Introduire un nouveau comportement

Résumé :

Comment introduire un nouveau comportement ?
On décrit ici l'ajout d'un nouveau comportement pour résoudre un problème non linéaire posé sur une structure, avec STAT_NON_LINE ou DYNA_NON_LINE, pour tous les éléments 2D / 3D (et coques, tuyaux, poutres multi-fibres, ...) ou ajouter un nouveau comportement métallurgique dans CALC_META.

Étapes essentielles :

- Écriture de la documentation de référence R (équations de la loi de comportement)
- Modification du catalogue de DEFI_MATERIAU (paramètres matériau de la loi de comportement)
- Ajout du catalogue python de la relation de comportement
- Choix de la méthode d'intégration parmi les possibilités suivantes :
  - écriture d'une routine autonome lc0nn intégrant le comportement en un point d'intégration
  - intégration explicite (ALGO_INTE='RUNGE_KUTTA') et écriture des routines associées
  - intégration implicite dans l'environnement PLASTI, implantation « complète » (ALGO_INTE='NEWTON')
  - intégration implicite dans l'environnement PLASTI, implantation «facile » (ALGO_INTE='NEWTON_PERT')
- Produire des tests !

Remarque : il est également possible de programmer des lois de comportement soit dans une routine de type *Umat* (cf. [U2.10.01]), soit à l'aide de *MFront* (cf. [U2.10.02] et le paragraphe 6 de ce document).

Table des Matières

1 Schéma général de résolution dans STAT_NON_LINE

Itération de Newton sur le système complet

- on part de la valeur obtenue à l'issue de la phase de prédiction :

$\Delta \mathbf{u}_i^0$

- on calcule la suite d'incréments $(\delta \mathbf{u}_i^{n+1})$

$\mathbf{K}_i^n . \delta \mathbf{u}_i^{n+1} = \mathbf{L}_i^{méca} - \mathbf{L}^{int}(\Delta \mathbf{u}_i^n, \boldsymbol{\sigma}_{i-1}, \boldsymbol{\alpha}_{i-1})$

- on actualise le déplacement $\mathbf{u}$:

$\Delta \mathbf{u}_i^{n+1} = \Delta \mathbf{u}_i^n + \delta \mathbf{u}_i^{n+1}$

- on teste la convergence et on réitère si besoin

Newton
i : n° de pas de temps
n : n° d'itération de Newton

Référence : Principe de résolution (algorithme de Newton) [R5.03.01] et module de cours Aster Non linéaire

au niveau global : l'itération *n* de Newton fournit $\Delta \mathbf{u}_i^n = \Delta \mathbf{u}_i^{n-1} + \delta \mathbf{u}_i^n$

au niveau de l'élément : calcul de $\boldsymbol{\varepsilon}(\Delta \mathbf{u}_i^n)$ **en chaque point de Gauss (PG)**

au niveau du PG : intégration de la loi de comportement

calcul des contraintes et variables internes $\begin{cases} \boldsymbol{\sigma}_{i-1}, \boldsymbol{\alpha}_{i-1} \\ \boldsymbol{\varepsilon}(\Delta \mathbf{u}_i^n) \end{cases} \rightarrow \boldsymbol{\sigma}_i^n, \boldsymbol{\alpha}_i^n$

calcul éventuel de la dérivée de $\boldsymbol{\sigma}$ par rapport à $\boldsymbol{\varepsilon}$ $\begin{cases} \boldsymbol{\sigma}_{i-1}, \boldsymbol{\alpha}_{i-1} \\ \boldsymbol{\varepsilon}(\Delta \mathbf{u}_i^n) \end{cases} \rightarrow \left(\frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{\varepsilon}}\right)_i^n$

seul endroit où intervenir

calcul des forces nodales élémentaires ${}^E(\mathbf{L}^{int})_i^n = \int_{\Omega_E} \mathbf{Q}^T \boldsymbol{\sigma}_i^n \, d\Omega$

calcul éventuel de la matrice tangente élémentaire ${}^E\mathbf{K}_i^n = \int_{\Omega_E} \mathbf{Q}^T \left(\frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{\varepsilon}}\right)_i^n \mathbf{Q} \, d\Omega$

assemblage des forces nodales $\mathbf{L}^{int}(\Delta \mathbf{u}_i^n, \boldsymbol{\sigma}_{i-1}, \boldsymbol{\alpha}_{i-1}) = \sum_E {}^E(\mathbf{L}^{int})_i^n$

calcul du résidu $\mathbf{L}_i^{meca} - \mathbf{L}^{int}(\Delta \mathbf{u}_i^n, \boldsymbol{\sigma}_{i-1}, \boldsymbol{\alpha}_{i-1})$

assemblage éventuel de la matrice tangente $\mathbf{K}_i^n = \sum_E {}^E\mathbf{K}_i^n$

1.1 Options de calcul concernées

1.1.1 Option FULL_MECA (Full Newton)

Au pas de temps i et à l'itération de Newton n , à partir des contraintes et variables internes à l'équilibre précédent $(\sigma_{i-1}, \alpha_{i-1})$ et de l'incrément de déformation $\varepsilon(\Delta \mathbf{u}_i^n)$ (et éventuellement avec les variables de commande : température, hydratation, …), calcul en chaque point de Gauss de chaque élément fini :

- des contraintes et variables internes (SIEF_ELGA,VARI_ELGA) :

$$\begin{cases} \sigma_{i-1}, \alpha_{i-1} \\ \varepsilon(\Delta \mathbf{u}_i^n) \end{cases} \rightarrow \sigma_i^n, \alpha_i^n$$

- de l'opérateur tangent :

$$\begin{cases} \sigma_{i-1}, \alpha_{i-1} \\ \varepsilon(\Delta \mathbf{u}_i^n) \end{cases} \rightarrow \left(\frac{\partial \sigma}{\partial \varepsilon}\right)_i^n$$

Cette option est calculée si REAC_ITER= m dans le fichier de commandes, et que le numéro d'itération n est multiple de m (réactualisation de la matrice tangente cohérente).

1.1.2 Option RAPH_MECA (Newton-Raphson)

Au pas de temps i et à l'itération de Newton n , à partir des contraintes et variables internes à l'équilibre précédent $(\sigma_{i-1}, \alpha_{i-1})$ et de l'incrément de déformation $\varepsilon(\Delta \mathbf{u}_i^n)$ (et éventuellement avec les variables de commande : température, hydratation, …), calcul en chaque point de Gauss de chaque élément fini :

- des contraintes et variables internes (SIEF_ELGA,VARI_ELGA) :

$$\begin{cases} \sigma_{i-1}, \alpha_{i-1} \\ \varepsilon(\Delta \mathbf{u}_i^n) \end{cases} \rightarrow \sigma_i^n, \alpha_i^n$$

Cette option est calculée si REAC_ITER=0 ou REAC_ITER=m dans le fichier de commandes, et que le numéro d'itération n n'est pas multiple de m .

1.1.3 Option RIGI_MECA_TANG (calcul de la matrice tangente en prédiction)

A l'itération 0 du pas de temps i (initialisation de l'algorithme de Newton), on choisit comme matrice tangente de prédiction la matrice tangente à l'équilibre précédent ($i-1$), soit $\mathbf{K}_i^0 = \mathbf{K}_{i-1}$. À partir des contraintes et variables internes à l'équilibre précédent $(\sigma_{i-1}, \alpha_{i-1})$, calcul en chaque points de Gauss de chaque élément fini :

- de l'opérateur tangent en prédiction :

$$\sigma_{i-1}, \alpha_{i-1} \rightarrow \left(\frac{\partial \sigma}{\partial \varepsilon}\right)_i^0$$

Cette option est calculée si REAC_INCR= m dans le fichier de commandes, et que le numéro de pas de temps i est multiple de m (réactualisation de l'opérateur tangent en prédiction).

1.2 Remarques concernant le calcul du résidu et de la matrice tangente

1.2.1 Calcul du résidu

Le calcul exact du résidu $\mathbf{L}_i^{\text{meca}} - \mathbf{L}^{\text{int}}(\Delta \mathbf{u}_i^n, \boldsymbol{\sigma}_{i-1}, \boldsymbol{\alpha}_{i-1})$ (et donc des contraintes et des variables internes) est fondamental : il garantit que l'on convergera vers la solution du problème. Une petite erreur dans l'évaluation du résidu peut avoir des conséquences graves .

1.2.2 Calcul de la matrice tangente

Matrice tangente dite cohérente ou consistante (Option FULL_MECA) :

Réactualisée à chaque itération, elle assure la meilleure vitesse de convergence (quadratique) à l'algorithme de Newton (figure 1.2.2-1). Son calcul reste cependant coûteux, et dans le cas où l'on utilise un solveur direct, il faut ajouter au coût de chaque réactualisation celui d'une factorisation. Enfin, pour de grands incréments de chargement, la matrice tangente cohérente peut conduire à des divergences de l'algorithme.

