# Intégration d'une approche variationnelle pour la méthode des éléments finis dans un environnement orienté-objet

## Application à un problème de convection non linéaire

**Dominique Eyheramendy — Thomas Zimmermann**

*Laboratoire de mécanique des structures et milieux continus*
*Ecole Polytechnique Fédérale de Lausanne*
*DGC-A2, Ecublens*
*CH-1015 Lausanne*
*Dominique.Eyheramendy@lsc.dgc.epfl.ch*

RÉSUMÉ. *L'élaboration et la maintenance des logiciels de calcul numérique représentent aujourd'hui encore un enjeu important. L'utilisation d'outils informatiques de haut niveau d'abstraction permet de pallier certaines lacunes des approches traditionnelles. Ainsi, logiciels de calculs symboliques et paradigmes de programmation modernes ouvrent de nouvelles voies dans les stratégies de résolution de problèmes aux équations aux dérivées partielles. Dans cette optique, le groupe de recherche des auteurs a proposé dans un premier temps une organisation de code numérique pour la méthode des éléments finis basée sur des concepts orientés-objets ; plus récemment, cette approche a été étendue aux approches variationnelles. On se propose d'illustrer ici cette nouvelle démarche sur un exemple de résolution d'un problème de convection non linéaire unidimensionnel.*

ABSTRACT. *Today, elaboration and maintenance of computational software remain challenging problems. The use of software tools with high level helps to overcome weaknesses of more traditional approaches. Symbolic manipulations software and modern programming paradigms open the door to new strategies for the solution of initial-boundary value problems. In earlier work, the authors' research group has proposed an approach to finite element programming based on object-oriented concepts; more recently, this approach has been extended to variational formulations. Our purpose in this paper is to give an illustration of the proposed strategy on the example of a one-dimensional nonlinear advection equation.*

MOTS-CLÉS : *formulations variationnelles, problèmes non linéaires, méthode éléments finis, programmation orientée-objet, calcul symbolique, calcul numérique.*

KEY WORDS: *variational formulations, nonlinear problems, finite element method, object-oriented programming, symbolic computation, numerical computation.*

Revue européenne des éléments finis. Volume 7 - n° 5/1998, pages 589 à 607

## 1. Introduction

### 1.1. *Calcul symbolique et approche orientée-objet*

Les outils informatiques modernes permettent d'aborder les problèmes de calcul numérique dans le domaine de la mécanique (fluide, structure,...) de façon nouvelle. En particulier, les outils pour le calcul symbolique permettent d'accélérer, de systématiser et d'améliorer le développement des codes numériques. Ainsi, depuis l'avènement de la méthode des éléments finis (MEF) de nombreux travaux ont cherché à tirer parti des approches symboliques pour l'élaboration des codes numériques. On peut scinder ces travaux en trois catégories. La première utilise les capacités de manipulations symboliques d'un environnement tel que Mathematica ou Mapple pour y intégrer un code numérique classique ; on effectue alors les calculs dans ces systèmes en conservant des variables sous forme symbolique (voir par exemple [CHO 92, IOA 93]). La deuxième n'utilise que la puissance de calcul de ces environnements afin de simplifier et d'optimiser les formes matricielles élémentaires de la MEF, avant de les introduire dans un code EF classique (par exemple [YAN 94, SIL 94]). Enfin une troisième catégorie génère un environnement pour créer automatiquement les formes symboliques, en s'aidant ou non d'un environnement de calcul tel que REDUCE, et en intégrant les formes symboliques dans un code éléments finis classique de façon plus ou moins automatique (par exemple [GUN 71, HOA 80, CEC 77, LUF 71, NOO 79, LEF 91, WAN 86]). Notre démarche peut être considérée comme une systématisation du second type d'approche. Plus récemment et parallèlement à ces approches, ont été développées des méthodologies orientée-objets (OO) de conception de codes éléments finis, par exemple dans [FOR 90, MIL 91, REH 89, ZIM 92a, DUB 92, MAC 92, SCH 92]. Depuis, de nombreux développements tirant parti de concepts OO ont été réalisés (cf. à titre d'exemples [DEV 92, MEN 93, CAR 94, DUB 95, GEL 95, FOE 96, DUB 97, BES 97, POT 97]).

Initialement dans [EYH 94], puis dans [EYH 95, ZIM 96], le groupe de recherche des auteurs a tiré parti des approches symboliques dans le contexte de code EF orientés-objets. Dans un premier temps, les concepts OO ont été étendus à l'approche variationnelle qui précède la modélisation d'éléments finis linéaires, ce qui a permis d'obtenir un environnement OO pour la dérivation symbolique de la MEF et l'écriture automatique de code EF. Ainsi, élaboration de formulation EF et environnement de calcul EF se trouvent complètement intégrés dans un même environnement OO, au sein duquel le développeur évolue plus naturellement. L'environnement symbolique FEM_Theory ([EYH 96, EYH 97b]) intégré dans l'environnement OO Smalltalk (cf. [VIS 95a, 95b]) permet ainsi de générer du code numérique pour le code EF FEM_Object, en Smalltalk ou en C++. L'avantage du premier modèle est d'intégrer calculs symboliques et numériques au sein de l'environnement Smalltalk, le deuxième lui permet d'atteindre un niveau d'efficacité numérique plus intéressant. On se propose, dans cet article, d'illustrer ces concepts dans le cadre de formulations non linéaires.

### 1.2. *Application aux formulations non linéaires*

Le but de cette article est de définir un cadre adéquat pour des formulations non linéaires dans l'environnement symbolique FEM_Theory [EYH 94]. Ainsi, on propose d'illustrer, sur un exemple simple, la démarche permettant d'introduire dans l'environnement symbolique les concepts nécessaires pour la résolution de problèmes non linéaires.

Un environnement capable de représenter différents types de formulations variationnelles linéaires est décrit dans différents articles [ZIM 96, EYH 96]. Dans [EYH 97b], un cadre de développement pour des formulations linéaires de type mixte ou à caractère convectif dominant, et de type spatio-temporelles discontinues en temps a été mis en place. Dans cette continuité, on se propose d'aborder ici certains problèmes non linéaires.

