Table des matières

Introduction

En analyse numérique, la vitesse de convergence d'une suite représente la vitesse à laquelle les termes de la suite se rapprochent de sa limite. Bien que cet ordre de grandeur de vitesse de convergence ne fournisse pas d'information sur toute partie finie de l'ensemble des termes de la suite, ce concept a une grande importance pratique, en général peu d'itérations sont nécessaires pour donner une valeur approchée intéressante lorsque la vitesse de convergence est grande. Afin d'accélérer la convergence des différentes suites nous utilisons des méthodes pour obtenir la solution exacte.

Ce travail a pour but d'étudié les méthodes d'accélération de la convergence qui sont utilisées pour accélérer les suites de nombres réels ou complexes, les suites de vecteurs ou de matrices, et les suites de fonctions.

Le premier chapitre est une présentation des méthodes d'accélération de la convergence comme les procédés de sommation linéaire, le procédé de Richardson, les algorithmes scalaires et non scalaires et confluents..., on donnons les règles de chaque méthodes présentées, et on montrons comment ils fonctionnent. De même ce chapitre donne quelques résultats théoriques sur ces méthodes contenant des théorèmes et des remarques pour mieux comprendre comment peut-on appliqué ces méthodes.

Le deuxième chapitre traite l'utilisation des méthodes d'accélération de la convergence des suites et des fonctions, il est consacré aux applications des méthodes d'accélération de la convergence étudiées dans le premier chapitre, nous traitons la résolution des équations différentielles, les systèmes linéaires, les systèmes non linéaires, les équations matricielles à l'aide des méthodes d'accélération, de plus nous montrons comment utiliser les algorithmes d'accélération de la convergence pour construire de nouvelles méthodes afin de résoudre les équations.

Le troisième chapitre contient des exemples numériques qui sont résolus par différents méthodes d'accélération de la convergence, nous avons comparé l'accélération de la convergence, le nombre d'itération et la vitesse de convergence entre les méthodes.

Chapitre 1

Présentation des algorithmes d'accélération de la convergence et quelques résultats théoriques

1.1 Procédure d'accélération

Une procédure d'accélération consiste à transformer une suite donnée $\{S_n\}$ en une nouvelle suite $\{V_n\}$ avec une convergence plus rapide. La suite $\{V_n\}$doit vérifier les propriétés suivantes :

1/ V_n converge.

2/ V_n converge aussi vers S (S_n converge vers S).

3/ V_n converge plus vite que S_n, c'est-à-dire :

$$\lim_{n\to\infty} \frac{V_n - S}{S_n - S} = 0$$

Si ces trois conditions sont satisfaites alors nous disons que la suite $\{V_n\}$ converge plus vite que $\{S_n\}$.

Il existe plusieurs méthodes d'accélération de la convergence, nous donnons dans la suite quelques unes:

1.2 Les procédés de sommation linéaire

Les procédés de sommation linéaire ont été conçus par **Hardy** pour donner une limite aux séries divergentes par une sorte de prolongement analytique. Ces procédés peuvent aussi être appliqués aux séries lentement convergentes dans le but de les accélérer.

La classe la plus importante des procédés de sommation linéaire est basée sur l'application d'une matrice triangulaire $A = (a_{ij})$ qui transforme la suite originale (S_n) en une suite (V_n) définie par :

$$\begin{pmatrix} V_0 \\ V_1 \\ . \\ . \end{pmatrix} = A + \begin{pmatrix} S_0 \\ S_1 \\ . \\ . \end{pmatrix} \text{ avec } \quad A = \begin{pmatrix} a_{00} & 0 & 0 & 0 & 0 \\ a_{10} & a_{11} & 0 & 0 & 0 \\ a_{20} & a_{21} & a_{22} & 0 & 0 \\ . & . & . & . & 0 \\ . & . & . & . & . \end{pmatrix} \tag{1.2.1}$$

On note par A ce procédé de sommation linéaire.

Soit S la limite de la suite (S_n), le procédé de sommation linéaire est dit régulier si la suite transformée (V_n) converge vers la même limite S.

Pour obtenir la régularité d'un procédé de sommation linéaire, **Toepliz** a donné dans le théorème suivant trois conditions suffisantes :

Théorème 1.2.1 *Théorème de* ***Toepliz***

Le procédé de sommation linéaire A est régulier si :

(i) $\sum_{k=0}^{\infty} |a_{nk}| < M, \forall n.$

(ii) $\lim_{n\to\infty} a_{nk} = 0, \forall k.$

(iii) $\lim_{n\to\infty} \sum_{k=0}^{\infty} a_{nk} = 1.$

1.2.1 Quelques exemples de procédés réguliers

Le procédé de Hölder noté (H,1).

La matrice $A = (H, 1)$est donnée par :

$$a_{ij} = \left\{ \begin{array}{ccc} \frac{1}{i+1} & \text{pour} & j = 1, ..., i \\ 0 & \text{pour} & j > i \end{array} \right\}$$

Elle s'écrit alors :

$$\begin{pmatrix} \frac{1}{1} & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ . & . & . & . & 0 \\ . & . & . & . & . \end{pmatrix}$$

et on obtient la suite transformée suivante :

$$V_0 = S_0,\ V_1 = \frac{S_0 + S_1}{2},\ V_2 = \frac{S_0 + S_1 + S_2}{3}, ...$$

Si avec la suite (V_n), la convergence n'est pas accélérée on généralise ce procédé en considérant la matrice A^K (K entier), on le note (H, K).

La matrice $(H, 2)$ est définie par :

$$(H, 2) = (H, 1) * (H, 1)$$

On obtient une nouvelle suite :

$$W_0 = V_0\ , W_1 = \frac{V_0 + V_1}{2},\ W_2 = \frac{V_0 + V_1 + V_2}{3}, ...$$

On continue de cette manière jusqu'à obtention d'accélération de la convergence de la suite (S_n).

Le procédé de Cesaro d'ordre K noté (C,K)

La matrice d'ordre K est donnée par :

$$(C, K) = (H, 1) * L^{K-1}$$

Où L est une matrice triangulaire inférieure gauche, avec tous les éléments égaux à 1.

Le procédé d'Euler

La matrice A de ce procédé est donnée par :

$$a_{ij} = \left\{ \begin{array}{cc} \frac{q^{i-j}}{(q+1)^i} C_j^i & \text{pour} \quad j = 0, 1, ..., i \\ 0 & \text{pour} \quad j > i \end{array} \right\} \quad \text{avec} \quad \sum_{j=0}^{\infty} a_{ij} = 1$$

Où le paramètre q est choisi arbitrairement $(q = 1)$.

Remarque 1.2.1 *1- Les trois procédés ci-dessus sont réguliers.*

2- Ces procédés vérifient

$$\sum_{k=0}^{n} a_{nk} = 1, \ \forall n$$

Cette égalité garantit la troisième condition du théorème de ***Toepliz.***

1.3 Le procédé d'extrapolation de Richardson

Ce procédé illustre l'accélération de la convergence d'une méthode numérique proposé en 1927 par **Lewis Fry-Richardson** $(1881-1953)$. L'extrapolation à la limite consiste à calculer plusieurs fois la même quantité avec un maillage différent. On pose :

$$T_0^{(n)} = S_n, \ \text{pour} \quad n = 0, 1, ...$$

Puis on calcule des quantités à deux indices $T_k^{(n)}$ à l'aide de la relation suivante :

$$T_{k+1}^{(n)} = \frac{x_n T_k^{(n+1)} - x_{n+k+1} T_k^{(n)}}{x_n - x_{n+k+1}}, \quad n, k = 0, 1, ... \tag{1.3.2}$$

Ce procédé permet de transformer la suite $\{S_n\}$ en un ensemble de suite $\left\{T_k^{(n)}\right\}$ pour k fixé et n variant (où n fixé et k invariant).

On place ces quantités dans un tableau à double entrée : l'indice inférieur représente une colonne et l'indice supérieur représente une diagonale descendante :

$$\begin{array}{llllll}
T_0^{(0)} = S_0 & & & & & \\
 & T_1^{(0)} & & & & \\
T_0^{(1)} = S_1 & & T_2^{(0)} & & & \\
 & T_1^{(1)} & & T_3^{(0)} & & \\
T_0^{(2)} = S_2 & & T_2^{(1)} & & \ddots & \\
 & T_1^{(2)} & & T_3^{(1)} & & \\
T_0^{(3)} = S_3 & & T_2^{(2)} & & \ddots & \\
 & T_1^{(3)} & & T_3^{(2)} & & \\
T_0^{(4)} = S_4 & \vdots & T_2^{(3)} & \vdots & \ddots & \\
\vdots & \vdots & \vdots & \vdots & &
\end{array} \tag{1.3.3}$$

Si l'on connaît $T_k^{(n)}$ et $T_k^{(n+1)}$ on peut calculer $T_{k+1}^{(n)}$ Ceci est représenté par :

$$\begin{array}{ccc} T_k^{(n)} & \searrow & \\ & & T_{k+1}^{(n)} \\ T_k^{(n+1)} & \nearrow & \end{array} \tag{1.3.4}$$

On dit dans ce cas que le procédé d'extrapolation de **Richardson** est un algorithme de triangle.

En pratique, on ne dispose pas d'un nombre infini de termes de la suite initiale $\{S_n\}$ mais on connaît seulement les valeurs de $S_0, S_1, ..., S_k$ pour k fixé. Le tableau infini (1.3.3) se réduit alors au tableau fini suivant:

$$\begin{array}{lccccc} T_0^{(0)} = S_0 & & & & & \\ & T_1^{(0)} & & & & \\ T_0^{(1)} = S_1 & & T_2^{(0)} & & & \\ & T_1^{(1)} & \vdots & \ddots & & \\ T_0^{(2)} = S_2 & \vdots & \vdots & & T_k^{(0)} & \\ \vdots & \vdots & \vdots & \iddots & & \\ \vdots & \vdots & T_2^{(k-2)} & & & \\ \vdots & T_1^{(k-1)} & & & & \\ T_0^{(k)} = S_k & & & & & \end{array} \tag{1.3.5}$$

Dans (1.3.2) si on prend

$$\begin{aligned} x_n &= \Delta S_n = S_{n+1} - S_n, \quad T_0^{(n)} = S_n, \qquad n = 0, 1, ... \\ T_{k+1}^{(n)} &= \frac{\Delta S_n T_k^{(n+1)} - \Delta S_{n+k+1} T_k^{(n)}}{\Delta S_n - \Delta S_{n+k+1}}, \qquad n, k = 0, 1, ... \end{aligned}$$

La convergence de cet algorithme est obtenue sous des conditions très restrictives sur les suites $\left(T_k^{(n)}\right)$ et $\left(T_{k+1}^{(n)}\right)$ et sur la suite auxiliaire (x_n), elles ne sont vérifiées que dans des cas particuliers. Les théorèmes suivants le montrent :

Théorème 1.3.1 *Soit (x_n) une suite strictement décroissante de nombres réels positifs et tendant vers zéro lorsque n tend vers l'infini. Une condition nécessaire et suffisante pour que :*

$$\lim_{k\to\infty} T_k^{(n)} = \lim_{k\to\infty} S_k$$

pour tout $n = 0, 1, ...$ et pour toute suite convergente $\{S_n\}$ est qu'il existe $a > 1$ tel que :

$$\frac{x_n}{x_{n+1}} \geq a > 1 \qquad n = 0, 1, ...$$

Cette condition est également une condition nécessaire et suffisante pour que :

$$\lim_{n\to\infty} T_k^{(n)} = \lim_{n\to\infty} S_n$$

pour tout $k = 0, 1, ...,$ et pour toute suite convergente $\{S_n\}$.

Remarque 1.3.1 *La condition du théorème ci-dessus entraîne la convergence des colonnes et des diagonales du tableau représentant le procédé d'extrapolation de* ***Richardson****.*

Théorème 1.3.2 *Supposons que la condition du théorème précédent soit satisfaite; alors, une condition nécessaire et suffisante pour que la suite $\left\{T_{k+1}^{(n)}\right\}$ (k fixé) converge plus vite que la suite $\left\{T_k^{(n)}\right\}$ (k fixé) est que:*

$$\lim_{n\longrightarrow\infty} \frac{T_{k+1}^{(n)} - S}{T_k^{(n)} - S} = \lim_{n\longrightarrow\infty} \frac{x_{n+k+1}}{x_n}$$

1.4 Le procédé d'Overholt

C'est une méthode d'accélération de la convergence qui est dû à **Overholt** ces règles sont :

$$\begin{aligned} V_0^{(n)} &= S_n, \qquad n = 0, 1, ... \qquad (1.4.6) \\ V_{k+1}^{(n)} &= \frac{(\Delta S_{n+k})^{k+1} V_k^{(n+1)} - (\Delta S_{n+k+1})^{k+1} V_k^{(n)}}{(\Delta S_{n+k})^{k+1} - (\Delta S_{n+k+1})^{k+1}}, \quad n = 0, 1, ... \end{aligned}$$

Nous allons placer ces quantités dans une table à double entrée : l'indice inférieur désignera une colonne et l'indice supérieur désignera une diagonale descendante.

En voit que (1.4.6) est analogue au schéma du procédé d'extrapolation de **Richardson** (schéma du triangle).

Remarque 1.4.1 ***1/*** *La différence du procédé de* ***Richardson*** *avec le procédé d'****Overholt*** *provient de la $1^{ère}$ colonne; en effet le calcul de $V_1^{(k-2)}$ nécessite la connaissance de S_{k-2}, S_{k-1} et S_k alors que $T_1^{(k-2)}$ peut être calculé seulement à partir de S_{k-2}, S_{k-1}.*

*2/ Le procédé d'**Overholt** est une généralisation du procédé Δ^2**d'Aitken,** on a en effet :*

$$V_1^{(n)} = \frac{S_n S_{n+2} - S_{n+1}^2}{S_{n+2} - 2S_{n+1} + S_n} = \varepsilon_2^{(n)},\ n = 0, 1, ...$$

1.5 Le procédé Δ^2d'Aitken

Ce procédé consiste à transformer la suite $\{S_n\}$ en une suite que nous noterons $\{e_1(S_n)\}$ définie par :

$$e_1(S_n) = \frac{S_n S_{n+2} - S_{n+1}^2}{S_{n+2} - 2S_{n+1} + S_n}, n = 0, 1, ...,\ \text{tq}\ S_{n+2} - 2S_{n+1} + S_n \neq 0.$$

La formule précédente peut être écrire comme suit :

$$e_1(S_n) = S_{n+1} - \frac{\Delta S_{n+1}}{\frac{\Delta S_{n+1}}{\Delta Sn} - 1}$$

D'où immédiatement le:

Théorème 1.5.1 *S'il existe deux constantes α et β avec $\alpha < 1 < \beta$ tels que*

$$\frac{\Delta S_{n+1}}{\Delta Sn} \notin [\alpha, \beta],$$

pour tout $n > N$ et si:

$$\lim_{n\to\infty} S_n = S$$

alors:

$$\lim_{n\to\infty} e_1(S_n) = S$$

On a également le:

Théorème 1.5.2 *Si on applique le procédé Δ^2**d'Aitken** à une suite $\{S_n\}$ qui converge vers S et si:*

$$\lim_{n\to\infty} = \frac{S_{n+1} - S}{S_n - S} = \lim_{n\to\infty} \frac{\Delta S_{n+1}}{\Delta S_n} = a \neq 1$$

Alors la suite $\{e_1(S_n)\}$ converge vers S et cela plus vite que la suite $\{S_{n+1}\}$.