Autres matrices « tangentes » :

On peut faire des erreurs ou des approximations dans le calcul de la matrice "tangente" : ceci conduit à dégrader la vitesse de convergence par rapport à celle qui est obtenue avec la matrice tangente cohérente réactualisée à chaque itération, mais la solution obtenue reste juste tant que le résidu est calculé de manière exacte. Il existe plusieurs variantes (méthodes de quasi-Newton) possibles autorisées par STAT_NON_LINE (pour plus de détails voir [R5.03.01]) :

- Matrice élastique (figure 1.2.2-2)
  - calculée une seule fois (économique) à partir des paramètres d'élasticité
  - recommandée en cas de décharge
  - convergence lente mais assurée
- Matrice tangente réactualisée tous les i_0 incréments de charge (figure 1.2.2-3) ou toutes les n_0 itérations de Newton (figure 1.2.2-4)
  - coût moindre
  - direction moins bien évaluée
  - diverge parfois dans les zones de forte non linéarité

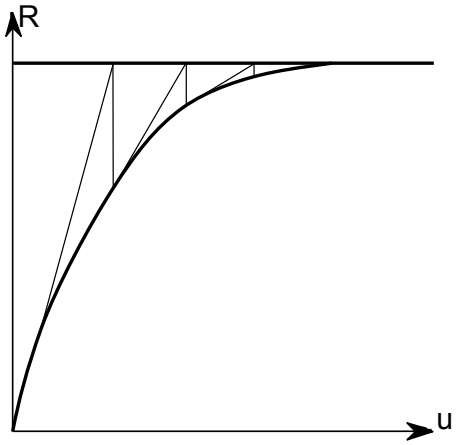


Figure 1.2.2-1: matrice tangente réactualisée à chaque itération

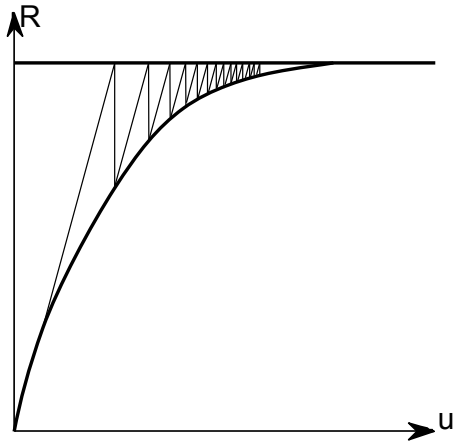


Figure 1.2.2-2: matrice élastique

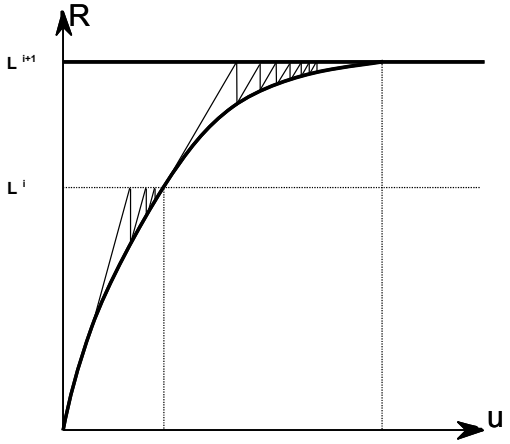


Figure 1.2.2-3: matrice tangente réactualisée à chaque incrément de chargement

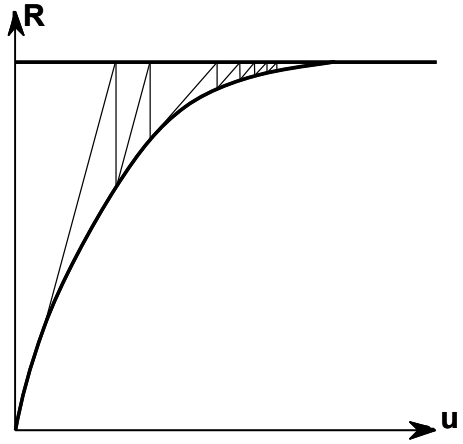


Figure 1.2.2-4: matrice tangente réactualisée toute les deux itérations de Newton

2 Documentation de Référence et choix de la méthode d'intégration

L'écriture de la Documentation de Référence est préalable à la phase de développement. Le document (*cf.* par exemple [R5.03.02]) doit spécifier selon le type de modélisation concernée (milieux continus `2D D_PLAN` / `AXIS` et `3D` , ou modèles à plasticité locale tels que les coques, les plaques et les tuyaux et `C_PLAN`...) :

- le choix de la méthode d'intégration (voir les différentes possibilités détaillées au § 5) ;
- les équations permettant de calculer les contraintes et variables internes ;
- les équations permettant de calculer les deux matrices tangentes (options `RIGI_MECA_TANG` et `FULL_MECA`).

On peut citer d'autres exemples de Documentations de Référence :

- [R5.03.04] : *Relations de comportement élasto-visco-plastique de Chaboche.*
- [R5.03.16] : *Relation de comportement élasto-plastique à écrouissage cinématique linéaire et isotrope non linéaire*.
- [R5.03.20] : *Relation de comportement élastique non linéaire en grands déplacements.*
- [R5.03.21] : *Modélisation élasto-plastique avec écrouissage isotrope en grandes déformations.*

3 Mode opératoire : les catalogues

3.1 Modification du catalogue de DEFI_MATERIAU

Le but de l'opérateur DEFI_MATERIAU [U4.43.01] est d'introduire des paramètres de comportement. Ces paramètres peuvent être communs à plusieurs relations de comportement (voir l'exemple ci-dessous pour les relations de comportement VMIS_ISOT_LINE et VMIS_CINE_LINE).

Il faut donc ajouter dans le catalogue defi_materiau.py (répertoire code_aster/Cata/Commands) un mot clé facteur correspondant au type de comportement que l'on souhaite introduire, et sous ce mot clé facteur, ajouter les mots clés simples représentant les paramètres de ce type de comportement.

Exemple :

```
ECRO_LINE = FACT(statut='f',
    D_SIGM_EPSI = SIMP(statut='o',typ='R',),
    SY          = SIMP(statut='o',typ='R',),),...
```

signifie que les deux mots-clés SY et D_SIGM_EPSI sont obligatoires pour ECRO_LINE (pour plus de précisions, se reporter à [U1.03.01] : *Superviseur et langage de commandes*)

3.2 Modification du catalogue C_RELATION

Il faut ajouter à la liste renvoyée par C_RELATION() le nom choisi pour la relation de comportement que l'on souhaite introduire ('MA_RELATION' dans l'exemple ci-dessous). Le catalogue à modifier est c_relation.py (répertoire code_aster/Cata/Commons).

Exemple :

```
def C_RELATION() : return (             "ELAS",             #COMMUN#
                                        ...
                                        "LAIGLE",
                                        "LEMAITRE",
                                        "LEMAITRE_IRRA",
                                        "LEMA_SEUIL",
                                        "LETK",
                                        "MA_RELATION",
                                        "MAZARS",
                                        "MAZARS_1D",
                                        ...
                                    )
```

3.3 Ajouter le catalogue de la loi de comportement

Ce catalogue est à ajouter dans le répertoire bibpyt/Comportement.

3.3.1 Contenu

Exemple : vmis_cine_line.py

```
loi = LoiComportement(
    nom            = 'VMIS_CINE_LINE',
    lc_type        = ('MECANIQUE',),
```

```
    doc            = """Loi de Von Mises - Prager avec ecrouissage
cinematique lineaire [R5.03.02]""" ,
    num_lc         = 3,
    nb_vari        = 7,
    nom_vari       = ('XCINXX','XCINYY','XCINZZ','XCINXY','XCINXZ',
        'XCINYZ','INDIPLAS',),
    mc_mater       = ('ELAS','ECRO_LINE',),
    modelisation   = ('3D','AXIS','D_PLAN','1D',),
    deformation    = ('PETIT','PETIT_REAC','GROT_GDEP','GDEF_LOG',),
    algo_inte      = ('ANALYTIQUE',),
    type_matr_tang = ('PERTURBATION','VERIFICATION',),
    proprietes     = None,
    syme_matr_tang = ('Yes',),
    exte_vari      = None,
    deform_ldc     = ('OLD',),
)
```

On fournit donc dans ce catalogue une grande partie des informations relatives au comportement :

- `nom` : nom de la loi, identique à celui fourni pour `COMPR_INCR` / `RELATION`
- `num_lc` : numéro de routine `lc00nn`
- `nb_vari / nom_vari` : nombre de variables internes, et leurs noms (`K8`)
- `mc_mater` : mots-clés utilisés dans `DEFI_MATERIAU`
- `modelisation` : types de modélisations possibles, pour les comportements de milieux continus : `3D, D_PLAN, AXIS, C_PLAN, COMP1D, INCO, GRADEPSI, GRADVARI, …`
- `deformation` : type de déformations possibles :'`PETIT`', '`PETIT_REAC`', '`GROT_GDEP`', '`GDEF_LOG`', '`GDEF_HYPO_ELAS`'.
- `algo_inte` : schémas d'intégration possibles : implicite (`'ANALYTIQUE'`, `'NEWTON_PERT'`...), explicite (`'RUNGE_KUTTA'`)
- `type_matr_tang` : types de matrices tangentes disponibles. Outre la matrice par perturbation, on peut aussi utiliser les matrices sécantes, et la combinaison `TANGENTE_SECANTE`.
- `proprietes` :
- `syme_matr_tang` :
- `exte_vari` : nom des variables de commandes prises en compte
- `deform_ldc` : type de déformation passée en entrée de la loi de comportement : `'OLD'`, `'MECANIQUE'` ou `'TOTALE'` (*cf.* §4)