La représentation de formulations non linéaires dans FEM_Theory ne pose pas de problème particulier. Cela peut être réalisé soit en ajoutant de nouveaux types de termes (objets **Term**), correspondant à des opérateurs non linéaires (de type « géométrique » ou « matériel »), ou en manipulant des produits d'opérateurs linéaires. L'approche choisie, en ce qui concerne cette étude, est l'introduction de techniques de linéarisation. Le problème linéarisé peut ensuite être résolu par une procédure itérative de type Newton-Raphson.

Dans le paragraphe 2, sont présentés les éléments théoriques que l'on souhaite introduire dans l'environnement FEM_Theory. Dans le paragraphe 3, leur implantation est décrite. Ceci est complété dans le paragraphe 4 dans lequel la généralisation de la procédure est commentée. Cette approche est enfin testée dans le paragraphe 5 sur la dérivation d'une formulation espace/temps discontinue en temps pour un problème de convection non linéaire unidimensionnel.

## 2. Une approche théorique pour la linéarisation

Dans ce paragraphe, on rappelle certains aspects théoriques liés à la linéarisation en vue de l'implantation de concepts liés à la linéarisation dans l'environnement symbolique FEM_Theory. Les fondements théoriques se trouvent dans [MAR 83 Chap. 4], et [HUG 78] pour une approche plus pragmatique.

### 2.1. *Définition de la linéarisation*

Soit une fonctionnelle $f$, $u \mapsto f(u)$, ayant des conditions de continuité suffisantes. La partie linéaire de $f$ à $u$ peut être obtenue en utilisant la formule d'expansion de Taylor tronquée au premier ordre :

$$L_u f(\delta u) = f(u) + Df(u) \cdot \delta u \qquad [1]$$

Le problème revient alors à trouver une dérivée adéquate afin de calculer le terme gradient $Df(u) \cdot \delta u$ de la partie linéaire de $f$. D'une manière générale, la dérivée directionnelle peut être utilisée afin de mener à bien la linéarisation.

REMARQUE. — $\delta u$ est la direction de recherche, ou incrément.

### 2.2. *Dérivée directionnelle*

*Définitions*

Etant donnée une fonctionnelle $f$ : $u \mapsto f(u)$, ayant des conditions de continuité suffisantes. La dérivée directionnelle de $f$ en $u$ dans la direction $\delta u$ est définie par :

$$\frac{d}{d\varepsilon} f(u + \varepsilon\, \delta u) \Big|_{\varepsilon=0} \qquad [2]$$

où $\varepsilon$ est un paramètre scalaire.

La dérivée directionnelle mesure le taux de variation de la fonctionnelle $f$ dans la direction $\delta u$ au point $u$. Elle peut être utilisée pour calculer le terme gradient du premier ordre de la partie linéaire de $f$ :

$$Df(u) \cdot \delta u = \frac{d}{d\varepsilon} f(u + \varepsilon\, \delta u) \Big|_{\varepsilon=0} \qquad [3]$$

Ceci permet de mener à bien la linéarisation consistante de nombreux problèmes. Des détails supplémentaires à propos de la dérivée directionnelle peuvent être trouvés dans [HUG 78] ; des exemples de son utilisation dans le domaine de la mécanique des structures peuvent être trouvés par exemple dans [KOZ 94, IBR 93].

*Propriété*

La dérivée directionnelle est linéaire.

Ainsi, pour $f$ et $g$ deux fonctionnelles données, satisfaisant à des conditions de continuité suffisantes, on a :

$$D(f.g)(u) \cdot \delta u = Df(u).\delta u\; g(u) + f(u)\; Dg(u) \cdot \delta u \qquad [4]$$

$$D(f + g)(u) \cdot \delta u = Df(u).\delta u + Dg(u) \cdot \delta u \qquad [5]$$

Cette propriété va être utilisée dans l'environnement symbolique.

REMARQUE. — La linéarisation d'une formulation variationnelle conduit généralement à l'utilisation d'algorithmes de type Newton. Une convergence quadratique de l'algorithme peut être envisagée si la linéarisation est consistante.

## 3. Concepts orientés-objet pour la dérivation symbolique de problèmes non linéaires

### 3.1. *Les objets nécessaires pour un schéma de linéarisation consistant dans FEM_Theory*

Deux concepts importants ont été introduits dans le paragraphe précédent. Le premier est une définition pour la partie linéaire d'une fonctionnelle. Le second est une méthode de calcul pratique pour le terme gradient du premier ordre, à savoir la dérivée directionnelle. Ces définitions théoriques et propriétés s'implantent naturellement dans l'environnement symbolique.

Un nouvel objet apparaît dans cette description : l'incrément ou direction de recherche. Le fait de le considérer en tant que incrément, permet de le manipuler comme un terme classique. Ainsi, il peut être implanté en tant que sous-classe de la classe Term. Un exemple de cet objet est représenté dans la figure 1. Il faut noter que cet objet est utilisé aussi bien pour la formulation continue, que pour la formulation discrète.

Classe **Increment**
Exemple: $\delta u$
***principaux attributs :***
- function : $u$
***principales tâches :***
- manipulations dans la forme variationnelle telles que la somme et le produit (hérité de la classe **Term**), l'analyse, la discrétisation

**Figure 1.** *Une instance de la classe Increment*

### 3.2. *Implantation dans FEM_Theory*

#### 3.2.1. *La procédure de linéarisation*

La procédure de linéarisation est appliquée à un objet équation, instance de la classe **IntEquation**. Ainsi, le problème permettant de trouver $u$ tel que $f(u)=0$ est remplacé par un problème approché où il s'agit de trouver $u$ tel que $L_u f(\delta u) = f(u) + Df(u)\cdot\delta u = 0$. Le code correspondant à cette opération est montré dans la figure 2. Une nouvelle instance de la classe **IntEquation** est créée ; le membre de gauche (attribut **lhs**) de cette équation est instancié par l'appel à la méthode *consistentLinearization*. Celle-ci renvoie le résultat de la linéarisation du membre de gauche de l'équation courante. Comme on peut le voir sur un exemple d'équation sur la figure 4, le message est envoyé au membre de gauche de l'équation, qui est une fonctionnelle (instance de la classe **Functional**). La méthode correspondante est montrée en figure 3. On peut remarquer que cette méthode n'est que la transcription exacte de la définition de la linéarisation donnée au paragraphe