*On voit donc que le procédé Δ^2**d'Aitken** permet d'accélérer la convergence d'une suite.*

1.6 La transformation de Shanks

Shanks a recherché quelle transformation de suite est susceptible de généraliser la relation:

$$a_0 (S_n - S) + a_1 (S_{n+1} - S) = 0, \forall n > N \tag{1.6.7}$$

sous la forme :

$$\sum_{i=0}^{k} a_i (S_{n+1} - S) = 0, \forall n > N \tag{1.6.8}$$

il a évidemment trouvé qu'il fallait qu'on ait:

$$\begin{vmatrix} S_n - S & .. & .. & S_{n+k} - S \\ S_{n+1} - S & .. & .. & S_{n+k+1} - S \\ .. & .. & .. & .. \\ S_{n-k} - S & .. & .. & S_{n+2k} - S \end{vmatrix} = 0$$

En d'autres termes, il a trouvé une transformation, notée $\varepsilon_{2k}^{(n)}$, qui transforme la suite S_n obéissant (1.6.7) en la suite constante $\{S\}$:

$$\varepsilon_{2k}^{(n)} = \frac{\begin{vmatrix} S_n & S_{n+1} & ... & S_{n+k} \\ \Delta S_n & \Delta S_{n+1} & ... & \Delta S_{n+k} \\ ... & ... & ... & \dots \\ \Delta S_{n+k-1} & \Delta S_{n+k} & ... & \Delta S_{n+2k+1} \end{vmatrix}}{\begin{vmatrix} 1 & 1 & ... & 1 \\ \Delta S_n & \Delta S_{n+1} & ... & \Delta S_{n+k} \\ ... & ... & ... & ... \\ \Delta S_{n+k-1} & \Delta S_{n+k} & .. & \Delta S_{n+2k+1} \end{vmatrix}}$$

Lorsque $k = 1$, on retrouve évidemment le procédé Δ^2**d'Aitken**. Lorsque $k > 1$, on découvre de nouveaux procédés d'accélération plus puissants que lui. Voyons à quels types de suites ilS s'appliquent préférentiellement en determinant leur nOyau.

La relation (1.6.7) montre clairement que $S_n - S$ obéit à une récurrence linéaire à coefficients constants dont la relation générale revêt nécessairement la forme d'exponentielles-polynômes.

Le noyau de la transformation de **Shanks** est donc constitué par l'ensemble des suites du type:

$$S_n = S + \sum_{i=1}^{N} p_i(n)\, e^{z_i n}$$

$p_i(n)$:les polynômes en n et où les z_i sont des nombres quelconques réels ou complexes.

La grosseur relative de ce noyau explique son importance pratique.

1.7 Algorithme de Wynn(1956)

L'algorithme de **Wynn** réalise la transformation de **Shanks** par une récurrence qui évite le calcul de déterminants:

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

Nous admettons alors le résultat suivant:

$\varepsilon_{2k}^{(n)}$ est le $T_n^{(k)}$ de la méthode de **Shanks**

1.8 Les algorithmes scalaires

Dans ce paragraphe, nous allons donner les règles des algorithmes qui permettent d'accélérer la convergence d'une suite de nombres réels ou complexes.

1.8.1 L'ε-algorithme et ses généralisations:

Une méthode d'accélération de la convergence la plus puissante comme a l'heure actuelle. L'ε-algorithme est une généralisation d'une méthode célèbre d'accélération de la convergence: **le procédé Δ^2d'Aitken**, dans cet algorithme on calcul également des quantités avec deux indices $\varepsilon_k^{(n)}$.On utilise pour cela les relations:

$$\begin{aligned} \varepsilon_{-1}^{(n)} = 0, \quad \varepsilon_0^{(n)} = S_n, \quad n = 0, 1, .. \\ \varepsilon_{k+1}^{(n)} = \varepsilon_{k-1}^{(n)} + \frac{1}{\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}}, \quad n, k = 0, 1, ... \end{aligned} \tag{1.8.9}$$

la relation(1.8.9) relie des quantités situées aux quatre sommets d'un losange:

$$\begin{array}{ccc} & \varepsilon_k^{(n)} & \\ \nearrow & & \searrow \\ \varepsilon_{k-1}^{(n+1)} & & \varepsilon_{k+1}^{(n)} \\ \searrow & & \nearrow \\ & \varepsilon_k^{(n+1)} & \end{array} \tag{1.8.10}$$

Si l'on connait $\varepsilon_{k-1}^{(n+1)}$, $\varepsilon_k^{(n)}$ et $\varepsilon_k^{(n+1)}$ la relation (1.8.9) permet de calculer $\varepsilon_{k+1}^{(n)}$(ce qui signifient (1.8.10)).

La relation (1.8.9) permet donc de progresser de la gauche vers la droite et de haut en bas à partir des conditions initiales $\varepsilon_{-1}^{(n)} = 0$ et $\varepsilon_0^{(n)} = S_n$ pour $n = 0, 1, ...$

-la théorie de l'ε-algorithme montre que les quantités d'indice inférieur impair ne sont que des calculs intermédiaires. Seule sont intéressantes les quantités d'indice inférieur pair, pour calculer directement les quantités d'indice inférieur impair. On peut utiliser la règle de la croix:

$$\begin{aligned} \varepsilon_{-2}^{(n)} &= \infty,\ \varepsilon_0^{(n)} = S_n && n = 0 \\ \frac{1}{\varepsilon_{2k-2}^{(n-1)} - \varepsilon_{2k}^{(n)}} + \frac{1}{\varepsilon_{2k+2}^{(n+1)} - \varepsilon_{2k}^{(n)}} &= \frac{1}{\varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)}} + \frac{1}{\varepsilon_{2k}^{(n-1)} - \varepsilon_{2k}^{(n)}}, && \text{pour } n, k = 0, 1, ... \end{aligned} \tag{1.8.11}$$

la relation (1.8.11) relie des quantités situées aux quatre sommets et au centre d'une croix:

$$\begin{array}{ccc} & \varepsilon_{2k}^{(n-1)} & \\ \varepsilon_{2k-2}^{(n+1)} & \varepsilon_{2k}^{(n)} & \varepsilon_{2k+2}^{(n-1)} \\ & \varepsilon_{2k}^{(n+1)} & \end{array} \tag{1.8.12}$$

Si on symbolise le tableau (1.8.12) par:

$$\begin{array}{ccc} & N & \\ W & C & E \\ & S & \end{array} \tag{1.8.13}$$

Alors la règle de la croix (1.8.11) peut s'écrire:

$$\frac{1}{N - C} + \frac{1}{S - C} = \frac{1}{W - C} + \frac{1}{E - C} \tag{1.8.14}$$

Remarque 1.8.1 *Si l'on introduit dans la relation de l'ε-algorithme une suite auxiliaire $\{x_n\}$ alors on peut obtenir deux généralisations intéressantes possèdent les mêmes caractéristiques que l'ε-algorithme, ils sont des algorithmes de losange où les colonnes pairs sont intéressantes.*

Généralisations

La première généralisation: Cette généralisation se fait en introduisant une suite auxiliaire $\{x_n\}$, ces règles sont:

$$\begin{array}{lll} \varepsilon_{-1}^{(n)} = 0, & \varepsilon_{-1}^{(n)} = S_n, & n = 0, 1, ... \\ \varepsilon_{k+1}^{(n)} = & \varepsilon_{k-1}^{(n+1)} + \frac{\Delta x_n}{\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}} & n, k = 0, 1, .. \\ \Delta x_n = & x_{n+1} - x_n & \forall n \end{array} \tag{1.8.15}$$

La deuxième généralisation:

Cette généralisation se fait aussi en introduisant une suite auxiliaire $\{x_n\}$, ces règles sont:

$$\begin{array}{lll} \varepsilon_{-1}^{(n)} = 0, & \varepsilon_{0}^{(n)} = S_n, & n = 0, 1, ... \\ \varepsilon_{k+1}^{(n)} = & \varepsilon_{k-1}^{(n+1)} + \frac{\Delta x_{n+k}}{\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}} & n, k = 0, 1, .. \end{array} \tag{1.8.16}$$

Dans ces deux généralisations, le calcul de $\varepsilon_{2k}^{(n)}$ nécessite la connaissance de $S_n, ..., S_{n+2k}$ et de $x_n, ..., x_{n+2k}$ en ce qui concerne la suite auxiliaire.

Donnons quelques résultats de l'ε-algorithme scalaire :

Définition 1.8.1 *On dit que la suite $\{S_n\}$ est totalement monotone si:*

$$\begin{array}{rcll} (-1)^k \Delta^k S_n & \geqslant & 0 & n, k = 0, 1, ... \\ où\ \ \Delta^0 S_n & = & S_n & \\ et\ pour\ tout\ n,\ \ \Delta^{k+1} S_n & = & \Delta^k S_{n+1} - \Delta^k S_n;\ n, k = 0, 1, .. & \end{array}$$

Théorème 1.8.1 *Si on applique l'ε-algorithme à une suite $\{S_n\}$ qui converge vers S et s'il existe deux constantes $a \neq 0$ et b telle que la suite $\{aS_n + b\}$ soit totalement monotone, alors :*

$$\begin{array}{rcll} \lim\limits_{n \longrightarrow \infty} \varepsilon_{2k}^{(n)} & = & S & k = 0, 1, ...fixé \\ \lim\limits_{k \longrightarrow \infty} \varepsilon_{2k}^{(n)} & = & S & n = 0, 1, ...fixé \end{array}$$

Définition 1.8.2 *On dit que la suite* $\{S_n\}$ *est totalement oscillante si la suite* $\{(-1)^n S_n\}$ *est totalement monotone.*

Théorème 1.8.2 *Si on applique l'*ε*-algorithme à une suite* $\{S_n\}$ *qui converge vers* S *et s'il existe deux constantes* $a \neq 0$ *et* b *telle que la suite* $\{aS_n + b\}$ *soit oscillante, alors :*

$$\begin{aligned} \lim_{n\longrightarrow\infty} \varepsilon_{2k}^{(n)} &= S \qquad k = 0, 1, ...\textit{fixé} \\ \lim_{k\longrightarrow\infty} \varepsilon_{2k}^{(n)} &= S \qquad n = 0, 1, ...\textit{fixé} \end{aligned}$$

Remarque 1.8.2 *Les deux derniers théorèmes n'accélèrent pas la convergence, mais montrent que l'*ε*-algorithme converge.*

De plus on remarque que :

$$\begin{aligned} \lim_{n\longrightarrow\infty} \frac{\varepsilon_{2k}^{(n)} - S}{S_{n+2k} - S} &= 0 \qquad k = 1, 2, ...\textit{fixé} \\ & \quad \textit{et} \\ \lim_{k\longrightarrow\infty} \frac{\varepsilon_{2k}^{(n)} - S}{S_{n+2k} - S} &= 0 \qquad n = 0, 1, ...\textit{fixé} \end{aligned}$$

Ces résultats montrent que la condition du théorème (1.8.3) *est satisfaite donc la suite* $\varepsilon_{2k}^{(n)}$ *converge plus vite que la suite* (S_{n+1}) *(totalement monotone ou totalement oscillante).*

1.8.2 Le ρ-algorithme:

Les règles du l'ρ-algorithme sont presque semblables à celles de l'ε-algorithme et de ses généralisations, il possède les mêmes caractéristiques que l'ε-algorithme (les colonnes pairs sont intéressantes).

Le calcul des quantités $\rho_k^{(n)}$ s'effectue selon un schéma analogue au tableau (1.8.10).

La théorie d'ε-algorithme et du ρ-algorithme est complètement différente, les règles de ρ-algorithme font intervenir une suite auxiliaire $\{x_n\}$:

$$\begin{aligned} \rho_{-1}^{(n)} = 0, \qquad \rho_0^{(n)} = S_n, \qquad & n = 0, 1, ... \\ \rho_{k+1}^{(n)} = \quad \rho_{k-1}^{(n+1)} + \frac{x_{n+k+1} - x_n}{\rho_k^{(n+1)} - \rho_k^{(n)}} \quad & n, k = 0, 1, .. \end{aligned} \tag{1.8.17}$$

La théorie du ρ-algorithme montre que la suite $\{x_n\}$ doit tendre vers ∞ quand n tend vers ∞

1.8.3 Le θ-algorithme

Le θ-algorithme est la méthode qui fait le lien entre l'ε-algorithme et le ρ-algorithme, ces règles sont:

$$\begin{aligned}
\theta_{-1}^{(n)} &= 0 \qquad \theta_0^{(n)} = S_n \qquad n = 0, 1, ... \\
\theta_{2k+1}^{(n)} &= \theta_{2k-1}^{(n+1)} + D_{2k}^{(n)} \qquad n, k = 0, 1... \\
\theta_{2k+2}^{(n)} &= \frac{D_{2k+1}^{(n+1)}\theta_{2k}^{(n+1)} - D_{2k+1}^{(n)}\theta_{2k}^{(n+2)}}{D_{2k+1}^{(n+1)} - D_{2k+1}^{(n)}} \\
avec \quad : \quad D_k^n &= \frac{1}{\left(\theta_k^{(n+1)} - \theta_k^{(n)}\right)} \qquad \text{pour } n, k = 0, 1, ...
\end{aligned} \tag{1.8.18}$$

la première relation de (1.10.18) relie des quantités situées aux quatre sommets d'un losange:

$$\begin{array}{ccccc}
 & & \theta_{2k}^{(n)} & & \\
 & \nearrow & & \searrow & \\
\theta_{2k-1}^{(n+1)} & & & & \theta_{2k+1}^{(n)} \\
 & \searrow & & \nearrow & \\
 & & \theta_{2k}^{(n+1)} & &
\end{array} \tag{1.8.19}$$

la seconde relation de (1.10.18) permet de calculer $\theta_{2k+2}^{(n)}$ à partir de $\theta_k^{(n+1)}$, $\theta_{2k}^{(n+2)}$, $\theta_{2k+1}^{(n)}, \theta_{2k+1}^{(n+1)}$ et $\theta_{2k+1}^{(n+2)}$:

$$\begin{array}{ccccccc}
 & & \theta_{2k+1}^{(n)} & & & & \\
 & \nearrow & & \searrow & & & \\
\theta_{2k}^{(n+1)} & & & & \searrow & & \\
 & \searrow & & & & \searrow & \\
 & & \theta_{2k+1}^{(n+1)} & \rightarrow & \longrightarrow & \longrightarrow & \theta_{2k+2}^{(n)} \\
 & \nearrow & & & & \nearrow & \\
\theta_{2k}^{(n+2)} & & & & \nearrow & & \\
 & \searrow & & \nearrow & & & \\
 & & \theta_{2k+1}^{(n+2)} & & & &
\end{array} \tag{1.8.20}$$

Ce n'est pas un algorithme de losange, la théorie du θ-algorithme montre que les colonnes pairs sont intéressantes, les colonnes impairs ne sont que des calculs intermédiaires.