Remarque s *:*

- *Les noms des variables internes de l'ensemble des comportements sont définis dans le catalogue python `cata_vari.py` , afin de nommer de façon identique les variables internes de même signification. Ce catalogue est disponible dans le répertoire `bibpyt/Comportement`. Pour un nouveau comportement, il est souhaitable de réutiliser des noms déjà existants. Si on ajoute de nouveaux noms, une erreur se produit à l'exécution ; il faut donc modifier `cata_vari.py` , en justifiant son choix lors de la restitution des développements.*
- *Les variables de commande (ou « external state variables ») prises en compte dans le catalogue sont spécifiques à certaines lois de comportement (voir § 5.4.3). Pour l'instant, cette liste ne décrit pas les variables génériques comme la température ou le séchage.*

Attention *:*

Lorsque que l'on ajoute un nouveau catalogue, bien vérifier la présence de la carte d'ajout en tête de fichier. Elle précise dans quelle bibliothèque python placer le fichier (ici `Comportement`) :
`#@ AJOUT maloidecomportement Comportement`

3.3.2 Cas d'une loi compatible avec la cinématique SIMO_MIEHE

Si la loi de comportement est développée pour être compatible avec une cinématique de type SIMO_MIEHE, en plus d'autres cinématiques, alors deux catalogues doivent être créés :

- un premier catalogue correspondant à toutes les cinématiques sauf SIMO_MIEHE
- un second correspondant uniquement à la cinématique SIMO_MIEHE

La distinction entre les deux catalogues se fait par l'intermédiaire du nom, sur le modèle suivant (chiffre 2 à la 4ème position) :
vmis_isot_line.py et vmis2isot_line.py, meta_p_il_pt.py et meta2p_il_pt.py, monocristal.py et mono2ristal.py...

Les deux catalogues sont identiques à l'exception des attributs doc et deformation :

```
vmis_isot_line.py :
  nom            = 'VMIS_ISOT_LINE',
  doc            = """Loi de plasticité de Von Mises à écrouissage linéaire
[R5.03.02]""",
  deformation    = ('PETIT','PETIT_REAC','GROT_GDEP','GDEF_LOG',),

vmis2isot_line.py :
  nom            = 'VMIS_ISOT_LINE',
  doc            = """Loi de plasticité de Von Mises à écrouissage linéaire
pour Simo_Miehe [R5.03.02]""",
  deformation    = ('SIMO_MIEHE',),
```

et éventuellement de l'attribut deform_ldc qui ne peut pas prendre la valeur 'MECANIQUE' pour le catalogue correspondant à SIMO_MIEHE : c'est justement pour permettre aux autres cinématiques de fonctionner selon le mécanisme deform_ldc = 'MECANIQUE' que ce système de dédoublement de catalogue a été mis en place.

Pour les lois n'étant développées **que** pour SIMO_MIEHE , alors il suffit de créer un seul catalogue sans modifier son nom, par exemple : rousselier.py , visc_isot_line.py ...

4 Déformation en entrée de la loi de comportement

Par définition, la loi de comportement permet de calculer les contraintes et les variables internes à partir des déformations dites « mécaniques », c'est-à-dire exemptes des déformations potentiellement générées par les variables de commande, comme les déformations thermiques dues à la température ou le retrait de dessiccation dû au séchage.

Avant de procéder à l'intégration, il faut donc avant toute chose préparer la déformation « mécanique » avec laquelle travaillera la loi de comportement.

Plusieurs mécanismes sont proposés au développeur de la loi de comportement :

- la préparation de la déformation se fait en amont de la loi de comportement, qui reçoit donc directement la déformation mécanique en entrée. Pour bénéficier de ce mécanisme, l'attribut deform_ldc doit être spécifié à la valeur 'MECANIQUE' dans le catalogue de la loi.
- la préparation est à la charge du développeur de la loi de comportement, qui reçoit donc la déformation totale en entrée. L'attribut deform_ldc du catalogue est alors à la valeur 'TOTALE'.
- si la loi de comportement n'a pas été modifiée depuis la mise en place de ce système 'MECANIQUE'/'TOTALE', alors l'attribut deform_ldc est à la valeur 'OLD'. Les lois de comportement en 'OLD' sont à terme destinées à disparaître.

Ces différents mécanismes ne sont proposés que pour les lois interfacées avec la routine nmcomp.F90 (fonctionnement lc).

Historiquement, la majorité des lois recevait en entrée la déformation totale. Le mécanisme deform_ldc = 'MECANIQUE' a été mis en place dans le cadre d'une modification de la phase de prédiction dans l'algorithme de Newton visant à améliorer la convergence du code dans certains cas. Les lois ont donc tout à gagner à travailler avec le mécanisme deform_ldc = 'MECANIQUE' dès lors que c'est possible.

Suivant le mécanisme que suit la loi de comportement, un terme particulier (correspondant au terme d'ordre 0 de la linéarisation de la contrainte sur laquelle est basé l'algorithme de Newton) est attendu et doit être transmis à travers la variable sigp en sortie de la loi de comportement en phase de prédiction.

Décrivons précisément les entrées/sorties attendues suivant les différents mécanismes.

4.1 Mécanisme deform_ldc = 'MECANIQUE'

Dans ce fonctionnement, on a entre autres les entrées – sorties suivantes pour la loi de comportement (c'est-à-dire pour la routine lc****.F90 correspondante) :

<table>
<tr><td>epsm</td><td colspan="2">Déformation mécanique à l'instant précédent $i-1$</td></tr>
<tr><td>deps</td><td colspan="2">Incrément de déformation mécanique sur le pas de temps $\Delta\epsilon_m$</td></tr>
<tr><td>sigp</td><td>En prédiction , le terme d'ordre 0 suivant est attendu :
$\tilde{\tilde{\sigma}}(\epsilon_m^{i-1}, c^i, \Delta t) - \frac{\partial \tilde{\tilde{\sigma}}}{\partial \epsilon_m}(\epsilon_m^{i-1}, c^i, \Delta t)\Delta\epsilon_c$
où $\tilde{\tilde{\sigma}}$ est la contrainte exprimée comme fonction de la déformation mécanique $\epsilon_m = \epsilon - \epsilon_c$ évaluée à l'instant $i-1$
c est la (ou les) variable de commande, évaluée à l'instant i
Δt est l'incrément de temps (dont la loi dépend explicitement dans le cas visqueux)
et $\Delta\epsilon_c$ est l'incrément de déformation due aux variables de commande, avec $\Delta\epsilon_c = \Delta\epsilon_{c1} + \Delta\epsilon_{c2} + \ldots$ dans le cas où il y en a plusieurs.</td><td>En correction, on attend classiquement les contraintes à l'instant actuel i :
$\sigma_i = \tilde{\tilde{\sigma}}(\epsilon_m^i, c^i, \Delta t)$.</td></tr>
<tr><td>dsidep</td><td colspan="2">En prédiction et en correction : $\frac{\partial \tilde{\tilde{\sigma}}}{\partial \epsilon_m}(\epsilon_m^{i-1}, c^i, \Delta t)$.</td></tr>
</table>

Périmètre de couverture :

Ce mécanisme ne convient que lorsque les variables de commande génèrent des déformations de type sphérique (proportionnelles à la matrice Identité).

Il n'est disponible que pour des cinématiques de type petites déformations pour lesquelles on a additivité de la déformation mécanique et des déformations dues aux variables de commande $\epsilon = \epsilon_m + \epsilon_c$. Il s'agit en fait de toutes les cinématiques sauf SIMO_MIEHE, soit : PETIT, PETIT_REAC, GDEF_LOG et en principe GROT_GDEP, bien que le mécanisme ne soit pour l'instant pas disponible pour cette dernière cinématique.

Cela signifie qu'une loi travaillant avec SIMO_MIEHE (et donc pour l'instant GROT_GDEP) ne doit pas travailler en deform_ldc = 'MECANIQUE' .

On précisera également que la prise en charge des déformations dues aux variables de commande et la préparation de la déformation mécanique sont effectuées pour les cas suivants :

- calcul et prise en compte de la déformation thermique avec un coefficient de dilatation thermique ALPHA isotrope / anisotrope / transverse isotrope, et différenciable selon la phase métallurgique dans le cas isotrope ;
- calcul et prise en compte du retrait de dessiccation et du retrait endogène dans le cas où les paramètres matériau les contrôlant, K_DESSIC et B_ENDOGE , sont isotropes ;
- prise en compte des déformations anélastiques 'EPSAXX' , 'EPSAYY' , 'EPSAZZ' , 'EPSAXY' , 'EPSAXZ' , 'EPSAXZ' .

Dans le cas où les variables de commande généreraient d'autres déformations et qu'il est nécessaire de sortir de ce périmètre (par exemple, si les déformations générées ne sont pas de type sphérique), alors le mécanisme deform_ldc = 'MECANIQUE' ne peut pas être adopté et il faut s'orienter vers le mécanisme deform_ldc = 'TOTALE'.

On notera que les lois de type MFront et UMAT suivent le mécanisme deform_ldc = 'MECANIQUE' .

4.2 Mécanisme deform_ldc = 'TOTALE'

Ce mécanisme est proposé pour les lois pour lesquelles le calcul des déformations dues aux variables de commande est plus compliqué que ce qui est couvert actuellement par le mécanisme deform_ldc = 'MECANIQUE', afin qu'elles puissent tout de même bénéficier d'une meilleure prédiction.

