Extrapolation, Méthodes de Lanczos et Polynômes Orthogonaux: Théorie et Conception de Logiciels

Michela REDIVO ZAGLIA

REMERCIEMENTS

J'exprime mes remerciements à Jean Della Dora, Professeur à l'Université Scientifique et Médical de Grenoble, pour avoir accepté de présider ce jury.

Je tiens à exprimer toute ma gratitude à Claude Brezinski, Professeur à l'Université des Sciences et Technologies de Lille, pour ses conseils précieux, pour avoir bien voulu me suivre, me guider et m'encourager dans mes travaux de recherche et surtout pour m'avoir permis de travailler avec lui et de réaliser cette thèse.

Un remerciement particulier à Maria Morandi Cecchi, Professeur à l'Università di Padova, pour m'avoir introduite et poussée à la recherche, pour son soutient continu, pour l'aide qu'elle a bien voulu me donner et pour tout le travail qu'on a fait ensemble.

Mes remerciement vont également à Florent Cordellier, Professeur à l'Université de Lille III, Jeannette van Iseghem, Professeur à l'USTL et Bernard Germain-Bonne, Maître de Conférences à l'USTL pour avoir accepté de juger ce travail.

Je remercie Annie Cuyt, Directeur de recherche au FNRS, d'avoir bien voulu être rapporteur de ce travail.

Un grand merci à toute l'équipe d'Analyse Numérique et d'Optimisation de l'Université de Lille et, en particulier à Hassane Sadok, pour leur amicale collaboration qui a été précieuse pour la concrétisation de ce travail.

Sono particolarmente debitrice a mio marito Renzo ed a mio figlio Enrico per la pazienza ed il sostegno morale di tutti questi anni che solo ha permesso di raggiungere questo importante traguardo.

INTRODUCTION

Tous les travaux composant cette thèse ont été publiés sous forme d'articles et d'un livre avec une disquette contenant les logiciels (voir bibliographie). Plutôt que de reprendre ici dans le détail l'ensemble des résultats obtenus, il nous a semblé plus intéressant d'en donner une vue d'ensemble qui permet de mieux comprendre la portée des résultats obtenus, de les situer les uns par rapport aux autres et de montrer leur apport et leur originalité.

Les recherches réalisées couvrent trois domaines

- méthodes d'extrapolation
- polynômes orthogonaux
- méthodes de type Lanczos.

Dans ces trois domaines nous nous sommes intéressés aux questions de

- théorie
- applications
- conception de logiciels.

Le premier chapitre traite des méthodes d'extrapolation. Nous avons d'abord commencé par montrer comment construire de nouvelles méthodes d'accélération de la convergence basées sur les informations concernant l'erreur de la suite à extrapoler. Ensuite nous nous sommes intéressés à l'instabilité numérique du Θ-algorithme pour lequel nous avons donné les règles particulières. Enfin la prédiction des termes inconnus d'une suite à été étudiée et appliquée à un problème d'équations aux dérivées partielles.

Le second chapitre est consacré aux polynômes orthogonaux dont on donne d'abord une nouvelle présentation ayant plusieurs avantages. Ensuite les moments modifiés sont utilisés pour la mise en œuvre de certaines formules de quadrature de Gauss.

L'objet du troisième chapitre est la méthode de bordage et la méthode de bordage par blocs pour résoudre les systèmes d'équations linéaires. Quand un saut se produit dans cette méthode, les solutions des systèmes intermédiaires qui ont été sautées ne sont pas calculées. On propose une nouvelle méthode, la méthode de bordage inverse, qui permet de revenir en arrière pour les obtenir.

Le chapitre quatre traite de la méthode de Lanczos. Une division par zéro peut se produire dans les algorithmes qui mettent en œuvre la méthode de Lanczos pour les

systèmes d'équations avec matrice quelconque. Il faut alors arrêter l'algorithme, c'est un breakdown. Celui ci est dû à la non-existence de certains polynômes orthogonaux. Pour éviter ce problème nous avons proposé de nouveaux algorithmes sans breakdown qui consistent à sauter au dessus des polynômes orthogonaux qui n'existent pas pour ne calculer que ceux qui existent. Quand on divise par un nombre proche de zéro, des erreurs d'arrondis importantes peuvent entâcher les algorithmes; c'est le near-breakdown qui a été résolu de façon analogue. Les mêmes remèdes s'appliquent également à l'algorithme CGS.

Les logiciels correspondants à toutes les questions développées dans ce travail ont été écrits (en FORTRAN 77) en mettant l'accent sur leur portabilité. Ils sont disponibles.

TABLE DES MATIÈRES

Chapitre 1

MÉTHODES D'EXTRAPOLATION

En analyse numérique et en mathématiques appliquées on a souvent une suite de valeurs obtenues par différents algorithmes et cette suite converge vers la solution exacte S du problème. Par exemple quand on construit des approximations qui dépendent d'un paramètre h (normalement le pas comme dans les méthodes de quadrature ou les méthodes pour les ODE et les PDE), dans les méthodes de perturbation, dans les méthodes de point fixe (en général dans les procédés itératifs), ou dans les sommes partielles des séries convergentes.

Souvent la convergence de la suite (S_n) vers S est lente et donc on a besoin d'accélérer sa convergence. Cela bien souvent peut se faire avec des *méthodes d'extrapolation.*

On connaît beaucoup de méthodes d'extrapolation et il y a beaucoup de domaines de l'analyse numérique dans lesquels on peut appliquer ces méthodes. Nous avons traité tout cela dans un livre qui vient de paraître et qui s'appelle *Extrapolation Methods. Theory and Practice* [14].

Si nous prenons une suite (S_n) on appelle *transformation de suite* une application T telle que

$$T : (S_n) \longrightarrow (T_n).$$

La plupart des transformations de suites T peut se mettre sous forme d'un rapport de déterminants comme cela été démontré par Brezinski et Walz [17]. Par exemple le procédé Δ^2 d'Aitken [1], la transformation de Shanks [40], le E-algorithme ou protocole BH obtenu séparement par Håvie [27] et Brezinski [4].

Dans la transformation de Shanks [40] on a

$$T_n = e_k(S_n) = \frac{\begin{vmatrix} S_n & S_{n+1} & \cdots & S_{n+k} \\ \Delta S_n & \Delta S_{n+1} & \cdots & \Delta S_{n+k} \\ \vdots & \vdots & & \vdots \\ \Delta S_{n+k-1} & \Delta S_{n+k} & \cdots & \Delta S_{n+2k-1} \end{vmatrix}}{\begin{vmatrix} 1 & 1 & \cdots & 1 \\ \Delta S_n & \Delta S_{n+1} & \cdots & \Delta S_{n+k} \\ \vdots & \vdots & & \vdots \\ \Delta S_{n+k-1} & \Delta S_{n+k} & \cdots & \Delta S_{n+2k-1} \end{vmatrix}}.$$

Si les déterminants qui paraîssent dans la définition de T_n ont une forme particulière,

alors il est possible, en appliquant des identités de déterminants au numérateur et au dénominateur, d'obtenir des règles pour calculer récursivement ces rapports, sans calculer les déterminants. Ces règles sont appelées un *algorithme d'extrapolation.*

Pour la transformation de Shanks, on a l' ε-algorithme de Wynn [46] qui est bien connu

$$\begin{aligned} \varepsilon_{-1}^{(n)} &= 0\,, \quad \varepsilon_0^{(n)} = S_n\,, \quad n = 0,1,\ldots \\ \varepsilon_{k+1}^{(n)} &= \varepsilon_{k-1}^{(n+1)} + \frac{1}{\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}}\,, \quad k,n = 0,1,\ldots \end{aligned}$$

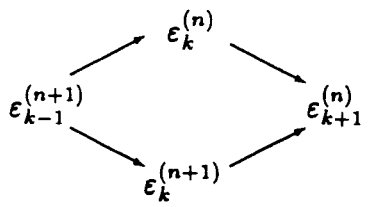
et l'on a $\varepsilon_{2k}^{(n)} = e_k(S_n)$.

Les quantités calculées avec un algorithme d'extrapolation peuvent être placées dans un tableau comme le suivant (dans le cas de l' ε-algorithme)

$$\begin{array}{lllllll}
\varepsilon_{-1}^{(0)} = 0 & & & & & & \\
 & \varepsilon_0^{(0)} = S_0 & & & & & \\
\varepsilon_{-1}^{(1)} = 0 & & \varepsilon_1^{(0)} & & & & \\
 & \varepsilon_0^{(1)} = S_1 & & \varepsilon_2^{(0)} & & & \\
\varepsilon_{-1}^{(2)} = 0 & & \varepsilon_1^{(1)} & & \varepsilon_3^{(0)} & & \\
\vdots & \varepsilon_0^{(2)} = S_2 & & \varepsilon_2^{(1)} & & \ddots & \\
\vdots & \vdots & \varepsilon_1^{(2)} & & \varepsilon_3^{(1)} & & \\
\vdots & \vdots & \vdots & \varepsilon_2^{(2)} & & \ddots & \\
\vdots & \vdots & \vdots & \vdots & \varepsilon_3^{(2)} & & \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots &
\end{array}$$

A partir des premières deux colonnes, $\left(\varepsilon_{-1}^{(n)} = 0\right)$ et $\left(\varepsilon_0^{(n)} = S_n\right)$, la règle de l'$\varepsilon$-algorithme permet de progresser, dans le calcul, de gauche à droite et du haut en bas (forme normale d'un algorithme d'extrapolation).

La règle de l'ε-algorithme contient des quantités qui se trouvent aux quatre sommets d'un losange (il y a beaucoup d'algorithmes d'extrapolation qui se trouvent dans la même situation et donc on les nomme souvent algorithmes de losange)



Dans l'ε-algorithme on sait que les nombres $\varepsilon_{2k}^{(n)}$ sont des approximations de la limite S de la suite (S_n) et que les $\varepsilon_{2k+1}^{(n)}$ sont des résultats intermédiaires. Puisque l'on n'est pas intéressé par les quantités auxiliaires $\varepsilon_{2k+1}^{(n)}$, on peut utiliser la *règle de la croix* de Wynn [49] qui contient seulement les quantités avec un indice inférieur pair.

Si nous posons

$$N = \varepsilon_{2k}^{(n)}$$

$$W = \varepsilon_{2k-2}^{(n+2)} \quad C = \varepsilon_{2k}^{(n+1)} \quad E = \varepsilon_{2k+2}^{(n)}$$

$$S = \varepsilon_{2k}^{(n+2)}$$

la règle de la croix est

$$(N-C)^{-1} + (S-C)^{-1} = (W-C)^{-1} + (E-C)^{-1}$$

c'est à dire que E peut se calculer par

$$E = C + \left[(N-C)^{-1} + (S-C)^{-1} - (W-C)^{-1}\right]^{-1} \tag{1.1}$$

avec $\varepsilon_{-2}^{(n)} = \infty$.

Dans les méthodes d'extrapolation, on désire que chaque colonne du tableau contienne une suite qui converge vers S plus vite que la suite initiale.

1.1 Construction de procédés d'extrapolation

Une méthode d'extrapolation donnée n'est capable d'accélérer la convergence que de suites dont l'erreur $S_n - S$ satisfait à certaines conditions car, comme il a été démontré par Delahaye et Germain-Bonne [22], il ne peut pas exister d'algorithme d'accélération capable d'accélérer n'importe quelle suite.

La méthode d'extrapolation la plus générale et la plus intéressante, quand on connaît un développement asymptotique de l'erreur, est le E-algorithme [4, 27] (voir [11] pour un exposé général). Cependant, dans la pratique, bien souvent, on ne connaît pas le développement asymptotique de l'erreur et donc il est très difficile de choisir la méthode la meilleure entre les méthodes d'extrapolation connues. C'est pour cela que nous avons pensé, au lieu d'utiliser une méthode d'extrapolation connue, d'en fabriquer une exprès selon les informations connues sur l'erreur [13].

Soit donc (S_n) une suite convergente vers S telle que $\forall n$

$$S_n - S = a_n g_n$$

où (a_n) et (g_n) sont des suites connues ou non et qui peuvent eventuellement dépendre de la suite (S_n). Naturellement, puisque (S_n) converge vers S, alors $(a_n g_n)$ converge vers zéro.

Supposons que (g_n) converge vers zéro mais qu'on ne sache rien de la convergence de (a_n) (ce qui n'enlève rien à la généralité du discours).

C'est par exemple le cas dans les situations suivantes

$S_n - S = O(g_n)$	qui correspond à $\lvert a_n\rvert \leq M, \forall n.$
$S_n - S = o(g_n)$	qui correspond à $\lim_{n\to\infty} a_n = 0.$
$S_n - S = a_1 g_1(n) + a_2 g_2(n) + \cdots$	qui correspond à $g_n = g_1(n)$ et $a_n = a_1 + a_2 g_2(n)/g_1(n) + \cdots = a_1 + \varepsilon_n$ avec $\lim_{n\to\infty} \varepsilon_n = 0.$
$S_n = c_o f_o + \cdots + c_n f_n$	qui correspond à $g_n = -c_{n+1}$ ou $g_n = -f_{n+1}$ ou $g_n = -c_{n+1} f_{n+1}$.

La première situation a été traitée par Szabo [44] qui a utilisé une schéma d'extrapolation similaire au nôtre.

Maintenant nous allons montrer comment construire, à partir de la suite (S_n) et avec certaines hypothèses, une nouvelle suite (T_n) qui converge vers S plus vite que (S_n), c'est à dire telle que

$$\lim_{n\to\infty} (T_n - S)/(S_n - S) = 0.$$

Les trois cas possibles sont

1. (a_n) et (g_n) connues
2. (a_n) inconnue et (g_n) connue (ou viceversa)
3. (a_n) et (g_n) inconnues.

Le premier cas n'est pas intéressant car $S = S_n - a_n g_n$ pour tout n. Les deux autres cas sont à étudier et c'est évident que la construction sera possible seulement si l'on connaît certaines informations sur la suite (a_n) ou sur les deux suites (a_n) et (g_n).

Trois des transformations que nous allons proposer ont la forme

$$T_n = \frac{S_{n+1} - b_n S_n}{1 - b_n}, \qquad n = 0, 1, \ldots$$

Si (S_n) est strictement monotone il est intéressant de connaître les conditions qu'il faut imposer à la suite (b_n) pour que (T_n) soit aussi monotone dans la même direction que (S_n) ou dans la direction contraire. Dans ce dernier cas si la monotonie est renversée, alors on peut obtenir les intervalles qui contiennent S. Ces conditions ont été données par Opfer [37] et d'autres méthodes pour construire des intervalles qui contiennent S peuvent être trouvées dans [7].

(a_n) inconnue, (g_n) connue

Dans ce cas nous avons proposé [13] trois constructions possibles d'après les informations sur (a_n) que l'on connaît.

Premier procédé d'extrapolation

Dans le premier cas, soit P un opérateur linéaire sur les suites qui transforme une suite (u_n) dans une suite $(v_n = P(u_n))$ et tel que $\forall(u_n), \forall a$ et $\forall b$ on a

$$P(au_n + b) = aP(u_n) + b \quad \text{pour} \quad n = 0, 1, \ldots$$

On suppose que pour cet opérateur P on a $\forall n, P(a_n) = 0$.

On sait que

$$(S_n - S)/g_n = a_n$$

et donc si on applique P aux deux membres on aura pour tout n

$$P((S_n - S)/g_n) = P(a_n) = 0$$

et, à cause de la linéarité de P

$$S = P(S_n/g_n)/P(1/g_n). \tag{1.2}$$

Cette approche a été suivie par Weniger [45] pour la construction de procédés d'extrapolation et si on arrive à connaître cet opérateur P on arrive au résultat exact S. Mais dans la pratique il n'est pas facile de trouver ce P et on arrive à le faire seulement dans des cas particuliers (par exemple si a_n est un polynôme de degré k en n, on peut prendre $P = \Delta^{k+1}$) et non dans le cas général.

Deuxième procédé d'extrapolation

Maintenant supposons connaître seulement une suite (b_n) qui est une approximation de (a_n).

Posons

$$T_n = S_n - b_n g_n, \qquad n = 0, 1, \ldots$$

Evidemment on a

$$(T_n - S)/(S_n - S) = 1 - b_n/a_n$$

et donc le théorème suivant

Théorème 1.1 *Une condition nécessaire et suffisante pour que (T_n) converge vers S plus vite que (S_n) est que $\lim_{n\to\infty} b_n/a_n = 1$.*

Donc dans ce cas il est assez facile d'accélérer la convergence de (S_n) si la suite (b_n) satisfait la condition du théorème (1.1).

Troisième procédé d'extrapolation

Si (b_n) n'est pas connue, on peut essayer de la construire. Donc on considère la fonction linéaire $f(x) = ax + b$ telle que $S_n = f(g_n)$ et $S_{n+1} = f(g_{n+1})$ et on va poser $T_n = f(0)$ c'est à dire que l'on extrapole en zéro et l'on pose

$$T_n = S_n - \frac{\Delta S_n}{\Delta g_n} g_n, \qquad n = 0, 1, \ldots$$

ce qui correspond au choix $(b_n = \Delta S_n / \Delta g_n)$.

Cette transformation est identique à la première colonne du E-algorithme avec $g_1(n) = g_n$ [4, 27] ou à la Θ-procédure [6] et nous avons

$$T_n - S = -\frac{\Delta a_n}{\Delta g_n} g_n \, g_{n+1}$$

et donc

$$\frac{T_n - S}{S_n - S} = \frac{1 - a_{n+1}/a_n}{1 - g_n/g_{n+1}} \qquad \text{et} \qquad \frac{T_n - S}{S_{n+1} - S} = \frac{1 - a_n/a_{n+1}}{1 - g_{n+1}/g_n}. \tag{1.3}$$

Il faut considérer séparement deux cas selon le fait que la suite (g_{n+1}/g_n) est logarithmique (c'est à dire $\lim_{n\to\infty} g_{n+1}/g_n = 1$) ou non logarithmique ($\exists \alpha < 1 < \beta, \exists N, \forall n \geq N, g_{n+1}/g_n \notin [\alpha, \beta]$ ce qui est équivalent à $g_n = O(\Delta g_n)$). Dans le cas non logarithmique, il rentre le cas particulier où $\lim_{n\to\infty} g_{n+1}/g_n = 0$ que nous avons traité séparément pour avoir le résultat le meilleur.

Cas non-logarithmique

Théorème 1.2 *Si (g_n) est non logarithmique et si $\exists 0 < m \leq M$, $\exists N$ tel que $\forall n \geq N$, $m \leq |g_{n+1}/g_n| \leq M$ alors une condition nécessaire et suffisante pour que (T_n) converge vers S plus vite que (S_n) et (S_{n+1}) est que $\lim_{n\to\infty} a_{n+1}/a_n = 1$.*

C'est évident que s'il existe une constante $a \neq 0$ telle que (a_n) converge vers a, alors la condition du théoreme 1.2 est satisfaite.

Encore, puisque (1.3) peut s'écrire

$$\frac{T_n - S}{S_n - S} = -\frac{\Delta a_n}{a_n} \cdot \frac{1}{1 - g_n/g_{n+1}} \qquad \text{et} \qquad \frac{T_n - S}{S_{n+1} - S} = -\frac{\Delta a_n}{a_{n+1}} \cdot \frac{1}{g_{n+1}/g_n - 1}$$

alors on peut donner la condition du théorème 1.2 par rapport à la limite de Δa_n (au lieu de a_{n+1}/a_n). Cependant les deux résultats ne sont pas équivalents car $\lim_{n\to\infty} a_{n+1}/a_n = 1$ n'implique pas que $\lim_{n\to\infty} \Delta a_n = 0$ et viceversa. Si $\lim_{n\to\infty} a_{n+1}/a_n = 1$ et si $\exists M$, $\exists N$ tel que $\forall n \geq N$, $|a_n| \leq M$ alors $\lim_{n\to\infty} \Delta a_n = 0$. Viceversa si $\lim_{n\to\infty} \Delta a_n = 0$ et si $\exists m$, $\exists N$ tel que $\forall n \geq N$, $m \leq |a_n|$ alors $\lim_{n\to\infty} a_{n+1}/a_n = 1$. On peut aussi utiliser le fait que si $\lim_{n\to\infty} |a_n| = \infty$ et si $\exists M$, $\exists N$ tel que $\forall n \geq N$, $|\Delta a_n| \leq M$ alors $\lim_{n\to\infty} a_{n+1}/a_n = 1$.

Nous allons maintenant considérer le cas particulier où $\lim_{n\to\infty} g_{n+1}/g_n = 0$. On aura

Théorème 1.3 *Si* $\lim_{n\to\infty} g_{n+1}/g_n = 0$ *et si* $\exists M$, $\exists N$ *tel que* $\forall n \geq N$, $|a_{n+1}/a_n| \leq M$ *ou si* $\exists m$, $\exists M$, $\exists N$ *tel que* $\forall n \geq N$, $m \leq |a_n|$ *et* $|\Delta a_n| \leq M$ *alors* (T_n) *converge vers* S *plus vite que* (S_n).

Puisque dans le calcul de T_n il intervient S_{n+1}, il vaut mieux aussi regarder la convergence de (T_n) par rapport à (S_{n+1}) et il faudra poser des conditions plus restrictives.

Théorème 1.4 *Si* $\lim_{n\to\infty} g_{n+1}/g_n = 0$ *alors une condition nécessaire et suffisante pour que* (T_n) *converge vers* S *plus vite que* (S_{n+1}) *est que* $\lim_{n\to\infty} a_{n+1}/a_n = 1$.

Les conditions des théorèmes 1.3 et 1.4 sont satisfaites si $\exists a \neq 0$ tel que $\lim_{n\to\infty} a_n = a$ et, comme dans le cas du théorème 1.2, la condition peut être remplacée par une condition sur (Δa_n).

Cas logarithmique

Ce cas présente plus de difficultés. On a

Théorème 1.5 *Si* (g_n) *est logarithmique alors une condition nécessaire et suffisante pour que* (T_n) *converge vers* S *plus vite que* (S_n) *est que* $\lim_{n\to\infty} \dfrac{1 - a_{n+1}/a_n}{1 - g_n/g_{n+1}} = 0$. *Une condition nécessaire et suffisante pour que* (T_n) *converge vers* S *plus vite que* (S_{n+1}) *est que* $\lim_{n\to\infty} \dfrac{1 - a_n/a_{n+1}}{1 - g_{n+1}/g_n} = 0$.

Ce théorème peut donc s'appliquer seulement avec des conditions verifiées soit sur (a_n) soit sur (g_n). Cependant si (T_n) converge plus vite que (S_n) et si $\exists m$, $\exists N$ tel que $\forall n \geq N$, $m \leq |a_{n+1}/a_n|$ alors (T_n) converge aussi plus vite que (S_{n+1}). Viceversa si (T_n) converge plus vite que (S_{n+1}) et si $\exists M$, $\exists N$ tel que $\forall n \geq N$, $|a_{n+1}/a_n| \leq M$ alors (T_n) converge aussi plus vite que (S_n).

Si les conditions des théorèmes précédents ne sont pas satisfaites, alors $(T_n - S)/(S_n - S)$ (ou $(T_n - S)/(S_{n+1} - S)$) ne tendent pas vers zéro et on a deux cas:

1. $\exists c \neq 1$ tel que $\lim_{n\to\infty} (T_n - S)/(S_n - S) = c$. Dans ce cas il est possible, à partir des suites (S_n) et (T_n), de construire une nouvelle suite (t_n) qui converge vers S plus vite que (S_n) en utilisant le ACCES-algorithme de Litovsky [31] (voir [14] pour la subroutine correspondante).

2. $c = 1$ ou le rapport $(T_n - S)/(S_n - S)$ n'a pas de limite. Dans ce cas on peut utiliser une *contractive sequence transformation* [10, 12].

(a_n) et (g_n) inconnues

Si l'on remplace a_n par $a_n g_n$ et g_n par 1 pour tout n, ce cas peut être traité comme le cas "(a_n) inconnue et (g_n) connue" mais les conditions des théorèmes deviennent trop difficiles à contrôler. Donc il vaut mieux considerer ce cas à part.

Premier procédé d'extrapolation

Soit encore P un opérateur linéaire tel que $\forall n, P(a_n) = 0$. La relation (1.2) est encore vraie mais on ne peut pas l'utiliser pour le calcul de S car (g_n) est inconnue. Donc on remplace (g_n) par une approximation (h_n) et l'on pose pour tout n

$$T_n = P(S_n/h_n)/P(1/h_n).$$

Dans le cas plus général dans lequel on ne specifie pas les proprietés de P, on ne connaît rien de (T_n) et donc on procède comme nous avons fait dans le deuxième procédé d'extrapolation du cas "(a_n) inconnue et (g_n) connue".

Soit (b_n) une approximation de (a_n). Posons

$$T_n = S_n - b_n h_n, \qquad n = 0, 1, \ldots$$

Le théorème 1.1 est encore valable si l'on remplace dans ses conditions b_n par $b_n h_n$ et a_n par $a_n g_n$. Cette condition est satisfaite, par exemple, si $\exists a \neq 0$ tel que $\lim_{n\to\infty} b_n/a_n = \lim_{n\to\infty} g_n/h_n = a$. Mais dans la pratique il est souvent difficile de contrôler les conditions car elles concernent (b_n) et (h_n). Seulement dans certains cas, que nous allons montrer, on peut avoir une seule condition mais plus compliquée.

Deuxième procédé d'extrapolation

Soit (a_n) inconnue et $(g_n = S_{n-1} - S)$. On a pour tout n

$$S = \frac{S_n - a_n S_{n-1}}{1 - a_n}.$$

On remplace (a_n) par une approximation (b_n) et l'on pose

$$T_n = \frac{S_n - b_n S_{n-1}}{1 - b_n}, \qquad n = 1, 2, \ldots$$

Evidement on aura

Théorème 1.6 *Une condition nécessaire et suffisante pour que (T_n) converge vers S plus vite que (S_{n-1}) est que $\lim_{n\to\infty}(a_n - 1)/(b_n - 1) = 1$.*

Le procédé d'Overholt [38] est basé sur cette idée. La condition du théorème 1.6 est satisfaite si $\exists a \neq 1$ tel que (a_n) et (b_n) convergent vers a. Et elle est aussi valable si $(a_n - b_n)$ converge vers zéro et si $\exists \alpha < 1 < \beta$, $\exists N$ tel que $\forall n \geq N$, $b_n \notin [\alpha, \beta]$, ce qui a déjà démontré par Lembarki [30]. Par alanogie avec les fractions continues b_n peut s'appeler *facteur de convergence.*

Troisième procédé d'extrapolation

Nous allons voir le cas plus général où (a_n) et (g_n) sont des suites arbitraires et inconnues.

On fait encore une extrapolation en zéro avec la fonction linéaire $f(x) = ax + b$ telle que $S_n = f(h_n)$ et $S_{n+1} = f(h_{n+1})$ où (h_n) est une approximation de (g_n). On aura

$$T_n = S_n - \frac{\Delta S_n}{\Delta h_n} h_n, \qquad n = 0, 1, \ldots$$

et donc

$$\frac{T_n - S}{S_n - S} = \left(1 - \frac{a_{n+1}}{a_n} \cdot \frac{g_{n+1}}{g_n} \cdot \frac{h_n}{h_{n+1}}\right) \cdot \left(1 - \frac{h_n}{h_{n+1}}\right)^{-1}$$

$$\text{et} \quad \frac{T_n - S}{S_{n+1} - S} = \left(\frac{a_n}{a_{n+1}} \cdot \frac{g_n}{g_{n+1}} \cdot \frac{h_{n+1}}{h_n} - 1\right) \cdot \left(\frac{h_{n+1}}{h_n} - 1\right)^{-1}.$$

Nous allons donc généraliser le cas "(a_n) inconnue et (g_n) connue" avec les résultats suivants

Théorème 1.7 *Si (h_n) est une suite non-logarithmique et si $\exists a \neq 0$ fini tel que $\lim_{n\to\infty} g_n/h_n = a$ alors une condition nécessaire et suffisante pour que (T_n) converge vers S plus vite que (S_n) et (S_{n+1}) est que $\lim_{n\to\infty} a_{n+1}/a_n = 1$.*

Théorème 1.8 *Si (h_n) est non-logarithmique alors une condition nécessaire et suffisante pour que (T_n) converge vers S plus vite que (S_n) et (S_{n+1}) est que $\lim_{n\to\infty} \frac{a_{n+1}}{a_n} \cdot \frac{g_{n+1}}{g_n} \cdot \frac{h_n}{h_{n+1}} = 1$.*

Dans le cas des suites logarithmiques on a

Théorème 1.9 *Si (h_n) est logarithmique alors une condition nécessaire et suffisante pour que (T_n) converge vers S plus vite que (S_n) est que*

$$\lim_{n\to\infty} \left(1 - \frac{a_{n+1}}{a_n} \cdot \frac{g_{n+1}}{g_n} \cdot \frac{h_n}{h_{n+1}}\right) \cdot \left(1 - \frac{h_n}{h_{n+1}}\right)^{-1} = 0.$$

Une condition nécessaire et suffisante pour que (T_n) converge vers S plus vite que (S_{n+1}) est que $\lim_{n\to\infty} \left(\frac{a_n}{a_{n+1}} \cdot \frac{g_n}{g_{n+1}} \cdot \frac{h_{n+1}}{h_n} - 1\right) \cdot \left(\frac{h_{n+1}}{h_n} - 1\right)^{-1} = 0.$

Dans le théorème 1.9, $a_n g_n$ et $a_{n+1} g_{n+1}$ peuvent être remplacées respectivement par $S_n - S$ et les conditions par $S_{n+1} - S$ et aussi par $\lim_{n\to\infty} \dfrac{(S_{n+1} - S)/(S_n - S) - 1}{h_{n+1}/h_n - 1} = 1$ et $\lim_{n\to\infty} \dfrac{(S_n - S)/(S_{n+1} - S) - 1}{h_n/h_{n+1} - 1} = 1$. Cependant ces conditions sont difficiles à vérifier.

1.1.1 Itération des procédés

Les procédés d'extrapolations que nous avons présentés peuvent être itérés.

On considère la transformation

$$T_n = S_n - b_n g_n, \qquad n = 0, 1, \ldots$$

pour laquelle on a

$$T_n - S = (a_n - b_n) g_n.$$

Si l'on connaît une approximation (c_n) de $(a_n - b_n)$, alors on peut considerer la suite (U_n) donnée par

$$U_n = T_n - c_n g_n, \qquad n = 0, 1, \ldots$$

et, pour le théorème 1.1, une condition nécessaire et suffisante pour que (U_n) converge vers S plus vite que (T_n) est que $\lim_{n\to\infty} c_n/(a_n - b_n) = 1$.

La connaissance de la suite (c_n) est possible seulement dans très peu de cas.

Nous allons considerer l'itération de la transformation déjà proposée qui correspond au choix $b_n = \Delta S_n/\Delta g_n$. Dans ce cas on a

$$T_n - S = a'_n g'_n, \qquad n = 0, 1, \ldots$$

avec $a'_n = -\Delta a_n$ et $g'_n = g_n g_{n+1}/\Delta g_n$.

Donc, pour itérer le procédé, les suites (a'_n) et (g'_n) doivent satisfaire aux mêmes propriétés que les suites (a_n) et (g_n). Si (g_n) est non-logarithmique, alors des nouvelles conditions très restrictives doivent être introduites pour démontrer que (g'_n) est aussi non-logarithmique dans le cas général et le seul cas simple arrive quand $\lim_{n\to\infty} g_{n+1}/g_n = a \neq 1$.

Maintenant on considère le cas logarithmique pour voir quelles conditions sont nécessaires pour que (g'_n) soit aussi logarithmique. Evidemment on a

Théorème 1.10 *Si* $\lim_{n\to\infty} g_{n+1}/g_n = \lim_{n\to\infty} \Delta g_{n+1}/\Delta g_n = 1$ *alors* $\lim_{n\to\infty} g'_{n+1}/g'_n = 1$.

Comme commentaire on remarque que si (g_n) est monotone et logarithmique alors il n'est pas possible que $\lim_{n\to\infty} \Delta g_{n+1}/\Delta g_n = a$ avec $a = 0, +\infty$ or $a \neq 1$. Donc ou $(\Delta g_{n+1}/\Delta g_n)$ tend vers 1 ou elle n'a pas de limite.

Cet dernier résultat a été obtenu pour la première fois par Kowalewski [29] avec une démontration bien plus longue. Elle pose $(S_{n+1} - S)/(S_n - S) = 1 - \lambda_n$ avec $\lim_{n\to\infty} \lambda_n = 0$. Si (S_n) est monotone alors $\forall n$, $\lambda_n > 0$ et elle a démontré que ou $(\lambda_{n+1}/\lambda_n)$ tend vers 1 ou n'a pas de limite. Ce qui est équivalent a notre résultat.

Encore dans [13] nous avons montré que

Théorème 1.11 *Si* $\lim_{n\to\infty} g_{n+1}/g_n = a \neq 1$ *alors* $\lim_{n\to\infty} g'_{n+1}/g'_n = a$.

Ce résultat est vrai même si $a = 0$.

1.1.2 Applications

Dans [13] nous avons presenté plusieurs exemples pour illustrer tous les procédés et les théorèmes donnés, au moins un exemple pour chaque résultat. On a montré les résultats sous forme de figures dans lesquelles nous avons dessiné deux courbes, l'une pour la suite donnée (tirets) et l'autre pour la suite transformée (ligne continue). Dans ces figures on a utilisé les notions de cinématique introduite par Brezinski dans [8] que nous allons décrire:

Soit (S_n) une suite qui converge vers S. Posons $\forall n$

$$d_n = -log_{10}|S_n - S|.$$

La *"vitesse"* de la suite (S_n) est définie par $(v_n = \Delta d_n)$ et son *"accélération"* par $(\gamma_n = \Delta v_n)$. Soient (v'_n) et (γ'_n) la vitesse et l'accélération de (T_n). Brezinski dans [8] a démontré que si $\exists k > 0$ tel que $\forall n \geq N$, $v'_n \geq v_n + k$ alors (T_n) converge plus vite que (S_n). Puisque cette condition est seulement suffisante il a aussi démontré que ce résultat est aussi valable si $\forall n \geq N$, $v'_n > v_n$ et $\gamma'_n \geq \gamma_n$. Donc dans les figures on a représenté d_n en fonction de n.

Au lieu de reprendre ici tous les détails des exemples proposés qui peuvent être trouvés dans [13], nous allons donner seulement une liste avec les théorèmes auxquels ils font référence.

Exemple 1, Théorème 1.1

Suite

$$S_n = n\sin(1/n), \qquad n = 1, 2, \ldots$$

qui converge vers $S = 1$.

On a

$$S_n - S = \frac{1}{n^2}\left(-\frac{1}{6} + \frac{1}{5!n^2} - \cdots\right)$$

et on pose $g_n = n^{-2}$ et $b_n = -1/6 + e_n$ où (e_n) est une suite arbitraire qui converge vers zéro.

On prend $e_n = 1/n$, $e_n = 0.9^n$ et $e_n = 1/n^2$.

Exemple 2, Théorème 1.2

Suite

$$S_n = \sum_{i=1}^{n} \frac{i(i+1)}{2} x^{i-1}, \qquad n = 1, 2, \ldots, \quad |x| < 1.$$

Dans [26] Gradshteyn et Ryzhik ont démontré que

$$S_n = \frac{1}{(1-x)^3} - x^n\left[\frac{1}{(1-x)^3} + \frac{n(n+3) + xn(n+1)}{2(1-x)^2}\right]$$

et donc $S = (1-x)^{-3}$.

On prend $g_n = x^n$ (donc (g_n) est non-logarithmique),

$$a_n = \frac{1}{2(1-x)^3}\Big[2 + (1-x)n(n+3+x(n+1))\Big]$$

et $x = 0.5$.

Exemple 3, Théorèmes 1.3 et 1.4

Suite

$$S_n = \frac{1\cdot 2}{3!} + \cdots + \frac{n\cdot 2^n}{(n+2)!} = 1 - \frac{2^{n+1}}{(n+2)!}$$

qui converge vers $S = 1$.

On prend $g_n = 1/(n+2)!$ et $a_n = 2^{n+1}$.

Exemple 4, Théorèmes 1.3 et 1.4

On considère un exemple qu'on peut trouver dans un article de Gautschi [24]:

$$e^t = 1 + \frac{t}{1!} + \cdots + \frac{t^n}{n!} + \frac{t^{n+1}}{(n+1)!}\, e^{t\tau_n}$$

avec

$$0 < \tau_n < 1 \text{ if } t > 0 \quad \text{and} \quad \lim_{n\to\infty} \tau_n = 0.$$

On pose $S_n = 1 + \frac{t}{1!} + \cdots + \frac{t^n}{n!}$, $S = e^t$, $g_n = t^{n+1}/(n+1)!$ et $a_n = -e^{t\tau_n}$.

Exemple 5, Théorèmes 1.3 et 1.4

On prend l'intégrale

$$S = \int_{-1}^{1} \frac{e^{2x}}{\sqrt{1-x^2}}dx = \pi\, I_0(2)$$

et avec la méthode de Gauss-Chebyshev on considère ses valeurs approchées

$$S_n = \sum_{i=0}^{n} A_i f(x_i)$$

avec $f(x) = e^{2x}$, $A_i = \pi/(n+1)$ et $x_i = \cos\dfrac{2i+1}{2n+2}\pi$. On sait que

$$S - S_n = \frac{2\pi f^{(2n+2)}(\zeta_n)}{4^{n+1}(2n+2)!}, \qquad \zeta_n \in [-1,+1].$$

On calcule la valeur exacte de $I_0(2) = J_0(2i)$ avec l'algorithme de Moshier [35] qui donne au moins une précision de $8.6\cdot 10^{-18}$.

On prend $g_n = \dfrac{1}{4^{n+1}(2n+2)!}$.

Exemple 6, Théorème 1.5

Suite

$$S_n = \frac{3}{1\cdot 2\cdot 4} + \cdots + \frac{n+2}{n(n+1)(n+3)} = \frac{29}{36} - \frac{1}{n+3}\left[1 + \frac{3}{2(n+2)} + \frac{4}{3(n+1)(n+2)}\right]$$

qui converge vers 29/36 [26] et

$$g_n = (n+3)^{-1}.$$

Exemple 7, Théorème 1.5

Suite

$$S_n = \frac{n}{sin(1/n)}$$

et $g_n = n^{-2}$.

Exemple 8, Théorème 1.5

Suite (S_n) obtenue à partir de

$$S_n - S = (S_{n-1} - S)\,\frac{1}{2}\sin(an + b/n), \qquad n = 1, 2, \ldots$$

$g_n = S_{n-1} - S$, $a_n = \frac{1}{2}\sin(an + b/n)$, $b_n = \frac{1}{2}\sin an$, $S_0 = 0$, $S = 1$, $a = 2$ et $b = 1$.

Exemple 9, Théorème 1.7

Suite

$$S_n = S + a_n g_n, \qquad n = 0, 1, \ldots$$

$g_n = \lambda^{n+1}(0.5 + 1/\ln(n+2))$, $a_n = 2 + (n+1)^{-1}$, $|\lambda| < 1$, $h_n = \lambda^{n+1}$, $\lambda = 0.95$ et $S = 1$.

Exemple 10, Théorème 1.8

Suite

$$S_n = S + a_n g_n, \qquad n = 0, 1, \ldots$$

$g_n = \lambda^{n+1}/(n+1)$, $a_n = 2 + (n+1)^{-1}$, $|\lambda| < 1$, $h_n = \lambda^{n+1}$, $\lambda = 0.95$ et $S = 1$.

Exemple 11, Théorème 1.9

Suite

$$S_n = S + a(n+b)^{-1}, \qquad n = 0, 1, \ldots, \quad b \geq 1$$

$h_n = (n+1)^{-1}$, $S = 1$, $a = 2$ et $b = 4$.

1.2 Instabilité numérique

Un sujet particulièrement délicat dans les méthodes d'extrapolation est celui de la propagation des erreurs de cancellation dues à l'arithmétique des ordinateurs. Plus un algorithme d'extrapolation est bon, plus il souffre des erreurs d'arrondi. Par exemple, si l'on considère de nouveau l'ε-algorithme et le schéma

$$\begin{array}{ccccc} & & N = \varepsilon_{2k}^{(n)} & & \\ & a = \varepsilon_{2k-1}^{(n+1)} & & b = \varepsilon_{2k+1}^{(n)} & \\ W = \varepsilon_{2k-2}^{(n+2)} & & C = \varepsilon_{2k}^{(n+1)} & & E = \varepsilon_{2k+2}^{(n)} \\ & e = \varepsilon_{2k-1}^{(n+2)} & & d = \varepsilon_{2k+1}^{(n+1)} & \\ & & S = \varepsilon_{2k}^{(n+2)} & & \end{array}$$

les problèmes principaux viennent des situations suivantes

- Si N est différent de C mais très proche de lui (tous les deux sont des approximations de S), alors il y aura une erreur de cancellation dans le calcul de b (*near-breakdown* dans l'algorithme) qui sera grand et mal calculé. Si S est différent de C mais très proche de lui, alors d sera grand et mal calculé. Donc, dans le calcul de E il y aura, dans le dénominateur, la différence de deux nombres grands et mal calculés. Si $N = C$ alors b est égal à l'infini (*breakdown* dans l'algorithme). Si S est aussi égal à C alors d est égal à l'infini. Donc E n'est pas défini.

- Si a est différent de e mais très proche de lui, C sera grand et mal calculé. Donc b et d seront à peu près égaux entre eux et E sera la somme algébrique de deux nombres grands et mal calculés (*near-breakdown* dans l'algorithme). Si $a = e$ alors C est égal à l'infini (*breakdown* dans l'algorithme). Si $N \neq C$ et $S \neq C$ alors $b = d$ et E ne sera pas défini.

Il y a deux façon pour éviter l'instabilité numérique dans un algorithme d'extrapolation:

- ses formes progressives
- ses règles particulières.

1.2.1 Forme progressive

Pour obtenir la forme progressive de l'ε-algorithme, on calcule la première diagonale descendante $\left(\varepsilon_k^{(0)}\right)$, par exemple avec la méthode de bordage [23] ou avec la méthode de bordage par blocs et la méthode de bordage inverse [15, 16]. Après on modifie la règle de l' ε-algorithme pour lui donner la forme

$$\varepsilon_{k+1}^{(n+1)} = \varepsilon_{k+1}^{(n)} + \frac{1}{\varepsilon_{k+2}^{(n)} - \varepsilon_k^{(n+1)}}.$$

Cela peut s'appliquer à d'autres méthodes d'extrapolation et dans le cas général la forme progressive permet, en partant de la première diagonale descendante $\left(\left(\varepsilon_k^{(0)}\right)\right.$ dans

l'ε-algorithme) et de la deuxième colonne ($\left(\varepsilon_0^{(n)} = S_n\right)$ dans le ε-algorithme) de calculer toutes les autres quantités du tableau. Naturellement même cette règle souffre d'instabilité numérique puisque, si k est pair, on doit calculer la différence de deux quantités presque égales. Cependant l'instabilité n'est pas si grave car, normalement, $\varepsilon_{2k+2}^{(n)}$ est une approximation de S meilleure que $\varepsilon_{2k}^{(n+1)}$ et ces deux quantités ont moins de chiffres en commun que $\varepsilon_{2k}^{(n)}$ et $\varepsilon_{2k}^{(n+1)}$ (forme normale de l'ε-algorithme).

Il existe aussi la forme progressive suivante de la règle de la croix de Wynn

$$S = C + \left[(W-C)^{-1} + (E-C)^{-1} - (N-C)^{-1}\right]^{-1}.$$

1.2.2 Règles particulières

Les règles particulières sont obtenues en modifiant la règle de l'algorithme afin d'avoir un algorithme plus stable.

Par exemple, dans le procédé Δ^2 d'Aitken on a

$$T_n = \frac{S_n S_{n+2} - S_{n+1}^2}{S_{n+2} - 2S_{n+1} + S_n}, \qquad n = 0, 1, \ldots$$

et cette formule est très instable numériquement car si S_n,S_{n+1} et S_{n+2} sont presque égaux, il y a une erreur de cancellation au numérateur ainsi qu'au dénominateur et donc T_n sera mal calculé.

Si nous écrivons la formule sous la forme

$$T_n = S_n - \frac{(S_{n+1} - S_n)^2}{S_{n+2} - 2S_{n+1} + S_n}, \qquad n = 0, 1, \ldots$$

on obtient une règle plus stable. Evidement il y a encore une erreur de cancellation quand on calcule $(S_{n+1} - S_n)^2$ et $S_{n+2} - 2\,S_{n+1} + S_n$, mais le terme $(S_{n+1} - S_n)^2 / (S_{n+2} - 2\,S_{n+1} + S_n)$ devient une correction du terme S_n et cela explique la meilleure stabilité de la deuxième formule.

Si nous utilisons les mêmes notations qu'avant, la forme normale de l' ε-algorithme donne

$$\begin{aligned} C &= W + 1/(e-a) \\ b &= a + 1/(C-N) \\ d &= e + 1/(S-C) \\ E &= C + 1/(d-b). \end{aligned}$$

Si a est presque égal à e, alors C sera grand et mal calculé (proche de l'infini), b et d serons presque égaux et, comme on l'a déjà vu, E ne sera pas défini.

Après des modifications algébriques, la règle de la croix pour l'ε-algorithme peut s'écrire comme

$$E = r(1 + r/C)^{-1}$$

avec $$r = S(1 - S/C)^{-1} + N(1 - N/C)^{-1} - W(1 - W/C)^{-1}.$$

Wynn [48] a démontré que cette règle est bien plus stable que la règle (1.1).

En effet si C est infini (c'est à dire $a = e$), elle permet de calculer E par

$$E = S + N - W$$

et donc on peut sauter la singularité car, si N et S sont tous les deux différents de C, alors $a = e = b = d$ et E n'est pas défini.

Cette règle est valable seulement s'il y a une seule singularité isolée, c'est à dire quand seulement deux quantités adjacentes dans la même colonne impaire (a et e dans notre exemple) sont égales ou presque égales. Naturellement elle peut aussi s'appliquer quand la singularité se trouve dans une colonne paire.

La règle particulière de Wynn a été étendue par Cordellier aux cas d'un nombre quelconque de singularités adjacentes dans l'ε-algorithme [20]. La règle particulière de Wynn peut aussi s'utiliser dans la première et dans la deuxième généralisation de l'ε-algorithme et dans le ϱ-algorithme. Wynn a aussi proposé une règle particulière pour le qd-algorithme. Utilisant le complément de Schur, Brezinski a obtenues des règles particulières tant pour le E-algorithme que pour des transformations de suites vectorielles, comme le RPA, le CRPA et le H-algorithme. Avec ces règles, on peut calculer directement les éléments de la colonne $m + k$ du tableau en utilisant les éléments de la colonne k, sans calculer les colonnes intermédiaires et donc en évitant la division par zéro ou l'instabilité numérique due aux erreurs de cancellation.

Pour l'ε-algorithme vectoriel, la règle normale de la croix est identique à celle de l'ε-algorithme scalaire et une règle particulière a été obtenue par Cordellier [21] si N, S et W sont différents de C et si $(N - C)^{-1} + (S - C)^{-1}$ n'est pas égal à $(W - C)^{-1}$.

1.2.3 Règles particulières pour le Θ–algorithme

Le Θ–algorithme est un algorithme d'extrapolation trés puissant pour accélérer les suites qui convergent lentement. Les expériences numériques de Smith et Ford [42, 43] ont montré que le Θ-algorithme est parmi les meilleurs algorithmes d'extrapolation et il donne presque toujours une bonne approximation de la limite. Comme les autres algorithmes d'extrapolation il est très sensible à la propagation des erreurs d'arrondi dues à la cancellation entre deux quantités très proches.

Nous avons donc pensé chercher des règles particulières pour cet algorithme qui soient utilisées à la place des normales quand, dans la même colonne, on trouve deux quantités adjacentes presque égales. Nous allons maintenant présenter ces règles et comment nous les avons obtenues [39]. Les exemples numériques nous montrent que ces règles peuvent dans certains cas augmenter la stabilité numérique de l'algorithme, tandis que, dans d'autres cas, l'amélioration est presque nulle.

Le Θ-algorithme a été obtenu par Brezinski [2] et ses règles normales sont les suivantes

$$\Theta_{-1}^{(n)} = 0\,, \quad \Theta_0^{(n)} = S_n\,, \qquad n = 0, 1, \ldots$$

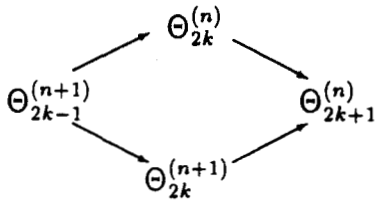
$$\Theta_{2k+1}^{(n)} = \Theta_{2k-1}^{(n+1)} + \frac{1}{\Theta_{2k}^{(n+1)} - \Theta_{2k}^{(n)}}\,, \qquad k, n = 0, 1, \ldots$$

$$\Theta_{2k+2}^{(n)} = \Theta_{2k}^{(n+1)} + \frac{\omega_k^{(n)}}{\Theta_{2k+1}^{(n+1)} - \Theta_{2k+1}^{(n)}}, \qquad k, n = 0, 1, \ldots$$

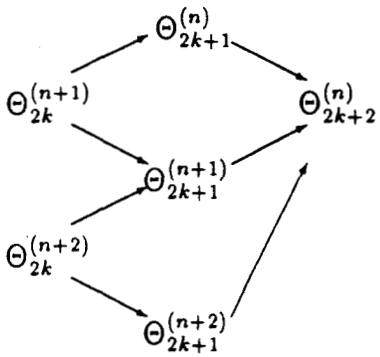
avec
$$D_k^{(n)} = \left(\Theta_k^{(n+1)} - \Theta_k^{(n)}\right)^{-1}$$
$$\omega_k^{(n)} = -\frac{\Delta\varepsilon_{2k}^{(n+1)}}{\Delta D_{2k+1}^{(n)}} = -\frac{\Theta_{2k}^{(n+2)} - \Theta_{2k}^{(n+1)}}{\left(\Theta_{2k+1}^{(n+2)} - \Theta_{2k+1}^{(n+1)}\right)^{-1} - \left(\Theta_{2k+1}^{(n+1)} - \Theta_{2k+1}^{(n)}\right)^{-1}}.$$

Comme dans l'ε-algorithme, les quantités qui appartiennent aux colonnes impaires sont des résultats intermédiaires.

La règle du Θ-algorithme pour les quantités des colonnes impaires relie, comme dans l'ε-algorithme, des quantités qui se trouvent aux quatre sommets d'un losange,



mais, dans le cas des colonnes paires, il y a plus de quantités à considérer dans la règle



Donc, nous allons considérer séparement deux cas, selon la présence dans la même colonne (impaire ou paire) de deux quantités adjacentes égales (ou presque égales).

1.2.4 Règle pour les colonnes paires

Posons

$$\begin{array}{ccccc} & & N = \Theta_{2k+1}^{(n)} & & \\ & a = \Theta_{2k}^{(n+1)} & & b = \Theta_{2k+2}^{(n)} & \\ W = \Theta_{2k-1}^{(n+2)} & & C = \Theta_{2k+1}^{(n+1)} & & E = \Theta_{2k+3}^{(n)} \\ & e = \Theta_{2k}^{(n+2)} & & d = \Theta_{2k+2}^{(n+1)} & \\ T = \Theta_{2k-1}^{(n+3)} & & S = \Theta_{2k+1}^{(n+2)} & & \\ & i = \Theta_{2k}^{(n+3)} & & & \\ & & V = \Theta_{2k+1}^{(n+3)} & & \end{array}$$

Si nous utilisons la règle normale de l'algorithme on a

$$
\begin{aligned}
C &= W + \frac{1}{e-a} \\
b &= a + \frac{\omega_k^{(n)}}{C-N} \qquad \text{avec} \qquad \omega_k^{(n)} = -\frac{e-a}{(S-C)^{-1}-(C-N)^{-1}} \\
d &= e + \frac{\omega_k^{(n+1)}}{S-C} \qquad \text{avec} \qquad \omega_k^{(n+1)} = -\frac{i-e}{(V-S)^{-1}-(S-C)^{-1}} \\
E &= C + \frac{1}{d-b}.
\end{aligned}
$$

Si a et e appartiennent à une colonne paire et E appartient à une colonne impaire, et si a est différent de e mais proche, alors C sera grand et mal calculé. Si N et S ne sont pas proches de C, c'est à dire qu'ils ne sont pas grands (singularité isolée), alors $\omega_k^{(n)}$ sera le rapport de deux quantités petites et mal calculées et dons b sera mal calculé. Si V n'est pas grand et i n'est pas proche de e, alors d sera presque égal à e. Donc E sera calculé à partir de quantités mal calculées. Si, en plus, i est proche de a et e alors la situation devient encore pire car S sera aussi grand et mal calculé et presque toutes les quantités qui interviennent dans le calcul de E seront mal calculées.