Les relations (1.10.18) permet de calculer $\theta_{2k+2}^{(n)}$ à partir de $\theta_{2k}^{(n)}$,....., $\theta_{2k}^{(n+1)}$ selon le schéma:

$$\begin{array}{ccc} \theta_{2k}^{(n)} & \searrow & \\ \theta_{2k}^{(n+1)} & \searrow & \\ & & \theta_{2k+2}^{(n)} \\ \theta_{2k}^{(n+2)} & \nearrow & \\ \theta_{2k}^{(n+3)} & \nearrow & \end{array} \tag{1.8.21}$$

1.9 Les algorithmes non scalaires

Nous allons donnés les règles des algorithmes qui permettent d'accélérer la convergence d'une suite de matrice ou de vecteurs dont les éléments sont des nombres réels ou complexes.

Les vecteurs de la suite $\{S_n\}$ appartiendront à $\mathbb{C}^p$ et les matrices sont des matrices carrées de p lignes et p colonnes.

Nous parlons dans ce paragraphe de l'ε-algorithme et le θ- algorithme.

1.9.1 L'ε-algorithme vectoriel

Soit$\{S_n\}$ une suite de vecteurs de $\mathbb{C}^p$, les règles de l'ε-algorithme vectoriel sont:

$$\begin{aligned} \varepsilon_{-1}^{(n)} &= 0 \in \mathbb{C}^p, \varepsilon_0^{(n)} = S_n \in \mathbb{C}^p \\ \varepsilon_{k+1}^{(n)} &= \varepsilon_{k-1}^{(n+1)} + \left[\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}\right]^{-1} \qquad n, k = 0, 1... \end{aligned} \tag{1.9.22}$$

pour appliquer cet algorithme, nous somme besoin de définir un inverse pour le vecteur $y \in \mathbb{C}^p$ qui est : $y^{-1} \in \mathbb{C}^p$ tel que :

$$y^{-1} = \frac{\bar{y}}{(y, y)} \tag{1.9.23}$$

$\bar{y}$: vecteur où les composantes sont réelles ou complexes.

(y, y): produit scalaire dans $\mathbb{C}^p$.

Soit $y_i (i = 1.....p)$ les composantes de y alors :

$$(y, y) = \sum_{i=1}^{p} y_i \bar{y}_i = \sum_{i=1}^{p} \mid y_i \mid^2 \tag{1.9.24}$$

Pour cet algorithme on fait tient que l'inverse d'un vecteur définit comme suit: (les propriétés)

$$(y^{-1})^{-1} = y$$

$$(y, y^{-1}) = 1$$

$$(y^{-1})^{-1} = -(y^{-1})$$

nous pourrons utiliser la définition suivante:

$$y^{-1} = \frac{y}{(y, y)} \tag{1.9.25}$$

dans (1.12.22) au lieu de (1.12.23).

1.9.2 L'ε-algorithme vectoriel normé

Soit $\{S_n\}$ une suite de vecteurs de $\mathbb{C}^p$. Ces règles sont presque les mêmes que l'ε-algorithme vectoriel sauf que le vecteur inverse y^{-1} se change, il est définit comme suit:

$$y^{-1} = \frac{\bar{y}}{\| y \|^2} \quad /y \in \mathbb{C}^p \tag{1.9.26}$$

$\| y \|$: n'importe quelle norme vectoriel; on peut prendre la plus simple à calculer:

$$\| y \| = \max_{1 \leqslant i \leqslant p} |y_i|$$

La relation (1.13.26) vérifie: $(y^{-1})^{-1} = y$.

Le produit scalaire (y, y^{-1}) n'est plus toujours égal à 1.

Afin de concentrer le maximum l'énoncé des résultats pour l'ε-algorithme vectoriel normé, nous allons appeler H_1 l'ensemble des hypothèses suivantes :

-$\{S_n\}$ est une suite de vecteurs de $\mathbb{C}^p$;

-$\{a_n\}$ est une suite de nombre réels tel que $a_n \neq 0$ pour tout n et que $\{\Delta a_n\}$ converge vers 0. Nous supposerons de plus qu'il existe deux constantes α et β telles que :

$$\alpha < 1 < \beta \quad \text{et} \quad a_n \notin [\alpha, \beta] \qquad \forall n$$

-La suite $\{S_n\}$ vérifie :

$$\begin{aligned} S_{n+1} - S_n &= a_n (S_n - S) \quad n = 0, 1, ... \\ \text{avec} \quad S &\in \mathbb{C}^p \end{aligned}$$

Théorème 1.9.1 *Si on applique l'ε-algorithme vectoriel normé à une suite $\{S_n\}$ qui vérifie les hypothèses H_1 et qui converge vers S alors :*

$$\lim_{n\longrightarrow\infty} \varepsilon_2^{(n)} = S$$

Si de plus, la suite $\{a_n\}$ admet une limite, alors :

$$\lim_{n\longrightarrow\infty} \frac{\left\|\varepsilon_2^{(n)} - S\right\|}{\|S_{n+1} - S\|} = 0$$

Appelons maintenant H_2 l'ensemble des hypothèses suivantes :

-$\{S_n\}$ est une suite de vecteurs de $\mathbb{C}^p$;

-$\{e_n\}$ est une suite de vecteurs de $\mathbb{C}^p$ qui converge vers zéro lorsque n tend vers l'infini ;

-$y \in \mathbb{C}^p$ et $y \neq 0$;

-a est un nombre réel tel que $0 < a < 1$;

-la suite $\{S_n\}$ vérifie :

$$S_n - S = a^n (y + e_n) \qquad n = 0, 1, ... \qquad \text{avec} \quad S \in \mathbb{C}^p$$

Théorème 1.9.2 *Si on applique l'ε-algorithme vectoriel normé à une suite $\{S_n\}$ qui vérifie les hypothèses H_2 alors :*

$$\begin{aligned} \lim_{n\longrightarrow\infty} \varepsilon_2^{(n)} &= S \\ &\textit{et} \quad : \\ \lim_{n\longrightarrow\infty} \frac{\left\|\varepsilon_2^{(n)} - S\right\|}{\|S_{n+k} - S\|} &= 0 \quad \forall k \geqslant 0 \quad \textit{fixé} \end{aligned}$$

Les deux derniers théorèmes montrent sous certaines conditions que l'ε-algorithme vectoriel normé (vectoriel) converge vers la limite de la suite originale et que cette convergence est accélérée

1.9.3 L'ε-algorithme topologique

Cet algorithme permet de traiter des suites de vecteurs, ces règles sont plus complexes que l'ε-algorithme vectoriel, il possède des propriétés supplémentaires.

Il existe deux ε-algorithmes topologique:

Les règles du premier ε-algorithme topologique

$$\begin{cases} \varepsilon_{-1}^{(n)} = 0 \in \mathbb{C}^p \qquad \varepsilon_0^{(n)} = S_n \in \mathbb{C}^p & n = 0, 1, ... \\ \varepsilon_{2k+1}^{(n)} = \varepsilon_{2k-1}^{(n+1)} + \frac{y}{\left(y, \varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)}\right)} & n, k = 0, 1, .. \\ \varepsilon_{2k+2}^{(n)} = \varepsilon_{2k}^{(n+1)} + \frac{\varepsilon_{2k}^{(n+2)} - \varepsilon_{2k}^{(n+1)}}{\left(\varepsilon_{2k+1}^{(n)} - \varepsilon_{2k+1}^{(n)}, \varepsilon_{2k}^{(n+2)} - \varepsilon_{2k}^{(n+1)}\right)} & n, k = 0, 1, ... \end{cases} \tag{1.9.27}$$

On plaçant ces vecteurs dans un tableau à double entrée:

·indice inférieur désigne une colonne.

·indice supérieur désigne une diagonale descendante.

La première relation de (1.14.27) relie des vecteurs situés aux quatre sommets d'un losange:

$$\begin{array}{ccc} & \varepsilon_{2k}^{(n)} & \\ \nearrow & & \searrow \\ \varepsilon_{2k-1}^{(n+1)} & & \varepsilon_{2k+1}^{(n)} \\ \searrow & & \nearrow \\ & \varepsilon_{2k}^{(n+1)} & \end{array} \tag{1.9.28}$$

La seconde relie des vecteurs disposés suivant le schéma suivant:

$$\begin{array}{ccc} \varepsilon_{2k}^{(n)} & & \\ \searrow & & \\ & \varepsilon_{2k+1}^{(n)} & \\ \nearrow & & \searrow \\ \varepsilon_{2k}^{(n+1)} & & \varepsilon_{2k+2}^{(n)} \\ \searrow & & \nearrow \\ & \varepsilon_{2k+1}^{(n+1)} & \end{array} \tag{1.9.29}$$

La relation (1.14.27) permet de progresser dans le tableau du premier ε-algorithme topologique de la gauche vers la droite et de haut vers le bas à partir des conditions initiales $\varepsilon_{-1}^{(n)} = 0 \in \mathbb{C}^p$ et $\varepsilon_0^{(n)} = S_n \in \mathbb{C}^p, n = 0, 1, ..$

Les règles du second $\varepsilon-$algorithme topologique:

$$\begin{cases} \varepsilon_{-1}^{(n)} = 0 \in \mathbb{C}^p, \varepsilon_0^{(n)} = S_n \in \mathbb{C}^p & , n = 0, 1... \\ \varepsilon_{2k+1}^{(n)} = \varepsilon_{2k-1}^{(n+1)} + \frac{y}{\left(y, \varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)}\right)} & , n, k = 0, 1... \\ \varepsilon_{2k+2}^{(n)} = \varepsilon_{2k}^{(n+1)} + \frac{\varepsilon_{2k}^{(n+2)} - \varepsilon_{2k}^{(n+1)}}{\left(\varepsilon_{2k+1}^{(n+1)} - \varepsilon_{2k+1}^{(n)}, \varepsilon_{2k}^{(n+2)} - \varepsilon_{2k}^{(n+1)}\right)} & , n, k = 0, 1, ... \end{cases} \tag{1.9.30}$$

Le tableau construit est identique à celui du premier ε-algorithme topologique.

La première règle de (1.14.30) relie des vecteurs selon le schéma (1.14.28), pour la deuxième on a:

$$\begin{array}{ccccc} & & \varepsilon_{2k+1}^{(n)} & & \\ & \nearrow & & \searrow & \\ \varepsilon_{2k}^{(n+1)} & & & & \varepsilon_{2k+2}^{(n)} \\ & \searrow & & \nearrow & \\ & & \varepsilon_{2k+1}^{(n+1)} & & \\ & \nearrow & & & \\ \varepsilon_{2k}^{(n+2)} & & & & \end{array} \tag{1.9.31}$$

1.9.4 Le $\theta-$algorithme topologique

On définit deux $\theta-$algorithme topologiques :

Les règles du premier $\theta-$algorithme topologique

$$\begin{aligned} \theta_{-1}^{(n)} &= 0, \theta_0^{(n)} = s_n, n = 0, 1, ... \\ \theta_{2k+1}^{(n)} &= \theta_{2k-1}^{(n+1)} + D_{2k}^{(n)}, n, k = 0, 1, ... \end{aligned}$$

avec:

$$\begin{aligned} D_{2k}^{(n)} &= \frac{y}{\left(y, \theta_{2k}^{(n+1)} - \theta_{2k}^{(n)}\right)} \\ \theta_{2k+2}^{(n)} &= \theta_{2k}^{(n+1)} + w_k^{(n)} D_{2k+1}^{(n)} \quad n, k = 0, 1, ... \end{aligned} \tag{1.9.32}$$

avec:

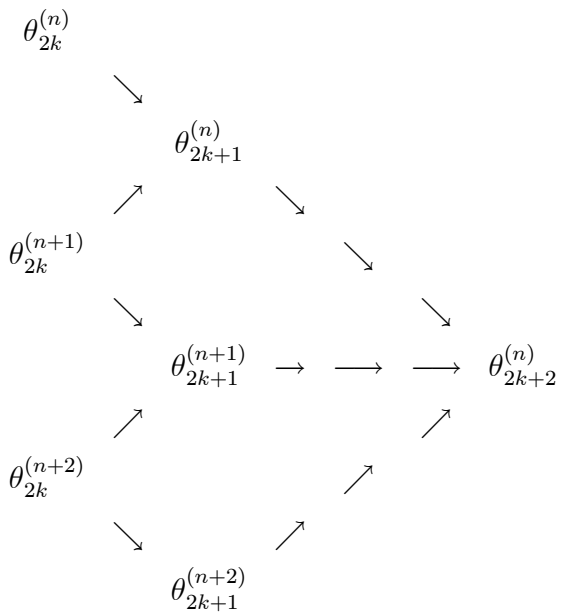
$$D_{2k+1}^{(n)} = \frac{\theta_{2k}^{(n+1)} - \theta_{2k}^{(n)}}{\left(\theta_{2k+1}^{(n+1)} - \theta_{2k+1}^{(n)}, \theta_{2k}^{(n+1)} - \theta_{2k}^{(n)}\right)}$$

et:

$$w_k^{(n)} = -\frac{\left(\tau, \theta_{2k}^{(n+2)} - \theta_{2k}^{(n+1)}\right)}{\left(\tau, D_{2k+1}^{(n+1)} - D_{2k+1}^{(n)}\right)}$$

La première relation de (1.15.32) relie des quantités situées aux quatre sommets d'un losange (schéma 1.10.19).

La seconde relation de (1.15.32) permet de calculer $\theta_{2k+2}^{(n)}$ à l'aide des vecteurs, ils sont reliés suivant le schéma:



(1.9.33)

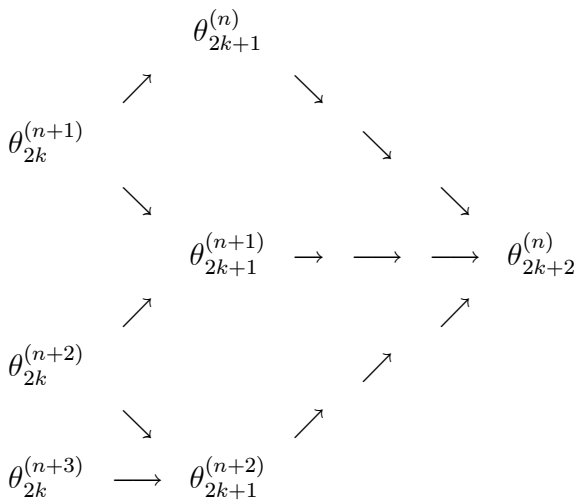
Dans cet algorithme les colonnes paires sont intéressantes.

Les règles du deuxième $\theta-$algorithme topologique

Ces règles sont analogue aux (1.15.32), seule la définition $D_{2k+1}^{(n)}$:

$$D_{2k+1}^{(n)} = \frac{\theta_{2k}^{(n+2)} - \theta_{2k}^{(n+1)}}{\left(\theta_{2k+1}^{(n+1)} - \theta_{2k+1}^{(n)}, \theta_{2k}^{(n+2)} - \theta_{2k}^{(n+1)}\right)} \tag{1.9.34}$$

Le calcul des colonnes impaires s'effectue toujours selon (1.10.19) pour les colonnes paires on a:



(1.9.35)

1.9.5 L'$\varepsilon-$algoritme matriciel

Cet algorithme étudie le cas ou la suite $\{S_n\}$ à transformer est une suite de matrice carrée complexe d'ordre p.

