Université de Lille

UFR de mathématiques

TRAVAIL ENCADRÉ DE RECHERCHE : Formules de quadratures rationnelles de Gauss-Tchebychev

Jonathan VANDERVEKEN

Nizar CHAKRI

Encadrant :
Karl DECKERS

10 mai 2019

Table des matières

Introduction

Le concept des fonctions rationnelles orthogonales (FRO) a été formalisé dans les années 60 par le mathématicien russe M.Džrbašian.

Ces fonctions représentent une généralisation des polynômes orthogonaux qu'on retrouve lorsqu'on travaille avec des pôles à l'infini, ce qui implique que plusieurs résultats valables pour ces polynômes peuvent être étendus au cas des (FRO).

Les (FRO) à pôles réels sur un intervalle ont fait l'objet d'études poussées en parallèle avec celles concernant les polynômes orthogonaux, cependant, le cas des pôles complexes restait moins bien connu.

Les recherches de K.DECKERS et A.BUTHEEL sont venues élargir cet aspect des ORFS, et nous vous proposons dans ce TER de découvrir une extension des formules de quadrature de Gauss-Tchebyshev (cas polynômial basé sur des polynômes orthogonaux) au cas rationnel (basé sur des fonctions rationnelles orthogonales).
En premier temps, nous vous proposons d'aborder la théorie derrière les (FRO).
Nous nous intéresserons ensuite au formules de quadratures rationnelles de Gauss-Tchebychev.

Finalement, nous analyserons l'algorithme RCHEB qui permet de calculer les noeuds et poids des formules de quadrature rationnelles de Gauss-Chebyshev.

Chapitre 1

Quadrature de Gauss-Tchebychev

1.1 Fonctions rationnelles orthogonales

Dans tout ce qui va suivre :

I désignera l'intervalle [-1,1], $\bar{\mathbb{C}}$ désignera $\mathbb{C} \cup \{\infty\}$ et enfin $\bar{\mathbb{C}}_I$ pour $\bar{\mathbb{C}}$ privé de I.
On note $\mathbb{D}$ disque (complexe) ouvert de rayon 1, $\mathbb{D} = \{z \in \mathbb{C} \text{ tel que } |z| < 1\}$
et $\mathbb{T}$ cercle (complexe) unité, $\mathbb{T} = \{z \in \mathbb{C} \text{ tel que } |z| = 1\}$. $\Re(z), \Im(z)$ respectivement la partie réelle et imaginaire d'un complexe z.

Définition 1.1. Pour toute fonction complexe f, on définit "l'opération involution" par $f_*(t) = \overline{f(1/\bar{t})}$ et "le super-c conjugué" par $f^c(t) = \overline{f(\bar{t})}$.

Les fonctions rationnelles avec lesquelles nous allons travailler sont de la forme suivante :

$$f_n(x) = \frac{(c_k x^k) + (c_{k-1} x^{k-1}) + (c_{k-2} x^{k-2}) + ... + c_0}{(1 - \frac{x}{\alpha_1})(1 - \frac{x}{\alpha_2})(1 - \frac{x}{\alpha_3})...(1 - \frac{x}{\alpha_k})}\,,\ \text{k=1,2,....}$$

avec $c_k, ...c_0$ arbitraires dans $\mathbb{C}$ et $\alpha_1, ...\alpha_k$ dans $\bar{\mathbb{C}}_0$ où $\bar{\mathbb{C}}_0$ correspond à $\bar{\mathbb{C}}$ privé de 0 .

Définissons les facteurs

$$Z_k(x) = \frac{x}{1 - \frac{x}{\alpha_k}} \text{ et les fonctions} \tag{1.1}$$

$$b_0(x) \equiv 1\,, b_k(x) = b_{k-1}(x) Z_k(x) \text{ pour } k = 1, 2, \ldots \tag{1.2}$$

Définition 1.2. Soit $\mathcal{L}_\text{k} = vect\{b_0, ..., b_k\}$ l'espace des fonctions rationnelles pour $k \geq 0$, les pôles $(\alpha_1, ...\alpha_k)$ sont dans $\bar{\mathbb{C}}_I$
$\mathcal{L}_{-1} = \{0\}$ et $\mathcal{L} = \bigcup_{k=0}^{\infty} \mathcal{L}_\text{k}$

On peut définir $\mathcal{L}^{\mathrm{c}}$ grâce au "super-c conjugué" défini plus tôt, $\mathcal{L}^{\mathrm{c}} = \{f, f^c \in \mathcal{L}\}$. Soit

$$\pi_0(x) \equiv 1 \;\; \pi_k(x) = \prod_{j=1}^{k}(1 - x/\alpha_j), k = 1, 2, \ldots$$

ce qui permet d'écrire $\mathcal{L}_{\mathrm{k}}$ d'une autre manière, $\mathcal{L}_{\mathrm{k}} = \{p_n/\pi_n : p_n \in \mathcal{P}_n\}$
où $\mathcal{P}_n = vect\{1, x, \ldots, x^n\}$.

Remarque. Si on est dans $\mathbb{T}$, les fonctions rationnelles avec lesquelles nous travaillerons seront alors de la forme

$$f_n(z) = \frac{(c_k z^k) + (c_{k-1} z^{k-1}) + (c_{k-2} z^{k-2}) + ... + c_0}{(1 - z\overline{\beta_1})(1 - z\overline{\beta_2})(1 - z\overline{\beta_3})...(1 - z\overline{\beta_k})}\,,\ \text{k=1,2,....}\ .$$

Définition 1.3. Soit

$$\zeta_k(z) := \zeta_{\beta_k}(z) = \eta_{\beta_k}\frac{z - \beta_k}{1 - \bar{\beta}_k z}$$

les facteurs de Blaschke avec k=1,2,... où β_k est un nombre complexe et

$$\eta_{\beta_k} = \begin{cases} 1 \; si \; \beta_k = 0 \\ -\dfrac{\bar{\beta}_k}{|\beta_k|} \; sinon \end{cases}$$

et soit $B_0(z) = 1$, $B_k(z) = B_{k-1}(z)\zeta_k(z)$ les produits de Blaschke.

Définition 1.4. On définit la "transformation super étoile" d'une fonction complexe $f_k(x) \in \mathcal{L}_{\mathrm{k}} \backslash \mathcal{L}_{\mathrm{k}-1}$ par

$$f_k^*(x) = f_k^c(x)\frac{b_k(x)}{b_k^c(x)}.$$

Considérons une mesure μ positive sur I et un produit scalaire

$$\langle f, g\rangle_\mu = \int_I f(x)g^c(x)d\mu(x) \quad \text{avec} \;\; f, g \in L_2\,.$$

Si μ est une mesure absolument continue on a :

$$\langle f, g\rangle_\mu = \int_I f(x)g^c(x)\mu'(x)dx = \int_I f(x)g^c(x)\omega(x)dx \tag{1.3}$$

où μ' s'appelle alors la dérivée de Radon-Nikodym et ω est une fonction de pondération (aussi nommée fonction poids). Notons que lorsque $x \in \mathbb{R}$ on a dans ce cas $g^c(x) = \overline{g(x)}$.

