République Algérienne Démocratique et Populaire

Ministère de l'enseignement supérieur et de la recherche scientifique

Université Mentouri de Constantine
Faculté des sciences et sciences de l'ingénieur
Département de Génie civil

Thèse
Présentée Pour obtenir le diplôme de doctorat en sciences
En génie civil
Option : structure

N0 d'ordre :……..
Série……

APPROCHE ORIENTEE OBJET DE LA METHODE DES ELEMENTS FINIS

Par
BELGASMIA MOURAD

Soutenue le 03/12/2006.

Devant le jury :

Président :	**CHIKH Nasr Eddine**	**Pr. Université de Constantine**
Rapporteur :	**Guenfoud Mohamed**	**Pr. Université de Guelma**
Examinateurs:	**Benmansour Toufik**	**MC. Université de Constantine**
	Zarour Nasr Eddine	**MC. Université de Constantine**
	Guenfoud Salah	**MC. Université de Guelma**
	Belounar Lamine	**MC. Université de Biskra**

session 2006

REMERCIEMENT

Que messieurs les professeurs M.Guenfoud & T. Zimermmann, trouvent ici l'expression de toute ma gratitude pour leurs aides inestimables, leur compréhension et leurs encouragements. Je leurs suis très reconnaissant pour l'orientation et le savoir si important que j'ai trouvés en leur noble personne.

Je tiens aussi à remercier le professeur François Frey d'avoir accepté de m'accueillir dans son laboratoire de Mécanique des structures et des milieux continus (Lausanne Suisse) et de mettre a ma disposition tous les moyens nécessaire pour pouvoir peaufiner ma recherche.

Je tiens aussi à présenter mes remerciements les plus ardents à messieurs les membres de jury d'avoir accordé à mon travail une importance très encourageante.

TABLE DES MATIERES

CHAPITRE IV : THEORIE D'ELASTICITE ANISOTROPE ET DES PLAQUES EN FLEXION

CHAPITRE V : ELEMENTS FINIS DE PLAQUE EN STATIQUE ET EN DYNAMIQUE LINEAIRE

CHAPITRE VII : LA PROGRAMMATION ORIENTEE OBJET EN ELEMENTS FINIS DANS LE DOMAINE NON LINEAIRE

CHAPITRE VIII : VALIDATION ET TESTS NUMERIQUES

INTRODUCTION GENERALE

Les logiciels actuels de modélisation numérique, et en particulier les codes d'analyse des structures par la méthode des éléments finis, reposent encore essentiellement sur des schémas et des méthodes des premiers temps de l'informatique. Il a été ainsi accepté jusqu'à récemment que les codes éléments finis ne puissent réaliser que des analyses sur des données statiques et bien définies au départ par l'utilisateur. Les résultats du calcul devaient également suivre un format déterminé. Ainsi a-t-on continué de développer les codes de modélisation avec le langage Fortran et le style procédural, bien adaptés à ce monde « de serveur numérique ». A l'opposé, en conception assisté par ordinateur, les données sont changeantes en valeur, mais aussi en forme. Elles sont modifiées par le logiciel ou, de manière interactive, par l'utilisateur. Le développement de tels logiciels de CAO se fait en plus en utilisant des langages modernes et des méthodologies logicielles fondées sur l'approche objet. Jusqu'à présent la cohabitation entre le monde de la conception et celui de l'analyse a été réalisée par des couplages entre des logiciels développés indépendamment, communiquant d'une manière limitée ou indirecte à travers des données partagées. Ce type d'intégration faible offre une solution à court ou moyen terme, donnant certaines satisfactions, mais figeant chaque partie dans son rôle primitif. Cela conforte ce qui a toujours existé au détriment de ce qui devrait ou pourrait exister.

Pourtant l'analyse numérique est entrain de vivre une triple révolution. En premier lieu, son champ d'action est de plus en plus vaste et complexe. Fortran, outil (logiciel) traditionnel des numériciens conçu il y a plus de 30 ans, a du mal à suivre cette évolution. La seconde mutation qui frappe l'analyse numérique s'appelle « adaptativité automatique » ; nous sommes loin des données statiques entrées par l'utilisateur, la programmation structurée ne nous semble pas un outil adéquat à cette nouvelle forme d'analyse numérique.

Enfin, l'accroissement considérable de la mémoire centrale des ordinateurs permet aujourd'hui d'envisager des structures de données plus riches, conçues pour le confort et la productivité du programmeur, et non plus dans le strict souci d'économiser des octets.

Depuis environ douze ans, un nombre de plus en plus de chercheurs et de développeurs des programmes d'analyse numérique s'intéressent ou font le pas vers la programmation orientée objet. Les langages utilisés sont variables, Object-pascal, CLOS, SmallTalk ou C++, avec une actuelle prédominance de ce dernier.

Selon nous, l'approche objet est une opportunité pour repenser et redéfinir l'ensemble des entités intervenant dans la méthode des éléments finis. Il serait dommage, et presque néfaste, de garder les représentations qui étaient adaptées à une autres forme de programmation. Pour aller plus loin dans les logiciels d'analyse numérique, il ne nous apparaît pas insensé de vouloir, à travers la programmation objet, pratiquement repartir à zéro.

