*(t+1);
6)    if termes>=nbrdetermes then RETURN ( `impossible de resoudre` ) fi;
7)    for ii from 1 to nbrdetermes do a(ii):=op(ii,w) od:
8)          ens:={seq(c[jj],jj=1..termes)}:
9)    s:={seq(sum(sum(k**d*c[j*n+jn+d],d=0..n)*a(kj+1),j=1..t+1),
          k=t+1..termes+t)}:
10)   solution:=[solve(s,ens)];
11)         if sol=[] then RETURN (0) else
            RETURN(assign(solution),[seq(c[kk],kk=1..termes)])
            fi;
      end:
```

Donnons une courte description du programme.

1) On lit la suite provenant d'un fichier.

2) On pose que la variable nbrdetermes est égal au nombre d'éléments de la liste qui contient la suite.

3) Appel de la procédure et on pose les variables locales.

4) On prend l'option "remember" , très importante.

5) On prend un nombre de termes suffisant pour résoudre le système d'équations linéaires.

6) Si le nombre de termes nécessaires est trop grand, un message d'erreur est imprimé.

7) On pose les constantes dans notre système d'équations. Ici ce sont les termes de la suite.

8) On pose les inconnues de notre système sous forme d'ensemble.

9) On pose les équations linéaires.

10) On tente de résoudre.

11) Si le système admet une solution nulle (liste vide ici) on retourne 0. Sinon on assigne les solutions trouvées.

Donc en entrée le programme accepte une suite numérique et teste si celle-ci satisfait une équation P-récurrente de degré d à k termes.

Le programme qui détermine si une suite satisfait une P-récurrence est une des méthodes les plus rapides et de plus, une fois la P-récurrence candidate trouvée, il est très facile d'obtenir des centaines de termes de la suite. En principe si on veut calculer les termes d'une suite une fois obtenue une P-récurrence, il suffit de la mettre telle quelle dans un programme. Il n'est cependant pas approprié d'utiliser une procédure qui soit purement récursive même si c'est d'abord ce qui vient à l'esprit. Il faut linéariser le temps de calcul d'une procédure qui s'appelle elle-même, sinon celui-ci devient vite exponentiel. L'exemple souvent donné dans les cours de programmation de base est la suite des nombres factoriels, 1,1,2,6,24,120,720,..., définie par $a_0 = 1$ et $a_n = n\, a_{n-1}$. Ce problème est facilement résoluble en Maple, puisque les procédures récursives peuvent être *linéarisées* simplement en écrivant "option remember" dans l'appel de la procédure. Maple se charge alors de ré-écrire la procédure en créant une table d'adressage (interne) automatiquement.

Comme avec les autres méthodes, nous avons utilisé la table de [PlSl] au complet. A chaque suite, le test a été effectué sur les degrés 1 à 4 et sur un nombre de termes variant de 1 à 5, compte tenu que l'expérience indique que la plupart des suites P-récurrentes ont un degré assez bas. Stanley [Sta80] donne un exemple de suite P-récurrente de degré 3 à 2 termes qui donne les nombres d'une suite de Apéry utilisée dans la preuve de l'irrationalité de $\zeta(3)$.

Exemple 2.1 : $n^3\, a_{(n)} + (n-1)^3\, a_{(n-2)} = (34\, n^3 - 51\, n^2 + 27\, n - 5)\, a_{(n-1)}$,

En tout, 250 des 1031 suites que contient la table en appendice, seraient P-récurrentes. De ces 250, 220 ont une fonction génératrice associée trouvée par d'autre méthodes. Il en reste donc 30 dont on ne connait que la P-récurrence. Sont comptées ici les suites P-récurrentes de degré 1 ou plus; les fractions rationnelles, au nombre de 580, sont aussi P-récurrentes mais de degré 0. C'est de loin la méthode la plus puissante, puisque au total, près de 81 % des suites qui ont une fonction génératrice connue sont P-récurrentes à des degrés divers, ce qui représente 18 % de tout le catalogue des suites de [PlSl].