Définition 1.5. On appelle fonctions rationnelles orthonormées, les fonctions $\varphi_k \in \mathcal{L}_k \backslash \mathcal{L}_{k-1}$ tel que $\forall k, h \in \mathbb{N}$ on ait

$$\langle \varphi_k, \varphi_h \rangle = \delta_{k,h} \quad \text{où } \delta_{k,h} = 1 \text{ si } k = h \text{ et vaut 0 sinon.}$$

Définition 1.6. Pour $\tau \in \mathbb{T}$, $n \geq 1$ et φ_n fonction rationnelle orthonormée, on définit les fonctions rationnelles para-orthogonales

$$Q_{n,\tau}(x) = \varphi_n(x) + \tau \varphi_n^*(x) \tag{1.4}$$

Théorème 1.1. *Soit $Q_{n,\tau} = \dfrac{q_n(x)}{\pi_n(x)}$ une fonction rationnelle para-orthogonale , supposons qu'aucun des zéros $x_{n,k}$ de q_n ne coïncide avec l'un des pôles (les zéros sont simples et réels).*
Définissons

$$\lambda_{n,k} = \frac{1}{\left(\sum_{j=0}^{n-1} |\varphi_j(x_{n,k})|^2\right)} \tag{1.5}$$

Alors la formule de quadrature suivante

$$\int_I f(x)\omega(x)dx \approx \sum_{k=1}^{n} \lambda_{n,k} f(x_{n,k}) \tag{1.6}$$

est exacte pour $f \in \mathcal{L}_{n-1}.\mathcal{L}^c_{n-1}$ où $\mathcal{L}_{n-1}.\mathcal{L}^c_{n-1} = \{\alpha_1, \overline{\alpha_1}, \ldots, \alpha_{n-1}, \overline{\alpha_{n-1}}\}$ et si α_n est réel, (1.6) *est exacte pour $f \in \mathcal{L}_n.\mathcal{L}^c_{n-1}$*

Démonstration. voir [1] page 14 □

On a la table suivante qui nous donne les différentes fonctions poids de Tchebychev

j	$\omega_j(x)$	c	d	p	q
1	$(1-x^2)^{-1/2}$	1	1	1	-1
2	$\left(\frac{1-x}{1+x}\right)^{1/2}$	3/2	0	1	1
3	$(1-x^2)^{1/2}$	2	0	2	1

TABLE 1.1 – Les fonctions poids de Tchebychev $\omega_j(x)$ et les paramètres c,d,p,q

1.2 Quadrature rationnelle de Gauss-Tchebychev

Lemme 1.2. *(admis) Supposons $\mathcal{A} = (\alpha_k)_{k=1}^{\infty} \subset \bar{\mathbb{C}}_I$ et $\mathcal{B} = (\beta_k)_{k=1}^{\infty} \subset \mathbb{D}$ et considérons les facteurs $Z_k(x), \zeta_k(z), \zeta_k^c(z)$. Alors il existe des constantes c_k et d_k qui ne dépendent que de $\mathcal{A}$ et de $\mathcal{B}$ telles que*

$$Z_k\left(\frac{z+z^{-1}}{2}\right) = c_k\zeta_k^c(z) + d_k + c_k\zeta_k^{-1}(z) \iff \alpha_k = \frac{\beta_k + \beta_k^{-1}}{2}$$

Les constantes c_k et d_k sont de la forme suivante

$$c_k = \frac{\eta_{\beta_k}(1+\beta_k^2)^2}{2(1-\beta_k^2)(1-|\beta_k|^2)} \text{ et } d_k = \frac{(\beta_k + \overline{\beta_k})(1+\beta_k^2)^2}{(1-\beta_k^2)(1-|\beta_k|^2)}$$

Pour les pôles complexes en dehors de I, si $\alpha_k = \dfrac{\beta_k + \beta_k^{-1}}{2}$ alors $b_k(x)$ est une combinaison linéaire de $B_k^{-1}(z), B_{k-1}^{-1}(z), \ldots, B_0^{-1}(z) = B_0^c(z), B_1^c(z), \ldots, B_{k-1}^c(z), B_k^c(z)$. Nous allons donner une expression pour les fonctions rationnelles orthonormées de Tchebychev liées aux fonctions de pondérations de la table [1.1], avec des pôles complexes en dehors de I.

Démonstration. voir [1] page 28 □

Théorème 1.3. *Soit $x = 0.5(z + z^{-1}) \in \bar{\mathbb{C}}$ et $\alpha_k = 0.5(\beta_k + \beta_k^{-1}) \in \bar{\mathbb{C}}$. Supposons que p et q soient définis comme la table [1.1].*
Soit N_k le facteur de normalisation donné par

$$N_k = \sqrt{\frac{2^j(1-|\beta_k|)}{\pi}} \tag{1.7}$$

alors les fonctions rationnelles orthonormées sont données par

$$\varphi_k(x) = \frac{qN_k}{2z^{j-1}+q-3}\left(\frac{z^j B_{k-1}^c(z)}{1-\beta_k z} - \frac{q}{(z-\beta_k)B_{k-1}(z)}\right) \tag{1.8}$$

pour $k \geq 1$ et $\varphi_0(x) \equiv \sqrt{p/\pi}$

Démonstration. voir [1] page 28 □

Ainsi avec le théorème précédent on a une expression de $\varphi_n(x)$ selon la fonction de pondération de Tchebychev $\forall n \geq 1$

$$\varphi_n(x) = \begin{cases} N_n\left(\dfrac{zB_{n-1}^c(z)}{1-\beta_n z} + \dfrac{B_{(n-1)*}(z)}{z-\beta_n}\right) & ,\omega(x) = \dfrac{1}{\sqrt{1-x^2}} \\ \dfrac{N_n\sqrt{2}}{z-1}\left(\dfrac{z^2 B_{n-1}^c(z)}{1-\beta_n z} - \dfrac{B_{(n-1)*}(z)}{z-\beta_n}\right) & ,\omega(x) = \sqrt{\dfrac{1-x}{1+x}} \\ \dfrac{N_n 2}{z^2-1}\left(\dfrac{z^3 B_{n-1}^c(z)}{1-\beta_n z} - \dfrac{B_{(n-1)*}(z)}{z-\beta_n}\right) & ,\omega(x) = \sqrt{1-x^2} \end{cases}$$

Nous allons maintenant construire les quadratures rationnelles de Gauss-Tchebychev, les noeuds sont les zéros de $Q_{n,\tau}$ définis par (1.4), on observe que $\varphi_n^c(x) = \overline{\varphi_n(x)}$ pour $x \in \mathbb{R} \cup \{\infty\}$.