O: la matrice carrée d'ordre p dont tous les éléments sont nuls, cet algorithme est défini par les relations suivante:

$$\begin{aligned} \varepsilon_{-1}^{(n)} &= 0, \varepsilon_0^{(n)} = S_n \quad ; \quad n = 0, 1, ... \\ \varepsilon_{k+1}^{(n)} &= \varepsilon_{k-1}^{(n+1)} + \left[\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}\right]^{-1} \quad ; n, k = 0, 1, ... \end{aligned} \tag{1.9.36}$$

$\left[\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}\right]$:la matrice inverse de la matrice $\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}$, et il fonctionne comme l'ε-algoritme scalaire.

1.10 Les algorithmes confluents

Dans ce paragraphe on va étudier les algorithmes qui permettent de transformer une fonction f en une autre fonction g.tel que: $g(t)$ admet une limite quand t tend vers ∞ (finis ou égale à S).

On dit que g converge plus vite que f si:

$$\lim_{t\to\infty} \frac{g(t) - s}{f(t) - s} = 0$$

Dans cet algorithme on remplace la variable discrète n dans les algorithmes scalaires par la variable continue $x = t + nh$ et on effectue les changements de fonctions correspondants et on fait tendre h vers le zéro.

Les formes confluentes de cet algorithme sont:

1.10.1 Forme confluente de l'ε-algoritme

Ces règles sont:

$$\begin{aligned} \varepsilon_{-1}(t) &= 0 \quad \varepsilon_0(t) = f(t) \\ \varepsilon_{k+1}(t) &= \varepsilon_{k-1}(t) + \frac{1}{\varepsilon_k^{'}} \quad k = 0, 1, ... \end{aligned} \tag{1.10.37}$$

$\varepsilon_0(t) = f(t)$ permet de construire $\varepsilon_1(t)$ ...

Le calcul de $\varepsilon_1(t)$ nécessite $f^{'}(t)$, le calcul de $\varepsilon_2(t)$ nécessite $f(t)$, $f^{'}(t)$, $f^{''}(t)$ et ainsi de suite.

La théorie d'ε-algorithme confluent montre que sauf les fonctions avec indice paire sont intéressantes.

Pour t fixé, si on connaît $f(t)$, $f^{'}(t)$, ..., $f^{(2k)}(t)$, la relation (1.17.37) permet de calculer $\varepsilon_{2k}(t)$ (il faut intervenir $\varepsilon_0^{'}(t)$, ..., $\varepsilon_{2k-1}^{'}(t)$).

1.10.2 Forme confluente du $\rho-$algorithme

Cet algorithme s'obtient à partir de la forme discrète de l'algorithme, la suite auxiliaire $\{x_n\}$ qui intervient dans cet algorithme est définie par: $x_n = a + nk$. On obtient:

$$\begin{aligned} \rho_{-1}(t) &= 0 \qquad \rho_0(t) = f(t) \\ \rho_{k+1}(t) &= \rho_{k-1}(t) + \frac{k+1}{\rho_k^{'}(t)} \quad k = 0, 1, ... \end{aligned} \tag{1.10.38}$$

Sauf les fonctions avec indice paire sont intéressantes, si on connaît les valeurs $f(t), ...,$ $f^{2k}(t)$, pour t fixé, la relation (1.14.38) permet de calculer $\rho_{2k}(t)$.

1.10.3 Forme confluente du $\theta-$algorithme

On a:

$$\begin{aligned} \theta_{-1}(t) &= 0, \theta_0(t) = f(t) \\ \theta_{2k+1}(t) &= \theta_{2k-1}(t) + \frac{1}{\theta'_{2k}(t)}, \qquad k = 0, 1, ... \\ \theta_{2k+2}(t) &= \theta_{2k}(t) + \frac{\theta'_{2k}(t)\,\theta'_{2k+1}(t)}{\theta''_{2k+1}(t)}, \quad k = 0, 1, ... \end{aligned} \tag{1.10.39}$$

Seulement les fonctions d'indice paire sont intéressantes.

Le calcul de θ_1 nécessite f' et le calcul de θ_2 nécessite f, f', θ'_1, θ''_1 si on connait les valeurs $f(t)$, $f'(t)$, ..., $f^{(3k)}(t)$, pour t fixé on peut calculer $\theta_{2k}(t)$.

1.10.4 Forme confluent du procédé d'overholt

À l'aide des mêmes changements de variables, on peut obtenir la forme confluente du procédé d'**Overholt**, on a:

$$\begin{aligned} v_0(t) &= f(t) \\ v_{k+1}(t) &= v_k(t) - \frac{f'(t)}{f''(t)} \frac{v'_k(t)}{k+1}, k = 0, 1, ... \end{aligned} \tag{1.10.40}$$

Pour calculer $v_1(t)$, il faut connaître $f(t)$, $f'(t)$, $f''(t)$ si on connait $v_1(t)$, $v'_1(t)$ on peut calculer $v_2(t)$ qui nécessite la connaissance de $f(t)$, $f(t)'$, $f(t)''$.

Chapitre 2

Utilisations et applications des méthodes d'accélération de la convergence

Ce chapitre est consacré aux applications des méthodes d'accélération qui ont été étudiées dans le premier chapitre, il traite aussi leur utilisations pour accélérer la convergence des suites et des fonctions, ainsi on a étudiés quelques applications: résolution des équations différentielles, les systèmes d'équations linéaire et non linéaire, les équations matricielles, ...

Ce chapitre est plus pratique, il montre comment utiliser les méthodes d'accélération correctement, et nous l'illustrons par de nombreux exemples numériques.

2.1 Résolution d'une équation

L'utilisation la plus simple des méthodes d'accélération de la convergence consiste à accélérer purement et simplement les méthodes itératives d'une équation.

Soit par exemple à rechercher x tel que $x = F(x)$ où F est une fonction réelle d'une variable réelle. Une des méthodes les plus couramment employée pour résoudre cette équation est la méthode **des approximations successives**.

2.1.1 Méthode des approximations successives

Dans les méthodes d'approximations successives, l'équation $F(x) = 0$ est remplacée par l'étude d'une suite numérique convergente

$$\begin{aligned} & x_0 \text{ donné} \\ x_{n+1} &= F(x) \end{aligned}$$

qui permet d'obtenir en un nombre fini d'itérations une solution approchée de l'équation. En général, on prend $\varphi(x) = x - cF(x)$ Dans la méthode de Lagrange, on remplace la fonction F par le segment de droite passant par les points $(a, F(a))$et $(b, F(b))$

$$\varphi(x) = a - F(a)\frac{x-a}{F(x)-F(a)}$$

Dans la méthode de Newton, on remplace la fonction F entre les points d'abscisse a et b par la tangente à la courbe en ces points

$$\varphi(x) = x - \frac{F(x)}{F'(x)}$$

On sait qu'une telle méthode converge vers x si F est une contraction, c'est-à-dire s'il existe une constante k telle que:

$$\begin{aligned} 0 &\leqslant k < 1 \\ & \text{et} \\ |F(y) - F(z)| &\leqslant k|y-z| \qquad \forall\ y, z \end{aligned}$$

Si la suite $\{x_n\}$ est du premier ordre; on voit alors que l'accélération de la convergence qui est obtenue est très appréciable.

Par contre, si la suite $\{x_n\}$ est d'ordre supérieur à un, alors l'ε-algorithme n'accélère pas la convergence, comme si on utilise la méthode de **Newton**

2.1.2 Méthode de Newton

$$\begin{aligned} & x_0 \text{ donné} \\ x_{n+1} &= x_n - f(x_n)/f'(x_n) \qquad n = 0, 1, ... \end{aligned}$$

Cet algorithme est initié à l'aide d'un seul point. Nous verrons plus loin qu'il donne lieu à une convergence beaucoup plus rapide.

Pour calculer la racine x de f.

· Si x est une racine simple :

Soit $f(x) = x - e^{-x}$, on trouve :

Méthode de **Newton**	ε-algorithme
$.0000000000000000 + 000$	$.0000000000000000 + 000$
$.5000000000000000 + 000$	$.5000000000000000 + 000$
$.5663110031972128 + 000$	$.7564499487951879 + 000$
$.5671431650348622 + 000$	$.5671537408683628 + 000$
$.5671432904097810 + 000$	$.5671431471513406 + 000$
$.5671432904097839 + 000$	$.5671432904097810 + 000$
$.5671432904097839 + 000$	$.5671432904098739 + 000$

La racine est évidemment $x = 0.5671432904098739...$; est une racine simple.

· Si x est une racine multiple de f:

La convergence de la méthode de **Newton** n'est plus quadratique (ordre 2) mais seulement linéaire (ordre 1). L'utilisation de l'ε-algorithme accélère donc la convergence, supprime ces oscillations et améliore très nettement la précision obtenue.

Soit par exemple:

$f(x) = x^6 + 6x^5 + 6x^4 - 18x^3 - 31x^2 - 24x - 36$

Ce polynôme possède $x = -3$ comme racine double, soit $x_0 = -3.5$, on obtient les

résultats suivants :

Méthode de **Newton**	ε-algorithme
$-.3500000000000000+001$	$-.3500000000000000+001$
$-.3314015312632922+001$	$-.3314015312632922+001$
$-.3186974034742264+001$	$-.2913160788588798+001$
$-.3105694680781266+001$	$-.2961331576850768+001$
$-.3057165552657127+001$	$-.3021542742532907+001$
$-.3029932985613312+001$	$-.3004720560384673+001$
$-.3015351668562098+001$	$-.2998019945539767+001$
$-.3007779399885537+001$	$-.2999797914487431+001$
$-.3003916605819156+001$	$-.3000052617516176+001$
$-.3001965163736187+001$	$-.3000002437371561+001$
$-.3000984314375852+001$	$-.2999999638429583+001$
$-.3000492592510335+001$	$-.2999999992351310+001$
$-.3000246405362310+001$	$-.3000000000619666+001$
$-.3000123229992691+001$	$-.3000000000006568+001$
$-.3000061621828629+001$	$-.2999999999999613+001$
$-.3000030812623127+001$	$-.3000000000001545+001$
$-.3000015406738995+001$	$-.3000000000000763+001$
$-.3000007703475447+001$	$-.3000000000001797+001$

Pour calculer toutes les racines d'un polynôme, on utilise la méthode de **Bairstow** (la plus utilisé), cette méthode consiste à déterminer la somme S et le produit P de deux racines du polynôme, c'est la méthode de **Newton** à double variables à convergence quadratique. Si les racines sont simples, elle est du premier ordre lorsque l'une des deux racines est multiple. On a comme exemple:

$x(x-8.01)^5$

Les deux premiers racines sont $x=0$ et $x=8.01$;

Leur somme S vaut 8.01 et leur produit P vaut 0 (nul), en partant de $S=P=0$, la méthode de **Bairstow** et de l'ε-algorithme nous donne pour les itérés qui convergent

vers $S = 8.01$.

n	Méthode de Bairstow	ε-algorithme
6	$.591022656000 + 001$	$.591022656000 + 001$
7	$.633018124800 + 001$	$.633018124800 + 001$
8	$.666614499840 + 001$	$.801000000000 + 001$
9	$.693491599872 + 001$	$.801000000000 + 001$
10	$.714993279898 + 001$	$.801000000000 + 001$
11	$.732194623918 + 001$	$.801000000000 + 001$

L'application de l'ε-algorithme est faite qu'à partir la sixième itéré de la méthode de **Bairstow** pour éviter les oscillations. Si les racines sont simples, alors l'ε-algorithme n'accélère pas la convergence.

Voyons maintenant comment utiliser les algorithmes d'accélération de la convergence pour construire de nouvelles méthodes afin de résoudre les équations.

Exemple 2.1.1 *L'utilisation du procédé* Δ^2***d'Aitken*** *pour obtenir la méthode de* ***Steffensen****:*

La méthode de ***Steffensen*** *consiste à l'itéré* n *à calculer :*

$$\begin{aligned} U_0 &= x_n \\ U_1 &= F(U_0) \\ U_2 &= F(U_1) \end{aligned}$$

Soit à calculer x *tel que* $x = F(x)$*;*

appliquons le procédé Δ^2***d'Aitken*** *à les trois valeurs précédentes puis à repartir de la quantité ainsi obtenue:*

$$x_{n+1} = \frac{U_0U_2 - U_1^2}{U_2 - 2U_1 + U_0}$$

Si x *racine simple (c.-à-d si* $F'(x) \neq 1$*), alors on démontre que la suite* $\{x_n\}$ *est d'ordre deux. La généralisation de cette méthode nous permet de résoudre les systèmes d'équations non linéaires sans calculer de dérivées.*

Remarque 2.1.1 *Pour des raisons de propagation des erreurs d'arrondis dans l'ordinateur,* x_{n+1} *sera calculé à l'aide de la relation suivante:*

$$x_{n+1} = U_1 + \frac{1}{\frac{1}{U_2-U_1} - \frac{1}{U_1-U_0}}$$

avec l'exemple: $x = e^{-x}$*, on obtient :*

$$\begin{aligned} x_0 &= 0 \\ x_1 &= 0,612 \\ x_2 &= 0,56735 \\ x_3 &= 0,5671432984 \\ x_4 &= 0,567143290409784 \end{aligned}$$

On voit que la convergence est quadratique c.-à-d que l'on double environ le nombre de chiffres exacts à chaque itération.

*Le procédé d'***Overholt** *peut construire une méthode itérative, si* F *est suffisamment dérivable au voisinage de* x*, cette méthode est d'ordre arbitraire* k *de la façon suivante:*

$$\begin{aligned} &x_0 \quad \textit{donné} \\ &(n+1)^{\grave{e}me} \textit{ itération:} \\ &U_0 = x_n \\ &U_1 = F(U_0) \\ &\qquad \vdots \\ &U_k = F(U_{k-1}) \end{aligned}$$

*En appliquant le procédé d'***Overholt** *à:*

$$\begin{aligned} V_0^{(0)} &= U_0 \\ &\vdots \\ V_0^{(k)} &= U_k \end{aligned}$$

et on prend:

$$x_{n+1} = V_{k-1}^{(0)}$$

La suite $\{x_n\}$ *ainsi obtenue est d'ordre* k

Exemple 2.1.2 *Pour les méthodes d'extrapolations polynomiales et rationnelles généralisées.*

Ces méthodes fournissent de même des méthodes de résolution de $x = F(x)$, nous introduisons une fonction ϕ supposant que cette fonction est strictement croissante (nous obtenons les mêmes résultats si ϕ est strictement décroissante).

On obtient plusieurs possibilités d'utilisations des méthodes d'extrapolation polynômiales et rationnelles généralisées, a partir de la supposition de la fonction f tel que:

$$f(x) = x - F(x)$$

Si elle comporte comme un polynôme ou comme une fonction rationnelle.

1/*Extrapolation linéaire des itérations de base:*

Les itérations de base sont les itérations $U_{n+1} = F(U_n)$

$$\begin{aligned} T_1^{(n)} &= \frac{f(U_n)U_{n+1} - f(U_{n+1})U_n}{f(U_n) - f(U_{n+1})} \\ &= \frac{U_nU_{n+2} - U_{n+1}^2}{U_{n+2} - 2U_{n+1} - U_n} \end{aligned}$$

C'est le procédé Δ^2 ***d'Aitken*** *appliqué à la suite $\{U_n\}$*

2/*Reprenons la méthode précédente et posons $U_{n+2} = T_1^{(n)}$, on trouve alors la méthode* ***Regula Falsi*** *qui est d'ordre $1,62$.*

Ces méthodes nous permettent d'obtenir de nouveaux algorithmes en utilisant l'extrapolation rationnelle généralisées au lieu de l'extrapolation polynômiale.