2.1 (équation [1]). Le message *computeDirectionalDerivative* parcourt l'arborescence hiérarchique de l'objet comme le montre la figure 4. On peut noter que les propriétés liées à la linéarité de la dérivée directionnelle sont implantées dans les classes **SumList** et **ProdList**. L'implantation de ces concepts demeure très proche des définitions théoriques. Elle est parfaitement générique et ouvre la porte aux futurs développements.

```
computeConsistentLinearization
    | directionalDerivative consistentLinearization reply |

    self putAllLhs expand.

    consistentLinearization := (self giveLhs) computeConsistentLinearization.

    reply := IntEquation new
                    lhs: consistentLinearization ;
                    rhs: ('0' formYourExpression);
                    listOfTerms: listOfTerms;
                    listOfUnknowns: listOfUnknowns;
                    spaceDimension: spaceDimension;
                    problemDimension: problemDimension;
                    problemType: problemType.
    ^ reply
```

**Figure 2.** *Méthode computeConsistentLinearization de la classe IntEquation*

```
computeConsistentLinearization


                ^ (self +(self getDirectionalDerivative))
```

**Figure 3.** *Méthode computeConsistentLinearization de la classe Functional*

### 3.2.2. *Classe* ***Increment***

La classe est décrite dans la table 1.

*Attributs*

La classe **Increment** a un attribut nommé **function**. Cette classe est utilisée aussi bien pour le problème continu que pour le problème discret ; par conséquent, les méthodes pour les deux modèles sont regroupées au sein de la même classe. L'attribut **function** peut donc *a priori* être une instance de **Term** ou de **DiscretizationMatrix**.

*Description des tâches*

Les tâches liées à l'intégration dans la structure arborescente des objets symboliques (cf. figure 4) et aux manipulations algébriques sont héritées des super-classes. L'ensemble des tâches peut être décomposé en trois groupes. La plupart des méthodes sont en fait des spécialisations de méthodes existant dans la classe **Term**.

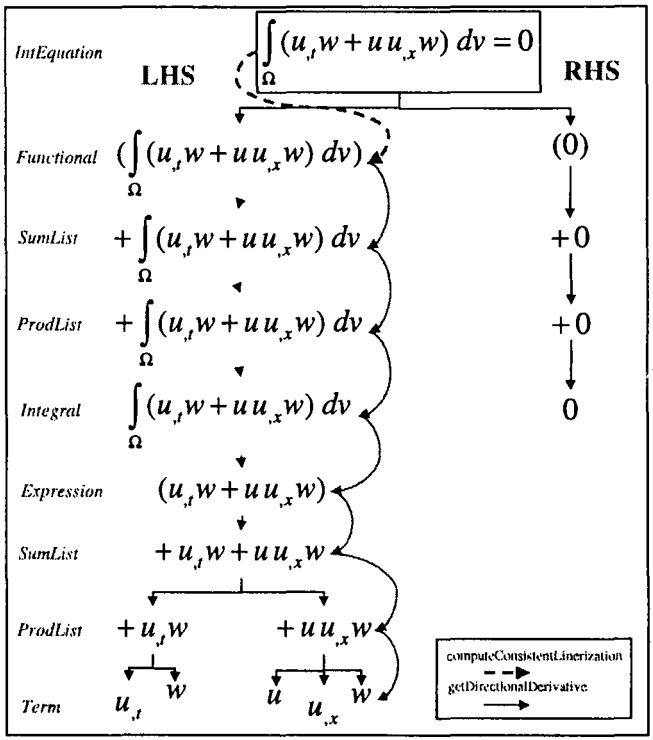



**Figure 4.** *Procédure de linéarisation sur une fonctionnelle*

1. Méthodes de manipulation : la plupart des tâches sont décentralisées vers l'attribut **function**.
2. Méthodes d'analyse : comme auparavant, les tâches d'analyse sont des spécialisations de la classe **Term**. Cette partie du comportement peut être reliée à celui d'analyse.
3. Discrétisation/approximation : les tâches liées à l'approximation sont décentralisées vers l'attribut **function** ; de ce fait, le schéma permettant de construire les contributions élémentaires ne nécessite aucune implantation spécifique (cf. [EYH 96] pour les détails). Le résultat est le produit (instance de **ProdList**) de la forme matricielle correspondant à l'attribut **function** et du vecteur de l'incrément de ses inconnues nodales.

| **Classe Increment**<br>**Hérite des classes : Term,StructureWithDimension, FEMTheory, Object** | | |
|---|---|---|
| **Tâches héritées** | **Attributs hérités** | **Méthodes héritées** |
| 1) accès aux données de hierarchic parent | hierarchicParent | getDiscretizationInfosForTerm: term<br>getListOfTerms<br>giveHierarchicParent<br>giveSpaceDimension<br>knowsAsUnknow: term |
| 2) partie de la classe **Term** | Attributs de la classe **Term** | |
| **Tâches** | **Attributs** | **Méthodes** |
| 1) manipulation | - function | findAllUnknowns<br>function: aTerm<br>get0KAFUnknownMatrix<br>getYourOperator<br>giveElementaryMatrix<br>giveFunction<br>giveMatrixType<br>printString<br>transpose |
| 2) analyse | | is0KAF<br>is0KAFUnknown<br>isAnUnknown<br>isIncrement<br>isKAF<br>isKAFUnknown<br>isUnknown<br>isVector |
| 3) discrétisation | | getDiscretizedForm |

**Table 1.** *Classe Increment*

### 3.3. *L'environnement graphique*

L'environnement FEM_Theory est enrichi d'un nouvel outil permettant à l'utilisateur de linéariser la formulation qu'il dérive. Celui-ci, outil *Compute Consistent Linearization*, apparaît dans la liste des outils disponibles dans la fenêtre principale de FEM_Theory (cf. [EYH 97a]).

Une nouvelle notation est introduite à ce stade afin de représenter la partie linéaire d'une fonctionnelle dans la fenêtre principale de l'environnement FEM_Theory. Ainsi, une instance de **Increment** est notée en ajoutant le symbole « ð » devant le nom de la fonction. Par exemple, l'incrément en $u$, $\delta u$, est noté « ðU » (cf. l'exemple dans le paragraphe 5).

On remarque ici que l'implantation graphique est très naturelle. La programmation orientée-objet se prête particulièrement bien à la représentation d'entités graphiques.