Théorème 1.4. *Soit $x+iy$ un nombre complexe et $arctan(y/x) \in [-\frac{\pi}{2}, \frac{3\pi}{2}[$ son argument, définissons*

$$\beta_{n,\tau} = \frac{\beta_n + \tau\overline{\beta_n}}{1+\tau}. \tag{1.9}$$

Supposons c et d définis comme dans la table [1.1] et on définit

$$F(\theta) = \sum_{j=1}^{n-1} f_{\beta_j}(\theta) + 0.5 f_{\beta_{n,\tau}}(\theta) - (n-c)\theta, \tag{1.10}$$

$$f_\beta(\theta) = arctan\left(\frac{sin(\theta) - \Im(\beta)}{cos(\theta) - \Re(\beta)}\right) + arctan\left(\frac{sin(\theta) + \Im(\beta)}{cos(\theta) - \Re(\beta)}\right) \tag{1.11}$$

Soient $x_{n,k}$ les zéros de $Q_{n,\tau}(x)$, fixons $x_{n,k} = cos(\theta_{n,k})$ et $\tau \in \mathbb{T}\backslash\{-1\}$ de telle sorte que $|\beta_{n,\tau}| < 1$, alors ces zéros vérifient l'équation suivante

$$F(\theta_{n,k}) = \pi(k - d/2)\ , k = 1, 2, \ldots n \tag{1.12}$$

Démonstration. voir [1] page 30 □

Notons que $\tau = -1$ est une valeur critique pour laquelle un des zéros de $Q_{n,\tau}(x)$ sera l'infini et donc en dehors de l'intervalle I, ce qui explique la condition sur τ dans le théorème ci-dessus. Si $\tau \neq 1$ alors $Q_{n,\tau}(x)$ aura n zéros.

Théorème 1.5. *Chaque racine de $Q_{n,\tau}(x) \in]-1, 1[$ si et seulement si $\tau \neq -1$ et τ est choisi de tel sorte que $|\beta_{n,\tau}| < 1$ avec $\beta_{n,\tau}$ donnée par (1.9)*

Démonstration. voir [1] page 32 □

Remarque. Observons que $\tau = 1$ est le meilleur choix pour $\tau \in \mathbb{T}$, en effet $\forall n$ de telle sorte que $\beta_n \in \mathbb{D}$ on a $|\beta_{n,1}| < 1$, ce qui n'est pas possible avec les autres $\tau \in \mathbb{T}$ car on peut trouvé un $\beta_n \in \mathbb{D}$ tel que $|\beta_{n,\tau}| \geq 1$.

Théorème 1.6. *Supposons que les pôles $\{\alpha_1, \alpha_2, \ldots\}$ sont délimités en dehors de I et que la distribution asymptotique des pôles est donnée par une mesure ν sur $\bar{\mathbb{C}}_I$, c'est à dire que pour toute fonction f continue avec un support compact*

$$\lim_{n\to+\infty} \frac{1}{n} \sum_{k=1}^{n} f(\alpha_k) = \int f(z) d\nu(z)$$

Si $\nu = s\delta_\infty + (1-s)\nu_0$ avec $0 \leq s \leq 1$ et $\int log|t| d\nu_0(t) < \infty$ alors la distribution asymptotique des zéros de $Q_{n,\tau}(x)$ est donnée par une mesure absolument continue λ avec la fonction poids suivante

$$\lambda'(x) = \frac{1}{\pi\sqrt{1-x^2}} \int \Re\{\frac{\sqrt{t^2-1}}{t-x}\} d\nu(t)$$

Démonstration. voir [1] page 34-35. □

Le théorème suivant nous permet d'obtenir une formule des poids $\lambda_{n,k}$ en fonction des noeuds $x_{n,k}$

Théorème 1.7. *Soient*

$$R(x,\beta) = 1 - 2\Re\{\beta\}x + |\beta|)^2 \text{ et } I(x,\beta) = 4(\Im\{\beta\})^2(1-x^2)$$

et définissons

$$g_n(x) = 2\sum_{j=1}^{n-1} \frac{R(x,\beta_j)(1-|\beta_j|)^2}{(R(x,\beta_j))^2 - I(x,\beta_j)} + \frac{1-\beta_{n,\tau}^2}{1-2\beta_{n,\tau}x+\beta_{n,\tau}^2}$$

Alors les poids dans les formules de quadrature de Gauss basés sur les fonctions rationnelles para-orthonormées $Q_{n,\tau}(x)$ peuvent être données comme fonctions des noeuds $x_{n,k}$ suivant

$$\lambda_{n,k} = 2\pi \frac{1-(1-d)x_{n,k}^{j-1}}{j+g_n(x_{n,k})},\ k = 1,2,\ldots,n \tag{1.13}$$

Chapitre 2

Explication de l'algorithme RCHEB

Dans ce chapitre, un mot entre guillemets va référer à l'algorithme qui lui correspond et " :" pour les variables de sorties. Les phrases entre accolades correspondent aux commentaires.

L'algorithme **RCHEB** prend trois arguments : un réel ε correspondant à la précision machine, un entier w entre 1 et 3 qui correspond aux fonctions poids de la table(1.1) et un vecteur complexe α (les pôles).
Il nous renvoie trois variables : les poids λ , les noeuds x_i et une erreur.
Dans un premier temps on vérifie que les pôles $\alpha_1, ...,$ ne sont pas trop proches de l'intervalle I sinon les points x_i seront proches de cet intervalle.
Ensuite on détermine le nombre de pôles distincts et on les trie grâce à "psort", puis on crée les points pour l'interpolation cubique d'Hermite par morceaux (qu'on appellera 'ttk').
Si les pôles sont tous égaux et valent $+\infty$, on est dans le cas polynomial ($x_i = cos(ttk)$, λ comme (1.13)) ; si les pôles ne sont pas tous égaux, on a $\beta_j = \dfrac{1}{\alpha_j + signe(\alpha_j)\sqrt{\alpha_j^2 - 1}}$.
On évalue F définie comme (1.10) et de sa dérivée aux points θ à l'aide de "ceval", "ftheta", "fdtheta", puis on effectue l'interpolation cubique d'Hermite par morceaux (que nous allons nommer pchip) afin d'obtenir une approximation de F^{inv}.
On utilise pchip plutôt qu'une autre interpolation par spline pour plusieurs raisons : elle conserve la monotonie, elle est moins coûteuse.
On utilise la méthode de Newton dans le but d'acquérir les noeuds (la méthode est quadratique, donc il y aura peu d'itérations avant de converger), si la méthode de Newton diverge, on cherche de nouvelles valeurs initiales avec "newinit" et on fait cela encore 2 fois si cela diverge.
Si après les autres tentatives on ne trouve pas de noeuds qui convergent on applique la méthode de dichotomie avec "bisect" et "bound".