Le but de ce travail, est l'implémentation d'un élément plaque isotrope et orthotrope dans un code élément finis orienté objet en utilisant le langage C++. Pour ce faire cette thèse se présente comme suit :

- Le premier chapitre est consacré à un rappel sur les structures de données traditionnelles et sur la programmation procédurale des codes éléments finis en Fortran. Nous soulignons leurs limitations face aux nouveaux défis de la modélisation numérique.
- Le chapitre II expose les principes de la programmation orientée objet.
- Le chapitre III est consacré à l'efficacité du C++ en programmation orientée objet en éléments finis.
- La théorie d'élasticité anisotrope et des plaques en flexions est décrite dans le chapitre IV.
- Le chapitre V est consacré à la formulation de l'élément quadrilatères isoparamétriques dans le cas statique et dynamique.
- Le chapitre VII expose la théorie de plasticité ainsi qu'à l'organisation d'une partie du code en C++.
- Enfin le chapitre VIII est consacré aux différents tests numériques ainsi que la validation des résultats.
- en trouve en annexe les classes les plus importantes du code orienté objet avec le langage C++.

CHAPITRE 1

LES LOGICIELS ELEMENTS FINIS TRADITIONNELS EN LANGAGE FORTRAN

1.1 Introduction

L'introduction du langage Fortran par John Backus en 1954 a coïncidé avec l'émergence des méthodes numériques utilisant les ressources des premiers ordinateurs. Depuis lors, Fortran n'a cessé de jouer un rôle central dans le domaine de plus en plus vaste de la simulation numérique. Ce langage de programmation permettant une compilation très efficace a été jusqu'à présent considéré comme étant la meilleure arme pour lutter contre l'ennemi numéro un de l'ingénieur numéricien : le temps de calcul. Les codes scientifiques devenaient bien toujours un peu plus complexes et donc toujours plus difficiles à maîtriser. Aucun autre langage ne pouvait offrir d'attraits rivalisant avec la rapidité du Fortran. Les bibliothèques mathématiques comme NAG, LINPACK ou EISPACK, naturellement écrites en Fortran, on imposé un peu plus le langage. Il n'est donc pas surprenant que la quasi-totalité des codes scientifiques, et en particulier des codes pour la méthode des éléments finis soit écrits avec ce langage.

Les choses sont pourtant en train d'évoluer rapidement et les structures de données classiques et le langage Fortran sont peut–être à remettre en cause.

Dans la première partie de ce chapitre, à travers le programme « FEAP » (Finite Element Analysis Program) de R.L.Taylor, nous ferons un rappel sur les structures de données traditionnelles ainsi que sur les architectures des codes éléments finis classiques, en analysant leurs forces et leurs limitations. En seconde partie, nous ferons un tour d'horizon des différents types de logiciel de modélisation et des défis qu'ils devront relever dans le futur et nous nous interrogeons sur leur capacité à les relever. Enfin, la troisième partie sera consacrée à nos propositions pour palier ces limitations.

1.2 STRUCTURE DE DONNEES ET ARCHITECTURE DU PROGRAMME « FEAP»

1.2.1 Les structures de base du Fortran

La simplicité du langage est due à la pauvreté de ses types de structures. Il n'en offre en fait qu'un seul, statique, le type tableau. L'avantage de ce type de structure est l'adressage direct donc simple et rapide, mais le problème inhérent à sa nature statique est qu'il impose une taille qui doit être définie à la compilation. Toutes les déclarations de tableaux ne peuvent se faire qu'avec des dimensions statiques.

1.2.2 Les structures de données de FEAP

Le programme FEAP (Finite Element Analysis Program) écrit par R.Ltaylor et détaillé dans le livre de Zienkiwicz & Taylor nous servira de référence dans cette partie. Bien sûr tous les autres codes éléments finis en Fortran écrits à travers le monde n'ont pas exactement les mêmes structures de données ni la même organisation. Ils en sont cependant très proches (par exemple RE_FLEX de [2] ou DLEARN de [22] et l'étude de FEAP suffit pour comprendre un nombre de points forts et de faiblesses de l'ensemble de ces codes traditionnels. A la base de FEAP nous trouverons le classique *common blanc* dimensionné par une constante

```
INTEGER * 2  M
COMMON M (32000)
COMMON /PSIZE/MAX
MAX = 32000
```

Le common blanc est partitionné pour recevoir les structures de données statiques et dynamiques : tableaux d'entiers, de caractères ou de réels. Un jeu d'entier permet de retrouver les positions des débuts de chacun de ces tableaux. On qualifie ces entiers de « pointeurs ». Dans FEAP, ils se trouvent dans deux commons et sont initialisés par la routine de contrôle PCONTR et la routine de gestion du common PSETM (fig.1.1). Les tableaux n'ont donc pas de caractère propre et explicite ; ils n'ont pas vraiment de réalité informatique et seuls les pointeurs les définissent de manière implicite. Le common blanc est en fait le seul véritable « objet » informatique. Tous les autres sont à gérer par le programmeur, ce qui donne certes une grande liberté, mais exige en contrepartie une gestion très rigoureuse.

```
COMMON / MDATA/NN,/ N0, N1, N2, N3, N4, N5, N6, N7, N8, N9,N10,N11,N12, N13
COMMON / MDTA2/ N11A, N11B, N11C, IA (2,11)
```

PCONTR doit lire sur un fichier d'entrée les divers entiers nécessaires : nombre de nœuds (NUMNP), d'éléments (NUMEL), de matériaux (NUMAT), le nombre maximum de degrés de liberté par nœud (NDF), la dimension d'espace du problème (NDM) et le nombre maximum de nœuds par éléments (NEN).