## 4. Procédure de programmation automatique

La procédure de linéarisation consiste en quelque sorte à remplacer un problème sous la forme classique $N(d) = f$ par un problème sous forme approchée en linéarisant $N(d) : L_d N(\delta d) = K_{tan}(d)\,\delta d + N(d) - f = 0$. Le problème est alors résolu en utilisant un algorithme de type Newton-Raphson décrit en table 2. Un concept supplémentaire est alors nécessaire pour une approche non linéaire dans le schéma de programmation automatique. Un nouveau choix conditionnel est nécessaire dans la création de code pour le calcul des contributions élémentaires pour le module tangent. Ceci est rajouté au niveau du produit, classe **ProdList** ; la méthode de génération de code de cette classe est donc généralisée.

Un nouveau concept est également ajouté à ce stade. Dans la procédure de résolution itérative, les quantités élémentaires sont évaluées au point d'intégration, puis sont sommées. Ainsi, à chaque point d'intégration, des valeurs sont calculées à partir des valeurs nodales de l'itération précédente. Dans le but d'accroître l'efficacité du code généré automatiquement, l'idée est d'introduire un nouvel objet dont la seule tâche sera de gérer le stockage et le calcul de valeurs intermédiaires. Cet aspect est illustré au paragraphe 5.3, dans le contexte d'un problème unidimensionnel de convection pure. On montre ainsi que des paradigmes de programmation complexes peuvent être gérés dans le contexte du codage automatique de formulation éléments finis.

---

Trouver $d$ grâce au schéma itératif suivant :

Soit $d^0$ une solution initiale.

A l'itération $i$ (pour $i \geq 1$), résoudre : $K_{tan}(d^{i-1})\,\delta d^i + N(d^{i-1}) - f = 0$

Mise à jour : $d^i = d^{i-1} + \delta d^i$

Etant donné un critère de convergence $\varepsilon$, vérifier $\dfrac{\left\| N(d^i) - f \right\|}{\left\| N(d^0) - f \right\|} \leq \varepsilon$

Si la condition ci-dessus est vérifiée, alors $d = d^i$ ; sinon $i = i + 1$.

---

**Table 2.** *Description d'un algorithme de type Newton-Raphson*

## 5. Application à un problème de convection non linéaire unidimensionnel

### 5.1. *Forme forte et forme faible du problème*

La forme forte du problème est rappelée en table 3. La formulation que l'on souhaite utiliser pour la résolution de ce problème est présentée en table 4. C'est une formulation spatio-temporelle discontinue en temps. On trouvera les détails théoriques sur ce type de formulation dans [HUG 88] dans le cas de l'élasticité en dynamique, et dans [TEZ 92a, 92b, HAN 92a, 92b] pour une application originale dans laquelle le domaine est mobile.

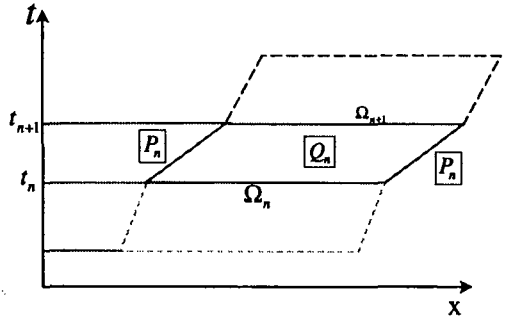
Trouver $u(x,t)$ défini sur $\Omega = [0,1]$ tel que :

$u_{,t} + uu_{,x} = 0$ sur $\Omega$

Conditions de bord : $u(0,t) = \overline{u}, u(1,t) = 0$

Conditions initiales : $u(x,0) = u_0$

**Table 3.** *Forme forte du problème de convection non linéaire 1-D*




Pour chaque intervalle de temps $[t_n, t_{n+1}]$ trouver $u^h \in (\mathcal{S}^h)_n$ tel que $\forall w^h \in (\mathcal{V}^h)_n$ :

$$\int_{Q_n} (u^h_{,t} + u^h u^h_{,x}) w^h \, dq + \sum_{e=1}^{n_{el}} \int_{Q_{ne}} (u^h_{,t} + u^h u^h_{,x}) \tau (w^h_{,t} + u^h w^h_{,x}) dq + \int_{\Omega_n} [[u^h]] (w^h)^+_n dv = 0$$

$$(\mathcal{S}^h)_n = \left\{ u^h \in [H^1(Q_n)]^h \mid u^h = \overline{u} \quad \text{sur } (P_n)_{\overline{u}} \right\}$$

$$(\mathcal{V}^h)_n = \left\{ u^h \in [H^1(Q_n)]^h \mid u^h = 0 \quad \text{sur } (P_n)_{\overline{u}} \right\}$$

où $\tau = \left( \left( \frac{2}{\Delta t} \right)^2 + \left( \frac{2|u|}{h} \right)^2 \right)^{-\frac{1}{2}}$ ($h$ longueur de l'élément)

où $[[u^h]] = (u^h)^+_n - (u^h)^-_n$ est le terme 'saut', défini de la façon suivante :

$(u^h)^\pm_n = \lim_{\varepsilon \to 0} u^h(t_n \pm \varepsilon)$ (pour les aspects théoriques voir [JOH 94, HUG 88]).

**Table 4.** *Formulation spatio-temporelle stabilisée discontinue en temps pour l'équation de convection 1-D*

### 5.2. *Dérivation de la formulation stabilisée spatio-temporelle pour l'équation de convection 1-D non linéaire dans FEM_Theory*

La formulation présentée dans le paragraphe 5.1 est dérivée dans FEM_Theory, et est présentée figure 5. Les notations sont proches de celles utilisées dans le paragraphe 5.1 (pour davantage de détails cf. [EYH 95]). Une brève description des