Dans ce mécanisme, on a entre autres les entrées/sorties de la routine lc****.F90 correspondante suivantes :

epsm	Déformation totale à l'instant précédent $i-1$	
deps	Incrément de déformation totale sur le pas de temps $\Delta\epsilon$	
sigp	**En prédiction** , le développeur peut retourner le terme d'ordre 0 qu'il estime le plus pertinent. Une possibilité est la quantité : $\widetilde{\sigma}(\epsilon^{i-1}, c^i, \Delta t)$ où $\widetilde{\sigma}$ est la contrainte exprimée comme fonction de la déformation totale ϵ évaluée à l'instant $i-1$ c est la (ou les) variable de commande, évaluée à l'instant i Δt est l'incrément de temps (dont la loi dépend explicitement dans le cas visqueux).	**En correction**, on attend classiquement les contraintes à l'instant actuel i : $\boldsymbol{\sigma}_i = \widetilde{\sigma}(\epsilon^i, c^i, \Delta t)$.
dsidep	**En prédiction et en correction :** $\frac{\partial \widetilde{\sigma}}{\partial \epsilon}(\epsilon^{i-1}, c^i, \Delta t)$.	

En d'autres termes, ce mécanisme offre la possibilité au développeur de communiquer le second membre de son choix à l'algorithme de Newton pour la phase de prédiction.

On insistera sur le fait que le développeur doit ici prendre en charge la préparation de la déformation mécanique à l'intérieur de la loi de comportement.

Les lois travaillant en SIMO_MIEHE doivent suivre ce mécanisme.

4.3 Mécanisme deform_ldc = 'OLD'

La valeur de l'attribut deform_ldc = 'OLD' indique ici seulement que la loi n'a pas été modifiée pour travailler avec les entrées/sorties décrites ci-dessus depuis que les mécanismes précédents ont été mis en place. Dans ce mécanisme, le terme d'ordre 0 est évalué avec les contraintes convergées à l'instant précédent (option FORC_NODA), sans communication avec la loi de comportement.
Ce système est destiné à disparaître, et une loi nouvellement développée ne devrait donc pas prendre cette valeur de l'attribut.

5 Mode opératoire : les routines à écrire

C'est à ce niveau que doit être fait le choix du type d'intégration. Il existe quatre possibilités :

1. Utiliser l'architecture de l'environnement d'intégration explicite par un schéma de Runge-Kutta d'ordre 2 [R5.03.14] (ALGO_INTE='RUNGE_KUTA') :

   - il s'agit de la méthode la plus simple. Outre la récupération des données matériaux, il suffit d'écrire une routine calculant les dérivées des variables internes
   - le calcul de l'opérateur tangent n'est pas disponible sous cet environnement, c'est l'opérateur de rigidité élastique qui est utilisé

2. Implantation « complète » du nouveau comportement dans l'environnement d'intégration implicite PLASTI [R5.03.14] (ALGO_INTE='NEWTON') :

   - résolution du système non linéaire local par la méthode de Newton. Outre la récupération des données matériaux, il faut écrire plusieurs routines appelées dans l'algorithme de Newton local (évaluation du seuil, calcul du résidu, calcul analytique de la matrice jacobienne...)
   - L'opérateur cohérent est obtenu directement à partir de la jacobienne du système local, et l'opérateur tangent en prédiction est par défaut l'opérateur de rigidité élastique
   - PLASTI ne permet pas d'obtenir des modèles optimisés en temps calcul

3. Implantation «facile » du nouveau comportement dans l'environnement d'intégration implicite PLASTI avec [R5.03.14] (ALGO_INTE='NEWTON_PERT') :

   - le système non-linéaire local peut être réécrit de telle sorte que l'évaluation du résidu ne nécessite de la part du développeur que de spécifier l'expression des dérivées des variables internes écrites dans le cadre de la méthode 2 (intégration explicite par RK2). On peut donc également réaliser une intégration implicite dans PLASTI avec les deux seules routines nécessaires à l'intégration explicite.
   - La jacobienne du système local est calculée par perturbation, le calcul est donc encore plus coûteux qu'avec la méthode 2. De la même manière, l'opérateur tangent cohérent est obtenu directement à partir de la jacobienne du système local

4. Créer une routine autonome d'intégration complète du comportement :

   - permet souvent d'obtenir les modèles les plus performants (par exemple, en réduisant le système à résoudre à une seule équation scalaire, non linéaire, voir par exemple [R5.03.04], [R5.03.16], [R5.03.21], ...)
   - nécessite plus de travail « sur le papier » pour optimiser les équations

Attention :

Dans le cas 4, on choisira un numéro de routine nn et on écrira la routine lc00nn. Dans les autres cas on choisira comme point d'entrée le numéro 32 : LC0032 appelle PLASTI ou NMVPRK (Runge-Kutta) suivant la valeur de ALGO_INTE choisie par l'utilisateur.

5.1 Première possibilité : introduire un nouveau comportement explicite – schéma de RUNGE - KUTTA

Ce type d'intégration correspond à ALGO_INTE='RUNGE_KUTTA', c'est la façon la plus rapide d'introduire un nouveau comportement.

Il faut écrire dans un premier temps une routine XXXMAT appelée par la routine d'aiguillage LCMATE afin de récupérer les paramètres matériaux et la taille du système différentiel non-linéaire à intégrer.

```
SUBROUTINE XXXMAT(FAMI,KPG,KSP,MOD,IMAT,NMAT,MATERD,MATERF,
                  MATCST,NDT,NDI,NR,NVI,VIND)
```

Arguments en entrée :

```
FAMI,KPG,KSP : famille et numéro de point de gauss / sous-point
IMAT         : adresse du matériau
MOD          : type de modelisation
NMAT         : dimension de MATERD / MATERF
```

Arguments en sortie :

```
MATERD       :  coefficients matériau a t
MATERF       :  coefficients matériau a t+dt
                MATERx(*,1) = caractéristiques élastiques
                MATERx(*,2) = caractéristiques plastiques
MATCST       :  'oui' si  matériau a t = matériau a t+dt
                'non' sinon
NDT          :  nb total de composantes des tenseurs
NDI          :  nb de composantes directes des tenseurs
NR           :  nb de composantes du système non-linéaire
NVI          :  nb de variables internes
```

On donne ci-dessous un exemple pour chacune des deux fonctions principales que doit remplir cette routine

- **AFFECTATION DES DIMENSIONS DU PROBLEME LOCAL (NDT,NDI,NR,NVI)**

```
NVI=7
IF ( MOD .EQ.'3D' ) THEN
   NDT = 6
   NDI = 3
   NR  = NDT+2
ELSE IF ( MOD .EQ. 'D_PLAN' .OR. MOD .EQ. 'AXIS') THEN
   NDT = 4
   NDI = 3
   NR  = NDT+2
ELSE
   CALL U2MESS('F',...)
ENDIF
```

- **RECUPERATION DU MATERIAU**

```
 NOMC(1) = 'E'
 NOMC(2) = 'NU'
```

```
 NOMC(3) = 'ALPHA'
 NOMC(4) = 'SY'
 NOMC(5) = 'D_SIGM_EPSI'
 CALL RCVALB(FAMI,KPG,KSP,'-',IMAT,' ','ELAS',0,' ',
&            0.D0,3,NOMC(1),MATERD(1,1),ICODRE,1)
 CALL RCVALB(FAMI,KPG,KSP,'-',IMAT,' ','ECRO_LINE',0,' ',
&            0.D0,2,NOMC(4),MATERD(1,2),ICODRE,1)
```

Il faut ensuite écrire une routine RKDXXX appelée par la routine d'aiguillage LCDVIN et donnant les dérivées temporelles des variables internes.

Exemples de routine RKDXXX : RKDCHA, RKDVEC, RKDHAY.

5.2 Deuxième possibilité : introduction « complète » d'un nouveau comportement dans PLASTI (implicite)

Cet type d'intégration correspond à ALGO_INTE='NEWTON'. L'environnement PLASTI permet d'intégrer de manière systématique des relations de comportement non-linéaires par une méthode de Newton locale (au niveau du point de Gauss). Connaissant les contraintes et les variables internes à l'instant $i-1$ ainsi que l'incrément de déformation totale $\Delta\boldsymbol{\varepsilon}_i^n$ donné par l'algorithme de Newton global, le système local d'équations à résoudre sous forme purement implicite est écrit de la façon suivante :

$$R(\Delta \boldsymbol{y})=\begin{pmatrix} g(\Delta \boldsymbol{y}) \\ l(\Delta \boldsymbol{y}) \\ f(\Delta \boldsymbol{y}) \end{pmatrix}=0 \text{ avec } \Delta \boldsymbol{y}=\begin{pmatrix} \Delta \boldsymbol{\sigma} \\ \Delta \text{ vari} \\ \Delta p \end{pmatrix}$$

La première équation représente par exemple la relation contrainte-déformation élastique (6 équations à 6 inconnues), avec $\mathbf{A}$ l'opérateur d'élasticité (éventuellement modifié pour les lois avec endommagement), $\Delta\boldsymbol{\varepsilon}^p$ la variation de déformation plastique et $\Delta\boldsymbol{\varepsilon}^{th}$ la variation de déformation thermique :

$$g(\Delta \boldsymbol{y})=\Delta\boldsymbol{\sigma}-\mathbf{A}(\Delta\boldsymbol{\varepsilon}_i^n-\Delta\boldsymbol{\varepsilon}^{th}-\Delta\boldsymbol{\varepsilon}^p)=0$$

la seconde représente l'ensemble des lois d'évolution des différentes variables internes scalaires et/ou vectorielles (n_v équations scalaires à n_v inconnues)

$$l(\Delta \boldsymbol{y})=0$$

la dernière représente le critère éventuel de plasticité (1 équation)

$$f(\Delta \boldsymbol{y})=0$$

Ce système de $6+n_v(+1)$ équations à $6+n_v(+1)$ inconnues est résolu par une méthode de Newton :

$$\begin{cases} \dfrac{\partial R}{\partial \Delta y}(\Delta y_k).d[\Delta y_k]=-R(\Delta y_k) \\ \Delta y_{k+1}=\Delta y_k+d(\Delta y_k) \end{cases}$$

A convergence,on obtient donc les incréments de contraintes et de variables internes. L'opérateur tangent cohérent est quant à lui calculé de manière systématique à partir de la jacobienne du système local par la routine LCOPTG (voir [R5.03.14] pour le détail des équations). Il faut donc programmer a

minima une routine définissant le résidu (formé des équations ci-dessus) ainsi qu'une routine construisant la matrice jacobienne. On décrit brièvement ci-dessous l'architecture générale de PLASTI, en indiquant au fur et à mesure la liste des routines à écrire.