Nous allons faire des manipulations algébriques, similaires à celles faites par Wynn, pour obtenir une formule pour E qui soit, si possible, plus stable.

$$
\begin{aligned}
E &= C + \frac{1}{d-b} \\
&= C + \frac{1}{e+\omega_k^{(n+1)}(S-C)^{-1} - a - \omega_k^{(n)}(C-N)^{-1}} \\
&= C + \frac{1}{(C-W)^{-1} + \omega_k^{(n+1)}(S-C)^{-1} - \omega_k^{(n)}(C-N)^{-1}} \\
&= C + \frac{C}{C(C-W)^{-1} - C\omega_k^{(n+1)}(C-S)^{-1} - C\omega_k^{(n)}(C-N)^{-1}} \\
&= C + \frac{C}{(C-W+W)(C-W)^{-1} - \omega_k^{(n+1)}(C-S+S)(C-S)^{-1} - \omega_k^{(n)}(C-N+N)(C-N)^{-1}} \\
&= C + \frac{C}{1 + W(C-W)^{-1} - \omega_k^{(n+1)}\left[1+S(C-S)^{-1}\right] - \omega_k^{(n)}\left[1+N(C-N)^{-1}\right]} \\
&= C - \frac{C}{\omega_k^{(n+1)} + \omega_k^{(n)} - 1 - W(C-W)^{-1} + \omega_k^{(n+1)}S(C-S)^{-1} + \omega_k^{(n)}N(C-N)^{-1}} \\
&= \frac{C\left(\omega_k^{(n+1)} + \omega_k^{(n)} - 1\right) - C - CW(C-W)^{-1} + \omega_k^{(n+1)}CS(C-S)^{-1} + \omega_k^{(n)}CN(C-N)^{-1}}{\omega_k^{(n+1)} + \omega_k^{(n)} - 1 - W(C-W)^{-1} + \omega_k^{(n+1)}S(C-S)^{-1} + \omega_k^{(n)}N(C-N)^{-1}} \\
&= \frac{C\left(\omega_k^{(n+1)} + \omega_k^{(n)} - 2\right) + \omega_k^{(n+1)}S(1-S/C)^{-1} + \omega_k^{(n)}N(1-N/C)^{-1} - W(1-W/C)^{-1}}{\omega_k^{(n+1)} + \omega_k^{(n)} - 1 + C^{-1}\left[\omega_k^{(n+1)}S(1-S/C)^{-1} + \omega_k^{(n)}N(1-N/C)^{-1} - W(1-W/C)^{-1}\right]}
\end{aligned}
$$

$$= \frac{C\left(\omega_k^{(n+1)}+\omega_k^{(n)}-2\right)+\omega_k^{(n+1)}S(1-S/C)^{-1}+\omega_k^{(n)}N(1-N/C)^{-1}-W(1-W/C)^{-1}}{1+C^{-1}\left[C\left(\omega_k^{(n+1)}+\omega_k^{(n)}-2\right)+\omega_k^{(n+1)}S(1-S/C)^{-1}+\omega_k^{(n)}N(1-N/C)^{-1}-W(1-W/C)^{-1}\right]}.$$

Si nous posons

$$r = C\left(\omega_k^{(n+1)}+\omega_k^{(n)}-2\right)+\omega_k^{(n+1)}S(1-S/C)^{-1}+\omega_k^{(n)}N(1-N/C)^{-1}-W(1-W/C)^{-1}$$

on obtient une expression de la même forme que celle de la règle particulière de Wynn

$$E = r(1+r/C)^{-1}.$$

Maintenant nous allons aussi modifier le calcul des quantités $\omega_k^{(n)}$ et $\omega_k^{(n+1)}$

$$\begin{aligned}\omega_k^{(n)} &= -\frac{e-a}{(S-C)^{-1}-(C-N)^{-1}} \\ &= \frac{(C-W)^{-1}}{(C-S)^{-1}+(C-N)^{-1}} \\ &= \frac{(1-W/C)^{-1}}{(1-S/C)^{-1}+(1-N/C)^{-1}}\end{aligned}$$

$$\begin{aligned}\omega_k^{(n+1)} &= -\frac{i-e}{(V-S)^{-1}-(S-C)^{-1}} \\ &= \frac{(S-T)^{-1}}{(S-V)^{-1}+(S-C)^{-1}} \\ &= \frac{(1-T/S)^{-1}}{(1-V/S)^{-1}+(1-C/S)^{-1}}.\end{aligned}$$

Si nous utilisons ces formules pour les ω on aura

$$\begin{aligned}r &= C\left(\omega_k^{(n+1)}+\omega_k^{(n)}-2\right)-\frac{C(1-V/S)}{(1-T/S)(2-C/S-V/S)}+ \\ &\quad \frac{N(1-S/C)}{(1-W/C)(2-N/C-S/C)}-\frac{W}{(1-W/C)^{-1}} \\ &= C\left[\omega_k^{(n+1)}+\omega_k^{(n)}-2-\frac{(1-V/S)}{(1-T/S)(2-C/S-V/S)}\right]+ \\ &\quad \frac{N(1-S/C)}{(1-W/C)(2-N/C-S/C)}-\frac{W}{(1-W/C)^{-1}}\end{aligned}$$

et

$$\begin{aligned}r/C &= \omega_k^{(n+1)}+\omega_k^{(n)}-2-\frac{(1-V/S)}{(1-T/S)(2-C/S-V/S)}+\frac{N/C(1-S/C)}{(1-W/C)(2-N/C-S/C)} \\ &\quad -\frac{W/C}{(1-W/C)^{-1}}\end{aligned}$$

Dans le cas d'une singularité isolée (par exemple $a = e$) alors $C = \infty$. Si $T \neq S, V \neq S$ et $S \neq 0$, d'après la règle particulière on aura

$$\omega_k^{(n)} = 1/2 \qquad \text{et} \qquad \omega_k^{(n+1)} = \frac{(1 - T/S)^{-1}}{(1 - V/S)^{-1}} \neq \infty$$

$$r = \alpha \cdot C + \beta \qquad \text{avec} \qquad \alpha = \omega_k^{(n)} + \omega_k^{(n+1)} - 2 \neq \infty \quad \text{et} \quad \beta = N/2 - W \neq \infty$$

$$r/C = \alpha \qquad \text{et donc} \qquad E = \infty.$$

Dans la pratique il est vraiment difficile de trouver une singularité isolée car souvent quand on trouve, dans une colonne paire, deux quantités presque égales, alors dans la même colonne toutes les quantités suivantes sont aussi presque égales aux deux premières. Dans ce cas, la règle précédente ne permet pas d'éviter les calculs instables. Par exemple, quand $a = e = i$ alors $C = S = \infty$ et quand on calcule les ω et r on a le rapport S/C qui n'est pas défini.

Cependant l'utilisation de la règle permet (à part pour les quantités $\Theta_1^{(n)}$) de sauter au-dessus du breakdown. En effet quand le programme qui implémente le Θ-algorithme trouve deux quantités adjacentes dans une colonne paire exactement égales (à cause de la précision finie de l'ordinateur), il doit s'arrêter à cause d'une division par zéro dans la colonne impaire suivante. Avec cette règle particulière, une quantité dans une colonne impaire peut toujours être calculée car les formules lient seulement des éléments qui appartiennent aux colonnes impaires et ces éléments sont d'habitude très grands (quand la colonne paire précédente contient une bonne approximation de la limite) mais ils ne sont presque jamais égaux.

1.2.5 Règle pour les colonnes impaires

Posons

$$\begin{array}{ccccc}
 & & N = \Theta_{2k}^{(n)} & & \\
 & a = \Theta_{2k-1}^{(n+1)} & & b = \Theta_{2k+1}^{(n)} & \\
W = \Theta_{2k-2}^{(n+2)} & & C = \Theta_{2k}^{(n+1)} & & E = \Theta_{2k+2}^{(n)} \\
 & e = \Theta_{2k-1}^{(n+2)} & & d = \Theta_{2k+1}^{(n+1)} & \\
T = \Theta_{2k-2}^{(n+3)} & & S = \Theta_{2k}^{(n+2)} & & \\
 & i = \Theta_{2k-1}^{(n+3)} & & h = \Theta_{2k+1}^{(n+2)} & \\
 & & V = \Theta_{2k}^{(n+3)} & &
\end{array}$$

Utilisant la forme normale de l'algorithme on a

$$C = W + \frac{\omega_{k-1}^{(n+1)}}{e-a} \quad \text{avec} \quad \omega_{k-1}^{(n+1)} = -\frac{T-W}{(i-e)^{-1}-(e-a)^{-1}}$$

$$\text{donc} \quad C = W - \frac{T-W}{(e-a)\,(i-e)^{-1}-1}$$

$$b = a + \frac{1}{C-N}, \qquad d = e + \frac{1}{S-C}, \qquad h = i + \frac{1}{S-V}$$

$$E = C + \frac{\omega_k^{(n)}}{d-b} \quad \text{avec} \quad \omega_k^{(n)} = -\frac{S-C}{(h-d)^{-1}-(d-b)^{-1}}$$

$$\text{donc} \quad E = C - \frac{S-C}{(d-b)\,(h-d)^{-1}-1}.$$

Si a est différent de e mais proche, alors le dénominateur de C sera presque égal à -1 et alors C sera presque égal à T. Si e est différent de i mais proche, alors le dénominateur de C sera grand et donc C sera presque égal à W. Dans ce cas il n'y a pas un problème de vraie instabilité dans le calcul de C et de toutes les autres quantités. La seule condition d'instabilité arrive quand a, e et i sont tous presque égaux, mais cette condition, pour les quantités adjacentes d'une colonne impaire, n'arrive presque jamais dans la pratique.

Cependant la forme normale pour les calcul des quantités d'une colonne paire, ne permet pas d'utiliser le Θ-algorithme si e et i (ou h et d) sont exactement égaux. Donc on cherche une formule différente qui permet d'éviter ce breakdown.

Nous faisons des modification algébriques et nous avons

$$\begin{aligned} C &= W - \frac{T-W}{(e-a)\,(i-e)^{-1}-1} = W - \frac{(T-W)(i-e)}{(e-a)-(i-e)} \\ &= \frac{W\,(2e-a-i)-(T-W)(i-e)}{(2e-a-i)} = \frac{W\,(e-a)-T(i-e)}{(2e-a-i)} \end{aligned}$$

Si les deux différences $e-a$ et $i-a$ ne sont pas presque égales, cette formule évite aussi partiellement l'erreur de cancellation de la forme normale, qui arrive quand des quantités W et T deviennent très proches.

1.2.6 Exemples numériques

On définit une valeur m qui représente le nombre de chiffres décimaux que l'utilisateur accepte de perdre à cause de l'erreur de cancellation.

La règle particulière pour les colonnes paires du Θ-algorithme sera utilisée pour calculer les $\Theta_{2k+3}^{(n)}$ quand, pour tout $k = 0, 1, \ldots$ et pour tout $n = 0, 1, \ldots$ la relation suivante est verifiée

$$\left|\frac{\Theta_{2k}^{(n+2)} - \Theta_{2k}^{(n+1)}}{\Theta_{2k}^{(n+1)}}\right| < 10^{-m} \qquad \text{ou} \qquad \left|\frac{\Theta_{2k+2}^{(n+1)} - \Theta_{2k+2}^{(n)}}{\Theta_{2k+2}^{(n)}}\right| < 10^{-m}.$$

Dans notre cas, nous avons utilisé la règle aussi quand il y a plusieures singularités, c'est à dire quand plusieurs quantités adjacentes de la même colonne ont été trouvées presque égales.

La règle particulière pour les colonnes impaires sera utilisée dans le calcul de $\Theta_{2k+2}^{(n)}$ quand

$$\left|\frac{\Theta_{2k+1}^{(n+2)} - \Theta_{2k+1}^{(n+1)}}{\Theta_{2k+1}^{(n+1)}}\right| < 10^{-m} \qquad \text{ou} \qquad \left|\frac{\Theta_{2k}^{(n+2)} - \Theta_{2k}^{(n+1)}}{\Theta_{2k}^{(n+1)}}\right| < 10^{-m}.$$

Dans tous les tableaux suivants on montre le nombre de chiffres décimaux exacts ($-\log_{10}|\text{erreur relative}|$). La valeur 999.0 indique que toutes les chiffres sont exacts. Quand une valeur manque, cela signifie que dans l'algorithme il y a eu un breakdown et donc la valeur du tableau n'est pas possible à calculer. La suite des valeurs données pour $n = 0, 1, 2, \ldots$ est $\Theta_0^{(0)}$, $\Theta_0^{(1)}$, $\Theta_0^{(2)}$, $\Theta_2^{(0)}$, $\Theta_2^{(1)}$, $\Theta_2^{(2)}$, $\Theta_4^{(0)}$, $\Theta_4^{(1)}$, ...

Les calculs ont été faits sur un PC avec le Microsoft FORTRAN Optimizing Compiler avec lequel on a une précision d'environ 15–16 chiffres décimaux.

Exemple 1

On considère la série

$$e^x = \sum_{i=0}^{\infty} x^i/i!$$

et la suite suivante dérivée de cette série

$$\begin{aligned} S_0 &= 0 \\ S_n &= \sum_{i=0}^{n-1} x^i/i!, \qquad n = 1, 2, \ldots \end{aligned}$$

Cet exemple a été déjà donné par Wynn [48] et il est intéressant car il y a, pour $x = 2$, une singularité isolée dans une colonne impaire ($\Theta_1^{(1)} = \Theta_1^{(2)} = 0.5$) et une autre quand $x = 4$ ($\Theta_1^{(3)} = \Theta_1^{(4)} = 9.3749999999999 \cdot 10^{-2}$).

Avec $m = 12$ on a obtenu

	$x = -4.5$		$x = -2.0$		$x = 2.0$		$x = 2.5$		$x = 4.0$	
n	Θ	Θ r.p.	Θ	Θ r.p.	Θ	Θ r.p.	Θ	Θ r.p.	Θ	Θ r.p.
3	−0.68	−0.68	0.32	0.32		0.06	0.02	0.02	0.00	0.00
4	−0.85	−0.85	0.75	0.75	0.49	0.49	0.22	0.22	0.02	0.02
5	−0.84	−0.84	1.26	1.26	1.28	1.28	0.80	0.80		0.12
6	0.79	0.79	3.02	3.02		1.59	0.60	0.60		0.00
7	1.17	1.17	3.92	3.92	3.67	3.67	2.53	2.53		0.17
8	1.58	1.58	4.78	4.78	4.47	4.47	3.73	3.73		1.16
9	4.11	4.11	6.98	6.98		4.82	4.44	4.44		0.61
10	4.76	4.76	8.20	8.20	6.55	6.55	5.17	5.17		3.63
11	5.36	5.36	9.49	9.49	9.16	9.16	7.10	7.10		4.02
12	7.27	7.27	11.56	11.56		10.25	8.20	8.20		3.64
13	8.14	8.14	12.33	12.33	11.10	11.10	10.21	10.21		4.46
14	9.14	9.14	12.76	12.76	11.67	11.67	10.82	10.82		8.29
15	11.68	11.68	13.21	13.21		12.10	9.46	9.46		9.70
16	11.61	11.61	12.93	12.92	12.12	12.12	11.53	11.53		11.68
17	11.77	11.77	15.21	15.09	12.53	12.53	11.98	11.98		10.06
18	12.29	12.29	15.39	15.39		11.80	12.02	12.02		10.07
19	11.59	11.59		999.0		13.22	11.99	11.99		10.07
20	13.32	13.32		999.0		14.31	12.07	12.07		10.07
21	12.66	12.66		999.0		14.97	12.01	12.01		10.07
22		15.81		999.0		999.0		11.80		10.06
23		15.81		999.0		999.0		13.45		10.22
24		15.81		999.0		999.0		12.16		10.07

Pour toutes les valeurs de x, quand deux valeurs successives de la suite donnée ne sont pas exactement égales, la règle particulière permet de calculer entièrement le tableau du Θ. Les règles normales, au contraire, trouvent des breakdowns dans les calculs de certaines valeurs. Pour ce qui concerne la stabilité, quand les valeurs sont toutes les deux calculables (sans et avec les règles particulières), les règles particulières donnent presque toujours le même nombre de chiffres exacts. Il y a beaucoup de cas dans lesquels les règles particulières donnent un résultat meilleur (par exemple $\Theta_4^{(15)}$, $\Theta_4^{(18)}$, $\Theta_6^{(11)}$, $\Theta_6^{(12)}$, $\Theta_6^{(13)}$, $\Theta_8^{(8)}$, $\Theta_8^{(9)}$ et $\Theta_{10}^{(6)}$, pour $x = 2.5$) et il y a aussi des cas dans lesquels le résultat est pire (par exemple $\Theta_6^{(11)}$, $\Theta_{10}^{(1)}$ et $\Theta_{10}^{(2)}$ pour $x = -2.0$).

Exemple 2

On considère la suite suivante qui converge vers $S = 1$

$$\begin{aligned} S_0 &= a \\ S_{n+1} &= \sqrt{S_n}, \qquad n = 0, 1, \ldots \end{aligned}$$

On obtient, pour différentes valeurs de a et m les résultats suivants

	$a=5$			$a=50$			$a=500$			$a=5000$		
n	Θ	Θ r.p. $m=5$	Θ r.p. $m=12$	Θ	Θ r.p. $m=5$	Θ r.p. $m=12$	Θ	Θ r.p. $m=5$	Θ r.p. $m=12$	Θ	Θ r.p. $m=5$	Θ r.p. $m=12$
12	11.05	11.05	11.05	7.83	7.83	7.83	6.86	6.86	6.86	6.57	6.57	6.57
13	12.45	12.45	12.45	10.33	10.33	10.33	8.68	8.68	8.68	7.90	7.90	7.90
14	13.73	13.73	13.73	12.22	12.22	12.22	11.20	11.20	11.20	10.05	10.05	10.05
15	14.84	14.91	14.91	13.30	13.31	13.30	12.32	12.32	12.32	11.54	11.54	11.54
16	10.63	12.23	12.70	14.88	14.75	14.88	13.37	13.37	13.37	13.14	13.13	13.14
17	14.61	14.61	14.61	13.36	13.28	13.33	14.07	14.06	14.09	13.47	13.45	13.47
18	11.64	13.26	13.68	13.43	13.27	13.38	14.51	14.48	14.51	13.92	13.90	13.91
19	14.81	14.81	14.81		14.40	14.17	15.11	15.11	14.91	14.44	14.45	14.44
20	14.51	14.45	14.57		14.44	14.45	14.45	14.81	15.26	14.51	14.51	14.51
21	14.48	14.48	14.51		14.41	14.45	14.65	15.05	15.65	14.48	14.48	14.48
22	14.33	14.22	14.42		14.46	14.46	15.11	15.65	15.35	14.56	14.57	14.56
23		15.35	15.18		14.45	14.46	14.95	15.18	15.35	15.11	15.26	15.18
24		15.35	15.18		14.44	14.46	15.00	15.26	999.0		14.91	14.78
25		15.65	15.65		14.45	14.46	14.33	15.35	13.66		15.35	15.35
26		15.65	999.0		14.45	14.45	13.32	13.78	12.63		15.35	15.05
27		15.65	999.0		14.45	14.46	14.07	14.31	13.12		999.0	15.95

Là encore les règles particulières permettent de sauter au-dessus de tous les breakdowns trouvés en utilisant les règles normales et en plus elles permettent de trouver une meilleure approximation de la limite.

Quand $a = 5$, les règles particulières ne peuvent pas continuer après $n = 27$ car il y a un breakdown dans la règle particulière pour les colonnes paires parce que r/C est exactement égal à -1. Cependant, avec la même valeur de a, pour $n = 16$ et $n = 18$ on a une bonne amélioration du nombre de chiffres exacts.

Exemple 3

On considère la série

$$\frac{\Pi^2}{6} = \sum_{i=0}^{\infty}(i+1)^{-2}$$

et la suite suivante derivée de cette série

$$S_n = \sum_{i=0}^{n}(i+1)^{-2}, \qquad n = 0, 1, \ldots$$

Pour différents choix de m on a obtenu

		Θ règ. part.					Θ règ. part.			
n	Θ	$m=5$	$m=6$	$m=7$	$m=9$	n	Θ	$m=5$	$m=6$	$m=7$	$m=9$
27	8.57	8.49	8.57	8.57	8.57	63	−3.81	4.38	5.54	−3.82	−3.83
28	8.05	8.29	8.05	8.05	8.05	64	−5.19	7.88	8.16	−5.18	−5.19
29	7.23	8.59	7.23	7.23	7.23	65	−6.12	6.84	7.45	−6.09	−6.11
30	7.50	8.37	7.50	7.50	7.50	66	−6.33	7.24	7.53	−6.31	−6.33
31	6.77	8.73	6.77	6.77	6.77	67	−7.27	6.23	7.00	−7.22	−7.25
32	7.65	8.73	7.65	7.65	7.65	68	−8.55	5.20	5.61	−8.49	−8.52
33	6.34	8.73	6.34	6.34	6.34	69	−8.41	5.29	5.95	−8.35	−8.38
34	7.70	8.73	7.55	7.70	7.70	70	−9.70	4.68	5.07	−9.62	−9.65
35	7.97	8.73	7.27	7.98	7.98	71	4.22	3.96	3.97	5.90	6.02
36	7.78	8.73	7.42	7.79	7.79	72	−10.85	4.28	4.66	−10.74	−10.78
37	8.14	8.73	7.12	8.15	8.15	73	3.89	3.65	3.66	5.69	5.89
38	9.98	8.73	8.07	9.94	9.94	74	3.48	3.45	3.45	5.63	5.73
39	8.29	8.73	6.95	8.31	8.31	75	3.57	3.37	3.37	5.65	5.77
40	8.73	8.72	6.68	8.74	8.74	76	3.16	3.17	3.17	4.55	5.61
41	7.85	7.21	4.87	7.86	7.86	77	2.62	2.68	2.68	2.78	5.39
42	8.06	8.71	5.36	8.07	8.07	78	2.83	2.91	2.90	3.12	5.49
43	7.39	6.91	3.56	7.40	7.40	79	2.29	2.42	2.42	1.43	5.28
44	6.77	6.92	2.33	6.77	6.77	80	1.58	1.87	1.87	−0.45	5.01
45	7.10	6.93	2.23	7.10	7.10	81	1.97	2.16	2.16	0.08	5.18
46	6.63	6.75	0.90	6.63	6.63	82	1.26	1.61	1.61	−1.80	4.97
47	6.48	6.91	−0.85	6.48	6.48	83	4.76	4.53	4.53	−3.32	4.71
48	6.67	6.82	−0.50	6.61	6.61	84	0.93	1.35	1.35	−3.15	5.09
49	6.48	6.51	−2.24	6.48	6.48	85	4.30	4.16	4.16	−4.67	4.24
50	6.42	6.80	6.92	6.42	6.42	86	3.75	3.19	3.19	−5.79	3.73
51	6.42	6.96	−3.60	6.41	6.41	87	3.83	3.75	3.75	−6.02	3.77
52	6.13	6.91	7.04	6.07	6.07	88	3.29	2.75	2.75	−7.15	3.26
53	4.47	7.11	7.26	4.45	4.45	89	2.59	0.93	0.93	−8.77	2.58
54	4.72	7.03	7.17	4.68	4.68	90	2.82	2.32	2.32	−8.50	2.80
55	3.28	7.25	7.40	3.25	3.25	91	2.12	0.50	0.50	−10.12	2.12
56	1.56	8.85	8.02	1.52	1.52	92	0.55	4.15	4.15	−12.57	0.55
57	2.10	7.40	7.51	2.07	2.07	93	1.66	0.06	0.06	−11.47	1.65
58	0.41	7.89	7.33	0.37	0.37	94	0.09	3.90	3.90	−13.93	0.08
59	−1.52	6.67	6.69	−1.55	−1.55	95	−0.28	3.65	3.65	−15.21	−0.29
60	−0.74	6.83	6.74	−0.78	−0.78	96	−0.38	3.65	3.65	−15.28	−0.38
61	−2.66	5.54	6.11	−2.69	−2.69	97	−0.75	3.40	3.40	−16.56	−0.75
62	−4.04	7.99	8.35	−4.05	−4.06	98	−1.78	3.10	3.10	−18.59	−1.79

Comme l'on voit, cet exemple est très sensible au choix de m. Si $m = 5$ ou plus petit, alors les deux règles particulières sont appliquées presque depuis le début pour le calcul des valeurs après le 10-ème terme de la troisième colonne et elle continuent à s'appliquer dans les calcul successifs pour toutes les autres valeurs du tableau. Cela permet à l'algorithme d'obtenir dans les trois premières valeurs des colonnes 18, 19, 20 et 21 (qui correspondent à $n = 27$ jusqu'à $n = 38$) un nombre de chiffres plus grand que celui des règles normales. En plus les résultats obtenus avec les règles particulières (même si parfois ils sont un peu moins bons) ne souffrent pas de l'instabilité qu'il y a dans certains endroits du tableau et elles donnent une approximation acceptable aussi pour des valeurs très grandes de n (par exemple quand $n = 96$ qui correspond à $\Theta_{64}^{(0)}$).

Si $m = 6$ la règle particulière est appliquée plus tard, après le 20-ème élément de la troisième colonne. Donc dans les colonnes 18, 19, 20 et 21 on ne trouve aucune amélioration et, au contraire, les règles particulières donnent une zone d'instabilité dans les environs de $n = 48$ qui ne se trouve pas dans le cas des règles normales. Mais dans ce cas là, les règles particulières permettent de retrouver de bons résultats comme dans le cas de $m = 5$.

Si $m = 7$ ou $m = 9$, à cause de l'application tardive des règles particulières (pour $m = 9$, par exemple, elles sont utilisées seulement à partir de la colonne 5) l'algorithme avec les règles particulières devient presque chaotique avec certaines valeurs meilleures et d'autres valeurs pires.

Pour n'importe quel choix de m, le nombre de chiffres exacts des résultats des colonnes 2 à 16, avec et sans les règles particulières, est presque pareil.

Pour terminer quelque commentaire général.

La théorie du Θ-algorithme ne permet pas, à l'heure actuelle, de construire des exemples qui donnent dans une colonne du tableau des valeurs algébriquement connues ou une singularité isolée. Cela est possible pour d'autres algorithmes d'extrapolation, par exemple pour l'ε-algorithme. La seule possibilité est de comparer les résultats avec ceux obtenus en utilisant une plus grande précision ou le calcul formel. En faisant cela pour certaines suites, nous avons remarqué que les valeurs de la colonne 1 (pour lesquelles les règles particulières ne peuvent pas être utilisées) sont déjà affectées par une erreur et donc, dans certains cas, les règles particulières ne sont pas capables de corriger l'instabilité des résultats.

L'interêt principal de ces règles particulières est d'éviter le breakdown des règles normales (excepté quand le breakdown arrive dans la colonne 1, quand deux valeurs de la suite donnée sont exactement égales, ou le cas rare $r/C = -1$).

En ce qui concerne le gain de chiffres décimaux, il n'est, en général, pas vraiment remarquable et cela est probablement dû au fait que les singularités ne sont pas isolées et qu'il y a des zones (parfois grandes) dans le tableau de singularités adjacentes et dans ce cas là les règles particulières ne sont pas appliquables (en théorie).

1.3 Prédiction

En général, quand on considère une suite, on se trouve dans l'un des trois cas suivants:

1. tous les termes de la suite sont analytiquement connus;
2. seulement une partie des termes de la suite est connue;
3. on peut calculer numériquement tous les termes de la suite par une méthode quelconque.

Quand on veut utiliser une méthode d'extrapolation la situation, par rapport à ces trois cas, est la suivante. Le premier cas est évidemment le meilleur car on connaît toute la suite et donc on peut choisir la méthode d'extrapolation la plus adaptée (selon

les caractéristiques connues de la suite) et donc il n'y a pas de problèmes (à part les problèmes dûs à l'utilisation de l'arithmétique de l'ordinateur). Dans le deuxième cas on est forcé d'utiliser la méthode d'extrapolation seulement sur les termes connus de la suite. Dans le troisième cas si la méthode numérique est assez précise, on se retrouve dans le premier cas. Si la méthode, au contraire, est très lourde comme temps de calcul ou présente une perte de précision dans le calculs des terms successives, l'utilisateur peut vouloir s'arrêter après un certain nombres de termes calculés et donc on rentre dans le deuxième cas.

La prédiction, est basée sur l'extrapolation et elle peut aider à résoudre les problèmes de détermination des termes inconnus de la suite dans le deuxième et dans le troisième cas. Donc au lieu d'obtenir une nouvelle suite qui converge vers la limite, plus vite que la suite donnée, on cherche à donner une valeur approchée des termes inconnus, à partir des termes connus. C'est à dire à partir, par exemple, de $S_0, S_1, \ldots, S_n$ on veut "prédire" $S_{n+1}, S_{n+2}, \ldots$.

Gilewicz dans [25] a été le premier à avoir cette idée et il l'a appliquée au calcul des termes inconnus d'une série en utilisant le développement en série formelle des approximants de Padé. Plus tard Sidi et Levin [41] l'ont encore appliquée avec la transformation t qui est une approximation rationnelle de type-Padé pour les séries de puissances.

Brezinski [9] a été le premier à montrer comme utiliser le procédé Δ^2 d'Aitken et le E-algorithme pour prédire les termes inconnus d'une suite.

Nous allons, dans la suite, reprendre les résultats obtenus par Brezinski [9].

On considère une suite (S_n) et une transformation de suite $T : (S_n) \longrightarrow (T_n)$. On appelle *noyau* de la transformation T l'ensemble

$$\mathcal{K}_T = \{(u_n) | \exists N, \exists S, \forall n \geq N, u_n = S\}$$

où S est la limite de la suite (S_n). T est dit une méthode d'extrapolation si la condition suivante est verifiée

$$\forall n,\ T_n = S \text{ si et seulement si } (S_n) \in \mathcal{K}_T.$$

Le noyau d'une transformation de suite, en général, dépend de plusieurs paramètres et, dans sa forme implicite, c'est une relation de la forme

$$R(S_n, \ldots, S_{n+q}; S) = 0$$

où R dépend aussi des paramètres $a_1, \ldots, a_p$.

Si l'on part de cette relation, et que l'on écrit le système suivant obtenu en utilisant $k+1$ termes de la suite et plus précisement $S_n, \ldots, S_{n+k}$ $(k = p+q)$,

$$R(S_i, \ldots, S_{i+q}, S) = 0\,, \qquad i = n, \ldots, n+p$$

En résolvant ce système on obtient la valeur de l'inconnue S (qui d'habitude dépend du premier indice n et de $k = p+q$ et qui a été, pour cette raison, appelé $T_k^{(n)}$). Evidemment, dans le cas général, $T_k^{(n)}$ est seulement une estimation de la limite S qui a été obtenue

par $k+1$ valeurs de la suite donnée et la connaissance des termes $S_{n+k+1}, S_{n+k+2}, \ldots$ n'est pas nécessaire.

L'on suppose donc avoir une méthode d'extrapolation $T_k^{(n)}$ obtenue à partir de la relation R et donc avoir $a_1, \ldots, a_p$ et $S = T_k^{(n)}$.

Il est évident que, à cause des conditions d'interpolation, la relation

$$R\left(S_i, \ldots, S_{i+q}; T_k^{(n)}\right) = 0,$$

qui est valable pour $i = n, \ldots, n+p$ est aussi valable pour $i = n+p+1, \ldots, n+k$. Donc les valeurs inconnues de la suite peuvent être obtenues en résolvant les équations qui sont données par cette relation (en partant de la valeur $i = n+p+1$) et en utilisant soit les valeurs connues (jusqu'à ce que ce ne soit plus possible) soit les valeurs précédentes prédites (voir Brezinski [9]).

Puisque les valeurs approchées dépendent de n et de k, on les indique avec la notation $S_{n+k+i}^{(k,n)}$ pour $i = 1, 2, \ldots$ (si n est fixé, il sera supprimé pour alléger la notation et de même pour k). Pour les calculer il faut résoudre l'équation

$$R\left(S_{n+p+1}, \ldots, S_{n+k}, S_{n+k+1}^{(k,n)}, T_k^{(n)}\right) = 0$$

puis après l'équation

$$R\left(S_{n+p+2}, \ldots, S_{n+k}, S_{n+k+1}^{(k,n)}, S_{n+k+2}^{(k,n)}, T_k^{(n)}\right) = 0$$

et ainsi de suite.

Avant de décrire comment l'on peut utiliser cette idée avec le procédé Δ^2 d'Aitken nous allons changer un peu les indices. On suppose connaître $n+k+1$ valeurs S_0, S_1, ..., S_n, ..., S_{n+k} d'une suite (S_n) et l'on veut utiliser des méthodes de prédiction pour obtenir des valeurs approchées pour S_{n+k+1}, S_{n+k+2}, ... Quelle est la signification des indices n et k dans ces méthodes? La valeur n indique le terme de départ de la suite que l'on veut considérer dans la méthode d'extrapolation. La valeur $k+1$, est le nombre de termes successifs (à partir de S_n) qui doivent être utilisés dans la méthode d'extrapolation.

Le procédé Δ^2 d'Aitken

Prenons d'abord en considération la méthode d'extrapolation la plus connue, c'est à dire le procédé Δ^2 d'Aitken. Pour cette méthode l'on a $p = q = 1$ $(k = 2)$ et la relation R est

$$R(u_i, u_{i+1}; u) = u_{i+1} - u + a_1(u_i - u) = 0 \quad \text{pour } i = 0, 1, \ldots.$$

Si l'on prend les valeurs S_n, S_{n+1}, S_{n+2}, on peut construire le système qui calcule l'estimation de la limite S de la façon suivante

$$\begin{cases} S_{n+2} &= S - a_1(S_{n+1} - S) \\ S_{n+1} &= S - a_1(S_n - S). \end{cases}$$

En résolvant ce système on obtient

$$a_1 = -\frac{\Delta S_{n+1}}{\Delta S_n} \qquad \text{et}$$
$$S = T_2^{(n)} = S_{n+2} - \frac{(\Delta S_{n+1})^2}{\Delta^2 S_n}$$

Si maintenant on résoud les équations

$$R\left(S_{n+i+1}^{(n)}, S_{n+i+2}^{(n)}; S\right) = S_{n+i+2}^{(n)} - S + a_1\left(S_{n+i+1}^{(n)} - S\right) = 0$$

pour $i = 1, 2, \ldots$, avec les valeurs de $S = T_2^{(n)}$ et a_1 que l'on vient de calculer, on obtient l'algorithme suivant

$$\begin{aligned} S_i^{(n)} &= S_i\,, & i &= 0, \ldots, n+2 \\ S_{n+i+2}^{(n)} &= S_{n+2} + \frac{\Delta S_{n+1}}{\Delta S_n} \cdot \left(S_{n+i+1}^{(n)} - S_{n+1}\right)\,, & i &= 1, 2, \ldots \end{aligned} \tag{1.4}$$

Une nouvelle suite $\left(S_{n+i+2}^{(n)}\right)$ est construite en utilisant les valeurs précédentes prédites et les valeurs S_n, S_{n+1} et S_{n+2} de la suite donnée (quatre valeurs sont donc nécessaires à chaque étape). Cet algorithme a été donné par Brezinski dans [9].

Dans cet article il y a aussi étudié les erreurs et il a donné les théorèmes suivants

Théorème 1.12 *Soit (S_n) une suite convergente telle que $\exists M$, N, $\forall n \geq N$, $|\Delta S_{n+1}/\Delta S_n| \leq M$. Alors $\forall i \geq 1$, $\lim_{n\to\infty}\left(S_{n+i+2} - S_{n+i+2}^{(n)}\right) = 0$.*

Donc plus la valeur de n est grande et plus les valeurs prédites seront meilleures. Il a donné aussi le résultat suivant

Théorème 1.13 *Soit (S_n) une suite convergente telle qu'il existe a avec $\lim_{n\to\infty} \Delta S_{n+1}/\Delta S_n = a$. Alors $\forall i \geq 1$, $\lim_{n\to\infty}\left(S_{n+i+2} - S_{n+i+2}^{(n)}\right)\Big/\Delta S_{n+1} = 0$. Si en plus $a \neq 0$, alors $\forall i \geq 1$, $\lim_{n\to\infty}\left(S_{n+i+2} - S_{n+i+2}^{(n)}\right)\Big/\Delta S_{n+i+1} = 0$.*

Si l'on prend la suite $S_n = 0.9^n/(n+1)$ les résultats donnés dans [9] montrent bien que la précision diminue quand n est fixé et que i augmente. Cela ne doit pas étonner car, dans le cas suivant, Brezinski a démontré que

Théorème 1.14 *Soit (S_n) une suite qui converge vers S et telle que $\forall n$, $\Delta S_n \leq 0$ et*

$$\frac{\Delta S_1}{\Delta S_0} \leq \frac{\Delta S_2}{\Delta S_1} \leq \cdots < 1.$$

Alors, $\forall n$

$$S - S^{(n)} \leq \cdots \leq S_{n+4} - S_{n+4}^{(n)} \leq S_{n+3} - S_{n+3}^{(n)} \leq 0$$

où $S^{(n)} = \lim_{i\to\infty} S_{n+i+2}^{(n)}$.

Ces théorèmes montrent que, sous certaines conditions faibles, si n augmente alors les valeurs prédites sont meilleures et quand n est fixé et que i augmente, la précision du terme prédit diminue.

Les conditions de ce théorème sont toujours verfiées par les suites totalement monotones, c'est à dire les suites pour lesquelles $\forall k, n, (-1)^k \Delta^k S_n \geq 0$. La condition $\Delta S_{n+1}/\Delta S_n < 1$ pour tout n n'est pas restrictive car si $\Delta S_{n+1}/\Delta S_n = 1$ pour certains indices n alors $\forall i \geq n$, $\Delta S_i = \Delta S_n$ et le suite ne peut pas converger. Les suites totalement monotones satisfont aussi aux théorèmes 1.12 et 1.13.

Maintenant nous allons proposer un algorithme pour utiliser la prédiction avec le procédé Δ^2 d'Aitken qui, par rapport à (1.4) pour le calcul d'un terme, n'utilise que les trois valeurs prédites précédentes.

Naturellement (1.4) est aussi valable si l'on pose $i-1$ à la place de i et donc on a

$$S_{n+i+1}^{(n)} = S_{n+2} + \frac{\Delta S_{n+1}}{\Delta S_n} \cdot \left(S_{n+i}^{(n)} - S_{n+1}\right). \tag{1.5}$$

Soustrayant la relation (1.5) de la relation (1.4) on obtient

$$S_{n+i+2}^{(n)} = S_{n+i+1}^{(n)} + \frac{\Delta S_{n+1}}{\Delta S_n} \cdot \left(S_{n+i+1}^{(n)} - S_{n+i}^{(n)}\right). \tag{1.6}$$

La relation (1.6) est aussi valable si l'on pose encore $i-1$ à la place de i et la nouvelle relation permet de calculer le rapport $\Delta S_{n+1}/\Delta S_n$ de la façon suivante

$$\frac{\Delta S_{n+1}}{\Delta S_n} = \frac{\Delta S_{n+i}^{(n)}}{\Delta S_{n+i-1}^{(n)}}.$$

Donc, en remplacant ce rapport dans (1.6), l'on obtient l'algorithme suivant

$$\begin{aligned} S_i^{(n)} &= S_i\,, & i &= 0, \ldots, n+2 \\ S_{n+i+2}^{(n)} &= S_{n+i+1}^{(n)} + \frac{\left(\Delta S_{n+i}^{(n)}\right)^2}{\Delta S_{n+i-1}^{(n)}}\,, & i &= 1, 2, \ldots \end{aligned}$$

E-algorithme

Maintenant nous supposons que la relation R soit de la forme

$$S_n - S - a_1\, g_1(n) - \cdots - a_k\, g_k(n) = 0$$

où les suites auxiliaires $(g_i(n))$ peuvent aussi dépendre de (S_n).

En écrivant cette relation pour $k+1$ termes de la suite (S_n) et en résolvant le système, on obtient la transformation E (voir Brezinski [4]) qui permet de calculer une estimation de la limite S par

$$E_k^{(n)} = \frac{\begin{vmatrix} S_n & \cdots & S_{n+k} \\ g_1(n) & \cdots & g_1(n+k) \\ \vdots & & \vdots \\ g_k(n) & \cdots & g_k(n+k) \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ g_1(n) & \cdots & g_1(n+k) \\ \vdots & & \vdots \\ g_k(n) & \cdots & g_k(n+k) \end{vmatrix}}.$$

En suivant le même procédé que pour le Δ^2 on obtient, pour $i = 1, 2, \ldots$

$$S_{n+k+i}^{(k,n)} = -\frac{\begin{vmatrix} S_n & \cdots & S_{n+k} & 0 \\ 1 & \cdots & 1 & 1 \\ g_1(n) & \cdots & g_1(n+k) & g_1(n+k+i) \\ \vdots & & \vdots & \vdots \\ g_k(n) & \cdots & g_k(n+k) & g_k(n+k+i) \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ g_1(n) & \cdots & g_1(n+k) \\ \vdots & & \vdots \\ g_k(n) & \cdots & g_k(n+k) \end{vmatrix}}.$$

Si $g_1(n+k+i), \ldots, g_k(n+k+i)$ dépendent aussi de $S_0, \ldots, S_{n+k+i-1}$, on doit remplacer dans leurs expressions S_{n+k+1} par $S_{n+k+1}^{(k,n)}, \ldots, S_{n+k+i-1}$ par $S_{n+k+i-1}^{(k,n)}$.

Les valeurs prédites $S_{n+k+i}^{(k,n)}$ peuvent se calculer récursivement à l'aide de la généralisation du schéma de Neville-Aitken pour l'interpolation polynomiale qui a été donné par Mühlbach [36]. C'est l'algorithme Mühlbach-Neville-Aitken (appelé aussi MNA), dont un cas particulier est le E-algorithme, voir Brezinski [5]. Si l'on assume que les valeurs $S_0, \ldots, S_m$ soient connues, alors pour $j > m$ et $n + k \leq m$, cet algorithme a la forme suivante

$$S_j^{(0,n)} = S_n\,, \quad g_{0,i}^{(n)} = g_i(n) - g_i(j)\,,$$
$$n = 0, 1, \ldots;\ i = 1, 2, \ldots$$

$$S_j^{(k,n)} = S_j^{(k-1,n)} - \frac{S_j^{(k-1,n+1)} - S_j^{(k-1,n)}}{g_{k-1,k}^{(n+1)} - g_{k-1,k}^{(n)}} \cdot g_{k-1,k}^{(n)}\,,$$
$$n = 0, 1, \ldots;\ k = 1, 2, \ldots$$

$$g_{k,i}^{(n)} = g_{k-1,i}^{(n)} - \frac{g_{k-1,i}^{(n+1)} - g_{k-1,i}^{(n)}}{g_{k-1,k}^{(n+1)} - g_{k-1,k}^{(n)}} \cdot g_{k-1,k}^{(n)}\,,$$
$$n = 0, 1, \ldots;\ k = 1, 2, \ldots;\ i > k.$$

Puisque les règles de cet algorithme, à part les initialisations, sont les mêmes de celles du E-algorithme, on peut utiliser la subroutine EALGO que nous avons donné dans [14].

Pour ce qui concerne la convergence on a (voir Brezinski [9])

Théorème 1.15 *Soit (S_n) une suite convergente. Si $\forall i$, il existe $b_i \neq 1$ tel que $\lim_{n\to\infty} g_i(n+1)/g_i(n) = b_i$ et si $\forall j \neq i$, $b_j \neq b_i$ alors $\forall i, k$, $\lim_{n\to\infty} \left(S_{n+k+i}^{(k,n)} - S_{n+k+i}\right) = 0$.*

Si l'on prend $g_i(n) = x_n^i$ alors le MNA-algorithme se simplifie et devient le schéma de Neville-Aitken pour l'interpolation polynomiale et l'on aura, pour $j > m$ et $n + k \leq m$

$$S_j^{(0,n)} = S_n \,, \qquad n = 0, 1, \ldots$$

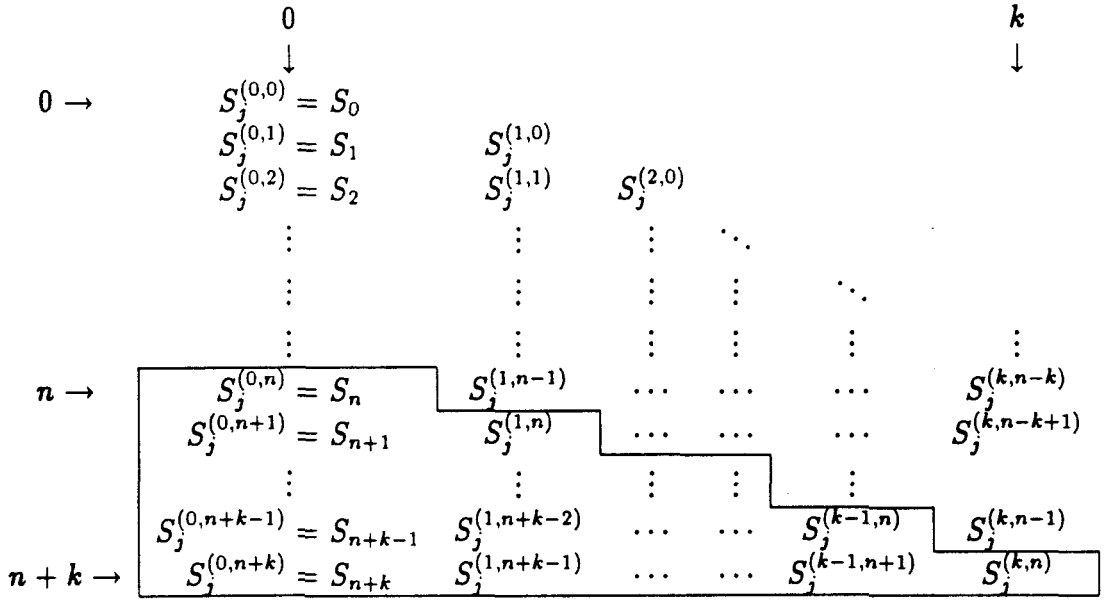
$$S_j^{(k,n)} = S_j^{(k-1,n)} - \frac{S_j^{(k-1,n+1)} - S_j^{(k-1,n)}}{x_{n+k} - x_n} \cdot (x_n - x_j) \,,$$

$$n = 0, 1, \ldots ; \; k = 1, 2, \ldots$$

Dans tous les cas, si $n + k > m$, alors les termes inconnus $S_{m+1}, \ldots, S_{n+k}$ doivent être remplacés par les termes prédits correspondants.

Si $\exists b \neq 0, \pm 1$ tel que $\lim_{n\to\infty} x_{n+1}/x_n = b$ alors les conditions du théorème 1.15 sont toujours satisfaites.

Les nombres $S_j^{(k,n)}$ du schéma récursif peuvent être placés dans un tableau à double entrée comme suit



L'indice qui est mis habituellement en bas et qui dénote la colonne, a été placé ici comme premier indice en haut. La deuxième indice en haut indique la diagonale descendante et l'indice en bas j indique le terme inconnu S_j de la suite (S_n) que l'on veut calculer.

1.3.1 Applications

Dans [34], avec Morandi Cecchi et Scenna, nous avons appliqué la prédiction à la détermination de la solution d'un problème parabolique d'évolution.

Il s'agissait de chercher la solution de

$$\begin{cases} u_t - \Delta u = f & (x,t) \in Q_T \\ u|_{\partial\Omega} = 0, & t \in [0,T] \\ u(x,0) = u_0(x) & x \in \Omega \end{cases}$$

où $u = u(x,t), f = f(x,t), Q_T = \Omega \times [0,T]$ et $\Omega \subset \mathbf{R}^d$, $(d = 1,\ldots,n)$.

Sous certaines conditions, décrites par Morandi Cecchi et Nociforo [32], la *formulation variationnelle* suivante de ce problème a une solution unique

trouver $u \in L^2(0,T;H_0^1(\Omega))$ tel que

$$\int_0^T [(\nabla u, \nabla\phi) - (u,\phi_t)]\,\mathrm{dt} = \int_0^T (f,\phi)\mathrm{dt} + (u_0(x),\phi(0)), \tag{1.7}$$
$$\forall\phi \in L^2\left(0,T;H_0^1(\Omega)\right), \phi' \in L^2\left(0,T;L^2(\Omega)\right), \phi(T) = 0,$$

où $(\cdot,\cdot)$ est le produit scalaire dans $L^2(\Omega)$.

On considère une discrétisation par rapport à t avec les valeurs $t_0 = 0 < t_1 < \cdots < t_N = T$. Dans chaque *slab* $\Omega \times [t_n, t_{n+1}]$ on prend une discrétisation par éléments finis du problème variationnel et les fonctions-test dans $S_h^p(\Omega) \otimes P^q([t_n,t_{n+1}])$ où $S_h^p(\Omega)$ est l'espace des éléments finis des polynômes par morceaux de degré p dans Ω, h le paramètre de la maille et $P^q([t_n,t_{n+1}])$ l'ensemble des polynômes de degré q dans $[t_n,t_{n+1}]$.

Passant d'un slab au suivant, la continuité dans le temps de la solution approchée est imposée. Donc on obtient un procédé itératif qui donne la solution au temps t_{n+1} à partir de la solution au temps t_n. Cette méthode est équivalente à appliquer la méthode des éléments finis de Galerkin en espace et après la méthode des différences finies en temps. Cette méthode a été appelée *space-time finite element method.*

Si f est indépendant de t on obtient une méthode itérative de la forme

$$U^{n+1} = MU^n + b \tag{1.8}$$

où U^n est la solution au temps $t_n = n\cdot\Delta t\,(n = 0,1,\ldots,N-1)$ aux nœuds de la subdivision de Ω.

La précision de cette méthode peut évidemment augmenter si l'on prend des polynômes en temps de degré plus grand que un, mais alors le calcul devient très lourd. D'autre part la condition de stabilité peut imposer un interval de temps très petit.

Donc, pour étudier le problème (1.7) pour des valeurs significatives de T, on utilise la méthode (1.8) et après on applique la prédiction pour évaluer la valeur de la solution U^N au temps T en ayant calculé seulement très peu de termes de la suite (U^n), par exemple $U^0, U^1, \ldots, U^m$ avec $m \ll N$.

Soit $\overline{U}^n \in \mathbf{R}^{N_h}$ le vecteur des coefficients de U^n. On considère ce vecteur composante par composante. Donc on aura des suites de nombres réels qui convergent vers les composantes

correspondantes de la solution du vecteur limite $\overline{U}$. Soit $(\overline{U}^n)_i$ la i-ème composante de $\overline{U}^n$.

Pour avoir la certitude que la prédiction (par exemple le procédé Δ^2 d'Aitken) puisse être bien appliquée à ce cas, il faut que certaines conditions sur les rapports suivants soient verifiées

$$\left(\Delta\overline{U}^{n+1}\right)_i / \left(\Delta\overline{U}^n\right)_i, \quad i = 1, ..., N_h.$$

La matrice M de (1.8) a N_h valeurs propres réelles $|\lambda_1| \geq \cdots \geq |\lambda_{N_h}|$, avec les vecteurs propres correspondants $\vec{x}_1, \ldots, \vec{x}_{N_h}$. Puisque la forme de la matrice M dépend de la triangularisation de Ω, on peut toujours avoir $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_{N_h}|$. On écrit $\Delta\overline{U}^0$ comme combinaison linéaire des vecteurs propres $\Delta\overline{U}^0 = \beta_1\vec{x}_1 + \cdots + \beta_{N_h}\vec{x}_{N_h}$, avec $\beta_1 \neq 0$.

On aura $\Delta\overline{U}^{n+1} = M\Delta\overline{U}^n = M^{n+1}\Delta\overline{U}^0$, et donc

$$\begin{aligned}
\Delta\overline{U}^{n+1} &= \beta_1 M^{n+1}\vec{x}_1 + \beta_2 M^{n+1}\vec{x}_2 + ... + \beta_{N_h} M^{n+1}\vec{x}_{N_h} \\
&= \beta_1\lambda_1^{n+1}\vec{x}_1 + \beta_2\lambda_2^{n+1}\vec{x}_2 + \cdots + \beta_{N_h}\lambda_{N_h}^{n+1}\vec{x}_{N_h} \\
&= \lambda_1^{n+1}\left(\beta_1\vec{x}_1 + \beta_2\left(\frac{\lambda_2}{\lambda_1}\right)^{n+1}\vec{x}_2 + \cdots + \beta_{N_h}\left(\frac{\lambda_{N_h}}{\lambda_1}\right)^{n+1}\vec{x}_{N_h}\right).
\end{aligned}$$

Si l'on considère cette égalité composante par composante, puisque $|\lambda_i/\lambda_1| < 1$, pour $i = 2, \ldots, N_h$, on aura

$$\lim_{n\to\infty}\left(\Delta\overline{U}^{n+1}\right)_i / \left(\Delta\overline{U}^n\right)_i = \lambda_1 \qquad \text{avec } |\lambda_1| < 1.$$

D'après les théorèmes démontrés par Brezinski [9] on aura

$$\lim_{n\to\infty}\left(\left(\overline{U}^{n+j+2}\right)_i - \left(\overline{U}^{n+j+2,(n)}\right)_i\right) = 0 \quad \text{et}$$

$$\lim_{n\to\infty}\frac{\left(\overline{U}^{n+j+2}\right)_i - \left(\overline{U}^{n+j+2,(n)}\right)_i}{\left(\Delta\overline{U}^{n+j}\right)_i} = 0 \quad \text{pour} \quad j = 1,2,\ldots;\ i = 1,\ldots,N_h$$

et donc la prédiction peut donner de bons résultats. En effet les exemples suivants montrent que la prédiction arrive à donner la même précision que la méthode elle même avec bien moins de calculs.