Décrivons quelques un de ces tableaux

D (18, NUMAT) données des matériaux (limité à 18 mots par matériau)

F (NDF, NUMNP) forces nodales et déplacements

ID (NDF, NUMNP) conditions aux limites puis numéro d'équations

IX (NEN1, NUMEL) connectivité des nœuds local -->globale

X (NDM, NUMNP) coordonnées des nœuds

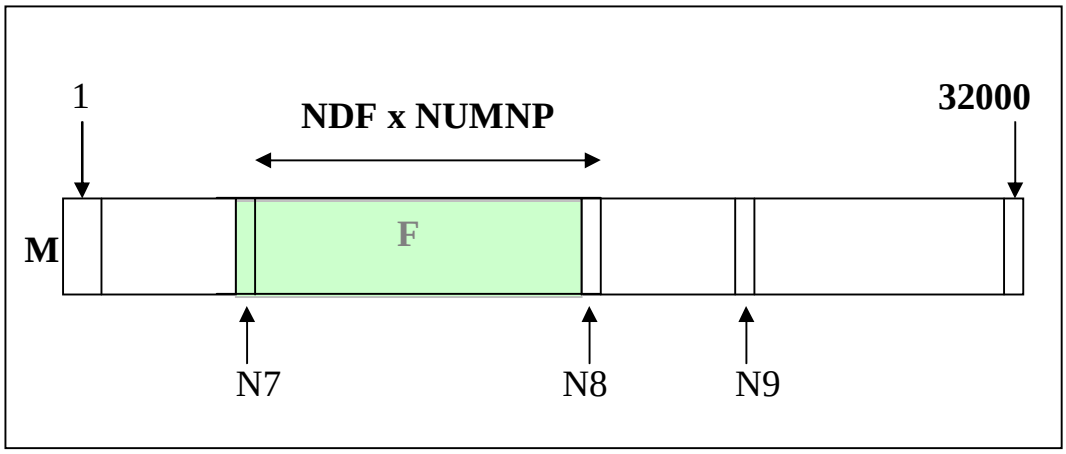


Figure 1.1 Le Common blanc, les pointeurs et le vecteur F

Common fixe

La taille du common est fixée à la compilation. C'est une limitation non naturelle imposée à l'utilisateur. Si MAXM est trop petit, son calcul ne passera pas ; il devra faire une modification dans le source et recompiler. Si en revanche MAXM est trop grand, les autres utilisateurs peuvent, sur certaines machines, être pénalisés par cette grosse réservation de mémoire.

Mémoire non dynamique

La taille des tableaux n'est pas toujours calculable a priori. Il existe de nombreux exemples dans l'implémentation de la méthode des éléments finis où, au cours de l'exécution, on a besoin de mémoire temporaire de taille inconnue : mailleurs, algorithmes de numérotation, algorithmes de recherche, post-processeur, etc. Ainsi, lorsqu'une routine a besoin de manière interne de mémoire dynamique, deux solutions sont généralement employées. La première, simple consiste à lui passer un vecteur de travail (de taille fixe pris dans le common blanc) en espérant qu'elle n'en aura pas besoin de plus. L'utilisateur (consommateur de mémoire) doit avoir un ordre d'idées a priori sur l'espace à réserver ce qui n'est ni naturel, ni fiable. Avec la seconde solution, on a recours à un ensemble de routines pour gérer l'espace mémoire du common blanc. Dans ce cas, il faut pouvoir allouer et désallouer à la demande des tableaux de taille quelconque. FEAP ne propose que l'allocation par sa routine PSETM. Dans DLEARN la routine s'appelle MPOINT et réalise quasiment le même service avec en plus le fait de pouvoir nommer les tableaux dans le common blanc par une chaîne de caractères.

Même pour les structures globales permanentes (tableaux ID, IX, F…) il devient vite très utile, lorsque le code se complexifie, de disposer d'un outil (gestionnaire) pour gérer les tableaux et les pointeurs à la place du programmeur. Celui ci n'a alors plus qu'à demander au gestionnaire la création, puis la position et enfin la libération d'un tableau en fournissant simplement une chaîne de caractère en guise d'identificateur. L'écriture d'un bon gestionnaire de mémoire dynamique est délicate en Fortran. On obtient de toutes façons, à cause de la taille limitée MAXM, qu'une allocation pseudo-dynamique. De plus, ces modules ne sont toujours pas indépendants car ils doivent connaître les spécifications (intitulés et paramètres) des routines du gestionnaire et exigent en plus le common blanc en paramètre, ce qui n'est vraiment pas naturel pour le client.

Les langages modernes (Pascal, C, C++, Eiffel, Ada…) offrent tous, des possibilités d'allocation et de désallocation de mémoire. Le programmeur a ainsi à sa disposition généralement deux mots réservés du langage : malloc ou new, et free ou unchecked_deallocation.

Stockage monolithique

Les tableaux ID et IX sont dimensionnés par le nombre maximum de degrés de liberté par nœud et le nombre maximum de nœud par élément. Dans le cas où l'on utilise dans le même maillage des éléments à nombre de nœuds ou de degrés de liberté (ddl) variable, l'utilisation de la mémoire est peu optimale.

Ce stockage monolithique est également particulièrement gênant lorsque des éléments ou des degrés de liberté sont ajoutés ou supprimés au cours du calcul comme lors d'un processus de raffinement-déraffinement du maillage. Les structures de données doivent dans ce cas être beaucoup plus sophistiquées, l'allocation dynamique et les pointeurs des langages modernes sont alors particulièrement utiles.