Architecture générale de PLASTI :

```
CALL LCMATE(...)
→ écriture nécessaire d'une routine spécifique XXXMAT de récupération du
  matériau identique à RUNGE_KUTTA

IF ( OPT .EQ. 'RAPH_MECA' .OR. OPT .EQ. 'FULL_MECA' ) THEN
  INTEGRATION ELASTIQUE SUR DT
  CALL LCELAS(...)
  → écriture éventuelle d'une routine spécifique XXXELA (par défaut
    LCELIN : élasticité linéaire)

  PREDICTION ETAT ELASTIQUE A T+DT
  CALL LCCNVX(...,SEUIL)
  → écriture nécessaire d'une routine spécifique XXXCVX d' évaluation du
    seuil

  IF ( SEUIL .GE. 0.D0 ) THEN
    CALL LCPLAS(...)
  ENDIF
ENDIF
```

La routine LCPLAS appelle LCPLNL, qui réalise **la boucle de Newton** dont la structure est la suivante :

Notations :

YD=(SIGD,VIND) : vecteur des inconnues (de dimension $6+n_v$) à l'instant T
YF=(SIGF,VINF) : vecteur des inconnues à l'instant T+DT
DY : incrément du vecteur des inconnues entre les instants T et T+DT
DDY : incrément de vecteur des inconnues entre deux itérations de Newton successives
R : résidu
DRDY : jacobienne

On résout donc : R(DY) = 0
Par une méthode de Newton DRDY(DYK) DDYK = - R(DYK)
DYK+1 = DYK + DDYK (DY0 DEBUT)
et on réactualise YF = YD + DY

```
CALCUL DE LA SOLUTION D ESSAI INITIALE DU SYSTEME NL EN DY
CALL LCINIT(...DY,...)
→ écriture éventuelle d'une routine spécifique XXXINI (par défaut DY est
  initialisé à 0)

ITERATIONS DE NEWTON
ITER = 0
1 CONTINUE
  ITER = ITER + 1

INCREMENTATION DE  YF = YD + DY
  CALL LCSOVN(NR,YD,DY,YF)

CALCUL DES TERMES DU SYSTEME A T+DT = -R(DY)
  CALL LCRESI(...,DY,R,IRET)
→ écriture nécessaire d'une routine spécifique XXXRES de calcul du
  résidu
```

```
CALCUL DU JACOBIEN DU SYSTEME A T+DT = DRDY(DY)
  si ALGO_INTE='NEWTON' calcul exact de la jacobienne
    CALL LCJACB(...DY,...DRDY,IRET)
    → écriture nécessaire d'une routine spécifique XXXJAC
  sinon si ALGO_INTE='NEWTON_PERT', calcul par perturbation (cf § 5.3)
    CALL LCJACP(...DRDY,...)

RESOLUTION DU SYSTEME LINEAIRE DRDY(DY).DDY = -R(DY)
  CALL LCEQMN(NR,DRDY,DRDY1)
  CALL LCEQVN(NR,  R ,  DDY)
  CALL MGAUSS('NCWP',DRDY1,DDY,NR,NR,1,RBID,IRET )

REACTUALISATION DE DY = DY + DDY
  CALL LCSOVN ( NR , DDY , DY , DY )

RECHERCHE LINEAIRE dans le cas ALGO_INTE='NEWTON_RELI'
  CALL LCRELI ( …)

ESTIMATION DE LA CONVERGENCE
  CALL LCCONV(DY,DDY,NR,ITMAX,TOLER,...,R,...,IRTET)
  → écriture éventuelle d'une routine spécifique XXXCVG du critère de
    convergence (critère relatif par défaut dans LCCONG)
  IF ( IRTET.GT.0 ) GOTO 1

CONVERGENCE -> INCREMENTATION DE  YF = YD + DY
CALL LCSOVN ( NDT+NVI , YD , DY , YF )

MISE A JOUR DE SIGF , VINF
CALL LCEQVN ( NDT ,   YF(1)      , SIGF )
CALL LCEQVN ( NVI-1 , YF(NDT+1) , VINF )
```

En résumé, pour une introduction « complète » d'un nouveau comportement dans PLASTI, il faut nécessairement écrire les routines spécifiques suivantes :

- XXXMAT appelée par LCMATE : récupération du matériau et de la taille du problème local
- XXXCVX appelée par LCCNVX : évaluation du seuil
- XXXRES appelée par LCRESI : calcul du résidu
- XXXJAC appelée par LCJACB : calcul de la jacobienne

Il peut aussi être utile, selon le besoin, d'écrire les routines spécifiques suivantes :

- XXXELA appelée par LCELAS : intégration élastique (si élasticité non-linéaire)
- XXXINI appelée par LCINIT : initialisation (pour une initialisation autre que DY0=0)
- XXXCVG appelée par LCCONV : pour modifier le critère de convergence

5.3 Troisième possibilité : utilisation des routines de l'intégration explicite dans une intégration implicite avec PLASTI

5.3.1 Principe

Ce dernier cas correspond à ALGO_INTE_='NEWTON_PERT', il s'agit de la méthode d'implantation «facile » du nouveau comportement dans l'environnement d'intégration implicite PLASTI. Il est possible d'utiliser directement les deux routines XXXMAT et RKDXXX (récupération des données matériau, et dérivées des variables internes) utilisées avec ALGO_INTE_='RUNGE_KUTTA' pour réaliser un intégration implicite. En effet, le système d'équations différentielles résolu par RUNGE_KUTTA peut s'écrire :

$$\begin{cases}\Delta\boldsymbol{\sigma}=\mathbf{A}(\Delta\boldsymbol{\varepsilon}_i^n-\Delta\boldsymbol{\varepsilon}^{th}-\Delta\boldsymbol{\varepsilon}^p(\boldsymbol{Y})) \\ \dfrac{d\boldsymbol{Y}}{dt}=F(\boldsymbol{Y},t;\boldsymbol{\sigma})\end{cases}$$

où $\boldsymbol{Y}$ représente l'ensemble des variables internes du modèle. La relation entre le tenseur des contraintes et la partie élastique du tenseur des déformations est généralement linéaire, mais peut être évaluée de façon non linéaire par une expression spécifique.

Une fois programmée la routine RKDXXX permettant de calculer $\dfrac{d\boldsymbol{Y}}{dt}=F(\boldsymbol{Y},t;\boldsymbol{\sigma})$, il est possible de l'utiliser pour une intégration implicite, ce qui consiste à résoudre (*cf.* [R5.03.14]) :

$$R(\Delta\boldsymbol{Z})=0=\begin{bmatrix}R_1(\Delta\boldsymbol{Z})\\ R_2(\Delta\boldsymbol{Z})\end{bmatrix}\text{, avec } \Delta\boldsymbol{Z}=\begin{pmatrix}\Delta\boldsymbol{\sigma}\\ \Delta\boldsymbol{Y}\end{pmatrix}=\boldsymbol{Z}(t+\Delta t)-\boldsymbol{Z}(t)$$

- Le premier système d'équations représente la relation contrainte - déformation élastique
  $$R_1(\Delta Z)=\boldsymbol{A}^{-1}\boldsymbol{\sigma}-(\Delta\boldsymbol{\varepsilon}_i^n-\Delta\boldsymbol{\varepsilon}^{th}-\Delta\boldsymbol{\varepsilon}^p(\boldsymbol{Y}))=\boldsymbol{A}^{-1}\boldsymbol{\sigma}-G(\boldsymbol{Y})=0$$
  Par convention, les premières valeurs de $\boldsymbol{Y}$ représentent la variation de déformation plastique, pour faciliter le calcul de $G(\boldsymbol{Y})$ (voir la routine LCRESA pour plus de détails)
- Le deuxième exprime les lois d'évolution des différentes variables internes, soit après discrétisation temporelle par un schéma d'Euler implicite :
  $$R_2(\Delta Z)=\Delta Y-\Delta t\,.\,F(Y,\boldsymbol{\sigma})=0$$

Ce système est résolu par la méthode de Newton proposée dans l'environnement PLASTI et décrite au paragraphe précédent:

$$\begin{cases}\dfrac{\partial R}{\partial\Delta Z}d(\Delta Z_k)=-R(\Delta Z_k)\\ \Delta Z_{k+1}=\Delta Z_k+d(\Delta Z_k)\end{cases}$$

Les quantités G et F intervenant dans le résidu sont calculées par la routine « explicite » RKDXXX à écrire, et le résidu est construit automatiquement par la routine LCRESA. Le matrice jacobienne est calculée automatiquement par perturbation (routine LCJACP). L'opérateur tangent cohérent est quant à lui calculé de manière systématique à partir de la jacobienne (routine LCOPTG, voir [R5.03.14] pour le détail des équations).