Si on prend comme exemple: $x = e^{-x}$ et $U_0 = 0$ avec la méthode (1), on trouve:

Extrapolation polynômiale	*Extrapolation rationnelle*
$0,612699836780282039$	$1,00000000000000000$
$0,567598911354531636$	$0,565828727712364331$
$0,567201829372173711$	$0,567441606777643165$
$0,567144303059276191$	$0,567142450175753599$
$0,567143299954168291$	$0,567143344665007396$
$0,567143290565629235$	$0,567143290387732216$

Exemple 2.1.3 *Soit à résoudre $f(x) = 0$. Posons $y = f(x)$ alors $x = f^{-1}(y)$.*

La résolution de $f(x) = 0$ *revient a rechercher* $\lim_{y\to 0} f^{-1}(y)$ *ou* $\lim_{t\to\infty} f^{-1}(1/t)$.

Posons $g(t) = f^{-1}(1/t)$, *appliquons la forme confluente du* ρ*-algorithme à* $\rho_0(t) = g(t)$.*Alors on trouve que:*

$$\rho_2(t) = x - 2\frac{f(x)f'(x)}{2f'^2(x) - f(x)f''(x)}$$

d'où l'idée de prendre la méthode itérative suivante:

$$x_{n+1} = x_n - 2\frac{f(x_n)f'(x_n)}{2f'^2(x) - f(x)f''(x)} \qquad n = 0, 1, ...$$

on trouve ainsi la méthode connue: méthode de ***Schröder*** *qui est d'ordre trois si* x *est racine simple de* f *et d'ordre un si* x *est racine multiple.*

2.2 Systèmes d'équations linéaires

On peut utiliser l'ε-algorithme vectoriel pour trouver la solution exacte d'un système d'équation lineaire comme il le montre le théorème suivant:

Théorème 2.2.1 *Si on applique l'*ε*-algorithme vectoriel à la suite* $\{x_n\}$ *produite par:*

$$\begin{aligned} & x_0 \quad \textit{donné} \\ x_{n+1} \;&=\; Ax_n + b \qquad n = 0, 1, ... \\ \textit{où}\;\; b \;&\in\; \mathbb{C}^p \quad \textit{et où } A \textit{ est une matrice carrée réelle et d'ordre } p \end{aligned}$$

et si la matrice $I - A$ *est inversible alors:*

$$\varepsilon_{2(m-r)}^{(n+r)} = (I - A)^{-1}b$$

où m *est le degré du polynôme minimal de* A *pour le vecteur* $x_0 - (I - A)^{-1}b$ *et où* r *est la multiplicité éventuelle de la racine zéro pour ce polynôme minimal (avec* $r = 0$ *si ce polynôme minimal ne possède pas de racine nulle).*

D'après ce théorème, on voit donc que l'ε-algorithme est une méthode dirécte de résolution des systèmes d'équations linéaires $(I - A)x = b$.

Il suffit pour résoudre les systèmes linéaires effectuer les itérations:

$$\begin{aligned} x_0 & \quad \text{donné} \\ x_{n+1} &= Ax_n + b \quad n = 0, 1, .. \end{aligned}$$

Où les x_n et b sont des vecteurs de $\mathbb{R}^p$ et où A est une matrice carrée réelle d'ordre p. On applique l'ε-algorithme vectoriel à la suite $\{x_n\}$ et si la matrice $I - A$ est inversible, alors:

$$\varepsilon_{2(m-r)}^{(n+r)} = (I - A)^{-1} b \quad n = 0, 1, ...$$

Où m est le degré du polynôme minimal de A pour le vecteur $x_0 - (I - A)^{-1} b$ et où r est la multiplicité éventuelle de la racine zéro pour ce polynôme minimal.

2.2.1 Méthode de Gauss-saidel

Dans la méthode de Gauss-Seidel, publiée en 1874 par **LudwigSeidel** $(1821 - 1896)$, on choisit $M = D - E$ et $N = F$ ce qui conduit à considérer la relation de récurrence

$$x_{k+1} = (D - E)^{-1} F x_k + (D - E)^{-1} b$$

C'est une amélioration de la méthode de Jacobi dans laquelle les valeurs calculées sont utilisées au fur et à mesure du calcul et non à l'issue d'une itération comme dans la méthode de Jacobi. On améliore ainsi la vitesse de convergence. Considérons un système à trois équations

$$\begin{cases} x = (b_1 - a_{12}y - a_{13}z) / a_{11} \\ y = (b_2 - a_{21}x - a_{23}z) / a_{22} \\ z = (b_3 - a_{31}x - a_{32}y) / a_{33} \end{cases}$$

À la première itération, on calcule à partir du vecteur initial

$$x_0 = \left(x^{(0)}, y^{(0)}, z^{(0)}\right)$$

la valeur $x^{(1)}$

$$x^{(1)} = \left(b_1 - a_{12}y^{(0)} - a_{13}z^{(0)}\right) / a_{11}$$

Cette valeur est réintroduite immédiatement dans le calcul de la deuxième composante (ce qui différencie cette méthode de la méthode de Jacobi, car on utilise ici la valeur $x^{(1)}$ et non $x^{(0)}$)

$$y^{(1)} = \left(b_2 - a_{21}x^{(1)} - a_{23}z^{(0)}\right)/a_{22}$$

De même, on porte $x^{(1)}$ et $y^{(1)}$ dans le calcul de $z^{(1)}$

$$z^{(1)} = \left(b_3 - a_{31}x^{(1)} - a_{32}y^{(1)}\right)/a_{33}$$

À chaque itération, on effectue $(n-1)$ multiplications, n additions et une division. Pour stocker A et les vecteurs b, x_k et x_{k+1}, on utilise (n^2+2n) mémoires. Si A et b sont calculés, on emploie n mémoires. La méthode ne converge pas toujours. On démontre que si A est une matrice définie positive, la méthode itérative converge. De même, si A est une matrice diagonalement dominante, c'est-à-dire si:

$$|a_i| > \sum_{j \neq i} |a_{ij}|$$

alors la méthode de Gauss-Seidel converge.

Exemple 2.2.1 *Soient les vecteurs* $\{x_n\}$ *générés par:*

$$\begin{aligned} x_0 &= 0 \\ x_{n+1} - x &= A(x_n - x) \qquad n = 0, 1, .. \end{aligned}$$

avec

$$x = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \text{ et } A = \begin{pmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{pmatrix}$$

En utilisant l'ε-algorithme scalaire on obtient:

k	$\left\{\varepsilon_{2k}^{(0)}\right\}$	$\left\{\varepsilon_{2k}^{(1)}\right\}$
1	$-2,79$	$1,65$
	$-0,82$	$0,18$
	$0,17$	$0,97$
2	$1,01$	$0,99994$
	$2,06$	$2,0003$
	$2,86$	$2,9997$
3	$0,9999999999999998$	
	$1,9999999999999998$	
	$3,0000000000000000$	

2.3 Systèmes d'équations non linéaires

L'ε-algorithme a comme intérêt fournir une méthode d'ordre deux pour résoudre les systèmes d'équations non linéaires sans calculer aucun dérivée ou inverser une matrice, cette méthode a plusieurs utilisations puisque beaucoup de problèmes peuvent se ramener à la résolution d'un système d'équation non linéaires.

La méthode que nous présentons ouvre un grand nombre de possibilités nouvelles.

2.3.1 Présentation de la méthode

Cette méthode a été trouvé par **Gekeler** et **Brezinski**.

Considérons le système suivant de p équations non linéaire à p inconnues:

$$\begin{aligned} x^{(1)} &= F_1\left(x^{(1)}, ..., x^{(p)}\right) \\ x^{(2)} &= F_2\left(x^{(1)}, ..., x^{(p)}\right) \\ &..... \\ x^{(p)} &= F_p\left(x^{(1)}, ..., x^{(p)}\right) \end{aligned}$$

Nous désignerons par x le vecteur de $\mathbb{R}^p$ dont les composantes sont $x^{(1)}$, $x^{(2)}$, ..., $x^{(p)}$ et par F l'application de $\mathbb{R}^p$ dans $\mathbb{R}^p$ qui fait correspondre le vecteur $F(x)$ dont les composantes sont: $F_1\left(x^{(1)}, ..., x^{(p)}\right)$, $F_2\left(x^{(1)}, ..., x^{(p)}\right)$, ..., $F_2\left(x^{(1)}, ..., x^{(p)}\right)$.

Avec ces notations, le système d'équations précédent se symbolise par:

$$x = F(x)$$

Nous allons rechercher les solutions de ce système, c'est-à-dire les vecteurs $x \in \mathbb{R}^p$ qui vérifient $x = F(x)$.

Nous supposerons que les dérivées partielles $\frac{\partial F_1}{\partial x^{(1)}}, ..., \frac{\partial F_1}{\partial x^{(p)}}, \frac{\partial F_2}{\partial x^{(1)}}, ..., \frac{\partial F_2}{\partial x^{(p)}}, ..., \frac{\partial F_p}{\partial x^{(1)}}, ..., \frac{\partial F_p}{\partial x^{(p)}}$ existent au voisinage de x (F est différentiable au sens de **Fréchet** au voisinage de x).

Nous appellerons matrice jacobéenne de F et nous noterons $F'(x)$ la matrice définie par:

$$F'(x) = \begin{pmatrix} \frac{\partial F_1}{\partial x^{(1)}} & ... & ... & \frac{\partial F_1}{\partial x^{(p)}} \\ \frac{\partial F_2}{\partial x^{(1)}} & ... & ... & \frac{\partial F_2}{\partial x^{(p)}} \\ ... & ... & ... & ... \\ \frac{\partial F_p}{\partial x^{(1)}} & ... & ... & \frac{\partial F_p}{\partial x^{(p)}} \end{pmatrix}$$

Où les dérivées partielles sont calculées au point fixe x de F.

L'algorithme proposé pour calculé x est le suivant:

$$\begin{aligned} &x_0 \in \mathbb{R}^p \quad \text{donné} \\ &(n+1)^{ème} \quad \text{itération:} \\ &U_0 = x_n \in \mathbb{R}^p \\ &U_k = F(U_{k-1}) \quad k = 1, ..., 2m - r \end{aligned}$$

puis on applique l'ε-algorithme vectoriel aux vecteurs U_0, $U_1, ..., U_{2m-r}$, ce qui fournit le vecteur noté $\varepsilon^{(r)}_{2(m-r)}$. On continue ensuite les itérations en prenant:

$$x_{n+1} = \varepsilon^{(r)}_{2(m-r)}$$

m: le degré du polynôme minimal de $F'(x)$ pour le vecteur $x_n - x$.

r: la multiplicité éventuelle de la racine zéro pour ce polynôme minimal ($r = 0$ si $F'(x)$ est inversible).

Les itérations $U_k = F(U_k - 1)$ pour $k = 1, ..., 2m - r$ s'appellent les itérations de

base (U_{K-1} étant connu), on calcule le vecteur U_k à l'aide des itérations:

$$\begin{aligned} U_k^{(1)} &= F_1(U_{k-1}^{(1)}, ..., U_{k-1}^{(p)}) \\ U_k^{(2)} &= F_2(U_{k-1}^{(1)}, ..., U_{k-1}^{(p)}) \\ & ..... \\ U_k^{(p)} &= F_p(U_{k-1}^{(1)}, ..., U_{k-1}^{(p)}) \end{aligned}$$

$U_k^{(i)}$:désigne la i^{ème} composante du vecteur $U_k \in \mathbb{R}^p$.

Le théorème fondamental concernant cette méthode est :

Théorème 2.3.1 *Si x est un point fixe de F, si F est différentiable au sens de **Fréchet** au voisinage de x et si la matrice $I - F'(x)$ est inversible, alors il existe un voisinage V de x tel que pour tout $x_0 \in V$ la méthode précédente converge vers x et que la suite $\{x_n\}$ soit d'ordre deux au moins c'est-à-dire que:*

$$\|x_{n+1} - x_n\| = 0(\|x_n - x\|^2)$$

Donnons quelques remarques fondamentales ont relation avec le théorème précédent:

Remarque 2.3.1 ***1/****si $p = 1$ l'algorithme se réduit à la méthode de **Steffensen**.*

2/*On peut utiliser l'ε-algorithme topologique ou l'ε-algorithme scalaire au lieu de l'ε-algorithme vectoriel en traitant séparément chacune des composantes des vecteurs U_k.*

3/*Si le système à résoudre s'écrit sous la forme $f(x) = 0$, alors pour appliquer* (2.3.1) *on le mettre sous la forme*

$$x = x + \alpha f(x)$$

α:nombre réel non nul ou matrice carrée inversible, on passe à $x = F(x)$ en posant $F(x) = x + \alpha f(x)$.

Exemple 2.3.1 *Considérons un système à deux équations:*

Où les inconnues sont x et y. La solution unique est $x = -1$ et $y = 1$.

Si on applique l'ε-algorithme vectoriel, on obtient:

n	$\{x_n\}$	$\{y_n\}$
0	0	0
1	$-0,85$	$0,87$
2	$-0,96$	$0,97$
3	$-0,9979$	$0,9978$
4	$-0,999989$	$0,999984$
5	$-0,9999999997$	$0,9999999991$

Si on applique l'ε-algorithme scalaire, on obtient:

n	$\{x_n\}$	$\{y_n\}$
0	0	0
1	$-0,85$	$0,87$
2	$-0,97$	$0,97$
3	$-0,9980$	$0,9978$
4	$-0,9999905$	$0,999984$
5	$-0,9999999998$	$0,9999999991$

Dans cet exemple, on avait $m = p = 2$ et $r = 0$.

Avec la méthode de ***Newton****, le calcul nécessite* 24 *évaluations de F:*

n	$\{x_n\}$	$\{y_n\}$
0	0	0
1	$-0,750000000000000000$	$0,304095859474086683$
2	$-0,765876880623479763$	$0,792662514760822501$
3	$-0,921281538821868353$	$0,938407511361889756$
4	$-0,987935311564206382$	$0,991734013418315369$
5	$-0,999776520512802722$	$0,999874805260563668$
6	$0,99999982677895456$	$1,000000006186878680$

Si l'on avait laissé se continuer les itérations de base, on aurait convergé vers la solution

mais lentement:

n	$\{x_n\}$	$\{y_n\}$
20	$-0,981427814583948488$	$0,984764782856446254$
100	$-0,999962123789110 92$	$0,999969219747354 28$
200	$-0,999999998993969 17$	$0,999999999182446 65$
300	$-0,999999999999973 25$	$0,999999999999978 26$

2.3.2 Stabilité numérique

Nous allons étudier la stabilité numérique de l'algorithme (2.3.1). L'instabilité numérique des itérations de base entraîne l'instabilité numérique de l'algorithme (2.3.1) dans son ensemble, ce que nous étudierons ici.