Loin de la formulation

L'accès aux éléments des tableaux se fait de manière très abstraite : un ou deux entiers servent à indicer chaque tableau. Pourtant les données ont très souvent une signification physique précise : coordonnée spatiale, type de ddl… dans les tableaux indicés par des entiers le sens physique n'apparaît plus, ce qui nuit à la compréhension et à la fiabilité du code. En Fortran, nous ne connaissons malheureusement pas de moyen d'éviter cela. De plus il y a un besoin évident pour une programmation à plus haut niveau d'abstraction, plus proche de la formulation mathématique

Peu fiable

Le common qui fait ici 32000 entiers de 2 octets soit 64 ko sert donc à stocker tous types de tableaux, des entiers simples, doubles, des réels simples ou doubles, des caractères, etc. c'est une grande souplesse qu'offre le compilateur mais c'est dangereux. Un entier peut très bien être reçu par erreur dans une variable réelle (ou vice versa) et le compilateur ne le vérifie pas.

```
SUBROUTINE PROFIL (JD, IDL, ID, IX, NDF, NEN1)
INTEGER *2  JD (1), IDL (1), ID (NDF, 1), IX (NEN1, 1)
CALL PROFIL ( M(N1,2),  M(N1,3),  M(N7),  M(N9),  NDF,  NEN1)
```

Dans cet exemple, la routine PROFIL attend 4 tableaux d'entiers, 2 monodimensinnels, 2 bidimensionnels, et 2 entiers. La taille de ces tableaux n'est même pas en paramètre (sauf pour le premier indice des tableaux 2D). C'est au programmeur de faire très attention à passer les adresses correctes de début de ses « structures », et à ce que les tailles correspondent. La routine PROFIL est peu autonome ; son bon fonctionnement dépend fortement de l'extérieur. On pourrait programmer de manière un peut plus sûre en écrivant

```
SUBROUTINE PROFIL ( JD, IDL , ID, IX, NDF, NUMNP, NEN1, NUMEL)
INTEGER* 2  JD(NDF*NUMNP) , ID(NDF,NUMNP) , IX (NEN1 ,NUMEL)
```

Mais cela augmente encore le nombre de paramètres et donc le risque de passage de données aberrantes. En Fortran il n'y a aucune vérification du type des paramètres et tout le monde a déjà fait de nombreuses fois l'expérience de bogue dû à une adresse de début erronée, à des réels reçus comme entiers ou à un dépassement de tableau.

Dans les langages fortement typés (presque tous les langages modernes), le compilateur effectue les vérifications de nombre et de type pour chaque appel à un sous-programme éliminant ainsi les risques du Fortran à cet égard.

Rapidité

Les structures de données en Fortran (i.e tableaux) sont très simple et sont donc réputées très rapides ; Ce n'est pourtant pas tout à fait vrai. En effet, dans un tableau bidimensionnel, par exemple X(NDM,NUMNP), l'accès à un élément de position (i, j)

Se fait en calculant la position X(1,1)+X(j-1)*NDM + i-1, soit une multiplication et quatre additions d'entiers. Dans les parties du code où il n'y pas d'optimisation (c'est fréquent), l'utilisation d'un tableau de dimension NUMNP de pointeurs sur chacun des petits vecteurs

de dimension NDM est plus efficace comme le montrent des exemples de gestionnaire de mémoire, et de solveur écrit en langage C.

1.3 LES LOGICIELS DE MODELISATION

1.3.1 Evolution des techniques numériques et des modèles physiques

Depuis le début du calcul scientifique, les techniques numériques ont considérablement évoluées :

- Des premières équations aux dérivées partielles résolues par différences finies, on est passé aux éléments finis et aux équations intégrales.
- Les méthodes de résolutions ont été bouleversées : méthodes directes, méthodes itératives, méthodes muti-grilles, méthodes de sous domaines….

Les modèles physiques se sont diversifiés et complexifiés :

- Passage de l'élasticité linéaire aux lois plastiques, viscoplastiques puis à l'endommagement (local ou non local).
- Passage de la formulation en petites déformations, petits déplacements aux grandes déformations, grands déplacements.
- Traitement des instabilités et bifurcation.
- Contact et frottement.
- Modèles dynamiques
- De plus en plus le couplage entre les différents domaines de la modélisation : mécanique des solides, thermique, mécanique des fluides, électromagnétisme.

Ces modèles physiques ont également introduit de nouvelles techniques numériques :

- Méthodes de résolution non-lineaires (Newton,…).
- Méthodes de traitement des instabilités (suivi d'arc).
- Méthode d'intégration en temps (Newmark, implicite–explicite).

Enfin, le besoin d'interfaces utilisateur conviviales permettant d'interroger le logiciel, aussi bien pendant le calcul que lors du maillage ou de post-traitement, exige des trois programmes de base de la méthode des éléments finis une plus grande cohésion et des structures de donnée communes.

1.3.2 Variété des logiciels

On peut, en suivant l'analyse de Breitkopt & Touzout, décomposer l'ensemble des logiciels de modélisation en 4 grandes familles

Les logiciels de recherche généraux, servant de base de travail pour tester des modèles ou méthodes novateurs, doivent être souples et peuvent subir d'assez profondes modifications. Ils devraient être conçus comme des constructions de « modules » indépendants et facilement modifiables.

Les logiciels de recherche spécialisés modélisent un type de phénomènes particuliers et utilisent des méthodes très spécifiques. Tout comme les cousins généraux, les logiciels de recherche spécialisés ont une interface utilisateur sommaire et les données que manipule le chercheur sont de très bas niveau (matrice et vecteur dans les commons) ; les commandes sont des instructions Fortran

Les logiciels industriels généraux englobent un maximum de modèles et d'algorithmes. Ils sont dotés d'une confortable interface utilisateur et peuvent communiquer avec de nombreux mailleurs et post-processeurs. L'utilisateur manipule ici des données de haut niveau (structures physiques, données géométriques) et les commandes sont lancées à la sourie. à cause de leurs taille ces logiciels sont peu évolutifs.