opérations réalisées est donnée. Sur la ligne 1, la formulation de Galerkin originale est imprimée. Elle est écrite sur le domaine espace/temps complet. Sur la ligne 2, le terme de 'saut' en « U » est ajouté ; la formulation se retrouve donc écrite sur un intervalle de temps, c'est une formulation discontinue en temps. Sur la ligne 3, la formulation est développée. La forme linéarisée de la formulation variationnelle est affichée ligne 4. Sur la ligne 5, la formulation approximée apparaît. L'élément de référence choisi pour cette application est un élément quadrangulaire, la fonction $u$ étant interpolée à chaque nœud. On peut noter que les contributions élémentaires au module tangent sont facteur du terme « ðd ». En ligne 5, la partie linéarisée des termes de stabilisation est rajoutée. Comme cela a été dit dans le paragraphe précédent, l'environnement permet de mener à bien une linéarisation consistante. Dans le but d'accroître l'efficacité numérique du code numérique, on se contente souvent d'une linéarisation approchée ou non-consistante. Ainsi, durant la phase de linéarisation certains termes sont artificiellement « fixés » ; par conséquent, la convergence quadratique d'un algorithme de Newton-Raphson classique n'est plus possible, mais le nombre de termes nécessaires au calcul numérique du module tangent peut diminuer sensiblement. L'idée est donc de « fixer » certains termes dans la formulation table 4. Cela est fait dans les termes de stabilisation ajoutés à la formulation de Galerkin originale, termes dans lesquels sont fixés le paramètre de stabilisation $\tau$ (pourtant dépendant de $u$) ainsi qu'une partie des termes de convection. Au lieu d'introduire les termes de stabilisation $\sum_{e=1}^{n_{el}} \int_{Q_{ne}} (u^h_{,t} + u^h u^h_{,x}) \cdot \tau(u) \cdot (w^h_{,t} + u^h w^h_{,x}) dq$, soit en utilisant la notation de FEM_Theory « U,t+UU,x » pondéré par « TW,t+TUW,x » et intégré sur l'élément, on introduit $\sum_{e=1}^{n_{el}} \int_{Q_{ne}} (u^h_{,t} + u^h u^h_{,x}) \cdot \tau \cdot (w^h_{,t} + A w^h_{,x}) dq$, soit « U,t+UU,x » pondéré par « TW,t+TAW,x » et intégré sur l'élément. Ainsi, une procédure de linéarisation non-consistante est obtenue. Une opération supplémentaire est alors nécessaire pour introduire l'égalité $A \equiv u^h$. L'expression résultant de l'addition de termes de stabilisation à la forme de Galerkin est affichée ligne 6.

Suivent les opérations classiques de la méthode des éléments finis (cf. divers exemples de dérivations décrits dans [EYH 97b]) :

- les fonctions tests étant choisies quelconques, il en résulte un système à une équation,
- les équations composant le système sont ensuite transposées,
- un changement de notations évident est enfin effectué, le résultat est affiché ligne 11.

Des fonctions d'interpolation bilinéaires sont choisies. Une intégration de Gauss à deux fois deux points est adoptée. Le problème ligne 11 est sous la forme $K_{tan}(d^i)\delta d^{i+1} = N(d^i)$. La nouvelle procédure de programmation automatique permet d'introduire les contributions élémentaires pour le module tangent et pour le résidu dans le code FEM_Object (cf. [DUB 93] pour le code, et [EYH 97b] pour la procédure).

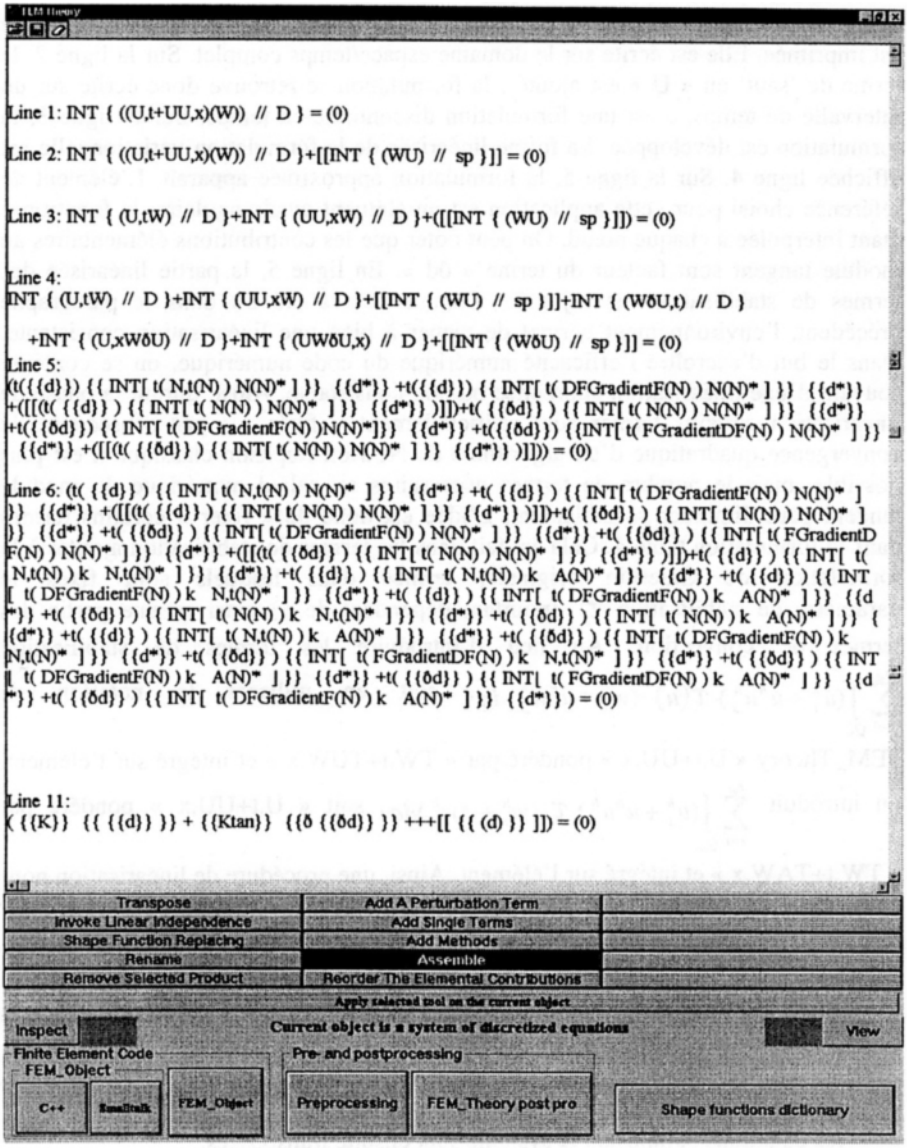



**Figure 5.** *Dérivation d'une formulation pour le problème de convection 1-D*

### 5.3. *Code numérique généré pour une formulation spatio-temporelle non linéaire discontinue en temps*

Le code correspondant à la formulation du paragraphe précédent est introduit dans le code orienté-objet FEM_Object version C++ (cf. [DUB 92b]). La hiérarchie montrant les nouvelles classes ajoutées pour la nouvelle formulation dans FEM_Object apparaît dans la figure 6. A la classe **NewElement**, on ajoute les méthodes montrées figure 7. On peut noter que ce nouvel élément est un élément dont la formulation permet d'utiliser une procédure de mouvement de maillage de type Deforming-Spatial-Domain/Space Time (DSD/ST) (cf. [TEZ 92a] pour les DSD/ST, ou [HAN 92a]) ; **NewElement**, qui est une sous-classe de **STF_ELEMENT**, possède une méthode pour calculer le terme « saut ». La méthode *computeTangentStiffnessMatrixAt: stepN* permet de calculer les contributions élémentaires au module tangent.