Considérons les itérations de base de l'algorithme (2.3.1), on peut l'écrire sous la forme:

$$U_k = U_{k-1} + h\left(F\left(U_{k-1}\right) - U_{k-1}\right) \tag{2.3.1}$$

avec $h = 1$.

A cause des erreurs d'arrondis dans l'ordinateur qui résultent des itérations antérieurs, on ne connaît pas la valeur exacte de U_{k-1}, mais $\overline{U}_{k-1} = U_{k-1} + e_{k-1}$, d'où:

$$U_k + e_k = \overline{U}_k = U_{k-1} + h\left(F\left(\overline{U}_{k-1}\right) - \overline{U}_{k-1}\right)$$

Comme F est différentiable dans un voisinage de point fixe x, on a donc:

$$e_k = \left[I + h\left(F'\left(U_{k-1}\right) - I\right)\right] e_{k-1} + o\left(e_{k-1}\right)$$

$o\left(e_k\right)$: vecteur où la norme tend vers zéro plus rapidement que la norme du $o\left(e_{k-1}\right)$.

Les itérations de base seront numériquement stables si la norme du vecteur d'erreur e_k n'augmente pas avec k.

L'algorithme (2.3.1) a de nombreuses applications, citons:

1/Schémas implicites de **Runge-Kutta.**

2/Les équations différentielles non résolus.

3/Les problèmes aux limites.

4/Les Problèmes de frontière libre.

2.4 Equations matricielles

L'algorithme (2.3.1) que nous avons utilisé pour résoudre les systèmes d'équations non linéaires peut être généralisé pour résoudre les équations matricielles.

Soit X une matrice et F une fonction non linéaire de X. On veut trouver la matrice X tel que: $X = F(X)$.

En utilisant l'ε-algorithme matriciel, dans le cas d'une seule équation matricielle, on a donc une généralisation du procédé de **Steffensen**:

$$(n+1)^{ème}\ itération : \begin{cases} X_0 \text{ donné} \\ u_0 = X_n \\ u_1 = F(u_0) \\ u_2 = F(u_1) \\ X_{n+1} = u_1 - (u_1 - u_0)(u_2 - 2u_1 + u_0)^{-1}(u_2 - u_1) \end{cases}$$

On vérifie facilement que si $F(X) = AX + B$ ou si $F(X) = XA + B$ où A et B sont des matrices carrées, alors X_1 est égal à la solution du système, quelle que soit la matrice de départ X_0.

Soit par exemple à résoudre à l'aide de cette méthode l'équation matricielle:

$$XA + BX = C$$

avec:

$$A = \begin{pmatrix} 1 & 3 \\ 0 & 2 \end{pmatrix} B = \begin{pmatrix} 4 & 0 \\ 1 & 3 \end{pmatrix} C = \begin{pmatrix} 5 & 15 \\ 13 & 31 \end{pmatrix}$$

la solution exacte est:

$$X = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

En partant de $X_0 = 0$ on obtient:

$$X_1 = \begin{pmatrix} 1.31 & 2.37 \\ 3.84 & 5.31 \end{pmatrix},\ X_2 = \begin{pmatrix} 1.05 & 2.10 \\ 3.16 & 4.21 \end{pmatrix},\ X_5 = \begin{pmatrix} 1.0009 & 2.001 \\ 3.002 & 4.004 \end{pmatrix},\ X_{10} = \begin{pmatrix} 1.00000004 & 2.0000008 \\ 3.000001 & 4.000002 \end{pmatrix}$$

$$X_{20} = \begin{pmatrix} 0.9999999999998 & 1.9999999999997 \\ 2.9999999999995 & 3.9999999999993 \end{pmatrix}$$

La convergence de cet algorithme ne semble pas être quadratique sur cet exemple et aucun résultat théorique n'a encore été démontré pour cette méthode.

2.5 Calcul des valeurs propres

Soit A une matrice carrée réelle d'ordre p. Une méthode très connue pour calculer la valeur propre de plus grand module de A est la méthode de la puissance quelque fois appelée méthode du quotient de Rayleigh; C'est une méthode itérative. Une fois obtenue cette première valeur propre, on peut modifier la méthode ($\lambda -$ différence) ou la matrice (déflation)pour converge vers la valeur propre du module immédiatement inférieur. L'inconvénient de cette façon de procéder est évident: il faut attendre que la méthode ait convergé vers une valeur propre avant de pouvoir commencer les itérations pour calculer la valeur propre suivante.

Dans ce paragraphe, nous allons donner deux généralisations de la méthode de la puissance à l'aide de l'ε-algorithme. Ces généralisations permettent, sous certaines hypothèses, le calcul simultané de toutes les valeurs propres et de tous les vecteurs propres de la matrice A.

Nous montrerons également comment il est possible d'accélérer la convergence de ces méthodes toujours avec l'ε-algorithme.

Nous donnerons aussi un exemple d'accélération de la convergence de différentes méthodes de calcul des valeurs propres.

Rappelons d'abord ce qu'est la méthode de la puissance:

soit A une matrice carrée réelle d'ordre p et λ_1 , $\lambda_2, ..., \lambda_p$ les valeurs propre de A et $v_1, v_2, ..., v_p$ les vecteurs propres correspondants, on a:

$$Av_i = \lambda_i v_i \qquad i = 1, 2, ..., p$$

nous supposerons que :

$$\mid \lambda_1 \mid \succ \mid \lambda_2 \mid \geq \mid \lambda_3 \mid \geq ... \geq \mid \lambda_p \mid$$

et que $\lambda_1 \neq 1$.

Soit x_0 un vecteur arbitraire de $\mathbb{R}^p$ tel que $(x_0, v_1) \neq 0$ et soit y un vecteur arbitraire de $\mathbb{R}^p$ qui vérifie également la même hypothèse.

La méthode de la puissance consiste à générer la suite de vecteurs $\{x_n\}$ par:

$$x_{n+1} = Ax_n \qquad n = 0, 1, 2, ...$$

On a alors:

$$\begin{aligned} \lim_{n\to\infty} \frac{(y, x_{n+1})}{(y, x_n)} &= \lambda_1 \\ \lim_{n\to\infty} \frac{x_n}{(y, x_n)} &= v_1 \end{aligned}$$

Nous allons maintenant donner deux généralisations de cette méthode

Premier algorithme Nous supposerons maintenant que :

$$| \lambda_1 | \succ | \lambda_2 | \succ | \lambda_3 | \succ ... \succ | \lambda_p |$$

et que $\lambda_i \neq 1$ pour $i = 1, 2, ..., p$.

La première hypothèse est assez restrictive car elle empêche évidement que la matrice A ait des valeurs propre complexe ou multiples mais elle empêche également que $\lambda_i = -\lambda_{i+1}$.

Cependant des modifications appropriées de l'algorithme, qui sont actuellement à l'étude, devraient permettre de traiter de tels cas. La seconde hypothèse n'est absolument pas restrictive: si l'une des valeurs propres de A est égale à un, alors il suffit de multiplier la matrice A par un nombre égale à un alors il suffit de multiplier la matrice A par un nombre réel non nul a et les valeurs propres seront également multipliées par a.

Soit $x_0 \in \mathbb{R}^p$, vecteur arbitraire tel que: $(x_0, v_i) \neq 0 \quad i = 1, 2, ..., p$ et $\{x_n\}$ suite de vecteurs, on a : $x_{n+1} = Ax_n$, $n = 0, 1, ...$

On applique l'ε-algorithme vectoriel à cette suite de vecteurs, on construit ainsi un ensemble de vecteurs de $\mathbb{R}^p$ notés $\varepsilon_k^{(n)}$.

Soit $y \in \mathbb{R}^p$, vecteur arbitraire tel que : $(y, v_i) \neq 0$ pour $i = 1, 2, ..., p$.

Considérons les rapports:

$$a_k^{(n)} = \frac{\left(y, \varepsilon_{2k}^{(n+1)}\right)}{\left(y, \varepsilon_{2k}^{(n)}\right)} \qquad k = 0, 1, ..., p-1 \text{ et } n = 0, 1, ...$$

$$b_k^{(n)} = \frac{\left(y, \varepsilon_{2k+1}^{(n)}\right)}{\left(y, \varepsilon_{2k+1}^{(n+1)}\right)} \qquad k = 0, 1, ..., p-1 \text{ et } n = 0, 1, ...$$

naturellement la suite $\left\{a_0^{(n)}\right\}$ est celle que nous avons considérée dans la méthode de la puissance. On peut démontrer le:

Théorème 2.5.1 $\lim a_k^{(n)} = \lim b_k^{(n)} = \lambda_{k+1} \qquad k = 0, ..., p-1$

$\lim \frac{\varepsilon_{2k}^{(n)}}{\left(y, \varepsilon_{2k}^{(n)}\right)} = \lim \left(y, \varepsilon_{2k+1}^{(n)}\right) \varepsilon_{2k}^{(n)} = v_{k+1}$ pour $k = 0, 1, ..., p-1$.

De plus

$a_k^{(n)} = \lambda_{k+1} + 0\left[\left(\frac{\lambda_{k+2}}{\lambda_{k+1}}\right)^{n+k+1}\right] \qquad k = 0, 1, ..., p-1$

Donnons un exemple numérique concernant cette méthode, considérons d'abord la matrice:

$$A = \begin{pmatrix} 1.1 & 0 & 0 \\ 1 & -0.2 & 0 \\ 1 & 1 & 0.9 \end{pmatrix}$$

dont les valeurs propres sont évidement $\lambda_1 = 1.1$, $\lambda_2 = 0.9$, $\lambda_3 = -0.2$.

En pernant pour x_0 et y des vecteurs dont toutes les composante sont égales à un, l'algorithme précédent nous donne:

n	$\left\{a_0^{(n)}\right\}$	$\left\{b_0^{(n)}\right\}$	$\left\{a_1^{(n)}\right\}$	$\left\{b_1^{(n)}\right\}$	$\left\{a_2^{(n)}\right\}$	$\left\{b_2^{(n)}\right\}$
5	1.17	1.047	0.85	0.95	−0.044	1.14
8	1.13	1.067	0.87	0.93	−0.18	0.46
11	1.12	1.080	0.887	0.917	−0.19993	−0.196
14	1.09	1.089	0.893	0.909	−0.1987	−0.202

pour λ_3 il est très net que l'on passe par la bonne valeur puis que la propagation des erreurs numériques vient détruire la convergence.

On peut d'ailleurs s'en assurer en passant le même programme en simple précision et en double précision. Les résultats numériques obtenus peuvent être complètement différents.

Deuxième algorithme Au lien d'utiliser l'ε-algorithme vectoriel, on peut utiliser l'application itérée de la forme vectorielle du procédé Δ^2**d'Aitken**.

Soit $\{x_n\}$ suite de vecteurs telle que: $x_{n+1} = Ax_n$ sous les mêmes hypothèses, appliquons lui le procédé Δ^2 **d'aitken** itéré, on pose: ${}_0\varepsilon_0^{(n)} = x_n$, $n = 0, 1, ...$

puis, pour $k = 0, 1, ..., p-2$, on fait:

$$\begin{aligned} {}_k\varepsilon_1^{(n)} &= \left[\Delta_k \varepsilon_0^{(n)}\right]^{-1} && n = 0, 1, ... \\ {}_k\varepsilon_2^{(n)} &= {}_k\varepsilon_0^{(n+1)} + \left[\Delta_k \varepsilon_1^{(n)}\right]^{-1} && n = 0, 1, ... \\ {}_{k+1}\varepsilon_0^{(n)} &= {}_k\varepsilon_2^{(n)} \quad n = 0, 1, ... \end{aligned}$$

Considérons les rapports:

$$\begin{aligned} c_k^{(n)} &= \frac{\left(y_{,k}\, \varepsilon_0^{(n)}\right)}{\left(y_{,k}\, \varepsilon_0^{(n)}\right)} && k = 0, 1, ..., p-1 \quad \text{et} \quad n = 0, 1, ... \\ d_k^{(n)} &= \frac{\left(y_{,k}\, \varepsilon_1^{(n)}\right)}{\left(y_{,k}\, \varepsilon_1^{(n+1)}\right)} && k = 0, 1, ..., p-1 \quad \text{et} \quad n = 0, 1, ... \end{aligned}$$

pour $k = 0$, la suite $\left\{c_0^{(n)}\right\}$ est exactement celle généreé par la méthode de la puissance, on a le

Théorème 2.5.2

$$\begin{aligned} \lim_{n\to\infty} c_k^{(n)} &= \lim_{n\to\infty} d_k^{(n)} = \lambda_{k+1} && k = 0, ...p-1 \\ \lim_{n\to\infty} \frac{{}_k\varepsilon_0^{(n)}}{\left(y_{,k}\, \varepsilon_0^{(n)}\right)} &= \lim_{n\to\infty} \left(y_{,k}\, \varepsilon_1^{(n)}\right)_k \varepsilon_0^{(n)} = v_{k+1} && \textit{pour} \quad k = 0, 1, ..., p-1 \\ & \textit{De plus} \\ c_k^{(n)} &= \lambda_{k+1} + 0\left[\left(\frac{\lambda_{k+2}}{\lambda_{k+1}}\right)^{n+k+1}\right] && k = 0, 1, ..., p-1 \end{aligned}$$

On obtient donc un résultat tout à fait identique à celui du théorème 2.5.1.

Il est évident que ces deux méthodes peuvent être utilisées pour ne calculer que quelques valeurs propres de plus grand module puis que l'on obtient celles-ci par ordre de modules décroissants.

2.6 Variantes des méthodes

Au lieu de l'ε-algorithme vectoriel, on peut utiliser dans la méthode du premier algorithme l'ε-algorithme topologique ou encore l'ε-algorithme vectoriel normé. On peut également utiliser cette même variante dans la seconde méthode de deuxième algorithme précédent les théorèmes 2.5.1 et 2.5.2 sont encore valables.

Si l'on prend par exemple pour tout vecteur $y \in \mathbb{R}^p$:

$$\| y \| = \sum_{i=1}^{p} | y_i |$$

alors, pour l'exemple du premier algorithme, on obtient:

n	$\left\{a_0^{(n)}\right\}$	$\left\{b_0^{(n)}\right\}$	$\left\{a_1^{(n)}\right\}$	$\left\{b_1^{(n)}\right\}$	$\left\{a_2^{(n)}\right\}$	$\left\{b_2^{(n)}\right\}$
5	1.17	1.058	0.84	0.95	0.82	0.48
8	1.13	1.075	0.87	0.91	0.79	0.88
11	1.12	1.085	0.886	0.8998	0.77	0.81
14	1.109	1.092	0.892	0.8977	0.76	0.78

Pour les deux premières valeurs propres, on voit que les résultats sont tout à fait comparables à ceux obtenus avec l'ε-algorithme vectoriel.

Pour λ_3, la convergence des suites $\left\{a_2^{(n)}\right\}$et $\left\{b_2^{(n)}\right\}$ n'a pas encore commencé à se manifester.

