Algorithmes de recalage

Résumé :

Dans ce document on présente les algorithmes de recalage implémentés dans MACR_RECAL. Il s'agit d'abord d'un algorithme de Levenberg-Marquardt avec bornes et ensuite d'un algorithme évolutionnaire.

Pour le premier on décrit dans un premier temps la méthode générale avant d'en préciser certains éléments. Sont détaillés le calcul de la fonctionnelle, de la matrice Jacobienne, la détermination du paramètre initial de régularisation ainsi que son évolution, la gestion des bornes et le critère de convergence.

Les principes généraux des algorithmes génétiques, et en particulier l'algorithme évolutionnaire, ainsi que la mise en œuvre dans *Code_Aster*, sont présentés dans la deuxième partie du document.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Table des Matières

1Introduction....3
2Algorithme de Levenberg-Marquardt....4
2.1Position du problème....4
2.2Résolution....4
2.3Mise en œuvre pratique....7
2.3.1Définition de la fonctionnelle....7
2.3.2Expression de la matrice Jacobienne....8
2.3.3Régularisation du système linéaire....9
2.3.3.1Valeur initiale de....9
2.3.3.2Évolution de la valeur de....9
2.3.4Limitations du domaine d'évolution des paramètres....9
2.3.5Adimensionnement....10
2.3.6Critère de convergence....11
2.4Algorithme global....12
3Algorithme évolutionnaire....12
3.1Principes générales....12
3.2Fonctionnement de l'algorithme....13
3.3Technique hybride de recalage....14
4Bibliographie....14
5Description des versions du document....15

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

1 Introduction

Avant d'aborder à proprement parler la problématique du recalage, il est utile de rappeler quelques éléments sur l'identification de paramètres. Supposons que l'on désire identifier n paramètres à partir d'un essai mécanique donné. Dans le cadre de cette identification, on définit les grandeurs :

- $\mathbf{c}$, le vecteur des n paramètres à identifier, appartenant à $\mathbf{O}$, convexe fermé de $\mathbb{R}^n$.
- $\mathbf{d}$, le vecteur des grandeurs calculées lors d'une simulation de l'essai en utilisant les paramètres $\mathbf{c}$, par opposition à $\mathbf{d}^{\exp}$, le vecteur des grandeurs mesurées lors d'un essai expérimental. Tous deux appartiennent à l'espace $\mathbf{L}$ des grandeurs observables. La simulation de l'essai expérimental, paramétrée par le vecteur $\mathbf{c}$, peut être réalisée par différentes méthodes : différences finies, éléments finis, éléments de frontière, ... C'est ce que nous appellerons le problème direct.

Le but de l'identification est de déterminer le jeu de paramètres $\mathbf{c}$ réduisant la différence entre grandeurs mesurées et expérimentales (en espérant fortement que la réduction de cet écart soit suffisante pour obtenir le jeu de paramètres souhaité ...). On introduit donc une fonctionnelle coût notée $\mathbf{J}$ dépendant de $\mathbf{c}$ et mesurant la distance entre $\mathbf{d}$ et $\mathbf{d}^{\exp}$.

$$\mathbf{J}(\mathbf{c}) = \|\mathbf{d} - \mathbf{d}^{\mathbf{exp}}\| \qquad \textbf{éq 1-1}$$

où $\|.\|$ désigne une norme sur $\mathbf{L}$.

L'identification s'exprime donc sous la forme du problème de minimisation suivant :

$$\text{Déterminer } \mathbf{c}^* \in \mathbf{O} \text{ tel que } \mathbf{J}(\mathbf{c}^*) = \underset{\mathbf{c}\in\mathbf{O}}{\text{Min}}\ \mathbf{J}(\mathbf{c})$$

Enfin, on définit le recalage comme la minimisation d'un type de fonctionnelles particulier dites « moindres carrés » qui s'expriment sous la forme :

$$\mathbf{J}(\mathbf{c}) = \sum_{n=1}^{N} j_n^2(\mathbf{c}) \qquad \textbf{éq 1-2}$$

ou $j_n(\mathbf{c})$ représente la composante n de l'écart entre le vecteur des grandeurs calculées et expérimentales.
Il est communément admis que parmi les algorithme de minimisation déterministes, le plus efficace pour ce type de fonctionnelles est l'algorithme de Levenberg-Marquardt. C'est ce dernier qui a été historiquement le premier implanté dans la commande MACR_RECAL de *Code_Aster* et que nous présentons dans la suite.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

2 Algorithme de Levenberg-Marquardt

2.1 Position du problème

Il existe plusieurs familles d'algorithmes de minimisation [bib1]. Pour les problèmes relativement réguliers, les plus utilisées sont les méthodes de descente. Leur principe est de générer de manière itérative une suite $\left(\mathbf{c}^{\mathbf{k}}\right)_{k\in N}$ définie par :

$$\left(\mathbf{c}^{\mathbf{k+1}}\right)=\mathbf{c}^{\mathbf{k}}+\alpha^{k}\ \mathbf{g}^{\mathbf{k}} \qquad \textbf{éq 2.1-1}$$

telle que, pour $f(x)=\mathbf{J}\left(\mathbf{c}^{\mathbf{k}}+x\ \mathbf{g}^{\mathbf{k}}\right),\ x\in R_{+}^{*}$