$x_i = cos(tk)$, puis on utilise "poids" pour calculer les poids λ liés aux x_i et "ceval" pour estimer les erreurs commises sur les noeuds.

Algorithm 1 RCHEB

Require : α, w, ε

$pinf \leftarrow 5/\varepsilon$

$ptol \leftarrow 5\varepsilon$

$xtol \leftarrow 50\varepsilon$

$n \leftarrow$ taille de α

On vérifie que les pôles ne sont pas trop proches de I.

$ww \leftarrow (\lfloor w/2 \rfloor, w-1)$

On initialise les points à 0.

$x_i \leftarrow (0, 0, \ldots, 0)$

$\lambda \leftarrow x_i$

On fixe à $+\infty$ les composantes de $\alpha \geq ptol$.

if la dernière composante de alpha(α_n) est $+\infty$ **then**

$bn \leftarrow 0$

else

= **if** $\Re_(\alpha_n) < 0$ **then**

$bn \leftarrow \Re(1/(\alpha_n))$

else

$bn \leftarrow \Re(1/(\alpha_n + 2(\Re(\alpha_n) - 1)\sqrt{(}\alpha_n^2 - 1))$

end if

if $|bn| \leq (1/(2 * pinf))$ **then**

$bn \leftarrow 0$

$\alpha_n \leftarrow +\infty$

else

$\alpha_n \leftarrow (bn + bn^{-1})/2$

end if

end if

if $n == 1$ **then**

return $xi \leftarrow (bn + ww_2 - 2 * ww_1)/(1 + ww_1)$

return $\lambda \leftarrow \pi/(1 + ww_2) - ww_1))$

return $err \leftarrow \varepsilon$

end if

On détermine le nombre de pôles distincts et les trier ("psort") : a .

$m \leftarrow$ nombre de colonnes de a

$ttk \leftarrow (\pi[n(1 - ww_1)/2)]/[(n + ww_2)/2], \ldots, \pi[(1 - ww_1)/2)]/[(n + ww_2)/2])$

if $m == 1$ **then**
  if $bn == 0$ **then**
    $xi \leftarrow cos(ttk)$
    $\lambda \leftarrow 2 * \pi * (1 - ww_1 * xi.ww_2))/(2 * n + ww_2)$
    $err = (\varepsilon, \varepsilon, \ldots, \varepsilon)$
  end if
  $\beta \leftarrow [bn; n]$
else
  for i=1,m **do**
    if $2 * \Re(a_{1,i}) \geq 0$ **then**
      $\beta_i \leftarrow 1/(a_{1,i} + 2\Re(a_{1,i}) - 1)\sqrt{(}a_{1,i}^2 - 1))$
    else
      $\beta_i \leftarrow 1/a_{1,i}$
    end if
  end for
  if $a_{1,m} == +\infty$ **then**
    La dernière composante de β est nulle
  end if
  for i=1,m **do**
    $\beta_i \leftarrow (\beta_i; a_{2,i}/2)$
  end for
end if
for j=1,m **do**
  $r_j \leftarrow |\beta_{1,j}|$
  $\varphi_j \leftarrow$l'angle de $\beta_{1,j}$
  $b_j \leftarrow (1 - r_j) \exp(i\varphi_j)$
  $\beta_j \leftarrow (\beta_j; r_j; \varphi_j; b_j)$
end for

On construit et on évalue les d'interpolation d'HERMITE par morceaux dans le but d'obtenir les valeurs initiales de nos noeuds (”ceval” et ”pchip”).
On utilise la méthode de newton sur nos noeuds (”newton”), cela nous donne 3 variables : $tk, k1, bnd$.

if k1 est non vide **then**
  {si cela diverge}On crée de nouvelles valeurs initiales ("newinit") : $tk0$.
  if tk0 est non vide **then**
    On appelle "newton" : $tk_{n-k1+1}, k2, bnd$.
    if k2 est non vide **then**
      if tk0 a 2 lignes **then**
        $compteur1 \leftarrow 1$
        $compteur2 \leftarrow 1$
        $s \leftarrow (0, \ldots, 0)$ {s de même taille que k2}
        while $compteur1 \leq$ taille que k2 **do**
          if $k1_{compteur2} == k2_{compteur1}$ **then**
            $compteur1 \leftarrow compteur2$
            $s_{compteur1} \leftarrow compteur1 + 1$
          end if
          $compteur2 \leftarrow compteur2 + 1$
        end while
      end if
      On appelle "newton" :$tk_{n-k2+1}, k3, bnd$.
      if k3 est non vide **then**
        {si cela diverge encore}On utilise la méthode de dichotomie avec k3 ("bound", "bissect")=
      end if
      On utilise la méthode de dichotomie avec k2 ("bound", "bissect")
    end if
    On utilise la méthode de dichotomie avec k1 ("bound", "bissect")
  end if
end if
On appelle "poids" : λ
On utilise "ceval" pour avoir nos erreurs : err
return $x_i \leftarrow cos(tk)$
return err
return λ

2.1 Explication des fonctions de l'algorithme

I) **Psort**

La Fonction Psort sert à déterminer le nombre de pôles distincts et à les trier par ordre croissant. Elle prend en argument un vecteur a de pôles complexes en dehors de [-1,1], la tolérance, et renvoie le vecteur a trié avec un vecteur de multiplicités.

Il se peut que a ait deux lignes (matrice), avec la deuxième ligne correspondant aux multiplicités : cela n'arrive que quand "psort" est appelée dans la fonction "newinit". Pour traiter ce cas également, on commence par considérer a comme une matrice.
On associe à p le nombre de lignes (= 1 ou 2) et à n le nombre de colonnes de a
Si n=1, c'est à dire qu'il n'existe qu'un pôle, alors on crée le vecteur contenant ce pôle et sa multiplicité qui est égale à 1.
Sinon :
On trie les éléments de la première ligne de a par ordre croissant (en module puis par angle) et on obtient un nouveau vecteur $a1$ qui correspond à la première ligne de a triée et un vecteur contenant les positions initiales des éléments de $a1$
Comme $a1$ est maintenant trié, une manière de vérifier l'égalité de deux pôles est de diviser élément par élément la différence entre les éléments adjacents par les n-1 premiers éléments de $a1$, de sorte à retrouver un vecteur, ensuite on observe le nombre d'éléments de ce vecteur qui sont plus grands que la tolérance et on en extrait leur position dans un vecteur $a2$
Si $a2$ est un vecteur vide, on crée at formé du dernier élément de a après tri et de n.
Sinon, on fait une différence entre les éléments adjacents de $a2$ afin de créer le vecteur $a3$, on finit par créer le vecteur formé par les pôles triés et leurs multiplicités.