Les exemples sont les suivants

- les exemples 1 et 3 sont des problèmes aux limites avec conditions homogènes de Dirichlet
- l'exemple 2 est un problème aux limites avec conditions homogènes de Neumann.

La théorie donnée dans [9] montre que la précision de la prédiction peut diminuer quand on fixe n et augmente j. L'exemple 1 montre en effet, que la prédiction doit commencer avec un n assez grand pour assurer une bonne convergence. Par contre, dans

les autres deux exemples, les premières trois valeurs dans le temps considerés ($\overline{U}^0, \overline{U}^1, \overline{U}^2$ dans l'exemple 2 et $\overline{U}^1, \overline{U}^2, \overline{U}^3$ dans l'exemple 3 car $\overline{U}^0$ est le vecteur nul) sont suffisantes pour prédire $\overline{U}(T)$ avec, au moins, la même précision de la méthode.

Dans les tableaux suivants, la solution exacte est comparée avec les deux solutions approchées, l'une obtenue avec le *space-time finite element method* et l'autre avec la prédiction. Pour les solutions approchées on donne le nombre des chiffres décimaux exacts ($-\log_{10}|$erreur relative$|$). N représente le nombre de pas en temps nécessaires à la *space-time finite element method* pour donner $\overline{U}(T)$. A cause de la symétrie dans le comportement de la solution, dans tous les tableaux seulement la moitié des nœuds traités en espace sont donnés.

Exemple 1. Prédiction avec $n = 100$ et $n = 300$.

domaine $\Omega = (-l, l)$, $N_h = 29$, $T = 0.05s$, $N = 500$
condition initiale $u_0(x) = V_0\,(l^2 - x^2)\,/l^2$
solution exacte (with $V_0 = 0.5$, $l = 1$, $m = 10$)

$$u(x,t) = 32V_0/\pi^3 \sum_{n=0}^{\infty} \left((-1)^n/(2n+1)^3\right) e^{-m(2n+1)^2\pi^2 t/4l^2} \cos\left((2n+1)\pi x/2l\right)$$

Exemple 1 - Prédiction avec $n = 100$ et $n = 300$					
	Solution exacte		Sol. approx.	Sol. prédite	
nœud	$\overline{U}^0$	$\overline{U}^N$		$n=100$	$n=300$
2	0.064444	0.015708	2.786	0.395	4.283
3	0.124444	0.031244	2.786	0.410	4.672
4	0.180000	0.046437	2.786	0.436	3.853
5	0.231111	0.061122	2.786	0.475	3.524
6	0.277778	0.075137	2.786	0.531	3.308
7	0.320000	0.088329	2.786	0.610	3.149
8	0.357778	0.100553	2.786	0.724	3.025
9	0.391111	0.111675	2.786	0.900	2.927
10	0.420000	0.121574	2.786	1.227	2.849
11	0.444444	0.130140	2.786	2.084	2.786
12	0.464444	0.137281	2.786	1.139	2.737
13	0.480000	0.142918	2.786	0.887	2.701
14	0.491111	0.146989	2.786	0.758	2.675
15	0.497778	0.149450	2.786	0.692	2.660
16	0.500000	0.150273	2.786	0.671	2.655

Exemple 2. Prédiction avec $n = 0$.

domaine $\Omega = (0, \pi)$, $N_h = 21$, $T = 1s$, $N = 200$
condition initiale $u_0(x) = \cos(x)$
solution exacte $u(x,t) = e^{-t}\cos(x)$

Exemple 2 - Prédiction avec $n = 0$				
	Solution exacte		Sol. approx.	Sol. prédite
nœud	$\overline{U}^0$	$\overline{U}^N$	n. de chiffres exacts	
1	1.000000	0.367879	2.539	2.539
2	0.987688	0.363350	2.539	2.539
3	0.951057	0.349874	2.539	2.539
4	0.891007	0.327783	2.539	2.539
5	0.809017	0.297621	2.539	2.539
6	0.707107	0.260130	2.539	2.539
7	0.587785	0.216234	2.539	2.539
8	0.453990	0.167014	2.539	2.539
9	0.309017	0.113681	2.539	2.539
10	0.156434	0.057549	2.539	2.539
11	0.000000	0.000000	all	all

Exemple 3. Prédiction avec $n = 1$.

domaine $\Omega = (0, 10)$, $N_h = 29$, $T = 3s$, $N = 300$, $m = 1$, $l = 10$
condition initiale $u_0(x) = 0$
solution exacte $u(x,t) = l^2\left(1 - e^{-\pi^2 mt/l^2}\right)\sin(\pi x/l)/(\pi^2 m)$

Exemple 3 - Prédiction avec $n = 1$				
	Solution exacte		Sol. approx.	Sol. prédite
nœud	$\overline{U}^0$	$\overline{U}^N$	n. de chiffres exacts	
2	0.000000	0.271423	4.891	4.891
3	0.000000	0.539872	4.891	4.891
4	0.000000	0.802406	4.891	4.891
5	0.000000	1.056149	4.891	4.891
6	0.000000	1.298320	4.891	4.891
7	0.000000	1.526267	4.891	4.891
8	0.000000	1.737491	4.891	4.891
9	0.000000	1.929680	4.891	4.891
10	0.000000	2.100726	4.891	4.891
11	0.000000	2.248756	4.891	4.891
12	0.000000	2.372149	4.891	4.891
13	0.000000	2.469552	4.891	4.891
14	0.000000	2.539897	4.891	4.891
15	0.000000	2.582416	4.891	4.891
16	0.000000	2.596640	4.891	4.891

1.4 Une application du E-algorithme

Le problème de Cauchy pour l'opérateur biharmonique est mal posé au sens d'Hadamard [28]. Lorsque les données initiales ne sont pas analytiques mais seulement différentiables un certain nombre de fois il peut sembler convenable de procéder comme suit: d'abord on approche la fonction qui exprime les données initiales par des polynômes et ensuite on résoud le problème en espérant que la solution obtenue de cette façon ne sera pas très différente de la solution du problème originel. Malheureusement cela est faux. On peut contourner cette difficulté ainsi: si les données ne sont pas exactement du type Cauchy, il est possible de résoudre le problème en imposant à la solution d'être uniformement bornée. La résolution du problème se réduit ainsi à la minimisation d'une certaine fonctionnelle avec des inégalités linéaires.

Habituellement ce problème est résolu par programmation linéaire ou par moindres carrés à partir des équations normales. Cette approche a été suivie dans [18, 19].

Dans [33] nous avons proposé une nouvelle méthode pour résoudre ce problème.

Soit C une courbe de Jordan lisse, simple, fermée et rectifiable dont l'intérieur G est supposé étoilé par rapport à l'origine. Soit $r = R(\theta)$ l'équation de C en coordonnées polaires. Le problème consiste à trouver une approximation de la solution $u = u(r, \theta)$ de

$$\begin{aligned} &\Delta^2 u = 0, \\ &u(R(\theta), \theta) = f(\theta) \\ &\frac{\partial u}{\partial n}(R(\theta), \theta) = g(\theta) \end{aligned}$$

où n est la direction de la normale intérieure à C et où on ne connaît que des approximations F, F_1 et G des fonctions réelles f, f' et g.

Le problème se réduit à approcher f par une fonction f^* qui peut s'exprimer comme combinaison linéaire

$$f^*(\theta) \;=\; c_0\phi_0(\theta) + c_1\phi_1(\theta) + \ldots + c_k\phi_k(\theta),$$

de $k+1$ fonctions $\phi_0, \phi_1, \ldots, \phi_k$, choisies à l'avance et où les coefficients $c_0, c_1, \ldots, c_k$ doivent être déterminés. La fonction f est connue aux m points distincts $P_j, j = 0, \ldots, m-1$. Nous voulons trouver $c_0, c_1, \ldots, c_k$ de sorte que l'égalité $f^*(\theta_j) = f(\theta_j)$ soit la plus vraie possible en tous les points. C'est à dire que nous voulons que

$$\| f^* - f \|_2^2 \;=\; \sum_{j=0}^{m-1} |f^*(\theta_j) - f(\theta_j)|^2$$

soit le plus petit possible.

Nous avons résolu ce problème de la façon suivante: nous définissons $3m$ fonctionnelles linéaires par

$$\begin{aligned} L_{1j}(\phi_\mu) &= \phi_\mu(\theta_j) \\ L_{2j}(\phi_\mu) &= \frac{d\phi_\mu}{d\theta}(\theta_j) \qquad j = 0, \ldots, m-1 \\ L_{3j}(\phi_\mu) &= \frac{\partial \phi_\mu}{\partial n}(\theta_j) \qquad \mu = 0, \ldots, 4N+1 \end{aligned}$$

avec

$$L_{1j}(f) = F(\theta_j), \quad L_{2j}(f) = F_1(\theta_j), \quad L_{3j}(f) = G(\theta_j).$$

Alors le système linéaire sur-déterminé représenté par $Ax = b$ est de la forme

$$\begin{pmatrix} \phi_0(\theta_j) & \phi_1(\theta_j) & \dots & \phi_{4N+1}(\theta_j) \\ \frac{d\phi_0}{d\theta}(\theta_j) & \frac{d\phi_1}{d\theta}(\theta_j) & \dots & \frac{d\phi_{4N+1}}{d\theta}(\theta_j) \\ \frac{\partial\phi_0}{\partial n}(\theta_j) & \frac{\partial\phi_1}{\partial n}(\theta_j) & \dots & \frac{\partial\phi_{4N+1}}{\partial n}(\theta_j) \end{pmatrix}(x) = \begin{pmatrix} F(\theta_j) \\ F_1(\theta_j) \\ G(\theta_j) \end{pmatrix}$$

$$j = 0, \dots, m-1$$

ou encore

$$\begin{pmatrix} L_{1j}(\phi_0) & L_{1j}(\phi_1) & \dots & L_{1j}(\phi_{4N+1}) \\ L_{2j}(\phi_0) & L_{2j}(\phi_1) & \dots & L_{2j}(\phi_{4N+1}) \\ L_{3j}(\phi_0) & L_{3j}(\phi_1) & \dots & L_{3j}(\phi_{4N+1}) \end{pmatrix}(x) = \begin{pmatrix} L_{1j}(f) \\ L_{2j}(f) \\ L_{3j}(f) \end{pmatrix}.$$

Ce système est résolu au sens des moindres carrés à l'aide du E-algorithme [4] dont on a modifié les initialisations selon l'algorithme de Mühlbach-Neville-Aitken [36].

Soit

$$L(f) = f(\theta), \ (f,g) = \sum_{j=0}^{m-1} (L_{1j}(f)L_{1j}(g) + L_{2j}(f)L_{2j}(g) + L_{3j}(f)L_{3j}(g))$$

où θ est l'angle dont les valeurs doivent être extrapolées.

Nous utilisons les formules récursives

$$\begin{aligned} f_k^{*(n)} &= f_{k-1}^{*(n)} + \phi_{k-1,k}^{(n)} \frac{f_{k-1}^{*(n)} - f_{k-1}^{*(n+1)}}{\phi_{k-1,k}^{(n+1)} - \phi_{k-1,k}^{(n)}} \\ \phi_{k,i}^{(n)} &= \phi_{k-1,i}^{(n)} + \phi_{k-1,k}^{(n)} \frac{\phi_{k-1,i}^{(n)} - \phi_{k-1,i}^{(n+1)}}{\phi_{k-1,k}^{(n+1)} - \phi_{k-1,k}^{(n)}} \\ & \qquad i = k+1, k+2, \dots \end{aligned}$$

avec les initialisations

$$\begin{aligned} f_0^{*(n)} &= (\phi_n, f)\frac{L(\phi_0)}{(\phi_n, \phi_0)} \\ \phi_{0,i}^{(n)} &= (\phi_n, \phi_i)\frac{L(\phi_0)}{(\phi_n, \phi_0)} - L(\phi_i) \end{aligned}$$

et nous obtenons

$$f_{4N+1}^{*(0)} = L(f^*).$$

Dans [33] on trouvera trois exemples numériques correspondants aux choix $f = r\cos\theta$, $f = 1 + x^2y$ et $f = 1 + \sinh y \sin x$. Le E-algorithme se compare très favorablement avec les autres méthodes en ce qui concerne la précision. Lorsque l'on a seulement besoin de certains points, l'algorithme est très performant du point de vue du temps de calcul et de plus la précision est meilleure que celle des autres méthodes.

1.5 Logiciels

Donnont maintenant les principes généraux qui régissent l'utilisation des sous-programmes mettant en œuvre les algorithmes d'extrapolation.

Prenons l'exemple de l'ε-algorithme, la situation étant exactement la même pour les autres algorithmes. La méthode la plus simple pour programmer l'ε-algorithm est de stocker dans la mémoire tous les $\varepsilon_k^{(n)}$. Partant d'un nombre donné de termes de la suite intiale, chaque colonne du tableau ε est calculée à partir de deux colonnes précedentes. Il faut remarquer que, d'après la règle de l'ε-algorithme, chaque colonne contient un terme de moins que la colonne précédente. Ainsi, partant de $S_0, \ldots, S_k$, on peut seulement calculer le tableau triangulaire de la figure (1.1). La même chose est vraie pour la forme progressive de l'algorithme: partant de $S_0, \ldots, S_k$ et de $\varepsilon_0^{(0)}, \ldots, \varepsilon_k^{(0)}$, on remplit le tableau diagonale descendante par diagonale descendante, chaque diagonale ayant un terme de moins que la diagonale au dessus d'elle.

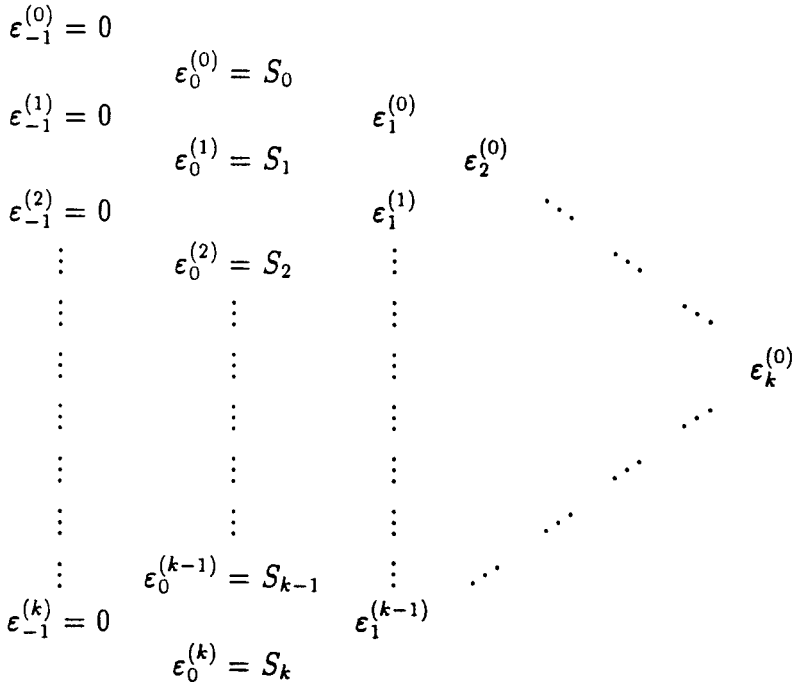


Figure 1.1: Le tableau ε.

Si un nouveau terme, S_{k+1}, est ajouté alors la nouvelle diagonale montante $\varepsilon_0^{(k+1)}$, $\varepsilon_1^{(k)}$, $\ldots$, $\varepsilon_k^{(1)}$, $\varepsilon_{k+1}^{(0)}$ est facilement calculée. La même chose est vraie pour la forme progressive de l'algorithme.

Supposons maintenant que nous voulions réduire l'encombrement mémoire. La façon la plus simple d'implémenter l'ε-algorithme est de garder seulement les deux dernières colonnes puis de calculer la nouvelle colonne et de faire un décalage. Cependant cette procédure souffre d'un inconvénient majeur: si l'on veut ajouter de nouveaux termes

$S_{k+1}, S_{k+2}, \ldots$ il faut alors recommencer tous les calculs. On peut éviter cet inconvénient en conservant dans la mémoire la dernière diagonale montante $\varepsilon_0^{(k)}, \varepsilon_1^{(k-1)}, \varepsilon_2^{(k-2)}, \ldots, \varepsilon_k^{(0)}$. Ainsi, quand on ajoute un nouveau terme $\varepsilon_0^{(k+1)} = S_{k+1}$, $\varepsilon_1^{(k)}$ est calculé à partir de $\varepsilon_{-1}^{(k+1)}, \varepsilon_0^{(k)}$ et de $\varepsilon_0^{(k+1)}$. Ensuite $\varepsilon_2^{(k-1)}$ est calculé à partir de $\varepsilon_0^{(k)}, \varepsilon_1^{(k-1)}$ et de $\varepsilon_1^{(k)}$. Puis, $\varepsilon_3^{(k-2)}$ est obtenu à partir de $\varepsilon_1^{(k-1)}, \varepsilon_2^{(k-2)}$ et de $\varepsilon_2^{(k-1)}$ et ainsi de suite. Ainsi, une nouvelle diagonale montante remplace peu à peu l'ancienne. Afin de programmer plus facilement cette technique, il est nécessaire d'utiliser des mémoires temporaires comme il a été expliqué par Wynn [47, 50].

Certains algorithmes sont plus difficiles à programmer et nécessitent en particulier le stockage de plusieurs diagonales ainsi que d'autres quantités auxiliaires ou même de tableaux. Cependant, pour l'utilisateur, la situation est exactement la même et cette technique permet une utilisation très simple des algorithmes et des sous-programmes correspondants.

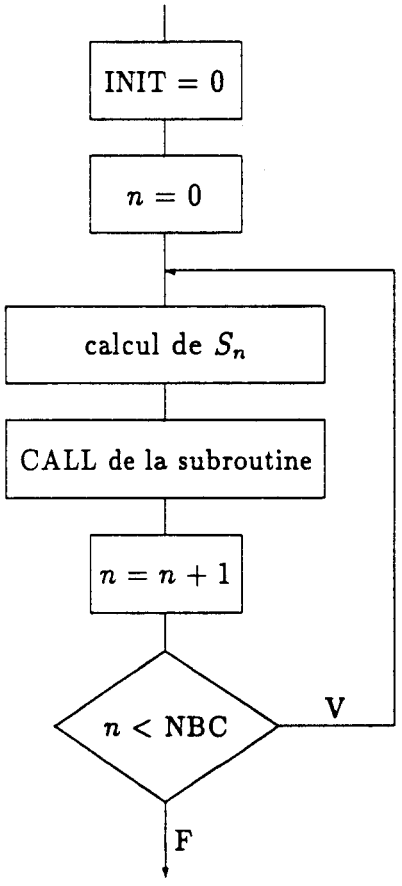


Figure 1.2: Organigramme des programmes.

On utilise les sous-programmes *en parallèle* avec le calcul des termes successifs de la

suite (S_n). Ces termes successifs sont calculés dans une boucle. Après le calcul de chaque nouveau terme de la suite, on fait un CALL du sous-programme et celui-ci calcule la nouvelle diagonale montante aussi loin que possible (ou jusqu'à une certaine colonne si c'est le choix de l'utilisateur). Ensuite le sous-programme retourne dans le programme principal, dans la boucle où sont calculés les termes de la suite. Naturellement, avant le début de la boucle, il faut indiquer au sous-programme (en mettant INIT à 0, et le sous-programme le mettra ensuite lui même à 1) qu'on lui donne une nouvelle suite à traiter. Les formes normales de tous les algorithmes ont été programmées de cette façon, voir Brezinski [3].

Ainsi l'utilisation des sous-programmes est décrite par l'organigramme de la figure 1.2, où NBC est le nombre maximum d'appels (voir aussi le pseudocode donné plus loin).

Les sous-programmes qui utilisent une règle particulière pour éviter l'instabilité numérique sont plus difficiles à programmer. En effet les règles particulières sont seulement valables quand deux quantités consécutives dans une colonne sont presque égales et elles ne peuvent plus être utilisées dans le cas de trois quantités presque égales. Il faut donc localiser ces points d'instabilité comme l'a fait Wynn [50] (voir aussi Brezinski [3]). Cependant l'utilisation d'un tel sous-programme est exactement la même que celle d'un sous-programme sans règles particulières.

Nous avons programmé en FORTRAN 77 les principaux algorithmes d'extrapolation scalaires et vectoriels. On peut les trouver dans la diskette contenue dans le livre [14]. Voici leur liste par ordre alphabétique

ACCES	ACCES algorithme
COMPOS	Transformations composites de suites
CRPA	CRPA algorithme
EALGO	E-algorithme
EPSRHA	ϱ, ε-algorithmes et généralisations simultanées
EPSRHO	ϱ-algorithme, ε-algorithme ou généralisations
EPSTOP	ε-algorithme topologique
EPSVEC	ε-algorithme vectoriel
GTRAN	G-transformation
HALGO	H-algorithme
IDELTA	Procédé Δ^2 d'Aitken itéré
INVLAP	Inversion de la transformée de Laplace
LEVINT	Transformations de Levin
OMEGA	ω-algorithme
OVERHO	Procédé d'Overholt
QDALGO	qd-algorithme
RICHAR	Procédé de Richardson
RSALGO	rs-algorithme
SEALGO	E-algorithme simultané
SELECT	Selection automatique
THETA	Θ-algorithme
VEALGO	E-algorithme vectoriel
VGRAN	G-transformation vectorielle

L'utilisation de ces sous-programmes peut se faire selon le pseudo-code suivant

1. **Spécifications des variables et des constantes**
   liste des variables et des paramètres entiers
   liste des variables et des paramètres réels
2. **Spécifications des paramètres**
   MAXCOL, NBC, EPS, ...
3. **Spécifications des vecteurs et des matrices**
   liste et dimensions des tableaux entiers
   liste et dimensions des tableaux réels
4. **Initialisations**
   INIT = 0
   autres initialisations demandées par l'algorithme
5. **Pour** $n = 0$ **à** NBC-1 **faire**
   calcul de S_n
   calcul d'autres quantités nécessaires à l'algorithme
   CALL du sous-programme
   contrôl du flag IER
   impréssion des résultats intermédiaires
   fin pour

BIBLIOGRAPHIE

[1] A. C. AITKEN, *On Bernoulli's numerical solution of algebraic equations*, Proc. R. Soc. Edinb., 46 (1926) 289–305.

[2] C. BREZINSKI, *Accélération de suites à convergence logarithmique*, C. R. Acad. Sci. Paris, 273 (1971) A:727–730.

[3] C. BREZINSKI, *Algorithmes d'Accélération de la Convergence*, Etude Numérique, Editions Technip, Paris, 1978.

[4] C. BREZINSKI, *A general extrapolation algorithm*, Numer. Math. 35 (1980) 175–187.

[5] C. BREZINSKI, *The Mühlbach-Neville-Aitken algorithm and some extensions*, BIT, 20 (1980) 444–451.

[6] C. BREZINSKI, *Some new convergence acceleration methods*, Math. Comput. 39 (1982) 133–145.

[7] C. BREZINSKI, *Error control in convergence acceleration processes*, IMA J. Numer. Anal. 3 (1983) 65–80.

[8] C. BREZINSKI, *Vitesse de convergence d'une suite* Rev. Roumaine Math. Pures Appl. 30 (1985) 403–417.

[9] C. BREZINSKI, *Prediction properties of some extrapolation methods*, Appl. Numer. Math., 1 (1985) 457–462.

[10] C. BREZINSKI, *Contraction properties of sequence transformations*, Numer. Math. 54 (1989) 565–574.

[11] C. BREZINSKI, *A survey of iterative extrapolation by the E-algorithm*, Det Kong. Norske Vid. Selsk. Skr. 2 (1989) 1–26.

[12] C. BREZINSKI, S. PASZKOWSKI, *Optimal linear contractive sequence transformations*, J. Comput. Appl. Math.

[13] C. BREZINSKI, M. REDIVO ZAGLIA, *Construction of extrapolation processes*, Appl. Numer. Math., 8 (1991) 11–23.

[14] C. BREZINSKI, M. REDIVO ZAGLIA, *Extrapolation Methods. Theory and Practice*, North-Holland, Amsterdam, 1991.

[15] C. BREZINSKI, M. MORANDI CECCHI, M. REDIVO ZAGLIA, *The reverse bordering method*, soumis.

[16] C. BREZINSKI, M. REDIVO ZAGLIA, H. SADOK, *A breakdown-free Lanczos type algorithm for solving linear systems*, Numer. Math., à paraître.

[17] C. BREZINSKI, G. WALZ, *Sequences of transformations and triangular recursion schemes with applications in numerical analysis*, J. Comput. Appl. Math., 34 (1991) 361–383.

[18] J. R. CANNON, M. MORANDI CECCHI, *The numerical solutions of some biharmonic problems by mathematical programming techniques*, SIAM J. Numer. Anal., 3 (1966) 451–466.

[19] J. R. CANNON, M. MORANDI CECCHI, *Numerical experiments on the solution of some biharmonic problems by mathematical programming techniques*, SIAM J. Numer. Anal., 4 (1967) 147–154.

[20] F. CORDELLIER, *Démonstration algébrique de l'extension de l'identité de Wynn aux tables de Padé non normales*, In L. Wuytack, ed., *Padé Approximation and its Applications*, volume 765 of *LNM*, pp. 36–60, Berlin, 1979. Springer-Verlag.

[21] F. CORDELLIER, *Particular rules for the vector ε-algorithm*, Numer. Math., 27 (1977) 203–207.

[22] J. P. DELAHAYE, B. GERMAIN-BONNE, *Résultats négatifs en accélération de la convergence*, Numer. Math. 35 (1980) 443–457.

[23] V. N. FADDEEVA, *Computational methods of linear algebra*, Dover, New-York, 1959.

[24] W. GAUTSCHI, *A note on the successive remainders of the exponential series*, El. Math. 37 (1982) 46–49.

[25] J. GILEWICZ, *Numerical detection of the best Padé approximant and determination of the Fourier coefficients of insufficiently sampled functions*, in "Padé Approximants and their Applications", P. R. Graves-Morris ed., Academic Press, New-York, 1973, pp. 99–103.

[26] I. S. GRADSHTEYN, M. RYZHIK, *Table of integrals, series, and products*, Academic Press, New-York, 1980.

[27] T. HÅVIE, *Generalized Neville type extrapolation schemes*, BIT 19 (1979) 204–213.

[28] J. HADAMARD, *Le problème de Cauchy et les équations aux derivées partielles linéaires hyperboliques*, Herman, Paris, 1932.

[29] C. KOWALEWSKI, *Accélération de la convergence pour certaines suites à convergence logarithmique*, in *"Padé approximation and its applications. Amsterdam 1980"*, M.G. de Bruin and H. Van Rossum eds., LNM 888, Springer-Verlag, Berlin, 1981, pp. 263–272.

[30] A. LEMBARKI, *Accélération des fractions continues*, Thèse d'Etat, Université des Sciences et Techniques de Lille Flandres-Artois, 1987.

[31] A. M. LITOVSKY, *Accélération de la convergence des ensembles synchronisables*, Thèse, Université des Sciences et Techniques de Lille Flandres-Artois, 1989.

[32] M. MORANDI CECCHI, R. NOCIFORO, *Discrete finite elements method in space-time domain for parabolic linear problem*, Le Matematiche, à paraître.

[33] M. MORANDI CECCHI, M. REDIVO ZAGLIA, *Mathematical programming techniques to solve biharmonic problems by a recursive projection algorithm*, J. Comput. Appl. Math., 32 (1990) 191–201.

[34] M. MORANDI CECCHI, M. REDIVO ZAGLIA, G. SCENNA, *Approximation of the numerical solution of parabolic problems*, in "Proceedings of the 13th IMACS World Congress. Computational and applied mathematics I - Algorithms and theory", C. Brezinski and U. Kulish, Elsevier, Amsterdam, à paraître.

[35] S. L. MOSHIER, *Methods and programs for mathematical functions*, Ellis Horwood, Chichester, 1989.

[36] G. MÜHLBACH, *Neville-Aitken algorithms for interpolating by functions of Čebyšev-systems in the sense of Newton and in a generalized sense of Hermite*, in "Theory of Approximation with Application", A. G. Law and B. N. Sahney eds., Academic Press, New-York, 1976, pp. 200–212.

[37] G. OPFER, *On monotonicity preserving linear extrapolation sequences*, Computing 5 (1970) 259–266.

[38] K. J. OVERHOLT, *Extended Aitken acceleration*, BIT 5 (1965) 122–132.

[39] M. REDIVO ZAGLIA, *Particular rules for the Θ-algorithm*, Numerical Algorithms, à paraître.

[40] D. SHANKS, *Non linear transformations of divergent and slowly convergent sequences*, J. Math. Phys., 34 (1955) 1–42.

[41] A. SIDI, D. LEVIN, *Prediction properties of the t-transformation*, SIAM J. Numer. Anal., 20 (1983) 589–598.

[42] D. A. SMITH, W. F. FORD, *Acceleration of linear and logarithmic convergence*, SIAM J. Numer. Anal., 16 (1979) 223–240.

[43] D. A. SMITH, W. F. FORD, *Numerical comparison of nonlinear convergence accelerators*, Math. Comput., 38 (1982) 481–499.

[44] B. A. SZABO, *Estimation and control of erreur based on p convergence*, in *"Accuracy estimates and adaptative refinements in finite elements computation"*, I. Babuška and al. eds., Wiley, Chichester, 1986, pp. 61-78.

[45] E. J. WENIGER, *Nonlinear sequence transformations for the acceleration of convergence and the summation of divergent series*, Comput. Phys. Rep. 10 (1989) 189–373.

[46] P. WYNN, *On a device for computing the $e_m(S_n)$ transformation*, MTAC, 10 (1956) 91–96.

[47] P. WYNN, *Acceleration techniques in numerical analysis, with particular reference to problems in one independent variable*, in *Proc. IFIP Congress*, North-Holland, Amsterdam, 1962, pp. 149–156.

[48] P. WYNN, *Singular rules for certain nonlinear algorithms*, BIT, 3 (1963) 175–195.

[49] P. WYNN, *Upon systems of recursions which obtain among the quotients of the Padé table*, Numer. Math., 8 (1966) 264–269.

[50] , P. WYNN *An arsenal of Algol procedures for the evaluation of continued fractions and for effecting the epsilon algorithm*, Chiffres, 4 (1966) 327–362.

Chapitre 2

POLYNÔMES ORTHOGONAUX

Soit c une fonctionnelle linéaire sur l'espace des polynômes complexes. La fonctionnelle c est complètement déterminée par ses moments

$$c\left(x^i\right) = c_i\,, \qquad i = 0, 1, \ldots$$

Soit $\{P_k\}$ la famille de polynômes orthogonaux formels par rapport à la fonctionnelle c c'est à dire la famille pour laquelle $\forall k$ on a

- P_k a exactement le degré k
- $c\left(x^i P_k\right) = 0 \qquad$ pour $\; i = 0, \ldots, k-1.$

Le polynôme P_k existe et est unique (à une constante multiplicative près) si et seulement si le déterminant de Hankel

$$H_k^{(0)} = \begin{vmatrix} c_0 & \cdots & c_{k-1} \\ \vdots & & \vdots \\ c_{k-1} & \cdots & c_{2k-2} \end{vmatrix} \neq 0.$$

Si cette condition est vérifiée pour tout k, alors la fonctionnelle c est dite *definie.*
Posons

$$P_k(x) = a_0 + \cdots + a_k x^k.$$

Si l'on applique les conditions d'orthogonalité on obtient

$$\begin{aligned} c(P_k(x)) &= a_0c_0 + a_1c_1 + \cdots + a_kc_k \\ c(xP_k(x)) &= a_0c_1 + a_1c_2 + \cdots + a_kc_{k+1} \\ &\cdots\cdots\cdots\cdots \\ c\left(x^{k-1}P_k(x)\right) &= a_0c_{k-1} + a_1c_k + \cdots + a_kc_{2k-1} \end{aligned}$$

c'est à dire le système suivant

$$\begin{cases} a_0c_0 + a_1c_1 + \cdots + a_kc_k &= 0 \\ a_0c_1 + a_1c_2 + \cdots + a_kc_{k+1} &= 0 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ a_0c_{k-1} + a_1c_k + \cdots + a_kc_{2k-1} &= 0. \end{cases}$$

Si l'on impose que P_k soit unitaire (ce qui est possible) les k coefficients a_i sont solution du système

$$\left\{\begin{array}{lcll} a_0c_0 & +\cdots+ & a_{k-1}c_{k-1} & = & -c_k \\ a_0c_1 & +\cdots+ & a_{k-1}c_k & = & -c_{k+1} \\ \vdots & & \vdots & & \vdots \\ a_0c_{k-1} & +\cdots+ & a_{k-1}c_{2k-2} & = & -c_{2k-1} \end{array}\right. \tag{2.1}$$

et P_k peut s'écrire comme le rapport de determinants suivant

$$P_k(x) = \left|\begin{array}{ccc} c_0 & \cdots & c_k \\ \vdots & & \vdots \\ c_{k-1} & \cdots & c_{2k-1} \\ 1 & \cdots & x^k \end{array}\right| \Big/ H_k^{(0)}.$$

Une exposition complète de la théorie des polynômes orthogonaux formels peut se trouver dans [2]. L'on sait, par exemple, que si c est defini alors les polynômes orthogonaux satisfont à la relation à trois termes suivante

$$\begin{array}{rcl} P_{-1}(x) & = & 0 \\ P_0(x) & = & \text{constante arbitraire non nulle} \\ P_{k+1}(x) & = & (A_{k+1}x + B_{k+1})P_k(x) - C_{k+1}P_{k-1}(x), \qquad k = 0, 1, \ldots \end{array} \tag{2.2}$$

où

$$A_{k+1} = \frac{h_{k+1}}{\beta_k}, \quad B_{k+1} = -\frac{h_{k+1}\alpha_k}{h_k\beta_k}, \quad C_{k+1} = \frac{\beta_{k-1}h_{k+1}}{\beta_k h_{k-1}}$$

avec

$$\alpha_k = c\left(xP_k^2\right), \quad \beta_k = c(xP_kP_{k+1}), \quad h_k = c\left(P_k^2\right).$$

Cette définition des coefficients A_{k+1}, B_{k+1} et C_{k+1} n'est pas utile dans la pratique car pour calculer P_{k+1} il faut connaître P_{k+1} lui même. Mais si l'on écrit différement P_k comme

$$P_k(x) = \sum_{i=0}^{k} p_i^{(k)}x^i = p_0^{(k)} + p_1^{(k)}x + \cdots + p_k^{(k)}x^k$$

alors l'on peut écrire A_{k+1}, B_{k+1} et C_{k+1} uniquement en fonction des polynômes P_{k-1}, P_k et au coefficient de la plus grande puissance de x c'est à dire que

$$A_{k+1} = \frac{p_{k+1}^{(k+1)}}{p_k^{(k)}}, \quad B_{k+1} = -\frac{p_{k+1}^{(k+1)}\alpha_k}{h_k p_k^{(k)}}, \quad C_{k+1} = \frac{p_{k-1}^{(k-1)}p_{k+1}^{(k+1)}}{\left(p_k^{(k)}\right)^2} \cdot \frac{h_k}{h_{k-1}}$$

avec

$$\alpha_k = p_k^{(k)}c\left(x^{k+1}P_k\right) + p_{k-1}^{(k)}c\left(x^kP_k\right).$$

Donc si l'on choisit les P_k unitaires, on aura $\forall k, A_{k+1} = 1$ c'est à dire que

$$\begin{aligned} P_{-1}(x) &= 0 \\ P_0(x) &= 1 \\ P_{k+1}(x) &= (x + B_{k+1})P_k(x) - C_{k+1}P_{k-1}(x), \qquad k = 0, 1, \ldots \end{aligned} \tag{2.3}$$

avec

$$\begin{aligned} B_{k+1} &= -\frac{\alpha_k}{h_k}, \quad C_{k+1} = \frac{h_k}{h_{k-1}} \\ h_k &= c\left(x^k P_k\right), \quad \alpha_k = c\left(x^{k+1} P_k\right) + p_{k-1}^{(k)} c\left(x^k P_k\right). \end{aligned} \tag{2.4}$$

2.1 Présentation nouvelle des polynômes orthogonaux

Dans [8] nous avons proposé une nouvelle présentation des polynômes orthogonaux basée sur l'introduction de deux familles auxiliaires de polynômes unitaires. De cette façon nous avons obtenu de nouveau les formules bien connues qui donnent les polynômes comme rapport de déterminants, leur relation à trois termes pour le cas defini et même celle du cas indefini trouvée par Draux [17]. L'on peut aussi très facilement obtenir de nouveau le *qd-algorithme* de Rutishauser [34].

Avec cette nouvelle formulation on peut construire de nouvelles relations qu'on a comparées, en proposant des exemples numériques, avec les relations habituelles.

Nous allons maintenant décrire cette présentation. Si l'on considère une autre famille de polynômes unitaires $\{\Pi_i\}$ tel que $\forall k$, $\Pi_0, \ldots, \Pi_k$ soient une base pour l'espace vectoriel P_k des polynômes de degré au plus k et si l'on écrive P_k dans cette base comme

$$P_k(x) = a_0\Pi_0 + \cdots + a_k\Pi_k, \tag{2.5}$$

avec $a_k = 1$ et les autres coefficients a_i qui dependent de k, on peut écrire les relations d'orthogonalité

$$c(\Pi_i P_k) = 0 \qquad \text{pour } i = 0, 1, \ldots, k-1 \tag{2.6}$$

et donc obtenir le système

$$\begin{cases} a_0\, c(\Pi_0\Pi_0) & + \cdots + & a_{k-1}\, c(\Pi_0\Pi_{k-1}) & = & -c(\Pi_0\Pi_k) \\ \vdots & & \vdots & & \vdots \\ a_0\, c(\Pi_{k-1}\Pi_0) & + \cdots + & a_{k-1}\, c(\Pi_{k-1}\Pi_{k-1}) & = & -c(\Pi_{k-1}\Pi_k). \end{cases}$$

$P_k(x)$ est donc donné par le rapport de déterminants suivant

$$P_k(x) = \frac{\begin{vmatrix} c(\Pi_0\Pi_0) & \cdots & c(\Pi_0\Pi_k) \\ \vdots & & \vdots \\ c(\Pi_{k-1}\Pi_0) & \cdots & c(\Pi_{k-1}\Pi_k) \\ \Pi_0 & \cdots & \Pi_k \end{vmatrix}}{\begin{vmatrix} c(\Pi_0\Pi_0) & \cdots & c(\Pi_0\Pi_{k-1}) \\ \vdots & & \vdots \\ c(\Pi_{k-1}\Pi_0) & \cdots & c(\Pi_{k-1}\Pi_{k-1}) \end{vmatrix}}.$$

Cette approche a été suivie par Szegö [36] (p. 23). Mais nous allons maintenant la généraliser en introduisant une deuxième famille de polynômes unitaires $\{R_i\}$ telle que $\forall k$, $R_0, \ldots, R_k$ soient encore une base pour P_k. De cette façon, dans [8], on a pu obtenir plusieurs formules différentes.

Nous prenons P_k dans la forme (2.5) mais nous allons écrire les conditions d'orthogonalité en utilisant R_k (au lieu de (2.6)) c'est à dire

$$c(R_i P_k) = 0 \qquad \text{pour } i = 0, 1, \ldots, k-1 \tag{2.7}$$

ce qui nous donne le système

$$\left\{ \begin{array}{ccccc} a_0\, c(R_0\Pi_0) & +\cdots+ & a_k\, c(R_0\Pi_k) & = & 0 \\ \vdots & & \vdots & & \vdots \\ a_0\, c(R_{k-1}\Pi_0) & +\cdots+ & a_k\, c(R_{k-1}\Pi_k) & = & 0. \end{array} \right. \tag{2.8}$$

Puisque $a_k = 1$, si l'on ajoute l'équation

$$-P_k + a_0\Pi_0 + \cdots + a_{k-1}\Pi_{k-1} = -\Pi_k$$

on aura que

$$P_k(x) = \frac{\begin{vmatrix} c(R_0\Pi_0) & \cdots & c(R_0\Pi_k) \\ \vdots & & \vdots \\ c(R_{k-1}\Pi_0) & \cdots & c(R_{k-1}\Pi_k) \\ \Pi_0 & \cdots & \Pi_k \end{vmatrix}}{\begin{vmatrix} c(R_0\Pi_0) & \cdots & c(R_0\Pi_{k-1}) \\ \vdots & & \vdots \\ c(R_{k-1}\Pi_0) & \cdots & c(R_{k-1}\Pi_{k-1}) \end{vmatrix}}.$$

Naturellement les polynômes $\Pi_0, \ldots, \Pi_k$ et $R_0, \ldots, R_{k-1}$ peuvent dépendre de la valeur de k.

On peut considérer trois choix particuliers pour les polynômes auxiliaires

1. $\Pi_i = R_i$ déjà considéré par Szegö

2. Π_i et/ou R_i famille de polynômes orthogonaux

3. $R_i = P_i$.

Dans le deuxième cas, si $\{\Pi_i\}$ est une famille de polynômes orthogonaux, alors les quantités $c(\Pi_i P_j)$ sont les *moments modifiés* qui peuvent se calculer récursivement avec l'une quelconque des méthodes presentées dans [26].

Dans le troisième cas (pour l'orthogonalité des P_i) le rapport de déterminants se simplifie dans

$$P_k(x) = \frac{\begin{vmatrix} c(P_0\Pi_0) & \cdots & \cdots & c(P_0\Pi_{k-1}) & c(P_0\Pi_k) \\ & \ddots & & \vdots & \vdots \\ & 0 & \ddots & \vdots & \vdots \\ & & & c(P_{k-1}\Pi_{k-1}) & c(P_{k-1}\Pi_k) \\ \Pi_0 & \cdots & \cdots & \Pi_{k-1} & \Pi_k \end{vmatrix}}{\begin{vmatrix} c(P_0\Pi_0) & \cdots & \cdots & c(P_0\Pi_{k-1}) \\ & \ddots & & \vdots \\ & 0 & \ddots & \vdots \\ & & & c(P_{k-1}\Pi_{k-1}) \end{vmatrix}}$$

avec le déterminant au dénominateur toujours différent de zéro (si c est definie alors $\forall i, c(x^i P_i) \neq 0$ et donc $c(P_i\Pi_i) \neq 0$). En effet dans ce cas le système devient triangulaire

$$\begin{cases} a_0\, c(P_0\Pi_0) + \cdots + a_{k-1}\, c(P_0\Pi_{k-1}) = -c(P_0\Pi_k) \\ \qquad\qquad \ddots \qquad\qquad \vdots \qquad\qquad \vdots \\ \qquad\qquad\qquad a_{k-1}\, c(P_{k-1}\Pi_{k-1}) = -c(P_{k-1}\Pi_k). \end{cases}$$

Les polynômes orthogonaux sont à la base de plusieurs applications en analyse numérique. Cependant, dans certains cas, la normalisation $a_k = 1$ n'est plus adaptée et l'on choisit une normalisation différente. Par exemple dans les méthodes de type Lanczos pour résoudre les systèmes linéaires [7, 9, 10, 11, 12, 13, 14] l'on prend $P_k(0) = 1$. Dans l'ε-algorithme topologique [4] (p. 189) qui est très voisin de la méthode du gradient biconjugué de Fletcher [18], l'on prend $P_k(1) = 1$.

Ces deux cas peuvent être résumés par $P_k(a) = 1$ avec $a = 0$ ou $a = 1$ et donc le système devient

$$\begin{cases} a_0\, \Pi_0(a) + \cdots + a_k\, \Pi_k(a) = 1 \\ a_0\, c(R_0\Pi_0) + \cdots + a_k\, c(R_0\Pi_k) = 0 \\ \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ a_0\, c(R_{k-1}\Pi_0) + \cdots + a_k\, c(R_{k-1}\Pi_k) = 0 \end{cases}$$

et l'on aura

$$P_k(x) = \frac{\begin{vmatrix} c(R_0\Pi_0) & \cdots & c(R_0\Pi_k) \\ \vdots & & \vdots \\ c(R_{k-1}\Pi_0) & \cdots & c(R_{k-1}\Pi_k) \\ \Pi_0 & \cdots & \Pi_k \end{vmatrix}}{\begin{vmatrix} c(R_0\Pi_0) & \cdots & c(R_0\Pi_k) \\ \vdots & & \vdots \\ c(R_{k-1}\Pi_0) & \cdots & c(R_{k-1}\Pi_k) \\ \Pi_0(a) & \cdots & \Pi_k(a) \end{vmatrix}}.$$

Le calcul des coefficients des polynômes P_k peut se faire, par exemple, à l'aide de la méthode de *bordage* de Faddeeva [19] où de la méthode de *bordage par blocs* [10] qui

permet de sauter directement du polynôme P_k au polynôme P_{k+i} quand les systèmes intermédiaires sont singuliers (et donc quand les polynômes $P_{k+1}, \ldots, P_{k+i-1}$ n'existent pas). Ces méthodes, que nous décrirons dans le chapitre 3, ne peuvent pas être utilisées par contre avec la normalisation $a_k = 1$ pour calculer les solutions mais seulement pour calculer récursivement les matrices inverses des systèmes.

Quand $P_k(a) = 1$ et que l'on choisit $R_i = P_i$ le système, qui était triangulaire dans le cas $a_k = 1$, devient un système avec une matrice de Hessenberg supérieure.

2.1.1 Relations de récurrence

Dans [8] on a retrouvé de nouveau, à l'aide de cette approche, les relations de récurrence dans les cas où la fonctionnelle c est définie ou non.

Dans le premier cas, si l'on choisit $\Pi_i = P_i$ pour $i = 0, \ldots, k-1$ et $\Pi_k = xP_{k-1}$ c'est à dire

$$P_k = a_0 P_0 + \cdots + a_{k-1} P_{k-1} + xP_{k-1}$$

et la normalisation $a_k = 1$, le système (2.8) devient

$$\begin{aligned}
&a_0\, c(R_0 P_0) = 0\\
&a_0\, c(R_1 P_0) + a_1\, c(R_1 P_1) = 0\\
&\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\\
&a_0\, c(R_{k-2} P_0) + \cdots + a_{k-2}\, c(R_{k-2} P_{k-2}) = -c(xR_{k-2} P_{k-1})\\
&a_0\, c(R_{k-1} P_0) + \cdots + a_{k-2}\, c(R_{k-1} P_{k-2}) + a_{k-1}\, c(R_{k-1} P_{k-1}) = -c(xR_{k-1} P_{k-1}).
\end{aligned}$$

Tous les coefficients a_i, pour $i = 0, \ldots, k-3$, sont égaux à zéro, tandis que les deux dernières équations donnent a_{k-2} et a_{k-1}. On a donc obtenu la relation de récurrence à trois termes

$$P_k(x) \;=\; (x + a_{k-1}) P_{k-1}(x) + a_{k-2} P_{k-2}(x) \tag{2.9}$$

avec

$$a_{k-2} = -\frac{c(xR_{k-2} P_{k-1})}{c(R_{k-2} P_{k-2})} \quad \text{et} \quad a_{k-1} - \frac{c(xR_{k-1} P_{k-1}) + a_{k-2}\, c(R_{k-1} P_{k-2})}{c(R_{k-1} P_{k-1})}.$$

Si l'on prend $R_i = P_i$ pour $i = 0, \ldots, k-1$ on retrouve les formules habituelles

$$a_{k-2} = -\frac{c(xP_{k-2} P_{k-1})}{c\left(P_{k-2}^2\right)} \quad \text{et} \quad a_{k-1} = -\frac{c\left(xP_{k-1}^2\right)}{c\left(P_{k-1}^2\right)}$$

car $c(xP_{k-2} P_{k-1}) = c\left(P_{k-1}^2\right)$ et le système devient diagonal. Comme R_i est unitaire, alors on a en plus $c(R_i P_i) = c(x^i P_i)$.

Maintenant on va considérer le cas de familles adjacentes de polynômes orthogonaux. Soit $\left\{P_k^{(1)}\right\}$ la famille de polynômes orthogonaux unitaires par rapport à la fonctionnelle $c^{(1)}$ définie par

$$c^{(1)}\left(x^i\right) = c\left(x^{i+1}\right) = c_{i+1}.$$

Soit $c^{(1)}$ définie ($\forall k, P_k^{(1)}$ existe). Prenons $\Pi_i = P_i$ pour $i = 0, \ldots, k-1$ and $\Pi_k = xP_{k-1}^{(1)}$. Les conditions (2.7) deviennent

$$c(R_i P_k) = a_0\, c(R_i P_0) + \cdots + a_{k-1} c(R_i P_{k-1}) + c^{(1)}\left(R_i P_{k-1}^{(1)}\right) = 0$$

pour $i = 0, \ldots, k-1$ et par l'orthogonalité de P_{k-1} et $P_{k-1}^{(1)}$ on aura

$$P_k(x) = a_{k-1}\, P_{k-1}(x) + xP_{k-1}^{(1)}(x)$$

avec $a_{k-1} = -c^{(1)}\left(R_{k-1} P_{k-1}^{(1)}\right) / c(R_{k-1} P_{k-1})$.

Si l'on écrit $P_k^{(1)}$ comme

$$P_k^{(1)} = a_0 \Pi_0 + \cdots + a_k \Pi_k$$

et que l'on choisit $\Pi_i = P_i^{(1)}$ (pour $i = 0, \ldots, k-1$) et $\Pi_k = P_k$ on aura

$$c^{(1)}\left(R_i P_k^{(1)}\right) = a_0\, c^{(1)}\left(R_i P_0^{(1)}\right) + \cdots + a_{k-1} c^{(1)}\left(R_i P_{k-1}^{(1)}\right) + c(xR_i P_k) = 0$$

pour $i = 0, \ldots, k-1$. Par l'orthogonalité de $P_{k-1}^{(1)}$ et de P_k on aura $a_0 = \cdots = a_{k-2} = 0$ et donc

$$P_k^{(1)}(x) = a_{k-1}\, P_{k-1}^{(1)}(x) + P_k(x)$$

avec $a_{k-1} = -c(xR_{k-1} P_k) / c^{(1)}\left(R_{k-1} P_{k-1}^{(1)}\right)$.

Ces deux relations sont une généralisation de celles habituelles entre familles adjacentes de polynômes orthogonaux.

Dans [8] nous avons aussi montré que si l'on considère la fonctionnelle linéaire $c^{(n)}$ définie par

$$c^{(n)}\left(x^i\right) = c_{n+i},$$

on peut très facilement retrouver le qd-algorithme de Rutishauser [34].