En résumé, ce procédé permet, avec les deux seules routines nécessaires à l'intégration explicite, (coefficients matériau et calcul des dérivées des variables internes) d'utiliser une intégration implicite, et de bénéficier d'une matrice tangente. Ce procédé est économique en termes de temps de développement, mais *a priori* moins efficace en temps CPU qu'une matrice jacobienne programmée explicitement.

On détaille ci-dessous le calcul de l'opérateur tangent par perturbation, ainsi que le critère de convergence de l'algorithme de Newton local.

Calcul par perturbation (LCJACP) : différences finies d'ordre 2

- Initialisation de la perturbation : $\eta=10^{-7}\|\Delta Z\|$
- boucle sur les colonnes j de la matrice à remplir :
  - calcul de $R(\Delta Z+\eta I_j)$ avec $I_j=[0\ \ 0\ \ \ldots\ \ 1\ \ \ldots\ \ 0\ \ 0]^T$ vecteur nul sauf à la ligne j

- calcul de $R(\Delta Z-\eta I_j)$
- calcul de la colonne j : $\left[\frac{\partial R}{\partial \Delta Z}\right]_{\ldots,j} \simeq \frac{R(\Delta Z+\eta I_j)-R(\Delta Z-\eta I_j)}{2\eta}$

Critère de convergence (LCCONG) :

On sépare les deux blocs R_1 et R_2 du résidu pour éviter les problèmes dus à des ordres de grandeur différents. A chaque itération k de l'algorithme de Newton local, on calcule :

$err_1=\frac{\left\|R_1(\Delta Z_k)\right\|_\infty}{\left\|R_1(\Delta Z_0)\right\|_\infty}$, avec $R_1(\Delta Z_0)=\Delta \varepsilon_i^n$ car on a $\Delta Z_0=0$ à l'initialisation. (cf. § 5.2)

$err_2=\frac{\left\|R_2(\Delta Z_k)\right\|_\infty}{\left\|Y(t)+\Delta Y_k\right\|_\infty}$

Le critère d'arrêt est alors le suivant :

$max(err_1, err_2)<\xi$, où ξ est donné par RESI_INTE_RELA.

5.3.2 Exemple

Un exemple : la loi visco-élasto-plastique de Hayhurst.

Routine de lecture des coefficients matériau (appelée par la routine d'aiguillage LCMATT) :
- HAYMAT

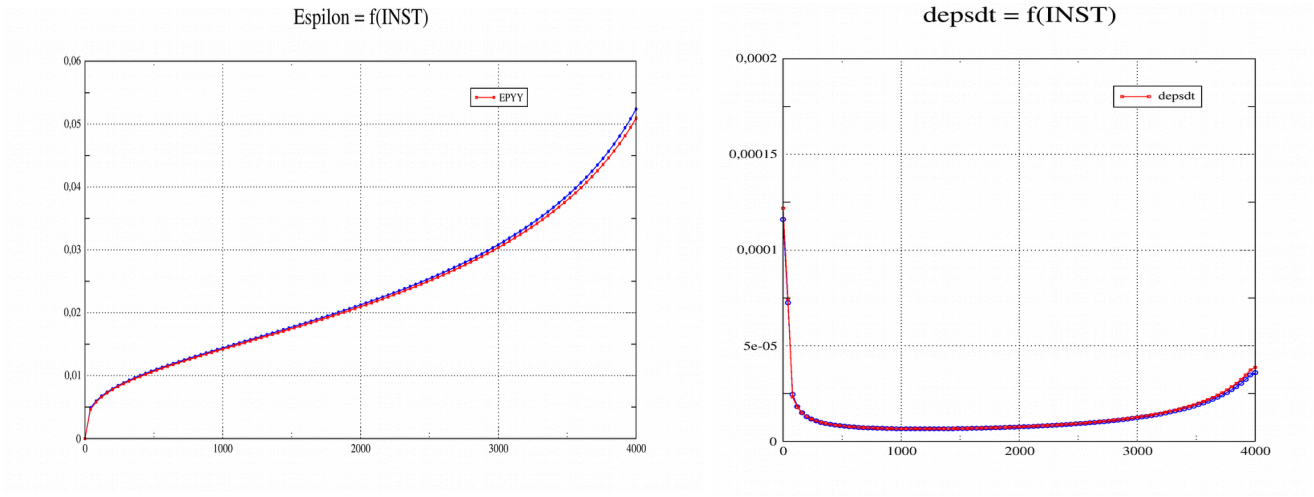
Routine de calcul des dérivées des variables internes (appelée par la routine d'aiguillage LCDVIN) :
- RKDHAY

Les développements explicite / implicite de cette relation de comportement sont testés et comparés dans le cas-test ssnv225 [V6.04.225]. Il s'agit d'un test au point matériel de fluage en grandes déformations permettant de valider les capacités du modèle de HAYHURST à représenter le fluage primaire, secondaire et tertiaire. Voici les caractéristiques d'exécution pour ce test en version 11.2 :

Modélisation A : ALGO_INTE='RUNGE_KUTTA'
- temps CPU : 90.59 s
- Nombre de pas de temps : 1660
- Nombre d'itérations de Newton : 7615

Modélisation B : ALGO_INTE='NEWTON_PERT' :
- temps CPU : 27.38 s
- Nombre de pas de temps : 520
- Nombre d'itérations de Newton : 1404

Les résultats sont quasi identiques :



5.4 Quatrième possibilité : écrire une routine lc00nn autonome

5.4.1 lc00nn : routine relative à un point d'intégration d'un élément, spécifique à une loi de comportement

Chercher dans le répertoire bibfor/algorith un numéro de routine lc00nn non utilisé $(50< nn< 100)$) et partir de cette routine vide.

Remarque :

L'appel à lc00nn par lc0000 (qui est la routine appelant toute les routines d'intégration des comportement disponibles dans code_aster) est déjà écrit. Cependant, il faut s'assurer que les arguments (ainsi que l'ordre de ces arguments) choisis à la déclaration de lc00nn par le développeur soient les mêmes qu'à l'appel dans lc0000.

Les arguments d'entrée d'une routine lc00nn sont *a minima* :

FAMIfamille de points de gauss (RIGI,MASS,...)
KPG,KSP numéro du point de gauss et du sous-point
NDIM dimension de l espace (3d=3 , 2d=2 , 1d=1)
IMATE adresse du matériau
COMPOR infos sur le comportement
compor(1) = relation de comportement (vmis_cine_...)
compor(2) = nombre de variables internes
compor(3) = type de déformation (petit,green...)
CRIT critères locaux
crit(1) = nombre d itérations maxi a convergence (ITER_INTE_MAXI)
crit(3) = valeur de la tolérance de convergence (RESI_INTE_RELA)
INSTAM instant t-
INSTAP instant t = t- + dt
EPSM déformation totale a t- (ou éventuellement le gradient de transformation suivant le type de déformation : les arguments de la routine appelante, lc0000, contiennent la dimension de EPSM, et DEPS),
DEPS incrément de déformation totale (même remarque)
SIGM contrainte a t-
VIM variables internes a t-
OPTION option de calcul
RIGI_MECA_TANG -> dsidep(t)
FULL_MECA -> dsidep(t+dt) , sig(t+dt)

RAPH_MECA -> sig(t+dt)
ANGMAS les trois angles (nautiques ou d'Euler) du mot_clef massif
TYPMOD type de modélisation : 3D, AXIS, D_PLAN, C_PLAN,...
ICOMP compteur pour le redécoupage local du pas de temps
NVI nombre de variables internes du comportement

les arguments de sortie sont, selon l'option de calcul :

VIP variables internes a l'instant actuel (options RAPH_MECA et FULL_MECA)
SIGP terme d'ordre 0 dans la linéarisation de la contrainte (option RIGI_MECA_TANG) – cf. §4
contraintes a l'instant actuel (options RAPH_MECA et FULL_MECA)
DSIDEP opérateur tangent cohérent ou en vitesse (option FULL_MECA ou RIGI_MECA_TANG).
CODRET code retour permettant d'indiquer (s'il est non nul) une problème d'intégration locale, donc d'effectuer un redécoupage du pas de temps

Remarques :

- *Le cas échéant, il est possible d'utiliser également un type dérivé en entrée (variable BEHinteg dans LC0000/LCnnnn). Ce type contient des arguments supplémentaires, par exemple une longueur caractéristique dans le cas des modèles non locaux...*
- *De même, il est possible de transférer des arguments supplémentaires en sortie de la routine LCnnnn/LC0000 via la variable BEHinteg.*
- *Attention, dans le cas RIGI_MECA_TANG, les tableaux relatifs aux contraintes et variables internes en fin de pas de temps ne sont pas alloués. Il ne faut donc pas les utiliser pour calculer la matrice tangente de prédiction.*

5.4.2 Organisation de la routine à écrire

On prend comme au § 3.3 l'exemple de la loi de Von Mises à écrouissage cinématique linéaire VMIS_CINE_LINE (num_lc=3 dans vmis_cine_line.py) :

```
 SUBROUTINE LC0003(FAMI,KPG,KSP,NDIM,IMATE,COMPOR,CRIT,INSTAM,
&                  INSTAP,EPSM,DEPS,SIGM,VIM,OPTION,ANGMAS,SIGP,VIP,
&                  TAMPON,TYPMOD,ICOMP,NVI,DSIDEP,CODRET)
```

Cette routine est en fait une routine d'aiguillage pour les comportements VMIS_CINE_LINE et VMIS_ECMI_*. L'intégration de VMIS_CINE_LINE est réalisée dans la routine NMCINE, appelée par lc0003 et dont le contenu est décrit ici brièvement.
On notera que dans cet exemple, la loi suit le mécanisme deform_ldc = 'OLD', et qu'à ce titre, les déformations thermiques sont calculées et retranchées à l'intérieur de la loi (cf. §4).