Les logiciels industriels spécialisés, ou logicie*ls « métier »,* sont souvent des ex-logiciels de recherche spécialisés qui ont été fiabilisés, nettoyés et documentés. Les laboratoires s'efforcent de transformer le plus rapidement possible leurs nouveautés en produit fiables afin de gagner en renommée. De même, les industriels cherchent naturellement à transférer rapidement les résultats de la recherche vers leur code. On devrait donc améliorer la continuité et la compatibilité entre logiciels de recherches et logiciels industriels, c'est–à-dire chercher à professionnaliser les logiciels de recherche trop souvent « artisanaux ».

1.3.3 Evolution des logiciels

Complexification

Comme nous l'avons souligné au paragraphe 1-3-1, les logiciels de modélisation ont déjà subi de profondes modifications pour intégrer les évolutions des modèles et des méthodes numériques et dans une moindre mesure aux évolutions du langage Fortran. Lorsque nous parlons d'évolution des logiciels, ce n'est pas dans le sens d'une évolution globale, où un logiciel mourant laisse la place à un nouveau entièrement réécrit, mais bien dans un sens d'évolution locale, chaque logiciel étant à chaque fois adapté par souci de réutilisabilité. Il s'ensuit des logiciels complexes peu lisibles qui ont multipliés les tests et les branchements, même au niveau des structures de données de base. Cette tendance ne peut se confirmer ; la réutilisabilité devient de plus en plus difficile à obtenir.

1.4 ORIENTATION SOUHAITABLE POUR LA NOUVELLE GENERATION DE LOGICIELS

On peut tenter de formuler quelques orientations pouvant permettre de résoudre les difficultés évoquées dans les deux paragraphes précédents.

1.4.1 L'architecture

Pour augmenter la vitesse de transfert entre la recherche et l'industrie, il est indispensable de réduire la différence entre logiciels de recherche et logiciels industriel. Un logiciel de recherche doit être dés le début très bien écrit, bien structuré et d'approche facile.

Il faudrait également limiter la multiplication des logiciels. Pour cela il est nécessaire de développer une base suffisamment souple pour que différents modules spécialisés viennent s'y adapter.

Il faudrait aussi coupler le logiciel avec un système expert, seul moyen aujourd'hui pour introduire de manière simple et efficace les règles nécessaires à la prise de décision. Il peut également acquérir ces connaissances de manière automatique avec l'expérience de calcul déjà effectué.

Il serait intéressant de classifier les modèles éléments finis dans le logiciel, c'est-à-dire de définir des relations de parenté entre modèles (comme pour la classification des espaces animales), on gagnera beaucoup en clarté dans l'organisation du code. On pourrait aussi écrire des procédures valables pour toute une classe d'élément donc gagner en réutilisabilité. Cela pourrait également aider le système expert à choisir le modèle le mieux adapté au problème posé.

La programmation orientée objet nous paraît apporter de bonnes solutions pour organiser le code de manière évolutive et claire. Elle peut, à notre avis, également faciliter l'introduction d'un pilotage par un système expert dans le code.

1.4.2 Le langage de modélisation

Il faut utiliser un langage de modélisation de plus haut niveau, s'abstraire des détails informatiques, et l'idéal ce serait de coder les formulations directement en langages mathématiques. Il faut définir des objets mathématiques (vecteurs, matrices, tenseurs…) ainsi que les opérateurs que l'on peut leur appliquer. On note que le langage c++ peut être très utile dans ce domaine grâce à la surcharge des opérateurs.

1.4.3 Le langage de codage

Il doit déjà aider à implémenter les souhaits formulés pour l'architecture et pour le langage de modélisation. Il faut donc un langage plus puissant avec un gestionnaire de mémoire intégré, pointeur à indice libre, un ensemble de types de structures de données plus riche. Il doit être très modulaire et très lisible. Il doit fiabiliser le code : Réaliser le maximum de vérifications à la compilation et à l'exécution et enfin il doit être portable et rapide.

Conclusion

La discussion de ce chapitre met en évidence certaines rigidités et insuffisances des logiciels de modélisation. Nous avons mis en avant quelques orientations qui nous paraissent indispensables à l'écriture des logiciels modernes, et la programmation orientée objet nous semble être des voies prometteuses. Dans le prochain chapitre, nous ferons un rappel sur la programmation orientée objet.

CHAPITRE 2
PROGRAMMATION ORIENTEE OBJET

2.1 Introduction

La programmation orientée objet commence depuis quelques années a être utilisée dans les codes éléments finis. Certes, de nombreux auteurs ont utilisé depuis longtemps certain concepts de la POO (modules données + sous- programme, envois des messages) comme C.Felippa, P.Breitkopf et G.Touzout, ou encore W.W.Tworzydlo et J.T.Oden, mais en utilisant toujours le langage Fortran ou parfois pour quelques sous-programmes, le langage C. Jusqu'en 1989, à notre connaissance, aucun papier ne traitait de l'emploi d'un langage orienté objet pour le calcul scientifique. B.Forde et al semblent être les premiers à s'intéresser au sujet en décrivant une version objet (Object-NAP) écrit en pascal-Objet d'un code éléments finis classique, NAP (Numerical Analysis Program). Depuis, plusieurs articles sur le sujet sont parus : Miller en CLOS, Mackie en pascal-Objet, puis Sholtz en c++ T.Zimmermann et al ont décrit un code prototype en SmalTalk [8],[9].