Ensuite, si p=2, on applique le même raisonnement à la première ligne, puis on modifie la deuxième pour en extraire les multiplicités.
Exemple numérique :
Soit a le vecteur (1+2i, 3-4i, 1+2i, -5+1i, -15-4i, -9+10i)
psort(a , 5*2.2204e-16) retourne

$$\text{a} = \begin{pmatrix} 1+2i & 3-4i & -5+1i & -9+10i & -15-4i \\ 2 & 1 & 1 & 1 & 1 \end{pmatrix}$$

II) **Ceval**

Une fois qu'on a calculé les betas en fonction des pôles $\left(\beta_j = \dfrac{1}{\alpha_j + signe(\alpha_j)\sqrt{\alpha_j^2 - 1}}\right)$, on en forme un vecteur en concaténant ces valeurs avec le vecteur de la moitié de la multiplicité des pôles, le vecteur des modules, le vecteur des arguments, et le vecteur contenant l'écriture exponentielle des betas.

Pour $F(\theta) = \sum_{j=1}^{n-1} f_{\beta_j}(\theta) + 0.5 f_{\beta_{n,\tau}}(\theta) - (n-c)\theta$,
Ceval permet d'évaluer $F(\theta)$ si $e = [0, 0]$
$dF(\theta)/d\theta$ si $e = [0, 1]$
$F(\theta) - (k-d)pi$ si $e = [1, 0]$
$(F(\theta) - (k-d)pi)/(dF(\theta)/d\theta)$ si $e = [1, 1]$ avec $d = (1-c_1)/2$ grâce aux fonction "ftheta" et "fdtheta".

On commence par calculer le nombre de pôles distincts et le nombre de points d'évaluation. On réalise un test pour déterminer la méthode d'estimation de f(θ) la plus rapide (boucle sur les points d'évaluation, sur les différents pôles, ou en utilisant une matrice d'argument). Lors de l'évaluation, on utilise la formule de Simpson pour calculer arg(z-β), tel que

$$arg(z-\beta) = \arctan\left(\frac{2sin\frac{\theta-\phi}{2}cos\frac{\theta+\phi}{2} + (1-r)sin(\phi)}{2sin\frac{\theta+\phi}{2}cos\frac{\theta-\phi}{2}t + (1-r)cos(\phi)}\right)$$

Ceci est fait afin d'éviter les risques de perte de précision quand z et proche de β.

III) **Newton**

Prend comme argument d'entrée des valeurs initiales pour les noeuds, une matrice *bnd*, la taille du vecteur des pôles, la matrice beta (pareil que dans ceval), w créé par la choix de la fonction de pondération, le vecteur allant de la taille du vecteur , et la tolérance de précision.
Se base sur l'algorithme de Newton-Raphson pour déterminer une approximation précise des zéros d'une fonction.
Dès qu'un noeud converge, il est omis des calculs suivant afin d'accélérer la performance. Dès qu'un noeud diverge, il est aussi omis des calculs. "Newton" retourne le vecteur des noeuds qui ont convergé et un intervalle contenant la solution exacte au cas où on en aurait besoin pour la méthode de dichotomie. Les itérations de cet algorithme sont faites sur plusieurs noeuds en même temps. On pose impose un nombre maximal d'itérations (= 10). Si la convergence ne s'obtient pas au bout de 5 itérations, la valeur initiale est considérée pas assez bonne.

IV) **Newinit**

Permet de déterminer un vecteur de nouvelles valeurs initiales qu'on nomme tk0 pour la méthode de Newton quand des noeuds divergent.
On commence par trouver les pôles proches de la frontière. S'il n'y en a pas, alors tk0 est un vecteur vide.

Ensuite on estime le maximum $F'(\theta)$, et on utilise "psort" afin de garder la multiplicité de ces estimations tout en enlevant les doublons, on renverse l'ordre des multiplicité. On utilise "ceval" pour calculer $F'(\theta)$ on l'appellera dft, on crée un vecteur contenant les dérivées de f notons le ds .
Pour chaque estimation on crée le plus petit intervalle possible le contenant en séparant dans deux vecteurs la position des noeuds x_i qui sont plus petits, et plus grands que cet estimateur.

V) **Bound**

Permet d'améliorer les limites supérieures et inférieures pour les noeuds qui n'ont pas convergé en se basant sur ceux qui l'ont fait. Elle sert à optimiser l'intervalle donné par "Newton".
Au vu de la monotonie de F(θ), on peut utiliser les noeuds qu'on a trouvé pour borner ceux qu'il nous reste à découvrir.

VI) **Bisect**

Cette fonction est utilisée comme dernier recours si toutes les valeurs initiales ont été utilisée (par "Newton") sans trouver tous les noeuds. On commence par différencier les noeuds qui sont assez précis de ceux qui ne le sont pas.
On applique la méthode de dichotomie pour un nombre maximal d'itérations égal à 52 (ce qui garantie la convergence).
Les itérations commencent dans un intervalle donné par la fonction "Newton" et amélioré avec la fonction "bound". En temps normal, cette fonction ne serait pas nécessaire, mais il est inévitable de l'utiliser s'il y'a plusieurs noeuds proches de l'intervalle.

VII) **Weight**

Il s'agit là de la dernière étape de l'algorithme, celle de calculer les poids de quadrature. On utilise une formule basée sur l'égalité :
$\lambda_{n,k} = 2\pi \dfrac{1-(1-d)x_{n,k}^{j-1}}{j+g_n(x_{n,k})},\ k = 1, 2, \ldots, n$ et on se base sur un paramètre t qui détermine l'efficacité et donc la formule à choisir en fonction de la taille du vecteur des pôles et du nombre de colonnes de la matrice d'entrée Beta.