La classe **NewElementGaussPoint** est générée automatiquement comme sous-classe de la classe **GaussPoint**. Dans ce cas particulier, cette classe possède la capacité de gérer et de stocker le calcul du scalaire $u$ et de son gradient. Cela permet d'augmenter l'efficacité numérique du code généré.

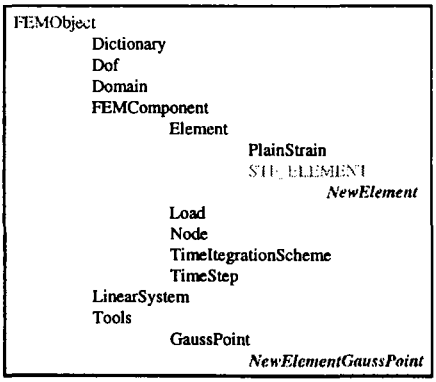



**Figure 6.** *Vue partielle des classes ajoutées pour le problème non linéaire*

### 5.4. *Résultats numériques*

Les tests numériques sont effectués en prenant comme conditions initiales la fonction montrée en figure 9. L'espace, à savoir le domaine [0,1], est décomposé en 20 éléments, et la hauteur de l'élément en temps est $\Delta t = 0{,}05$. Les conditions de bords sont $u(0) = 1$ et $u(1) = 0$. Le test est tout d'abord effectué sur un maillage fixe, c'est-à-dire que la procédure DSD/ST (cf. paragraphe 5.3 et [TEZ 92a]) n'est pas appliquée. Les résultats sont montrés en figure 10 ; il sont en accord avec la solution théorique ; on peut en effet observer le développement d'une onde de choc, puis son mouvement (cf. par exemple [ZIE 91]). Il est intéressant de noter que cette formulation permet de capturer et de suivre l'onde de choc créée sans aucun artifice supplémentaire. Seules de faibles oscillations demeurent de part et d'autre du choc.

computeGaussPts
giveGaussPtsArray

computeTangentStiffnessMatrixAt( )
computeTangentStiffnessMatrixAt11 ( )
computeTangentStiffnessMatrixAt111 ( )
computeTangentStiffnessMatrixAt111FunctionAt (
computeTangentStiffnessMatrixAt112 ( )
computeTangentStiffnessMatrixAt112FunctionAt (
computeTangentStiffnessMatrixAt12 ( )
...

computeStiffnessMatrixAt ( )
computeStiffnessMatrixAt11 ( )
computeStiffnessMatrixAt111 ( )
computeStiffnessMatrixAt111FunctionAt ( )
computeStiffnessMatrixAt112 ( )
computeStiffnessMatrixAt112FunctionAt ( )
computeStiffnessMatrixAt12 ( )
...

ComputeJumpTermAt()
...

giveJacobianMatrixAt ( )
giveJacobianMatrixAt11 ( )

*constructor and destructor*

**Figure 7.** *Vue partielle des méthodes ajoutées à la classe NewElement*

**NewElementGaussPoint**
subclass of **GaussPoint**

Attributes :
- GradientU
- ScalarU

Methods :
- giveGradientU( )
- giveScalarU( )

**Figure 8.** *Description partielle de la classe NewElementGaussPoint*

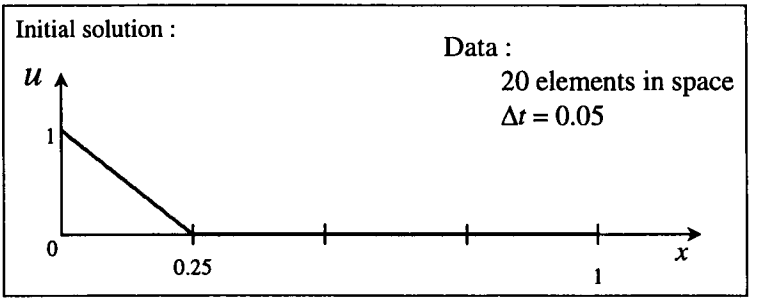



**Figure 9.** *Description du test numérique pour le problème de convection non linéaire 1-D*

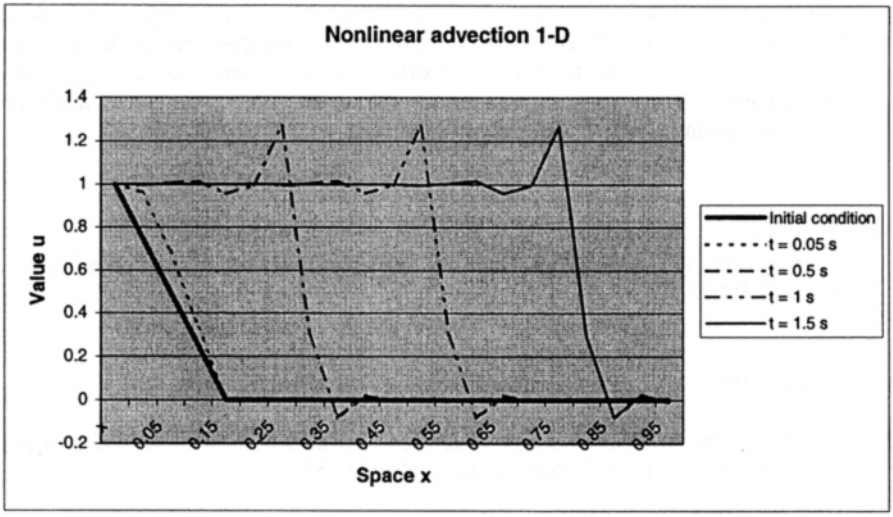



**Figure 10.** *Résultats numériques du test numérique pour le problème de convection non linéaire 1-D*

## 6. Conclusion : vers un environnement général pour la résolution de problèmes non linéaires

Dans cet article, un environnement de base pour traitement de problèmes non linéaires a été présenté. Cela entre dans le cadre d'une application à la dérivation de formulations non linéaires dans l'environnement symbolique FEM_Theory. La démarche adoptée repose sur trois idées principales :

- représentation des termes non linéaires dans les formulations,
- implantation de la procédure de linéarisation,
- généralisation de la procédure de codage automatique.