Maintenant nous allons considérer le cas dans lequel c est non defini. Soit $1 = n_0 < n_1 < n_2 < \cdots$ la suite d'indices pour laquelle on a $H_{n_k}^{(0)} \neq 0$, tandis que tous les autres déterminants d'Hankel sont nuls. Donc les polynômes qui existent (nommés *réguliers*) sont les polynômes de degré n_k pour $k = 0, 1, \ldots$ que l'on dénote P_k. Draux [17] a prouvé que le polynôme P_{k+1} de degré $n_{k+1} = n_k + m_k$ peut se déterminer par les conditions suivantes

$$c\left(x^i P_k\right) = 0 \qquad \text{pour } i = 0, \ldots, n_k + m_k - 2$$
$$\text{et } c\left(x^{n_k+m_k-1} P_k\right) \neq 0.$$

Si l'on écrit P_{k+1} comme

$$P_{k+1} = a_0 \Pi_0 + \cdots + a_{n_{k+1}-1} \Pi_{n_{k+1}-1} + \Pi_{n_{k+1}}$$

avec $\Pi_{n_i} = P_i$, $\Pi_{n_i+j} = x^j P_{n_i}$ pour $j = 0, \ldots, m_i - 1$, $i = 0, \ldots, k$ et $\Pi_{n_{k+1}} = x^{m_k} P_{n_k}$, $\Pi_0 = P_0 = 1$ alors ces polynômes forment une base et les coefficients a_i sont les solutions du système

$$(A_0, \ldots, A_i, \ldots, A_k) \begin{pmatrix} a_0 \\ \vdots \\ a_{n_{k+1}-1} \end{pmatrix} = - \begin{pmatrix} c(x\Pi_{n_k+m_k-1}) \\ \vdots \\ c(x^{n_k+m_k}\Pi_{n_k+m_k-1}) \end{pmatrix}.$$

Dans le second membre on aura

$$c\left(x^i \Pi_{n_k+m_k-1}\right) = 0 \qquad \text{pour } i = 0, \ldots, n_k - 1.$$

Les matrices rectangulaires A_i ont (pour $i = 1, \ldots, k$) la forme

$$\begin{pmatrix} \boxed{\begin{matrix} c(x^0\Pi_{n_i}) \\ \vdots \\ \vdots \\ c(x^{n_i+m_i-2}\Pi_{n_i}) \end{matrix}} & \boxed{\begin{matrix} c(x^0\Pi_{n_i+1}) \\ \vdots \\ c(x^{n_i+m_i-3}\Pi_{n_i+1}) \end{matrix}} & \cdots & \cdots & \boxed{\begin{matrix} c(x^0\Pi_{n_i+m_i-1}) \\ \vdots \\ c(x^{n_i-1}\Pi_{n_i+m_i-1}) \end{matrix}} \\ c(x^{n_i+m_i-1}\Pi_{n_i}) & c(x^{n_i+m_i-2}\Pi_{n_i+1}) & & & c(x^{n_i}\Pi_{n_i+m_i-1}) \\ \vdots & \vdots & & & \vdots \\ c(x^{n_k+m_k-1}\Pi_{n_i}) & c(x^{n_k+m_k-1}\Pi_{n_i+1}) & \cdots & \cdots & c(x^{n_k+m_k-1}\Pi_{n_i+m_i-1}) \end{pmatrix}$$

et la matrice A_0

$$\begin{pmatrix} \boxed{\begin{matrix} c(x^0\Pi_0) \\ \vdots \\ c(x^{m_0-2}\Pi_0) \end{matrix}} & \cdots & \cdots & \boxed{c(x^0\Pi_{m_0-2})} & c(x^0\Pi_{m_0-1}) \\ c(x^{m_0-1}\Pi_0) & & & c(x\Pi_{m_0-2}) & c(x\Pi_{m_0-1}) \\ \vdots & & & \vdots & \vdots \\ c(x^{n_k+m_k-1}\Pi_0) & \cdots & \cdots & c(x^{n_k+m_k-1}\Pi_{m_0-2}) & c(x^{n_k+m_k-1}\Pi_{m_0-1}) \end{pmatrix}.$$

Toutes les quantités encadrées sont nulles et donc les seuls coefficients non nuls sont a_{n_k-1} et les a_i, pour $i = n_k, \ldots, n_k + m_k - 1$. On a donc retrouvé la relation de Draux [17] avec une démontration bien plus simple:

$$P_{k+1}(x) = \left(\alpha_0 + \cdots + \alpha_{m_k-1}x^{m_k-1} + x^{m_k}\right) P_k(x) - C_{k+1}P_{k-1}(x), \qquad k = 0, 1, \ldots$$

avec $P_{-1}(x) = 0$, $P_0(x) = 1$, $C_1 = 0$ et

$$\begin{aligned} C_{k+1} &= c\left(x^{n_k+m_k-1}P_k\right) \Big/ c\left(x^{n_k-1}P_{k-1}\right) \\ \alpha_{m_k-1}c\left(x^{n_k+m_k-1}P_k\right) &+ c\left(x^{n_k+m_k}P_k\right) = C_{k+1}c(x^{n_k}P_{k-1}) \\ &\vdots \\ \alpha_0 c\left(x^{n_k+m_k-1}P_k\right) + \cdots + \alpha_{m_k-1}c\left(x^{n_k+2m_k-2}P_k\right) &+ c\left(x^{n_k+2m_k-1}P_k\right) = \\ & C_{k+1}c\left(x^{n_k+m_k-1}P_{k-1}\right). \end{aligned}$$

D'autres démontrations se trouvent dans [10, 28]. Avec la même approche on peut trouver, pour le cas d'une fonctionnelle c non definie, des relations pour les familles adjacentes de polynômes orthogonaux.

2.1.2 Création d'autres formules

Avec cette façon de procéder on peut aussi très facilement construire de nouvelles relations entre polynômes orthogonaux qui permettent de calculer récursivement un polynôme en utilisant un certain nombre de polynômes de la même famille ou de familles adjacentes. Pour expliquer comment procéder on peut donner deux exemples.

Exemple 1

On cherche à trouver une formule récursive de la forme

$$P_{k+i}(x) = a_0 P_0(x) + \cdots + a_k P_k(x) + a_{k+1}\, x\, P_k(x) + \cdots + a_{k+i}\, x^i\, P_k(x)$$

avec $a_{k+i} = 1$.

Si l'on multiplie par R_j et que l'on applique c on aura

$$c(R_j P_{k+i}) = a_0\, c(R_j P_0) + \cdots + a_k c(R_j P_k) + a_{k+1} c(x\, R_j P_k) + \cdots + a_{k+i} c\big(x^i\, R_j P_k\big) = 0$$

pour $j = 0, \ldots, k+i-1$ et l'orthogonalité de P_i par rapport aux polynômes de degré au plus $i-1$ permet de déduire que tous les coefficients a_i, pour $i = 0, \ldots, k-i-1$, sont nuls et donc

$$P_{k+i}(x) = a_{k-i} P_{k-i}(x) + \cdots + a_k P_k(x) + a_{k+1}\, x\, P_k(x) + \cdots + a_{k+i}\, x^i\, P_k(x)$$

avec $a_{k+i} = 1$ et

$$a_{k-i}\, c(R_j P_{k-i}) + \cdots + a_k c(R_j P_k) + a_{k+1} c(x\, R_j P_k) + \cdots + a_{k+i-1} c\big(x^{i-1}\, R_j P_k\big) = -c\big(x^i\, R_j P_k\big)$$

pour $j = k-i, \ldots, k+i-1$.

Le polynôme P_{k+i} peut aussi s'écrire comme le rapport de déterminants suivant

$$P_{k+i}(x) = \frac{\begin{vmatrix} c(R_{k-i}P_{k-i}) & \cdots & c(R_{k-i}P_k) & c(x\,R_{k-i}P_k) & \cdots & c(x^i R_{k-i}P_k) \\ \vdots & & \vdots & \vdots & & \vdots \\ c(R_{k+i-1}P_{k-i}) & \cdots & c(R_{k+i-1}P_k) & c(x\,R_{k+i-1}P_k) & \cdots & c(x^i R_{k+i-1}P_k) \\ P_{k-i}(x) & \cdots & P_k(x) & x\,P_k(x) & \cdots & x^i P_k(x) \end{vmatrix}}{\begin{vmatrix} c(R_{k-i}P_{k-i}) & \cdots & c(R_{k-i}P_k) & c(x\,R_{k-i}P_k) & \cdots & c(x^{i-1} R_{k-i}P_k) \\ \vdots & & \vdots & \vdots & & \vdots \\ c(R_{k+i-1}P_{k-i}) & \cdots & c(R_{k+i-1}P_k) & c(x\,R_{k+i-1}P_k) & \cdots & c(x^{i-1} R_{k+i-1}P_k) \end{vmatrix}}.$$

Exemple 2

On cherche à trouver une formule récursive de la forme

$$P_{k+i}^{(n)}(x) = a_0 P_0^{(n)}(x) + \cdots + a_k P_k^{(n)}(x) + a_{k+1}\, x\, P_k^{(n+1)}(x) + \cdots + a_{k+i}\, x^i\, P_k^{(n+i)}(x)$$

avec $a_{k+i} = 1$.

On multiplie pour R_j et l'on applique $c^{(n)}$. D'après l'orthogonalité on a $c^{(n)}\left(x^i\, R_j P_k^{(n+i)}\right) = c^{(n+i)}\left(R_j P_k^{(n+i)}\right) = 0$ pour $j = 0, \ldots, k-1$, et donc les coefficients $a_0, \ldots, a_{k-1}$ sont nuls et on obtient

$$P_{k+i}^{(n)}(x) = a_k P_k^{(n)}(x) + a_{k+1}\, x\, P_k^{(n+1)}(x) + \cdots + a_{k+i}\, x^i\, P_k^{(n+i)}(x)$$

avec $a_{k+i} = 1$ et

$$a_k\, c^{(n)}\left(R_{k+j} P_k^{(n)}\right) + \cdots + a_{k+i-1} c^{(n+i-1)}\left(R_{k+j} P_k^{(n+i-1)}\right) = -c^{(n+i)}\left(R_{k+j} P_k^{(n+i)}\right)$$

pour $j = 0, \ldots, i-1$.

Le rapport de déterminants qui donne $P_{k+i}^{(n)}(x)$ est

$$P_{k+i}^{(n)}(x) = \frac{\begin{vmatrix} c^{(n)}\left(R_k P_k^{(n)}\right) & \cdots & c^{(n+i)}\left(R_k P_k^{(n+i)}\right) \\ \vdots & & \vdots \\ c^{(n)}\left(R_{k+i-1} P_k^{(n)}\right) & \cdots & c^{(n+i)}\left(R_{k+i-1} P_k^{(n+i)}\right) \\ P_k^{(n)}(x) & \cdots & x^i P_k^{(n+i)}(x) \end{vmatrix}}{\begin{vmatrix} c^{(n)}\left(R_k P_k^{(n)}\right) & \cdots & c^{(n+i-1)}\left(R_k P_k^{(n+i-1)}\right) \\ \vdots & & \vdots \\ c^{(n)}\left(R_{k+i-1} P_k^{(n)}\right) & \cdots & c^{(n+i-1)}\left(R_{k+i-1} P_k^{(n+i-1)}\right) \end{vmatrix}}.$$

2.1.3 Applications

On prend les moments suivants

$$c_i = \int_{-1}^{1} x^i \, \mathrm{d}x$$

et donc, pour $i = 0, 1, \ldots$, on a $c_{2i} = 2/(2i+1)$ et $c_{2i+1} = 0$. Les polynômes orthogonaux correspondants sont les polynômes de Legendre dans l'intervalle $[-1, +1]$. Si l'on impose qu'ils soient unitaires on aura la relation de récurrence suivante

$$P_k(x) = x\, P_{k-1}(x) - \frac{(k-1)^2}{4(k-1)^2 - 1} \cdot P_{k-2}(x), \qquad k = 0, 1, \ldots$$

avec $P_{-1}(x) = 0$ et $P_0(x) = 1$.

Nous avons utilisé la formule (2.9) pour différents choix des polynômes auxiliaires $\{R_k\}$:

1. $R_k(x) = x^k$
2. $R_k(x) = P_k(x)$
3. $R_k(x) = 1 + x + \cdots + x^k$
4. $R_k(x) = a^k + a^{k-1}x + \cdots + a\,x^{k-1} + x^k$ avec $a = 3$
5. même R_k avec $a = 1/3$
6. même R_k avec $a = 3/2$
7. même R_k avec $a = 2/3$.

Nous avons fait les calculs avec environ 16 chiffres décimaux et on a obtenu toujours exactement zéro, pour les coefficients a_{k-1} de (2.9). Dans le tableau suivant on donne le nombre de chiffres exactes obtenu pour le coefficient a_{k-2} de (2.9) (sa valeur exacte étant $-(k-1)^2/(4(k-1)^2-1)$).

k	1	2	3	4	5	6	7
2	16.0	16.0	16.0	16.0	16.0	16.0	16.0
3	15.7	15.7	15.7	14.8	15.7	16.0	15.7
4	14.8	15.0	14.8	13.1	14.8	14.2	14.7
5	14.3	14.6	14.6	11.6	14.1	13.3	14.0
6	14.0	14.6	13.4	11.0	14.0	13.0	13.4
7	14.0	12.9	12.8	8.5	14.0	11.2	12.9
8	13.7	12.8	12.9	7.4	12.8	10.6	12.8
9	11.8	11.7	13.7	5.4	11.6	9.6	12.0
10	11.0	12.2	10.5	4.5	11.1	8.7	11.1
11	10.3	10.1	9.5	3.4	10.9	7.8	10.7
12	9.7	9.7	9.0	2.6	10.0	6.8	9.8
13	9.3	9.7	8.9	0.0	9.0	5.7	8.7
14	9.2	8.4	7.8	0.0	8.2	4.5	7.8
15	8.4	7.5	6.8	0.0	7.5	3.8	7.1
16	7.2	6.4	5.9	0.0	6.8	3.2	6.3
17	6.2	5.9	5.0	0.0	6.1	1.7	5.6
18	5.4	4.8	4.3	0.0	5.3	0.6	4.9
19	4.7	4.1	3.6	0.0	4.5	0.1	4.2
20	4.0	3.5	2.9	0.0	3.6	0.0	3.6
21	3.3	2.8	2.1	0.0	2.7	0.0	2.8
22	2.5	2.0	1.3	0.0	1.9	0.0	2.1
23	1.6	0.5	0.6	0.0	1.1	0.0	1.3
24	0.7	0.0	0.0	0.0	0.3	0.0	0.5
25	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Les résultats montrent bien qu'ils sont très sensibles au choix de $\{R_k\}$. L'instabilité dans certains choix est due probablement au produit de deux polynômes fait en calculant $c(PQ)$.

Si les *moments modifiés* $Z_{k,n} = c(P_k x^n)$, ou $c(R_k x^n)$, ou $c(P_k R_n)$ sont connus dans une forme analytique exacte, ce produit n'intervient plus et donc la formule (2.9) devient plus stable.

2.2 Relation de récurrence, moments et moments modifiés

Comme l'on a déjà vu dans le début de ce chapitre, le calcul d'un polynôme $P_k \in \mathsf{P}_k$, qui satisfait aux conditions d'orthogonalité

$$c\left(x^i P_k\right) = 0, \qquad i = 0, \ldots, k-1$$

où c est une fonctionnelle linéaire sur l'espace P des polynômes réels (voir [4]) déterminée par ses moments

$$c(x^i) = c_i \qquad i = 0, 1, \ldots$$

peut se faire en imposant une normalisation pour le polynôme (par exemple qu'il soit unitaire) et en déterminant ses k coefficients en résolvant le système classique (2.1).

Cependant, comme ce système est souvent mal conditionné, l'on préfère utiliser la relation de récurrence à trois termes (2.2) ou (2.3) (si l'on désire les polynômes unitaires) qui existe pour tous les polynômes orthogonaux.

Dans la suite nous cherchons toujours les polynômes unitaires et donc le problème de la détermination du polynôme P_k se réduit à la détermination des $2k$ coefficients B_i, C_i, pour $i = 1, \ldots, k$.

C'est évident que pour bien calculer le polynôme P_k, il faut calculer ces coefficients de la façon la plus stable possible. Il existe beaucoup de méthodes qui permettent le calcul des coefficients de la relation de récurrence et qui utilisent les moments ou les moments modifiés. Dans toutes ces méthodes la partie la plus délicate concerne la détermination des moments c_i et des moments modifiés (qu'on va noter v_i).

Dans [32] nous avons proposé une application des polynômes orthogonaux à un problème de quadrature en proposant un algorithme complet pour sa résolution. Après, dans [33], en reprenant la même application, nous avons présenté une comparaison entre différentes méthodes basées sur les moments et sur les moments modifiés en utilisant, dans cette application particulière, soit le calcul numérique, soit le calcul formel avec *Mathematica* pour essayer de voir si l'utilisation des moments modifiés peut apporter une véritable amélioration de la stabilité du calcul. Ce qu'on a obtenu démontre que l'avantage peut être, dans certains cas, faible.

Dans la suite nous allons d'abord reprendre un certains nombre de méthodes que nous avons utilisées dans nos applications.

2.2.1 Méthodes basées sur les moments

Méthode de Chebyshev

La première méthode que nous allons considérer est la *méthode de Chebyshev* [15]. Cette méthode est souvent dans la pratique mal conditionnée.

On considère la matrice Z dont les éléments sont définis de la façon suivante

$$z_{k,i} = c\left(P_k x^i\right), \qquad k, i = 0, 1, 2, \ldots$$

où les $z_{k,i}$, pour $i < k$, sont toujours nuls d'après les conditions d'orthogonalité. La première ligne de cette matrice contient les moments car $z_{0,i} = c(P_0 x^i) = c(x^i) = c_i$.

Si l'on multiplie (4.1) par x^i et que l'on applique la fonctionnelle c on obtient la méthode suivante de Chebyshev

$$\begin{aligned} z_{k,i} &= z_{k-1,i+1} + B_k z_{k-1,i} - C_k z_{k-2,i} \quad i = k, k+1, \ldots \\ B_{k+1} &= \frac{z_{k-1,k}}{z_{k-1,k-1}} - \frac{z_{k,k+1}}{z_{k,k}} \\ C_{k+1} &= \frac{z_{k,k}}{z_{k-1,k-1}}. \end{aligned} \tag{2.10}$$

pour $k = 1, 2, \ldots$ et avec les conditions initiales

$$\begin{aligned} z_{-1,i} &= 0, && i = 1, 2, \ldots \\ z_{0,i} &= c_i, && i = 0, 1, 2, \ldots \\ B_1 &= -\frac{c_1}{c_0} \\ C_1 &= c_0. \end{aligned}$$

Méthode de bordage

La deuxième méthode, que nous avons utilisée aussi dans [32], donne directement soit les coefficients de la relation de récurrence, soit les coefficients du polynôme orthogonal P_{k+1}. Donc on n'a pas à effectuer de multiplications additionelles comme dans (2.3).

Si l'on substitue dans (2.3) les valeurs données par (2.4) et si l'on considère les coefficients des puissance de x, alors les coefficients $p_i^{(k)}$ peuvent se calculer de la façon suivante (comme il a été montré dans [4])

$$\begin{aligned} p_{-1}^{(-1)} &= p_0^{(-1)} = 0\,, \quad h_{-1} = 1 \\ p_{-1}^{(0)} &= p_1^{(0)} = 0\,, \quad p_0^{(0)} = 1 \\ p_{-1}^{(k+1)} &= p_{k+2}^{(k+1)} = 0\,; \quad p_{k+1}^{(k+1)} = 1 \\ p_i^{(k+1)} &= p_{i-1}^{(k)} + B_{k+1} p_i^{(k)} - C_{k+1} p_i^{(k-1)}\,, \quad i = 0, \ldots, k \end{aligned} \tag{2.11}$$

où, pour $k = 0, 1, \ldots$, les quantités suivantes doivent être calculées

$$\begin{aligned} h_k &= \sum_{i=0}^{k} c_{k+i} p_i^{(k)} \\ \gamma_k &= \sum_{i=0}^{k} c_{k+i+1} p_i^{(k)} \\ \alpha_k &= \gamma_k + p_{k-1}^{(k)} h_k \\ B_{k+1} &= -\frac{\alpha_k}{h_k} \\ C_{k+1} &= \frac{h_k}{h_{k-1}}. \end{aligned}$$

Cette méthode récursive peut aussi se considérer comme une application de la méthode de bordage aux matrices d'Hankel (voir [19, 37]) et elle a été aussi utilisée par Brezinski pour les calcul des approximants de Padé [3].

Cette méthode a été codée et pour l'optimiser nous avons pensé à ranger les coefficients des polynômes $P_{k+1}(x)$ selon un tableau comme le suivant où dans chaque colonne l'on trouve les coefficients du k-ème polynôme

$k=-2$	$k=-1$	$k=0$	$k=1$	$k=2$	$\cdots$
$p_{-1}^{(-1)}=0$	$p_{-1}^{(0)}=0$	$p_{-1}^{(1)}=0$	$p_{-1}^{(2)}=0$	$p_{-1}^{(3)}=0$	$\cdots$
$p_0^{(-1)}=0$	$p_0^{(0)}=1$	$p_0^{(1)}$	$p_0^{(2)}$	$p_0^{(3)}$	$\cdots$
	$p_1^{(0)}=0$	$p_1^{(1)}=1$	$p_1^{(2)}$	$p_1^{(3)}$	$\cdots$
		$p_2^{(1)}=0$	$p_2^{(2)}=1$	$p_2^{(3)}$	$\cdots$
			$p_3^{(2)}=0$	$p_3^{(3)}=1$	$\cdots$
				$p_4^{(3)}=0$	$\cdots$
					$\vdots$

Dans chaque étape du procédé, le calcul des nouveaux coefficients a besoin seulement des valeurs mémorisées dans les deux colonnes précédentes du tableau et de la valeur h_{k-1}.

2.2.2 Méthodes basées sur les moments modifiés

Les méthodes qui calculent les coefficients de la relation de récurrence à partir des moments peuvent être très mal conditionnées, comme il a été montré par Gautschi dans [20].

Dans [35], en utilisant des familles de polynômes orthogonaux classiques (par exemple Legendre, Chebyshev, ...), Sack et Donovan ont présenté une méthode appelée *méthode des moments modifiés*, pour le calcul des coefficients de la relation de récurrence.

La sensibilité des polynômes orthogonaux par rapport à cette méthode a été étudiée par Gautschi dans [23] et, si les moments modifiés ont été calculés d'une façon suffisament soignée, alors les résultats proposés dans plusieurs articles (voir, par exemple, [22, 35]) semblent montrer que le calcul des coefficients B_i et C_i devient plus stable.

Les moments sont calculés dans une forme modifiée de la façon suivante

$$v_i = c(\Pi_i), \qquad i = 0, 1, \ldots$$

où $\{\Pi_n\}$ est une famille auxiliaire de polynômes orthogonaux unitaires qui satisfont aux relations suivantes

$$\begin{aligned} \Pi_{-1}(x) &= 0 \\ \Pi_0(x) &= 1 \\ \Pi_{k+1}(x) &= (x + B'_{k+1})\Pi_k(x) - C'_{k+1}\Pi_{k-1}(x), \qquad k = 0, 1, \ldots. \end{aligned} \tag{2.12}$$

Evidemment quand $\Pi_i(x) = x^i$ alors $v_i = c_i$.

L'approche de Sack et Donovan a été généralisée dans d'autres travaux ultérieurs et, comme nous avons montré dans [8], la famille auxiliaire $\{\Pi_n\}$ peut ne pas être orthogonale et donc les moments modifiés peuvent se calculer à partir d'une famille quelconque de polynômes. Récemment, Golub et Gutknecht [26], ont présenté un travail qui regroupe tous les algorithmes de base.

Algorithme de Chebyshev modifié

Quand les intervalles d'orthogonalité des familles $\{P_n\}$ et $\{\Pi_n\}$ sont finis, un algorithme qui semble être assez stable pour le calcul des coefficients B_i et C_i est l'*algorithme de Chebyshev modifié*. Cette méthode a été donnée par Wheeler dans [38] (voir aussi [4, 22]) et elle est décrite dans la suite.

On considère la matrice Z avec les éléments

$$z_{k,i} = c(P_k\Pi_i), \qquad k, i = 0, 1, 2, \ldots.$$

Tous les $z_{k,i}$, avec $i < k$, sont égaux à zéro, d'après les conditions d'orthogonalité et la première ligne contient les moments modifiés car $z_{0,i} = c(P_0\Pi_i) = c(\Pi_i) = v_i$.

La méthode, pour $k = 1, 2, \ldots$, est donc la suivante

$$\begin{aligned} z_{k,i} &= z_{k-1,i+1} - B'_{i+1}z_{k-1,i} + C'_{i+1}z_{k-1,i-1} + B_k z_{k-1,i} - C_k z_{k-2,i}, \quad i = k, k+1, \ldots \\ B_{k+1} &= B'_{k+1} + \frac{z_{k-1,k}}{z_{k-1,k-1}} - \frac{z_{k,k+1}}{z_{k,k}} \\ C_{k+1} &= \frac{z_{k,k}}{z_{k-1,k-1}} \end{aligned}$$

avec les conditions initiales

$$\begin{aligned} z_{-1,i} &= 0, & & i = 1, 2, \ldots \\ z_{0,i} &= v_i, & & i = 0, 1, 2, \ldots \\ B_1 &= B'_1 - \frac{v_1}{v_0} \\ C_1 &= v_0. \end{aligned}$$

Evidemment, si l'on prend $\Pi_i = x^i$, cette méthode coïncide avec la méthode de Chebyshev qui a été décrite dans la section précédente.

2.2.3 Application à la quadrature

Nous allons donc présenter un cas d'application des polynômes orthogonaux pour la résolution d'un problème de quadrature pour un ensemble d'intégrales qui est très important dans les calculs de mécanique quantique (voir [32, 33]). C'est l'ensemble

$$\int_{-1}^{1} e^{-\alpha x} f(x)dx\,, \quad \alpha \in \mathbf{R}$$

où $f(x)$ est connue en un nombre fini de points et où la fonction poids $e^{-\alpha x}$ est positive pour tout $x \in [-1,1]$ et telle que $\int_{-1}^{1} e^{-\alpha x}dx < +\infty$.

On veut utiliser une formule de quadrature de Gauss telle que

$$I = \int_{-1}^{1} e^{-\alpha x} f(x)dx = \sum_{i=1}^{n} A_i^{(n)} f(x_i) + R_n = I_n + R_n$$

où les x_i, $(-1 < x_1 < \cdots < x_n < 1)$ sont les racines réelles et distinctes du polynôme $P_n \in \mathsf{P}_n$ orthogonal dans l'interval $[-1,1]$ par rapport au poids $e^{-\alpha x}$ et les $A_i^{(n)}$ sont les poids de la formule de quadrature qui dépendent de n et de la distribution des nœuds. De cette manière la formule est exacte sur P_{2n-1}.

Ce problème d'intégration avait déjà été étudié par Morandi dans [31] et les résultats (racines et poids) avaient été donnés sous forme de tableaux. Dans [32] nous avons, à la place, présenté un algorithme complet qui permet de résoudre le problème pour n'importe quelle valeur de la constante α et qui permet de continuer le calcul d'une façon itérative en essayant d'atteindre la précision désirée.

On sait qu'une telle formule de quadrature est convergente et stable sur $C_\infty[-1,1]$ dans le sens que, si l'on considère un espace général de fonctions qui contient l'espace des polynômes, et une fonctionnelle c positive et definie sur l'espace de la fonction, alors $c(f)$ peut être approximé par $c(L_n) = \sum_{i=1}^{n} A_i^{(n)} f(x_i)$, où L_n est le polynôme qui interpole f aux zéros du polynôme orthogonal P_n (voir Brezinski [4], ch. 2). En plus l'erreur de cette formule peut s'écrire (voir Ghizzetti [24], ch. 6)

$$R_n = \sum_{i=0}^{n} \int_{x_i}^{x_{i+1}} \varphi_i(x) f^{(2n)}(x)dx$$

où les x_i $(i = 1,\ldots,n)$ sont les zéros du polynôme orthogonal P_n, $x_0 = -1, x_{n+1} = 1$ et les φ_i $(i = 0,\ldots,n+1)$ sont les intégrales de l'équation différentielle

$$\frac{d^{2n}\varphi}{dx^{2n}} = e^{-\alpha x}$$

et en plus $\varphi_0(x), \varphi_{n+1}(x)$ doivent satisfaire aux conditions suivantes

$$\varphi_0(-1) = \varphi_0'(-1) = \cdots = \varphi_0^{(2n-1)}(-1) = 0$$

$$\varphi_{n+1}(1) = \varphi_{n+1}'(1) = \cdots = \varphi_{n+1}^{(2n-1)}(1) = 0.$$

On obtient donc

$$\begin{aligned}
\varphi_0(x) &= \frac{e^{-\alpha x}}{\alpha^{2n}} - \frac{e^{\alpha}}{\alpha^{2n}} + \frac{e^{\alpha}(x+1)}{\alpha^{2n-1}} + \cdots + \\
&\quad (-1)^{t+1} \cdot \frac{e^{\alpha}(x+1)^t}{t!\alpha^{2n-t}} + \cdots + \frac{e^{\alpha}(x+1)^{2n-1}}{(2n-1)!\alpha} \\
\varphi_{n+1}(x) &= \frac{e^{-\alpha x}}{\alpha^{2n}} - \frac{e^{-\alpha}}{\alpha^{2n}} + \frac{e^{-\alpha}(x-1)}{\alpha^{2n-1}} + \cdots + \\
&\quad (-1)^{t+1} \cdot \frac{e^{-\alpha}(x-1)^t}{t!\alpha^{2n-t}} + \cdots + \frac{e^{-\alpha}(x-1)^{2n-1}}{(2n-1)!\alpha} \\
\varphi_i(x) &= \varphi_0(x) - \sum_{j=1}^{i} A_j^{(n)} \cdot \frac{(x-x_j)^{2n-1}}{(2n-1)!}, \quad i = 1, \ldots, n.
\end{aligned}$$

L'expression de l'erreur devient

$$\begin{aligned}
R_n &= \int_{-1}^{1} \left(\frac{e^{-\alpha x}}{\alpha^{2n}} + \sum_{t=0}^{2n-1} (-1)^{t+1} \cdot \frac{e^{\alpha}(x+1)^t}{t!\alpha^{2n-t}} \right) f^{(2n)}(x)dx \\
&\quad - \sum_{i=1}^{n} \sum_{j=1}^{n} A_j^{(n)} \int_{x_i}^{x_i+1} \frac{(x+x_j)^{2n-1}}{(2n-1)!} f^{(2n)}(x)dx.
\end{aligned}$$

2.2.4 Calcul des moments et des moments modifiés

Pour calculer les polynômes orthogonaux il faut calculer de la façon la plus précise possible, les moments et les moments modifiés. Comme nous avons montré dans [32, 33] ils peuvent être analytiquement calculés.

Soit donc c la fonctionnelle definie sur l'espace des polynômes réels P par la relation

$$c(x^i) = c_i = \int_{-1}^{1} x^i e^{-\alpha x} dx, \qquad \alpha \in \mathsf{R}, \quad i = 0, 1, \ldots$$

Les moments peuvent se calculer analytiquement soit récursivement soit directement avec les théorèmes suivants

Théorème 2.1 *Si* $c_i = \int_{-1}^{1} x^i e^{-\alpha x} dx, \alpha \in \mathsf{R}, i = 0, 1, \ldots,$ *alors on a*

$$\begin{aligned}
c_0 &= -\frac{1}{\alpha} \cdot \left(e^{-\alpha} - e^{\alpha} \right) \\
\alpha \cdot c_i &= -\left[e^{-\alpha} - (-1)^i e^{\alpha} \right] + i \cdot c_{i-1}, \qquad i = 1, 2, \ldots
\end{aligned}$$

Démonstration.

Si l'on considère l'intégrale indéfinie $I_i(x) = \int x^i e^{-\alpha x} dx$, en utilisant le changement de variables $t = -\alpha x$, on aura

$$\begin{aligned}
I_i(x) &= \frac{1}{(-\alpha)^{i+1}} \cdot \int t^i e^t dt = \frac{1}{(-\alpha)^{i+1}} \cdot I_i(t) \\
\text{avec} \qquad c_i &= |I_i(x)|_{-1}^{1} = \frac{1}{(-\alpha)^{i+1}} \cdot |I_i(t)|_{\alpha}^{-\alpha}.
\end{aligned}$$

En intégrant par parties $I(t)$ on obtient

$$\begin{aligned} I_0(t) &= e^t \\ I_i(t) &= t^i e^t - i \cdot I_{i-1}(t)\,, \qquad i = 1, 2, \ldots \end{aligned}$$

donc

$$\begin{aligned} c_0 &= \frac{1}{-\alpha} \cdot |I_0(t)|_\alpha^{-\alpha} = -\frac{1}{\alpha} \cdot \left(e^{-\alpha} - e^{\alpha}\right) \\ c_i &= \frac{1}{(-\alpha)^{i+1}} \cdot \left| t^i e^t - i \cdot I_{i-1}(t) \right|_\alpha^{-\alpha} \\ &= \frac{1}{(-\alpha)^{i+1}} \cdot \left\{ (-\alpha)^i \left[e^{-\alpha} - (-1)^i e^{\alpha} \right] - i \cdot (-\alpha)^i \cdot c_{i-1} \right\} \\ &= -\frac{1}{\alpha} \cdot \left[e^{-\alpha} - (-1)^i e^{\alpha} - i \cdot c_{i-1} \right] \end{aligned}$$

et finalement $\alpha \cdot c_i = -\left[e^{-\alpha} - (-1)^i e^{\alpha}\right] + i \cdot c_{i-1}$. □

Théorème 2.2

$$c_i = (-1)^i \cdot i! \cdot \sum_{j=0}^{i} \frac{(-1)^j}{\alpha^{j+1}(i-j)!} \cdot \left[e^{\alpha} - (-1)^{i+j} e^{-\alpha} \right]$$

Démonstration.
Si $i = 0$, alors

$$c_0 = (-1)^0 \cdot 0! \cdot \frac{(-1)^0}{\alpha \cdot 0!} \cdot \left[e^{\alpha} - (-1)^0 e^{-\alpha} \right] = \frac{1}{\alpha} \cdot \left(e^{\alpha} - e^{-\alpha} \right).$$

On suppose que la formule soit vraie pour i. D'après la formule de récurrence on a

$$\begin{aligned} c_{i+1} &= -\frac{1}{\alpha} \left[e^{-\alpha} - (-1)^{i+1} e^{\alpha} \right] + \frac{(i+1)}{\alpha} \cdot c_i \\ &= -\frac{1}{\alpha} \left[e^{-\alpha} - (-1)^{i+1} e^{\alpha} \right] + (-1)^{i+1} (i+1)! \sum_{j=1}^{i+1} \frac{(-1)^j}{\alpha^{j+1}(i+1-j)!} \cdot \left[e^{\alpha} - (-1)^{i+1+j} e^{-\alpha} \right]. \end{aligned}$$

Si dans le second membre on pose $j = 0$, on obtient le premier membre, puisque

$$(-1)^{i+1}(i+1)! \cdot \frac{(-1)^0}{\alpha(i+1)!} \cdot \left[e^{\alpha} - (-1)^{i+1} e^{-\alpha} \right] =$$
$$\frac{(-1)^{i+1}}{\alpha} \cdot \left[e^{\alpha} - (-1)^{i+1} e^{-\alpha} \right] = -\frac{1}{\alpha} \cdot \left[e^{-\alpha} - (-1)^{i+1} e^{\alpha} \right]$$

et donc

$$c_{i+1} = (-1)^{i+1}(i+1)! \sum_{j=0}^{i+1} \frac{(-1)^j}{\alpha^{j+1}(i+1-j)!} \cdot \left[e^{\alpha} - (-1)^{i+1+j} e^{-\alpha} \right]$$

et l'expression pour c_i est verifiée. □

Pour ce qui concerne les moments modifiés, vu l'intervalle d'intégration on a choisi la famille de polynômes orthogonaux unitaires suivante

$$\begin{aligned} \Pi_0(x) &= T_0(x) \\ \Pi_n(x) &= \frac{T_n(x)}{2^{n-1}}, \qquad n = 1, 2, \ldots \end{aligned}$$

où les T_n sont les polynômes de Chebyshev de première espèce.

Donc les polynômes auxiliaires des relations (2.12) seront définis par

$$\begin{aligned} \Pi_{-1}(x) &= 0 \\ \Pi_0(x) &= T_0(x) = 1 \\ \Pi_{k+1}(x) &= x\Pi_k(x) - C'_{k+1}\Pi_{k-1}(x), \quad k = 0, 1, \ldots \\ B'_{k+1} &= 0, \qquad\qquad\qquad\quad k = 0, 1, \ldots \\ C'_{k+1} &= 1/2, \qquad\qquad\qquad\quad k = 0, 1 \\ C'_{k+1} &= 1/4, \qquad\qquad\qquad\quad k = 2, 3, \ldots \end{aligned}$$

Pour calculer les moments modifiés on a deux théorèmes qui sont obtenus directement à partir des théorèmes (2.1) et (2.2).

Si l'on considère les moments modifiés $v_i = c(\Pi_i) = \int_{-1}^{1} \Pi_i(x)e^{-\alpha x}dx$, $\alpha \in \mathbb{R}$, $i = 0, 1, \ldots$, où Π_i est le i–ème polynôme unitaire de Chebyshev de première espèce, on aura

Théorème 2.3

$$\begin{aligned} v_0 &= c_0 \\ v_1 &= c_1 \\ v_i &= i \cdot \sum_{m=0}^{[\frac{i}{2}]} (-1)^m \cdot \frac{(i-m-1)!}{m!(i-2m)!} \cdot 2^{-2m} \cdot c_{i-2m}, \qquad i = 2, 3, \ldots \end{aligned}$$

Démonstration. Puisque $\Pi_0(x) = 1$ et $\Pi_1(x) = x$, la première des deux relations est immédiatement vraie.

La forme explicite de $T_i(x)$ (voir, pour exemple, [1]) est la suivante

$$\begin{aligned} T_i(x) &= \frac{i}{2} \cdot \sum_{m=0}^{[\frac{i}{2}]} (-1)^m \cdot \frac{(i-m-1)!}{m!(i-2m)!} \cdot (2x)^{i-2m} \\ &= i \cdot 2^{i-1} \cdot \sum_{m=0}^{[\frac{i}{2}]} (-1)^m \cdot \frac{(i-m-1)!}{m!(i-2m)!} \cdot 2^{-2m} \cdot x^{i-2m} \end{aligned}$$

donc

$$\Pi_i(x) = i \cdot \sum_{m=0}^{[\frac{i}{2}]} (-1)^m \cdot \frac{(i-m-1)!}{m!(i-2m)!} \cdot 2^{-2m} \cdot x^{i-2m}.$$

Si l'on applique la fonctionnelle c aux deux membres de la relation on obtient

$$\begin{aligned} v_i = c(\Pi_i(x)) &= i \cdot \sum_{m=0}^{\left[\frac{i}{2}\right]} (-1)^m \cdot \frac{(i-m-1)!}{m!(i-2m)!} \cdot 2^{-2m} \cdot c\left(x^{i-2m}\right) \\ &= i \cdot \sum_{m=0}^{\left[\frac{i}{2}\right]} (-1)^m \cdot \frac{(i-m-1)!}{m!(i-2m)!} \cdot 2^{-2m} \cdot c_{i-2m} \end{aligned}$$

□

Théorème 2.4

$$v_i = i \cdot \sum_{m=0}^{\left[\frac{i}{2}\right]} (-1)^{i-m} \cdot 2^{-2m} \cdot \frac{(i-m-1)!}{m!} \cdot \sum_{j=0}^{i-2m} \frac{(-1)^j}{\alpha^{j+1}(i-2m-j)!} \cdot \left[e^{\alpha} - (-1)^{i-2m+j} e^{-\alpha}\right],$$
$$i = 2, 3, \ldots$$

Démonstration. En substituant la formule du théorème 2.2 de celle du théorème 2.3 on obtient le résultat après des simplifications évidentes. □

2.2.5 Comparaisons entre les moments et les moments modifiés

Dans [33], en prenant comme exemple notre intégrale, nous avons présenté plusieurs comparaisons qui concernent le calcul numérique et symbolique des moments, des moments modifiés et des coefficients B_i et C_i de la formule de récurrence (2.3).

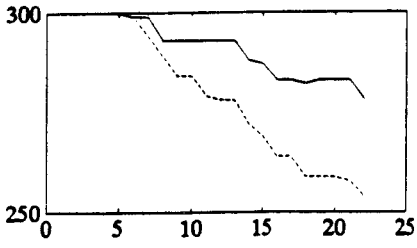
Les résultats numériques ont été obtenus sur un μVAX 3100 en utilisant la mémorisation de type H-float pour obtenir une meilleure précision (environ 33 chiffres décimaux significatifs). Ces résultats en effet, n'ont pas permis de décider si, dans notre application, les méthodes qui utilisent les moments modifiés, donnent vraiment un calcul meilleur des B_i et C_i. Donc on a decidé d'utiliser le calcul formel pour essayer de donner une réponse à cette question.

Pour le calcul formel nous avons utilisé le logiciel *Mathematica* [39], sur un Macintosh II. Ce logiciel permet d'effectuer les calcul formels en demandant un certain nombre de chiffres décimaux exacts (nous en avons démandé 300). Pendant le calcul, le logiciel diminue automatiquement le nombre de chiffres en donnant seulement ceux qu'il considère bien calculés (ce sont les chiffres que dans la suite nous indiquerons comme *exacts*).

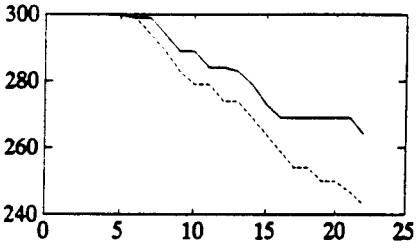
Dans toutes les comparaisons nous avons choisi trois valeurs de la variable α, c'est à dire $\alpha = 2.0$, 5.0 et 15.0 et un degré pour le polynôme orthogonal P_i qui varie de $i = 1$ à $i = 22$.

Dans la figure 1 nous avons d'abord considéré une comparaison entre les coefficients B_i et C_i obtenus formellement avec *Mathematica* et en utilisant la méthode de Chebyshev et la méthode de Chebyshev modifiée. En x on représente les valeurs de i, tandis que, en y, on considère le nombre de chiffres décimaux exacts donnés par *Mathematica*. Comme l'on peut voir, l'algorithme de Chebyshev modifié (ligne continue) donne pour les valeurs $\alpha = 2.0$ et $\alpha = 5.0$, des résultats meilleurs que ceux obtenus par la méthode de Chebyshev (tirets). Quand α augmente les deux méthodes deviennent presque équivalentes.

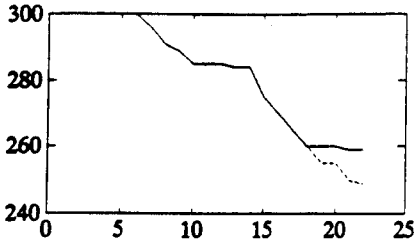
B_i



$\alpha = 2.0$

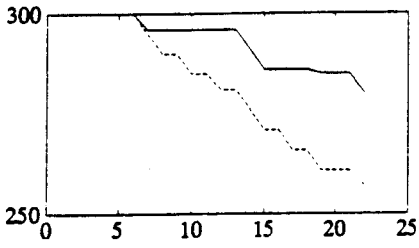


$\alpha = 5.0$

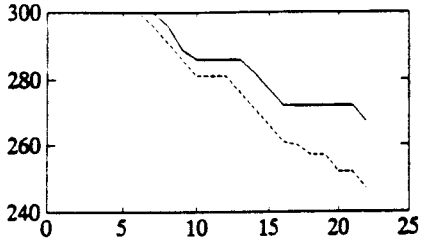


$\alpha = 15.0$

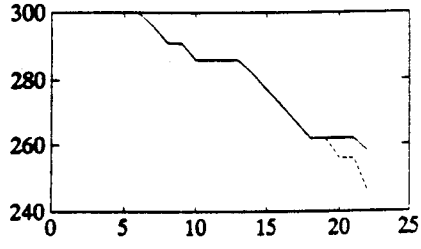
C_i



$\alpha = 2.0$



$\alpha = 5.0$

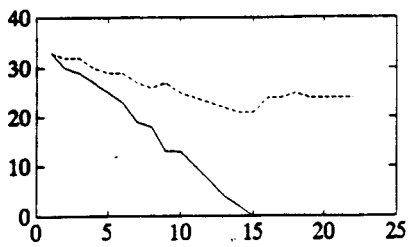


$\alpha = 15.0$

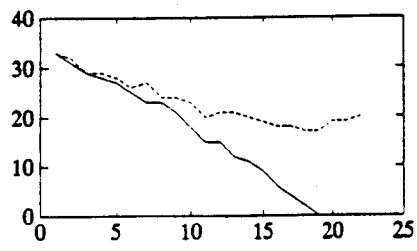
Fig. 1

Dans la figure 2 on montre une comparaison entre les résultats numériques et les résultats obtenus avec *Mathematica*. Ici les tirets montrent le nombre de chiffres en commun entre les résultats des trois méthodes suivantes: méthode de Chebyshev, méthode de bordage et méthode de Chebyshev modifiée. Les lignes continues représentent le nombre de chiffres en commun entre les trois résultats numériques et les résultats obtenus avec *Mathematica*. On voit très bien que, même si l'on pourrait croire que les chiffres en commun entre les trois méthodes numériques sont "exacts", dans la réalité seulement un très petit nombre d'entre eux est vraiment "exact". Donc toutes les trois méthodes donnent une fausse réponse de la même façon.

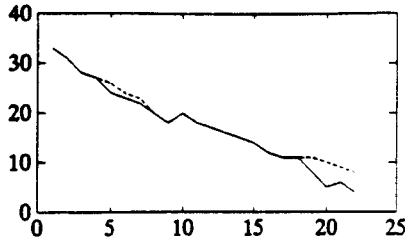
B_i



$\alpha = 2.0$



$\alpha = 5.0$



$\alpha = 15.0$

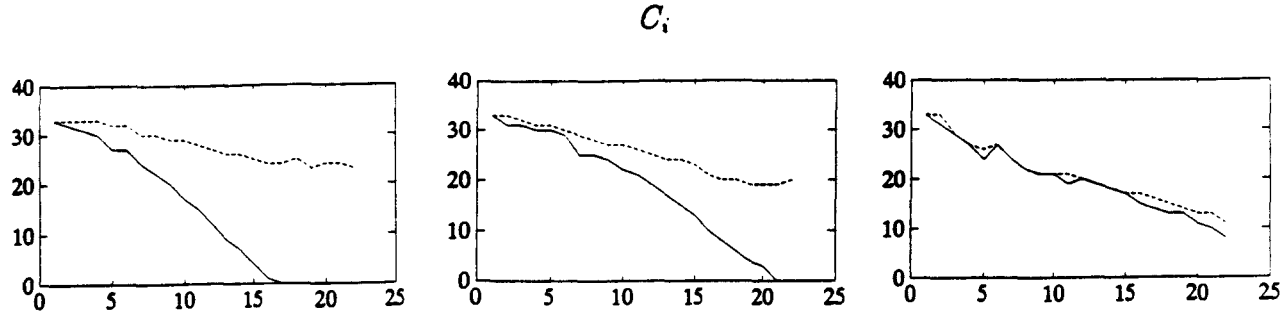


Fig. 2

Il s'agit donc de comprendre pourquoi la méthode des moments modifiés (qui est certainement plus stable, comme on l'a vu dans la figure 1) va perdre ses avantages surtout quand $\alpha = 2.0$ et $\alpha = 5.0$. La figure 3 montre le nombre de chiffres en commun entre les résultats numériques et les résultats formels pour les moments (tirets) et les moments modifiés (lignes continues). On peut bien voir que le calcul des moments est plus stable que celui des moments modifiés et cela explique probablement les résultats de la figure 2.

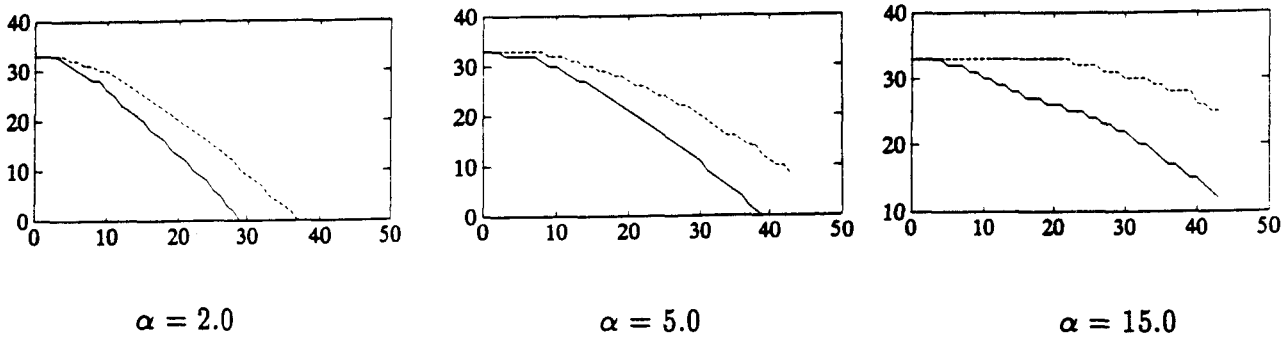


Fig. 3

Pour essayer de vérifier notre hypothèse, nous avons donné au logiciel qui faisait les calculs numériques par la méthode de Chebyshev (tirets) et la méthode de Chebyshev modifiée (lignes continues), les valeurs des moments et des moments modifiés obtenues avec *Mathematica*, donc on a donné les valeurs "exactes" de ces quantités. De cette manière on a retrouvé dans la figure 4 un résultat semblable à celui de la figure 1 et donc la méthode des moments modifiés est vraiment plus stable que l'autre.

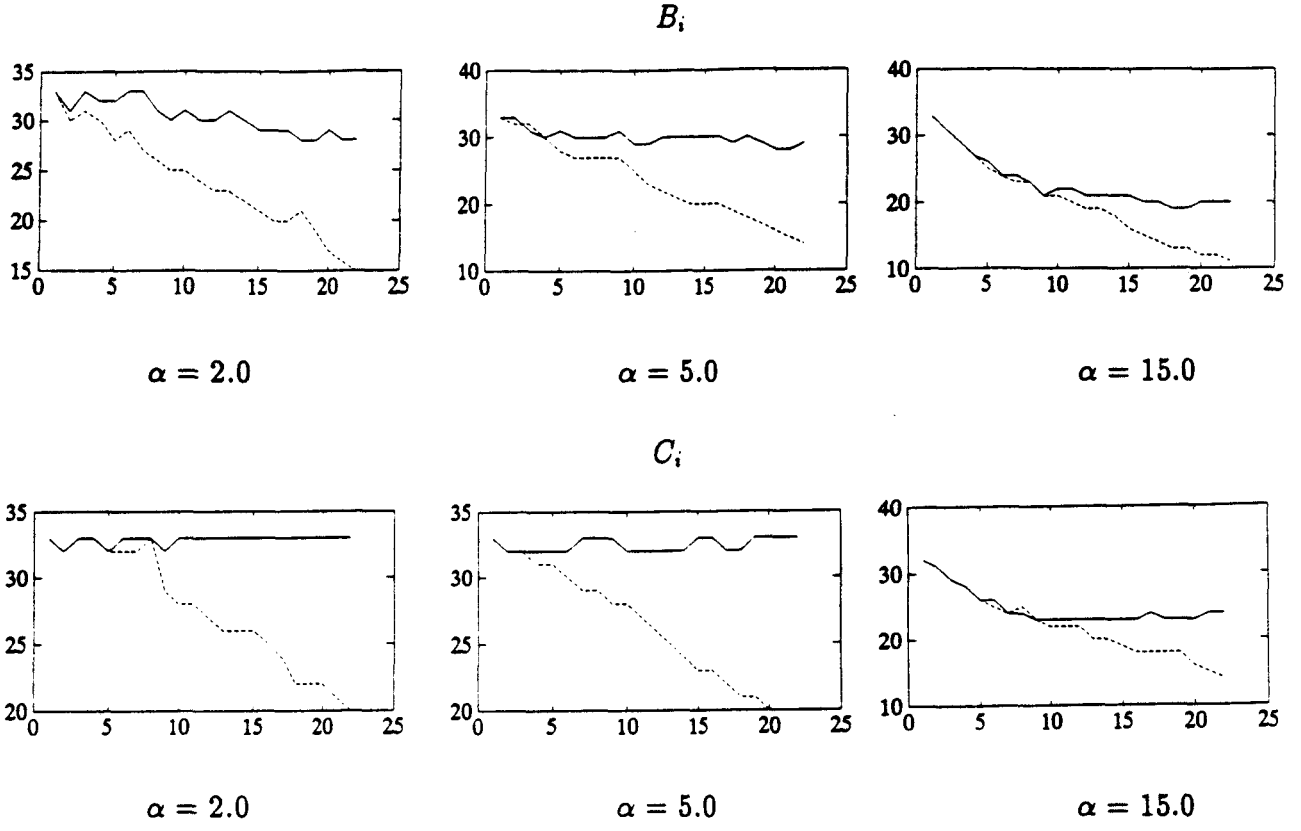


Fig. 4

D'après ce que nous avons trouvé on peut donc déduire que, dans notre application, si l'on utilise la méthode des moments modifiés, on obtient des résultats meilleurs pour les coefficients de la relation de récurrence, mais cela est vrai seulement si l'on connaît ces moments modifiés dans leur forme analytique. Mais même cela n'est pas suffisant car si le calcul des moments modifiés à partir de leur formule est moins précis que celui des moments, l'amélioration apportée de la méthode des moments modifiés peut disparaître.

Sur le problème de la quadrature de Gauss avec les moments modifiés on peut voir l'article [21] de Gautschi.

2.2.6 L'algorithme de quadrature

Dans [32] nous avons donc proposé l'algorithme complet pour l'approximation de notre intégrale. Ce travail est antérieur à [33] et nous avions utilisé, pour le calcul des coefficients des polynômes orthogonaux, la méthode de bordage (2.11). Comme nous l'avons vu dans la section précédente, en faisant uniquement les calculs d'une façon numérique, cette méthode est comparable à la méthode des moments modifiés.