- $f(x)$ est décroissante au voisinage de 0^{+}
- $f\left(\alpha^{k}\right)=\underset{x>0}{\mathrm{Min}}\, f(x)$

$\mathbf{g}^{\mathbf{k}}$ est la direction de descente au pas k . C'est la méthode de détermination de $\mathbf{g}^{\mathbf{k}}$ qui conditionne la nature donc l'efficacité de l'algorithme utilisé, sachant que ces techniques sont principalement basées sur des approximations de $\mathbf{J}$ à l'ordre 1 ou à l'ordre 2. Pour l'algorithme de Levenberg-Marquardt, on manipule une approximation à l'ordre 2 de la fonctionnelle.

2.2 Résolution

Dans le cadre du recalage, on manipule des fonctionnelles coût moindre carré du type :

$$\mathbf{J}(\mathbf{c})=\sum_{n=1}^{N} j_{n}^{2}(\mathbf{c}) \qquad \textbf{éq 2.2-1}$$

où par exemple $j_{n}(\mathbf{c})=\left(F_{n}^{calc}(\mathbf{c})-F_{n}^{\exp}\right)$, avec des notations évidentes.

La particularité de ces fonctionnelles coût réside dans le fait que l'on connaît la forme de leurs dérivées premières et secondes :

$$\left(\nabla_{c}\mathbf{J}(\mathbf{c})\right)_{i}=2\sum_{n=1}^{N} j_{n}(\mathbf{c})\frac{\partial j_{n}}{\partial c_{i}} \qquad \textbf{éq 2.2-2}$$

$$\left(\mathbf{H}(\mathbf{c})\right)_{ij}=2\sum_{n=1}^{N}\left(\frac{\partial j_{n}}{\partial c_{i}}\cdot\frac{\partial j_{n}}{\partial c_{j}}+j_{n}(\mathbf{c})\frac{\partial^{2} j_{n}}{\partial c_{i}\partial c_{j}}\right) \qquad \textbf{éq 2.2-3}$$

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Alors, en supposant que le deuxième terme de l'équation précédente est négligeable devant le premier (ce qui est vrai quand les j_k sont linéaires en $\mathbf{c}$: ce terme est nul), on peut réécrire :

$$\left(\mathbf{H}(\mathbf{c})\right)_{ij} \approx 2\sum_{n=1}^{N} \frac{\partial j_n}{\partial c_i} . \frac{\partial j_n}{\partial c_j} \qquad \textbf{éq 2.2-4}$$

Il est intéressant à ce niveau d'introduire la matrice de sensibilité ou matrice Jacobienne définie par :

$$\mathbf{A} = \begin{bmatrix} \frac{\partial j_1}{\partial c_1} & \frac{\partial j_1}{\partial c_2} & \cdots & \frac{\partial j_1}{\partial c_n} \\ \frac{\partial j_2}{\partial c_1} & \frac{\partial j_2}{\partial c_2} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial j_N}{\partial c_1} & \frac{\partial j_N}{\partial c_2} & \cdots & \frac{\partial j_N}{\partial c_n} \end{bmatrix} \qquad \textbf{éq 2.2-5}$$

On peut ainsi exprimer le gradient et le Hessien par :

$$\nabla_c \mathbf{J}\left(\mathbf{c}^{\mathbf{k}}\right) = 2\,\mathbf{A}^{\mathbf{T}}\,\mathbf{j} \qquad \textbf{éq 2.2-6}$$

$$\mathbf{H}\left(\mathbf{c}^{\mathbf{k}}\right) \approx 2\,\mathbf{A}^{\mathbf{T}}\mathbf{A} \qquad \textbf{éq 2.2-7}$$

avec $\mathbf{j} = \left[j_1, \ldots, j_N\right]^T$.

Écrivons alors le développement limité à l'ordre 2 de $\mathbf{J}$:

$$\mathbf{J}(\mathbf{c}) \approx \mathbf{J}\left(\mathbf{c}^{\mathbf{k}}\right) + \left(\mathbf{c}-\mathbf{c}^{\mathbf{k}}\right)^T . \nabla_c \mathbf{J}\left(\mathbf{c}^{\mathbf{k}}\right) + \frac{1}{2}\left(\mathbf{c}-\mathbf{c}^{\mathbf{k}}\right)^T \mathbf{H}\left(\mathbf{c}^{\mathbf{k}}\right).\left(\mathbf{c}-\mathbf{c}^{\mathbf{k}}\right) \qquad \textbf{éq 2.2-8}$$

Soit $\mathbf{g}^{\mathbf{k}} = \mathbf{c} - \mathbf{c}^{\mathbf{k}}$, le pas de descente au point $\mathbf{c}^{\mathbf{k}}$, il doit vérifier la condition de stationnarité de l'approximation quadratique :

$$\nabla_c \mathbf{J}\left(\mathbf{c}^{\mathbf{k}}\right) + \mathbf{H}\left(\mathbf{c}^{\mathbf{k}}\right)\,\mathbf{g}^{\mathbf{k}} = 0 \qquad \textbf{éq 2.2-9}$$