2.2 Les suites hypergéométriques.

Dans [GKP] on fait une remarque très simple au sujet des P-récurrences d'un certain type. Si une suite t_k satisfait une P-récurrence de type (d,1), c'est donc que le quotient des termes successifs t_{k+1}/t_k = P(k)/Q(k), où P(k) et Q(k) sont deux polynômes. La fonction hypergéométrique est à peu de chose près la même chose. En effet, la définition de celle-ci étant

$$F\left(\begin{matrix}a_1, a_2, \ldots a_m \\ b_1, b_2, \ldots, b_n\end{matrix}\middle| z\right) = \sum_{k=0}^{\infty} \frac{a_1^{\overline{k}} \ldots a_m^{\overline{k}}}{b_1^{\overline{k}} \ldots b_n^{\overline{k}}} \frac{z^k}{k!} \qquad (2.2)$$

où le membre de gauche en est l'écriture avec les paramètres en a et en b et où le membre de droite en est le développement en série sous forme de somme de quotients de produits de polynômes factoriels ascendants. En spécifiant que les termes en b ne s'annulent nulle part, on évite la division par zéro; il suffit simplement pour cela qu'ils soient toujours positifs. Considérons le rapport de deux termes successifs et en posant que le premier terme t_0=1,

$$\frac{t_{k+1}}{t_k} = \frac{a_1^{\overline{k+1}} \ldots a_m^{\overline{k+1}}}{a_1^{\overline{k}} \ldots a_m^{\overline{k}}} \frac{b_1^{\overline{k}} \ldots b_n^{\overline{k}}}{b_1^{\overline{k+1}} \ldots b_n^{\overline{k+1}}} \frac{k!}{(k+1)!} \frac{z^{k+1}}{z^k}$$

il est alors facile de simplifier cette expression en revenant à la définition d'un polynôme factoriel ascendant de degré k+1 et de degré k. D'où l'expression:

$$\frac{t_{k+1}}{t_k} = \frac{(k+a_1)\ldots(k+a_m)z}{(k+b_1)\ldots(k+b_m)(k+1)} \quad .$$

On obtient alors une fraction rationnelle en k seulement. Donc si on a une suite qui débute avec 1 et dont le rapport des termes successifs est une fraction rationnelle (une P-récurrence de type (d,1)), elle pourra être "lue" directement comme étant une série hypergéométrique. L'avantage énorme de la représentation d'une suite comme "hypergéométrique" est que le programme de calcul symbolique Maple est en mesure de manipuler et de simplifier de telles séries. Dans sa version 5, Maple utilise les tables d'identités hypergéométriques qui se trouvent dans [AS1]. Ce livre étant une véritable bible de formules mathématiques, nous avons à notre disposition un outil excessivement puissant. En fait, dès que l'on sait qu'une suite satisfait une P-récurrence de type (d,1) nous disposons déjà d'une information très précieuse.

Cette représentation en série hypergéométrique ouvre la porte à d'autres formes de fonctions génératrices. Le programme Maple est en effet capable, dans certains cas, de donner directement la

fonction génératrice explicite sous forme simplifiée. Il suffit de faire appel à la procédure “simplify” qui réussit à reconnaître les expressions contenant des termes hypergéométriques. C’est alors que les tables d’identités de [AS1] sont appelées et, si la forme le permet, Maple retourne directement une expression algébrique explicite.

Conformément aux autres méthodes nous avons donc, encore une fois, testé toute la table de [PlSl] en recherchant des P-récurrences de type (d,1). Plus de 94 suites satisfont à ce type de récurrence. Dans certains cas, la forme hypergéométrique a été directement simplifiée automatiquement par le programme Maple. Les résultats sont présentés dans la table de fonctions génératrices en appendice.

2.3 L’algorithme LLL[2] .

Nous décrirons ici la méthode qui est la plus complexe et puissante de toute cette étude. On s’intéresse aux suites qui sont P-récurrentes de type (d,k) en général. Cette méthode ne s’applique que si on peut avoir autant de termes de la suite que l’on veut. Comme nous l’avons vu à la section précédente, Maple est en mesure, dans les cas où la P-récurrence est de type (d,1), de donner une forme hypergéométrique et une fois obtenu cette forme, de produire directement la fonction génératrice algébrique lorsqu’elle s’y prête. C’est donc que : les P-récurrences de type (d,1) sont quelquefois algébriques. Il en est de même pour les P-récurrences de d’ordre plus élevé. Ce qui nous manque est la façon d’obtenir la forme close. On ne dispose malheureusement pas de moyen de savoir quel type de P-récurrence représente une suite qui a une fonction génératrice algébrique. D’après Stanley [Sta80], une fonction génératrice algébrique est toujours P-récurrente. Ici c’est l’inverse qu’on cherche, malheureusement ce n’est pas toujours vrai : la fonction exp(x) est P-récurrente mais certainement pas algébrique.

Une suite a une fonction génératrice algébrique si elle satisfait à

$$\sum_{0 \le j,\, k \le m} c_{j,k} S(z)^j z^k = 0 \tag{2.3}$$

[2] **Nommé ainsi à cause des travaux de Lenstra, Lenstra et Lovasz.**

où S(z) est la série qui représente la suite a_n et les $c_{j,k}$ sont constantes. On pourra alors obtenir la fonction génératrice close si on peut isoler S(z). Le problème est double ici: il faut d'abord obtenir l'équation (2.3) et de plus on n'est pas assuré de pouvoir isoler S(z). Ce qui vient à l'esprit est d'essayer de trouver "à tâtons" une équation en S(z) et z qui s'annulera. Il est possible effectivement de faire un programme qui fonctionnerait sur le même principe que les P-récurrences. Mais malheureusement la forme qu'on obtiendra ne sera pas, en général, la plus simple. Par exemple, la suite N0577, les nombres de Catalan, satisfait à une telle équation. Elle est de degré 2 : $S(z)^2z - S(z) + 1 = 0$. Si on résout cette équation par rapport à S(z), on obtient une fonction génératrice close des nombres de Catalan. On s'aperçoit alors qu'il il y a une infinité de telles équations que l'on peut poser. On pourrait peut-être en obtenir une plus simple.