Maintenant, pour appliquer la formule de quadrature, il nous faut trouver les racines des polynômes. Si l'on considère

$$P_n(x) = \left(x - x_1^{(n)}\right) \cdot \left(x - x_2^{(n)}\right) \cdot \ldots \cdot \left(x - x_n^{(n)}\right)$$

on sait que, vu la fonction poids $e^{-\alpha x}$, les n zéros de P_n seront dans l'intervalle $[-1,1]$. Pour les trouver nous avons utilisé la proprieté des séparation des racines des polynômes orthogonaux. Comme il est bien connu (voir, par exemple, Davis and Rabinowitz [16]) si l'on a un polynôme P_n orthogonal par rapport à une fonction poids non négative dans un intervalle $[a,b]$ alors les proprietés suivantes sont valables

- Les racines du polynôme P_n ne coïncident pas avec celles du polynôme P_{n+1}.
- Dans chaque intervalle $\left[a, x_1^{(n)}\right]$, $\left[x_2^{(n)}, x_3^{(n)}\right], \ldots, \left[x_{n-1}^{(n)}, x_n^{(n)}\right], \left[x_n^{(n)}, b\right]$ des racines de P_n il y a exactement une racine de P_{n+1}.

Le calcul récursif de P_n implique le calcul de tous les $(n-1)$ polynômes orthogonaux précédents. On peut donc utiliser ces proprietés pour calculer les racines de P_n. Donc, à chaque étape k, on va appliquer une méthode pour trouver un zéro dans chaque sous-intervalle de $[-1,1]$ obtenu à l'étape précédente.

L'algorithme pour le calcul des racines est donc le suivant

1. Calculer les zéros $x_1^{(2)}$ et $x_2^{(2)}$ de $P_2(x)$ (avec $x_1^{(2)} < x_2^{(2)}$)
2. Pour $k = 3, \ldots, n$
   - Calculer $-1 < x_1^{(k)} < x_1^{(k-1)}$
   - Pour $j = 2, \ldots, k-1$
     - Calculer $x_{j-1}^{(k-1)} < x_j^{(k)} < x_j^{(k-1)}$
   - Calculer $x_{k-1}^{(k-1)} < x_k^{(k)} < 1$

où n est le degré du polynôme orthogonal que l'on veut.

Pour le calcul des racines on a choisi l'algorithme de Brent [2] qui est une combinaison d'interpolation linéaire, d'interpolation quadratique inverse et de bisection, et qui a une convergence super-linéaire.

Si l'on a déterminés les zéros de $P_n(x)$ on obtient une formule exacte sur P_{2n-1}. En plus, pour les polynômes $Q_j(x)$ de degré $j \leq 2n-1$ on aura R_n=0 et la formule devient

$$\int_{-1}^{1} Q_j(x) e^{-\alpha x} dx = \sum_{i=1}^{n} A_i^{(n)} Q_j(x_i).$$

Si l'on considère $Q_j(x) = x^j, j = 0, 1, \ldots, n-1$ alors

$$\int_{-1}^{1} Q_j(x) e^{-\alpha x} dx = c(x^j) = c_j$$

où les c_j sont les moments connus. On obtient donc le système linéaire à n inconnues $A_i^{(n)}$ suivant

$$\begin{cases} A_1^{(n)} + A_2^{(n)} + \cdots + A_n^{(n)} & = & c_0 \\ A_1^{(n)} x_1 + A_2^{(n)} x_2 + \cdots + A_n^{(n)} x_n & = & c_1 \\ A_1^{(n)} x_1^2 + A_2^{(n)} x_2^2 + \cdots + A_n^{(n)} x_n^2 & = & c_2 \\ \vdots & & \vdots \\ A_1^{(n)} x_1^n + A_2^{(n)} x_2^n + \cdots + A_n^{(n)} x_n^n & = & c_{n-1}. \end{cases}$$

Un tel système, avec matrice de Vandermonde, est toujours non singulier mais il est mal conditionné. Mais le calcul des $A_i^{(n)}$ peut se faire en utilisant une formule que dérive de la relation de récurrence [4]

$$A_i^{(k)} = \frac{h_{k-1}}{P_k'(x_i) \cdot P_{k-1}(x_i)}, \quad i = 1, \ldots, k$$

où $x_1, \ldots, x_k$ sont les zéros de $P_k(x)$.

Donc on a finalement obtenu l'algorithme général pour l'approximation de l'intégrale $\int_{-1}^{1} e^{-\alpha x} f(x) dx$ avec une formule de quadrature de Gauss.

Algorithme de quadrature

1. Donner une valeur de $\alpha \in \mathbb{R}$.
2. Initialiser les coefficients de $P_{-1}(x)$ et de $P_0(x)$.
3. Pour $k = 0, 1, \ldots, n-1$
   - Calculer les moments c_{2k} et c_{2k+1} (avec la formule directe ou récursive).
   - Calculer les coefficients de $P_{k+1}(x)$ avec la méthode de bordage.
   - Calculer les racines de $P_{k+1}(x)$ dans chaque sous-intervalle déterminé par les racines de $P_k(x)$ avec l'algorithme de Brent.
4. Calculer les poids $A_i^{(n)}, i = 1, \ldots, n$.

Si l'on désire passer de n à $n+1$ nœuds, il est suffisant d'exécuter encore une fois la boucle interne (index k) c'est à dire de calculer deux moments additionnels et calculer les coefficients de $P_{n+1}(x)$ en utilisant les coefficients connus de $P_{n-1}(x)$ et de $P_n(x)$. Après l'algorithme de Brent pourra calculer les $(n+1)$ nouveaux nœuds et ensuite on calculera les nouveaux poids $A_i^{(n+1)}$.

Résultats numériques

Puisque l'on voulait obtenir une bonne approximation, nous avons developpé un logiciel qui a été executé sur un VAX-11/750 en utilisant la mémorisation H-float (environ 33 chiffres décimaux significatifs). Le logiciel, écrit en FORTRAN-77 et que l'on va donner partiellement dans la section suivante, a été essayé sur des exemples par lesquels on connaît la valeur exacte de l'intégrale. Pour l'algorithme de Brent, on a pris la subroutine de la bibliothèque IMSL [30] (pp. 770-772) en la modifiant pour la mémorisation H-float et en l'adaptant pour le cas particulier d'une fonction polynomiale.

Nous allons montrer l'un de ces exemples qui concerne le calcul approché de l'intégrale

$$I = \int_{-1}^{1} e^{-\alpha x} \left(1 + \sin \frac{\pi}{2} x\right) dx.$$

Sa valeur est (voir, par exemple, [27], p. 196)

$$I = \frac{1}{\alpha} \cdot \left(e^{\alpha} - e^{-\alpha}\right) - \frac{4\alpha}{4\alpha^2 + \pi^2} \cdot \left(e^{\alpha} + e^{-\alpha}\right).$$

n	3	7	10	15	3	7	10	15
α	Nouveau algorithme				Gauss-Legendre			
2.5	4	13	21	31	1	8	14	26
5.5	4	13	19	30	2	4	9	19
16.5	5	15	23	25	0	1	2	7

Figure 2.1: Nombre de chiffres significatifs décimaux exacts.

Dans la figure 2.1 est répresenté le nombre de chiffres décimaux exacts trouvés avec trois valeurs de α et quatre valeurs du degré n de $P_n(x)$. Comme l'on peut voir, l'algorithme est précis pour chaque valeur de n et aussi pour les polynômes de degré plus grand que 10.

Dans la même figure il y a une comparaison entre la valeur exacte et l'approximation que l'on obtient en appliquant la quadrature de Gauss avec une fonction poids égale à 1 et en utilisant les polynômes unitaires classiques de Legendre dans l'intervalle $[-1, 1]$. Dans ce cas nous avons utilisé encore une subroutine de la IMSL [30] (pp. 611–613) modifiée pour la mémorisation H-float. Cette subroutine qui s'appelle GAUSSQUADRULE a été proposée par Golub et Welsh [25] et elle obtient aussi les points et les poids en utilisant les coefficients de la relation de récurrence à trois termes.

Dans la suite nous allons montrer une comparaison entre le calcul approché des moments et leur valeur obtenue avec la formule directe (voir théorhème 2.2). On a cherché à obtenir le degré maximum pour les polynômes, en gardant la précision des résultats. Dans la figure

α rang	Maximum degré n	Min. chiffres décimales exactes
$]0.0, 4.0]$	19	26
$]4.0, 8.0]$	22	24
$]8.0, 12.0]$	25	22
$]12.0, 16.0]$	27	21
$]16.0, 20.0]$	28	19

Figure 2.2: Comparaison entre deux calculs différents pour les moments.

2.2 pour différentes valeurs de α on donne le degré maximum n qu'on arrive à atteindre avec une bonne précision et le nombre le plus petit de chiffres décimaux exacts trouvés pour ces polynômes.

Pour chaque polynôme de degré n nous avons calculé et comparé les moments c_i ($i = 0, \ldots, 2n-1$) obtenus avec les deux formules suivantes

$$c_i = (-1)^i \cdot i! \cdot \sum_{j=0}^{i} \frac{(-1)^j}{\alpha^{j+1}(i-j)!} \cdot \left[e^{\alpha} - (-1)^{i+j} e^{-\alpha}\right]$$

$$c_i = \int_{-1}^{1} x^i e^{-\alpha x} dx = \sum_{j=1}^{n} A_j^{(n)} x_j^i \,.$$

Pour d'autres considérations sur le calcul des moments, voir Gautschi [22].

2.3 Logiciels

En ce qui concerne les logiciels que nous avons développés sur les sujets de ce chapitre il faut mentionner les logiciels en FORTRAN que nous avons écrits pour les applications de la section 2.1.3, des méthodes de calcul des coefficients de la relation de récurrence de la section 2.2 qui utilisent les moments et les moments modifiés. En plus il y a l'application à la quadrature de la section 2.2.3 pour laquelle, dans la suite, nous allons montrer une partie du logiciel que nous avons developpé et qui montre l'algorithme itératif complet ainsi que l'implémentation de la méthode de bordage (2.11). Naturellement il y a aussi tous les logiciels en *Mathematica* que l'on a devéloppé pour les comparaisons de la section 2.2.5.

```
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      PROGRAM QUADR
C PURPOSE: EVALUATE THE INTEGRAL VIA A GAUSSIAN QUADRATURE +
C          FORMULA THROUGH THE MONIC ORTHOGONAL POLYNOMIAL +
C                                              -alpha * x +
C          WITH RESPECT TO THE WEIGHT FUNCTION e           +
C          IN [-1,1].                                      +
```

```
C INPUT:   ALPHA   REAL CONSTANT OF THE EXPONENTIAL        +
C          N       NUMBER OF KNOTS OR DEGREE OF POLYNOMIAL +
C          EPS     THE PRECISION FOR THE SOLUTION          +
C OUTPUT:  QUADINT APPROXIMATED VALUE OF INTEGRAL          +
C EXTERNAL FUNCTIONS:                                      +
C          POL     COMPUTE THE VALUE OF POLYNOMIAL P(x)    +
C          POLD    COMPUTE THE VALUE OF THE DERIVATE OF    +
C                  POLYNOMIAL P(x)                         +
C          CMOM    COMPUTE THE MOMENTS                     +
C          FUN     COMPUTE THE FUNCTION f(x)               +
C          RINT    COMPUTE THE ANALYTICAL VALUE OF INTEGRAL+
C EXTERNAL SUBROUTINES:                                    +
C          CBORD   COMPUTE THE COEFFICIENTS OF P(x)        +
C          ROOT    COMPUTE THE ROOTS OF A SECOND DEGREE    +
C                  EQUATION                                +
C          ZBREN   COMPUTE THE ZERO OF THE POLYNOMIAL P(x) +
C                  IN [A,B]                                +
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      IMPLICIT REAL*16 (A-H,O-Z)
      DIMENSION C(0:100), COEFF1(-1:100), COEFF2(-1:100),
     *          ZEROS(100), H(2), WEIGHT(100)
      EXTERNAL POL, POLD
C
C ... INIT VALUES FOR THE PROGRAM AND INPUT THE VALUES FOR
C     ALPHA, N AND EPS
     .....................................................
C ... INIT THE COEFFICIENTS FOR RECURSIVE METHOD CBORD
      CALL CBORD (COEFF1, COEFF2, C, -2, H)
C
C ... LOOP FOR FAMILY OF POLYNOMIALS TO REACH THE DEGREE N
      DO 400 IDEG = 1, N
100      CONTINUE
C
C ... COMPUTE THE ADDITIONAL TWO MOMENTS
         ISTART   = 2 * IDEG - 2
         DO 200 I = ISTART, ISTART+1
             C(I) = CMOM (I, ALPHA)
200      CONTINUE
C
C ... COMPUTE THE COEFFICIENTS OF THE ORTHOGONAL POLYNOMIAL
C     BY THE BORDERING METHOD
         K = IDEG - 1
         IF ( MOD(K,2) .EQ. 0 ) THEN
            CALL CBORD (COEFF1, COEFF2, C, K, H)
         ELSE
            CALL CBORD (COEFF2, COEFF1, C, K, H)
```

```
         END IF
C
C ... COMPUTE THE ZEROS
         IF ( IDEG .EQ. 1 ) THEN
            ZEROS(1) = - COEFF1(0)
         ELSE IF ( IDEG .EQ. 2 ) THEN
C
C ... COMPUTE THE ZEROS OF THE SECOND DEGREE POLYNOMIAL
            CALL ROOT (COEFF2(2),COEFF2(1),COEFF2(0),
     *                  ZEROS(1), ZEROS(2))
         ELSE
C
C ... COMPUTE THE ZEROS OF P(X) BY BRENT'S ALGORITHM
            DO 300 II = 1, IDEG
               IF (II .EQ. 1) THEN
                  A = -1.0Q0
               ELSE
                  A = ZEROS(II-1)
                  ZEROS(II-1) = B
               END IF
               MAXFNTMP = MAXFN
               IF (II .EQ. IDEG) THEN
                  B = 1.0Q0
               ELSE
                  B = ZEROS(II)
               END IF
               IF ( MOD(K,2) .EQ. 0 ) THEN
                  CALL ZBREN (POL, ERRABS, ERRREL, A, B,
     *            MAXFNTMP, COEFF1, IDEG, IER)
               ELSE
                  CALL ZBREN (POL, ERRABS, ERRREL, A, B,
     *            MAXFNTMP, COEFF2, IDEG, IER)
               END IF
               IF (II .EQ. IDEG) ZEROS(II) = B
300         CONTINUE
         END IF
400   CONTINUE
C
C ... COMPUTE THE WEIGHTS
      IF ( MOD(K,2) .EQ. 0 ) THEN
         DO 500 II = 1, N
            WEIGHT(II) = H(1) / (POLD(ZEROS(II), COEFF1, N)*
     *                   POL(ZEROS(II), COEFF2, N-1))
500      CONTINUE
      ELSE
         DO 600 II = 1, N
```

```
            WEIGHT(II) = H(1) / (POLD(ZEROS(II), COEFF2, N)*
     *                   POL(ZEROS(II), COEFF1, N-1))
600      CONTINUE
      END IF
C
C ... COMPUTE THE INTEGRAL USING THE ZEROS AND THE WEIGHTS
      QUADINT = 0.0Q0
      DO 700  II = 1, N
         QUADINT = QUADINT + WEIGHT(II) * FUN(ZEROS(II))
700   CONTINUE
C
C ... COMPUTE THE ANALYTICAL VALUE OF THE INTEGRAL
      REALINT = RINT(ALPHA)
C
C ... PRINT THE REAL AND APPROXIMATE VALUES OF INTEGRAL
     .................................................
C ... REPEAT ONCE MORE THE INTERNAL LOOP UNTIL THE DESIRED
C     PRECISION WITH RESPECT TO THE REAL VALUE IS REACHED
      IF (QABS(QUADINT - REALINT)/REALINT .GT. EPS) THEN
         N    = N + 1
         IDEG = N
         GO TO 100
      END IF
      STOP
      END
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
            SUBROUTINE CBORD (VEC1, VEC2, MOM, K, H)
C PURPOSE: COMPUTE THE COEFFICIENTS OF THE ORTHOGONAL      +
C          POLYNOMIAL P   (x) BY THE BORDERING METHOD.     +
C                      k+1                                 +
C          THE SUBROUTINE STORE IN VEC1 THE NEW COLUMN     +
C          VECTOR OF THE TABLE UTILIZING THE OLD VECTOR    +
C          VEC1, THE VECTOR VEC2 AND THE VALUE OF H(1).    +
C INPUT:   MOM  MOMENTS VECTOR                             +
C          VEC2 VECTOR OF COEFFICIENTS OF P (x)            +
C          K    INTEGER CONSTANT          k                +
C INPUT/OUTPUT:                                            +
C          VEC1 IN INPUT IS THE P   (x) COEFFICIENT VECTOR +
C                               k-1                        +
C               IN OUTPUT IT CONTAINS THE COEFFICIENTS OF  +
C               P   (x)                                    +
C                k+1                                       +
C          H    2-DIMENSION VECTOR WHICH CONTAINS IN INPUT +
C               h    AND h  AND IN OUTPUT H(1) CONTAINS h  +
C                k-1      k                            k +
C EXTERNAL FUNCTIONS:                                      +
```

```
C          PINNER  COMPUTE THE INNER PRODUCT BETWEEN TWO   +
C                  VECTORS                                 +
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      IMPLICIT REAL*16 (A-H,O-Z)
      DIMENSION VEC1(-1:1), VEC2(-1:1), MOM(0:1), H(1)
C
C ... FIRST CALL OF THE SUBROUTINE
      IF (K.EQ.-2) THEN
          VEC1(K+1) = 0.0Q0
          VEC1(K+2) = 0.0Q0
          VEC2(K+1) = 0.0Q0
          VEC2(K+2) = 1.0Q0
          VEC2(K+3) = 0.0Q0
          H(1)      = 1.0Q0
          RETURN
      ELSE
C
C ... SET THE FIXED COEFFICIENTS FOR ALL THE POLYNOMIAL
          VEC1(K+1) = 1.0Q0
          VEC1(K+2) = 0.Q0
C
C ... COMPUTE THE TEMPORARY VARIABLE
          H(2)  = PINNER (MOM, VEC2, K, K)
          GAMMA = PINNER (MOM, VEC2, K, K+1)
          ALPHA = GAMMA + VEC2(K-1) * H(2)
          BREC  = -ALPHA / H(2)
          CREC  = H(2) / H(1)
C
C ... COMPUTE THE NEW VALUES OF THE COEFFICIENTS
          DO 100 I = 0, K
            VEC1(I) = VEC2(I-1)+BREC*VEC2(I)-CREC*VEC1(I)
100       CONTINUE
C
C ... SAVE THE H(2) VALUE FOR THE NEXT CALL
          H(1) = H(2)
      END IF
      RETURN
      END
```

BIBLIOGRAPHIE

[1] M. ABRAMOWITZ, I. A. STEGUN, *Handbook of Mathematical Functions*, Dover Publ., Inc., New York, 1965.

[2] R. P. BRENT, *An algorithm with guaranteed convergence for finding a zero of a function*, The Comput. J., 14(4) (1971) 422–425.

[3] C. BREZINSKI, *Computation of Padé approximants and continued fractions*, J. Comput. Appl. Math., 2(2) (1976) 113–123.

[4] C. BREZINSKI, *Padé-type Approximation and General Orthogonal Polynomials*, ISNM vol. 50, Birkhäuser-Verlag, Basel, 1980.

[5] C. BREZINSKI, *Bordering methods and progressive forms for sequence transformations*, Zastosow. Math., 20 (1990) 435–443.

[6] C. BREZINSKI, M. REDIVO ZAGLIA, *Extrapolation Methods. Theory and Practice*, North-Holland, Amsterdam, 1991.

[7] C. BREZINSKI, *CGM: a whole class of Lanczos-type solvers for linear systems*, soumis.

[8] C. BREZINSKI, M. REDIVO ZAGLIA, *A new presentation of orthogonal polynomials with application to their computation*, Numerical Algorithms, 1 (1991) 207–222.

[9] C. BREZINSKI, M. REDIVO ZAGLIA, *Treatment of near-breakdown in the CGS algorithm*, soumis.

[10] C. BREZINSKI, M. REDIVO ZAGLIA, H. SADOK, *A breakdown-free Lanczos type algorithm for solving linear systems*, Numer. Math., à paraître.

[11] C. BREZINSKI, M. REDIVO ZAGLIA, H. SADOK, *Avoiding breakdown and near-breakdown in Lanczos type algorithms*, Numerical Algorithms, 1 (1991) 261-284.

[12] C. BREZINSKI, M. REDIVO ZAGLIA, H. SADOK, *Addendum to "Avoiding breakdown and near-breakdown in Lanczos type algorithms"*, Numerical Algorithms, 2 (1992), sous presse.

[13] C. BREZINSKI, H. SADOK, *Avoiding breakdown in the CGS algorithm*, Numerical Algorithms, 1 (1991) 199-206.

[14] C. BREZINSKI, H. SADOK, *Lanczos type methods for systems of linear equations*, soumis.

[15] P. L. CHEBYSHEV, *Sur les fractions continues*, J. Math. Pures Appl., ser. II, 3 (1858) 289–323.

[16] P. J. DAVIS, P. RABINOWITZ, *Methods of Numerical Integration*, Academic Press, 1984.

[17] A. DRAUX, *Polynômes Orthogonaux Formels. Applications*, LNM 974, Springer-Verlag, Berlin, 1983.

[18] R. FLETCHER, *Conjugate gradient methods for indefinite systems*, in *Numerical Analysis*, G. A. Watson ed., LNM 506, Springer-Verlag, Berlin, 1976, pp. 73–89.

[19] V. N. FADDEEVA, *Computational methods of linear algebra*, Dover, New-York, 1959.

[20] W. GAUTSCHI, *Construction of Gauss-Christoffel quadrature formulas*, Math. Comp., 22 (1968) 251–270.

[21] W. GAUTSCHI, *On the construction of Gaussian quadrature rules from modified moments*, Math. Comp., 24 (1970) 245–260.

[22] W. GAUTSCHI, *On generating orthogonal polynomials*, SIAM J, Sci. Stat. Comput., (1982) 289–317.

[23] W. GAUTSCHI, *On the sensitivity of Orthogonal Polynomials to perturbation in the moments*, Numer. Math., 48 (1986) 369–382.

[24] A. GHIZZETTI, *Lezioni di Analisi Superiore*, Libreria Eredi V. Veschi, Roma, 1955-56.

[25] G. H. GOLUB, J. H. WELSH, *Calculation of Gaussian Quadrature Rules*, Math. Comput., 23 (1969) 221–230.

[26] G. H. GOLUB, M. H. GUTKNECHT, *Modified moments for indefinite weight functions*, Numer. Math., 57 (1990) 607–624.

[27] I. S. GRADSHTEYN, I. M. RYZHIK, *Table of Integrals, Series and Products*, Academic Press, 1965.

[28] M. H. GUTKNECHT, *A complete theory of the unsymmetric Lanczos process and related algorithms. Parts I, II* , SIAM J. Matrix Anal. Appl., à paraître.

[29] P. HENRICI, *Applied and Computational Complex Analysis*, vol. 1, Wiley, New-York, 1974.

[30] *IMSL Math/Library - FORTRAN Subroutines for Mathematical Applications*, IMSL User's Manual, ver. 1.1, Aug. 1989.

[31] M. MORANDI CECCHI, *L'integrazione numerica di una classe di integrali utili nei calcoli quantomeccanici*, Calcolo, 4, 3 (1967) 363–368.

[32] M. MORANDI CECCHI, M. REDIVO ZAGLIA, *A new recursive algorithm for a gaussian quadrature formula via orthogonal polynomials*, in *IMACS Transactions on Orthogonal Polynomials and Their Applications*, Editors C. Brezinski, L. Gori et al., Baltzer AG, Basel, 1991, pp. 353–358.

[33] M. MORANDI CECCHI, M. REDIVO ZAGLIA, *Computing the coefficients of recurrence formula for numerical integration by moments and modified moments*, soumis.

[34] H. RUTISHAUSER, *Der Quotienten-Differenzen-Algorithms*, Z. Angew. Math. Physik, 5 (1954) 233–251.

[35] R. A. SACK, A. F. DONOVAN, *An Algorithm for Gaussian Quadrature Given Modified Moments*, Numer. Math., 18 (1971/72) 465–478.

[36] G. SZEGÖ, *Orthogonal Polynomials*, American Mathematical Society, Providence, 1939.

[37] W. F. TRENCH, *An algorithm for the inversion of finite Hankel matrices*, SIAM J., 13 (1965) 1102–1107.

[38] J. C. WHEELER, *Modified moments and Gaussian quadrature*, Rocky Mountain J. Math., 4 (1974) 287–296.

[39] S. WOLFRAM, *Mathematica. A System for doing Mathematics by Computer*, Addison–Wesley, Redwood City, second edition, 1991.

Chapitre 3

SYSTÈMES D'ÉQUATIONS LINÉAIRES

La résolution des systèmes linéaires est l'un des problèmes les plus traités en analyse numérique. La méthode de bordage est l'une des méthodes qui permet de résoudre d'une façon récursive ce problème. Elle considère (à partir de la matrice de dimension 1 formée par le premier coefficient de la première équation) les differentes matrices construites en ajoutant une ligne et une colonne à la matrice précédente jusqu'à obtenir la matrice du système à résoudre. A chaque étape, la solution du nouveau système est obtenue en utilisant la solution du système précédent. Cette méthode a des limites dues au fait que si, à une étape, une certaine quantité est égale à zéro, la méthode itérative doit s'arrêter.

Dans [4] nous avons modifié cette méthode en proposant une nouvelle méthode qui a été appelée *Block Bordering Method* qui permet de continuer le procédé en ajoutant plusieurs lignes et plusieurs colonnes à la fois jusqu'au moment où la quantité devient différente de zéro (ou inversible). Evidemment cette méthode peut aussi s'appliquer quand la quantité est différente de zéro mais très petite et donc son utilisation peut apporter une amélioration aux erreurs dues à l'arithmétique de l'ordinateur.

Cette méthode présente un inconvénient quand elle est utilisée dans la résolution récursive de systèmes (non singuliers) et que l'on a besoin de toutes les solutions intermédiaires donc aussi des solutions des systèmes qu'on avait sautés pour améliorer les solutions suivantes. Dans ce cas nous avons proposée [5] une méthode appelée *Reverse Bordering Method* qui permet, après avoir sauté, de revenir en arrière pour calculer les solutions des étapes qu'on avait sautées.

Ces méthodes s'appliquent à plusieurs domaines comme le calcul récursif des polynômes orthogonaux, l'approximation de Padé et les formes progressives des algorithmes d'extrapolation.

3.1 La méthode de bordage

Pour commencer nous allons rappeler la méthode bien connue de bordage, basée sur la méthode de Sherman et Morrison [10] et qui peut se trouver, par exemple, dans le livre de Faddeeva [7].

Soit A_k une matrice carrée régulière de dimension k et d_k un vecteur de dimension k. Soit z_k la solution du système

$$A_k z_k = d_k.$$

Soit u_k un vecteur colonne de dimension k, v_k un vecteur ligne de dimension k et a_k une constante. On considère la matrice bordée A_{k+1} de dimension $k+1$ donnée par

$$A_{k+1} = \begin{pmatrix} A_k & u_k \\ v_k & a_k \end{pmatrix}. \tag{3.1}$$

Sa matrice inverse est donnée par

$$A_{k+1}^{-1} = \begin{pmatrix} A_k^{-1} + A_k^{-1}u_k v_k A_k^{-1}/\beta_k & -A_k^{-1}u_k/\beta_k \\ -v_k A_k^{-1}/\beta_k & 1/\beta_k \end{pmatrix}$$

avec $\beta_k = a_k - v_k A_k^{-1} u_k$.

Soit f_k une constante et z_{k+1} la solution du système bordé

$$A_{k+1} z_{k+1} = d_{k+1} = \begin{pmatrix} d_k \\ f_k \end{pmatrix}.$$

On a

$$z_{k+1} = \begin{pmatrix} z_k \\ 0 \end{pmatrix} + \frac{f_k - v_k z_k}{\beta_k} \cdot \begin{pmatrix} -A_k^{-1}u_k \\ 1 \end{pmatrix} \tag{3.2}$$

et donc on a obtenu la solution du système bordé en fonction de la solution du système précédent.

L'on peut éviter le calcul et la mémorisation de la matrice A_k^{-1} si l'on pose $q_k = -A_k^{-1}u_k$ et ensuite que l'on calcule cette quantité avec la même méthode de bordage. Comme cela on a une variante de la méthode qui a besoin seulement de la matrice A_k.

Donc soit $q_k^{(i)}$ la solution du système

$$A_i q_k^{(i)} = -u_k^{(i)}$$

où $u_k^{(i)}$ est le vecteur formé par les i premières composantes de u_k et évidemment $u_i^{(i)} = u_i$ et $q_i^{(i)} = q_i$ pour tous les i. Soit A_i la matrice de dimension i formée par les i premières lignes et colonnes de A_k.

Puisque A_1 est une constante, on aura

$$\begin{aligned} q_k^{(1)} &= -\frac{u_k^{(1)}}{A_1} \\ q_k^{(i+1)} &= \begin{pmatrix} q_k^{(i)} \\ 0 \end{pmatrix} - \frac{u_{k,i+1} + v_i q_k^{(i)}}{a_i + v_i q_i^{(i)}} \cdot \begin{pmatrix} q_i^{(i)} \\ 1 \end{pmatrix}, \quad i = 1, \ldots, k-1 \end{aligned} \tag{3.3}$$

où $u_{k,i+1}$ est la $(i+1)$-ème composante de u_k.

Alors

$$q_k^{(k)} = q_k = -A_k^{-1} u_k$$

et cette quantité est celle qui nous sert dans (3.2) pour le calcul de z_{k+1} sans avoir besoin de A_k^{-1}.

Sur cette méthode et sur ses applications aux formes progressives des algorithmes d'extrapolation l'on peut voir [2]. Dans [3] nous avons donné la subroutine BORDER qui utilise cette variante de la méthode de bordage.

3.2 Bordage par blocs

La méthode de bordage ne peut évidemment s'appliquer que si $\beta_k \neq 0$ pour tout k. Quand cette condition n'est pas verifiée nous avons trouvé [4] une méthode appelée ***block bordering method*** qui permet de sauter les singularités des étapes intermédiaires et arriver quand même à résoudre le système.

Maintenant on suppose que toutes les quantités de (3.1) sont des matrices avec les dimensions suivantes

$$\begin{array}{cc} A_k & n_k \times n_k \\ u_k & n_k \times p_k \\ v_k & p_k \times n_k \\ a_k & p_k \times p_k \\ A_{k+1} & n_{k+1} \times n_{k+1} \end{array}$$

avec $n_{k+1} = n_k + p_k$.

Posons

$$\beta_k = a_k - v_k A_k^{-1} u_k.$$

La matrice inverse deviendra

$$A_{k+1}^{-1} = \begin{pmatrix} A_k^{-1} + A_k^{-1} u_k \beta_k^{-1} v_k A_k^{-1} & -A_k^{-1} u_k \beta_k^{-1} \\ -\beta_k^{-1} v_k A_k^{-1} & \beta_k^{-1} \end{pmatrix}.$$

Si maintenant f_k est un vecteur avec p_k composantes on aura

$$z_{k+1} = \begin{pmatrix} z_k \\ 0 \end{pmatrix} + \begin{pmatrix} -A_k^{-1} u_k \\ I_k \end{pmatrix} \beta_k^{-1} \left(f_k - v_k z_k\right)$$

où I_k est la matrice identité de dimension p_k.

Toujours dans [3] l'on peut trouver la subroutine BLBORD qui réalise cette méthode.

Comme on l'a fait pour la méthode de bordage, ici aussi l'on peut éviter de calculer et mémoriser l'inverse A_k^{-1} si l'on pose $q_k = -A_k^{-1} u_k$ (de dimensions $n_k \times p_k$) et que l'on calcule cette matrice récursivement avec la méthode de bordage. Soit donc $u_k^{(i)}$ la matrice $n_i \times p_k$ qui contient les n_i premières lignes de u_k pour $i \leq k$, $n_i \leq n_k$ (avec $u_i^{(i)} = u_i$) et soit $q_k^{(i)}$ la matrice de dimension $n_i \times p_k$ qui satisfait aux conditions $A_i q_k^{(i)} = -u_k^{(i)}$ pour $i \leq k$ (avec $q_i^{(i)} = q_i$).

Si l'on pose

$$q_k^{(1)} = -A_1^{-1} u_k^{(1)}$$

on obtient

$$q_k^{(i+1)} = \begin{pmatrix} q_k^{(i)} \\ 0 \end{pmatrix} - \begin{pmatrix} q_i^{(i)} \\ I_i \end{pmatrix} \beta_i^{-1} \left(u_{k,i+1} + v_i q_k^{(i)}\right), \quad i = 1, \ldots, k-1$$

avec $\beta_i = a_i + v_i q_i^{(i)}$ et avec $u_{k,i+1}$ étant la matrice formée par les lignes $n_i + 1, \ldots, n_i + p_i$ de u_k.

Quand dans une matrice intermédiaire il y a β_k égal à zéro, on aurait pu aussi utiliser le pivotage et donc si $\beta_k = 0$ remplacer la dernière ligne par une autre ligne pour laquelle $\beta_k \neq 0$. Mais le pivotage peut être adopté quand on a besoin uniquement de la solution finale du système, tandis qu'il n'est pas adapté quand on a besoin des solutions intermédiaires. De là l'intérêt de la méthode de bordage par blocs.

3.3 La méthode de bordage inverse

Si l'on désire augmenter la stabilité du procédé et donc améliorer les solutions, on peut appliquer la méthode de bordage par blocs même quand la matrice β_k n'est pas singulière mais presque singulière. Donc dans ce cas on saute des systèmes intermédiaires qui sont réguliers. On peut donc avoir le problème de connaître aussi les solutions qu'on a sautées. C'est, par exemple, le cas pour le calcul des polynômes orthogonaux (voir par exemple ce que nous avons fait dans [9]) ou dans l'approximation de Padé ou dans les formes progressives des procédés d'extrapolation [3] (pag. 28 et suivantes).

L'idée que nous avons eue [5] a été de sauter quand la matrice β_k est presque singulière, puis de revenir en arrière avec un autre algorithme pour calculer les solutions qu'on avait sautées.

Donc, après avoir sauté, nous connaissons la matrice A_{k+1}^{-1} du système qu'on vient de résoudre et l'on supprime la dernière ligne et la dernière colonne de la matrice A_{k+1} pour calculer la solution z_k à partir de la solution z_{k+1}. On peut aussi revenir en arrière en supprimant les p_k dernières lignes et colonnes où, naturellement, ce p_k est plus petit que celui utilisé à l'aller dans le bordage par blocs.

Nous avons d'abord la nécessité de trouver A_k^{-1} à partir de A_{k+1}^{-1}.

Supposons que A_{k+1}^{-1}, de dimension n_{k+1} soit de la forme

$$A_{k+1}^{-1} = \begin{array}{c} \begin{matrix} n_k & p_k \end{matrix} \\ \begin{pmatrix} A'_k & u'_k \\ v'_k & a'_k \end{pmatrix} \end{array} \begin{matrix} n_k \\ p_k \end{matrix}.$$

D'après le bordage par blocs on sait que

$$\begin{aligned} A'_k &= A_k^{-1} + A_k^{-1} u_k \beta_k^{-1} v_k A_k^{-1} \\ u'_k &= -A_k^{-1} u_k \beta_k^{-1} \qquad v'_k = -\beta_k^{-1} v_k A_k^{-1} \\ a'_k &= \beta_k^{-1} \qquad \beta_k = a_k - v_k A_k^{-1} u_k. \end{aligned} \tag{3.4}$$

On a donc

$$a_k'^{-1} = \beta_k$$

alors

$$u'_k \, a_k'^{-1} = -A_k^{-1} u_k$$

et finalement

$$u'_k\, a'^{-1}_k\, v'_k = A_k^{-1} u_k \beta_k^{-1} v_k A_k^{-1}.$$

Si l'on remplace cette expression dans (3.4) on aura

$$A_k^{-1} \;=\; A'_k - u'_k\, a'^{-1}_k\, v'_k. \tag{3.5}$$

Cette formule a été déjà donnée par Duncan dans l'année 1944 [6]. C'est le complément de Schur et donc on a aussi

$$\begin{aligned} \det A_k^{-1} &= \det A_{k+1}^{-1} \big/ \det a'_k \\ \det A_{k+1} &= \det A_k \cdot \det \beta_k. \end{aligned}$$

La formule (3.5) peut aussi s'obtenir à partir de la formule de Sherman-Morrison (voir Hager [8]), pour laquelle on a

$$\begin{aligned} & A'_k = \left(A_k - u_k\, a_k^{-1}\, v_k\right)^{-1} \\ \text{et} \quad & A_k = \left(A'_k - u'_k\, a'^{-1}_k\, v'_k\right)^{-1} = A'^{-1}_k + A'^{-1}_k\, u'_k\, \beta'^{-1}_k\, v'_k\, A'^{-1}_k \\ \text{avec} \quad & \beta'_k = a'_k - v'_k\, A'^{-1}_k\, u'_k. \end{aligned}$$

Et on a aussi

$$\begin{aligned} A_k &= A'^{-1}_k + u_k\, a_k^{-1}\, v_k = \left(A'_k - u'_k\, a'^{-1}_k\, v'_k\right)^{-1} \\ A'_k &= A_k^{-1} + u'_k\, a'^{-1}_k\, v'_k = \left(A_k - u_k\, a_k^{-1}\, v_k\right)^{-1}. \end{aligned}$$

Maintenant nous allons calculer la solution z_k du système

$$A_k z_k = d_k$$

à partir de la solution z_{k+1} du système bordé

$$A_{k+1} z_{k+1} = d_{k+1} = \begin{pmatrix} d_k \\ f_k \end{pmatrix} \begin{matrix} n_k \\ p_k \end{matrix}.$$

D'après (3.2) on aura

$$\begin{aligned} z_{k+1} = \begin{pmatrix} z'_k \\ c_k \end{pmatrix} &= \begin{pmatrix} z_k \\ 0 \end{pmatrix} + \begin{pmatrix} -A_k^{-1} u_k \\ I_k \end{pmatrix} \beta_k^{-1} \left(f_k - v_k z_k\right) \\ &= \begin{pmatrix} z_k \\ 0 \end{pmatrix} + \begin{pmatrix} u'_k \\ a'_k \end{pmatrix} \left(f_k - v_k z_k\right). \end{aligned}$$

Donc

$$c_k = a'_k f_k - a'_k v_k z_k$$

c'est à dire

$$a'^{-1}_k c_k = f_k - v_k z_k \quad \text{or} \quad v_k z_k = f_k - a'^{-1}_k c_k$$

et

$$z'_k = z_k + u'_k\,(f_k - v_k z_k) = z_k + u'_k\,a'^{-1}_k\,c_k$$

et finalement

$$z_k = z'_k - u'_k\,a'^{-1}_k\,c_k.$$

Pour calculer z_k on peut aussi partir de la formule (3.5), et l'on a

$$A_k^{-1} d_k = A'_k d_k - u'_k\,a'^{-1}_k\,v'_k\,d_k.$$

Mais

$$\begin{pmatrix} z'_k \\ c_k \end{pmatrix} = \begin{pmatrix} A'_k & u'_k \\ v'_k & a'_k \end{pmatrix} \begin{pmatrix} d_k \\ f_k \end{pmatrix} = \begin{pmatrix} A'_k d_k + u'_k f_k \\ v'_k d_k + a'_k f_k \end{pmatrix}$$

et donc

$$A'_k d_k = z'_k - u'_k f_k \quad \text{and} \quad v'_k d_k = c_k - a'_k f_k.$$

Le z_k sera donc

$$\begin{aligned} z_k = A'_k d_k &= z'_k - u'_k f_k - u'_k a'^{-1}_k\,(c_k - a'_k f_k) \\ &= z'_k - u'_k f_k - u'_k\,a'^{-1}_k\,c_k + u'_k\,a'^{-1}_k\,a'_k\,f_k \\ &= z'_k - u'_k a'^{-1}_k c_k. \end{aligned}$$

3.4 Applications

Exemple 1

On considère le système

$$\begin{pmatrix} 1 & 1 & 1 & 1 & -1 & 0 & -1 \\ 1 & 1 & 2 & 0 & 1 & 1 & -1 \\ 1 & 1 & -1 & 0 & 2 & -2 & 0 \\ -1 & 1 & 2 & 0 & -1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{pmatrix} = \begin{pmatrix} -2 \\ 13 \\ -2 \\ 22 \\ 25 \\ 18 \\ 9 \end{pmatrix}$$

dans lequel les systèmes intermédiaires de dimension 2, 3 et 6 sont singuliers. Pour les rendre presque singuliers on ajoute une perturbation ε_1 à a_{11} et à a_{55} et, pour avoir la même solution finale, on ajoute aussi ε_1 à la première composante du second membre et $5 \cdot \varepsilon_1$ à sa cinquième composante.

Soit ϵ le seuil pour le saut dans le bordage par blocs et ε_2 le seuil pour le saut dans le bordage inverse (qui doit être plus petit pour ne pas refaire exactement le même saut qu'à l'aller).

On obtient les résultats suivants

Solutions avec la méthode de bordage

$\varepsilon_1 = 1.0 \cdot 10^{-14}$

n						
1	$-0.2000 \cdot 10^{1}$					
2	$-0.1501 \cdot 10^{16}$	$0.1501 \cdot 10^{16}$				
3	$-0.1001 \cdot 10^{16}$	$0.1001 \cdot 10^{16}$	$0.5000 \cdot 10^{1}$			
4	$-0.4500 \cdot 10^{1}$	$0.7500 \cdot 10^{1}$	$0.4992 \cdot 10^{1}$	$-0.1001 \cdot 10^{2}$		
5	$-0.2950 \cdot 10^{2}$	$-0.9297 \cdot 10^{1}$	$0.1333 \cdot 10^{2}$	$0.4833 \cdot 10^{2}$	$0.2500 \cdot 10^{2}$	
6	$-0.7006 \cdot 10^{15}$	$-0.1171 \cdot 10^{16}$	$0.9348 \cdot 10^{15}$	$0.1635 \cdot 10^{16}$	$0.7006 \cdot 10^{15}$	$-0.7006 \cdot 10^{15}$
7	0.9222	$0.1859 \cdot 10^{1}$	$0.3006 \cdot 10^{1}$	$0.3570 \cdot 10^{1}$	$0.5078 \cdot 10^{1}$	$0.5922 \cdot 10^{1}$ $0.7000 \cdot 10^{1}$

Solutions avec le bordage par blocs

$\varepsilon = 1.0 \cdot 10^{-14}$ $\varepsilon_1 = 1.0 \cdot 10^{-14}$

n							
1	$-0.2000 \cdot 10^{1}$						
2	———	———					
3	———	———	———				
4	$-0.4500 \cdot 10^{1}$	$0.7500 \cdot 10^{1}$	$0.5000 \cdot 10^{1}$	$-0.1000 \cdot 10^{2}$			
5	$-0.2950 \cdot 10^{2}$	$-0.9167 \cdot 10^{1}$	$0.1333 \cdot 10^{2}$	$0.4833 \cdot 10^{2}$	$0.2500 \cdot 10^{2}$		
6	———	———	———	———	———	———	
7	$0.1000 \cdot 10^{1}$	$0.2000 \cdot 10^{1}$	$0.3000 \cdot 10^{1}$	$0.4000 \cdot 10^{1}$	$0.5000 \cdot 10^{1}$	$0.6000 \cdot 10^{1}$	$0.7000 \cdot 10^{1}$

Solutions avec le bordage par blocs et le bordage inverse

$\varepsilon = 1.0 \cdot 10^{-14}$ $\varepsilon_1 = 1.0 \cdot 10^{-14}$ $\varepsilon_2 = 1.0 \cdot 10^{-20}$

n							
1	$-0.2000 \cdot 10^{1}$						
2	$-0.1501 \cdot 10^{16}$	$0.1501 \cdot 10^{16}$					
3	$-0.1001 \cdot 10^{16}$	$0.1001 \cdot 10^{16}$	$0.5000 \cdot 10^{1}$				
4	$-0.4500 \cdot 10^{1}$	$0.7500 \cdot 10^{1}$	$0.5000 \cdot 10^{1}$	$-0.1000 \cdot 10^{2}$			
5	$-0.2950 \cdot 10^{2}$	$-0.9167 \cdot 10^{1}$	$0.1333 \cdot 10^{2}$	$0.4833 \cdot 10^{2}$	$0.2500 \cdot 10^{2}$		
6	$-0.7006 \cdot 10^{15}$	$-0.1168 \cdot 10^{16}$	$0.9341 \cdot 10^{15}$	$0.1635 \cdot 10^{16}$	$0.7006 \cdot 10^{15}$	$-0.7006 \cdot 10^{15}$	
7	$0.1000 \cdot 10^{1}$	$0.2000 \cdot 10^{1}$	$0.3000 \cdot 10^{1}$	$0.4000 \cdot 10^{1}$	$0.5000 \cdot 10^{1}$	$0.6000 \cdot 10^{1}$	$0.7000 \cdot 10^{1}$

Exemple 2

Maintenant nous allons donner un exemple qui concerne l'ε-algorithme [3].
Les $\varepsilon_{2k}^{(0)}$ peuvent être interpretés (voir [1]) comme

$$\varepsilon_{2k}^{(0)} = 1/\sum_{i=0}^{k} a_i$$

où les a_i sont solution du système

$$\begin{cases} a_0 S_0 + a_1 S_1 + \cdots + a_k S_k &= 1 \\ a_0 S_1 + a_1 S_2 + \cdots + a_k S_{k+1} &= 1 \\ \vdots & \vdots \\ a_0 S_k + a_1 S_{k+1} + \cdots + a_k S_{2k} &= 1. \end{cases}$$

On applique l'ε-algorithme aux sommes partielles de la série de

$$f(x) = \frac{1 + b_1 x + \ldots + b_{m-1} x^{m-1} + x^m}{1 + x^m}.$$

D'après la théorie de l'ε-algorithme et sa liaison avec les approximants de Padé on sait que

$$\varepsilon_{2m}^{(0)} = f(x).$$

Dans nos exemples les systèmes de dimension 3, 4, 5 et 6 sont presque singuliers et dans les tableaux suivants nous allons donner différentes valeurs aux quantités $\varepsilon, \varepsilon_1, \varepsilon_2, m, x, b_i$. Le R à coté des valeurs de la deuxième colonne signifie que les valeurs correspondantes de la colonne précédente, ont été obtenues par la méthode de bordage inverse.

$\varepsilon = 10^{-6}, \varepsilon_1 = 0.25, \varepsilon_2 = 10^{-30}, m = 10, x = 2, b_i = i \cdot \varepsilon_1, f(x) = 2.998536585365854$

k	Méthode de bordage	Bordage par blocs et inverse	
0	1.000000000000000	1.000000000000000	
1	0.833333333333333	0.833333333333333	
2	1.500000000000007	1.500000000000007	
3	1.500000000000006	1.500000000000007	R
4	1.500000000000004	1.499999999999987	R
5	1.500000000000004	1.499999999999984	R
6	1.500000000000006	1.500000000000006	R
7	1.500000000000007	1.500000000000006	
8	1.057370161706715	1.061205132114060	
9	5.442384375014805	5.530461077969034	
10	2.965829933964836	2.998536585365856	

On voit bien que, avec la méthode de bordage, on obtient seulement 2 chiffres exacts tandis qu'avec la méthode de bordage par blocs et la méthode de bordage inverse on a 15 chiffres exacts.

$\varepsilon = 10^{-3}, \varepsilon_1 = 0.1, \varepsilon_2 = 10^{-30}, m = 10, x = 1, b_i = \varepsilon_1 / i, f(x) = 1.141448412698413$

k	Méthode de bordage	Bordage par blocs et inverse	
0	1.000000000000000	1.000000000000000	
1	1.200000000000000	1.200000000000000	
2	1.299999999999898	1.299999999999898	
3	1.366666666630687	1.366666666630683	R
4	1.416666663166007	1.416666665980086	R
5	1.416669182535733	1.416669185348150	R
6	1.370133070676631	1.370133070822927	R
7	1.363779438853716	1.363779438821982	
8	1.299241827763746	1.299241827748625	
9	1.221626694740750	1.221626694709201	
10	1.141448412733146	1.141448412698013	

On a donc amélioré la précision.

3.5 Logiciels

Pour toutes les méthodes que nous avons présentées nous avons codé des logiciels en FORTRAN 77. Les sous-programmes ont été utilisés dans plusieurs applications et exemples que nous avons présentés dans les autres chapitres.

Certains d'entre eux se trouvent dans la diskette du livre que nous avons écrit [6] et où l'on peut trouver aussi un programme principal d'appel. Ils sont

- **BORDER** pour la méthode de bordage avec la variante (3.3).
  Les sous-programmes et les fonctions utilisés sont QBORD, INNERC.

- **BLBORD** pour la méthode de bordage par blocs.
  Les sous-programmes et les fonctions utilisés sont BETABOR, GAUSS, INVERS, RCPROD, INNERC.

Pour la méthode de bordage inverse nous avons modifié le sous-programme BLBORD en insérant un appel à un nouveau sous-programme que nous avons appelé REVBORD qui est utilisé dès qu'il y a un saut. Dans ce dernier cas, REVBORD revient en arrière pour calculer les solutions intérmédiaires sautées et après les avoir calculées, l'algorithme de bordage par blocs continue en avant, à partir de la solution qu'il avait obtenu après le saut.

BIBLIOGRAPHIE

[1] C. BREZINSKI, *Résultats sur les procédés de sommation et sur l'ε-algorithme*, RIRO, R3 (1970) 147–153.

[2] C. BREZINSKI, *Bordering methods and progressive forms for sequence transformations*, Zastosow. Math., 20 (1990) 435–443.

[3] C. BREZINSKI, M. REDIVO ZAGLIA, *Extrapolation Methods. Theory and Practice*, North-Holland, Amsterdam, 1991.

[4] C. BREZINSKI, M. REDIVO ZAGLIA, H. SADOK, *A breakdown-free Lanczos type algorithm for solving linear systems*, Numer. Math., à paraître.

[5] C. BREZINSKI, M. MORANDI CECCHI, M. REDIVO ZAGLIA, *The Reverse Bordering Method*, soumis.

[6] W. J. DUNCAN, *Some devices for the solution of large sets of simultaneous linear equations*, Philos. Mag. Ser. 7, 35 (1944) 660–670.

[7] V. N. FADDEEVA, *Computational Methods of Linear Algebra*, Dover, New York, 1959.

[8] W. W. HAGER, *Updating the inverse of a matrix*, SIAM Rev., 31 (1989) 221–239.

[9] M. MORANDI CECCHI, M. REDIVO ZAGLIA, *A new recursive algorithm for a Gaussian quadrature formula via orthogonal polynomials*, in "Orthogonal Polynomials and their Applications", C. Brezinski et al. eds., Baltzer, Basel, 1991, pp. 353–358.

[10] J. SHERMAN, W. J. MORRISON, *Adjustement of an inverse matrix corresponding to changes in the elements of a given column or a given row of the original matrix*, Annals of Math. Stat., 20 (1949) 621.

Chapitre 4

MÉTHODES DE TYPE LANCZOS

En 1950 Lanczos [26] a présenté une méthode pour la tridiagonalisation d'une matrice et cette méthode peut aussi s'appliquer à la résolution d'un système d'équations linéaires en donnant une méthode iterative qui fournit (en théorie, c'est à dire s'il n'y avait pas d'erreurs dues à l'arithmétique de l'ordinateur) la solution exacte x en n iterations au plus où n est la dimension du système à résoudre [27].

Soit à résoudre le système d'équations linéaires

$$Ax = b$$

où $A \in \mathbb{C}^{n\times n}, b \in \mathbb{C}^n$ et $x \in \mathbb{C}^n$. La méthode de Lanczos consiste à construire une suite de vecteurs (x_k) de la façon suivante

- choisir deux vecteurs arbitraires x_0 et y dans $\mathbb{C}^n$
- calculer $r_0 = Ax_0 - b$
- déterminer x_k tel que
$$x_k - x_0 \in E_k = \text{span}(r_0, Ar_0, \ldots, A^{k-1}r_0)$$
$$r_k = b - Ax_k \perp F_k = \text{span}(y, A^*y, \ldots, A^{*k-1}y).$$

où A^* est la transposée conjuguée de A.

Ces deux conditions déterminent complétement le vecteur x_k. En effet $x_k - x_0$ peut donc s'écrire

$$x_k - x_0 = a_1 r_0 + \cdots + a_k A^{k-1} r_0$$

et encore, en multipliant pour A, en ajoutant et en soustrayant b, on obtient

$$r_k = r_0 + a_1 A r_0 + \cdots + a_k A^k r_0. \tag{4.1}$$

Les conditions d'orthogonalité donnent

$$\left(A^{*i} y, r_k\right) = 0 \qquad \text{pour } i = 0, \ldots, k-1 \tag{4.2}$$

et on a ainsi un système de k équations linéaires à k inconnues $a_1, \ldots, a_k$. Ce système est singulier si $r_0, Ar_0, \ldots, A^{k-1}r_0$ ou $y, A^*y, \ldots, A^{*k-1}$ sont linéairement dépendants.