Chapitre 3

Application numérique

On considère des intégrales de la forme suivant

$$I_j(f) = \int_I f(x)\omega_j(x)dx$$

avec j qui prend comme valeurs $\{1, 2, 3\}$ et $\omega_j(x)$ comme dans la table [1.1].
Soient

$$I_{j,n}(f) = \sum_{k=1}^{n} \lambda_{n,k}^{(j)} f(x_{n,k}^{(j)})$$

les approximations de $I_j(f)$ où $x_{n,k}^{(j)}$ et $\lambda_{n,k}^{(j)}$ sont calculés par l'algorithme RCHEB. On a par exemple

$$I_3(f) = \int_I f(x)\sqrt{1-x^2}(x)dx$$

et

$$I_{3,n} = \sum_{k=1}^{n} \lambda_{n,k}^{(3)} f(x_{n,k}^{(3)})$$

Rappelons que les approximations sont exactes pour d'après le théorème 1.1
On définit l'erreur relative par

$$\Delta_{j,f} = \frac{|I_j(f) - I_{j,n}(f)|}{|I_j(f)|}$$

On peut prendre par exemple la fonction f définie par $f(x) = \sin \dfrac{1}{x^2 + w^2}$ avec $w \in \overline{\mathbb{R}}\backslash\{0\}$ et on choisit arbitrairement $\omega = 0.03$.
Grâce à MAPLE on calcule la solution exacte $I_3(f)$ et obtient avec une très grande précision 40 chiffre après la virgule

$$I_3(f) = 0.2699968183335572698879103691877252310478 0$$

Calculons l'erreur commise par l'algorithme RCHEB (on utilise l'implémentation fournie en MATLAB dû à des problèmes sur l'implémentation FORTRAN).
Pour notre fonction f(x), les pôles α_k sont données par

$$\begin{aligned} \alpha_k &= \mathrm{i}\omega && \text{si k est impair} \\ \alpha_k &= -\mathrm{i}\omega && \text{si k est pair} \end{aligned}$$

On obtient le tableau suivant

n	$\Delta_{j,f}$
101	0.3311268
301	0.09186472
701	1.178409e-05

Conclusion

Le travail que nous avons réalisé s'est structuré en deux parties :
Nous nous sommes tout d'abord intéressés à la théorie des fonctions rationnelles orthogonales grâce au livre fourni par notre professeur encadrant, puis nous nous sommes attelés à la mise en place de l'algorithme Rcheb en FORTRAN.
Cette dernière partie fut la plus difficile au vu de l'absence de certaines fonctions intrinsèques Matlab dans le langage FORTRAN.
De ce fait, nous avons passé la majeure partie de notre temps à essayer de régler les erreurs rencontrées.
Ce TER a été une expérience enrichissante tant sur le niveau pédagogique que sur le plan personnel.
Nous tenons à remercier particulièrement M.DECKERS pour sa patience et sa confiance, d'autant plus qu'il s'agissait d'étudier un domaine sur lequel il a personnellement travaillé.

Bibliographie

[1] K.DECKERS, *Orthogonal Rational Functions :Quadrature, reccurence and rational krylov*