Si l'on prend maintenant comme norme dans l'ε-algorithme vectoriel normé:

$$\| y \| = \max_{1 \le i \le p} | y_i |$$

alors , toujours sur le même exemple ,on obtient:

n	$\left\{a_0^{(n)}\right\}$	$\left\{b_0^{(n)}\right\}$	$\left\{a_1^{(n)}\right\}$	$\left\{b_1^{(n)}\right\}$	$\left\{a_2^{(n)}\right\}$	$\left\{b_2^{(n)}\right\}$
5	1.17	1.046	0.85	0.95	-0.08	1.38
8	1.13	1.066	0.87	0.93	-0.202	-0.161
11	1.12	1.080	0.887	0.917	-0.199966	-0.2004
14	1.109	1.088	0.893	0.9097	-0.1986	-0.2025

on voit que l'on trouver bien les trois valeurs propres mais que la précision obtenue sur λ_3 se dégarde rapidement.

Il existe également une variante des méthodes précédentes qui diminue considérablement le volume remédier partiellement à cet inconvénient, effectuer les calculs sur deux composantes des vecteurs x_n au lieu d'une seule.

On pourra évidement utiliser également l'ε-algorithme scalaire séparément sur chacune des composantes des vecteurs x_n.

On peut aussi générer les vecteurs $\{x_n\}$ comme précédement et appliquer l'ε-algorithme scalaire à la suite $\{S_n = (y, x_n)\}$

où y est un vecteur arbitraire tel que $(y, v_i) \neq 0$ pour $i = 1, ..., p$.

On aura alors:

$$\lim_{n\to\infty} \frac{\varepsilon_{2k}^{(n+1)}}{\varepsilon_{2k}^{(n)}} = \lambda_{k+1} = \lim_{n\to\infty} \frac{\varepsilon_{2k+1}^{(n)}}{\varepsilon_{2k+1}^{(n+1)}} \qquad k = 0, ..., p-1$$

si l'on avait utilisé l'application itérée du procédé Δ^2 **d'Aitken** pour transformer la suite $\{S_n\}$ précédente, on aurait obtenu de même:

$$\lim_{n\to\infty\infty} \frac{{}_k\varepsilon_0^{(n+1)}}{{}_k\varepsilon_0^{(n)}} = \lambda_{k+1} = \lim_{n\to\infty} \frac{{}_k\varepsilon_1^{(n)}}{{}_k\varepsilon_1^{(n+1)}} \qquad k = 0, ..., p-1$$

cette variante des algorithmes ne permet naturellement pas de calculer les vecteurs propres de la matrice.

Une fois que les valeurs propres ont été calculées, ces méthodes permettent toutes de calculer les coefficients du polynôme caractéristique de la matrice.

2.7 Accélération de la convergence

Il est bien connu que le procédé Δ^2**d'aitken** accélére la convergence de la méthode de la puissance. Ainsi si on l'applique à la suite:

$$a_0^{(n)} = \lambda_1 + 0\left[\left(\frac{\lambda_2}{\lambda_1}\right)^{n+1}\right]$$

on obtiendra:

$$\varepsilon_2^{(n)} = \lambda_1 + 0\left[\left(\frac{\lambda_3}{\lambda_1}\right)^{n+1}\right]$$

on peut donc appliquer l'ε-algorithme scalaire pour accélérer la convergence des suites $\left\{a_k^{(n)}\right\}$et $\left\{c_k^{(n)}\right\}$ pour k fixé.

En utilisant le théorème suivant.

Si on applique l'ε-algorithme scalaire aux suites $\left\{a_k^{(n)}\right\}$ ou $\left\{c_k^{(n)}\right\}$ pour k fixé $(k = 0, 1, ..., p-1)$ alors il produit des nombres $\varepsilon_{2m}^{(n)}$ tel que:

$$\varepsilon_{2m}^{(n)} = \lambda_{k+1} + 0\left[\left(\frac{\lambda_{k+m+2}}{\lambda_{k+1}}\right)^{n+k}\right] m \quad \text{et} \quad k \quad \text{fixés.}$$

On voit donc que l'utilisation de l'ε-algorithme scalaire accélére la convergence des suites $\left\{a_k^{(n)}\right\}$et $\left\{c_k^{(n)}\right\}$vers λ_{k+1}.

Ainsi:

$$\lim_{n\to\infty} \frac{a_k^{(n+1)} - \lambda_{k+1}}{a_k^{(n)} - \lambda_{k+1}} = \frac{c_k^{(n+1)} - \lambda_{k+1}}{c_k^{(n)} - \lambda_{k+1}} = \frac{\lambda_{k+1}}{\lambda_{k+1}}$$

et:

$$\lim_{n\to\infty} \frac{\varepsilon_{2m}^{(n+1)} - \lambda_{k+1}}{\varepsilon_{2m}^{(n)} - \lambda_{k+1}} = \frac{\lambda_{k+m+2}}{\lambda_{k+1}}$$

et par conséquent:

$$\lim \frac{\varepsilon_{2m+2}^{(n)} - \lambda_{k+1}}{\varepsilon_{2m}^{(n+2)} - \lambda_{k+1}} = 0$$

quand l'ε-algorithme scalaire est appliqué à $\varepsilon_0^{(n)} = a_k^{(n)}$ ou à $\varepsilon_0^{(n)} = c_k^{(n)}$.

Les calculs à effectuer mais ne fournit pas les vecteurs propres de la matrice. Au lieu d'appliquer l'ε-algorithme vectoriel ou le procédé Δ^2 **d'aitken** itéré aux vecteurs $\{x_n\}$ celle variante consiste à ne conserver que certaines composantes des vecteurs x_n (toujours les mêmes lorsque n vraie cependant) . Si les vecteurs réduits x_0 et y vérifient toujours $(x_0, v_i) \neq 0$ et $(y, v_i) \neq 0$ pour $i = 1, ..., p$ où v_i est le vecteur formé des composantes correspondantes du vecteurs propre v_i de la matrice A. On pourra, par exemple ne conserver que la première composante $(x_n)_1$ de l'ε-algorithme scalaire à la suite $\varepsilon_0^{(n)} = (x_n)_1$ et sous réserve que $(x_0)_1 . (v_i)_1 \neq 0$ pour $i = 1, ...p$ et qu'il n'y pas de division par zéro dans l'application des algorithmes, on aura:

$$\lim_{n\to\infty} \frac{\varepsilon_{2k}^{(n+1)}}{\varepsilon_{2k}^{(n)}} = \lim_{n\to\infty} \frac{\varepsilon_{2k+1}^{(n)}}{\varepsilon_{2k+1}^{(n+1)}} = \lambda_{k+1} \qquad k = 0, 1, ..., p-1$$

et:

$$\lim_{n\to\infty} \frac{{}_k\varepsilon_0^{(n+1)}}{{}_k\varepsilon_0^{(n)}} = \lim_{n\to\infty} \frac{{}_k\varepsilon_1^{(n)}}{{}_k\varepsilon_1^{(n+1)}} = \lambda_{k+1} \qquad k = 0, 1, ..., p-1$$

Cette variante réduit considérablement le volume des calculs ainsi que le nombre de mémoires nécessaires à la programmation des méthodes.

En effet pour programmer les méthodes complètes qui ont été définies aux premier et deuxième algorithme, il faut $2p^2 + p$ mémoires seul composante des vecteurs x_n il n'en faut que $2p + 1$.

D'autre part, l'ε-algorithme scalaire est numériquement plus stable que l'ε-algorithme vectoriel. Le seul inconvénient de cette variante est qu'elle ne permet pas de calculer les vecteurs propres et qu'il peut y avoir plus facilement une division par zéro dans l'application de l'ε-algorithme scalaire sur une composante que dans l'application de l'ε-algorithme vectoriel sur l'ensemble des vecteurs $\{x_n\}$.

Chapitre 3

Quelques exemples numériques et comparaison des méthodes d'accélération de la convergence

Ce chapitre présente une comparaison entre quelques méthodes d'accélération de la convergence à l'aide de quelques exemples numériques, en chaque exemple nous obtenons la meilleure méthode de convergence qui nous permet d'avoir une précision, un nombre d'itération et une vitesse de convergence beaucoup mieux.

3.0.1 Comparaison entre l'ε-algorithme et le θ-algorithme et le ρ-algorithme

Exemple 3.0.1 *Soit $\{S_n\}$ une suite de nombre réel, tel que:*

$$S_n = 1 + 3e^{-1.4x_n} \qquad avec \;\; x_n = (1,1)^{n-1}$$

En accélére la convergence de cette suite en utilisant l'ε-algorithme scalaire:

calculons la valeur de S_n pour $n = 0, 1, ..., 5$

$$S_0 = 1 + 3e^{-1,4x_0} = 1,84020 \quad x_0 = (1,1)^{-1} = 0.90909$$
$$S_1 = 1 + 3e^{-1,4x_1} = 1,73979 \quad x_1 = (1,1)^{0} = 1$$
$$S_2 = 1 + 3e^{-1,4x_2} = 1,64314 \quad x_2 = (1,1)^{1} = 1,1$$
$$S_3 = 1 + 3e^{-1,4x_3} = 1,55134 \quad x_3 = (1,1)^{2} = 1,21$$
$$S_4 = 1 + 3e^{-1,4x_4} = 1,46543 \quad x_4 = (1,1)^{3} = 1,331$$
$$S_5 = 1 + 3e^{-1,4x_5} = 1,38630 \quad x_5 = (1,1)^{4} = 1,4641$$

Présentons les calculs sur le schéma suivant:

$$\begin{array}{lllll}
\varepsilon_{-1}^{(0)} = 0 & & & & \\
& \varepsilon_{0}^{(0)} = S_0 = 1,84020 & & & \\
\varepsilon_{-1}^{(1)} = 0 & & \varepsilon_{1}^{(0)} = -9,95916 & & \\
& \varepsilon_{0}^{(1)} = S_1 = 1,73979 & & \varepsilon_{2}^{(0)} = -0,84118 & \\
\varepsilon_{-1}^{(2)} = 0 & & \varepsilon_{1}^{(1)} = -10,34661 & & \varepsilon_{3}^{(0)} = -8,81972 \\
& \varepsilon_{0}^{(2)} = S_2 = 1,64314 & & \varepsilon_{2}^{(1)} = -0,18625 & \varepsilon_{4}^{(0)} = 2,11154 \\
\varepsilon_{-1}^{(3)} = 0 & & \varepsilon_{1}^{(2)} = -10,89324 & & \varepsilon_{3}^{(1)} = -8,38452 \\
& \varepsilon_{0}^{(3)} = S_3 = 1,55134 & & \varepsilon_{2}^{(2)} = 0,21236 & \\
\varepsilon_{-1}^{(4)} = 0 & & \varepsilon_{1}^{(3)} = -11,64008 & & \\
& \varepsilon_{0}^{(4)} = S_3 = 1,46543 & & &
\end{array}$$

Si on applique le ρ-algorithme scalaire pour accélérer la suite S_n, on obtient les résultats suivants:

$$\begin{array}{lllll}
\rho_{-1}^{(0)} = 0 & & & & \\
& \rho_{0}^{(0)} = S_0 = 1,84020 & & & \\
\rho_{-1}^{(1)} = 0 & & \rho_{1}^{(0)} = -0,90528 & & \\
& \rho_{0}^{(1)} = S_1 = 1,73979 & & \rho_{2}^{(0)} = 0,26422 & \\
\rho_{-1}^{(2)} = 0 & & \rho_{1}^{(1)} = -1,03466 & & \rho_{3}^{(0)} = 2,12516 \\
& \rho_{0}^{(2)} = S_2 = 1,64314 & & \rho_{2}^{(1)} = 0,35495 & \rho_{4}^{(0)} = 6,36227 \\
\rho_{-1}^{(3)} = 0 & & \rho_{1}^{(2)} = -1,19825 & & \rho_{3}^{(1)} = 2,36318 \\
& \rho_{0}^{(3)} = S_3 = 1,55134 & & \rho_{2}^{(2)} = 0,40323 & \\
\rho_{-1}^{(4)} = 0 & & \rho_{1}^{(3)} = -1,40845 & & \\
& \rho_{0}^{(4)} = S_3 = 1,46543 & & &
\end{array}$$

De même, on applique le θ-algorithme on obtient les résultats suivants:

$$
\begin{array}{ccccccccc}
\theta_{-1}^{(0)}=0 & & & & & & & & \\
 & \theta_{0}^{(0)}=1,84020 & & & & & & & \\
\theta_{-1}^{(1)}=0 & & \theta_{1}^{(0)}=-9,95916 & & & & & & \\
 & \theta_{0}^{(1)}=173979 & & \theta_{2}^{(0)}=1,40788 & & & & & \\
\theta_{-1}^{(2)}=0 & & \theta_{1}^{(1)}=-10,34661 & & \theta_{3}^{(0)}=-19,67751 & & & & \\
 & \theta_{0}^{(2)}=1,64979 & & \theta_{2}^{(1)}=1,30070 & & & \theta_{4}^{(0)}=1,16407 & & \\
\theta_{-1}^{(3)}=0 & & \theta_{1}^{(2)}=-10,89324 & & \theta_{3}^{(1)}=-21,83296 & & & & \\
 & \theta_{0}^{(3)}=1,55134 & & \theta_{2}^{(2)}=1,20929 & & & & & \\
\theta_{-1}^{(4)}=0 & & \theta_{1}^{(3)}=-11,64008 & & \theta_{3}^{(2)}=-28,43551 & & & & \\
 & \theta_{0}^{(4)}=1,46543 & & \theta_{2}^{(3)}=1,14975 & & & & & \\
\theta_{-1}^{(5)}=0 & & \theta_{1}^{(4)}=-12,63743 & & & & & & \\
 & \theta_{0}^{(5)}=1,38630 & & & & & & & \\
\theta_{-1}^{(6)}=0 & & \theta_{1}^{(5)}=-13,96843 & & & & & & \\
 & \theta_{0}^{(6)}=1,31471 & & & & & & &
\end{array}
$$

Remarque 3.0.1 *Cet exemple montre que le θ-algorithme accélére la convergence de la suite $\{S_n\}$ que l'ε-algorithme et le ρ-algorithme.*