Dans la pratique, le calcul est fait récursivement sans inverser la matrice A.

La méthode du gradient conjugué de Hestenes et Stiefel [22] est la plus connue et elle s'applique lorsque la matrice A est symétrique définie positive.

Dans cette méthode l'on pose

$$x_0 = 0, \quad r_0 = -b, \quad y = z_0 = -r_0$$

et après on fait les itérations suivantes

$$\begin{aligned}
\lambda_k &= -\frac{(r_k, r_k)}{(r_{k-1}, r_{k-1})}, \\
z_k &= -r_k + \lambda_k z_{k-1}, \\
\mu_k &= -\frac{(z_k, r_k)}{(z_k, Az_k)}, \\
x_{k+1} &= x_k + \mu_k z_k, \\
r_{k+1} &= r_k + \mu_k A z_k
\end{aligned}$$

où $(\cdot, \cdot)$ est le produit scalaire dans $\mathbb{C}^n$.

On voit que, dans cet algorithme, apparaissent des constantes dont l'expression est un rapport de deux produits scalaires. On remarquera que, puisque la matrice A est symétrique définie positive, aucun des produits scalaires qui se trouvent dans un dénominateur ne peut être nul.

Lorsque la matrice A du système est quelconque, la méthode du gradient conjugué a été généralisée par Fletcher [18] qui a proposé une méthode appelée gradient bi-conjugué. L'algorithme de cette méthode est le suivant

$$\begin{aligned}
x_0 = 0, \qquad & r_0 = \bar{r}_0 = -b, \quad y = z_0 = \bar{z}_0 = b \\
\lambda_k &= -\frac{(\bar{r}_k, r_k)}{(\bar{r}_{k-1}, r_{k-1})}, \\
z_k &= -r_k + \lambda_k z_{k-1}, \\
\bar{z}_k &= -\bar{r}_k + \lambda_k \bar{z}_{k-1}, \\
\mu_k &= -\frac{(z_k, \bar{r}_k)}{(\bar{z}_k, Az_k)}, \\
x_{k+1} &= x_k + \mu_k z_k, \\
r_{k+1} &= r_k + \mu_k A z_k. \\
\bar{r}_{k+1} &= \bar{r}_k + \mu_k A^{\mathrm{T}} \bar{z}_k.
\end{aligned}$$

Dans cette méthode on voit que certains coefficients sont encore donnés sous forme de rapports de produits scalaires. Cependant, pour une matrice A quelconque, il se peut que certains produits scalaires dans un dénominateur soient nuls. Il y a alors un *breakdown* dans l'algorithme qui doit être stoppé. De même si un produit scalaire dans un dénominateur est voisin de zéro il y a un *near-breakdown* qui provoque une propagation importante d'erreurs numériques car, bien souvent, une quantité voisine de zéro provient

de la différence de deux nombres voisins et elle est donc entachée d'une importante erreur de cancellation.

Dans [10, 11, 12] nous avons complétement résolu les problèmes du *breakdown* et du *near-breakdown* en utilisant, comme nous allons maintenant l'expliquer, la théorie des polynômes orthogonaux formels.

La méthode du gradient biconjugué est issue de la méthode de tridiagonalisation de Lanczos qui fait appel à une relation de récurrence à trois termes. Une telle relation de récurrence fait immédiatement penser aux polynômes orthogonaux formels qui satisfont également une telle relation. En fait, dans son article original, Lanczos [26] avait lui même établi la connection entre les polynômes orthogonaux et sa méthode de tridiagonalisation. Bien qu'une très abondante littérature sur le sujet ait parue (voir [20] pour une liste et une analyse), les chercheurs travaillant sur ces questions ont rapidement laissé de côté les polynômes orthogonaux pour ne plus utiliser que de pures techniques d'algèbre linéaire. La connection avec les polynômes orthogonaux formels, telle qu'elle a été exposée de nouveau dans [2], est la base de la résolution des problèmes de *breakdown* et de *near-breakdown*.

4.1 Breakdown dans Lanczos

Soit à résoudre dans $\mathbb{C}^n$ le système d'équations linéaires $Ax = b$.

Si nous posons

$$P_k(\xi) = 1 + a_1\xi + \ldots + a_k\xi^k$$

alors, d'après (4.1), l'on a

$$r_k = P_k(A)r_0.$$

De plus, si nous posons

$$c_i = (y, A^i r_0), \qquad i = 0, 1, \ldots$$

et si nous définissons la forme linéaire c sur l'espace vectoriel des polynômes par

$$c(\xi^i) = c_i, \qquad i = 0, 1, \ldots$$

alors les relations d'orthogonalité (4.2) s'écrivent

$$c(\xi^i P_k) = 0 \quad \text{pour } i = 0, \ldots, k-1.$$

Ces relations montrent que P_k est le polynôme de degré k au plus appartenant à la famille de polynômes orthogonaux formels par rapport à la fonctionnelle linéaire c [2]. Un tel polynôme orthogonal n'est défini qu'à une constante multiplicative près qui, dans notre cas, a été choisie de sorte que

$$P_k(0) = 1.$$

Les relations d'orthogonalité précédentes donnent le système

$$\begin{pmatrix} 1 & 0 & \cdots & 0 \\ c_0 & c_1 & \cdots & c_k \\ \vdots & \vdots & & \vdots \\ c_{k-1} & c_k & \cdots & c_{2k-1} \end{pmatrix} \begin{pmatrix} 1 \\ a_1 \\ \vdots \\ a_k \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

C'est à dire que l'on peut calculer P_k par le rapport de déterminants suivant

$$P_k(\xi) = \begin{vmatrix} 1 & \xi & \cdots & \xi^k \\ c_0 & c_1 & \cdots & c_k \\ \vdots & \vdots & & \vdots \\ c_{k-1} & c_k & \cdots & c_{2k-1} \end{vmatrix} \Big/ \begin{vmatrix} c_1 & c_2 & \cdots & c_k \\ c_2 & c_3 & \cdots & c_{k+1} \\ \vdots & \vdots & & \vdots \\ c_k & c_{k+1} & \cdots & c_{2k-1} \end{vmatrix},$$

et l'on a ainsi

$$x_k - x_0 = \begin{vmatrix} 0 & r_0 & \cdots & A^{k-1}r_0 \\ c_0 & c_1 & \cdots & c_k \\ \vdots & \vdots & & \vdots \\ c_{k-1} & c_k & \cdots & c_{2k-1} \end{vmatrix} \Big/ \begin{vmatrix} c_1 & c_2 & \cdots & c_k \\ c_2 & c_3 & \cdots & c_{k+1} \\ \vdots & \vdots & & \vdots \\ c_k & c_{k+1} & \cdots & c_{2k-1} \end{vmatrix},$$

et

$$r_k = \begin{vmatrix} r_0 & Ar_0 & \cdots & A^k r_0 \\ c_0 & c_1 & \cdots & c_k \\ \vdots & \vdots & & \vdots \\ c_{k-1} & c_k & \cdots & c_{2k-1} \end{vmatrix} \Big/ \begin{vmatrix} c_1 & c_2 & \cdots & c_k \\ c_2 & c_3 & \cdots & c_{k+1} \\ \vdots & \vdots & & \vdots \\ c_k & c_{k+1} & \cdots & c_{2k-1} \end{vmatrix},$$

où les déterminants des numérateurs sont les vecteurs obtenus avec les règles classiques pour développer un déterminant selon sa première ligne.

A cause donc de la normalisation $P_k(0) = 1$, ce polynôme existe et il est unique si et seulement si le déterminant de Hankel

$$H_k^{(1)} = \begin{vmatrix} c_1 & c_2 & \cdots & c_k \\ c_2 & c_3 & \cdots & c_{k+1} \\ \vdots & \vdots & & \vdots \\ c_k & c_{k+1} & \cdots & c_{2k-1} \end{vmatrix}$$

est différent de zéro.

Il est possible de calculer récursivement ces polynômes P_k de différentes façons:

- par leur relation de récurrence à trois termes
- en passant par les polynômes unitaires $P_k^{(1)}$ orthogonaux par rapport à la forme linéaire $c^{(1)}$ définie par $c^{(1)}(\xi^i) = c(\xi^{i+1}) = c_{i+1}$

- en passant par des polynômes proportionnels aux polynômes $P_k^{(1)}$
- etc.

Comme il a été montré dans [14], ces différentes procédures conduisent aux différentes méthodes de type Lanczos connues sous les noms de: gradient conjugué, Orthores, Orthodir, Orthomin, Biores, Biodir, gradient biconjugué, ... Toutes ces variantes possèdent la même propriété de convergence finie, à savoir que $\exists k \leq n$ (dimension du système à résoudre) tel que $r_k = 0$ c'est à dire $x_k = x = A^{-1}b$.

Cependant dans toutes ces méthodes, interviennent des coefficients avec un produit scalaire comme dénominateurs. Si l'un de ces produits scalaires est nul, il y a un *breakdown* dans l'algorithme qui doit alors être stoppé.

Un tel *breakdown* correspond en fait à la non-existence du polynôme orthogonal que l'on cherche à calculer. Le remède est donc de sauter au dessus de ce polynôme qui n'existe pas et d'aller calculer directement le premier polynôme suivant qui existe. La technique que nous allons exposer pour éviter le *breakdown* est basée sur cette remarque.

4.1.1 MRZ (Method of Recursive Zoom)

Nous voulons donc chercher récursivement seulement les polynômes P_k qui existent (et que l'on appelle réguliers).

Soit donc P_k le polynôme de degré n_k, normalisé par la condition $P_k(0) = 1$, qui satisfait les conditions d'orthogonalité

$$c\left(\xi^i P_k\right) = 0 \qquad \text{pour } i = 0, \ldots, n_k - 1. \tag{4.3}$$

Soit $P_k^{(1)}$ le polynôme unitaire de degré n_k, qui appartient a la famille de polynômes orthogonaux formels par rapport à la fonctionnelle $c^{(1)}$ definie par $c^{(1)}\left(\xi^i\right) = c\left(\xi^{i+1}\right) = c_{i+1}$ et qui satisfait donc aux conditions d'orthogonalité

$$c^{(1)}\left(\xi^i P_k^{(1)}\right) = c\left(\xi^{i+1} P_k^{(1)}\right) = 0 \quad \text{pour } i = 0, \ldots, n_k - 1.$$

L'on peut très facilement prouver que le polynôme P_k avec $P_k(0) = 1$ existe si et seulement si le polynôme unitaire $P_k^{(1)}$ existe et est unique.

Si nous posons $n_{k+1} = n_k + m_k$, Draux [16] a prouvé que les polynômes $P_k^{(1)}$ qui existent, satisfont les conditions suivantes

$$\begin{aligned} & c^{(1)}\left(\xi^i P_k^{(1)}\right) = 0 \qquad \text{pour } i = 0, \ldots, n_k + m_k - 2 \\ \text{et} \quad & c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right) \neq 0. \end{aligned} \tag{4.4}$$

Dans [10] nous avons prouvé que

$$P_{k+1}(\xi) = P_k(\xi) - \xi\, w_k(\xi) P_k^{(1)}(\xi) \tag{4.5}$$

où w_k est un polynôme de degré au plus $m_k - 1$.

Le calcul des polynômes $P_{k+1}^{(1)}(\xi)$, comme a été prouvé par Draux [16] et par nous, avec une démonstration plus courte [7], peut se faire avec la relation à trois termes suivante

$$P_{k+1}^{(1)}(\xi) = q_k(\xi)\, P_k^{(1)}(\xi) - C_{k+1}\, P_{k-1}^{(1)}(\xi) \tag{4.6}$$

avec $P_{-1}^{(1)} = 0$, $P_0^{(1)}(\xi) = 1$ et $C_1 = 0$ et où q_k est un polynôme unitaire de degré m_k.

Le calcul de C_{k+1} et des coefficients des polynômes w_k et q_k sera traité dans la suite en imposant les conditions l'orthogonalité sur P_{k+1} et $P_{k+1}^{(1)}$.

Si nous posons

$$\begin{aligned} r_k &= P_k(A)\, r_0 \\ z_k &= P_k^{(1)}(A)\, r_0 \end{aligned}$$

d'après (4.5) et (4.6) l'on a

$$\begin{aligned} r_{k+1} &= r_k - A\, w_k(A)\, z_k \\ x_{k+1} &= x_k - w_k(A)\, z_k \\ z_{k+1} &= q_k(A)\, z_k - C_{k+1}\, z_{k-1} \end{aligned}$$

avec $z_0 = r_0$, $z_{-1} = 0$ et $C_1 = 0$.

Le saut m_k que l'on doit faire pour passer d'un polynôme P_k de degré n_k au suivant P_{k+1} de degré n_{k+1} qui existe, peut se déterminer par les conditions suivantes

$$\begin{aligned} &\left(y, A^{i+1} z_k\right) = 0\,, \qquad \text{pour } i = 0, \ldots, n_k + m_k - 2 \\ \text{et} \quad &\left(y, A^{n_k+m_k} z_k\right) \neq 0. \end{aligned}$$

Cette méthode presentée dans [10] a été appelée *Method of Recursive Zoom* (MRZ) car les sauts que l'on fait pour passer d'un polymôme au suivant qui existe sont différents et leur recherche ressemble à un *zoom*. Elle est une généralisation de la méthode *Orthodir* et de l'algorithme BIODIR. Elle n'est sujette à aucun breakdown, à part le *incurable hard breakdown* qui se produit quand

$$(y, A^n z_k) = 0$$

avec n dimension du système.

L'on a donc obtenu l'algorithme suivant

Algorithme MRZ

1. Choisir x_0 et y d'une façon arbitraire.
2. Poser $z_0 = r_0 = Ax_0 - b$, $z_{-1} = 0$, $k = 0$, $n_0 = 0$.
3. Déterminer m_k de façon que

$$\begin{aligned} &\left(y, A^{i+1} z_k\right) = 0\,, \qquad \text{pour } i = n_k, \ldots, n_k + m_k - 2 \\ \text{et} \quad &\left(y, A^{n_k+m_k} z_k\right) \neq 0. \end{aligned}$$

4. Calculer les coefficients de w_k et poser
   $r_{k+1} = r_k - Aw_k(A)z_k$, $x_{k+1} = x_k - w_k(A)z_k$.
5. Calculer les coefficients de q_k et poser
   $z_{k+1} = q_k(A)z_k - C_{k+1}z_{k-1}$.
6. Si $r_{k+1} = 0$, alors $x_{k+1} = x$ et stop.
7. Poser $n_{k+1} = n_k + m_k$.
8. Si $n_{k+1} < n$, alors remplacer k avec $k+1$
   et retourner à 3, autrement stop.

4.1.2 Mise en œuvre de l'algorithme MRZ

Nous allons d'abord montrer comment calculer C_{k+1} et les coefficients des polynômes w_k et q_k.

Posons

$$\begin{aligned} w_k(\xi) &= \beta_0 + \cdots + \beta_{m_k-1}\xi^{m_k-1} \\ q_k(\xi) &= \alpha_0 + \cdots + \alpha_{m_k-1}\xi^{m_k-1} + \xi^{m_k}. \end{aligned}$$

Pour obtenir

$$\beta_0, \ldots, \beta_{m_k-1}$$

et

$$\alpha_0, \ldots, \alpha_{m_k-1}$$

il est suffisant d'utiliser les relations (4.5) et (4.6) et d'appliquer les conditions d'orthogonalité de P_k et de $P_k^{(1)}$ et l'on obtient les systèmes suivants

$$\begin{cases} \beta_{m_k-1}c^{(1)}\left(\xi^{n_k+m_k-1}P_k^{(1)}\right) = c(\xi^{n_k}P_k) \\ \beta_{m_k-2}c^{(1)}\left(\xi^{n_k+m_k-1}P_k^{(1)}\right) + \beta_{m_k-1}c^{(1)}\left(\xi^{n_k+m_k}P_k^{(1)}\right) = c(\xi^{n_k+1}P_k) \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \beta_0 c^{(1)}\left(\xi^{n_k+m_k-1}P_k^{(1)}\right) + \beta_1 c^{(1)}\left(\xi^{n_k+m_k}P_k^{(1)}\right) + \cdots + \\ \qquad \beta_{m_k-1}c^{(1)}\left(\xi^{n_k+2m_k-2}P_k^{(1)}\right) = c(\xi^{n_k+m_k-1}P_k) \end{cases}$$

$$\begin{cases} c^{(1)}\left(\xi^{n_k+m_k-1}P_k^{(1)}\right) = C_{k+1}c^{(1)}\left(\xi^{n_k-1}P_{k-1}^{(1)}\right) \\ \alpha_{m_k-1}c^{(1)}\left(\xi^{n_k+m_k-1}P_k^{(1)}\right) + c^{(1)}\left(\xi^{n_k+m_k}P_k^{(1)}\right) = C_{k+1}c^{(1)}\left(\xi^{n_k}P_{k-1}^{(1)}\right) \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \alpha_0 c^{(1)}\left(\xi^{n_k+m_k-1}P_k^{(1)}\right) + \alpha_1 c^{(1)}\left(\xi^{n_k+m_k}P_k^{(1)}\right) + \cdots + \alpha_{m_k-1}c^{(1)}\left(\xi^{n_k+2m_k-2}P_k^{(1)}\right) + \\ \qquad c^{(1)}\left(\xi^{n_k+2m_k-1}P_k^{(1)}\right) = C_{k+1}c^{(1)}\left(\xi^{n_k+m_k-1}P_{k-1}^{(1)}\right). \end{cases}$$

Ces deux systèmes à résoudre sont triangulaires et ils sont toujours non singuliers car $c^{(1)}\left(\xi^{n_k+m_k-1}P_k^{(1)}\right)$ est différent de zéro.

Puisque l'on a

$$c\left(\xi^i P_k\right) = \left(y, A^i P_k(A) r_0\right) = \left(y, A^i r_k\right)$$

$$\text{et} \qquad c^{(1)}\left(\xi^i P_k^{(1)}\right) = \left(y, A^{i+1} P_k^{(1)}(A) r_0\right) = \left(y, A^{i+1} z_k\right)$$

les systèmes deviennent

$$\begin{cases} \beta_{m_k-1}\,(y, A^{n_k+m_k} z_k) = (y, A^{n_k} r_k) \\ \beta_{m_k-2}\,(y, A^{n_k+m_k} z_k) + \beta_{m_k-1}\,(y, A^{n_k+m_k+1} z_k) = (y, A^{n_k+1} r_k) \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \beta_0 (y, A^{n_k+m_k} z_k) + \beta_1 (y, A^{n_k+m_k+1} z_k) + \cdots + \beta_{m_k-1} (y, A^{n_k+2m_k-1} z_k) = \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (y, A^{n_k+m_k-1} r_k) \end{cases}$$

et

$$\begin{cases} (y, A^{n_k+m_k} z_k) = C_{k+1}\,(y, A^{n_k} z_{k-1}) \\ \alpha_{m_k-1}\,(y, A^{n_k+m_k} z_k) + (y, A^{n_k+m_k+1} z_k) = C_{k+1}\,(y, A^{n_k+1} z_{k-1}) \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \alpha_0\,(y, A^{n_k+m_k} z_k) + \alpha_1\,(y, A^{n_k+m_k+1} z_k) + \cdots + \alpha_{m_k-1}\,(y, A^{n_k+2m_k-1} z_k) + \\ \qquad\qquad (y, A^{n_k+2m_k} z_k) = C_{k+1}\,(y, A^{n_k+m_k} z_{k-1})\,. \end{cases}$$

Si nous posons

$$\begin{aligned} d_i &= \left(y, A^{n_k+i} r_k\right) \\ b_i &= \left(y, A^{n_k+m_k+i} z_k\right) \\ p_i &= \left(y, A^{n_k+i} z_{k-1}\right) \end{aligned}$$

les systèmes triangulaires ont la forme

$$\begin{cases} \beta_{m_k-1} b_0 = d_0 \\ \beta_{m_k-2} b_0 + \beta_{m_k-1} b_1 = d_1 \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ \beta_0 b_0 + \beta_1 b_1 + \cdots + \beta_{m_k-1} b_{m_k-1} = d_{m_k-1} \end{cases}$$

et

$$\begin{cases} b_0 = C_{k+1}\, p_0 \\ \alpha_{m_k-1} b_0 + b_1 = C_{k+1}\, p_1 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \alpha_0 b_0 + \alpha_1 b_1 + \cdots + \alpha_{m_k-1} b_{m_k-1} + b_{m_k} = C_{k+1}\, p_{m_k}. \end{cases}$$

L'analyse de ces systèmes montre qu'ils peuvent être réunis en un seul système avec deux seconds membres différents selon le schéma suivant

Système de m_k équations pour β
Système de m_k équations pour α $(C_{k+1} = b_0/p_0)$

$$\begin{array}{ccccccc|cc} \beta_0 & \beta_1 & \cdots & \cdots & \beta_{m_k-1} & \text{pour} & & \text{pour} & \\ \alpha_0 & \alpha_1 & \cdots & \cdots & \alpha_{m_k-1} & \beta & & \alpha & \\ \downarrow & \downarrow & & & \downarrow & \downarrow & & \downarrow & \\ b_0 & b_1 & \cdots & \cdots & b_{m_k-1} & d_{m_k-1} & & -b_{m_k} + p_{m_k} b_0/p_0 & \\ & b_0 & \cdots & \cdots & b_{m_k-2} & d_{m_k-2} & & -b_{m_k-1} + p_{m_k-1} b_0/p_0 & \\ & & \ddots & & \vdots & \vdots & & \vdots & \\ & & & \vdots & \vdots & \vdots & & \vdots & \\ & 0 & & b_0 & b_1 & d_1 & & -b_2 + p_2 b_0/p_0 & \\ & & & & b_0 & d_0 & & -b_1 + p_1 b_0/p_0 & \end{array}$$

Les relations qui donnent les valeurs de x_{k+1}, r_{k+1} et z_{k+1} peuvent s'écrire

$$\begin{aligned} x_{k+1} &= x_k - [\beta_0 z_k + \beta_1 A z_k + \cdots + \beta_{m_k-1} A^{m_k-1} z_k] \\ r_{k+1} &= r_k - [\beta_0 A z_k + \beta_1 A^2 z_k + \cdots + \beta_{m_k-1} A^{m_k} z_k] \\ z_{k+1} &= \alpha_0 z_k + \alpha_1 A z_k + \cdots + \alpha_{m_k-1} A^{m_k-1} z_k + A^{m_k} z_k - C_{k+1} z_{k-1}. \end{aligned}$$

Si l'on pose

$$s_i = A^i z_k = A\, s_{i-1}$$

avec

$$s_0 = z_k$$

on aura

$$\begin{aligned} x_{k+1} &= x_k - [\beta_0 s_0 + \beta_1 s_1 + \cdots + \beta_{m_k-1} s_{m_k-1}] \\ r_{k+1} &= r_k - [\beta_0 s_1 + \beta_1 s_2 + \cdots + \beta_{m_k-1} s_{m_k}] \\ z_{k+1} &= \alpha_0 s_0 + \alpha_1 s_1 + \cdots + \alpha_{m_k-1} s_{m_k-1} + s_{m_k} - C_{k+1} z_{k-1}. \end{aligned}$$

Donc dans cet algorithme on aura besoin à chaque étape k des quantités suivantes

$$\begin{aligned} d_i &= (y, A^{n_k+i} r_k) \quad \text{pour } i = 0, \ldots, m_k - 1 \\ b_i &= (y, A^{n_k+m_k+i} z_k) \quad \text{pour } i = 0, \ldots, m_k \\ p_i &= (y, A^{n_k+i} z_{k-1}) \quad \text{pour } i = 0, \ldots, m_k \\ s_i &= A^i z_k \quad \text{pour } i = 0, \ldots, m_k . \end{aligned}$$

Nous allons maintenant montrer comment on a diminué le volume des calculs dans cet algorithme pour les quantités précédentes.

Dans le MRZ on doit toujours calculer des produits scalaires de la forme

$$(y, A^{i+j} r_k) \quad \text{pour } i + j = n_k, \ldots, n_k + m_k - 1$$

$$\text{et} \quad (y, A^{i+j} z_k) \quad \text{pour } i + j = n_k + m_k, \ldots, n_k + 2m_k$$

et, en plus, pour le calcul de x_{k+1}, r_{k+1} et de z_{k+1} il faut calculer et garder en mémoire les vecteurs

$$s_i = A^i z_k \quad \text{pour } i = 0, \ldots, m_k.$$

Il faut avoir aussi les produits scalaires

$$(y, A^{i+j} z_{k-1}) \quad \text{pour } i + j = n_k, \ldots, n_k + m_k.$$

Mais dans l'iteration précédente l'on avait déjà calculé

$$\begin{aligned} (y, A^{i+j} z_{k-1}) \quad \text{pour } i + j &= n_{k-1} + m_{k-1}, \ldots, n_{k-1} + 2m_{k-1} \\ &= n_k, \ldots, n_k + m_{k-1}. \end{aligned}$$

Il y a donc à calculer de nouveaux produits scalaires seulement si $m_k > m_{k-1}$:

$$(y, A^{i+j} z_{k-1}) \quad \text{pour } i + j = n_k + m_{k-1} + 1, n_k + m_{k-1} + 2, \ldots, n_k + m_k.$$

Pour calculer tous ces produits scalaires on utilise la proprieté des produits scalaires qui donne

$$\begin{aligned} \left(y, A^{i+j} r_k\right) &= \left(A^{T^i} y, A^j r_k\right), \\ \left(y, A^{i+j} z_k\right) &= \left(A^{T^i} y, A^j z_k\right), \\ \left(y, A^{i+j} z_{k-1}\right) &= \left(A^{T^i} y, A^j z_{k-1}\right). \end{aligned}$$

et donc, si à chaque étape k l'on sort avec la valeur

$$A^{T^{n_k}} y = \tilde{y}$$

à la place de y, il est suffisant de calculer

$$\left(A^{T^{n_k}}y, A^j r_k\right) = (\tilde{y}, A^j r_k) \qquad \text{pour } j = 0, \ldots, m_k - 1,$$

$$\left(A^{T^{n_k}}y, A^j z_k\right) = (\tilde{y}, A^j z_k) \qquad \text{pour } j = m_k, \ldots, 2m_k,$$

et $$\left(A^{T^{n_k}}y, A^j z_{k-1}\right) = (\tilde{y}, A^j z_{k-1}) \quad \text{pour } j = m_{k-1}+1, \ldots, m_k$$
(seulement si $m_k > m_{k-1}$).

Dans chaque étape, pendant le calcul du saut m_k, on peut déjà calculer et garder en mémoire les vecteurs

$$s_i \;=\; A^i z_k \quad \text{pour } i = 0, \ldots, m_k$$

et pour calculer les quantités d_i, b_i et p_i on applique l'algorithme suivant qui permet aussi, à la fin de l'étape k, de sortir toujours avec $y = A^{T^{n_k}}y$.

$\tilde{y} \leftarrow y$
$d_0 \leftarrow (\tilde{y}, r_k) = \left(A^{T^{n_k}}y, r_k\right)$
$b_0 \leftarrow (\tilde{y}, s_{m_k}) = \left(A^{T^{n_k}}y, A^{m_k}z_k\right)$
For $i = 1, \ldots, m_k$ **do:**
  $\tilde{y} \leftarrow A^{\mathrm{T}}\tilde{y}$
  $b_i \leftarrow (\tilde{y}, s_{m_k}) = \left(A^{T^{n_k+i}}y, A^{m_k}z_k\right)$
  If $i \neq m_k$ **then**
    $d_i \leftarrow (\tilde{y}, r_k) = \left(A^{T^{n_k+i}}y, r_k\right)$
  end if
  If $k \neq 0$ and $i > m_{k-1}$ **then**
    $p_i \leftarrow (\tilde{y}, z_{k-1}) = \left(A^{T^{n_k+i}}y, z_{k-1}\right)$
  end if
end for
$y = \tilde{y}$

Dans [11] nous avons montré une autre façon pour calculer les produits scalaires nécessaires dans l'algorithme et une façon différente pour calculer les coefficients qui conduit à une méthode semblable à la méthode BIODIR de Gutknecht [19].

4.1.3 Pseudo-code du MRZ

D'après tout ce que l'on a vu on peut donner maintenant le pseudo-code complet de l'algorithme MRZ.

Algorithm MRZ (A, b, x_0, y)

1. **Initializations:**
   $z_{-1} \leftarrow 0$
   $r_0 \leftarrow A x_0 - b$
   $s_0 = z_0 = r_0$
   $n_0 \leftarrow 0$
2. **For** $k = 0, 1, 2, \ldots$ **until** convergence **do:**
   If $n_k = n$ **then**
   solution not obtained after n iterations.
   stop.
   end if
   $s_1 \leftarrow A s_0$
   If $(y, s_1) = 0$ **and** $n_k = n - 1$ **then**
   incurable breakdown.
   stop.
   end if
   $m_k \leftarrow 1$
3. While $(y, s_{m_k}) = 0$ and $m_k < n - n_k$ do:
   $m_k \leftarrow m_k + 1$
   $s_{m_k} \leftarrow A s_{m_k - 1}$
   end while
   If $m_k = n - n_k$ **and** $(y, s_{m_k}) = 0$ **then**
   incurable breakdown.
   stop.
   end if
   $d_0 \leftarrow (y, r_k)$
   $b_0 \leftarrow (y, s_{m_k})$
   $\beta_{m_k - 1} \leftarrow d_0 / b_0$
4. **For** $i = 1, \ldots, m_k$ **do:**
   $y \leftarrow A^{\mathrm{T}} y$
   $b_i \leftarrow (y, s_{m_k})$
   If $i \neq m_k$ **then**
   $d_i \leftarrow (y, r_k)$
   compute $\beta_{m_k - i - 1}$
   end if
   If $k \neq 0$ and $i > m_{k-1}$ **then**
   $p_i \leftarrow (y, z_{k-1})$
   end if
   end for
5. compute $x_{k+1} = x_k - w_k(A)\, z_k$
   compute $r_{k+1} = r_k - A\, w_k(A)\, z_k$
   If $r_{k+1} = 0$ **then**
   solution obtained.

```
            stop.
        end if
6.      n_{k+1} ← n_k + m_k
        If k = 0 then
            C_1 ← 0
            p_0 ← 0
        else
            C_{k+1} ← b_0/p_0
        end if
7.      For i = 1,...,m_k do:
            If k = 0 then
                p_i ← 0
            end if
            compute α_{m_k - i}
        end for
8.      For i = 0,...,m_k do:
            p_i ← b_i
        end for
        s_0 ← z_{k+1} = q_k(A) z_k − C_{k+1} z_{k-1}
    end for
```

4.1.4 Exemples numériques

On considère le système suivant qui a été donné dans [1]

$$\begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & -1 \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \\ \vdots \\ n \end{pmatrix} = \begin{pmatrix} -n \\ 1 \\ 2 \\ \vdots \\ n-1 \end{pmatrix}.$$

Evidemment, comme l'on fait des calculs numériques et donc que l'on a des erreurs d'arrondi, au lieu de tester, dans la subroutine, l'égalité à zéro nous allons donner une valeur ε. Quand une quantité dans l'algorithme est plus petite en valeur absolue que l'ε qu'on a choisi, alors elle est considerée comme nulle. Les calculs ont été faits sur un PC en utilisant la double précision.

Avec le subroutine MRZ on obtient les résultats suivants en utilisant $n = 12, x_0 = 0, y = (1,\ldots,1)^T$ et $\varepsilon = 10^{-1}$

k	1	2	3	4	5	6	7	8	9	10
n_k	1	2	3	6	7	8	9	10	11	12
$\|r_k\|$	15.0	18.3	37.5	41.1	41.1	945.3	948.8	37.7	18.3	7.0

Pour $\varepsilon = 10^{-2}, 10^{-3}, 10^{-5}, 10^{-10}, 10^{-11}$, on obtient

k	1	2	3	4	5	6	7	8
n_k	1	2	3	4	9	10	11	12
$\|r_k\|$	15.0	18.3	37.5	58.2	58.2	37.6	18.2	$3.2 \cdot 10^{-9}$

Pour $\varepsilon = 10^{-12}$, on a

k	1	2	3	4	5	6	7	8	9
n_k	1	2	3	4	8	9	10	11	12
$\|r_k\|$	15.0	18.3	37.5	58.2	$2.5 \cdot 10^{41}$	$2.9 \cdot 10^{25}$	$9.8 \cdot 10^{24}$	$8.1 \cdot 10^{25}$	$2.8 \cdot 10^{26}$

Pour $\varepsilon = 10^{-13}$, on obtient

k	1	2	3	4	5	6	7	8	9	10	11
n_k	1	2	3	4	6	7	8	9	10	11	12
$\|r_k\|$	15.0	18.3	37.5	58.2	47.7	47.7	$1.0 \cdot 10^{14}$	58.2	42.3	54.3	$8.4 \cdot 10^{4}$

Finalement, pour $\varepsilon = 10^{-14}, 10^{-15}$ et 10^{-16} on trouve

k	1	2	3	4	5	6	7	8	9	10	11	12
n_k	1	2	3	4	5	6	7	8	9	10	11	12
$\|r_k\|$	15.0	18.3	37.5	58.2	63.6	73.6	73.6	63.6	58.2	37.6	18.2	57.4

Donc ces résultats sont très sensibles au choix de ε et cela est compréhensible car cette valeur contôle le début du saut qui, pour donner de bons résultats, doit être de 4 à 9. Quand le saut est bien detecté la solution exacte est obtenue.

4.2 Les variantes du MRZ

Si l'on change la façon de calculer récursivement les polynômes $P_{k+1}^{(1)}$ (dans le MRZ on utilisait la formule de récurrence à trois termes pour ces polynômes) on peut obtenir des variantes de la méthode.

Dans la première variante, $P_{k+1}^{(1)}$ est calculé en utilisant $P_k^{(1)}$ et P_k (qui sont les mêmes polynômes que ceux utilisés dans le calcul de P_{k+1} et donc on a appelé cette méthode SMRZ, où S signifie *symmetric*, car il y a une symétrie entre les deux relations).

La deuxième variante a été nommée BMRZ, où B signifie *balancing*, car la méthode calcule P_{k+1} à partir de $P_k^{(1)}$ et de P_k et après elle calcule $P_{k+1}^{(1)}$ à partir de P_{k+1} et de $P_k^{(1)}$. Donc il y a une espèce de balancement entre les deux relations.

4.2.1 SMRZ (Symmetric Method of Recursive Zoom)

Dans le SMRZ, on calcule $P_{k+1}^{(1)}$ avec la relation

$$P_{k+1}^{(1)}(\xi) = t_k(\xi)\, P_k^{(1)}(\xi) - D_{k+1}\, P_k(\xi) \tag{4.7}$$

où t_k est un polynôme unitaire de degré m_k et D_{k+1} est une constante.

Dans [11] nous avons montré que le SMRZ peut être utilisé seulement si $c(\xi^{n_k} P_k) \neq 0$ et que $c(\xi^{n_k} P_k) = 0$ si et seulement si P_{k+1} est identique à P_k.

Si $c(\xi^{n_k} P_k) = 0$ alors $P_{k+1}^{(1)}$ ne peut pas non plus être calculé comme combinaison linéaire (utilisant des coefficients polynomiaux) de $P_k^{(1)}$ et de P_k, c'est à dire qu'il n'existe pas de relation de la forme

$$P_{k+1}^{(1)}(\xi) = t_k(\xi)\, P_k^{(1)}(\xi) - u_k(\xi)\, P_k(\xi)$$

avec t_k et u_k polynômes de degré au plus m_k.

Donc cette variante est moins générale que le MRZ.

4.2.2 Mise en œuvre de l'algorithme SMRZ

Si nous posons

$$t_k(\xi) = \delta_0 + \cdots + \delta_{m_k-1}\xi^{m_k-1} + \xi^{m_k}$$

et si l'on multiplie (4.7) par ξ^{i+1} et que l'on applique la fonctionnelle c on aura

$$\begin{aligned} c^{(1)}\left(\xi^i P_{k+1}^{(1)}\right) \;=\; & \delta_0\, c^{(1)}\left(\xi^i P_k^{(1)}\right) + \cdots + \delta_{m_k-1}\, c^{(1)}\left(\xi^{i+m_k-1} P_k^{(1)}\right) + \\ & c^{(1)}\left(\xi^{i+m_k} P_k^{(1)}\right) - D_{k+1}\, c\left(\xi^{i+1} P_k\right). \end{aligned}$$

Pour obtenir D_{k+1} et les coefficients du polynôme t_k

$$\delta_0, \ldots, \delta_{m_k-1}$$

il est suffisant d'imposer les conditions d'orthogonalité

$$c^{(1)}\left(\xi^i P_{k+1}^{(1)}\right) = 0 \qquad \text{pour } i = 0, \ldots, n_k + m_k - 1.$$

Ces conditions sont déjà verifiées pour $i = 0, \ldots, n_k - 2$ car on a $c^{(1)}\left(\xi^{i+m_k} P_k^{(1)}\right) = 0$ pour $i = 0, \ldots, n_k - 2$ et $c(\xi^{i+1} P_k) = 0$ pour $i = 0, \ldots, n_k - 2$. Donc il est suffisant d'imposer les $m_k + 1$ conditions restantes pour obtenir le système suivant qui permet le calcul des inconnues

$$\left\{ \begin{array}{l} c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right) = D_{k+1} c(\xi^{n_k} P_k) \\ \\ \delta_{m_k-1} c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right) + c^{(1)}\left(\xi^{n_k+m_k} P_k^{(1)}\right) = D_{k+1} c(\xi^{n_k+1} P_k) \\ \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \\ \delta_0 c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right) + \delta_1 c^{(1)}\left(\xi^{n_k+m_k} P_k^{(1)}\right) + \cdots + \delta_{m_k-1} c^{(1)}\left(\xi^{n_k+2m_k-2} P_k^{(1)}\right) + \\ \qquad\qquad c^{(1)}\left(\xi^{n_k+2m_k-1} P_k^{(1)}\right) = D_{k+1} c(\xi^{n_k+m_k} P_k). \end{array} \right.$$

c'est à dire

$$\begin{cases} (y, A^{n_k+m_k} z_k) = D_{k+1}\,(y, A^{n_k} r_k) \\ \\ \delta_{m_k-1}\,(y, A^{n_k+m_k} z_k) + (y, A^{n_k+m_k+1} z_k) = D_{k+1}\,(y, A^{n_k+1} r_k) \\ \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \\ \delta_0\,(y, A^{n_k+m_k} z_k) + \delta_1\,(y, A^{n_k+m_k+1} z_k) + \cdots + \delta_{m_k-1}\,(y, A^{n_k+2m_k-1} z_k) + \\ \qquad (y, A^{n_k+2m_k} z_k) = D_{k+1}\,(y, A^{n_k+m_k} r_k)\,. \end{cases}$$

Comme $c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right) \neq 0$, ce système est determiné si et seulement si $c(\xi^{n_k} P_k) \neq 0$ et l'on retrouve la condition déjà vue.

Avec les mêmes notations que pour le MRZ

$$\begin{aligned} d_i &= \left(y, A^{n_k+i} r_k\right) \\ b_i &= \left(y, A^{n_k+m_k+i} z_k\right) \end{aligned}$$

ce système triangulaire est

$$\begin{cases} b_0 = D_{k+1}\, d_0 \\ \\ \delta_{m_k-1} b_0 + b_1 = D_{k+1}\, d_1 \\ \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ \\ \delta_0 b_0 + \delta_1 b_1 + \cdots + \delta_{m_k-1} b_{m_k-1} + b_{m_k} = D_{k+1}\, d_{m_k} \end{cases}$$

qui est de la même forme que celle du système qui sert pour calculer les $\beta_0, \ldots, \beta_{m_k-1}$ et donc on a le schéma suivant

Système de m_k équations pour β
Système de m_k équations pour δ $(D_{k+1} = b_0/d_0)$

β_0	β_1	$\cdots$	$\cdots$	β_{m_k-1}	pour	pour
δ_0	δ_1	$\cdots$	$\cdots$	δ_{m_k-1}	β	δ
$\downarrow$	$\downarrow$			$\downarrow$	$\downarrow$	$\downarrow$
b_0	b_1	$\cdots$	$\cdots$	b_{m_k-1}	d_{m_k-1}	$-b_{m_k} + d_{m_k} b_0/d_0$
	b_0	$\cdots$	$\cdots$	b_{m_k-2}	d_{m_k-2}	$-b_{m_k-1} + d_{m_k-1} b_0/d_0$
		$\ddots$		$\vdots$	$\vdots$	$\vdots$
			$\vdots$	$\vdots$	$\vdots$	$\vdots$
	0		b_0	b_1	d_1	$-b_2 + d_2 b_0/d_0$
				b_0	d_0	$-b_1 + d_1 b_0/d_0$

En ce qui concerne les relations qui donnent les valeurs de x_{k+1}, r_{k+1} et z_{k+1} seulement la dernière sera changée et elle deviendra

$$z_{k+1} = \delta_0 z_k + \delta_1 A z_k + \cdots + \delta_{m_k-1} A^{m_k-1} z_k + A^{m_k} z_k - D_{k+1} r_k$$

c'est à dire

$$z_{k+1} = \delta_0 s_0 + \delta_1 s_1 + \cdots + \delta_{m_k-1} s_{m_k-1} + s_{m_k} - D_{k+1} r_k$$

avec $s_i = A^i z_k = A\, s_{i-1}$ et $s_0 = z_k$.

Donc dans cet algorithme, en faisant une comparaison avec le MRZ, nous n'aurons plus besoin, dans l'étape k, des quantités p_i ni du vecteur z_{k-1} mais on devra calculer les d_i jusqu'à $i = m_k$.

4.2.3 Pseudo-code du SMRZ

Algorithm SMRZ (A, b, x_0, y)

1. **Initializations:**
   $z_{-1} \leftarrow 0$
   $r_0 \leftarrow A\, x_0 - b$
   $s_0 = z_0 = r_0$
   $n_0 \leftarrow 0$
2. **For** $k = 0, 1, 2, \ldots$ **until** convergence **do:**
   If $n_k = n$ **then**
   solution not obtained after n iterations.
   stop.
   end if
   $s_1 \leftarrow A\, s_0$
   If $(y, s_1) = 0$ **and** $n_k = n - 1$ **then**
   incurable breakdown.
   stop.
   end if
   $m_k \leftarrow 1$
3. **While** $(y, s_{m_k}) = 0$ **and** $m_k < n - n_k$ **do:**
   $m_k \leftarrow m_k + 1$
   $s_{m_k} \leftarrow A\, s_{m_k-1}$
   end while
   If $m_k = n - n_k$ **and** $(y, s_{m_k}) = 0$ **then**
   incurable breakdown.
   stop.
   end if
   $d_0 \leftarrow (y, r_k)$
   If $d_0 = 0$ **then**
   $(y, A^{n_k} r_k) = 0$
   use the MRZ algorithm.

```
        stop.
    end if
    b_0 ← (y, s_{m_k})
    β_{m_k-1} ← d_0/b_0
4.  For i = 1, ..., m_k do:
        y ← A^T y
        b_i ← (y, s_{m_k})
        d_i ← (y, r_k)
        If i ≠ m_k then
            compute β_{m_k-i-1}
        end if
    end for
5.  compute x_{k+1} = x_k − w_k(A) z_k
    compute r_{k+1} = r_k − A w_k(A) z_k
    If r_{k+1} = 0 then
        solution obtained.
        stop.
    end if
6.  n_{k+1} ← n_k + m_k
    D_{k+1} ← b_0/d_0
7.  For i = 1, ..., m_k do:
        compute δ_{m_k-i}
    end for
8.  s_0 ← z_{k+1} = t_k(A) z_k − D_{k+1} r_k
end for
```

4.2.4 BMRZ (Balancing Method of Recursive Zoom)

Dans ce cas $P_{k+1}^{(1)}$ est calculé par

$$P_{k+1}^{(1)}(\xi) \;=\; A_{k+1}\,P_{k+1}(\xi) + B_{k+1}\,P_k^{(1)}(\xi) \tag{4.8}$$

où A_{k+1} et B_{k+1} sont des constantes. Nous n'avons plus à calculer les coefficients d'un polynôme, mais il est suffisant de calculer deux constantes. Multipliant par ξ^{i+1} et appliquant la fonctionnelle c, on obtient

$$c^{(1)}\left(\xi^i P_{k+1}^{(1)}\right) = A_{k+1}\,c\left(\xi^{i+1}P_{k+1}\right) + B_{k+1}\,c^{(1)}\left(\xi^i P_k^{(1)}\right).$$

Les conditions d'orthogonalité de $P_{k+1}^{(1)}$ sont toujours verifiées pour $i = 0, \ldots, n_k + m_k - 2$ (car $c^{(1)}\left(\xi^i P_k^{(1)}\right) = 0$ pour $i = 0, \ldots, n_k + m_k - 2$ et $c(\xi^{i+1}P_{k+1}) = 0$ pour $i = 0, \ldots, n_k + m_k - 2$). Donc il nous reste à appliquer la condition pour $i = n_k + m_k - 1$ qui nous donne

$$A_{k+1}\,c\left(\xi^{n_k+m_k}P_{k+1}\right) + B_{k+1}\,c^{(1)}\left(\xi^{n_k+m_k-1}P_k^{(1)}\right) = 0.$$

Comme l'on veut $P_{k+1}^{(1)}$ unitaire et que $P_{k+1}^{(1)}$ a le degré $n_k + m_k$ alors (4.8) est valable si et seulement si P_{k+1} a le degré $n_k + m_k$.

La première relation du MRZ (4.5) donne comme coefficient de $\xi^{n_k+m_k}$, dans le polynôme P_{k+1}, la valeur $-\beta_{m_k-1}$. Donc pour que $P_{k+1}^{(1)}$ soit unitaire il faut que

$$A_{k+1} = -\frac{1}{\beta_{m_k-1}} = -\frac{c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right)}{c(\xi^{n_k} P_k)}$$

et ainsi l'on obtient

$$B_{k+1} = \frac{1}{\beta_{m_k-1}} \cdot \frac{c(\xi^{n_k+m_k} P_{k+1})}{c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right)} = \frac{c(\xi^{n_k+m_k} P_{k+1})}{c(\xi^{n_k} P_k)}.$$

Cette relation est la même de Draux [16] (pp. 397–398) et elle est une généralisation de la deuxième relation entre familles adjacentes de polynômes orthogonaux, c'est à dire la relation qui implique $e_k^{(0)}$ (voir Brezinski [2] (relation 2.9, p. 92 ou p. 95)).

Dans le BMRZ aussi, si $c(\xi^{n_k} P_k) = 0$, alors $P_{k+1}^{(1)}$ ne peut pas être calculé comme combinaison linéaire de $P_k^{(1)}$ et de P_k (car dans ce cas $P_{k+1} = P_k$). En effet le BMRZ est une version simplifiée de l'algorithme SMRZ car si l'on remplace dans (4.8) P_{k+1} par (4.5) on obtient

$$P_{k+1}^{(1)}(\xi) = (B_{k+1} - A_{k+1}\,\xi\, w_k(\xi))\, P_k^{(1)}(\xi) + A_{k+1}\, P_k(\xi)$$

c'est à dire la relation (4.7) du SMRZ.

4.2.5 Mise en œuvre de l'algorithme BMRZ

Dans le BMRZ nous avons seulement un système à résoudre qui nous donne les $\beta_0, \ldots, \beta_{m_k-1}$. En ce qui concerne A_{k+1} et B_{k+1}, si l'on pose toujours $r_k = P_k(A)\, r_0$ et $z_k = P_k^{(1)}(A)\, r_0$, $d_0 = (y, A^{n_k} r_k)$ et $b_0 = (y, A^{n_k+m_k} z_k)$ on aura

$$\begin{aligned} A_{k+1} &= -\frac{b_0}{d_0} \\ B_{k+1} &= \frac{(y, A^{n_k+m_k} r_{k+1})}{d_0}. \end{aligned}$$

Cette formulation est celle du gradient biconjugué de Fletcher [18] (voir aussi Brezinski [2] p. 91, où z_{k+1} est calculé à partir de r_{k+1} et z_k).

4.2.6 Pseudo-code du BMRZ

Algorithm BMRZ (A, b, x_0, y)

1. **Initializations:**

   $z_{-1} \leftarrow 0$

   $r_0 \leftarrow A\, x_0 - b$

$s_0 = z_0 = r_0$
$n_0 \leftarrow 0$

2. **For** $k = 0, 1, 2, \ldots$ **until** convergence **do:**
   If $n_k = n$ **then**
      solution not obtained after n iterations.
      stop.
   end if
   $s_1 \leftarrow A\, s_0$
   If $(y, s_1) = 0$ **and** $n_k = n - 1$ **then**
      incurable breakdown.
      stop.
   end if
   $m_k \leftarrow 1$
3. **While** $(y, s_{m_k}) = 0$ **and** $m_k < n - n_k$ **do:**
      $m_k \leftarrow m_k + 1$
      $s_{m_k} \leftarrow A\, s_{m_k - 1}$
   end while
   If $m_k = n - n_k$ **and** $(y, s_{m_k}) = 0$ **then**
      incurable breakdown.
      stop.
   end if
   $d_0 \leftarrow (y, r_k)$
   If $d_0 = 0$ **then**
      $(y, A^{n_k} r_k) = 0$
      use the MRZ algorithm.
      stop.
   end if
   $b_0 \leftarrow (y, s_{m_k})$
   $\hat{y} \leftarrow y$
   $\beta_{m_k - 1} \leftarrow d_0 / b_0$
4. **For** $i = 1, \ldots, m_k$ **do:**
      $y \leftarrow A^{\mathrm{T}} y$
      If $i \neq m_k$ **then**
         $b_i \leftarrow (y, s_{m_k})$
         $d_i \leftarrow (y, r_k)$
         compute $\beta_{m_k - i - 1}$
      end if
   end for
5. compute $x_{k+1} = x_k - w_k(A)\, z_k$
   compute $r_{k+1} = r_k - A\, w_k(A)\, z_k$
   If $r_{k+1} = 0$ **then**
      solution obtained.
      stop.

end if

6. $n_{k+1} \leftarrow n_k + m_k$
7. **For** $i = 1, \ldots, m_k$ **do:**
   $\hat{y} \leftarrow A^{\mathrm{T}}\, \hat{y}$
   end for
   $A_{k+1} \leftarrow -1 / \beta_{m_k - 1}$
   $B_{k+1} \leftarrow (\hat{y}, r_{k+1}) / d_0$
8. $s_0 \leftarrow z_{k+1} = A_{k+1}\, r_{k+1} + B_{k+1}\, z_k$

end for

4.2.7 Exemples numériques

Nous allons reprendre le système utilisé dans le paragraphe 4.1.4 et avec la même signification pour la valeur ε.

Nous allons donner une comparaison entre le MRZ et ses variantes en montrant la norme du residu pour les différents algorithmes. Quand une valeur manque, cela veut dire que la solution n'a pas été obtenue (à cause d'une division par zéro dans l'algorithme qui provient de la condition supplémentaire).

On obtient les résultats suivants avec $x_0 = 0$ et $y = (1, \ldots, 1)^T$

n	MRZ	BMRZ	SMRZ
	$\varepsilon = 10^{-8}$	$\varepsilon = 10^{-8}$	$\varepsilon = 10^{-8}$
4	$2.74 \cdot 10^{-15}$	$1.58 \cdot 10^{-15}$	$1.46 \cdot 10^{-15}$
5	$7.20 \cdot 10^{-15}$	$6.21 \cdot 10^{-14}$	$1.62 \cdot 10^{-13}$
6	$1.33 \cdot 10^{-11}$	$5.39 \cdot 10^{-14}$	$2.63 \cdot 10^{-13}$
7	$5.49 \cdot 10^{-13}$		
8	$6.53 \cdot 10^{-12}$		
9	$4.23 \cdot 10^{-11}$		
10	$5.09 \cdot 10^{-11}$		
11	$1.10 \cdot 10^{-11}$		
12	$3.33 \cdot 10^{-11}$		

Avec $y = r_0$ on trouve

n	MRZ	BMRZ	SMRZ
	$\varepsilon = 10^{-8}$	$\varepsilon = 10^{-8}$	$\varepsilon = 10^{-8}$
4	0.0		
5	$1.06 \cdot 10^{-10}$	$3.39 \cdot 10^{-13}$	$8.12 \cdot 10^{-13}$
6	$2.32 \cdot 10^{-8}$	$1.53 \cdot 10^{-10}$	$1.90 \cdot 10^{-10}$
7	$3.02 \cdot 10^{-10}$	$2.62 \cdot 10^{-12}$	$3.54 \cdot 10^{-12}$
8	$2.04 \cdot 10^{-11}$		
9	$4.20 \cdot 10^{-11}$		
10	$4.57 \cdot 10^{-10}$		
11	$5.76 \cdot 10^{-10}$		
12	$1.80 \cdot 10^{-9}$		

Ces résultats semblent montrer que le BMRZ et le SMRZ sont plus stables que le MRZ mais il est nécessaire d'effectuer d'autres essais pour mieux comprendre la stabilité numérique de ces algorithmes.