Notant que l'approche restait classique et le langage n'était employé que pour sa fiabilité, sa lisibilité. Reconnaissons une préférence de plus en plus marquée des auteurs pour le C++, au détriment des langages interprétés comme le CLOS ou SmallTlak.

Nous consacrons ce chapitre à des rappels sur la POO. On définira le concept de classe et d'objet, l'héritage, de polymorphisme et de liaison dynamique.

2. 2 RAPPELS SUR LA PROGRAMMATION ORIENTEE OBJET [4] [12] [25]

2.2.1 Classes et objet – Encapsulation

Comme le note J.Prosen, tout langage informatique est caractérisé par des « briques de bases » pouvant se combiner pour donner des éléments de plus haut niveau. Dans les langages procéduraux (Fortran, C, Pascal..) Ce sont des sous-programmes, les données étant écartées de cette décomposition. Ces langages offrent la possibilité de définir des structures de données et les routines de manipulation de ces structures dans des parties différentes du code. Il y a découplage entre la définition de la structure de données et les structures de programmes associées.

Les langages orientés objet utilisent comme éléments fondamentaux des entités autonomes (objets) qui représentent des abstractions du monde réel. Un programme ne se caractérise plus par une suite de sous-programmes à exécuter mais par une décomposition en classes, modèles d'objet réel dont les instances communiquent entre elles. L'objet est à la fois

une structure de données et un ensemble de sous programmes. Il regroupe des attributs (données) et des méthodes (procédures et fonctions). Ses attributs sont cachés au « monde extérieur » pour dissocier vue interne et vue externe et garantir ainsi son intégrité. On parle d'encapsulation des données. Cette caractéristique peut être plus ou moins assouplie suivant les langages.

Dans la vaste galaxie des langages à objet Masini et al distinguent trois grandes familles. Ceux qui privilégient le point de vue structurel appelés **langage de classe :** « *L'objet est un type de données, qui définit un modèle pour la structure de ses représentants physiques et un ensemble d'opérations applicables à cette structure* ». On peut citer des langages comme Simula , SmallTalk , Ada], CLOS, C++ , qui sont généralement à typage statique. Les langages privilégiant le point de vue conceptuel sont appelés **langage de représentation de connaissances** (ou langages de frame) : « *L'objet est une unité de connaissance, représentant le protocole d'un concept »* ce sont des langages proches de l'intelligence artificielle comme KRL. Les **langages d'acteurs** privilégient le point de vue acteur : « *l'objet est une entité autonome et active, qui se produit par copie ».* Act1. Enfin viennent les **langages hybrides** aux croisées des chemins précédents avec LOOPS, YAFOOL etc.

A part certains langages de classe, très peu ont atteint un état industriel et un nombre suffisant d'utilisateurs pour garantir leur pérennité, souvent à cause d'une trop grande spécialisation ou d'une trop grande lenteur d'exécution (langages interprétés). Nous nous restreignons donc, dans cet aperçu des langages à objets, aux langages de classes « généraux », à typage statique.

En langage de classe, on appelle **classe** et **objet** ce qu'on appelle **type** et **variable** dans l'approche classique.

La programmation orientée objet s'appuie sur quatre concepts : l'Objet, la Classe, l'Héritage et le Polymorphisme.

Objet :

Un objet est caractérisé par une identité, un état et un comportement. L'identité est une propriété qui distingue un objet d'un autre. L'état d'un objet est sa mémoire, et son exécution par des variables qui représentent ses données. Le comportement montre l'action de l'objet sous son propre contrôle ; comment ce dernier se comporte aux excitations extérieures. (Voir figure 2.1)

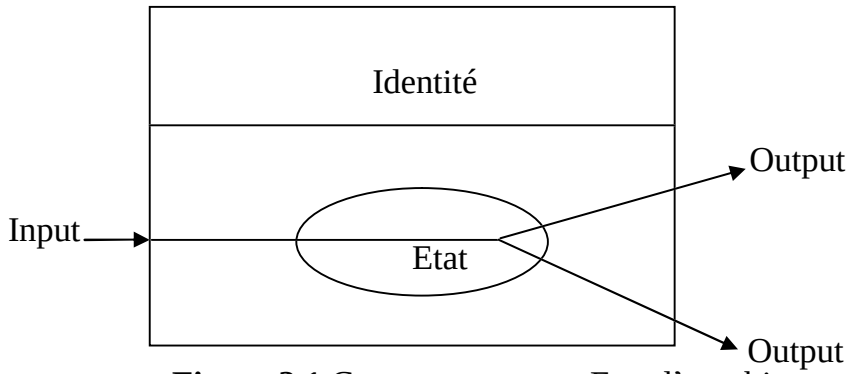


Figure 2.1 Comportement et Etat d'un objet

Le comportement de l'objet est exécuté par un ensemble d'opérations travaillant avec ses propres variables. Une opération souvent appelée méthode est une action que l'objet exécute sous son propre contrôle. Cependant, les méthodes et les données sont rassemblées dans la même enceinte (objet), et ils représentent l'interface qui est fourni à l'extérieur (voir figure 2.2).