Les formulations non linéaires peuvent être représentées dans l'environnement symbolique FEM_Theory. La brève analyse du paragraphe 2 a permis de mettre en évidence les concepts et les propriétés nécessaires à l'implantation d'une technique de linéarisation basée sur la dérivée directionnelle. L'approche objet adoptée est décrite dans le paragraphe 3, accompagnée des classes et méthodes. Dans le paragraphe 4, la généralisation du schéma de programmation automatique permettant d'intégrer les nouvelles formulations dans un algorithme de type Newton-Raphson est évoquée. Pour conclure la démonstration, la dérivation d'une formulation espace/temps discontinue en temps pour une équation de convection unidimensionnelle est donnée au paragraphe 5, suivie d'une application numérique. La démarche suivie pour cet article est tout à fait générale, et peut être appliquée à tout nouveau concept que l'on souhaiterait introduire au sein de l'environnement.

On a donc démontré que l'approche proposée dans [ZIM 96, EYH 95] peut se généraliser hors du cadre des formulations linéaires. Ceci contribue à confirmer l'intérêt que peuvent avoir les approches mixtes symbolique/numérique. La

souplesse d'implantation est obtenue grâce à l'approche orientée-objet. On notera tout de même qu'un environnement de programmation tel que Smalltalk est d'un apport considérable pour un tel travail ; l'efficacité numérique est, elle, obtenue grâce au langage C++. L'application du paradigme orienté-objet peut être considérée comme essentielle au succès d'un tel développement.

Remerciements

Cette étude est financée par le Fond National Suisse pour la Recherche Scientifique, subside n$^0$ 20-45697.95.

## 7. Bibliographie

[BES 97] J. BESSON, R. FOERCH, Large scale object-oriented finite element code design, *Comput. Methods Appl. Mech. Engrg.*, 142 (1997) 165-187.

[CAR 94] A. CARDONA, I. KLAPKA, M. GÉRADIN, Design of a new finite element programming environment, *Engineering Computations*, vol. 11 (1994) pp. 365-381.

[CEC 77] M.M. CECCHI, C. LAMI, Automatic generation of stiffness matrices for finite element analysis, *Internat. J. Numer. Methods Engrg.*, vol. 11 (1977) pp. 396-400.

[CHO 92] D.K. CHOI, S. NOMURA, Application of symbolic computation to two-dimensional elasticity, *Computers & Structures*, vol. 43 (1992) pp. 645-649.

[DUB 92a] Y. DUBOIS-PÈLERIN, TH. ZIMMERMANN, P. BOMME, Object-oriented finite element programming : II. A prototype program in Smalltalk, *Comput. Methods Appl. Mech. Engrg.*, vol. 98 (1992) pp. 361-397.

[DUB 92b] Y. DUBOIS-PÈLERIN, TH. ZIMMERMANN, *Object-Oriented finite element programming : Theory and C++ Implementation for FEM_Object$_{C++}$™ 001*, Elmepress international (1992).

[DUB 93] Y. DUBOIS-PÈLERIN, TH. ZIMMERMANN, Object-oriented finite element programming : III. An efficient implementation in C++, *Comput. Methods Appl. Mech. Engrg.*, vol. 108 (1993) pp. 165-183.

[DEV 92] P.R.B. DEVLOO, C.A. MAGALHAES, A.T. NOEL, On the implementation of the p-adaptive finite element method using the object oriented programming philosofy, in *Numerical methods in engineering and applied sciences*, part 1, CIMNE, Barcelona (1992).

[DUB 95] Y. DUBOIS-PÈLERIN, P. PEGON, Object-Oriented programming in nonlinear finite element analysis, Submitted to *Computers & Structures* (1995).

[EYH 94] D. EYHERAMENDY, TH. ZIMMERMANN, Object-oriented finite element programming : Beyond fast prototyping, *Proceedings of CST 94*, Athens Greece, vol. Artificial intelligence and object oriented approaches for structural engineering, Civil Comp Press, (1994) pp. 121-127.

[EYH 95] D. EYHERAMENDY, TH. ZIMMERMANN, Programmation orientée objet appliquée à la méthode des éléments finis : dérivations symboliques, programmation automatique, La *Revue Européenne des éléments finis*, vol. 4 (1995) pp. 327-360.

[EYH 96] D. EYHERAMENDY, TH. ZIMMERMANN, Object-oriented finite elements : II. A symbolic environment for automatic programming, *Comput. Methods Appl. Mech. Engrg*, 132 (1996) pp. 259-276.

[EYH 97b] D. EYHERAMENDY, TH. ZIMMERMANN, Fonctionnalité d'un environnement orienté objet pour le développement de code éléments finis, *Actes du 3$^{e}$ Colloque national en calcul des structures de Giens*, Hermès (1997) pp. 553-558.

[EYH 97 d] D. EYHERAMENDY, TH. ZIMMERMANN, Object-oriented finite elements : III. Theory and application of automatic programming, *Comput. Methods Appl. Mech. Engrg.*, (1997) in press.

[EYH 97f] D. EYHERAMENDY, Object-Oriented Finite Element Programming : Symbolic derivation and automatic programming, rapport de thèse de doctorat, Ecole Polytechnique Fédérale de Lausanne, (1997) in preparation.

[FOE 96] R. FOERCH, Un environnement orienté objet pour la modélisation numérique des matériaux en calcul des structures, rapport de thèse de doctorat, Ecole Nationale Supérieure des Mines de Paris, (1996).

[FOR 90] B.W.R FORDE, R.O. FOSCHI, S.F STIEMER, Object-Oriented Finite Element Analysis, *Computers & Structures*, vol. 34 (1990) pp. 355-374.

[GEL 95] J. C. GELIN, L. WALTERTHUM, Conception d'un logiciel orienté-objets pour la simulation de processus de formage, *Actes du 2$^{nd}$ Colloque national en calcul des structures de Giens*, Hermès (1995) pp. 552-558.

[HAN 92a] P. HANSBO, The characteristic streamline diffusion method for convection-diffusion problems, *Comput. Methods Appl. Mech. Engrg.*, vol. 96 (1992) pp. 239-253.