4.3 Near-breakdown dans Lanczos

On a vu qu'il y a un breakdown dans une méthode de type Lanczos quand

$$c^{(1)}\left(\xi^{n_k} P_k^{(1)}\right) = 0$$

car, dans ce cas, le polynôme orthogonal unitaire de degré $n_k + 1$ par rapport à la fonctionnelle $c^{(1)}$ n'existe pas. On doit donc chercher le premier polynôme régulier $P_{k+1}^{(1)}$ de degré $n_{k+1} = n_k + m_k$ qui suit $P_k^{(1)}$. Le saut m_k sera determiné de façon que les relations (4.4) soient vraies.

Si $\left|c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right)\right|$ n'est pas zéro mais voisin de zéro cela signifie qu'en fait le polynôme orthogonal suivant risque d'être calculé avec une faible précision due à une erreur de cancellation. Cela est vrai aussi si les quantités $\left|c^{(1)}\left(\xi^i P_k^{(1)}\right)\right|$ ne sont pas zéro pour $i = n_k, \ldots, n_k + m_k - 2$ mais petites. Cette situation a été appelé *near-breakdown* et le remède consiste encore à sauter au dessus du ou des polynômes qui risquent d'être mal calculés, pour aller calculer directement le premier polynôme suivant qui est calculé correctement.

Soit donc $\varepsilon \geq 0$ et $m_k \geq 1$ l'indice pour lequel on a

$$\left|c^{(1)}\left(\xi^i P_k^{(1)}\right)\right| \leq \varepsilon \qquad \text{pour } i = n_k, \ldots, n_k + m_k - 2$$

$$\text{et} \quad \left|c^{(1)}\left(\xi^i P_k^{(1)}\right)\right| > \varepsilon \qquad \text{pour } i = n_k + m_k - 1.$$

Soit encore $P_{k+1}^{(1)}$ le polynôme orthogonal régulier de degré $n_{k+1} = n_k + m_k$ par rapport à $c^{(1)}$ et P_{k+1} le polynôme orthogonal correspondant de degré au plus n_{k+1} par rapport à c, normalisé par la condition $P_{k+1}(0) = 1$.

Dans [11] nous avons prouvé que $P_{k+1}(\xi)$ peut s'écrire comme

$$P_{k+1}(\xi) = P_k(\xi) - \xi\, w_k(\xi)\, P_k^{(1)}(\xi) - \xi\, v_k(\xi)\, P_k(\xi) \tag{4.9}$$

où w_k est un polynôme de degré au plus $m_k - 1$, v_k un polynôme de degré au plus $m_k - 2$ et avec $P_{-1}^{(1)} = 0$, $P_0^{(1)}(\xi) = 1$.

Cette relation correspond à la relation (4.5) du breakdown car dans ce cas là v_k est le polynôme identiquement nul.

Si nous posons encore

$$\begin{aligned} r_k &= P_k(A)\, r_0 \\ \text{et} \quad z_k &= P_k^{(1)}(A)\, r_0 \end{aligned}$$

d'après (4.9), on a

$$\begin{aligned} r_{k+1} &= r_k - A\, w_k(A)\, z_k - A\, v_k(A) r_k \\ x_{k+1} &= x_k - w_k(A)\, z_k - v_k(A) r_k \end{aligned}$$

avec $z_0 = r_0$.

Le calcul récursif des polynômes P_{k+1} peut se faire de différentes façons, comme on l'a fait dans le MRZ, le SMRZ ou dans le BMRZ.

En revenant au calcul des $2m_k - 1$ coefficients inconnus des polynômes w_k et v_k, la relation (4.9) nous donne

$$c\left(\xi^i P_{k+1}\right) = c\left(\xi^i P_k\right) - c^{(1)}\left(\xi^i\, w_k\, P_k^{(1)}\right) - c\left(\xi^{i+1}\, v_k\, P_k\right).$$

Et on a encore

$$\begin{aligned} c\left(\xi^i P_k\right) &= 0 \qquad i = 0, \ldots, n_k - 1 \\ c^{(1)}\left(\xi^i\, w_k\, P_k^{(1)}\right) &= 0 \qquad i = 0, \ldots, n_k - m_k \\ c\left(\xi^{i+1}\, v_k\, P_k\right) &= 0 \qquad i = 0, \ldots, n_k - m_k. \end{aligned}$$

Donc

$$c\left(\xi^i P_{k+1}\right) = 0 \qquad \text{pour } i = 0, \ldots, n_k - m_k.$$

Nous avons besoin de considérer deux cas différents, selon les valeurs possibles de n_k et de m_k:

Cas $m_k \leq n_k + 1$

Posons

$$\begin{aligned} w_k(\xi) &= \beta_0 + \cdots + \beta_{m_k-1}\xi^{m_k-1} \\ v_k(\xi) &= \beta'_0 + \cdots + \beta'_{m_k-2}\xi^{m_k-2}. \end{aligned}$$

On impose les conditions d'orthogonalité de P_{k+1} pour $i = n_k - m_k + 1, \ldots, n_k + m_k - 1$ et l'on obtient le système suivant avec $2m_k - 1$ relations et $2m_k - 1$ inconnues.

Cas $m_k \leq n_k + 1$: $2m_k - 1$ équations avec:

m_k inconnues $\quad \beta_0, \ldots, \beta_{m_k-1}$
$m_k - 1$ inconnues $\quad \beta'_0, \ldots, \beta'_{m_k-2}$

$$
\begin{cases}
\beta_{m_k-1}\, c^{(1)}\left(\xi^{n_k} P_k^{(1)}\right) + \beta'_{m_k-2}\, c(\xi^{n_k} P_k) = 0 \\
\qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\beta_1\, c^{(1)}\left(\xi^{n_k} P_k^{(1)}\right) + \cdots + \beta_{m_k-1}\, c^{(1)}\left(\xi^{n_k+m_k-2} P_k^{(1)}\right) + \\
\quad \beta'_0\, c(\xi^{n_k} P_k) + \cdots + \beta'_{m_k-2}\, c(\xi^{n_k+m_k-2} P_k) = 0 \\
\beta_0\, c^{(1)}\left(\xi^{n_k} P_k^{(1)}\right) + \cdots + \beta_{m_k-1}\, c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right) + \\
\quad \beta'_0\, c(\xi^{n_k+1} P_k) + \cdots + \beta'_{m_k-2}\, c(\xi^{n_k+m_k-1} P_k) = c(\xi^{n_k} P_k) \\
\qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\beta_0\, c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right) + \cdots + \beta_{m_k-1}\, c^{(1)}\left(\xi^{n_k+2m_k-2} P_k^{(1)}\right) + \\
\quad \beta'_0\, c(\xi^{n_k+m_k} P_k) + \cdots + \beta'_{m_k-2}\, c(\xi^{n_k+2m_k-2} P_k) = c(\xi^{n_k+m_k-1} P_k)\,.
\end{cases}
$$

ou bien

$$
\begin{cases}
\beta_{m_k-1}\,(y, A^{n_k+1} z_k) + \beta'_{m_k-2}\,(y, A^{n_k} r_k) = 0 \\
\qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\beta_1\,(y, A^{n_k+1} z_k) + \cdots + \beta_{m_k-1}\,(y, A^{n_k+m_k-1} z_k) + \\
\quad \beta'_0\,(y, A^{n_k} r_k) + \cdots + \beta'_{m_k-2}\,(y, A^{n_k+m_k-2} r_k) = 0 \\
\beta_0\,(y, A^{n_k+1} z_k) + \cdots + \beta_{m_k-1}\,(y, A^{n_k+m_k} z_k) + \\
\quad \beta'_0\,(y, A^{n_k+1} r_k) + \cdots + \beta'_{m_k-2}\,(y, A^{n_k+m_k-1} r_k) = (y, A^{n_k} r_k) \\
\qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\beta_0\,(y, A^{n_k+m_k} z_k) + \cdots + \beta_{m_k-1}\,(y, A^{n_k+2m_k-1} z_k) + \\
\quad \beta'_0\,(y, A^{n_k+m_k} r_k) + \cdots + \beta'_{m_k-2}\,(y, A^{n_k+2m_k-2} r_k) = (y, A^{n_k+m_k-1} r_k)\,.
\end{cases}
$$

Évidemment, dans le cas du breakdown, $c^{(1)}\left(\xi^i P_k^{(1)}\right) = 0$ pour $i = n_k, \ldots, n_k + m_k - 2$, et donc les conditions pour $i = n_k - m_k + 1, \ldots, n_k - 1$ donnent $\beta'_0 = \cdots = \beta'_{m_k-2} = 0$ (même si $c(\xi^i P_k) = 0$ pour $i = n_k, \ldots, n_k + m_k - 2$ car dans ce cas les β'_i sont arbitraires) et les conditions pour $i = n_k, \ldots, n_k + m_k - 1$ sont les mêmes de celles du MRZ.

Cas $m_k > n_k + 1$

Posons

$$\begin{aligned} w_k(\xi) &= \beta_0 + \cdots + \beta_{m_k-1}\xi^{m_k-1} \\ v_k(\xi) &= \beta'_0 + \cdots + \beta'_{n_k-1}\xi^{n_k-1} \end{aligned}$$

car dans ce cas nous avons seulement $n_k + m_k < 2m_k - 1$ équations (les relations d'orthogonalité pour $i = 0, \ldots, n_k + m_k - 1$) pour déterminer les $2m_k - 1$ coefficients inconnus et donc l'on doit choisir un polynôme v_k de degré au plus $n_k - 1$.

Nous avons

Cas $m_k > n_k + 1$: $n_k + m_k$ équations avec:

m_k inconnues $\beta_0, \ldots, \beta_{m_k-1}$
n_k inconnues $\beta'_0, \ldots, \beta'_{n_k-1}$

$$\left\{\begin{array}{l} \beta_{n_k} c^{(1)}\left(\xi^{n_k} P_k^{(1)}\right) + \cdots + \beta_{m_k-1} c^{(1)}\left(\xi^{m_k-1} P_k^{(1)}\right) + \beta'_{n_k-1} c(\xi^{n_k} P_k) = 0 \\ \qquad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \beta_1 c^{(1)}\left(\xi^{n_k} P_k^{(1)}\right) + \cdots + \beta_{m_k-1} c^{(1)}\left(\xi^{n_k+m_k-2} P_k^{(1)}\right) + \\ \quad \beta'_0 c(\xi^{n_k} P_k) + \cdots + \beta'_{n_k-1} c(\xi^{2n_k-1} P_k) = 0 \\ \beta_0 c^{(1)}\left(\xi^{n_k} P_k^{(1)}\right) + \cdots + \beta_{m_k-1} c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right) + \\ \quad \beta'_0 c(\xi^{n_k+1} P_k) + \cdots + \beta'_{n_k-1} c(\xi^{2n_k} P_k) = c(\xi^{n_k} P_k) \\ \qquad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \beta_0 c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right) + \cdots + \beta_{m_k-1} c^{(1)}\left(\xi^{n_k+2m_k-2} P_k^{(1)}\right) + \\ \quad \beta'_0 c(\xi^{n_k+m_k} P_k) + \cdots + \beta'_{n_k-1} c(\xi^{2n_k+m_k-1} P_k) = c(\xi^{n_k+m_k-1} P_k) \end{array}\right.$$

et donc

$$\left\{\begin{array}{l} \beta_{n_k}(y, A^{n_k+1} z_k) + \cdots + \beta_{m_k-1}(y, A^{m_k} z_k) + \beta'_{n_k-1}(y, A^{n_k} r_k) = 0 \\ \qquad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \beta_1(y, A^{n_k+1} z_k) + \cdots + \beta_{m_k-1}(y, A^{n_k+m_k-1} z_k) + \\ \quad \beta'_0(y, A^{n_k} r_k) + \cdots + \beta'_{n_k-1}(y, A^{2n_k-1} r_k) = 0 \\ \beta_0(y, A^{n_k+1} z_k) + \cdots + \beta_{m_k-1}(y, A^{n_k+m_k} z_k) + \\ \quad \beta'_0(y, A^{n_k+1} r_k) + \cdots + \beta'_{n_k-1}(y, A^{2n_k} r_k) = (y, A^{n_k} r_k) \\ \qquad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \beta_0(y, A^{n_k+m_k} z_k) + \cdots + \beta_{m_k-1}(y, A^{n_k+2m_k-1} z_k) + \\ \quad \beta'_0(y, A^{n_k+m_k} r_k) + \cdots + \beta'_{n_k-1}(y, A^{2n_k+m_k-1} r_k) = (y, A^{n_k+m_k-1} r_k). \end{array}\right.$$

De nouveau, dans le cas du breakdown, les conditions pour $i = 0, \ldots, n_k - 1$ donnent $\beta'_0 = \cdots = \beta'_{n_k-1} = 0$ et on retrouve le MRZ.

Dans la suite on verra comment calculer récursivement les polynômes $P^{(1)}_{k+1}$.

4.3.1 GMRZ (General Method of Recursive Zoom)

D'abord on considère la possibilité de généraliser la relation à trois termes (4.6) du MRZ (G signifie *general*).

Soient $P^{(1)}_{k-1}$ et $P^{(1)}_k$ deux polynômes orthogonaux réguliers successifs par rapport à la fonctionnelle $c^{(1)}$. Successifs veut dire que tous les polynômes de degré $n_{k-1}+1, \ldots, n_k - 1$ n'existent pas. On cherche à calculer $\overline{P}^{(1)}_{k+1}$, qui est un polynôme orthogonal régulier de degré $n_{k+1} = n_k + m_k$ par rapport à la fonctionnelle $c^{(1)}$ mais qui n'est pas forcément le successeur de $P^{(1)}_k$ (ce qui signifie qu'il peut exister un polynôme régulier de degré entre $n_k + 1$ et $n_{k+1} - 1$) sous la forme

$$\overline{P}^{(1)}_{k+1}(\xi) \;=\; q_k(\xi)\, P^{(1)}_k(\xi) + t_k(\xi)\, P^{(1)}_{k-1}(\xi) \tag{4.10}$$

où q_k est un polynôme unitaire de degré m_k et t_k est un polynôme de degré au plus $m_k - 1$.

Après avoir déterminé $\overline{P}^{(1)}_{k+1}$ il faut déterminer le polynôme orthogonal régulier successeur ou prédecesseur pour pouvoir réappliquer la méthode à l'étape suivante.

Supposons appeler son prédecesseur $\overline{P}^{(1)}_k$. Il peut se calculer en imposant

$$\begin{aligned} & c^{(1)}\left(\xi^i \overline{P}^{(1)}_k\right) = 0 \qquad\qquad \text{pour } i = 0, \ldots, \overline{n}_k + \overline{m}_k - 2 \\ \text{et} \quad & c^{(1)}\left(\xi^{\overline{n}_k + \overline{m}_k - 1} \overline{P}^{(1)}_k\right) \neq 0 \end{aligned}$$

(donc le polynôme $\overline{P}^{(1)}_k$ a un degré compris entre $n_k + 1$ et $n_{k+1} - 1$, où n_{k+1} est le degré de $\overline{P}^{(1)}_{k+1}$).

Donc à l'étape suivante dans (4.10), à la place de $P^{(1)}_{k-1}$ et de $P^{(1)}_k$, on utilise respectivement $\overline{P}^{(1)}_k$ et $\overline{P}^{(1)}_{k+1}$. Cependant, à cause des conditions qui conduisent au *near-breakdown*, $\overline{P}^{(1)}_k$ sera mal calculé. C'est seulement si

$$\begin{aligned} & c^{(1)}\left(\xi^i \overline{P}^{(1)}_k\right) = 0 \qquad\qquad \text{pour } i = 0, \ldots, n_k + m_k - 2 \\ \text{et} \quad & c^{(1)}\left(\xi^{n_k + m_k - 1} \overline{P}^{(1)}_k\right) \neq 0 \end{aligned}$$

(donc quand les polynômes réguliers de degrés $n_k + 1, \ldots, n_{k+1} - 1$ n'existent pas) que l'on aura $\overline{P}^{(1)}_k = P^{(1)}_k$.

Donc il vaut mieux de calculer son successeur régulier que l'on appellera $\overline{P}^{(1)}_{k+2}$. Donc il faudra calculer m_{k+1} tel que

$$\begin{aligned} c^{(1)}\left(\xi^i \overline{P}^{(1)}_{k+1}\right) \;&=\; 0 \qquad \text{pour } i = 0, \ldots, n_{k+1} + m_{k+1} - 2 \\ &\neq\; 0 \qquad \text{pour } i = n_{k+1} + m_{k+1} - 1 \end{aligned}$$

et après calculer $\overline{P}_{k+2}^{(1)}$ avec la relation du GMRZ à partir de $P_k^{(1)}$ et $P_{k-1}^{(1)}$.

A l'étape suivante dans (4.10), à la place de $P_{k-1}^{(1)}$ et de $P_k^{(1)}$, l'on utilisera respectivement $\overline{P}_{k+1}^{(1)}$ et $\overline{P}_{k+2}^{(1)}$ qui seront deux polynômes réguliers successifs comme l'on doit avoir dans l'algorithme. Evidemment, à cause des erreurs d'arrondi, en général m_{k+1} sera égal à 1 puisque la quantité $\left|c^{(1)}\left(\xi^{n_{k+1}} P_{k+1}^{(1)}\right)\right|$ ne sera pas exactement égale à zéro.

Donc si l'on pose

$$z'_k \quad \text{successeur de} \quad z_k$$

il faudra d'abord calculer z_{k+1} par

$$z_{k+1} = q_k(A)\, z'_k + t_k(A) z_k$$

et après il faudra, pour pouvoir réappliquer l'algorithme, calculer

$$z'_{k+1} \quad \text{successeur de} \quad z_{k+1}$$

et cela peut encore se faire en utilisant z'_k et z_k avec la relation

$$z'_{k+1} = q'_k(A)\, z'_k + t'_k(A) z_k$$

où cette fois ci q'_k est (en général) un polynôme unitaire de degré $m_k + m_{k+1}$ et t'_k est un polynôme de degré au plus $m_k + m_{k+1} - 1$.

Donc finalement on aura

$$\begin{aligned} r_{k+1} &= r_k - A\, w_k(A)\, z'_k - A\, v_k(A) r_k \\ x_{k+1} &= x_k - w_k(A)\, z'_k - v_k(A) r_k \\ z_{k+1} &= q_k(A)\, z'_k + t_k(A) z_k \\ z'_{k+1} &= q'_k(A)\, z'_k + t'_k(A) z_k \end{aligned}$$

avec $z'_0 = r_0$, $z_0 = 0$ et donc l'algorithme (appelé GMRZ), peut s'écrire

Algorithme GMRZ

1. Choisir x_0 et y d'une façon arbitraire et
   choisir la valeur d'ε.
2. Poser $z'_0 = r_0 = Ax_0 - b$, $z_0 = 0$, $k = 0$, $n_0 = 0$.
3. Déterminer m_k de sorte que

$$\left|\left(y, A^{i+1} z'_k\right)\right| \le \varepsilon\,, \quad \text{pour } i = n_k, \ldots, n_k + m_k - 2$$

$$\text{et} \quad \left|\left(y, A^{n_k+m_k} z'_k\right)\right| > \varepsilon.$$

4. Calculer les coefficients de w_k et de v_k et poser
   $r_{k+1} = r_k - Aw_k(A)z'_k - Av_k(A)r_k$,
   $x_{k+1} = x_k - w_k(A)z'_k - v_k(A)r_k$.

5. Si le système est singulier, remplacer m_k avec $m_k + 1$ et retourner à 4.
6. Calculer les coefficients de q_k et de t_k et poser
   $z_{k+1} = q_k(A)z'_k + t_k(A)z_k$.
7. Si le système est singulier, remplacer m_k avec $m_k + 1$ et retourner à 4.
8. Déterminer m_{k+1} de sorte que

$$\left(y, A^{i+1}z_{k+1}\right) = 0, \quad \text{pour } i = n_{k+1}, \ldots, n_{k+1} + m_{k+1} - 2$$
$$\text{et} \quad \left(y, A^{n_{k+1}+m_{k+1}} z_{k+1}\right) \neq 0.$$

9. Calculer les coefficients de q'_k et de t'_k et poser
   $z'_{k+1} = q_k(A)z'_k + t_k(A)z_k$.
10. Si le système est singulier, remplacer m_{k+1} avec $m_{k+1} + 1$ et retourner à 9.
11. Si $r_{k+1} = 0$, alors $x_{k+1} = x$ et stop.
12. Poser $n_{k+1} = n_k + m_k + 1$.
13. Si $n_{k+1} < n$, alors remplacer k avec $k + 1$
    et retourner à 3, autrement stop.

4.3.2 Mise en œuvre de l'algorithme GMRZ

Nous avons

$$c^{(1)}\left(\xi^i P^{(1)}_{k+1}\right) = 0 \qquad \text{pour } i = 0, \ldots, n_k + m_k - 1$$

et comme l'on a

$$c^{(1)}\left(\xi^i\, q_k\, P^{(1)}_k\right) = c^{(1)}\left(\xi^i\, t_k\, P^{(1)}_{k-1}\right) = 0 \quad \text{pour } i = 0, \ldots, n_k - m_k - 1,$$

les conditions d'orthogonalité pour $P^{(1)}_{k+1}$ sont satisfaites pour les mêmes indices.

Considérons, comme d'habitude, deux cas différents selon les valeurs possibles de m_k et de n_k:

Cas $m_k \leq n_k$

Posons

$$\begin{aligned} q_k(\xi) &= \alpha_0 + \cdots + \alpha_{m_k-1}\xi^{m_k-1} + \xi^{m_k} \\ t_k(\xi) &= \alpha'_0 + \cdots + \alpha'_{m_k-1}\xi^{m_k-1}. \end{aligned}$$

En imposant les conditions d'orthogonalité pour $i = n_k - m_k, \ldots, n_k + m_k - 1$ on aura

$$\begin{aligned} c^{(1)}\left(\xi^i P^{(1)}_{k+1}\right) = {} & \alpha_0\, c^{(1)}\left(\xi^i P^{(1)}_k\right) + \cdots + \alpha_{m_k-1}\, c^{(1)}\left(\xi^{i+m_k-1} P^{(1)}_k\right) + c^{(1)}\left(\xi^{i+m_k} P^{(1)}_k\right) + \\ & \alpha'_0\, c^{(1)}\left(\xi^i P^{(1)}_{k-1}\right) + \cdots + \alpha'_{m_k-1}\, c^{(1)}\left(\xi^{i+m_k-1} P^{(1)}_{k-1}\right) = 0 \end{aligned}$$

ce qui donne $2m_k$ relations pour la détermination des $2m_k$ coefficients inconnus.

Cas $m_k \leq n_k$: $2m_k$ équations avec:

m_k inconnues $\alpha_0, \ldots, \alpha_{m_k-1}$
m_k inconnues $\alpha'_0, \ldots, \alpha'_{m_k-1}$

$$
\begin{cases}
\alpha'_{m_k-1}\, c^{(1)}\left(\xi^{n_k-1} P_{k-1}^{(1)}\right) = -c^{(1)}\left(\xi^{n_k} P_k^{(1)}\right) \\
\alpha_{m_k-1}\, c^{(1)}\left(\xi^{n_k} P_k^{(1)}\right) + \alpha'_{m_k-2}\, c^{(1)}\left(\xi^{n_k-1} P_{k-1}^{(1)}\right) + \alpha'_{m_k-1}\, c(\xi^{n_k+1} P_k) = \\
\qquad - c^{(1)}\left(\xi^{n_k+1} P_k^{(1)}\right) \\
\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\alpha_1\, c^{(1)}\left(\xi^{n_k} P_k^{(1)}\right) + \cdots + \alpha_{m_k-1}\, c^{(1)}\left(\xi^{n_k+m_k-2} P_k^{(1)}\right) + \alpha'_0\, c^{(1)}\left(\xi^{n_k-1} P_{k-1}^{(1)}\right) + \cdots + \\
\qquad \alpha'_{m_k-1}\, c^{(1)}\left(\xi^{n_k+m_k-2} P_{k-1}^{(1)}\right) = -c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right) \\
\alpha_0\, c^{(1)}\left(\xi^{n_k} P_k^{(1)}\right) + \cdots + \alpha_{m_k-1}\, c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right) + \alpha'_0\, c^{(1)}\left(\xi^{n_k} P_{k-1}^{(1)}\right) + \cdots + \\
\qquad \alpha'_{m_k-1}\, c^{(1)}\left(\xi^{n_k+m_k-1} P_{k-1}^{(1)}\right) = -c^{(1)}\left(\xi^{n_k+m_k} P_k^{(1)}\right) \\
\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\alpha_0\, c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right) + \cdots + \alpha_{m_k-1}\, c^{(1)}\left(\xi^{n_k+2m_k-2} P_k^{(1)}\right) + \\
\quad \alpha'_0\, c^{(1)}\left(\xi^{n_k+m_k-1} P_{k-1}^{(1)}\right) + \cdots + \alpha'_{m_k-1}\, c^{(1)}\left(\xi^{n_k+2m_k-2} P_{k-1}^{(1)}\right) = -c^{(1)}\left(\xi^{n_k+2m_k-1} P_k^{(1)}\right)
\end{cases}
$$

ou encore

$$
\begin{cases}
\alpha'_{m_k-1}\,(y, A^{n_k} z_k) = -\,(y, A^{n_k+1} z'_k) \\
\alpha_{m_k-1}\,(y, A^{n_k+1} z'_k) + \alpha'_{m_k-2}\,(y, A^{n_k} z_k) = \\
\qquad -\alpha'_{m_k-1}\,(y, A^{n_k+1} z_k) - (y, A^{n_k+2} z'_k) \\
\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\alpha_1\,(y, A^{n_k+1} z'_k) + \cdots + \alpha_{m_k-1}\,(y, A^{n_k+m_k-1} z'_k) + \\
\quad \alpha'_0\,(y, A^{n_k} z_k) + \cdots + \alpha'_{m_k-2}\,(y, A^{n_k+m_k-2} z_k) = \\
\qquad -\alpha'_{m_k-1}\,(y, A^{n_k+m_k-1} z_k) - (y, A^{n_k+m_k} z'_k) \\
\alpha_0\,(y, A^{n_k+1} z'_k) + \cdots + \alpha_{m_k-1}\,(y, A^{n_k+m_k} z'_k) + \\
\quad \alpha'_0\,(y, A^{n_k+1} z_k) + \cdots + \alpha'_{m_k-2}\,(y, A^{n_k+m_k-1} z_k) = \\
\qquad -\alpha'_{m_k-1}\,(y, A^{n_k+m_k} z_k) - (y, A^{n_k+m_k+1} z'_k) \\
\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
\alpha_0\,(y, A^{n_k+m_k} z'_k) + \cdots + \alpha_{m_k-1}\,(y, A^{n_k+2m_k-1} z'_k) + \\
\quad \alpha'_0\,(y, A^{n_k+m_k} z_k) + \cdots + \alpha'_{m_k-2}\,(y, A^{n_k+2m_k-2} z_k) = \\
\qquad -\alpha'_{m_k-1}\,(y, A^{n_k+2m_k-1} z_k) - (y, A^{n_k+2m_k} z'_k)\,.
\end{cases}
$$

Dans le cas du breakdown on retrouve le MRZ.

Cas $m_k > n_k$

Posons

$$\begin{aligned} q_k(\xi) &= \alpha_0 + \cdots + \alpha_{m_k-1}\xi^{m_k-1} + \xi^{m_k} \\ t_k(\xi) &= \alpha'_0 + \cdots + \alpha'_{n_k-1}\xi^{n_k-1} \end{aligned}$$

car, dans ce cas, on a $n_k + m_k < 2m_k$ équations pour calculer les $2m_k$ inconnues.

En imposant les conditions d'orthogonalité de $P_{k+1}^{(1)}$ pour $i = 0, \ldots, n_k + m_k - 1$ on aura le système suivant

Cas $m_k > n_k$: $n_k + m_k$ équations avec:

m_k inconnues $\alpha_0, \ldots, \alpha_{m_k-1}$
n_k inconnues $\alpha'_0, \ldots, \alpha'_{n_k-1}$

$$\left\{\begin{array}{l} \alpha_{n_k}\, c^{(1)}\left(\xi^{n_k} P_k^{(1)}\right) + \cdots + \alpha_{m_k-1}\, c^{(1)}\left(\xi^{m_k-1} P_k^{(1)}\right) + c^{(1)}\left(\xi^{m_k} P_k^{(1)}\right) + \\ \quad \alpha'_{n_k-1}\, c^{(1)}\left(\xi^{n_k-1} P_{k-1}^{(1)}\right) = 0 \\ \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \\ \alpha_1\, c^{(1)}\left(\xi^{n_k} P_k^{(1)}\right) + \cdots + \alpha_{m_k-1}\, c^{(1)}\left(\xi^{n_k+m_k-2} P_k^{(1)}\right) + c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right) + \\ \quad \alpha'_0\, c^{(1)}\left(\xi^{n_k-1} P_{k-1}^{(1)}\right) + \cdots + \alpha'_{n_k-1}\, c^{(1)}\left(\xi^{2n_k-2} P_{k-1}^{(1)}\right) = 0 \\ \\ \alpha_0\, c^{(1)}\left(\xi^{n_k} P_k^{(1)}\right) + \cdots + \alpha_{m_k-1}\, c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right) + c^{(1)}\left(\xi^{n_k+m_k} P_k^{(1)}\right) + \\ \quad \alpha'_0\, c^{(1)}\left(\xi^{n_k} P_{k-1}^{(1)}\right) + \cdots + \alpha'_{n_k-1}\, c^{(1)}\left(\xi^{2n_k-1} P_{k-1}^{(1)}\right) = 0 \\ \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \\ \alpha_0\, c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right) + \cdots + \alpha_{m_k-1}\, c^{(1)}\left(\xi^{n_k+2m_k-2} P_k^{(1)}\right) + c^{(1)}\left(\xi^{n_k+2m_k-1} P_k^{(1)}\right) + \\ \quad \alpha'_0\, c^{(1)}\left(\xi^{n_k+m_k-1} P_{k-1}^{(1)}\right) + \cdots + \alpha'_{n_k-1}\, c^{(1)}\left(\xi^{2n_k+m_k-2} P_{k-1}^{(1)}\right) = 0 \end{array}\right.$$

ou

$$\left\{\begin{array}{l}
\alpha_{n_k}(y, A^{n_k+1}z'_k)+\cdots+\alpha_{m_k-1}(y, A^{m_k}z'_k)+ \\
\qquad\qquad \alpha'_{n_k-1}(y, A^{n_k}z_k) = -(y, A^{m_k+1}z'_k) \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
\alpha_1(y, A^{n_k+1}z'_k)+\cdots+\alpha_{m_k-1}(y, A^{n_k+m_k-1}z'_k)+ \\
\quad \alpha'_0(y, A^{n_k}z_k)+\cdots+\alpha'_{n_k-1}(y, A^{2n_k-1}z_k) = -(y, A^{n_k+m_k}z'_k) \\
\alpha_0(y, A^{n_k+1}z'_k)+\cdots+\alpha_{m_k-1}(y, A^{n_k+m_k}z'_k)+ \\
\quad \alpha'_0(y, A^{n_k+1}z_k)+\cdots+\alpha'_{n_k-1}(y, A^{2n_k}z_k) = -(y, A^{n_k+m_k+1}z'_k) \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
\alpha_0(y, A^{n_k+m_k}z'_k)+\cdots+\alpha_{m_k-1}(y, A^{n_k+2m_k-1}z'_k)+ \\
\quad \alpha'_0(y, A^{n_k+m_k}z_k)+\cdots+\alpha'_{n_k-1}(y, A^{2n_k+m_k-1}z_k) = -(y, A^{n_k+2m_k}z'_k).
\end{array}\right.$$

S'il y a un breakdown, alors $\alpha'_1 = \cdots = \alpha'_{n_k-1}$ et l'on retrouve le MRZ.

Si les systèmes qui donnent $\overline{P}^{(1)}_{k+1}$ sont singuliers, alors $\overline{P}^{(1)}_{k+1}$ n'existe pas et il faut augmenter la valeur de m_k jusqu'à que l'on trouve un polynôme regulier $\overline{P}^{(1)}_{k+1}$ qui existe.

En ce qui concerne le calcul des coefficients des polynômes p'_k et t'_k il sera suffisant d'utiliser les mêmes systèmes qu'avant où, à la place de la valeur de m_k, on considère la valeur m_{k+1}.

Cet algorithme n'a pas encore été codé.

4.3.3 BSMRZ (Block Symmetric Method of Recursive Zoom)

L'on cherche à généraliser le SMRZ et les $P^{(1)}_{k+1}(\xi)$ sont pris sous la forme

$$P^{(1)}_{k+1}(\xi) \;=\; q_k(\xi)\,P^{(1)}_k(\xi) + t_k(\xi)\,P_k(\xi) \tag{4.11}$$

où q_k est un polynôme unitaire de degré m_k et t_k un polynôme de degré au plus $m_k - 1$.

Maintenant on aura

$$\begin{aligned}
r_{k+1} &= r_k - A\,w_k(A)\,z_k - A\,v_k(A)r_k \\
x_{k+1} &= x_k - w_k(A)\,z_k - v_k(A)r_k \\
z_{k+1} &= q_k(A)\,z_k + t_k(A)r_k
\end{aligned}$$

avec $z_0 = r_0$ et donc l'algorithme (appelé BSMRZ, où B signifie *block*) peut s'écrire

Algorithme BSMRZ

1. Choisir x_0 et y d'une façon arbitraire et
   choisir la valeur d'ε.

2. Poser $z_0 = r_0 = Ax_0 - b$, $k = 0$, $n_0 = 0$.
3. Déterminer m_k de façon que
$$\left|\left(y, A^{i+1} z_k\right)\right| \leq \varepsilon, \quad \text{pour } i = n_k, \ldots, n_k + m_k - 2$$
$$\text{et} \quad \left|\left(y, A^{n_k+m_k} z_k\right)\right| > \varepsilon.$$
4. Calculer les coefficients de w_k et de v_k et poser
$r_{k+1} = r_k - Aw_k(A)z_k - Av_k(A)r_k$,
$x_{k+1} = x_k - w_k(A)z_k - v_k(A)r_k$.
5. Si le système est singulier, remplacer m_k avec $m_k + 1$ et retourner à 4.
6. Calculer les coefficients de q_k et de t_k et poser
$z_{k+1} = q_k(A)z_k + t_k(A)r_k$.
7. Si le système est singulier, remplacer m_k avec $m_k + 1$ et retourner à 4.
8. Si $r_{k+1} = 0$, alors $x_{k+1} = x$ et stop.
9. Poser $n_{k+1} = n_k + m_k$.
10. Si $n_{k+1} < n$, alors remplacer k avec $k + 1$
et retourner à 3, autrement stop.

Cette méthode (comme le SMRZ) ne peut pas être utilisée quand

$$c(\xi^{n_k} P_k) = (y, A^{n_k} r_k) = 0.$$

Dans ce cas là il faut utiliser le GMRZ.

4.3.4 Mise en œuvre de l'algorithme BSMRZ

En revenant au calcul des $2m_k$ coefficients inconnus des polynômes q_k et t_k, la relation (4.11) nous donne

$$c^{(1)}\left(\xi^i P_{k+1}^{(1)}\right) = c^{(1)}\left(\xi^i q_k P_k^{(1)}\right) + c\left(\xi^{i+1} t_k P_k\right).$$

Comme on a

$$c^{(1)}\left(\xi^i q_k P_k^{(1)}\right) = c\left(\xi^{i+1} t_k P_k\right) = 0 \quad \text{pour } i = 0, \ldots, n_k - m_k - 1,$$

les conditions d'orthogonalité pour $P_{k+1}^{(1)}$ sont satisfaites pour les mêmes indices.

Encore cette fois ci nous avons besoin de considérer deux cas différents, selon les valeurs possibles de n_k et de m_k:

Cas $m_k \leq n_k$

Posons

$$\begin{aligned} q_k(\xi) &= \alpha_0 + \cdots + \alpha_{m_k-1}\xi^{m_k-1} + \xi^{m_k} \\ t_k(\xi) &= \alpha'_0 + \cdots + \alpha'_{m_k-1}\xi^{m_k-1} \end{aligned}$$

et donc

$$\begin{aligned} c^{(1)}\left(\xi^i P_{k+1}^{(1)}\right) &= \alpha_0\, c^{(1)}\left(\xi^i P_k^{(1)}\right) + \cdots + \alpha_{m_k-1}\, c^{(1)}\left(\xi^{i+m_k-1} P_k^{(1)}\right) + c^{(1)}\left(\xi^{i+m_k} P_k^{(1)}\right) + \\ &\qquad \alpha'_0\, c\left(\xi^{i+1} P_k\right) + \cdots + \alpha'_{m_k-1}\, c\left(\xi^{i+m_k} P_k\right). \end{aligned}$$

On impose les conditions d'orthogonalité de $P_{k+1}^{(1)}$ pour $i = n_k - m_k, \ldots, n_k + m_k - 1$ et l'on obtient le système suivant avec $2m_k$ relations et $2m_k$ inconnues.

Cas $m_k \leq n_k$: $2m_k$ équations avec:

m_k inconnues $\alpha_0, \ldots, \alpha_{m_k-1}$
m_k inconnues $\alpha'_0, \ldots, \alpha'_{m_k-1}$

$$\left\{\begin{array}{l} \alpha'_{m_k-1}\, c(\xi^{n_k} P_k) = -c^{(1)}\left(\xi^{n_k} P_k^{(1)}\right) \\ \\ \alpha_{m_k-1}\, c^{(1)}\left(\xi^{n_k} P_k^{(1)}\right) + \alpha'_{m_k-2}\, c(\xi^{n_k} P_k) + \alpha'_{m_k-1}\, c(\xi^{n_k+1} P_k) = \\ \qquad -c^{(1)}\left(\xi^{n_k+1} P_k^{(1)}\right) \\ \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \\ \alpha_1\, c^{(1)}\left(\xi^{n_k} P_k^{(1)}\right) + \cdots + \alpha_{m_k-1}\, c^{(1)}\left(\xi^{n_k+m_k-2} P_k^{(1)}\right) + \alpha'_0\, c(\xi^{n_k} P_k) + \cdots + \\ \qquad \alpha'_{m_k-1}\, c(\xi^{n_k+m_k-1} P_k) = -c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right) \\ \\ \alpha_0\, c^{(1)}\left(\xi^{n_k} P_k^{(1)}\right) + \cdots + \alpha_{m_k-1}\, c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right) + \alpha'_0\, c(\xi^{n_k+1} P_k) + \cdots + \\ \qquad \alpha'_{m_k-1}\, c(\xi^{n_k+m_k} P_k) = -c^{(1)}\left(\xi^{n_k+m_k} P_k^{(1)}\right) \\ \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \\ \alpha_0\, c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right) + \cdots + \alpha_{m_k-1}\, c^{(1)}\left(\xi^{n_k+2m_k-2} P_k^{(1)}\right) + \\ \qquad \alpha'_0\, c(\xi^{n_k+m_k} P_k) + \cdots + \alpha'_{m_k-1}\, c(\xi^{n_k+2m_k-1} P_k) = -c^{(1)}\left(\xi^{n_k+2m_k-1} P_k^{(1)}\right) \end{array}\right.$$

ou encore

$$\left\{\begin{array}{l}
\alpha'_{m_k-1}(y, A^{n_k} r_k) = -(y, A^{n_k+1} z_k) \\
\\
\alpha_{m_k-1}(y, A^{n_k+1} z_k) + \alpha'_{m_k-2}(y, A^{n_k} r_k) = \\
\qquad -\alpha'_{m_k-1}(y, A^{n_k+1} r_k) - (y, A^{n_k+2} z_k) \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
\\
\alpha_1(y, A^{n_k+1} z_k) + \cdots + \alpha_{m_k-1}(y, A^{n_k+m_k-1} z_k) + \\
\quad \alpha'_0(y, A^{n_k} r_k) + \cdots + \alpha'_{m_k-2}(y, A^{n_k+m_k-2} r_k) = \\
\qquad -\alpha'_{m_k-1}(y, A^{n_k+m_k-1} r_k) - (y, A^{n_k+m_k} z_k) \\
\\
\alpha_0(y, A^{n_k+1} z_k) + \cdots + \alpha_{m_k-1}(y, A^{n_k+m_k} z_k) + \\
\quad \alpha'_0(y, A^{n_k+1} r_k) + \cdots + \alpha'_{m_k-2}(y, A^{n_k+m_k-1} r_k) = \\
\qquad -\alpha'_{m_k-1}(y, A^{n_k+m_k} r_k) - (y, A^{n_k+m_k+1} z_k) \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
\\
\alpha_0(y, A^{n_k+m_k} z_k) + \cdots + \alpha_{m_k-1}(y, A^{n_k+2m_k-1} z_k) + \\
\quad \alpha'_0(y, A^{n_k+m_k} r_k) + \cdots + \alpha'_{m_k-2}(y, A^{n_k+2m_k-2} r_k) = \\
\qquad -\alpha'_{m_k-1}(y, A^{n_k+2m_k-1} r_k) - (y, A^{n_k+2m_k} z_k).
\end{array}\right.$$

Dans le cas du breakdown on retrouve le SMRZ et si $c(\xi^{n_k} P_k) = 0$ alors le système est singulier et (comme nous avons déjà dit) il faut utiliser le GMRZ. Mais dans la pratique $|c(\xi^{n_k} P_k)|$ ne sera jamais exactement nul mais seulement petit.

Cas $m_k > n_k$

Posons

$$\begin{aligned}
q_k(\xi) &= \alpha_0 + \cdots + \alpha_{m_k-1}\xi^{m_k-1} + \xi^{m_k} \\
t_k(\xi) &= \alpha'_0 + \cdots + \alpha'_{n_k-1}\xi^{n_k-1}
\end{aligned}$$

car, dans ce cas, on a $n_k + m_k < 2m_k$ équations pour calculer les $2m_k$ inconnues.

Nous avons

$$\begin{aligned}
c^{(1)}\left(\xi^i P^{(1)}_{k+1}\right) = {} & \alpha_0\, c^{(1)}\left(\xi^i P^{(1)}_k\right) + \cdots + \alpha_{m_k-1}\, c^{(1)}\left(\xi^{i+m_k-1} P^{(1)}_k\right) + c^{(1)}\left(\xi^{i+m_k} P^{(1)}_k\right) + \\
& \alpha'_0\, c\left(\xi^{i+1} P_k\right) + \cdots + \alpha'_{n_k-1}\, c\left(\xi^{i+n_k} P_k\right)
\end{aligned}$$

et en imposant les conditions d'orthogonalité de $P^{(1)}_{k+1}$ pour $i = 0, \ldots, n_k + m_k - 1$ on aura le système suivant

Cas $m_k > n_k$: $n_k + m_k$ équations avec:

m_k inconnues $\alpha_0, \ldots, \alpha_{m_k-1}$
n_k inconnues $\alpha'_0, \ldots, \alpha'_{n_k-1}$

$$
\begin{cases}
\alpha_{n_k}\, c^{(1)}\left(\xi^{n_k} P_k^{(1)}\right) + \cdots + \alpha_{m_k-1}\, c^{(1)}\left(\xi^{m_k-1} P_k^{(1)}\right) + c^{(1)}\left(\xi^{m_k} P_k^{(1)}\right) + \\
\quad \alpha'_{n_k-1}\, c(\xi^{n_k} P_k) = 0 \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
\alpha_1\, c^{(1)}\left(\xi^{n_k} P_k^{(1)}\right) + \cdots + \alpha_{m_k-1}\, c^{(1)}\left(\xi^{n_k+m_k-2} P_k^{(1)}\right) + c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right) + \\
\quad \alpha'_0\, c(\xi^{n_k} P_k) + \cdots + \alpha'_{n_k-1}\, c(\xi^{2n_k-1} P_k) = 0 \\
\alpha_0\, c^{(1)}\left(\xi^{n_k} P_k^{(1)}\right) + \cdots + \alpha_{m_k-1}\, c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right) + c^{(1)}\left(\xi^{n_k+m_k} P_k^{(1)}\right) + \\
\quad \alpha'_0\, c(\xi^{n_k+1} P_k) + \cdots + \alpha'_{n_k-1}\, c(\xi^{2n_k} P_k) = 0 \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
\alpha_0\, c^{(1)}\left(\xi^{n_k+m_k-1} P_k^{(1)}\right) + \cdots + \alpha_{m_k-1}\, c^{(1)}\left(\xi^{n_k+2m_k-2} P_k^{(1)}\right) + c^{(1)}\left(\xi^{n_k+2m_k-1} P_k^{(1)}\right) + \\
\quad \alpha'_0\, c(\xi^{n_k+m_k} P_k) + \cdots + \alpha'_{n_k-1}\, c(\xi^{2n_k+m_k-1} P_k) = 0
\end{cases}
$$

ou

$$
\begin{cases}
\alpha_{n_k}\,(y, A^{n_k+1} z_k) + \cdots + \alpha_{m_k-1}\,(y, A^{m_k} z_k) + \\
\qquad \alpha'_{n_k-1}\,(y, A^{n_k} r_k) = -(y, A^{m_k+1} z_k) \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
\alpha_1\,(y, A^{n_k+1} z_k) + \cdots + \alpha_{m_k-1}\,(y, A^{n_k+m_k-1} z_k) + \\
\quad \alpha'_0\,(y, A^{n_k} r_k) + \cdots + \alpha'_{n_k-1}\,(y, A^{2n_k-1} r_k) = -(y, A^{n_k+m_k} z_k) \\
\alpha_0\,(y, A^{n_k+1} z_k) + \cdots + \alpha_{m_k-1}\,(y, A^{n_k+m_k} z_k) + \\
\quad \alpha'_0\,(y, A^{n_k+1} r_k) + \cdots + \alpha'_{n_k-1}\,(y, A^{2n_k} r_k) = -(y, A^{n_k+m_k+1} z_k) \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
\alpha_0\,(y, A^{n_k+m_k} z_k) + \cdots + \alpha_{m_k-1}\,(y, A^{n_k+2m_k-1} z_k) + \\
\quad \alpha'_0\,(y, A^{n_k+m_k} r_k) + \cdots + \alpha'_{n_k-1}\,(y, A^{2n_k+m_k-1} r_k) = -(y, A^{n_k+2m_k} z_k)\,.
\end{cases}
$$

S'il y a un breakdown, alors $\alpha'_1 = \cdots = \alpha'_{n_k-1}$ et l'on retrouve le SMRZ.

Donc on peut présenter un résumé des choix possibles:

$$w_k(\xi) = \beta_0 + \cdots + \beta_{m_k-1}\xi^{m_k-1}$$

$$v_k(\xi) = \beta'_0 + \cdots + \beta'_{m_k-2}\xi^{m_k-2} \qquad \text{pour } m_k \leq n_k$$

$$v_k(\xi) = \beta'_0 + \cdots + \beta'_{n_k-1}\xi^{n_k-1} \qquad \text{pour } m_k > n_k$$

$$q_k(\xi) = \alpha_0 + \cdots + \alpha_{m_k-1}\xi^{m_k-1} + \xi^{m_k}$$

$$t_k(\xi) = \alpha'_0 + \cdots + \alpha'_{m_k-1}\xi^{m_k-1} \qquad \text{pour } m_k \leq n_k$$

$$t_k(\xi) = \alpha'_0 + \cdots + \alpha'_{n_k-1}\xi^{n_k-1} \qquad \text{pour } m_k > n_k.$$

Comme l'on voit, le cas $m_k = n_k + 1$ a été inséré dans le cas $m_k > n_k$ car les deux systèmes qui calculent les β_i et les β'_i pour cette valeur sont coïncidants et cela nous va permettre d'unifier ces systèmes avec ceux qui calculent les α_i et les α'_i.

Si nous posons

$$\begin{aligned} d_i &= \left(y, A^{n_k+i} r_k\right) \\ c_i &= \left(y, A^{n_k+1+i} z_k\right) \end{aligned}$$

les quatre systèmes deviennent:

Systèmes pour les β et les β'

Cas $m_k \leq n_k$: $2m_k - 1$ équations avec:

m_k inconnues $\quad \beta_0, \ldots, \beta_{m_k-1}$
$m_k - 1$ inconnues $\quad \beta'_0, \ldots, \beta'_{m_k-2}$

$$\begin{cases} \beta_{m_k-1}c_0 + \beta'_{m_k-2}d_0 = 0 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \beta_1 c_0 + \cdots + \beta_{m_k-1}c_{m_k-2} + \beta'_0 d_0 + \cdots + \beta'_{m_k-2}d_{m_k-2} = 0 \\ \beta_0 c_0 + \cdots + \beta_{m_k-1}c_{m_k-1} + \beta'_0 d_1 + \cdots + \beta'_{m_k-2}d_{m_k-1} = d_0 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \beta_0 c_{m_k-1} + \cdots + \beta_{m_k-1}c_{2m_k-2} + \beta'_0 d_{m_k} + \cdots + \beta'_{m_k-2}d_{2m_k-2} = d_{m_k-1} \end{cases}$$

Cas $m_k > n_k$: $n_k + m_k$ équations avec:

m_k inconnues $\quad \beta_0, \ldots, \beta_{m_k-1}$
n_k inconnues $\quad \beta'_0, \ldots, \beta'_{n_k-1}$

$$\begin{cases} \beta_{n_k} c_0 + \cdots + \beta_{m_k-1} c_{m_k-n_k-1} + \beta'_{n_k-1} d_0 = 0 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \beta_1 c_0 + \cdots + \beta_{m_k-1} c_{m_k-2} + \beta'_0 d_0 + \cdots + \beta'_{n_k-1} d_{n_k-1} = 0 \\ \beta_0 c_0 + \cdots + \beta_{m_k-1} c_{m_k-1} + \beta'_0 d_1 + \cdots + \beta'_{n_k-1} d_{n_k} = d_0 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \beta_0 c_{m_k-1} + \cdots + \beta_{m_k-1} c_{2m_k-2} + \beta'_0 d_{m_k} + \cdots + \beta'_{n_k-1} d_{n_k+m_k-1} = d_{m_k-1} \end{cases}$$

Systèmes pour les α et les α'

Cas $m_k \leq n_k$: $2m_k$ équations avec:

m_k inconnues $\alpha_0, \ldots, \alpha_{m_k-1}$
m_k inconnues $\alpha'_0, \ldots, \alpha'_{m_k-1}$

$$\begin{cases} \alpha'_{m_k-1} d_0 = -c_0 \\ \alpha_{m_k-1} c_0 + \alpha'_{m_k-2} d_0 = -\alpha'_{m_k-1} d_1 - c_1 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \alpha_1 c_0 + \cdots + \alpha_{m_k-1} c_{m_k-2} + \alpha'_0 d_0 + \cdots + \alpha'_{m_k-2} d_{m_k-2} = \\ \qquad\qquad -\alpha'_{m_k-1} d_{m_k-1} - c_{m_k-1} \\ \alpha_0 c_0 + \cdots + \alpha_{m_k-1} c_{m_k-1} + \alpha'_0 d_0 + \cdots + \alpha'_{m_k-2} d_{m_k-1} = \\ \qquad\qquad -\alpha'_{m_k-1} d_{m_k} - c_{m_k} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \alpha_0 c_{m_k-1} + \cdots + \alpha_{m_k-1} c_{2m_k-2} + \alpha'_0 d_{m_k} + \cdots + \alpha'_{m_k-2} d_{2m_k-2} = \\ \qquad\qquad -\alpha'_{m_k-1} d_{2m_k-1} - c_{2m_k-1} \end{cases}$$

Cas $m_k > n_k$: $n_k + m_k$ équations avec:

m_k inconnues $\alpha_0, \ldots, \alpha_{m_k-1}$
n_k inconnues $\alpha'_0, \ldots, \alpha'_{n_k-1}$

$$\begin{cases} \alpha_{n_k}c_0+\cdots+\alpha_{m_k-1}c_{m_k-n_k-1}+\alpha'_{n_k-1}d_0=-c_{m_k-n_k} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \alpha_1 c_0+\cdots+\alpha_{m_k-1}c_{m_k-2}+\alpha'_0 d_0+\cdots+\alpha'_{n_k-1}d_{n_k-1}=-c_{m_k-1} \\ \alpha_0 c_0+\cdots+\alpha_{m_k-1}c_{m_k-1}+\alpha'_0 d_0+\cdots+\alpha'_{n_k-1}d_{n_k}=-c_{m_k} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \alpha_0 c_{m_k-1}+\cdots+\alpha_{m_k-1}c_{2m_k-2}+\alpha'_0 d_{m_k}+\cdots+\alpha'_{n_k-1}d_{n_k+m_k-1}=-c_{2m_k-1}\,. \end{cases}$$

D'après l'analyse de ces systèmes on peut considérer les schémas suivants

Système pour le cas $m_k \leq n_k$

Système de $2m_k - 1$ équations pour β et β'
Système de $2m_k - 1$ équations pour α et α'
$(\alpha'_{m_k-1} = -c_0/d_0)$

β_0 α_0 $\downarrow$	β_1 α_1 $\downarrow$	$\cdots$	β_{m_k-1} α_{m_k-1} $\downarrow$	β'_0 α'_0 $\downarrow$	$\cdots$	β'_{m_k-2} α'_{m_k-2} $\downarrow$	pour β,β' $\downarrow$	pour α,α' $\downarrow$
c_{m_k-1}	c_{m_k}	$\cdots$	c_{2m_k-2}	d_{m_k}	$\cdots$	d_{2m_k-2}	d_{m_k-1}	$-c_{2m_k-1}+d_{2m_k-1}c_0/d_0$
c_{m_k-2}	c_{m_k-1}	$\cdots$	c_{2m_k-3}	d_{m_k-1}	$\cdots$	d_{2m_k-3}	d_{m_k-2}	$-c_{2m_k-2}+d_{2m_k-2}c_0/d_0$
$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$
c_0	c_1	$\cdots$	c_{m_k-1}	d_1	$\cdots$	d_{m_k-1}	d_0	$-c_{m_k}+d_{m_k}c_0/d_0$
	c_0	$\cdots$	c_{m_k-2}	d_0	$\cdots$	d_{m_k-2}	0	$-c_{m_k-1}+d_{m_k-2}c_0/d_0$
		$\ddots$	$\vdots$		$\ddots$	$\vdots$	$\vdots$	$\vdots$
	0		c_1	0		d_1	$\vdots$	$-c_2+d_2c_0/d_0$
			c_0			d_0	0	$-c_1+d_1c_0/d_0$

Système pour le cas $m_k > n_k$

Système de $n_k + m_k$ équations pour β et β'
Système de $n_k + m_k$ équations pour α et α'

β_0	β_1	$\cdots$	β_{n_k}	β_{n_k+1}	$\cdots$	β_{m_k-1}	β'_0	$\cdots$	β'_{n_k-1}	pour β,β'	pour α,α'
α_0	α_1	$\cdots$	α_{n_k}	α_{n_k+1}	$\cdots$	α_{m_k-1}	α'_0	$\cdots$	α'_{n_k-1}	β,β'	α,α'
$\downarrow$	$\downarrow$		$\downarrow$	$\downarrow$		$\downarrow$	$\downarrow$		$\downarrow$	$\downarrow$	$\downarrow$
c_{m_k-1}	c_{m_k}	$\cdots$	$c_{n_k+m_k-1}$	$c_{n_k+m_k}$	$\cdots$	c_{2m_k-2}	d_{m_k}	$\cdots$	$d_{n_k+m_k-1}$	d_{m_k-1}	$-c_{2m_k-1}$
c_{m_k-2}	c_{m_k-1}	$\cdots$	$c_{n_k+m_k-2}$	$c_{n_k+m_k-1}$	$\cdots$	c_{2m_k-3}	d_{m_k-1}	$\cdots$	$d_{n_k+m_k-2}$	d_{m_k-2}	$-c_{2m_k-2}$
$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$
c_0	c_1	$\cdots$	c_{n_k-1}	c_{n_k}	$\cdots$	c_{m_k-1}	d_1	$\cdots$	d_{n_k}	d_0	$-c_{m_k+1}$
	c_0	$\cdots$	c_{n_k}	c_{n_k+1}	$\cdots$	c_{m_k-2}	d_0	$\cdots$	d_{n_k-1}	0	$-c_{m_k}$
		$\ddots$	$\vdots$	$\vdots$		$\vdots$		$\ddots$	$\vdots$	$\vdots$	$\vdots$
	0		c_1	c_2	$\cdots$	$c_{m_k-n_k}$	0		d_1	$\vdots$	$-c_{m_k-n_k+1}$
			c_0	c_1	$\cdots$	$c_{m_k-n_k-1}$			d_0	0	$-c_{m_k-n_k}$

Comme l'algorithme pour la détermination de m_k calcule ces systèmes itérativement jusqu'à ce qu'ils deviennent non singuliers, il nous a semblé plus efficace de leur donner dans le programme, une structure plus intéressante pour leur résolution soit par la méthode de Gauss (celle que l'on trouve dans les logiciels) soit par la méthode de *bordage* [17] ou celle de *bordage par blocs* [10].