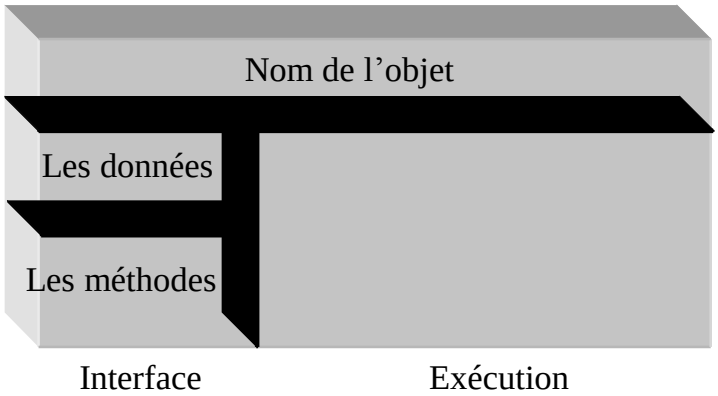


Figure 2.2 Encapsulation des données

L'objet communique avec un autre objet par le biais des messages ces derniers spécifient :

- La destination de l'objet auquel la demande est adressée.
- La sélection de la méthode à exécuter.

L'objet est utilisé pour diverses raisons :

- Facilité de compréhension et de réutilisation du code.
- Amélioration de la fiabilité.
- Facilité de compréhension.
- Abstraction accrue.
- Encapsulation accrue.

- Masquage amélioré des informations.

Classe :

Une classe regroupe les objets partageants les mêmes propriétés ; qui sont : les données et les méthodes. Elle est aussi une collection d'objets indifférents de leurs états respectifs, ainsi l'objet est identifié comme étant une variable abstraite.

La classe spécifie un certain nombre d'attributs et les méthodes pouvant opérer sur ces attributs Figure.2.3. A partir de la classe, on instancie des objets que l'on active par voie des messages. Le message active l'action, mais le détail de l'opération est laissé au contrôle de l'objet. On parle alors d'encapsulation des données et du comportement de l'état.

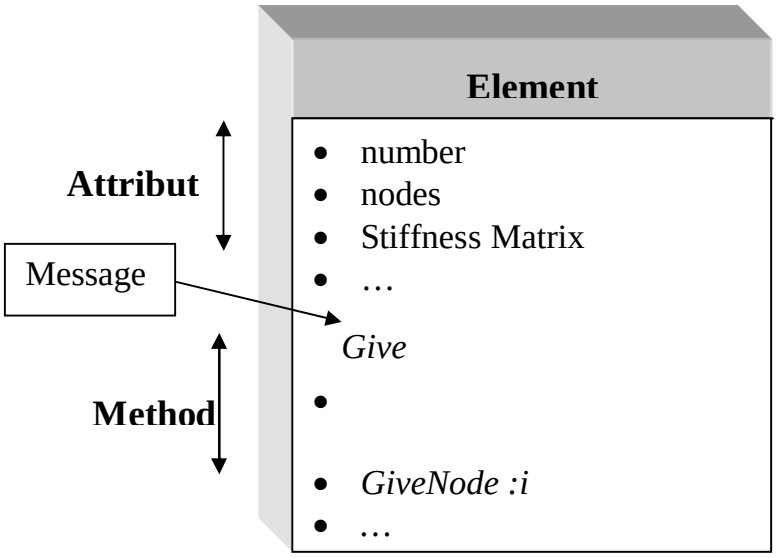


Figure 2.3 Concept de classe

Définir les objets par la seule encapsulation revient à définir ce qu'on appelle généralement des **types de données abstraits** (TDA).

On introduit en plus une organisation entre les objets de deux manières assez complémentaires : la composition et la classification

La **composition** exprime le fait de définir un objet en le constituant d'autres objets plus élémentaires. Les attributs d'un objet peuvent donc être eux-mêmes des objets et il y a communication entre composés et composants. Par exemple une voiture est composée d'objets « roues », qui ont leur comportement propre mais qui obéissent à la voiture. La conception orientée objet (COO) de G.Booch utilise uniquement la composition.

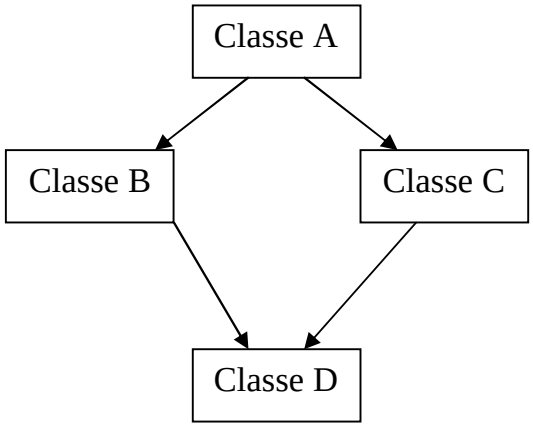
La **classification** exprime le fait que les classes puissent se regrouper symboliquement en classes plus vastes et plus générales : La classe des voitures peut être une sous-classe de la

classe véhicules… un objet d'une sous-classe partage les propriétés de sa classe ancêtre. On parle d'héritage.

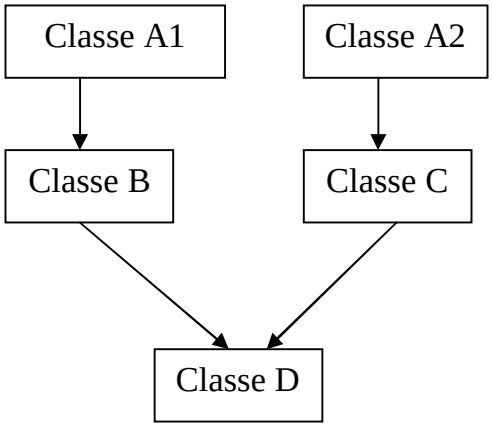
2.2.2Héritage

L'héritage est une propriété essentielle des langages orientés objet (LOO). Elle permet de définir une nouvelle classe en partant d'une classe déjà existante en ne rajoutant que la différence. L'héritage est dual : lorsque l'on indique qu'une classe F hérite d'une classe A, on obtient d'une part un héritage des attributs, ainsi la classe F possède déjà au départ toute la structure de A, mais aussi un héritage des méthodes de A. si l'on ne fait rien de plus, les deux classes sont alors équivalentes (à ceci près que F est une fille de A et non l'inverse). On a heureusement la possibilité d'enrichir la classe F et de modifier son héritage.