[HAN 92b] P. HANSBO, The characteristic streamline diffusion method for the time-dependent incompressible Navier-Stokes equations, *Comput. Methods Appl. Mech. Engrg.*, vol. 96 (1992) pp. 239-253.

[HOA 80] S. V. HOA, S. SANKAR, A computer program for automatic generation of stiffness and mass matrices in finite-element analyis, *Computers & Structures*, vol. 11 (1980) pp. 147-161.

[HUG 88a] T.J.R. HUGHES, G.M. HULBERT, Space-time finite element methods for elastodynamics : formulations and error estimates, *Comput. Methods Appl. Mech. Engrg.*, 66 (1988) pp. 339-363.

[HUG 78] T.J.R. HUGHES, K. S. PISTER, Consistent linearization in mechanics of solids and structures, *Computers & Structures*, vol. 8 (1978) pp. 391-397.

[IBR 93] A. IBRAHIMBEGOVIC, F. FREY, Geometrically non-linear method of incompatible modes in application to finite elasticity with independent rotations, *Int. J. Num. Methods in Eng.*, 36 (1993) 4185-4200.

[JOH 94] C. JOHNSON, *Numerical solution of partial differential equations by the finite element method*, Cambridge University Press (1994).

[KOZ 94] I. KOZAR, A. IBRAHIMBEGOVIC, Finite Element formulation of finite rotation solid element, LSC internal report, Swiss Federal Institute of Technology, (1994).

[LEF 91] L. LEFF, Y.Y. YUN, The symbolic finite element analysis system, *Computers & Structures*, vol. 41 (1991) pp. 227-231.

[MAC 92] R.I. MACKIE, Object-oriented programming of the finite element method, *Int. J. Num. Meth. Engr.*, vol. 35 (1992) 425-436.

[MAR 83] J.E. MARSDEN, T.J.R. HUGHES, *Mathematical foundations of elasticity*, Prentice-Hall, (1983).

[MIL 91] G.R MILLER., An Object-Oriented Approach to Structural Analysis and Design, *Computers & Structures*, vol. 40 n° 1 (1991) pp. 75-82.

[MEN 93] PH. MENÉTREY, TH. ZIMMERMANN, Object-Oriented Non-Linear Finite Element Analysis : Application to J2 plasticity, *Computers & Structures*, vol. 49 n° 5 (1993) pp. 767-777.

[NOO 79] A.K. NOOR, C.M. ANDERSEN, Computerized symbolic manipulation in structural mechanics-progress and potential, *Computers & Structures*, vol. 10 (1979) pp. 95-118.

[NOO 90] Symbolic computations and their impact on mechanics, Winter Annual Meeting of the American Society of Mechanical Engineers, Dallas, Texas, November 25-30, 1990, ed. by A.K. Noor, I. Elishakoff and G. Hulbert, PVP, vol. 205 (1990).

[POT 97 a] POTAPOV S., Un algorithme ALE de dynamique rapide basé sur une approche mixte Eléments finis-Volumes finis. Implémentation en langage orienté objet C++, rapport de thèse de doctorat, Ecole Centrale Paris, (1997).

[REH 89] REHAK D.R., BAUGH JR. J.W., Alternative Programming Techniques for Finite Element Programming Development, *Proceedings IABSE Colloquium on Expert Systems in Civil Engineerings*, Bergamo, Italy. IABSE, (1989).

[SCH 92] S.P. SCHOLZ, Elements of an object-oriented FEM ++ program in C++, *Comp. and Struct.*, 43 (1992) pp. 517-529.

[SIL 94] P. P. SILVESTER, S. V. CHAMLIAN, Symbolic Generation of Finite Elements for Skin-Effect Integral Equations, *IEEE Transactions on magnetics*, vol 30, n° 5 (1994) pp. 3594-3597.

[TEZ 92b] T.E. TEZDUYAR, M. BEHR, J. LIOU, A new strategy for finite element computations involving moving boundaries and interfaces - The deforming-spatial-domain/space-time procedure : I. The concept and the preliminary numerical tests, *Comput. Methods Appl. Mech. Engrg.*, 94 (1992) pp. 339-351.

[TEZ 92c] T.E. TEZDUYAR, M. BEHR, S. MITTAL, J. LIOU, A new strategy for finite element computations involving moving boundaries and interfaces - The deforming-spatial-domain/space-time procedure : II. Computation of free-surface flows, two liquid flows, and flows with drifting cylinders, *Comput. Methods Appl. Mech. Engrg.*, 94 (1992) pp. 353-371.

[VIS 95a] VisualSmalltalk Enterprise - 32 Bit Pure Object-Oriented Programming System, User's guide, ParkPlace Digitalk, (1995).

[VIS 95b] VisualSmalltalk Enterprise - 32 Bit Pure Object-Oriented Programming System, Language Reference, ParkPlace Digitalk, (1995).

[WAN 86] P.S. WANG, FINGER : A symbolic System For Automatic Generation of Numerical Programs in Finite Element Analysis, *J. Symbolic Computation*, vol. 2 (1986) pp 305-316.

[YAN 94] C.Y. YANG, An algebraic-expressed finite element model for symbolic computation Computers & Structures, vol. 52 n° 5 (1994) pp. 1069-1077.

[YU 94] G.G. YU, Object-oriented models for numerical and finite element analysis, PhD thesis report, The Ohio State University, (1994).

[ZIE 91] O. C. ZIENKIEWICZ, R. L. TAYLOR, *The finite element method*, 4th ed. Vol. 2, Solid and fluid mechanics, Dynamics and Non-Linearity, McGraw-Hill (1991).

[ZIM 92a] TH. ZIMMERMANN, Y. DUBOIS-PÈLERIN, P. BOMME, Object-oriented finite element programming : I. Governing principles, *Comput. Methods Appl. Mech. Engrg.*, vol. 98 (1992) pp. 291-303.

[ZIM 92b] TH. ZIMMERMANN, Y. DUBOIS-PÈLERIN, P. BOMME, *Object-oriented finite element programming : Theory and Smalltalk V Implementation for FEM_Object$_{pc}$™001*, Elmepress international (1992).

[ZIM 96] TH. ZIMMERMANN, D. EYHERAMENDY, Object-oriented finite elements : I. Principles of symbolic derivations and automatic programming, *Comput. Methods Appl. Mech. Engrg.*, 132 (1996) pp. 277-304.