Il existe un algorithme implanté en Maple qui porte le nom de "minpoly". Il fait appel à l'algorithme LLL. Disons simplement qu'il permet de résoudre numériquement le problème exactement inverse de trouver une racine d'un polynôme. La recherche numérique des racines d'un polynôme est un problème résolu. Mais nous posons la question suivante: étant donné un nombre réel, de quel polynôme minimal est-il racine ? Mentionnons dès maintenant qu'on parle ici d'un nombre réel donné avec une certaine précision numérique. On ne pourra (une fois l'opération réussie) qu'isoler un polynôme qui *semble* avoir ce nombre réel comme racine. Il serait un peu long de donner tous les détails qui font qu'aujourd'hui ce problème est pour ainsi dire *numériquement résolu.* Mentionnons cependant qu'au moins trois programmes de calcul symbolique ont implanté cet algorithme: soit Maple, Mathematica et Pari-GP. Pour la description de cet algorithme, on pourra consulter [BaKa] ou l'article original de [LLL]. La meilleure version de cet algorithme et de loin la plus rapide est celle qui existe sur Pari-GP [Pari]; elle est au moins 800 fois plus rapide que la version équivalente sur Maple. Quant à Mathematica, disons qu'il est, de façon générale, 4 fois plus lent que Maple dans tous les calculs. Nous ne l'avons pas considéré ici.

Cette procédure accepte donc en entrée un nombre décimal et donne (selon la précision numérique en vigueur) le polynôme minimal dont il serait racine. La précision numérique en vigueur est celle que l'utilisateur demande. Elle devrait idéalement être infinie. Plus raisonnablement, la limite est d'environ 100 chiffres décimaux sur les machines à notre disposition avec Maple et d'environ 500 chiffres décimaux avec Pari-GP. Le degré maximal du polynôme que l'on puisse demander dépend largement de

la précision. Dans la pratique, la limite est un polynôme de degré 20. Ceci est quand même suffisant pour obtenir des résultats intéressants.

Evidemment, si la fonction génératrice close qui représente S(z) est algébrique et si z=1/m est un nombre rationnel, le résultat, S(1/m) sera alors un nombre algébrique. C'est précisément ici que l'on utilise l'algorithme LLL. Les centaines de termes que nous donnent la P-récurrence serviront pour évaluer S(z) en un point 1/m "très petit", de telle sorte que le résultat soit un nombre algébrique approché à une grande précision numérique. On ira ensuite chercher avec celui-ci le polynôme dont S(1/m) est racine. Une fois le polynôme candidat trouvé, on réévalue la série S(z) en un autre point rationnel ,1/(m+1), et on répète l'appel à l'algorithme. Il se trouve que la version de LLL sur le programme Pari-GP est extrêmement efficace. Non seulement la procédure (qui s'appelle "algdep") retourne en général le bon polynôme, mais de surcroît il est simplifié au maximum. De plus, les solutions trouvées sont *stables*; elles sont stables au point qu'elles permettent de reconstruire la fonction génératrice algébrique. Une fois ces solutions trouvées en fait, la reconstruction de la fonction génératrice se résume à un calcul d'interpolation assez simple. Comme on l'a mentionné plus tôt, l'appel de la procédure demande un nombre décimal et un degré. Pour arrêter notre choix sur le bon polynôme, il nous suffit de rejeter ceux dont la taille est trop grande (en nombre de caractères). Nous utilisons le même critère que notre méthode des approximants de Padé.

La procédure est la suivante, avec en entrée une suite de la table:

1) On teste si la suite est P-récurrente. Si oui on passe à l'étape 2), sinon on arrête.
2) On calcule plusieurs centaines de termes de la récurrence (dans la pratique 200 termes suffisent).
3) On construit une série S(z) avec ces 200 termes.
4) On évalue la série S(z) en des points rationnels 1/m, 1/(m+1),1/(m+2),... . En pratique m=100 et le nombre de termes = 12.
5) On appelle la procédure "algdep" de Pari-GP avec les 12 valeurs trouvées.
6) On teste avec des polynômes de degré 2,3,4,..., (dans la pratique les degrés 2 à 8 sont suffisants).
7) On récupère les bons polynômes, on pose la variable comme étant x.
8) On identifie les coefficients de même degré et on calcule le polynôme d'interpolation en t avec la méthode de Newton.
9) On substitue t=1/z dans l'expression trouvée.
10) On résout (si le degré de l'expression le permet).

Cet algorithme, quoique très technique, fonctionne très rapidement. Il nous a permis de trouver 32 fonctions génératrices algébriques de degré et de complexité assez élevés.

Illustrons cet algorithme en donnant un exemple.

Exemple 2.3 La suite N0768 des cartes planaires.