- **LECTURE DES CARACTERISTIQUES ELASTIQUES DU MATERIAU (TEMPS T = +)**

```
 NOMRES(1)='E'
 NOMRES(2)='NU'
 CALL RCVALB(FAMI,KPG,KSP,'+',IMATE,' ','ELAS',0,' ',
&            0.D0,2,NOMRES,VALRES,ICODRE,2)
 E  = VALRES(1)
 NU = VALRES(2)
```

Remarque :

RCVALB est une routine générale permettant d'interpoler les valeurs des coefficients matériaux par rapport aux variables de commandes dont ils dépendent (voir les utilitaires).

RCVARC est une routine qui permet de récupérer la valeur d'une variable de commande (température, séchage , irradiation, ...) à l'instant considéré, et au point de Gauss considéré (voir les utilitaires). Exemple :

```
CALL RCVARC(' ','TEMP','-',FAMI,KPG,KSP,TM,IRET)
CALL RCVARC(' ','TEMP','+',FAMI,KPG,KSP,TP,IRET)
```

- **LECTURE DES CARACTERISTIQUES D'ECROUISSAGE**

```
 NOMRES(1)='D_SIGM_EPSI'
 NOMRES(2)='SY'
 CALL RCVALB(FAMI,KPG,KSP,'+',IMATE,' ','ECRO_LINE',0,' ',
&            0.D0,2,NOMRES,VALRES,ICODRE, 2)
 DSDE=VALRES(1)
 SIGY=VALRES(2)
 C  = 2.D0/3.D0*DSDE/(1.D0-DSDE/E)
```

- **CALCU L DES CONTRAINTES ELASTIQUES ET DU CRITERE DE VON MISES**

```
      DO 110 K=1,3
        DEPSTH(K)   = DEPS(K) -EPSTHE
        DEPSTH(K+3) = DEPS(K+3)
110   CONTINUE
      EPSMO = (DEPSTH(1)+DEPSTH(2)+DEPSTH(3))/3.D0
      DO 115 K=1,NDIMSI
        DEPSDV(K)   = DEPSTH(K) - EPSMO * KRON(K)
115   CONTINUE
      SIGMO = (SIGM(1)+SIGM(2)+SIGM(3))/3.D0
      SIELEQ = 0.D0
      DO 114 K=1,NDIMSI
        SIGDV(K) = SIGM(K) - SIGMO*KRON(K)
        SIGDV(K) = DEUXMU/DEUMUM*SIGDV(K)
        SIGEL(K) = SIGDV(K) + DEUXMU * DEPSDV(K)
        SIELEQ   = SIELEQ   + (SIGEL(K)-C/CM*VIM(K))**2
114   CONTINUE
      SIGMO = TROISK/TROIKM * SIGMO
      SIELEQ = SQRT(1.5D0*SIELEQ)
      SEUIL  = SIELEQ - SIGY
      DP = 0.D0
      PLASTI=VIM(7)
```

- **CALCUL DES CONTRAINTES ET DES VARIABLES INTERNES**

Les expressions des contraintes et des variables internes (RAPH_MECA et FULL_MECA) sont données dans [R5.03.02]

```
      IF ( OPTION(1:9) .EQ. 'RAPH_MECA' .OR.
     &     OPTION(1:9) .EQ. 'FULL_MECA' ) THEN
        IF (SEUIL.LT.0.D0) THEN
          VIP(7) = 0.D0
          DP = 0.D0
          SIELEQ = 1.D0
          A1 = 0.D0
          A2 = 0.D0
        ELSE
          VIP(7) = 1.D0
          DP = SEUIL/(1.5D0*(DEUXMU+C))
          A1 = (DEUXMU/(DEUXMU+C))*(SEUIL/SIELEQ)
          A2 = (C     /(DEUXMU+C))*(SEUIL/SIELEQ)
```

```
        ENDIF
        PLASTI=VIP(7)
        DO 160 K = 1,NDIMSI
          SIGDV(K) = SIGEL(K) - A1*(SIGEL(K)-VIM(K)*C/CM)
          SIGP(K)  = SIGDV(K) + (SIGMO + TROISK*EPSMO)*KRON(K)
          VIP(K)   = VIM(K)*C/CM   + A2*(SIGEL(K)-VIM(K)*C/CM)
160     CONTINUE
      ENDIF
```

- **CALCUL DE L'OPERATEUR TANGENT 'DSIPSEP' : `RIGI_MECA_TANG` (en vitesse) ou `FULL_MECA` (cohérent)**

```
      IF ( OPTION(1:14) .EQ. 'RIGI_MECA_TANG' .OR.
     &     OPTION(1:9)    .EQ. 'FULL_MECA' ) THEN
        CALL MATINI(6,6,0.D0,DSIDEP)
        DO 120 K=1,6
          DSIDEP(K,K) = DEUXMU
120     CONTINUE
        IF ( OPTION(1:14) .EQ. 'RIGI_MECA_TANG') THEN
          DO 174 K = 1,NDIMSI
            SIGDV(K) = SIGDV(K) - VIM(K)*C/CM
174       CONTINUE
        ELSE
          DO 175 K = 1,NDIMSI
            SIGDV(K) = SIGDV(K) - VIP(K)
175       CONTINUE
        ENDIF
        SIGEPS = 0.D0
        DO 170 K = 1,NDIMSI
          SIGEPS = SIGEPS + SIGDV(K)*DEPSDV(K)
170     CONTINUE
        A1 = 1.D0/(1.D0+1.5D0*(DEUXMU+C)*DP/SIGY)
        A2 = (1.D0+1.5D0*C*DP/SIGY)*A1
        IF(PLASTI.GE.0.5D0.AND.SIGEPS.GE.0.D0) THEN
          COEF = -1.5D0*(DEUXMU/SIGY)**2 / (DEUXMU+C) * A1
          DO 135 K=1,NDIMSI
            DO 135 L=1,NDIMSI
              DSIDEP(K,L) = A2 * DSIDEP(K,L) + COEF*SIGDV(K)*SIGDV(L)
135       CONTINUE
          LAMBDA = LAMBDA + DEUXMU**2*A1*DP/SIGY/2.D0
        ENDIF
        DO 130 K=1,3
          DO 131 L=1,3
            DSIDEP(K,L) = DSIDEP(K,L) + LAMBDA
131       CONTINUE
130     CONTINUE
      ENDIF
```

5.4.3 Variables de commande (external state variables)

La dépendance des paramètres matériau aux variables de commande (séchage, température, etc.) est prise en compte par l'usage de la routine RCVALB . Il existe en sus des variables de commandes spécifiques à certaines lois de comportement qui doivent être ***calculées*** (en général au niveau de l'élément) et non transmises par le mécanisme AFFE_MATERIAU/AFFE_VARC , il s'agit de :

- ELTSIZE1 : calcul de la taille d'un élément (pour la loi BETON_DOUBLE_DP) ;
- ELTSIZE2 : calcul de la taille d'un élément (pour la loi ENDO_PORO_BETON) ;
- COORGA : coordonnées des point de Gauss de l'élément (pour la loi META_LEMA_ANI) ;
- GRADVELO : pour le gradient de la vitesse de déformation (pour la loi MONOCRISTAL) ;
- HYGR : hygrométrie (à partir de la fonction de désorption FONC_DESORP).

Les lois de comportement utilisant ces variables utilisent le mot-clef exte_vari dans leur catalogue. L'ajout d'une autre variable de ce style nécessite un développement spécifique qui ne peut donc pas être détaillé ici.
Par contre, on peut utiliser les variables définies plus haut. Il faut :

- impacter le catalogue de la loi de comportement et ajouter cette variable dans exte_vari ;
- récupérer la valeur de cette variable via le module calcul

Variables	Variable(s) dans le module calcul
HYGR	ca_vext_hygrm_ ca_vext_hygrp_
ELTSIZE1	ca_vext_eltsize1_
ELTSIZE2	ca_vext_eltsize2_(9)
COORGA	ca_vext_coorga_(27,3)
GRADVELO	ca_vext_gradvelo_(9)

Par exemple :

```
use calcul_module, only : ca_vext_gradvelo_

real (kind=8) :: l(3,3)
integer :: i, j

    do i = 1, 3
        do j = 1, 3
            l(i,j)=ca_vext_gradvelo_(3*(i-1)+j)
        end do
    end do
```

6 Cas particulier des lois MFront

En complément du document [u2.10.02] qui décrit comment utiliser une loi de comportement MFront en mode prototype, on précise dans ce paragraphe les particularités des lois de comportement MFront qui sont officiellement (sous assurance qualité) intégrées à code_aster.