[2] `https://www.nsc.liu.se/~boein/f77to90/a5.html`

[3] `https://gcc.gnu.org/onlinedocs/gfortran/Intrinsic-Procedures.html`

[4] `https://people.sc.fsu.edu/~jburkardt/f_src/pchip/pchip.html`

Annexe

Ceci est une partie de notre code FORTRAN :

```
MODULE moduleTER

  INTERFACE repemat !interface
    MODULE PROCEDURE repemat_c
    MODULE PROCEDURE repemat_r
    MODULE PROCEDURE repemat_i
  END INTERFACE repemat

  INTERFACE diff !interface
    MODULE PROCEDURE diff_c
    MODULE PROCEDURE diff_r
  END INTERFACE diff

  INTERFACE tri !interface
    MODULE PROCEDURE tri_c
    MODULE PROCEDURE tri_i
  END INTERFACE tri


CONTAINS

include "pchip.f90"


  SUBROUTINE RCHEB(xi,lambda,err,alphacomplex,w)
    !! subroutine permettant de calculer les noeuds xi , les poids lambda ainsi que l'erreur c
    !sur ces poids
    IMPLICIT none
        !déclaration des types

    double complex ,dimension(:),allocatable,intent(in):: alphacomplex
    double complex ,dimension(:),allocatable:: alphac,b,a
    double precision,dimension(:),allocatable :: alphare, alphaima
    double complex :: an
    double precision :: bn
    INTEGER,   intent(in) ::w
    DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE , intent(out) :: xi
    DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE, intent(out) :: lambda
    DOUBLE PRECISION,dimension(:),allocatable, intent(out) :: err
    integer, DIMENSION(:),allocatable::z
```

```
    DOUBLE PRECISION :: pinf, ptol, xinf,xtol
    double complex,DIMENSION(:,:),allocatable ::abis,abis2
    DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE:: r,phi,v,ttk,ttk1,vide,f,f1
    double COMPLEX,DIMENSION(:,:),ALLOCATABLE :: beta1,beta,beta2
    DOUBLE PRECISION :: infini,pi
    DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: bnd,bnd1,bnd2,bnd3,tk0
    INTEGER, DIMENSION(:,:), ALLOCATABLE :: k1,k2,k3,ktest
    DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: tk,tk1 ,xis,x
    integer ,dimension(:),allocatable ::  d,k,s ,vectk1,vectk2,vectk3

    double precision,dimension(:,:),allocatable :: tauk0,tauk1,tauk2

    INTEGER :: n,m,taillebeta,j,l,i

    pi=4*datan(1d0)
    infini=exp(dble(709))
    allocate(alphare(size(alphacomplex,1)))
    allocate(alphaima(size(alphacomplex,1)))
    alphare=real(alphacomplex)
    alphaima=aimag(alphacomplex)
    !on définit plusieurs constantes
    pinf = 5/epsilon(dble(0))
    ptol = 5*epsilon(dble(0))
    xtol = 50*epsilon(dble(0))

    n = size(alphacomplex) ! on prend la taillle de notre vecteur complex
    !on vérifie que les pôles ne sont pas trop proches de l'intervalle [-1,1]
    ! car sinon les points xi seront proches de cet invervalle
    if ( any((abs(alphaima) <= ptol) .AND. (abs(alphare) <= 1 + ptol))) then
        print*,"les␣pôles␣sont␣trop␣proches␣de␣l'intervalle␣[-1,1]."
        stop
    end if

    ! on vérifie que w est bien 1 ,2 ou 3 sinon cela ne correspond pas à aux fonctions de pond
    !   de Gauss-Tchebychev
    if (w==1 .OR. w==2 .OR. w==3) then
      allocate(z(2)) ! on crée un vecteur z, dépendant de de w
      z=(/floor(w/2.)-1,w-1/)
    else
      print*, "le␣poids␣est␣mauvais"
      return
    end  if

    allocate(xi(n)) ! on alloue de taille n
    allocate(lambda(n))
    xi=0 ! on initialise xi et lambda à 0
    lambda=xi
    allocate(alphac(n))
    alphac=alphacomplex

    !si le module des composantes de alpha sont supérieur à une certaine valeur, alors on les
à + l'infini
    do l=1,size(alphacomplex,1)
```

```
    if (abs(alphac(l))>=pinf) then
      alphac(l)=infini
    end if
  end do


  an=alphac(n) ! on extrait la dernière composante de alpha
  if ((isnan(real(an)) .EQV. .TRUE.) .and.(isnan(aimag(an)) .EQV. .TRUE.)) then
    bn=0
  else
    if (real(an)>=0) then
      bn = real(1 / (an+(2*1) * sqrt(an*an-1)));
    else
      bn = real(1 / (an+(1) * sqrt(an*an-1)));
    END IF
    if (abs(bn)<=(1/(2*pinf))) then
    bn = 0;
    alphac(n)= infini
    else
      alphac(n) = (bn+1/bn)/2;
    end if
  end if

  !on crée notre vecteur erreur de taille n
  allocate(err(n))

  if (n == 1) then     !on est dans le cas explicite
    xi = (bn+z(2) - 2*z(1)) / (1+z(1))
    lambda = pi /(1+z(2)-z(1))
    err=epsilon(dble(0))
  end if

  allocate(a(2*n-1))
  a=(/alphac, alphac(1:n-1)/)
  allocate(abis(2,size(a)))
  abis(1,:)=a
  abis(2,:)=0
 call psort(abis, ptol,abis2) !! on appelle psort afin de déterminer et de trier les pôles
  m = size(abis2,2) ! on prend le nombre de colonnes
  !m=1

  allocate(v(n))
  allocate(ttk(n))
  v=(/(n-j,j=1,n)/) ! on crée un vecteur de taille n , il va de n à 1
  ttk = pi *(v-(1-z(1))/2) / (n+z(2)/2)! on crée les points pour pchip

  allocate(beta(2,size(abis2,2)))
  if (m==1) then ! si les pôles sont taux égaux
    if (bn==0) then  ! si les pôles valent tous  l'infini
      ! on est dans le cas polynomial
      xi=cos(ttk)
      lambda = 2*pi*(1-z(1)*xi**z(2))/(2*n+z(1));
      err=epsilon(dble(0))*(/(1,j=1,n)/)
```

```
      return
    end if
    beta(1,:)=bn
    beta(2,:)=cmplx(n)
  else ! les pôles ne sont pas tous égaux
    ! on extrait la position des coordonnées de abis qui sont plus grande que 0
     call vectsupx((real(abis2(1,:))),d,dble(0))
     do i=1, size(beta,2)
        if (2*real(abis2(1,i))>=0) then
          beta(1,i)=1/a(i)*(2*real(abis2(1,i))*sqrt((abis2(1,i)**2)-1))
        else
          beta(1,i)=1/a(i)
        end if
     end do
    if (isnan(real(abis2(1,n))) .and. isnan(aimag(abis2(1,n)))) THEN
      beta(size(beta,1),size(beta,2))=0
    end if
      allocate(beta1(3,size(beta,2)))
      beta1(1:2,:)=beta(:,:)
      beta1(3,:)=abis2(2,:)/2

  end if

  allocate(r(size(beta,2)))
  allocate(phi(size(beta,2)))
  allocate(b(size(beta,2)))

  !on définit le module, l'angle de beta ainsi que b, qui seront utile plus tard
  r = abs(beta(1,:))
  phi=angle(beta(1,:))
  b = (1-r)* exp(CMPLX(0,1)*phi)
  allocate(beta2(5,size(beta,2)))
  beta2(1:2,:)=beta(:,:)
  beta2(3,:)=dcmplx(r)
  beta2(4,:)=dcmplx(phi)
  beta2(5,:)=b


  ALLOCATE(k(0))!on crée un vecteur "vide"

  !on appelle ceval afin d'évaluer F aux points ttk afin de construire et d'évaluer pchi
  !pour obtenir des valeurs initiales pour nos noeuds
  call ceval(ttk,n,beta2,z,k,(/dble(0),dble(0)/),f)
  ttk=(/pi,ttk,dble(0)/)
  f1=(/(dble(n)+z(2))/2*pi,f,dble(0)/)
  allocate(tk0(n,1))
 call pchci(n,real(f1),real(ttk),real(tk0),0)


  vide=(/dble(0),pi/)
 !on utilise la méthode de newton dans le  but de calculer les noeuds
 call repemat( vide,bnd,n)
 deallocate(vide)
```

```
 allocate(ktest(1,n))
 ktest(1,:)=(/(n-j+1,j=1,n) /)
 call newton(tk0,bnd,n,beta2,z,ktest,xtol,tk,k1,bnd1)

 ! on trouve des valeurs initiales supplémentaires s'il y a divergence
if (size(k1)>0) then
  call newinit(n,beta2,z,k1,ttk,f,ptol,tk0) ! on cherche de nouvelles valeurs initiales
  if (size(tk0,1)>0) then
    !on cherche de nouveau un ensemble valeurs initiales
    allocate(tauk0(1,size(tk0,2)))
    tauk0(1,:)= tk0(1,:)
    vectk1=pack( k1,mask=k1==k1 )
    vectk2=pack( k2,mask=k2==k2 )
     call newton(tauk0,bnd1,n,beta2,z,k1,xtol,xis,k2,bnd2 )
     tk(n-vectk1+1)=xis