Cette suite porte le nom de “Rooted Maps” dans [Sl] mais le titre a été modifié dans [PlSl]. Avec l’étape 1) de notre algorithme, on trouve que la suite satisfait la P-récurrence :

$$(n + 1)\, a_n = (12\, n - 18)\, a_{n-1}.$$

C’est une P-récurrence candidate pour notre méthode hypergéométrique plutôt que pour l’algorithme LLL. On procède donc avec celle-ci et il s’avère que c’est une hypergéométrique:

$$ {}_2F_1([1, 1/2], [3], 12\, z).$$

En demandant à Maple de la simplifier avec “simplify”, celui-ci retourne effectivement une expression algébrique.

Mais cette expression n’est pas très élégante:

```
                                    1/2                1/2
        (1 - 12 z + 24 z I (12 z - 1)    - I (12 z - 1)   ) I
   - 1/9 ------------------------------------------------------
                                  1/2             1/2
                 z (1 + I (12 z - 1)   ) (12 z - 1)
```

On voudrait avoir une expression sans valeurs complexes et simplifiée que l’on obtiendrait de façon automatique. On peut toujours la manipuler à la main, mais notre but est d’obtenir une forme close *automatiquement.* On essaye donc avec une autre méthode: la dérivée et les approximants de Padé.
En dérivant S(z) on obtient une expression qui, mise sous forme d’approximant de Padé, nous donne : (une fois factorisée).

```
                  4        3        2                 2
          (81 z  - 648 z  + 234 z  - 27 z + 1) (9 z  - 9 z + 1)
   - 2 ------------------------------------------------------------
                     3       2                  3       2
      (9 z - 1) (27 z  - 81 z  + 18 z - 1) (81 z  - 81 z  + 18 z - 1)
```

Si on intègre par rapport à z, on devrait retrouver l’expression, mais il y a des polynômes qui sont du 4è degré et la solution n’est pas élégante non plus.

On s'en remet donc à notre méthode LLL.

(étape 1) On reprend la P-récurrence et on recalcule la suite mais avec 200 termes.

(étape 2 et 3). On réévalue la série avec ces mêmes 200 termes et nos points d'interpolation 1/(m+i) avec i=0..4 (5 points d'interpolation devraient suffire)

(étape 4). La première valeur, en m=100, nous donne le premier nombre réel à tester, soit :

1.020958097948815111768685182190012108060775963049210932333987559073395437883368700157841649413257744890532928226947206 8...

(étape 5 et 6) En appelant la procédure "algdep" avec ce nombre réel bon à 118 décimales et un polynôme de degré 2, on obtient, pour les valeurs 1/m, 1/(m+1),1/(m+2),1/(m+3),1/(m+4)

(étape 7) On récupère les bons polynômes:

$$27x^2 + 8200x - 8400$$

$$27x^2 + 8383x - 8585$$

$$27x^2 + 8568x - 8772$$

$$27x^2 + 8755x - 8961$$

$$27x^2 + 8944x - 9152$$

(étape 8) Il nous reste à identifier les coefficients de même degré et à calculer les polynômes d'interpolation correspondants. On aura: pour le coefficient de x^2, les valeurs 27,27,27, ... ,. pour le coefficient de x, les valeurs 8200, 8383, 8568, 8755 et 8944, aux points d'interpolation 100,101,102,103 et 104. Enfin, pour le coefficient constant, on aura les valeurs -8400, -8585, -8772, -8961 et -9152 aux mêmes points d'interpolation. On applique alors simplement la formule d'interpolation de Newton pour trouver une expression polynômiale pour chaque degré. On peut faire appel à la procédure de la librairie Maple appelée "interp" qui effectue ce calcul automatiquement. Ce qui nous donnera deux variables, x et t.

(étape 9) Il restera à substituer t=1/z. On obtient finalement:

$$\frac{-1 + 16z + x^2 - 18xz + 27x^2z^2}{z^2}$$

(étape 10) Il ne reste qu'à résoudre cette équation par rapport à x: on prendra alors la solution positive. Finalement l'expression algébrique de notre suite de départ serait :

$$\frac{1}{54}\,\frac{-1+18z+\left(-(12z-1)^2\right)^{3/2}}{z^2}$$

C'est l'expression la plus simple qu'on ait obtenu pour cette suite. La magie de cet algorithme LLL est qu'il trouve une expression polynômiale pour un nombre réel qui est en général minimale. Des expressions de plus haut degré encore ont été obtenues de cette façon, la plus grosse étant de degré 8 et elles sont répertoriées dans notre table en appendice.

CHAPITRE 3

LA MÉTHODE D'EULER

Ainsi nommée parce qu'elle semble avoir été développée à l'époque d'Euler. Nous n'avons pas trouvé de références historiques sur cette méthode, bien que Andrews [And] la mentionne.