Le nom du fichier MFront doit être identique au nom du comportement décrit dans le fichier.

Quand on compile les lois de comportement MFront officielles, la procédure de construction doit savoir quels fichiers sont produits par la conversion du fichier MFront en C++. Habituellement, deux fichiers sont produits : un du nom du comportement et un nommé aster + nom du comportement.
Si un autre fichier est produit, il faut l'indiquer dans le fichier MFront par exemple (dans PlasticityTH.mfront) :

```
//output Acier_ElasticYield-mfront
```

6.1 Catalogue pour DEFI_MATERIAU et RELATION

Le catalogue de commande des paramètres matériaux est automatiquement intégré dans la commande DEFI_MATERIAU par la procédure de description (lors de l'étape waf install) :

- le mot-clé facteur est le nom du comportement (exemple : AnisoLemaitre),
- le nom des paramètres matériaux correspond aux @MaterialProperty.

Si les paramètres sont des tableaux (exemple : @MaterialProperty real a[3]), ils seront nommés a_0, a_1 et a_2.
Un deuxième mot-clé facteur suffixé par _FO (exemple : AnisoLemaitre_FO) est créé dans lequel tous les paramètres sont des fonctions.

Le nom du comportement est automatiquement ajouté pour le mot-clé RELATION.

6.2 Catalogue du comportement

La description du comportement est légèrement différente. Il faut créer un objet de type LoiComportementMFront et indiquer le nom du symbole produit lors de la compilation.

7 Cas des lois de comportement métallurgique (CALC_META)

On considère ici les lois de comportement permettant de calculer l'état métallurgique dans l'opérateur CALC_META et non les lois de comportement mécaniques utilisant ces phases.

Une loi de comportement métallurgique est constituée de deux parties :
- les phases métallurgiques sur laquelle elle agit (ACIER ou ZIRC) ;
- la loi de comportement métallurgique proprement dite.

L'ajout d'un nouveau type de phase est très impactant :
- impact dans CALC_META et ses périphériques ;
- impact dans AFFE_MATERIAU (pour les variables de commande AFFE_VARC) ;
- impact dans les lois de comportement mécanique utilisant la métallurgie.

7.1 Modification du catalogue de commande

On peut ajouter un nouveau modèle de loi de comportement métallurgique en le liant aux phases (ACIER ou ZIRC) dans le mot-clef facteur COMPORTEMENT de CALC_META. Il faut impacter le mot-clef simple LOI_META.

7.2 Modification du catalogue de comportement

Une loi de comportement métallurgique a son catalogue (dans bibpyt/Comportement) qui ne contient que peu d'informations (par rapport aux lois de comportement mécaniques). Par exemple pour la loi WAECKEL :

```
loi = LoiComportement(
    nom             = 'WAECKEL',
    lc_type         = ('MODELE_METALLURGIQUE',),
    doc             = """Modèle métallurgique standard pour l'acier""",
    num_lc          = 2,
    nb_vari         = 3,
    nom_vari        = ('TAILLE_GRAIN','TEMP','TEMP_MARTENSITE',
        ),
    mc_mater        = ('META_ACIER',),
    modelisation    = ('3D','AXIS','D_PLAN',),
)
```

On ne peut pas modifier modelisation (car la métallurgie agit toujours sur ces trois modélisations). Le reste se modifie comme dans le §3.3 .

Une loi de comportement métallurgique est composée d'un modèle (LOI_META) et d'une phase (RELATION).
Le nombre et le nom des variables internes sont liés par un système de KIT. La phase donnera la liste des phases standards (acier ou Zircaloy), le type permettra éventuellement d'ajouter d'autres "variables internes" (typiquement aujourd'hui, pour le modèle de Waeckel, on a la température, le temps de transformation, la taille des grains, etc).

Pour déterminer la routine appelée pour le calcul, c'est le même principe que pour les lois de comportement mécaniques, ce sera l'addition de la nature des phases (acier/Zircaloy) et du type de modèle utilisé.

Le mécanisme est de type `nzcomp.F90+lzxxxx.F90` (comme `nmcomp.F90+lcxxxx.F90`).

- Acier : `num_lc = 20000`
- Zircaloy : `num_lc = 30000`

Le modèle de Wackel a `num_lc = 2`, donc la loi de comportement métallurgique de l'acier avec le modèle de Waeckel appelera la routine `lz20002.F90`

8 Validation et maintenance

La réflexion sur les tests de validation et d'identification peut être amorcée très tôt, avant même le développement proprement dit. On peut distinguer :

- les fichiers de commande (souvent simple, utilisant par exemple `SIMU_POINT_MAT`) permettant d'identifier certains paramètres (à l'aide de `MACR_RECAL`).
- les tests à restituer : il doivent permettre de valider le comportement dans tous ses aspects, pour une gamme de valeur de paramètres couvrant correctement le domaine de validité. Certains tests sont quasiment obligatoires :
- les tests de robustesse et d'invariance par rapport à une rotation, un changement d'unité (COMP001, COMP002, COMP003)
- les tests vérifiant la bonne prise en compte des variables de commandes (COMP008, COMP010)

De plus le développeur peut être amené à corriger ou analyser son comportement suite à une anomalie rencontrée lors d'une étude. Dans ce cas, en plus des techniques habituelles de débogage, il est possible d'utiliser une fonctionnalité mise en œuvre en cas d'échec d'intégration du comportement : les premiers points en échec produisent un petit fichier de commandes permettant de rejouer la scène avec `SIMU_POINT_MAT`, pour mieux analyser le problème, essayer d'autres méthodes. Il suffit de recopier la définition du matériau ; toutes les autres données (contraintes, déformations, variables internes initiales, et incrément de déformation permettent de simuler le comportement à l'instant et au point où l'échec s'est produit. Par exemple pour le test forma03c :

```
#-----------------------------------------------------------------------
# test pour analyser l'échec d'intégration sur la maille <M83>, point <3>
#-----------------------------------------------------------------------
DEBUT()

# recopier DEFI_MATERIAU(...)
# COURBE DE TRACTION
CTRAC = LIRE_FONCTION(UNITE=21,NOM_PARA='EPSI',PROL_DROITE='CONSTANT',)

MAT=DEFI_MATERIAU(ELAS=_F(E=200000.,NU=0.3),TRACTION=_F(SIGM=CTRAC))

LIST=DEFI_LIST_REEL(DEBUT=1.944000000000000E+02 ,
                    INTERVALLE=_F(JUSQU_A= 2.187000000000000E+02 , NOMBRE=1))

DEFLIST = DEFI_LIST_INST(  DEFI_LIST=_F(LIST_INST=LIST,),
                           ECHEC=_F(SUBD_NIVEAU=10,SUBD_PAS=4),)

EXX=DEFI_FONCTION(NOM_PARA='INST',
                  VALE=( 1.944000000000000E+02 , -9.407813329102166E-04,
                         2.187000000000000E+02,7.373776084156800E+06))

EYY=DEFI_FONCTION(NOM_PARA='INST',
                  VALE=( 1.944000000000000E+02 , 1.718362911427018E-04,
                         2.187000000000000E+02, -5.221483651275956E+06))
```

```
EZZ=DEFI_FONCTION(NOM_PARA='INST',
                  VALE=(1.944000000000000E+02, 0.000000000000000E+00,
                        2.187000000000000E+02,-9.224110429927666E+05))

EXY=DEFI_FONCTION(NOM_PARA='INST',
                  VALE=( 1.944000000000000E+02 , 3.621278829822514E-04,
                        2.187000000000000E+02, -4.710098874951571E+06))

RESU=SIMU_POINT_MAT ( INFO=1, MATER=MAT, INCREMENT=_F(LIST_INST=DEFLIST),
                      EPSI_IMPOSE=_F(EPXX=EXX, EPYY=EYY,EPZZ=EZZ,EPXY=EXY),
                      SUPPORT='ELEMENT', MODELISATION='C_PLAN',
                      EPSI_INIT=_F(EPXX=-9.407813329102166E-04,
                                   EPYY=1.718362911427018E-04,
                                   EPZZ= 0.000000000000000E+00,
                                   EPXY=3.621278829822514E-04,
                                   EPXZ=0.000000000000000E+00,
                                   EPYZ= 0.000000000000000E+00 ),
                      SIGM_INIT=_F(SIXX=-1.887253339740370E+02,
                                   SIYY=-2.497632316081525E+01,
                                   SIZZ= 0.000000000000000E+00,
                                   SIXY=7.537194347798921E+01,
                                   SIXZ=0.000000000000000E+00,
                                   SIYZ= 0.000000000000000E+00 ),
                      VARI_INIT=_F(VALE=(3.931095545774485E-05,
                                         1.000000000000000E+00,)),
                      COMPORTEMENT=_F(RELATION='VMIS_ISOT_TRAC',ITER_INTE_MAXI=20,),
                      NEWTON=_F(REAC_ITER=1),)

FIN()
```