On les a donc mémorisés de la façon suivante

Système pour le cas $m_k \leq n_k$

Système de $2m_k - 1$ équations pour β et β'
Système de $2m_k - 1$ équations pour α et α'
$(\alpha'_{m_k-1} = -c_0/d_0)$

β_0	β'_0	β_1	$\cdots$	β'_{m_k-2}	β_{m_k-1}	pour β,β'	pour α,α'
α_0	α'_0	α_1	$\cdots$	α'_{m_k-2}	α_{m_k-1}	β,β'	α,α'
$\downarrow$	$\downarrow$	$\downarrow$		$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
c_{m_k-1}	d_{m_k}	c_{m_k}	$\cdots$	d_{2m_k-2}	c_{2m_k-2}	d_{m_k-1}	$-c_{2m_k-1} + d_{2m_k-1}c_0/d_0$
c_{m_k-2}	d_{m_k-1}	c_{m_k-1}	$\cdots$	d_{2m_k-3}	c_{2m_k-3}	d_{m_k-2}	$-c_{2m_k-2} + d_{2m_k-2}c_0/d_0$
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$
c_0	d_1	c_1	$\cdots$	d_{m_k-1}	c_{m_k-1}	d_0	$-c_{m_k} + d_{m_k}c_0/d_0$
	d_0	c_0	$\cdots$	d_{m_k-2}	c_{m_k-2}	0	$-c_{m_k-1} + d_{m_k-1}c_0/d_0$
				$\vdots$	$\vdots$	$\vdots$	$\vdots$
				$\vdots$	$\vdots$	$\vdots$	$\vdots$
	0			d_1	c_1	$\vdots$	$-c_2 + d_2c_0/d_0$
				d_0	c_0	0	$-c_1 + d_1c_0/d_0$

Système pour le cas $m_k > n_k$

Système de $n_k + m_k$ équations pour β et β'
Système de $n_k + m_k$ équations pour α et α'

$$
\begin{array}{cccccccccc}
\beta_0 & \beta'_0 & \beta_1 & \cdots & \beta'_{n_k-1} & \beta_{n_k} & \beta_{n_k+1} & \cdots & \beta_{m_k-1} & \text{pour } \beta,\beta' \quad \text{pour } \alpha,\alpha' \\
\alpha_0 & \alpha'_0 & \alpha_1 & \cdots & \alpha'_{n_k-1} & \alpha_{n_k} & \alpha_{n_k+1} & \cdots & \alpha_{m_k-1} & \\
\downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow \quad \downarrow
\end{array}
$$

$$
\begin{array}{|ccccccccc|cc}
c_{m_k-1} & d_{m_k} & c_{m_k} & \cdots & d_{n_k+m_k-1} & c_{n_k+m_k-1} & c_{n_k+m_k} & \cdots & c_{2m_k-2} & d_{m_k-1} & -c_{2m_k-1} \\
c_{m_k-2} & d_{m_k-1} & c_{m_k-1} & \cdots & d_{n_k+m_k-2} & c_{n_k+m_k-2} & c_{n_k+m_k-1} & \cdots & c_{2m_k-3} & d_{m_k-2} & -c_{2m_k-2} \\
\vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\
c_0 & d_1 & c_1 & \cdots & d_{n_k} & c_{n_k-1} & c_{n_k} & \cdots & c_{m_k-1} & d_0 & -c_{m_k+1} \\
 & d_0 & c_0 & \cdots & d_{n_k-1} & c_{n_k} & c_{n_k+1} & \cdots & c_{m_k-2} & 0 & -c_{m_k} \\
 & & & & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\
 & 0 & & & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\
 & & & & d_1 & c_1 & c_2 & \cdots & c_{m_k-n_k} & \vdots & -c_{m_k-n_k+1} \\
 & & & & d_0 & c_0 & c_1 & \cdots & c_{m_k-n_k-1} & 0 & -c_{m_k-n_k}
\end{array}
$$

En ce qui concerne le calcul de x, r et z avec le BSMRZ, maintenant on aura

$$
\begin{aligned}
r_{k+1} &= r_k - A\,w_k(A)\,z_k - A\,v_k(A)r_k \\
x_{k+1} &= x_k - w_k(A)\,z_k - v_k(A)r_k \\
z_{k+1} &= q_k(A)\,z_k + t_k(A)r_k
\end{aligned}
$$

avec $z_0 = r_0$ et donc

$m_k \leq n_k$

$$
\begin{aligned}
x_{k+1} &= x_k - \big[\beta_0 z_k + \beta'_0 r_k + \beta_1 A z_k + \beta'_1 A r_k + \cdots + \beta_{m_k-2} A^{m_k-2} z_k + \beta'_{m_k-2} A^{m_k-2} r_k \\
&\qquad + \beta_{m_k-1} A^{m_k-1} z_k\big] \\
r_{k+1} &= r_k - [\beta_0 A z_k + \beta'_0 A r_k + \beta_1 A^2 z_k + \beta'_1 A^2 r_k + \cdots + \beta_{m_k-2} A^{m_k-1} z_k \\
&\qquad + \beta'_{m_k-2} A^{m_k-1} r_k + \beta_{m_k-1} A^{m_k} z_k\big] \\
z_{k+1} &= \alpha_0 z_k + \alpha'_0 r_k + \alpha_1 A z_k + \alpha'_1 A r_k + \cdots + \alpha_{m_k-2} A^{m_k-2} z_k + \alpha'_{m_k-2} A^{m_k-2} r_k \\
&\qquad + \alpha_{m_k-1} A^{m_k-1} z_k + \alpha'_{m_k-1} A^{m_k-1} r_k + A^{m_k} z_k\,.
\end{aligned}
$$

$$\boxed{m_k > n_k}$$

$$\begin{aligned}
x_{k+1} &= x_k - \big[\beta_0 z_k + \beta_0' r_k + \beta_1 A z_k + \beta_1' A r_k + \cdots + \beta_{n_k-1} A^{n_k-1} z_k + \beta_{n_k-1}' A^{n_k-1} r_k \\
&\qquad + \beta_{n_k} A^{n_k} z_k + \cdots + \beta_{m_k-1} A^{m_k-1} z_k\big] \\
r_{k+1} &= r_k - \big[\beta_0 A z_k + \beta_0' A r_k + \beta_1 A^2 z_k + \beta_1' A^2 r_k + \cdots + \beta_{n_k-1} A^{n_k} z_k + \beta_{n_k-1}' A^{n_k} r_k \\
&\qquad + \beta_{n_k} A^{n_k+1} z_k + \cdots + \beta_{m_k-1} A^{m_k} z_k\big] \\
z_{k+1} &= \alpha_0 z_k + \alpha_0' r_k + \alpha_1 A z_k + \alpha_1' A r_k + \cdots + \alpha_{n_k-1} A^{n_k-1} z_k + \alpha_{n_k-1}' A^{n_k-1} r_k \\
&\qquad + \alpha_{n_k} A^{n_k} z_k + \cdots + \alpha_{m_k-1} A^{m_k-1} z_k + A^{m_k} z_k .
\end{aligned}$$

Si l'on pose

$$s_i = A^i z_k \quad \text{et} \quad u_i = A^i r_k$$

on obtient

$$\boxed{m_k \leq n_k}$$

$$\begin{aligned}
x_{k+1} &= x_k - \big[\beta_0 s_0 + \beta_0' u_0 + \beta_1 s_1 + \beta_1' u_1 + \cdots + \beta_{m_k-2} s_{m_k-2} + \beta_{m_k-2}' u_{m_k-2} \\
&\qquad + \beta_{m_k-1} s_{m_k-1}\big] \\
r_{k+1} &= r_k - \big[\beta_0 s_1 + \beta_0' u_1 + \beta_1 s_2 + \beta_1' u_2 + \cdots + \beta_{m_k-2} s_{m_k-1} + \beta_{m_k-2}' u_{m_k-1} \\
&\qquad + \beta_{m_k-1} s_{m_k}\big] \\
z_{k+1} &= \alpha_0 s_0 + \alpha_0' u_0 + \alpha_1 s_1 + \alpha_1' u_1 + \cdots + \alpha_{m_k-2} s_{m_k-2} + \alpha_{m_k-2}' u_{m_k-2} \\
&\qquad + \alpha_{m_k-1} s_{m_k-1} + \alpha_{m_k-1}' u_{m_k-1} + s_{m_k} .
\end{aligned}$$

$$\boxed{m_k > n_k}$$

$$\begin{aligned}
x_{k+1} &= x_k - \big[\beta_0 s_0 + \beta_0' u_0 + \beta_1 s_1 + \beta_1' u_1 + \cdots + \beta_{n_k-1} s_{n_k-1} + \beta_{n_k-1}' u_{n_k-1} \\
&\qquad + \beta_{n_k} s_{n_k} + \cdots + \beta_{m_k-1} s_{m_k-1}\big] \\
r_{k+1} &= r_k - \big[\beta_0 s_1 + \beta_0' u_1 + \beta_1 s_2 + \beta_1' u_2 + \cdots + \beta_{n_k-1} s_{n_k} + \beta_{n_k-1}' u_{n_k} + \beta_{n_k} s_{n_k+1} \\
&\qquad + \cdots + \beta_{m_k-1} s_{m_k}\big] \\
z_{k+1} &= \alpha_0 s_0 + \alpha_0' u_0 + \alpha_1 s_1 + \alpha_1' u_1 + \cdots + \alpha_{n_k-1} s_{n_k-1} + \alpha_{n_k-1}' u_{n_k-1} \\
&\qquad + \alpha_{n_k} s_{n_k} + \cdots + \alpha_{m_k-1} s_{m_k-1} + s_{m_k} .
\end{aligned}$$

Les quantités nécessaires pour le BSMRZ sont à chaque étape k

$$\begin{array}{lcll} & & m_k \leq n_k & m_k > n_k \\ c_i & = & (y, A^{n_k+1+i} z_k) & i = 0, \ldots, 2m_k - 1 \quad i = 0, \ldots, 2m_k - 1 \\ d_i & = & (y, A^{n_k+i} r_k) & i = 0, \ldots, 2m_k - 1 \quad i = 0, \ldots, n_k + m_k - 1 \\ s_i & = & A^i z_k & i = 0, \ldots, m_k \quad i = 0, \ldots, m_k \\ u_i & = & A^i r_k & i = 0, \ldots, m_k - 1 \quad i = 0, \ldots, n_k \end{array}$$

que l'on calculera d'une façon semblable à ce que l'on a fait dans le MRZ.

4.3.5 Pseudo-code du BSMRZ

Algorithm BSMRZ $(A, b, x_0, y, \varepsilon)$

1. **Initializations:**
   $r_0 \leftarrow A x_0 - b$
   $s_0 = z_0 = r_0$
   $u_0 = r_0$
   $n_0 \leftarrow 0$
2. **For** $k = 0, 1, 2, \ldots$ **until** convergence **do:**
   If $n_k = n$ **then**
   solution not obtained after n iterations.
   stop.
   end if
   $d_0 \leftarrow (y, r_k)$
   If $d_0 = 0$ **then**
   impossible to use the BSMRZ.
   stop.
   end if
   $s_1 \leftarrow A s_0$
   $c_0 \leftarrow (y, s_1)$
   If $|c_0| \leq \varepsilon$ **and** $n_k = n - 1$ **then**
   incurable near-breakdown.
   stop.
   end if
   $m_k \leftarrow 1$
3. **While** $|c_{m_k-1}| \leq \varepsilon$ **and** $m_k < n - n_k$ **do:**
   $m_k \leftarrow m_k + 1$
   $y \leftarrow A^{\mathrm{T}} y$
   $c_{m_k-1} \leftarrow (y, s_1)$
   $d_{m_k-1} \leftarrow (y, r_k)$

$s_{m_k} \leftarrow A\, s_{m_k-1}$
end while
If $m_k = n - n_k$ **and** $|c_{m_k-1}| \leq \varepsilon$ **then**
incurable near-breakdown.
stop.
end if
$y \leftarrow A^{\mathrm{T}} y$
$\hat{y} \leftarrow y$
$c_{m_k} \leftarrow (y, s_1)$
If $n_k \neq 0$ **then** $d_{m_k} \leftarrow (y, r_k)$
$\hat{y} \leftarrow y$
4. **If** $m_k \neq 1$ **then**
For $i = 1, \ldots, m_k - 1$ **do:**
$\hat{y} \leftarrow A^{\mathrm{T}} \hat{y}$
$c_{m_k+i} \leftarrow (\hat{y}, s_1)$
If $i < n_k$ **then** $d_{m_k+i} \leftarrow (\hat{y}, r_k)$
If $i \leq n_k$ **then** $u_i \leftarrow A\, u_{i-1}$
end for
end if
5. **Repeat**
If $m_k \leq n_k$ **then**
compute β_i, $(i = 0, \ldots, m_k - 1)$
compute β'_j, $(j = 0, \ldots, m_k - 2)$
compute α_i, $(i = 0, \ldots, m_k - 1)$
compute α'_j, $(j = 0, \ldots, m_k - 1)$
else
compute β_i, $(i = 0, \ldots, m_k - 1)$
compute β'_j, $(j = 0, \ldots, n_k - 1)$
compute α_i, $(i = 0, \ldots, m_k - 1)$
compute α'_j, $(j = 0, \ldots, n_k - 1)$
end if
6. **If** singular system **then**
$m_k \leftarrow m_k + 1$
If $m_k + n_k = n + 1$ **then**
incurable near-breakdown.
stop.
end if
$y \leftarrow A^{\mathrm{T}} y$
For $i = 2$ **downto** 1 **do:**
$\hat{y} \leftarrow A^{\mathrm{T}} \hat{y}$
$c_{2m_k-i} \leftarrow (\hat{y}, s_1)$
$d_{2m_k-i} \leftarrow (\hat{y}, r_k)$
end for

$s_{m_k} \leftarrow A\, s_{m_k-1}$
If $m_k - 1 \leq n_k$ then $u_{m_k-1} \leftarrow A\, u_{m_k-2}$
end if
until not singular system.
7. compute $x_{k+1} = x_k - w_k(A)\, z_k - v_k(A) r_k$
compute $r_{k+1} = r_k - A\, w_k(A)\, z_k - A\, v_k(A) r_k$
If $r_{k+1} = 0$ then
solution obtained.
stop.
end if
8. $n_{k+1} \leftarrow n_k + m_k$
$s_0 \leftarrow z_{k+1} = q_k(A)\, z_k + t_k(A) r_k$
$u_0 \leftarrow r_{k+1}$
end for

4.3.6 Exemples numériques

Nous allons reprendre toujours le même système utilisé dans les essais effectués avec les algorithmes qui évitent le breakdown.

Maintenant on considère deux valeurs ε et ε_1. La première valeur sert pour tester le near-breakdown, tandis que la deuxième est utilisée pour tester les pivots dans la méthode de Gauss.

En utilisant le BSMRZ, on obtient les résultats suivants avec $x_0 = 0$ et $y = (1, \ldots, 1)^T$ et différents choix des ε

	$\varepsilon = 10^{-8}$ $\varepsilon_1 = 10^{-14}$		$\varepsilon = 10^{-1}$ $\varepsilon_1 = 10^{-11}$		$\varepsilon = 1$ $\varepsilon_1 = 10^{-11}$	
n	$\|r_k\|$	n_k	$\|r_k\|$	n_k	$\|r_k\|$	n_k
4	$1.83 \cdot 10^{-15}$	4	$1.83 \cdot 10^{-15}$	4	$1.83 \cdot 10^{-15}$	4
5	$1.65 \cdot 10^{-13}$	5	$1.65 \cdot 10^{-13}$	5	$1.65 \cdot 10^{-13}$	5
6	$1.47 \cdot 10^{-13}$	6	$1.47 \cdot 10^{-13}$	6	$1.14 \cdot 10^{-13}$	6
7	$8.34 \cdot 10^{-14}$	11	$2.13 \cdot 10^{-13}$	7	$3.45 \cdot 10^{-13}$	7
8	$4.59 \cdot 10^{-14}$	13	$2.09 \cdot 10^{-12}$	8	$1.96 \cdot 10^{-12}$	8
9	$1.72 \cdot 10^{-14}$	15	$2.51 \cdot 10^{-12}$	9	$2.23 \cdot 10^{-12}$	9
10	$5.07 \cdot 10^{-14}$	17	$2.02 \cdot 10^{-12}$	10	$3.29 \cdot 10^{-12}$	10
11	$3.85 \cdot 10^{-14}$	21	$5.55 \cdot 10^{-12}$	11	$3.86 \cdot 10^{-12}$	11
12			$2.67 \cdot 10^{-12}$	12	$1.68 \cdot 10^{-12}$	12

n_k est la dimension du sous-espace de Krylov correspondant (sans erreurs d'arrondi on doit avoir $n_k \leq n$).

Pour $x_0 = 0$ et $y = r_0$ on a

	$\varepsilon = 10^{-8}$ $\varepsilon_1 = 10^{-14}$		$\varepsilon = 10^{-1}$ $\varepsilon_1 = 10^{-11}$		$\varepsilon = 1$ $\varepsilon_1 = 10^{-11}$	
n	$\|\|r_k\|\|$	n_k	$\|\|r_k\|\|$	n_k	$\|\|r_k\|\|$	n_k
5	$4.27 \cdot 10^{-13}$	5	$2.56 \cdot 10^{-13}$	5	$2.56 \cdot 10^{-13}$	5
6	$1.09 \cdot 10^{-10}$	6	$1.09 \cdot 10^{-10}$	6	$2.22 \cdot 10^{-13}$	6
7	$4.00 \cdot 10^{-12}$	7	$2.08 \cdot 10^{-12}$	7	$2.08 \cdot 10^{-12}$	7
8	$3.39 \cdot 10^{-12}$	13	$3.66 \cdot 10^{-13}$	13	$1.21 \cdot 10^{-12}$	8
9	$2.77 \cdot 10^{-12}$	15			$1.87 \cdot 10^{-12}$	9
10	$4.72 \cdot 10^{-12}$	17			$1.74 \cdot 10^{-12}$	10
11	$4.63 \cdot 10^{-12}$	19			$5.88 \cdot 10^{-12}$	11
12	$2.11 \cdot 10^{-12}$	21	$2.07 \cdot 10^{-10}$	12	$3.73 \cdot 10^{-12}$	12

Si l'on compare ces résultats avec ceux obtenus dans le paragraphe 4.2.7, on voit que dans certains cas le MRZ (qui est valable seulement quand il y a un breakdown) donne de meilleurs résultats que le BSMRZ (qui est valable pour le breakdown et le near-breakdown). La raison semble être due au fait que, à cause des erreurs d'arrondi, le breakdown est trouvé quand il y a une quantité proche de zéro mais non pas exactement égale à zéro. Quand on utilise le MRZ, à la place du BSMRZ, quand il y a un véritable breakdown qui est seulement detecté par une certaine quantité qui est voisine de zéro, on corrige en fait l'arithmétique de l'ordinateur en imposant que cette quantité (et peut être également d'autres) soit exactement nulle et ainsi on améliore les résultats.

Dans la programmation du BSMRZ en effet il faudrait placer quatre ε différents indépendants

- ε pour tester si $\left|c^{(1)}(\xi^i P_k^{(1)})\right| \leq \varepsilon$ et sauter
- ε_1 pour tester les pivots dans la méthode de Gauss et sauter
- ε_2 pour tester si $||r_k|| \leq \varepsilon_2$ et s'arrêter
- ε_3 pour tester si $|c(\xi^{n_k} P_k)| \leq \varepsilon_3$ car l'algorithme ne peut pas être utilisé si cette quantité est nulle.

Pour mettre en évidence l'amélioration apportée par le BSMRZ nous allons donner dans le tableau suivant le nombre k d'itérations qui sont nécessaires et la dimension n_k du sous-espace de Krylov correspondant qui permet d'obtenir une norme du vecteur residu plus petite que $5 \cdot 10^{-12}$ quand on utilise le subroutine avec $\varepsilon = 10^{-8}$ et $\varepsilon_1 = 10^{-14}$ et aussi avec $\varepsilon = \varepsilon_1 = 10^{-30}$ (dans ce dernier cas on ne passe jamais par les règles de l'algorithme).

	$\varepsilon = 10^{-8}$ $\varepsilon_1 = 10^{-14}$ $y = (1,\ldots,1)^T$		$\varepsilon = 10^{-30}$ $\varepsilon_1 = 10^{-30}$ $y = (1,\ldots,1)^T$		$\varepsilon = 10^{-8}$ $\varepsilon_1 = 10^{-14}$ $y = r_0$		$\varepsilon = 10^{-30}$ $\varepsilon_1 = 10^{-30}$ $y = r_0$	
n	k	n_k	k	n_k	k	n_k	k	n_k
7	11	11	11	11	7	7	7	7
8	12	13	13	13	13	13	13	13
9	13	15	15	15	14	15	15	15
10	14	17	17	17	15	17	17	17
11	15	21	19	19	16	19	19	19
12			21	21	17	21	21	21

Maintenant, quand le système est de dimension $n = 50$ et pour $y = r_0$, $\varepsilon_1 = 10^{-15}$, les normes du vecteur residu quand $n_k = 50$ sont

ε	k	$n.j$	$\|r_k\|$	m	n_m
1	12	2	$1.22 \cdot 10^4$	57	95
10^{-1}	7	2	$5.46 \cdot 10^2$	54	97
10^{-2}	15	2	$4.78 \cdot 10^5$	57	92
10^{-3}	11	2	$3.17 \cdot 10^4$	54	93
10^{-4}	9	2	$1.33 \cdot 10^4$	55	96
10^{-5}	8	1	$4.92 \cdot 10^2$	55	97
10^{-6}	8	1	$4.92 \cdot 10^2$	55	97
10^{-7}	8	1	$4.92 \cdot 10^2$	55	97
10^{-8}	8	1	$4.92 \cdot 10^2$	55	97
10^{-9}	8	1	$4.92 \cdot 10^2$	55	97
10^{-10}	8	1	$4.92 \cdot 10^2$	57	99
10^{-11}	30	1	$2.81 \cdot 10^5$	92	112
10^{-12}	30	2	$5.68 \cdot 10^5$		
10^{-13}	47	2	$2.39 \cdot 10^6$		
10^{-14}	49	1	$6.16 \cdot 10^7$		
10^{-15}	50	0	$1.87 \cdot 10^8$		

$n.j$ représente le nombre de sauts, m l'itération telle que $\|r_m\| \leq \varepsilon$ et n_m la dimension du sous-espace de Krylov correspondant (avec une borne supérieure de 150).

4.4 Near–breakdown dans CGS

Une variante de la méthode de Lanczos, qui peut être très puissante et intéressante, a été proposée par Sonneveld [31] et elle a été nommée *conjugate gradient squared method* (CGS). Dans cette variante les residus sont définis par la formule suivante

$$r_k = P_k^2(A)\, r_0$$

et elle peut être considérée comme une alternative au gradient biconjugué car elle n'a pas besoin dans l'algorithme d'utiliser la matrice A^*.

Le breakdown dans cet algorithme a été résolu par Brezinski et Sadok [13] en utilisant les relations (4.5), (4.6), (4.7) et (4.8) élévées au carré et ils ont obtenu trois nouvelles méthodes sans breakdown appelées MRZS, SMRZS et BMRZS (où S signifie *squared*).

Pour éviter le near-breakdown dans [9] nous avons fait la même chose en prenant les relations (4.9) et (4.11) du BSMRZ au carré.

On a donc

$$P_{k+1}^2 = (1-\xi v_k)^2 P_k^2 - 2(1-\xi v_k)\xi w_k P_k P_k^{(1)} + \xi^2 w_k^2 P_k^{(1)^2}$$

et il faut calculer récursivement les polynômes $P_{k+1}^{(1)^2}$ et $P_{k+1}P_{k+1}^{(1)}$.

Mais nous avons aussi

$$P_{k+1}^{(1)^2} = q_k^2 P_k^{(1)^2} + 2q_k t_k P_k P_k^{(1)} + t_k^2 P_k^2$$

et donc, en mulipliant (4.9) par (4.11) on aura

$$P_{k+1}P_{k+1}^{(1)} = (q_k - \xi q_k v_k - \xi t_k w_k) P_k P_k^{(1)} - \xi q_k w_k P_k^{(1)^2} + t_k(1-\xi v_k) P_k^2.$$

Si nous posons

$$\begin{aligned} r_k &= P_k^2(A)\, r_0 \\ z_k &= P_k^{(1)^2}(A)\, r_0 \\ s_k &= P_k(A) P_k^{(1)}(A)\, r_0 \end{aligned}$$

on aura l'algorithme suivant appelé BSMRZS

$$\begin{aligned} r_{k+1} &= (I - Av_k(A))^2 r_k - 2\,(I - Av_k(A))\, A\, w_k(A)\, s_k + A^2 w_k^2(A)\, z_k \\ x_{k+1} &= x_k - (Av_k(A) - 2I)\, v_k(A)\, r_k + 2\,(I - Av_k(A))\, w_k(A)\, s_k - A\, w_k^2(A)\, z_k \\ z_{k+1} &= q_k^2(A)\, z_k + 2q_k(A)\, t_k(A)\, s_k + t_k^2(A)\, r_k \\ s_{k+1} &= (q_k(A) - A\, q_k(A)\, v_k(A) - A\, t_k(A)\, w_k(A))\, s_k - A\, q_k(A)\, w_k(A)\, z_k \\ &\qquad + t_k(A)\,(I - Av_k(A))\, r_k. \end{aligned} \tag{4.12}$$

4.4.1 Mise en œuvre de l'algorithme BSMRZS

Pour réaliser cet algorithme il faut calculer plusieurs produits entre polynômes.

Si nous posons

$$\begin{aligned} P(\xi) &= a_0 + \cdots + a_n \xi^n \\ Q(\xi) &= b_0 + \cdots + b_k \xi^k \end{aligned}$$

avec $n \leq k$, leur produit peut être calculé par

$$P(\xi)Q(\xi) = d_0 + \cdots + d_{n+k}\xi^{n+k}$$

avec

$$d_i = \sum_{j=max(0,i-n)}^{min(k,i)} b_j a_{i-j}.$$

Dans le MRZS on n'avait pas besoin de la matrice A^*, mais cela n'est plus vrai dans l'algorithme BSMRZS qui évite le near-breakdown car l'on doit calculer pour $i = 0, \ldots, 2m_k - 1$, les quantités

$$c(\xi^{n_k+i} P_k) = \left(y, A^{n_k+i} P_k(A) r_0\right) = \left(y, A^{n_k+i} r'_k\right) = \left(A^{*n_k} y, A^i r'_k\right)$$

$$c^{(1)}\left(\xi^{n_k+i} P_k^{(1)}\right) = \left(y, A^{n_k+i+1} P_k^{(1)}(A) r_0\right) = \left(y, A^{n_k+i+1} z'_k\right) = \left(A^{*n_k} y, A^{i+1} z'_k\right)$$

où $r'_k = P_k(A) r_0$ et $z'_k = P_k^{(1)}(A) r_0$.

Ces deux vecteurs peuvent se calculer avec les relations (4.9) et (4.11) et l'on aura

$$\begin{aligned} r'_{k+1} &= (I - A v_k(A))\, r'_k - A\, w_k(A)\, z'_k \\ z'_{k+1} &= q_k(A)\, z'_k + t_k(A)\, r'_k \end{aligned} \tag{4.13}$$

avec $r'_0 = z'_0 = r_0$.

Pour calculer les coefficients des polynômes w_k, v_k, q_k et t_k on aura des systèmes à résoudre qui ont la même forme de ceux du BSMRZ, mais les coefficients des équations seront calculés (à l'iteration $k+1$) par

	$m_k \le n_k$	$m_k > n_k$
$d_i = (y, A^{n_k+i} r'_k)$	$i = 0, \ldots, 2m_k - 1$	$i = 0, \ldots, n_k + m_k - 1$
$c_i = (y, A^{n_k+i+1} z'_k)$	$i = 0, \ldots, 2m_k - 1$	$i = 0, \ldots, 2m_k - 1.$

Tous les vecteurs r_{k+1}, x_{k+1}, z_{k+1}, s_{k+1}, r'_{k+1} et z'_{k+1}, seront obtenus comme combinaison linéaire des coefficients des polynômes qui paraissent dans (4.12) and (4.13) avec des vecteurs de la forme

	$m_k \le n_k$	$m_k > n_k$
$p_i = A^i z_k$	$i = 0, \ldots, 2m_k$	$i = 0, \ldots, 2m_k$
$g_i = A^i r_k$	$i = 0, \ldots, 2m_k - 2$	$i = 0, \ldots, 2n_k$
$u_i = A^i s_k$	$i = 0, \ldots, 2m_k - 1$	$i = 0, \ldots, n_k + m_k$
$p'_i = A^i z'_k$	$i = 0, \ldots, m_k$	$i = 0, \ldots, m_k$
$g'_i = A^i r'_k$	$i = 0, \ldots, m_k - 1$	$i = 0, \ldots, n_k.$

Tous ces vecteurs seront calculés seulement une fois et mémorisés dans cinq matrices différentes.

Dans l'algorithme on utilisera trois valeurs pour les tests des quantités petites:

- ε pour tester si $\left|c^{(1)}(\xi^i P_k^{(1)})\right| \leq \varepsilon$ et sauter
- ε_1 pour tester les pivots dans la méthode de Gauss et sauter
- ε_2 pour tester si $||r_k|| \leq \varepsilon_2$ et s'arrêter.

On utilisera aussi d'autres quantités que nous allons définir. Le CGS trouve la solution quand la valeur de n_k est plus petite ou égal a n, où k est la valeur pour laquelle $\|r_k\| \leq \varepsilon_2$. Comme les ordinateurs ont une précision finie, il peut arriver que la solution soit trouvée avec un n_k plus grand que n et donc on a introduit une valeur $n_{\max} \geq n$ pour laisser l'algorithme continuer après n (mais sans créer une boucle infinie).

Une autre quantité que l'on veut fixer à l'avance est la longueur du saut m_k car si elle est trop grande, le nombre de mémoires nécessaires au programme pour les vecteurs et les matrices de mémorisation peuvent devenir impossibles à réaliser. Dans la théorie le saut peut être au maximum $n_{\max}$ car, à la première iteration on a $n_0 = 0$, mais dans la pratique, quand le système à résoudre est très grand, on doit fixer $m_{k\max}$.

4.4.2 Pseudo-code du BSMRZS

Algorithm BSMRZS $(A, b, x_0, y, n, n_{\max}, m_{k\max}, \varepsilon, \varepsilon_1, \varepsilon_2)$

1. **Initializations:**
   $r_0 \leftarrow A\,x_0 - b$
   $p_0 = z_0 = r_0$
   $g_0 = r_0$
   $u_0 = s_0 = r_0$
   $p'_0 = z'_0 = r_0$
   $g'_0 = r'_0 = r_0$
   $n_0 \leftarrow 0$
   If $\|r_0\| \leq \varepsilon_2$ **then**
   solution obtained.
   stop.
   end if
2. **For** $k = 0, 1, 2, \ldots$ **until** convergence **do:**
   $d_0 \leftarrow (y, r'_k)$
   $p_1 \leftarrow A\,p_0$
   $u_1 \leftarrow A\,u_0$
   $p'_1 \leftarrow A\,p'_0$
   $c_0 \leftarrow (y, p'_1)$
   If $|c_0| \leq \varepsilon$ **and** $n_k = n_{\max} - 1$ **then**
   solution not obtained at $n_{\max}$.
   stop.
   end if
   $m_k \leftarrow 1$

3. **While** $|c_{m_k-1}| \leq \varepsilon$ **and** $m_k < n_{\max} - n_k$ **and** $m_k \leq m_{k\max}$ **do:**
   $m_k \leftarrow m_k + 1$
   $p_{m_k} \leftarrow A\, p_{m_k-1}$
   $u_{m_k} \leftarrow A\, u_{m_k-1}$
   $p'_{m_k} \leftarrow A\, p'_{m_k-1}$
   $y \leftarrow A^{\mathrm{T}} y$
   $c_{m_k-1} \leftarrow (y, p'_1)$
   $d_{m_k-1} \leftarrow (y, g'_0)$
   end while
   If $(m_k = n_{\max} - n_k$ **and** $|c_{m_k-1}| < \varepsilon)$ **or** $(m_k > m_{k\max})$ **then**
   solution not obtained at $n_{\max}$ or
   solution not obtained because the jump is greater than $m_{k\max}$.
   stop.
   end if
   If $m_k \leq n_k$ **and** $|d_0| = 0$ **then**
   impossible to use the BSMRZS.
   stop.
   end if
4. $y \leftarrow A^{\mathrm{T}} y$
   $\hat{y} \leftarrow y$
   $c_{m_k} \leftarrow (y, p'_1)$
   $d_{m_k} \leftarrow (y, g'_0)$
   If $m_k \neq 1$ **then**
   For $i = 1, \ldots, m_k - 1$ **do:**
   $p_{m_k+i} \leftarrow A\, p_{m_k+i-1}$
   If $i \leq n_k$ **then**
   $g_i \leftarrow A\, g_{i-1}$
   $g'_i \leftarrow A\, g'_{i-1}$
   $u_{m_k+i} \leftarrow A\, u_{m_k+i-1}$
   end if
   $\hat{y} \leftarrow A^{\mathrm{T}} \hat{y}$
   $c_{m_k+i} \leftarrow (\hat{y}, p'_1)$
   If $i < n_k$ **then** $d_{m_k+i} \leftarrow (\hat{y}, g'_0)$
   end for
   end if
   If $m_k \leq n_k$ **then**
   For $i = 1, \ldots, m_k - 1$ **do** $g_{m_k+i-1} \leftarrow A\, g_{m_k+i-2}$
   else
   For $i = 1, \ldots, n_k$ **do** $g_{n_k+i} \leftarrow A\, g_{n_k+i-1}$
   end if
   $p_{2m_k} \leftarrow A\, p_{2m_k-1}$
5. **Repeat**
   compute β_i, $i = 0, \ldots, m_k - 1$ (coefficients of $w_k(\xi)$)

compute α_i, $i = 0, \ldots, m_k$ (coefficients of $q_k(\xi)$)
If $m_k \leq n_k$ **then**
  compute α'_i, $i = 0, \ldots, m_k - 1$ (coefficients of $t_k(\xi)$)
  If $m_k \neq 1$ **then** compute β'_i, $i = 0, \ldots, m_k - 2$ (coefficients of $v_k(\xi)$)
else if $n_k \neq 0$ **then**
  compute α'_i, $i = 0, \ldots, n_k - 1$ (coefficients of $t_k(\xi)$)
  compute β'_i, $i = 0, \ldots, n_k - 1$ (coefficients of $v_k(\xi)$)
end if

6. **If** system singular **then**
  $m_k \leftarrow m_k + 1$
  If $(m_k = n_{\max} - n_k + 1)$ **or** $(m_k > m_{k\max})$ **then**
    solution not obtained at $n_{\max}$ or
    solution not obtained because the jump is greater than $m_{k\max}$.
    stop.
  end if
  $y \leftarrow A^{\mathrm{T}} y$
  $p_{2m_k-1} \leftarrow A\, p_{2m_k-2}$
  $p_{2m_k} \leftarrow A\, p_{2m_k-1}$
  $p'_{m_k} \leftarrow A\, p'_{m_k-1}$
  If $m_k \leq n_k + 1$ **then**
    $g_{2m_k-3} \leftarrow A\, g_{2m_k-4}$
    $g_{2m_k-2} \leftarrow A\, g_{2m_k-3}$
    $g'_{m_k-1} \leftarrow A\, g'_{m_k-2}$
    $u_{2m_k-2} \leftarrow A\, u_{2m_k-3}$
    $u_{2m_k-1} \leftarrow A\, u_{2m_k-2}$
  else
    $u_{n_k+m_k} \leftarrow A\, u_{n_k+m_k-1}$
  end if
  $\hat{y} \leftarrow A^{\mathrm{T}} \hat{y}$
  $c_{2m_k-2} \leftarrow (\hat{y}, p'_1)$
  If $m_k > n_k$ **then**
    $d_{n_k+m_k-1} \leftarrow \left(y, g'_{n_k-1}\right)$
  else
    $d_{2m_k-2} \leftarrow (\hat{y}, g'_0)$
  end if
  $\hat{y} \leftarrow A^{\mathrm{T}} \hat{y}$
  $c_{2m_k-1} \leftarrow (\hat{y}, p'_1)$
  If $m_k \leq n_k$ **then** $d_{2m_k-1} \leftarrow (\hat{y}, g'_0)$

until system non singular (pivot $> \varepsilon_1$).

7. compute $x_{k+1} = x_k - (A\, v_k(A) - 2I) v_k(A) r_k + 2(I - A\, v_k(A)) w_k(A)\, s_k$
  $-A\, w_k^2(A)\, z_k$
  compute $r_{k+1} = (I - A\, v_k(A))^2 r_k - 2(I - A\, v_k(A))\, A\, w_k(A)\, s_k$
  $+A^2\, w_k^2(A) z_k$

If $\|r_{k+1}\| \leq \varepsilon_2$ **then**
solution obtained.
stop.
end if

8. $n_{k+1} \leftarrow n_k + m_k$
If $n_{k+1} = n_{\max}$ **then**
solution not obtained at $n_{\max}$.
stop.
end if
compute $z_{k+1} = t_k^2(A)\, r_k + 2q_k(A)\, t_k(A)\, s_k + q_k^2(A)\, z_k$
compute $s_{k+1} = (I - A\, v_k(A))\; q_k(A)\, s_k - A\, t_k(A)\, w_k(A)\, s_k -$
$A q_k(A)\, w_k(A)\, z_k + t_k(A)(I - A v_k(A))\, r_k$
compute $r'_{k+1} = r'_k - A\, w_k(A)\, z'_k - A\, v_k(A) r'_k$
compute $z'_{k+1} = q_k(A)\, z'_k + t_k(A) r'_k$
$p_0 \leftarrow z_{k+1}$
$g_0 \leftarrow r_{k+1}$
$u_0 \leftarrow s_{k+1}$
$p'_0 \leftarrow z'_{k+1}$
$g'_0 \leftarrow r'_{k+1}$
end for

4.4.3 Exemples numériques

On rappelle que pour l'algorithme qui implémente le near-breakdown dans le CGS nous devons choisir les trois valeurs ε, ε_1 et ε_2, selon la signification expliquée à la fin du paragraphe 4.4.1. Les résultats numériques ont été obtenus sur un PC avec coprocesseur mathématique.

On considère d'abord le système qui a été déjà utilisé dans les exemples numériques du MRZ, SMRZ, BMRZ et BSMRZ

$$\begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & -1 \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \\ \vdots \\ n \end{pmatrix} = \begin{pmatrix} -n \\ 1 \\ 2 \\ \vdots \\ n-1 \end{pmatrix}.$$

Pour $n = 12, y = (1, \ldots, 1)^T, \varepsilon_1 = 10^{-16}$ et $\varepsilon_2 = 10^{-7}$, il y a un saut de $n_3 = 3$ à $n_4 = 9$ si l'on prend $\varepsilon = 10^{-3}$ et un saut de $n_2 = 2$ à $n_3 = 6$ si l'on prend $\varepsilon = 10^{-1}$. Cependant, dans ces deux cas, les itérations ne convergent pas, même si l'on continue jusqu'à $n_k = 25$. Quand $\varepsilon = 1$, nous avons un saut de $n_2 = 2$ à $n_3 = 10$ et l'on obtient $n_5 = 12$ et $\|r_5\| = 1.20 \cdot 10^{-8}$.

La même chose arrive si l'on prend $n = 20$. Avec $\varepsilon = 10^{-1}$ nous avons un saut de $n_2 = 2$ à $n_3 = 7$ mais les itérations ne convergent pas. Avec $\varepsilon = 1$ nous avons un saut de

$n_2 = 2$ à $n_3 = 18$ et l'on obtient $n_5 = 20$ et $||r_5|| = 5.31 \cdot 10^{-7}$.
Maintenant, pour $n = 12$ et $y = r_0$, si $\varepsilon = 10^{-3}$ les itérations ne convergent pas. Avec $\varepsilon = 1$ nous avons un saut de $n_2 = 2$ à $n_3 = 11$ et l'on obtient $n_4 = 12$ et $||r_4|| = 3.66 \cdot 10^{-8}$.

Comme deuxième exemple, considérons la matrice diagonale par blocs de dimension $n \times n$ suivante

$$A = \begin{pmatrix} M_1 & & & \\ & M_2 & & 0 \\ & & \ddots & \\ & 0 & & M_{n/2} \end{pmatrix}$$

avec

$$M_j = \begin{pmatrix} 1 & j-1 \\ a & -1 \end{pmatrix} \qquad \text{pour } j = 1, \ldots, n/2.$$

Le cas où $a = 0$ a été déjà considéré dans [28]. Dans ce cas on a un breakdown à la première itération du CGS car le polynôme minimal de la matrice A est de degré 2.

La solution de ce système est la suivante

$$x_{2j-1} = -\frac{b_{2j-1} + (j-1)b_{2j}}{-1 - a(j-1)}$$

$$x_{2j} = \frac{b_{2j} - ab_{2j-1}}{-1 - a(j-1)}$$

où les b_i sont les composantes du vecteur second membre b.

Si nous prenons $b = (5, 3, 4, -4, 0, \ldots, 0)^T, x_0 = (27, 0, \ldots, 0)^T, y = (1, 2, \ldots, n)^T, n = 20, a = -0.4, \varepsilon_1 = 10^{-30}$ et $\varepsilon_2 = 10^{-10}$ nous avons les résultats suivants

	saut		solution obtenue		
ε	de	à	k	n_k	$\|\|r_k\|\|$
10^{-30}			4	4	$5.2 \cdot 10^{-14}$
1	$n_3 = 3$	$n_4 = 6$	4	6	$6.6 \cdot 10^{-11}$
2	$n_3 = 3$	$n_4 = 10$	4	10	$9.8 \cdot 10^{-12}$
3	$n_2 = 2$	$n_3 = 5$	3	5	$4.6 \cdot 10^{-13}$
4	$n_2 = 2$	$n_3 = 7$	3	7	$9.1 \cdot 10^{-15}$
5	$n_2 = 2$	$n_3 = 11$	3	11	$3.6 \cdot 10^{-14}$

Avec $y = r_0$, on obtient

	saut		solution obtenue		
ε	de	à	k	n_k	$\|\|r_k\|\|$
10^{-30}			4	4	$5.8 \cdot 10^{-15}$
1			4	4	$5.8 \cdot 10^{-15}$
2	$n_2 = 2$	$n_3 = 4$	3	4	$5.3 \cdot 10^{-15}$
3	$n_2 = 2$	$n_3 = 6$	3	6	$1.2 \cdot 10^{-12}$
4	$n_2 = 2$	$n_3 = 6$	3	6	$1.2 \cdot 10^{-12}$
5	$n_2 = 2$	$n_3 = 8$	3	8	$7.4 \cdot 10^{-13}$

Comme on l'a déjà vu pour le BSMRZ (voir aussi [12]), les résultats sont très sensibles au choix des différents ε.

Il est évident qu'une étude approfondie devra être faite dans la suite pour mieux comprendre la stabilité numérique de cet algorithme en fonction du début du saut et de sa longueur qui dépendent évidemment du choix des ε.

BIBLIOGRAPHIE

[1] D. L. BOLEY, S. ELHAY, G. H. GOLUB, M. H. GUTKNECHT, *Nonsymmetric Lanczos and finding orthogonal polynomials associated with indefinite weights*, Numerical Algorithms, 1 (1991).

[2] C. BREZINSKI, *Padé-type Approximation and General Orthogonal Polynomials*, ISNM vol.50, Birkhäuser-Verlag, Basel, 1980.

[3] C. BREZINSKI, *Some determinantal identities in a vector space, with applications*, in *Padé Approximations ant its Applications. Bad-Honnef 1983*, H. Werner and H. J. Bünger eds., LNM 1071, Springer-Verlag, Berlin, 1984, pp. 1–11.

[4] C. BREZINSKI, *Other manifestations of the Schur complement*, Linear Alg. Appl., 111 (1988) 231–247.

[5] C. BREZINSKI, *Biorthogonality and its Applications to Numerical Analysis*, Marcel Dekker, New-York, 1991.

[6] C. BREZINSKI, *CGM: a whole class of Lanczos-type solvers for linear systems*, soumis.

[7] C. BREZINSKI, M. REDIVO ZAGLIA, *A new presentation of orthogonal polynomials with application to their computation*, Numerical Algorithms, 1 (1991) 207–222.

[8] C. BREZINSKI, M. REDIVO ZAGLIA, *Extrapolation Methods. Theory and Practice*, North-Holland, Amsterdam, 1991.

[9] C. BREZINSKI, M. REDIVO ZAGLIA, *Treatment of near-breakdown in the CGS algorithm*, soumis.

[10] C. BREZINSKI, M. REDIVO ZAGLIA, H. SADOK, *A breakdown-free Lanczos type algorithm for solving linear systems*, Numer. Math., à paraître.

[11] C. BREZINSKI, M. REDIVO ZAGLIA, H. SADOK, *Avoiding breakdown and near-breakdown in Lanczos type algorithms*, Numerical Algorithms, 1 (1991) 261-284.

[12] C. BREZINSKI, M. REDIVO ZAGLIA, H. SADOK, *Addendum to "Avoiding breakdown and near-breakdown in Lanczos type algorithms"*, Numerical Algorithms, 2 (1992), sous presse.

[13] C. BREZINSKI, H. SADOK, *Avoiding breakdown in the CGS algorithm*, Numerical Algorithms, 1 (1991) 199-206.

[14] C. BREZINSKI, H. SADOK, *Lanczos type methods for systems of linear equations*, soumis.

[15] P.J. DAVIS, *Interpolation and Approximation*, Dover, New York, 1975.

[16] A. DRAUX, *Polynômes Orthogonaux Formels. Applications*, LNM 974, Springer-Verlag, Berlin, 1983.

[17] V. N. FADDEEVA, *Computational methods of linear algebra*, Dover, New-York, 1959.

[18] R. FLETCHER, *Conjugate gradient methods for indefinite systems*, in *Numerical Analysis*, G. A. Watson ed., LNM 506, Springer-Verlag, Berlin, 1976, pp. 73–89.

[19] M. H. GUTKNECHT, *A complete theory of the unsymmetric Lanczos process and related algorithms. Parts I, II* , SIAM J. Matrix Anal. Appl., à paraître.

[20] G. H. GOLUB, D. P. O'LEARY, *Some history of the conjugate gradient and Lanczos algorithms*, SIAM Rev., 31 (1989) 50–102.

[21] M. H. GUTKNECHT, *The unsymmetric Lanczos algorithms and their relations to Padé approximation, continued fractions, and the qd algorithm*, à paraître.

[22] M. R. HESTENES, E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, J. Res. NBS, 49 (1952) 409–436.

[23] E. HENDRIKSEN, H. VAN ROSSUM, *Moment methods in Padé approximation*, J. Approx. Theory, 35 (1982) 250–263.

[24] W. D. JOUBERT, T. A. MANTEUFEL, *Iterative methods for nonsymmetric linear systems*, in *Iterative methods for large linear systems*, D. R. Kincaid and L. J. Hayes eds. , Academic Press, New-York, 1990, pp. 149–171.

[25] H. B. KELLER, *The bordering algorithm and path following near singular points of higher nullity*, SIAM J. Sci. Stat. Comput., 4 (1983) 573–582.

[26] C. LANCZOS, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. Natl. Bur. Stand., 45 (1950) 225–282.

[27] C. LANCZOS, *Solution of systems of linear equations by minimized iterations*, J. Res. Natl. Bur. Stand., 49 (1952) 33–53.

[28] N.M. NACHTIGAL, S.C. REDDY, L.N. TREFETHEN, *How fast are nonsymmetric matrix iterations?*, SIAM J. Sci. Stat. Comp., à paraître.

[29] B. N. PARLETT, D. R. TAYLOR, Z. A. LIU, *A look-ahead Lanczos algorithm for unsymmetric matrices*, Math. Comput., 44 (1985) 105–124.

[30] Y. SAAD, *The Lanczos biorthogonalization algorithm and other oblique projection methods for solving large unsymmetric systems*, SIAM J. Numer. Anal., 19 (1982) 485–506.

[31] P. SONNEVELD, *CGS, a fast Lanczos-type solver for nonsymmetric linear systems*, SIAM J. Sci. Stat. Comp., 10 (1989) 36–52.

[32] G. W. STRUBLE, *Orthogonal polynomials: variable-signed weight functions*, Numer. Math. , 5 (1963) 88–94.

[33] H. A. VAN DER VORST, *Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 13 (1992) 631–644.

[34] YU. V. VOROBYEV, *Method of moments in applied mathematics*, Gordon and Breach, New-York, 1965.

[35] D. M. YOUNG, K. C. JEA, *Generalized conjugate-gradient acceleration of nonsymmetrizable iterative methods*, Linear Alg. Appl., 34 (1984) 159–194.