L'idée en est simple: étant donné une suite a_n dont on suppose la série génératrice de la forme,

$$S(z) = 1 + \sum_{n=1}^{\infty} a_n z^n = \prod_{n=1}^{\infty} (1 - z^n)^{-c_n}, \tag{3.1}$$

la question est : comment trouver les c_n en fonction des a_n. Comme l'explique Andrews à la page 104, il suffit d'utiliser la formule d'inversion de Möbius. En effet, puisque le membre de droite de (3.1) est un produit infini, c'est en prenant le logarithme ou la dérivée logarithmique que nous retrouvons alors une somme ordinaire. En identifiant le coefficient de degré n (pour exprimer chaque coefficient de a_n) et en inversant (par Möbius) par rapport à la somme, nous obtenons les coefficients c_n en fonction des a_n. La somme s'exprimera en termes des diviseurs de n. Inversement, si nous connaissons les c_n et que l'on cherche les a_n , l'opération est directe; il suffit de développer le produit en série. En prenant soin de garder le même ordre de grandeur des séries correspondantes, nous obtenons le même nombre de termes pour les c_n que pour les a_n. Autrement dit, si les k premiers coefficients de a_n sont connus, il y aura alors k coefficients de bons pour les c_n.

On peut donc programmer la transformation dans les deux sens en une vingtaine de lignes. La procédure accepte en entrée une suite et donne du même coup une représentation en "partages", c'est-à-dire qu'elle propose un produit infini. Par exemple, la suite N0244 énumère les partages

ordinaires de l'entier n. En effectuant le calcul on trouve la suite 1,1,1,1,1,.... C'est la forme de produit infini de ce type la plus simple. Mais pour détecter un bon candidat de produit infini avec ce type de fonction génératrice, dans un cadre plus général, nous avons utilisé la méthode des approximants de Padé qui permet de détecter les "motifs" dans les exposants. En tout, 94 produits infinis ont ainsi été isolés grâce à cette méthode. Les résultats sont présentés dans la table en appendice.

CHAPITRE 4

LA MÉTHODE DES RECOUPEMENTS

4.1 Les recoupements indirects.

L'hypothèse que l'on pose ici est que la suite dont on cherche la fonction génératrice est en fait une suite connue mais transformée. Par exemple, la suite des partages d'entiers N0244 de [Sl] est très facile à détecter. Il suffit de prendre la méthode d'Euler et le programme nous propose immédiatement un produit infini très simple. Mais si on effectue la translation a_n+3, le programme ne détectera pas ce produit infini. Pour une bonne raison car, si la suite ne commence pas naturellement à 1, alors l'opération d'Euler n'est pas valide et même si on l'effectue, les termes seront des nombres rationnels (non entiers). Donc afin de pouvoir isoler le plus possible de suites, on se sert, comme base de comparaison, de la table des suites qui en contient 4568. En prenant chaque suite transformée de façon élémentaire, on compare avec la table afin de voir s'il n'y aurait pas un croisement. En tout, nous avons répertorié 97 transformations élémentaires d'une suite susceptibles de se retrouver dans la table, soit 54 transformations avec la suite sous forme de série ordinaire et 43 avec la suite sous forme de série exponentielle.

Il serait fastidieux de les énumérer toutes, mais en voici quelques unes. Avec S(z) : la suite sous forme de série ordinaire par exemple, nous avons $S(z) + cz/(1-z)$ ou $c=\pm1,\pm2,\pm3$, $1/S(z)$, $S(z)^2$, $S(z)^3$, $S(z)/(1-z)$ ce qui équivaut à considérer la suite des sommes partielles de la suite. D'autres transformations sont plus simples encore, comme $\mathbf{N} \setminus \{a_n\}$, la différence ensembliste des entiers et de la suite. On ne tient pas compte ici de la multiplicité des termes. On a considéré aussi de prendre $a_n/\text{pgcd}\ (a_0, a_1, a_2,\ldots a_k)$ ou de prendre la suite avec les indices de rang pairs et impairs. L'idée est de prendre des transformations les plus simples possibles. La transformation d'Euler dans les 2 sens complète la liste.

On pourrait les classer en ces quelques catégories :

1) Translations : S(z) +/- cz/(1-z), avec c=1,2,3.

2) Inverses : $1/S(z), 1/S(z)^2, 1/S(z)^3$.

3) Puissances : $S(z)^k$ avec k=1,2,3.

4) Sommes et différences.

5) Transformation de type Euler (voir chapitre 3).

6) Transformations de type ensembliste comme $\mathbf{N} \setminus \{a_n\}$.

7) Transformations avec le p.g.c.d. .

Les autres sont données en considérant des combinaisons de ces dernières.