D'après l'expression du gradient et du Hessien de $\mathbf{J}$, on peut écrire :

$$\left(\mathbf{A}^{\mathbf{T}}\mathbf{A}\right)\,\mathbf{g}^{\mathbf{k}} = -\mathbf{A}^{\mathbf{T}}\,\mathbf{j} \qquad \textbf{éq 2.2-10}$$

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

La résolution de cette équation mène à un algorithme connu sous le nom de Gauss-Newton, très efficace mais qui présente néanmoins quelques inconvénients :

- $\left(\mathbf{A}^{\mathbf{T}}\mathbf{A}\right)$ peut être quasiment singulière et causer la non-existence de solution.
- On n'a aucun contrôle sur $\mathbf{g}^{\mathbf{k}}$, qui peut être trop grand et donc sortir les paramètres de l'espace admissible.

Pour pallier ces inconvénients, on préfère utiliser l'algorithme de Levenberg-Marquardt qui propose une régularisation de l'algorithme de Gauss-Newton :

$$\left(\mathbf{A}^{\mathbf{T}}\mathbf{A}+\lambda\ \mathbf{I}\right)\ \mathbf{g}^{\mathbf{k}}=-\mathbf{A}^{\mathbf{T}}\ \mathbf{j} \qquad \textbf{éq 2.2-11}$$

où λ est un scalaire et $\mathbf{I}$ la matrice identité.

On remarque que si $\lambda=0$, on retrouve la direction donnée par Gauss-Newton et si $\lambda\to+\infty$, on retrouve la direction donnée par l'opposée du gradient de $\mathbf{J}$ *i.e.* la plus grande pente.

L'algorithme de Levenberg-Marquardt consiste donc, en partant d'une valeur de λ « assez élevée », de la diminuer d'un facteur 10 par exemple, à chaque décroissance de $\mathbf{J}$. On passe ainsi graduellement d'un algorithme de plus grande pente à l'algorithme de Gauss-Newton. On peut donc présenter cette procédure sous la forme :

- Choix d'un point de départ $\mathbf{c}^{\mathbf{0}}$ et d'une valeur initiale de λ
- A l'itération k, résoudre
  $\left(\mathbf{A}^{\mathbf{T}}\mathbf{A}+\lambda\ \mathbf{I}\right)\ \mathbf{g}^{\mathbf{k}}=-\mathbf{A}^{\mathbf{T}}\mathbf{j}$
  $\mathbf{c}^{\mathbf{k+1}}=\mathbf{c}^{\mathbf{k}}+\mathbf{g}^{\mathbf{k}}$
- Si $\mathbf{J}\left(\mathbf{c}^{\mathbf{k+1}}\right)<\mathbf{J}\left(\mathbf{c}^{\mathbf{k}}\right)$, alors $\lambda=\lambda/10$ sinon $\lambda=\lambda*10$
- Test de convergence

Remarque :

Nous avons considéré ci-dessus l'algorithme de Levenberg-Marquardt sous l'angle de la régularisation de l'algorithme de Gauss-Newton. Il est possible de donner un éclairage différent à cette algorithme en le considérant comme un algorithme de région de confiance [bib 2]. En effet, on peut montrer aisément que le système [éq 2.2-11] est la condition de stationnarité du problème de minimisation :

Déterminer $\mathbf{g}^{\mathbf{k}}$ *tel que* $\mathbf{g}^{\mathbf{k}}=\mathrm{ArgMin}\left(\mathbf{g}^{\mathbf{k}\,\mathbf{T}}.\mathbf{A}^{\mathbf{T}}\mathbf{j}+\frac{1}{2}\mathbf{g}^{\mathbf{k}\,\mathbf{T}}\mathbf{A}^{\mathbf{T}}\mathbf{A}\ \mathbf{g}^{\mathbf{k}}\right)$ *soumis à* $\|\mathbf{g}^{\mathbf{k}}\|\le D^{k}$

Où $D^{k}=-\left(\mathbf{A}^{\mathbf{T}}\ \mathbf{A}+\lambda\ \mathbf{I}\right)^{-1}\mathbf{A}^{\mathbf{T}}\mathbf{j}$ **et** $\lambda\ge 0$.

Il s'agit là d'une implantation très simple de l'algorithme de Levenberg-Marquardt au sein de laquelle différentes questions ne sont pas abordées :

- Comment définir la fonctionnelle $\mathbf{J}$ quand on dispose de plusieurs essais ?
- Comment choisir la valeur initiale de λ ?
- Comment faire évoluer λ de manière plus fine ?
- Comment définir le domaine d'évolutions de chaque paramètre ?