      if (size(k2)>0) then
        if (size(tk0,1) == 2) then
          ! on crée le nouvel ensemble et on cherche
          !la position des noeuds qui ne convergent pas encore
          i = 1
          j = 1
          allocate(s(size(k2)))
          s = (/(0,j=1,size(k2)) /)
          do while (i <= size(k2))
          if (vectk1(j) == vectk2(i)) then
                              s(i) = j
                              i = i + 1
          end if
                      j = j + 1
          end do
          allocate(tauk0(1,size(tk0(2,s))))
          tauk2(1,:)= tk0(2,s)
          deallocate(xis)
          call newton(tauk2,bnd2,n,beta2,z,k2,xtol,xis,k3,bnd3)
          tk(n-vectk2+1)=xis
        if (size(k3,1)>0) then ! cela diverge toujours malgré les tentatives
          ! on utilise la méthode de dichotomie
                      call bound(bnd3,n,tk,k3,pi)
          call bisect(bnd3,n,beta2,z,k3,xtol,x)
          tk(n+1-vectk3)=x
        end if
            else
                    call bound(bnd2,n,tk,k2,pi)
        call bisect(bnd2,n,beta2,z,k2,xtol,x)
        tk(n+1-vectk2)=x
      end if
          end if
  else ! si de nouvelles valeurs ne peuvent pas être touvé , on utilise la méthode de dichot
          call bound(bnd1,n,tk,k1,pi)
    call bisect(bnd1,n,beta2,z,k1,xtol,x)
    tk(n+1-vectk1)=x
  end if
end if
```

```
 xi = cos(tk) ! on calcule nos noeuds(zeros de tk)
 call poids(lambda,xi,tk,n,beta2,z,pi) ! on calcule le poids
 deallocate(d)
 allocate(d(n))
 d=(/(n-j+1,j=1,n) /)

 call ceval(tk,n,beta2,z,d,(/dble(1),dble(1)/),err)! on estime l'erreur comise

end SUBROUTINE RCHEB
 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

subroutine ceval(theta,n,beta,c,k,e,f)!subroutine permettant de calculer f et sa dérivée en th
!on evalue la fonction F en theta si e=[0,0]
!on calcule
 IMPLICIT NONE
!déclaration des variables
 INTEGER :: n,m,n2,t,j
 integer,dimension(:),allocatable::k,c

 DOUBLE precision, DIMENSION(:), ALLOCATABLE :: mb,phi,phitest,mbtest,vectphin,vectmbn
 double complex,dimension(:),allocatable:: b,btest,vectbn,vectbetan,vectbetatest
 DOUBLE PRECISION, PARAMETER :: pi = 4 * atan (1.0_16)
 DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: ft,ft1,fdt,df,theta,thetatest,vectthetam1
 double precision,dimension(2),intent(in):: e
 DOUBLE complex, DIMENSION(:,:), ALLOCATABLE :: beta, betan,bn
 DOUBLE precision, DIMENSION(:,:), ALLOCATABLE ::thetam,thetam1, phin,mbn
 double precision,allocatable,dimension(:),intent(out):: f
 double precision :: fd1,f1,c1,c2


 m = size(beta,2)
 allocate(mb(m))
 allocate(phi(m))
 mb(:) = dble(beta(2,:))
 phi(:) = dble(beta(4,:))
 allocate(b(m))
 b(:) = beta(5,:)
 n2 = size(theta) ! nombre de points à évaluer

 t=0 !
 allocate(ft(n2))
 allocate(df(n2))
 if ((m > n2).OR.(10 * m > (n2 + 780))) then
  t=t+1
 end if
 if (((m <= 108).AND.(n2 <= 300).AND.(30 * m > (n2 + 120)).AND.(10 * n2 > (m + 36)))) then
  t=t+2
 end if
 if ((e(1) == 1) .OR. (e(2) == 0)) then
      c2=c(2)
```

```
f = -(n - 1 - c2/2) * theta ! dépend de la fonction poids
if (e(1) == 1) then
   c1=c(1)
   f = f - (pi * (k - (1 - c1)/2))
END IF
  if (m == 1) then  ! on a un seul pôles
     f = f + (2 * n - 1) * atan2(sin(theta),cos(theta)-dble(beta(1,1)))
  else if (n2 == 1) then
   allocate(thetatest(1))
   thetatest=theta(1)
   call ftheta(thetatest,phi,b,mb,pi,ft)
     f = f + ft
  else
   ! plusieurs choix selon la valeur de t
   SELECT CASE (t)
         CASE (0)
             DO j = 1,n2
               allocate(thetatest(1))
                 thetatest=theta(j)
                 call ftheta(thetatest,phi,b,mb,pi,ft1)
                 f1=ft(1)
                 f(j) = f(j) + f1
               deallocate(thetatest)
             END DO
         CASE (1)
             DO j = m,1,-1
               allocate(phitest(1))
               phitest=phi(j)
               allocate(btest(1))
               btest=b(j)
               allocate(mbtest(1))
               mbtest=mb(j)
               call ftheta(theta,phitest,btest,mbtest,pi,ft1)
               f1=ft1(1)
               f = f + f1
               deallocate(phitest)
               deallocate(btest)
               deallocate(mbtest)
             END DO
         CASE DEFAULT
              call repemat(theta,thetam,m)
              allocate(thetam1(size(thetam,2),size(thetam,1)))
              thetam1=transpose(thetam)
               deallocate(thetam)
              call repemat(b,bn,n2)
              call repemat(phi,phin,n2)
              call repemat(mb,mbn,n2)

              allocate(vectthetam1(size(thetam1)))
              allocate(vectbn(size(bn)))
              allocate(vectphin(size(phin)))
              allocate(vectmbn(size(mbn)))
              vectthetam1=pack( thetam1,mask=thetam1==thetam1 )
```

```
                  vectbn=pack( bn,mask=bn==bn )
                  vectphin=pack( phin,mask=phin==phin )
                  vectmbn=pack( mbn,mask=mbn==mbn )
                  call ftheta(vectthetam1,vectphin,vectbn,vectmbn,pi,ft1)
                 f = f + ft1
       END SELECT
    END IF
ENDIF

if (e(2) == 1) then ! on a aussi besoin de la dérivé
   df = -(n - 1 - c(2)/2) * (/(1,j=1,n2)/)
   if (m == 1) then ! les pôles sont tous égaux
       df = df + (2 * n - 1) * (1 - beta(1,1) * cos(theta)) &
           & /(1 - 2 * beta(1,1) * cos(theta) + beta(1,1)**2)
   else if (n2 == 1) then ! le nombre de zeros à calculer vaut 1
     call fdtheta(theta,beta(1,:),mb,fdt)
       df=df+fdt
   else
     select case (t)
           case (0)
               do j = 1,n2
                 allocate(thetatest(1))
                     thetatest=theta(j)
                 call fdtheta(thetatest,beta(1,:),mb,fdt)
                 fd1=fdt(1)
                 df=df+fd1
                 deallocate(thetatest)
               end  do
           case (1)
               do j = m,1,-1
                 allocate(vectbetatest(1))
                 vectbetatest=beta(1,j)
                 allocate(mbtest(1))
                 mbtest=mb(j)
                 call fdtheta(theta,vectbetatest,mbtest,fdt)
                 fd1=fdt(1)
                 df=df+fd1
                 deallocate(vectbetatest)
                 deallocate(mbtest)
                  end do
             case DEFAULT
               if (e(1)==0) then
                   call repemat(theta,thetam,m)
                   call repemat(mb,mbn,n2)
               end if
               vectbn=beta(1,:)
               call repemat(vectbn,betan,n2)
               deallocate(vectbn)
               allocate(vectthetam1(size(thetam)))
               allocate(vectbetan(size(betan)))
               allocate(vectmbn(size(mbn)))
               vectthetam1=pack( thetam,mask=thetam==thetam )
               vectbetan=pack( betan,mask=betan==betan )
```

```
                vectmbn=pack( mbn,mask=mbn==mbn )
                call fdtheta(vectthetam1,vectbetan,vectmbn,fdt)
                df = df + fdt
      end select
    end if

    if (e(1) == 1) then
        f = f /df
    else
        f = df
    end if
  end if


end subroutine ceval
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```