Par exemple, de la suite N0577 de [Sl] (les nombres de Catalan), on en obtient 97 autres et en comparant ces 97 suites avec la table, 6 autres suites au moins seraient liées à cette dernière. C'est donc que, si on connait déjà la fonction génératrice des nombres de Catalan obtenue avec d'autres méthodes, alors du même coup on obtient la fonction génératrice de ces 6 autres suites. C'est un avantage, parce que justement avec cet exemple, si l'on prend a_n-1 et que l'on compare avec la table, on retrouve la suite N1409 de [Sl]. Cette suite n'est pas hypergéométrique en vertu d'un critère assez simple de [GKP], elle ne commence pas par 1. De plus son inverse fonctionnel est impossible à effectuer pour le même genre de raisons; le premier terme est nul mais le deuxième terme n'est pas 1. Elle est cependant algébrique et c'est avec la méthode LLL (beaucoup plus lourde) que la fonction génératrice a été trouvée. En fait, elle est évidemment de la forme S(z) -1/(1-z) où S(z) est la fonction génératrice des nombres de Catalan. Mais ceci constitue un raisonnement a posteriori. Donc cette méthode des recoupements peut mener à des résultats très intéressants en autant que le traitement informatique des 97 transformations appliquées aux 4568 suites et comparées avec ces dernières à chaque fois ne soit pas trop lourd également.

Un détail ne doit cependant pas être oublié. La comparaison de 2 suites entre elles peut mener à des erreurs. On doit faire la comparaison à partir du deuxième terme, parce que souvent la suite est répertoriée mais les premiers termes peuvent être d'indices 0 ou 1. C'est-à-dire que la suite ne débute pas au terme de rang 0. Également on ne doit pas prendre toute la suite: il ne faut pas oublier que certaines suites de la table sont très courtes et ne contiennent que quelques termes. Elles ne sont pas moins importantes, par exemple la suite N0323 de [Sl]. Il y a un juste milieu et l'expérience montre que les

indices de rang 2 à 16 sont suffisants, c'est-à-dire les 15 premiers termes de la suite à partir du rang 2.

A cet effet un programme appelé HIS (Handbook of Integer Sequences) a été mis au point. Il n'est cependant pas public comme le programme gfun. Dans HIS se trouve la table numérique des suites et 2 procédures appelées "find" et "findhard". La première sert simplement à savoir si une suite se trouve dans la table et la deuxième fait une recherche dans la table après avoir effectué les 97 transformations en question. Le programme et la table sont entièrement contenus en Maple. Le programme est donc de cette façon transportable sur toute machine qui peut recevoir Maple.

La procédure "find" qui en principe ne fait que regarder si une suite se trouve dans la table emploie une procédure de recherche mise au point par Bruno Salvy de l'INRIA. Il était essentiel d'avoir à notre disposition un algorithme de recherche qui soit très rapide étant donné le nombre important de comparaisons à chaque opération. Une structure de données adaptée à ces besoins a été construite sous forme d'arbre binaire. En effectuant une boucle de calcul sur toute la table avec la procédure "findhard", une banque de données des croisements a été obtenue. En tout il y aurait 3800 croisements. Une proportion appréciable des ces croisements, soit environ 25% selon nous, est fortuite ou accidentelle. Ceci est relié à la décision de ne prendre qu'une partie de chaque suite pour comparer. Donc pour pouvoir retrouver la fonction génératrice, il y a un travail de vérification nécessaire.

Ce travail de vérification est très long, mais il en vaut la peine. Evidemment, beaucoup de croisements ne sont pas surprenants: à titre d'exemple, la suite de Fibonacci, qui est très connue et qui a une fonction génératrice assez simple croise avec une bonne centaine de suites. Aucun de ces croisements n'est vraiment nouveau. C'est lorsque la suite est intrinsèquement plus complexe que le jeu en vaut la chandelle. Sans exagérer, nous avons effectué patiemment des centaines d'heures de calcul et de vérification pour trouver ces résultats et il y en a beaucoup à faire encore, puisque cette table des 3000 bons croisements environ n'a pas été passée en revue au complet. Par ce procédé, 38 fonctions génératrices ont été obtenues. Elles sont répertoriées dans la table en appendice.

4.2 Les tableaux.

Le programme Maple manipule des données numériquement aussi bien que symboliquement. La procédure “ratpoly” est capable de trouver une fraction rationnelle de séries à une variable aussi bien qu’à 2 variables comme les tableaux à 2 dimensions. Un bon exemple est le triangle de Pascal. Il suffit de le mettre sous forme de tableau “carré” où chaque rangée sera un polynôme. En prenant les 5 premières rangées, on aura la suite

$$1,\ 1 + t,\ 1 + 2\,t + t^2,\ 1 + 3\,t + 3\,t^2 + t^3,\ 1 + 4\,t + 6\,t^2 + 4\,t^3 + t^4\ .$$

Si cette suite (de polynômes) est maintenant convertie en série de puissances en z et passée à la procédure “ratpoly”, elle retourne immédiatement

$1/(1 - t\,z - z)$. Si on développe en série par rapport à t, on obtient la fonction génératrice de chaque colonne et inversement, en développant par rapport à z, on obtient la fonction génératrice de chaque rangée (qui sont ici des polynômes). Notons que 4 termes suffisent pour trouver la fonction génératrice du tableau.