Nous allons préciser ces différents points dans la suite.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

2.3 Mise en œuvre pratique

2.3.1 Définition de la fonctionnelle

Lors d'un recalage, l'utilisateur dispose souvent de plusieurs mesures différentes ; il s'agit de grandeurs physiques discrètes, éventuellement de natures différentes, mesurées au cours d'un ou plusieurs essais. Ce sont des fonctions d'un paramètre donné noté t (temps, abscisse, …) que l'on peut donc représenter par :

$$F_1^{\exp}=\begin{bmatrix} t_1 & f_1^{\exp}(t_1) \\ \vdots & \vdots \\ t_N & f_1^{\exp}(t_N) \end{bmatrix} \quad F_2^{\exp}=\begin{bmatrix} t_1 & f_2^{\exp}(t_1) \\ \vdots & \vdots \\ t_M & f_2^{\exp}(t_M) \end{bmatrix} \quad F_L^{\exp}=\begin{bmatrix} t_1 & f_L^{\exp}(t_1) \\ \vdots & \vdots \\ t_P & f_L^{\exp}(t_P) \end{bmatrix} \quad \textbf{éq 3.1-1}$$

Chacune de ces mesures expérimentales dispose de son pendant

$$F_1^{\text{calc}}(\mathbf{c^k})=\begin{bmatrix} \tilde{t}_1 & f_1^{\text{calc}}(\mathbf{c^k},\tilde{t}_1) \\ \vdots & \vdots \\ \tilde{t}_I & f_1^{\text{calc}}(\mathbf{c^k},\tilde{t}_I) \end{bmatrix} F_2^{\text{calc}}(c^k)=\begin{bmatrix} \tilde{t}_1 & f_2^{\text{calc}}(\mathbf{c^k},\tilde{t}_1) \\ \vdots & \vdots \\ \tilde{t}_J & f_2^{\text{calc}}(\mathbf{c^k},\tilde{t}_J) \end{bmatrix} F_L^{\text{calc}}(\mathbf{c^k})=\begin{bmatrix} \tilde{t}_1 & f_L^{\text{calc}}(\mathbf{c^k},\tilde{t}_1) \\ \vdots & \vdots \\ \tilde{t}_K & f_L^{\text{calc}}(\mathbf{c^k},\tilde{t}_K) \end{bmatrix}$$

éq 3.1-2

calculé pour un jeu de paramètre c^k donné. Remarquons que les grandeurs calculées ne sont pas forcément en même nombre que les grandeurs mesurées ni évaluées pour la même valeur du paramètre t . On peut alors définir la fonctionnelle moindres carrés à minimiser par :

$$\mathbf{J}(\mathbf{c^k})=\frac{\sum_{i=1}^{N}\left(\frac{f_1^{\exp}(t_i)-f_1^{\text{calc}}(\mathbf{c^k},t_i)}{f_1^{\exp}(t_i)}\right)^2+\sum_{i=1}^{M}\left(\frac{f_2^{\exp}(t_i)-f_2^{\text{calc}}(\mathbf{c^k},t_i)}{f_2^{\exp}(t_i)}\right)^2+\ldots+\sum_{i=1}^{P}\left(\frac{f_L^{\exp}(t_i)-f_L^{\text{calc}}(\mathbf{c^k},t_i)}{f_L^{\exp}(t_i)}\right)^2}{\mathbf{J}(\mathbf{c^0})}$$

éq 3.1-3

Il est important de remarquer que :

- si une mesure calculée f_j^{calc} n'est pas définie à un instant t_i , alors on interpole linéairement sa valeur
- si une mesure expérimentale $f_j^{\exp}$ est nulle, on ne divise pas la quantité $f_j^{\exp}(t_i)-f_j^{\text{calc}}(\mathbf{c^k},t_i)$ et elle est présente telle quelle dans l'expression de la fonctionnelle
- la fonctionnelle $\mathbf{J}$ est normalisée de manière à valoir 1. au début des itérations de recalage

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

2.3.2 Expression de la matrice Jacobienne

Pour le calcul de la matrice Jacobienne, on définit le vecteur $\mathbf{j}$ des erreurs par :

$$\mathbf{j}=\begin{bmatrix} \dfrac{f_1^{\exp}(t_1)-f_1^{\text{calc}}(\mathbf{c}^{\mathbf{k}},t_1)}{f_1^{\exp}(t_1)} \\ \vdots \\ \dfrac{f_1^{\exp}(t_N)-f_1^{\text{calc}}(\mathbf{c}^{\mathbf{k}},t_N)}{f_1^{\exp}(t_N)} \\ \dfrac{f_2^{\exp}(t_1)-f_2^{\text{calc}}(\mathbf{c}^{\mathbf{k}},t_1)}{f_2^{\exp}(t_1)} \\ \vdots \\ \vdots \\ \dfrac{f_P^{\exp}(t_L)-f_P^{\text{calc}}(\mathbf{c}^{\mathbf{k}},t_L)}{f_P^{\exp}(t_L)} \end{bmatrix} \quad \textbf{éq 3.2-1}$$

Soit :

$$j_i^K=\frac{f_K^{\exp}(t_i)-f_K^{\text{calc}}(\mathbf{c}^{\mathbf{k}},t_i)}{f_K^{\exp}(t_i)} \quad \textbf{éq 3.2-2}$$