3.0.2 Comparaison entre l'ε-algorithme et le procédé d'Overholt

Exemple 3.0.2 *Soit $x = e^{-x}$, on accélérons la convergence de la suite $\{x_n\}$ à l'aide de l'$\varepsilon-$algorithme et de procédé d'overholt; on obtient:*

premierment on appliqant l'$\varepsilon-$algorithme:

$$\varepsilon_0^{(0)} = S_0 = 0,367879$$

$$\varepsilon_1^{(0)} = 3,083364$$

$$\varepsilon_0^{(1)} = S_1 = 0,692200 \qquad \varepsilon_2^{(0)} = 0,571706$$

$$\varepsilon_1^{(1)} = -5,215749 \qquad .....$$

$$\varepsilon_0^{(2)} = S_2 = 0,500473 \qquad \varepsilon_2^{(1)} = 0,568638$$

$$\varepsilon_1^{(2)} = 9,454476 \qquad .....$$

$$\varepsilon_0^{(3)} = S_3 = 0,606243 \qquad \varepsilon_2^{(2)} = 0,567617$$

$$\varepsilon_1^{(3)} = -16,434663 \qquad .....$$

$$\varepsilon_0^{(4)} = S_4 = 0,545396 \qquad \varepsilon_2^{(3)} = 0,567296$$

$$\varepsilon_1^{(4)} = 29,226093 \qquad .....$$

$$\varepsilon_0^{(5)} = S_5 = 0,579612 \qquad \varepsilon_2^{(4)} = 0,567193$$

$$\varepsilon_1^{(5)} = -51,289942 \qquad .....$$

$$\varepsilon_0^{(6)} = S_6 = 0,560115 \qquad \varepsilon_2^{(5)} = 0,567182$$

$$\varepsilon_1^{(6)} = 90,195724$$

$$\varepsilon_0^{(7)} = S_7 = 0,571202$$

Ensuite, on appliqant le procédé **d'overholt;** on obtient:

$$
\begin{array}{lll}
v_0^{(0)} = S_0 = 0,367879 & & \\
 & v_1^{(0)} = 3,083364 & \\
v_0^{(1)} = S_1 = 0,692200 & & v_2^{(0)} = 0,571706 \\
 & v_1^{(1)} = -5,215749 & ..... \\
v_0^{(2)} = S_2 = 0,500473 & & v_2^{(1)} = 0,567296 \\
 & v_1^{(2)} = 9,454476 & ..... \\
v_0^{(3)} = S_3 = 0,606243 & & v_2^{(2)} = 0,567193 \\
 & v_1^{(3)} = -16,434663 & ..... \\
v_0^{(4)} = S_4 = 0,545396 & & v_2^{(3)} = 0,567182 \\
 & v_1^{(4)} = 29,226093 \quad ..... & ..... \\
v_0^{(5)} = S_5 = 0,579612 & & \\
 & v_1^{(5)} = -51,289942 \quad ..... & ..... \\
v_0^{(6)} = S_6 = 0,560115 \quad ..... & & \\
 & & \\
v_0^{(7)} = S_7 = 0,571202 \quad ..... & &
\end{array}
$$

Remarque 3.0.2 *Après l'accélération de la convergence à l'aide de l'ε-algorithme et de procédé d'Overholt, nous remarquons que le procédé d'Overholt accélère la convergence en un nombre d'itérations moins que l'ε-algorithme.*

3.0.3 Exemple de résolution d'un système linéaire

Exemple 3.0.3 *Soit le système* $A = \begin{pmatrix} 2 & 1 & 3 & 4 \\ 1 & -3 & 1 & 5 \\ 3 & 1 & 6 & -2 \\ 4 & 5 & -2 & -1 \end{pmatrix}, x = \begin{pmatrix} 10 \\ 4 \\ 8 \\ 6 \end{pmatrix}, x_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$

À l'aide de la méthode itérative de ***Gauss-Seidel,*** *la matrice admet comme solution pour* $k = 4$

$$x_1 = \begin{pmatrix} 5 \\ -1,333 \\ 1,333 \\ -6 \end{pmatrix}, x_2 = \begin{pmatrix} 15,66 \\ -9,22 \\ -2,94 \\ 4,66 \end{pmatrix}, x_3 = \begin{pmatrix} 4,69 \\ 10,68 \\ -3,41 \\ 16,44 \end{pmatrix}, x_4 = \begin{pmatrix} -28,12 \\ 26,50 \\ 2,68 \\ 73,02 \end{pmatrix}$$

Pour trouver la solution exacte de ce système, nous pouvons utiliser l'ε-algorithme vectoriel.

La matrice A est une matrice carrée, nous calculons à la suite $(I-A)^{-1}$:

$$(I-A)^{-1} = \begin{pmatrix} \frac{87}{752} & -\frac{73}{376} & -\frac{71}{752} & -\frac{15}{94} \\ -\frac{73}{376} & \frac{31}{188} & \frac{25}{376} & -\frac{2}{47} \\ -\frac{71}{752} & \frac{25}{376} & -\frac{89}{752} & \frac{9}{94} \\ -\frac{15}{94} & -\frac{2}{47} & \frac{9}{94} & -\frac{1}{47} \end{pmatrix}$$

Calculons maintenant $(I-A)^{-1} * b$

$$(I-A)^{-1} * b = \begin{pmatrix} -1,3324 \\ -1,0053 \\ -1,0505 \\ -1,1277 \end{pmatrix}$$

Calculons $x_0 - (I-A)^{-1} * b$

$$x_0 - (I-A)^{-1} * b = \begin{pmatrix} 1,3324 \\ 1,0053 \\ 1,0505 \\ 1,1277 \end{pmatrix}$$

donc $\varepsilon_{2(m-r)}^{(n+r)} = (I-A)^{-1} * b \quad n = 0, 1, ...$

m: le degré de polynôme minimal de A pour le vecteur $x_0 - (I-A)^{-1} * b$

Calculons le polynôme minimal de A :

-le polynôme caractéristique de A est : $X^4 - 4X^3 - 73X^2 + 260X + 568$,

-le polynôme minimal de la matrice A :$m_A(X) = X^4 - 4X^3 - 73X^2 + 260X + 568$

puisque $m_A(X) = 0$. Donc $m = 4$ et $r = 0$ (r est la multiplicité éventuelle de la racine zéro), alors :

$$\varepsilon_8^{(n)} = (I - A)^{-1} * b$$

$$\varepsilon_8^{(n)} = \begin{pmatrix} 1,3324 \\ 1,0053 \\ 1,0505 \\ 1,1277 \end{pmatrix}$$

On obtient alors:

k	$\left\{\varepsilon_{2k}^{(0)}\right\}$
0	0.
	0.
	0.
	0.
1	−4.41
	0.27
	0.32
	−2.16
2	4.06
	−1.34
	−0.92
	−4.89
3	−0.64
	1.60
	0.63
	1.24
4	1.0
	1.0
	1.0
	1.0

Conclusion

Les méthodes présentées dans ce travail sont des méthodes pour accélérer la convergence des suites et des fonctions. Ces méthodes offrent de nombreuses possibilités, il est évident que ces méthodes présentent un certain nombre de limitation, la plus importante est sans doute l'instabilité numérique.

Dans ce domaine de l'analyse numérique de nombreuses questions sont encore sans réponses. Des recherches sont actuellement en cours pour obtenir de nouveaux algorithmes plus puissants, pour démontrer des résultats de convergence et d'accélération de la convergence.

Malgré tout, il y'a de nombreux domaines où ces méthodes peuvent rendre des grands services et aussi permettre la résolution de problèmes nouveaux.

Bibliographie

[1] [1]-A,C,Aitken-on Bernouilli numerical solution of algebraic equations, proc, Roy, soc, Edinburgh, 46(1962).

[2]-A, Dranx, l'ε-algorithme, publ, Ano 115, Univ de Lille 1(1983).

[3]-B, Germain-Bonne, Estimation de la limite de suites et formalisation de procédés d'accélération de convergence, thèse, lille, 1978.

[4]-Claude Brezinski, Algorithmes d'accélération de la convergence étude numérique, Editions technip-paris 1978.

[5]-D,Pett, Etude de quelques procédés d'accélération de la convergence, Thèse 3ème cycle, Lille, 1977.

[6]-E,Gekeler, On the solution of systémes of equations by the epsilon algorithme of Wynn, Math, Comp, 26(1972), pp, 427-436.

[7]-F, cordellier, particular rules for the vector ε-algorithm, Numer, Math, 27(1977), pp, 203-207.

[8]-Franck Jedrzejewski, Introduction aux méthodes numériques, deuxième édition, Springer.

[9]-JEAN-Paul de la haye, optimalité du procédé Δ^2d'aitken pour l'accélération de la convergence linéaire, RAIRO-analyse numérique,Tome 15,n$^\circ$4(1981), p, 321-330.

[10]-Master 1 Métiers de l'enseignement, mathématiques, Fiche de mathématiques 3-le Δ^2d'aitken et l'$\varepsilon-$algorithme, Ulco, la Mi-voix, 2011/2012.

[11]-M,N,Barber, c j, Hamer, Extrapolation of sequences using a generalized epsilon algorithm, J,Austral, Math, soc, ser. B, 23(1982), pp.229-240.

[12]-M, Kateb, Itérations d'algorithmes d'accélération de la convergence, thesis 3rd cycle, Univ, de Lille(1983).

[13]-P,J, Laurent-approximation et optimisation, Hermam, Paris 1972.

[14]-S Akesbi, M R Laydi et M Mokhtar kharroubi,Décomposition d'opérateurs et accélération de la convergence en neutronique,C,R,Acad.sci.Paris,t.319,série I,p.

[15]-.Takéo Takahashi, cours électif CE 33, Ecole des Mines de Nancy-Campus Artem 54042 Nancy cedex.

[16]-Thèse de Christophe Ronald, Lille1,2005, Méthodes d'accélération de convergence en Analyse numérique et statistique, Université des sciences et Technologiques de Lille

[17] P,Hillion, méthode d'aitken itérée pour suites oscillantes d'approximations, C,R,A cad, sci paris, 280 A(1975),pp,1701-1703.

[18] Thèse de Mohammed Ziani, accélération de la convergence - méthodes de type Newton pour la résolution des systèmes non linéaires, université de Rennes-France.

Annexes

Méthode de Bairstow

La méthode de Bairstow permet de déterminer les zéros d'un polynôme.C'est une application de la méthode de Newton-Raphson, qui consiste à factoriser à chaque étape un trinôme du second degré dont les racines sont les racines du polynôme initial. Soit

$$P(x) = a_0x^n + a_1x^{n-1} + ... + a_{n-1}x + a_n$$

On écrit P sous la forme

$$P(x) = (x^2 + px + q)P_{n-2}(x) + R(x) + S$$

avec

$$P_{n-2}(x) = b_0x^{n-2} + ... + b_{n-3}x + b_{n-2}$$

et on cherche à déterminer p et q de façon à annuler R et S (ce qui n'est pas toujours possible). On pose $b_{n-1} = R$ et $S = pb_{n-1} + b_n$ L'algorithme est alors le suivant : On se donne deux constantes arbitraires p_0 et q_0 et on calcule les coefficients b_n définis par:

$$\begin{cases} b_0 = a_0 \\ b_1 = a_1 - p_0b_0 \\ b_2 = a_2 - p_0b_1 - q_0b_0 \\ \dots \\ b_{n-1} = a_{n-1} - p_0b_{n-2} - q_0b_{n-3} \\ b_n = a_n - p_0b_{n-1} - q_0b_{n-2} \end{cases}$$

On calcule ensuite les coefficients C_n définis par:

$$\begin{cases} C_0 = b_0 \\ C_1 = b_1 - p_0C_0 \\ C_2 = b_2 - p_0C_1 - q_0C_0 \\ \dots \\ C_{n-1} = b_{n-1} - p_0C_{n-2} - q_0C_{n-3} \\ C_n = b_n - p_0C_{n-1} - q_0C_{n-2} \end{cases}$$

En posant

$$\begin{aligned} p_1 &= \frac{b_{n-1}C_{n-2} - b_n C_{n-3}}{C_{n-2}^2 - C_{n-1}C_{n-3}} \\ q_1 &= \frac{b_n C_{n-2} - b_{n-1}C_{n-1}}{C_{n-2}^2 - C_{n-1}C_{n-3}} \end{aligned}$$

On reprend les mêmes opérations avec les valeurs de p_1, q_1 à la place de p_0, q_0, et ainsi de suite. On obtient de cette manière un couple (p_j, q_j). On arrête l'itération lorsque le test d'arrêt suivant pour ε donné, est vérifié

$$\frac{|p_j - p_{j-1}| + |q_j - q_{j-1}|}{|p_j + q_j|} < \varepsilon$$

La résolution du trinôme $x^2 + p_j x + q_j = 0$ donne deux racines de $P_n(x)$.

On recommence les mêmes opérations avec le polynôme $P_{n-2}(x)$ jusqu'à ce que le polynôme résiduel soit de degré inférieur à 2.

Méthode de Jacobi

Dans la méthode de Jacobi, encore appelée méthode des déplacements simultanés, la matrice A du système $Ax = b$ est décomposée A dans laquelle on a remplacé les éléments de la diagonale par des zéros $N = E + F$ La matrice $J = M^{-1}N = D^{-1}(E + F) = I - D^{-1}A$ est appelée matrice deJacobi. À chaque pas, on calcule

$$x_i^{(k+1)} = (b_i - \sum_{j \neq i, j=1}^{n} a_{ij} x_j^{(k)})/a_{ii}$$

À chaque itération, on effectue $(n-1)$ multiplications, n additions et une division. Pour stocker A et les vecteurs b, x_k et x_{k+1} on utilise $(n^2 + 3n)$ mémoires. La méthode ne converge pas toujours. On démontre que si A est une matrice définie positive, la méthode itérative converge. De même, si A est une matrice diagonalement dominante, c'est-à-dire si

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|$$

alors la méthode de Jacobi converge. Par conséquent, on peut avoir intérêt à réarranger les termes de A de façon à mettre A sous la forme d'une matrice dont les éléments diagonaux sont les plus grands possibles. On démontre que si A est une matrice tridiagonale par blocs, la méthode converge.

Méthode de Regula Falsi

Soit $f : [a; b] \to \mathbb{R}$ une fonction continue avec $f(a)f(b) < 0$ (cette fonction satisfait donc les hypothèses permettant d'appliquer la méthode de la bissection). La méthode de la bissection converge lentement car à chaque étape la longueur de l'intervalle $[a; b]$ encadrant la racine r recherchée est exactement multipliée par $\frac{1}{2}$

Le but de la méthode **Regula Falsi** (appelée aussi méthode de la fausse position) est de fournir à chaque étape un nouvelle encadrement qui multiplie la

longueur de l'encadrement précédemment considéré par un facteur inférieur à $\frac{1}{2}$. Pour cela, nous approximons la fonction f par la droite passant par les points $(a; (f(a)))$ et $(b; f(b))$ et l'abscisse x_1 du point d'intersection de cette droite avec l'axe des x constitue une des bornes du nouvelle encadrement. Le nouvelle encadrement considéré ($[a; x_1]$ ou $[x_1; b]$) est choisi, comme pour la méthode de la bissection, de façon à ce que les images des bornes soient de signes opposés. Nous répétons l'opération jusqu'à atteindre la précision désirée.

Méthodes de Schröder

• **Conditions d'utilisation:**

Recherche d'une racine unique dans le voisinage I de x_0.Cette méthode itérative nécessite le calcul des m premières dérivées pour une méthode de Schröder d'ordre $m + 1$. Il faut donc que la fonction dont on cherche la racine soit au moins m fois dérivable dans l'intervalle I

• **Principe de la méthode:**

On bâtit une suite récurrente de la façon suivante :

-autour du point x_n, effectue un développement limité de $f(x_n)$ jusqu'à l'ordre m.

-On définit x_{n+1} comme la racine du développement limité de $f(x_n)$

-On itère...

Schröder d'ordre 2: méthode de Newton

• Méthode du 2nd ordre : on néglige les termes à partir du 2nd ordre:

$$f(x_n) = 0 = f(x_n) + (x_{n+1} - x_n)f'(x_n)$$

•x_{n+1}: racine du développement limité à l'ordre 1 de f au voisinage de x_n:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Schröder d'ordre 3:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} - \frac{f''(x_n)f^2(x_n)}{2f'^3(x_n)}$$

Schröder d'ordre 4:

$$x_{n+1} = x_n - \frac{f''(x_n)f^2(x_n)}{2f'^3(x_n)} - \frac{3f''^2(x_n)f'(x_n)f'''(x_n)}{6f'^5(x_n)}f^3(x_n)$$