Contrairement aux autres méthodes, il n’existe pas de livre ou de catalogue de tels tableaux. Il y en a un bon nombre dans la littérature, mais ce qui a été fait plutôt est d’en générer de façon *ad hoc* . Il faut prendre un modèle de tableaux assez général, par exemple dans [GKP] ou [Théo], où on introduit les tableaux $A_{[n,k]}$ définis par la relation de récurrence

$$A_{[n+1,k+1]} = (r\,n+s\,k+t)\,A_{[n,k+1]} + (a\,n+b\,k+c)\,A_{[n,k]}$$

où a,b,c,r,s et t sont entiers. Il se trouve qu’une bonne partie des tableaux étudiés en combinatoire sont de ce type: les coefficients binomiaux, les nombres de Stirling de 1ère et de 2ème espèce, les nombres eulériens, les coefficients des polynômes de Tchébycheff, etc. On peut consulter [Théo] à ce sujet où une étude approfondie de ces tableaux a été menée. Il reste donc à en générer un bon nombre en prenant les entiers a,b,c,r,s et t compris entre -4 et 4 et de *tenter* de trouver la fonction génératrice. Sur des milliers tableaux générés de cette façon, 430 fonctions génératrices à deux variables ont été trouvées, couvrant la plupart des cas simples de ces tableaux. On obtient ainsi un échantillonnage assez important de formules, suffisamment important pour y trouver la fonction génératrice de centaines de suites de notre table. En tout, 20 nouvelles fonctions génératrices ont été isolées. Ces résultats sont

présentés dans la table en appendice.

CONCLUSION

On conclut que nos méthodes peuvent dans 23 % des cas donner la fonction génératrice d'une suite d'entiers "quelconque". Le mot quelconque signifie ici: ce qui est catalogué dans la table de suites [PlSl]. Nous croyons qu'il en est de même avec toute suite d'entiers qui se présente au mathématicien dans ses recherches, quel que soit son domaine. Nous souhaitons que ces méthodes deviennent des outils de travail.

Il reste cependant beaucoup à faire. Il faut trouver une explication raisonnable au fait que nous sommes passés à côté de 77% des suites. On pourrait peut-être étendre encore les méthodes en formulant d'autres modèles de fonctions génératrices. En fait, il en existe déjà. Par exemple, la fonction "plancher" ou partie entière permet de construire des suites très simples que nos méthodes n'ont pas détectées; la suite $[(3/2)^n]$ en est un bon représentant. On pourrait également mettre dans la même catégorie les suites définies avec des nombres irrationnels comme $[\sqrt{2}n]$. Un autre modèle pourrait être basé sur les récurrences quadratiques comme la suite 2,4,16,256,... (en mettant au carré à chaque fois). Elle est extrèmement simple mais indétectable par nos méthodes. Un autre serait basé sur les suites "doublement" récurrentes, là où il y a une fonction de l'indice comme $a_{a(n)}$. On pourrait multiplier les exemples de suites très simplement définies mais indétectables. Ce qui caractérise une table de suites comme [Sl] ou [PlSl], c'est la variété et c'est précisément ce qui nous passionne.

BIBLIOGRAPHIE

[AABBJPS] J.P. Allouche, A. Arnold, J. Berstel, S. Brlek, W. Jockusch, S. Plouffe, B. Sagan, *A Sequence related to that of Thue-morse*, preprint 1992. Suite A3159.

[And] G.E. Andrews, *q-Series : Their development and application in analysis, number theory, combinatorics, physics, and computer algebra.* Regional Conference Series in Mathematics, number 66. Providence, 1986. AMS Publication.

[AS1] M. Abramowitz and I. A. Stegun, Handbook of Mathematical Functions, National Bureau of Standards, Washington DC, 1964; Dover, NY, 1965.

[BaKa] A. Bachem, R. Kannann, *Lattices and the basis reduction algorithm,* Carnegie Mellon University, rapport interne. 1984.

[BP] F. Bergeron, S. Plouffe, *Computing the generating function of a serie given it's first terms*, Rapport de recherche #164, Université du Québec à Montréal, octobre 1991.

[gfun] F. Bergeron, S. Plouffe, B. Salvy, P. Zimmermann, Programme gfun en MapleV de la librairie partagée et publique de Maple. Disponible par transfert électronique à l'Université de Waterloo. Version Juin 1992.

[GKP] R. L. Graham, D. E. Knuth and O. Patashnik, Concrete Mathematics, Addison-Wesley, Reading, MA, 1990.

[LLL] A.K. Lenstra, H.W. Lenstra, et L. Lovàsz, *Factoring Polynomials with rational coefficients,* Mathematische Annalen 261 (1982), pages 513-534.

[M5] B.W. Char, K.O. Geddes, G.H. Gonnet, B.L. Leong, M.B. Monagan, S.M. Watt, MAPLE V Library Reference Manual, Springer Verlag, (1991), Waterloo Maple Publishing.

[Pari] C. Batut, D. Bernardi, H. Cohen, M. Olivier, User's guide to PARI-GP, Version 1.36, Université Bordeaux I, document interne, 8 Décembre 1991.