On retrouve alors l'expression de la matrice Jacobienne de [éq 2.2-5] :

$$\mathbf{A}=\begin{bmatrix} \dfrac{\partial j_1^1}{\partial c_1} & \dfrac{\partial j_1^1}{\partial c_2} & \cdots & \dfrac{\partial j_1^1}{\partial c_n} \\ \cdots & \cdots & \cdots & \cdots \\ \dfrac{\partial j_N^1}{\partial c_1} & \dfrac{\partial j_N^1}{\partial c_2} & \cdots & \dfrac{\partial j_N^1}{\partial c_n} \\ \dfrac{\partial j_1^2}{\partial c_1} & \dfrac{\partial j_1^2}{\partial c_2} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \dfrac{\partial j_L^P}{\partial c_1} & \dfrac{\partial j_L^P}{\partial c_2} & \cdots & \dfrac{\partial j_L^P}{\partial c_n} \end{bmatrix} \quad \text{éq 3.2-3}$$

Où les termes sont calculés par différences finies directes :

$$\frac{\partial j_1^1}{\partial c_i}(c_1,\ldots,c_i,\ldots,c_n)\approx\frac{j_1^1(c_1,\ldots,c_i+\alpha c_i,\ldots,c_n)-j_1^1(c_1,\ldots,c_i,\ldots,c_n)}{\alpha c_i} \quad \textbf{éq 3.2-4}$$

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

2.3.3 Régularisation du système linéaire

Nous abordons ici le problème de la détermination et de l'évolution du paramètre de régularisation λ. On définit pour ce faire :

- $\lambda_{\max} = \mathrm{Max}(\text{Valeurs propres de } \mathbf{A}^{\mathbf{T}}\mathbf{A})$, $\lambda_{\min} = \mathrm{Min}(\text{Valeurs propres de } \mathbf{A}^{\mathbf{T}}\mathbf{A})$, $cond = \lambda_{max}/\lambda_{min}$ si $\lambda_{min} \neq 0$
- $\mathbf{Q}(\mathbf{c}) = \mathbf{J}(\mathbf{c}^{\mathbf{k}}) + (\mathbf{c}-\mathbf{c}^{\mathbf{k}})^{T}.\mathbf{A}^{\mathbf{T}}\mathbf{j} + \frac{1}{2}(\mathbf{c}-\mathbf{c}^{\mathbf{k}})^{T}.(\mathbf{A}^{\mathbf{T}}\mathbf{A} + \lambda\ \mathbf{I}).(\mathbf{c}-\mathbf{c}^{\mathbf{k}})$

2.3.3.1 Valeur initiale de λ

Connaissant les grandeurs ci-dessus, on définit l'algorithme suivant :

- Si $\lambda_{\min} = 0$, alors $\lambda = 1.\mathrm{E}{-3}\,\lambda_{\max}$
- Sinon
  - Si $cond < 1.\mathrm{E}5$, alors $\lambda = 1.\mathrm{E}{-16}\,\lambda_{\max}$
  - Sinon $\lambda = \left|(1.\mathrm{E}5\,\lambda_{\min} - \lambda_{\max})\right| / 10001$

Remarque :

Dans le dernier cas, la valeur attribuée à λ a pour effet de ramener le conditionnement de $A^{T}A$ à $1.\mathrm{E}5$

2.3.3.2 Évolution de la valeur de λ

Pour faire évoluer λ, on définit le ratio $R^{k} = \frac{\mathbf{J}(\mathbf{c}^{\mathbf{k}}) - \mathbf{J}(\mathbf{c}^{\mathbf{k+1}})}{\mathbf{Q}(\mathbf{c}^{\mathbf{k}}) - \mathbf{Q}(\mathbf{c}^{\mathbf{k+1}})}$, qui permet d'évaluer la validité de l'approximation quadratique de $\mathbf{J}$: plus il est proche de 1, plus cette approximation est valable. On en déduit l'algorithme suivant [bib2] :

- Si $R^{k} < 0.25$, alors $\lambda = \lambda \times 10$
- Si $R^{k} > 0.75$, alors $\lambda = \lambda / 15$

2.3.4 Limitations du domaine d'évolution des paramètres

Pour des raisons diverses telles que pour garantir la validité physique des paramètres (module d'Young strictement positif, coefficient de Poisson compris entre 0 et 0.5, ...), il est nécessaire de limiter leur domaine d'évolution. On impose donc que $\mathbf{c}$ reste dans un domaine admissible O, convexe fermé de R^{n}. Ceci impose donc des contraintes sur les paramètres :

$$\mathbf{c_{sup}} > \mathbf{c}^{\mathbf{k}} + \mathbf{g}^{\mathbf{k}} > \mathbf{c_{inf}}$$

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Après dualisation de ces conditions par introduction des multiplicateurs de Lagrange $\boldsymbol{\mu}_{\mathbf{inf}}$ et $\boldsymbol{\mu}_{\mathbf{sup}}$, on résout le système :