[Plsl] S. Plouffe, N.J.A. Sloane, *The New Book of Integer Sequences*, preprint 1992, titre provisoire.

[Sl] N.J.A. Sloane, *A Handbook of Integer Sequences*, Academic Press, New York, 1973.

[Sta80] R. Stanley, *Differentiably finite power series*, European Journal of Combinatorics, vol. 1,(1980), p.175-188.

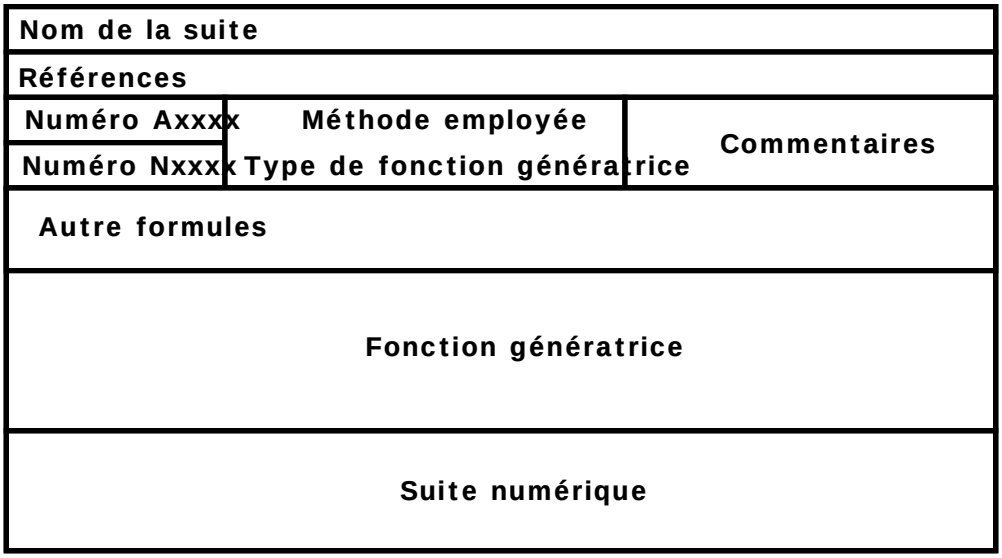
[Théo] P. Théoret, Thèse de Ph. D., "*Etude des doubles suites définies par une récurrence du premier degré*", Université du Québec à Montréal, preprint 1992.

A.0 NOTES À L'UTILISATEUR DE LA TABLE

Chaque fonction génératrice trouvée à l'aide de l'une de nos méthodes est répertoriée dans la table qui suit sous forme de fiche. Chaque fiche contient les informations pertinentes à cette suite :

- Numéro séquentiel Axxxx et Nxxxx (s'il existe)
- Nom de la suite
- Les références bibliographiques avec dans l'ordre : Périodique Volume Page Année
- La méthode employée pour trouver la fonction génératrice
- Le type de fonction génératrice
- Commentaires additionnels
- Autres formules connues ou trouvées
- La fonction génératrice
- La suite numérique

Elles apparaissent selon le schéma suivant:



- Les références bibliographiques sont notées exactement comme dans le livre [Sl]. La liste des ouvrages se trouve dans une bibliographie séparée à la fin de la table.
- La fonction génératrice qui apparaît au centre est toujours une fonction génératrice ordinaire à moins qu'il en soit indiqué autrement (exponentielle ou double exponentielle).
- Les fiches ont été triées par ordre numérique sur les numéros Axxxx. Cette table est une pile Hypercard. On peut donc l'utiliser sur tout ordinateur Macintosh et la consulter comme une banque de donnée. Nous prévoyons un accès à Maple. De cette façon l'utilisateur pourra vérifier chaque formule.
- W(z) désigne la fonction Oméga, définie implicitement par W(z) exp(W(z)) = z. On la connait aussi sous

sa forme de série exponentielle dont les coefficients sont donnés, en valeur absolue, $|a_0| = 0$, $|a_n| = n^{n-1}$ pour $n>0$ (série alternante à terme constant nul). Son rayon de convergence est 1/e et elle est souvent utilisée pour le développement en série de certaines fonctions génératrices de structures arborescentes. Elle est très commode dans les calculs.

• La fonction génératrice qui apparaît au centre de chaque fiche est la plus simple ou plus élégante expression que nous connaissons donnant les termes de la suite.

• Le nom de chaque suite (s'il est présent) est tel qu'il apparaît dans [PlSl]. Quand il est omis c'est qu'il est d'une forme que nous jugeons redondante par rapport à la fonction génératrice.

• Les P-récurrences qui apparaissent dans la case "fonction génératrice" ou "autres formules" ont leur conditions initiales données par les premiers termes de la suite.

• La suite qui apparaît dans la case "suite numérique" est telle qu'elle apparaît dans [PlSl], plus de termes peuvent être évidemment obtenus avec la fonction génératrice.

• Certaines fiches ont été imprimées en format pleine grandeur pour plus de lisibilité