Trouver $\mathbf{g}^{\mathbf{k}}$ $\boldsymbol{\mu}_{\mathbf{inf}}$ $\boldsymbol{\mu}_{\mathbf{sup}}$ *tels que*

$$\left(\mathbf{A}^{\mathbf{T}}\mathbf{A}+\lambda\ \mathbf{I}\right)\ \mathbf{g}^{\mathbf{k}}+\boldsymbol{\mu}_{\mathbf{inf}}+\boldsymbol{\mu}_{\mathbf{sup}}=-\mathbf{A}^{\mathbf{T}}\mathbf{j}$$

$$\begin{cases}\mathbf{c}^{\mathbf{k}}+\mathbf{g}^{\mathbf{k}}>\mathbf{c}_{\mathbf{inf}}\\ \boldsymbol{\mu}_{\mathbf{inf}}>0\\ (\mathbf{c}^{\mathbf{k}}+\mathbf{g}^{\mathbf{k}}-\mathbf{c}_{\mathbf{inf}})_i\ (\boldsymbol{\mu}_{\mathbf{inf}})_i=0 \qquad \forall i=[1,n]\end{cases}$$

$$\begin{cases}\mathbf{c}^{\mathbf{k}}+\mathbf{g}^{\mathbf{k}}<\mathbf{c}_{\mathbf{sup}}\\ \boldsymbol{\mu}_{\mathbf{sup}}<0\\ (\mathbf{c}^{\mathbf{k}}+\mathbf{g}^{\mathbf{k}}-\mathbf{c}_{\mathbf{sup}})_{\mathbf{i}}\ (\boldsymbol{\mu}_{\mathbf{sup}})_i=0 \qquad \forall i=[1,n]\end{cases}$$

Cette résolution s'effectue à l'aide d'un algorithme de contraintes actives. Pour toute précision sur cet algorithme, se reporter à [bib3] ou [bib4].

2.3.5 Adimensionnement

On est souvent amenés à identifier des paramètres de natures physiques diverses. Les ordres de grandeur de ces paramètres peuvent être extrêmement différents. Ceci peut engendrer de très fortes différences dans les ordres de grandeur des composantes du gradient et du Hessien de la fonctionnelle coût et compromettre la résolution.
Pour pallier cette difficulté, il est impératif d'adimensionner les inconnues avant de débuter la résolution. Voici une procédure simple et efficace.

Soit $\mathbf{c}^{\mathbf{0}}={}^{T}\left[c_1^0, c_2^0, \ \ldots, \ c_n^0\right]$, le vecteur initial des grandeurs à reconstruire. On définit la matrice d'adimensionnement $\mathbf{D}$:

$$\mathbf{D}=\begin{bmatrix} c_1^0 & & & & \\ & c_2^0 & & & \\ & & \ddots & & \\ & & & c_{n-1}^0 & \\ & & & & c_n^0 \end{bmatrix} \qquad \textbf{éq 3.5-1}$$

Alors, si les c_i^0 sont tous non nuls, on peut définir les inconnues adimensionnelles par :

$$\tilde{\mathbf{c}}^{0}=\mathbf{D}^{-1}.\ \mathbf{c}^{\mathbf{0}} \qquad \textbf{éq 3.5-2}$$

De même, on introduit une fonctionnelle coût adimensionnelle :

$$\tilde{\mathbf{J}}(\tilde{\mathbf{c}})=\mathbf{J}(\mathbf{D}.\tilde{\mathbf{c}})=\mathbf{J}(\mathbf{c}) \qquad \textbf{éq 3.5-3}$$

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Ainsi que son gradient :

$$\nabla_{\tilde{\mathbf{c}}}\tilde{\mathbf{J}}(\tilde{\mathbf{c}})=\frac{\partial\tilde{\mathbf{J}}(\tilde{\mathbf{c}})}{\partial\tilde{\mathbf{c}}}=\frac{\partial\mathbf{J}(\mathbf{D}.\tilde{\mathbf{c}})}{\partial\tilde{\mathbf{c}}}=\frac{\partial\mathbf{J}(\mathbf{c})}{\partial\mathbf{c}}.\frac{\partial\mathbf{c}}{\partial\tilde{\mathbf{c}}}=\mathbf{D}.\nabla_{c}\mathbf{J}(\mathbf{c}) \qquad \textbf{éq 3.5-4}$$

Et sa matrice Jacobienne :

$$\tilde{A}_{ij}=A_{ij}\times c_j^0 \qquad \textbf{éq 3.5-5}$$

D'un point de vue algorithmique, le calcul de la matrice Jacobienne se fait classiquement avec la fonctionnelle $\mathbf{J}$, puis elle est adimensionnée ainsi que les paramètres courants $\mathbf{c}$, avant d'être transmis à l'algorithme de minimisation. À la sortie de ce dernier, les paramètres $\tilde{\mathbf{c}}$ sont redimensionnés pour permettre le calcul de la fonctionnelle $\mathbf{J}$.

2.3.6 Critère de convergence

Le critère de convergence utilisé dans MACR_RECAL consiste à tester la décroissance du gradient de la fonctionnelle. On rappelle que l'utilisation de ce critère se justifie naturellement par le fait que l'objectif de l'algorithme de recalage est d'annuler ce gradient.

$$\frac{||\nabla_{\mathbf{c}}\mathbf{J}(\mathbf{c}^{\mathbf{k}})||}{||\nabla_{\mathbf{c}}\mathbf{J}(\mathbf{c}^{\mathbf{0}})||}<Prec. \qquad \textbf{éq 3.6-1}$$

Où $Prec$ est par défaut pris égal à 1.E-3.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

2.4 Algorithme global

De manière à expliciter l'enchaînement des diverses opérations décrites ci-dessus, on présente formellement l'algorithme du recalage :

- Initialisations
  - $k=0$
  - calcul de $\mathbf{A}$, adimensionnement de $\mathbf{A}$
  - Calcul de λ initial
  - **Itérations globales**
    - Adimensionnement de $\mathbf{c}^{\mathbf{k}}$
    - Résolution de l'équation de Levenberg-Marquardt
    - Imposition du respect des bornes
    - Redimensionnement de $\mathbf{c}^{\mathbf{k+1}}$
    - Calcul de $\mathbf{J}\left(\mathbf{c}^{\mathbf{k+1}}\right)$
    - Actualisation de λ
    - Calcul de $\mathbf{A}$, adimensionnalisation de $\mathbf{A}$
    - Test de convergence
    - $k=k+1$
- Fin

3 Algorithme évolutionnaire

3.1 Principes générales

Les algorithmes évolutionnaires font partie de méthodes stochastiques d'optimisation globale basées sur les principes de l'évolution darwinienne des populations biologiques, méthodes communément nommées algorithmes génétiques. On commence d'ailleurs par rappeler les principales phases d'un algorithme génétique simple :

- *codage* : chaque paramètre à recaler est codé en base binaire ; on crée la population ;

- *évaluation* : chaque individu de la population se voit attribuer une mesure de son adaptation, calculée à partir de la fonction coût à minimiser ;

- *sélection* : les individus qui vont produire la prochaine génération de la population sont sélectionnés, on retient naturellement les meilleurs au sens de l'adaptation ;

- *croisement* : de nouveaux individus sont créés par les parents désignés à la phase précédente. En pratique il s'agit d'échanger une partie de la chaîne binaire entre deux parents ;

- *mutation* : mécanisme aléatoire qui perturbe un ou plusieurs bits de la chaîne binaire des enfants afin de maintenir un certain niveau de diversité dans la population;

- *remplacement* : une nouvelle population, de la même taille, est constituée en remplaçant les parents avec les enfants

Même si, par abus de langage, on utilise souvent le terme génétique à la place d'évolutionnaire, il faut rappeler les quelques différences qui particularisent les algorithmes évolutionnaires [5]:

- les individus de la population sont représentés par des vecteurs de nombres réels et non plus dans le codage binaire ;

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

• le processus de sélection est différent : alors que les algorithmes génétiques sélectionnent n parents pour créer n enfants qui les remplacent totalement dans la population, les programmes évolutionnaires génèrent *m* enfants à partir de n parents puis sélectionnent la nouvelle population en gardant les n meilleurs parmi l'ensemble $n+m$;

• les probabilités de croisement et de mutation peuvent varier au cours des générations.

L'intérêt de l'introduction d'un algorithme évolutionnaire dans *Code_Aster* tient dans sa capacité d'exploration multi-directionnelle de l'espace topologique des paramètres en évitant aussi que possible les minima locaux de la fonctionnelle à minimiser.

L'implémentation de cet algorithme dans Code_Aster est le fruit du partenariat entre le département AMA et Politecnico di Milano.

On note également que le défaut le plus critiqué des algorithmes évolutionnaires, celui d'être gourmands en temps CPU, peut être facilement éliminé par les nombreuses variantes de parallélisation possibles.

3.2 Fonctionnement de l'algorithme

L'algorithme est lancé en choisissant `METHODE= 'GENETIQUE'` ou `'HYBRIDE'` dans les options de la commande `MACR_RECAL`. Dans le deuxième cas, l'algorithme évolutionnaire va réaliser une recherche grossière suivie d'une optimisation par l'algorithme de Levenberg-Marquardt.

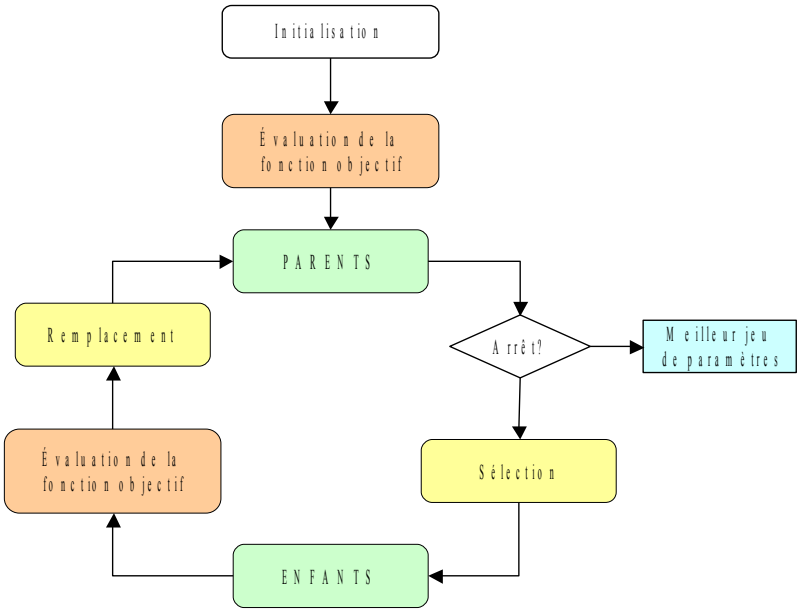


Figure 3.2-a Schéma logique du fonctionnement de l'algorithme évolutionnaire

Le fonctionnement global de l'algorithme est illustré sur le schéma logique de la Figure 3.2-a et on décrit dans ce qui suit les étapes:

a) **Initialisation**: une population de jeux de paramètres est générée et tous les individus sont initialisés avec les valeurs initiales des paramètres à recaler. La taille de cette population est donnée par la valeur du paramètre `NB_PARENTS` imposée par l'utilisateur. Cette valeur dépend de plusieurs facteurs comme l'incertitude sur la solution: plus cette incertitude est grande plus la population doit être grande. On conseille donc de démarrer une étude de recalage en prenant pour cette valeur 10 et de l'affiner en fonction de résultats.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

b) **Évaluation de la fonctionnelle**: il s'agit de la fonctionnelle présentée dans l'équation 3.1-3. Dans un premier temps, après l'étape d'initialisation, une seule évaluation est faite car tous les individus de la population sont identiques. Ensuite, dans la boucle de recalage, on fait autant d'évaluations de la fonctionnelle lors une itération que le nombre d'enfants (NB_FILS) imposé par l'utilisateur. Plus ce paramètre qui gère le taux de renouvellement de la population est grand, plus l'algorithme est gourmand en temps CPU dans cette étape. Par défaut le nombre d'enfants est fixé à 5 (moitié de la taille de la population imposée aussi par défaut).

c) **Critère d'arrêt**: une fois la population des n parents constituée et chaque individus ayant sa valeur de la fonctionnelle, on vérifie:
- la meilleure valeur de la fonctionnelle;
- le nombre d'itérations déjà réalisée par l'algorithme
Si la meilleure valeur de la fonctionnelle est inférieure au résidu relatif imposée par l'utilisateur (1.E-3 par défaut) ou si le nombre maximal d'itérations est atteint, le meilleur individus de la population est fourni comme solution.

d) **Sélection**: le meilleur individu de la population est sélectionné et c'est **le seul** (dans cette implémentation) qui a le droit de se reproduire et de générer les m enfants. Cette génération est quasi-aléatoire, autour du meilleur individu, avec un écart type imposée par l'utilisateur (mot-clé ECART_TYPE). Lors de cette étape, l'algorithme gère les bornes imposées aux paramètres par l'utilisateur. Les individus ainsi générés ne sont acceptés comme enfants que si ils sont à l'intérieur des bornes.

e) **Remplacement**: après avoir généré les *m* enfants, la population globale à ce stade de l'itération est de $n+m$ (NB_PARENTS + NB_FILS). L'opérateur de remplacement réalise ici une hiérarchie des individus en fonction des valeurs associées de la fonctionnelle et remplace les individus de la population PARENTS avec les n meilleurs parmi les $n+m$.

Une des particularités de cette implémentation de l'algorithme évolutionnaire dans *Code_Aster* est l'absence, pour l'instant, des mécanismes de mutation et croisement.

3.3 Technique hybride de recalage

En choisissant METHODE='HYBRIDE' dans MACR_RECAL, l'utilisateur a la possibilité de combiner les avantages des algorithmes stochastiques (celui évolutionnaire ici) avec ceux des algorithmes déterministes (Levenberg-Marquardt en occurrence dans notre cas). Il s'agit donc de réaliser une première recherche « grossière » dans l'espace topologique des paramètres ce qui permettra à l'algorithme de Levenberg-Marquardt de démarrer l'optimisation avec un point de départ plus proche de la solution **globale** de la fonctionnelle.
Le dosage entre stochastique et déterministe est à fixer par l'utilisateur compte-tenu de son expertise. Une grande incertitude sur les valeurs optimales des paramètres à recaler doit imposer:
- plus d'itérations de l'algorithme évolutionnaire;
- un écart type plus grand pour les tirages au sort;
- une taille de la population plus importante.

Ces sont les trois leviers que l'utilisateur a à sa disposition pour réaliser un recalage de qualité avec un niveau de performance (temps CPU et mémoire utilisée) satisfaisant.

4 Bibliographie

1. J.C. CULIOLI : « Introduction à l'optimisation », Ellipses, 1994

2. M. S. BAZARAA, H.D. SHERALI & C.M. SHETTY : « Nonlinear Programming, Theory and Algorithms », Wiley & Sons, 1979

3. « Formulation discrète du contact-frottement », Document *Code_Aster* [R5.03.50]

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

4. K. KUNISCH & F. RENDL ; « An infeasible active set method for quadratic programming with simple bounds », SIAM Journal on Optimization, Volume 14, Number 1, 2003

5. Z. Michalewicz. «Genetic Algorithms+Data Structures = Evolution Programs». Springer Verlag, 1992--1996.

5 Description des versions du document

Version Aster	Auteur(s) Organisme(s)	Description des modifications
7.1	N.TARDIEU-R&D/AMA	Texte initial
10.1	I.NISTOR - R&D/AMA	

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)