Donc un type de donnée abstrait peut être spécifié par la création d'un sous-type ou bien d'une sous-classe qui hérite les structures de données et les méthodes à partir de leur ancêtre appelé souvent super classe. Le mécanisme d'héritage peut être total comme dans le langage Smalltalk, cela veut dire que tous les attributs et les méthodes sont héritées par leurs sous-classes ceci à partir de leur super classe, mais quelques langages comme C++ et Java, permettent le mécanisme d'héritage partiel en sélectionnant les variables et les méthodes héritables. Tout en sachant que l'héritage peut être simple ou multiple figure (2.4).



Cas d'héritage avec des conflits



Cas ou la classe A n'est pas déclarée virtuell

Figure 2.4 régler les conflits dus à l'héritage par des déclarations virtuelles

Enrichissement

La structure de données de F peut être enrichie en ajoutant de nouveaux attributs à la structure héritée. Si *matrix-sym* possède un seul attribut *dimension*, alors *matrix_sym_skyline* classe dérivée le possédera de manière automatique et l'on pourra rajouter par exemple les attributs *ENV* et *PENV* pour décrire le profil selon le stockage « en ligne de ciel ». De même, on peut enrichir les méthodes héritées.

Modification

Alors que l'on peut qu'enrichir l'ensemble des attributs (dans la plus part des LOO), il est possible de modifier l'héritage des méthodes en les surchargeant. Une méthode doit être systématiquement surchargée : le *constructeur* qui alloue l'espace pour un objet, l'initialise et met en place les mécanismes pour la liaison dynamique (que l'on va bientôt définir). On peut également facilement concevoir que des méthodes très spécifiques, comme l'ajout d'un réel dans la matrice(Add), soient redéfinies pour chaque classe (puisque les représentations internes sont différentes entre les classes *matrix_sym_skyline* et *matrix-sym-sparse* par exemple). Dans tous les cas il ne peut y avoir appauvrissement tant en attributs qu'en méthodes. L'héritage s'appuie donc essentiellement sur un *arbre d'héritage* à la racine duquel on trouve la classe la plus primitive ; plus on descend dans l'arbre, plus on précise et on enrichit les classes. C'est un moyen très puissant pour économiser les lignes de source et pour réutiliser l'existant, donc pour réduire le nombre d'erreurs. Cependant l'arbre d'héritage

(figure2.5) a un autre intérêt : il permet la classification des classes entre elles et ce qu'on l'on appelle le *polymorphisme*

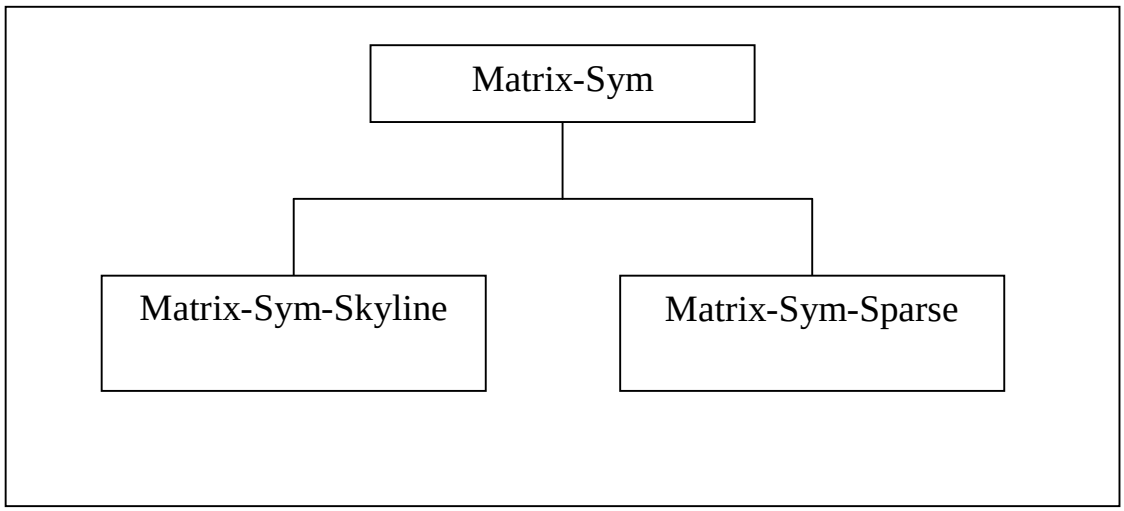


Figure 2.5 Arbre d'héritage

2.2.3 Polymorphisme :

Le terme polymorphisme signifie la possibilité de prendre plusieurs formes. Le polymorphisme peut apparaître sous différentes formes, a titre d'exemple le polymorphisme paramétrique est l'habilité de plusieurs objets de classes différentes de répondre au même message de différentes manières; en d'autres termes le message peut être défini dans différentes classes par différentes manières d'exécution, comme l'indique la figure 2.6

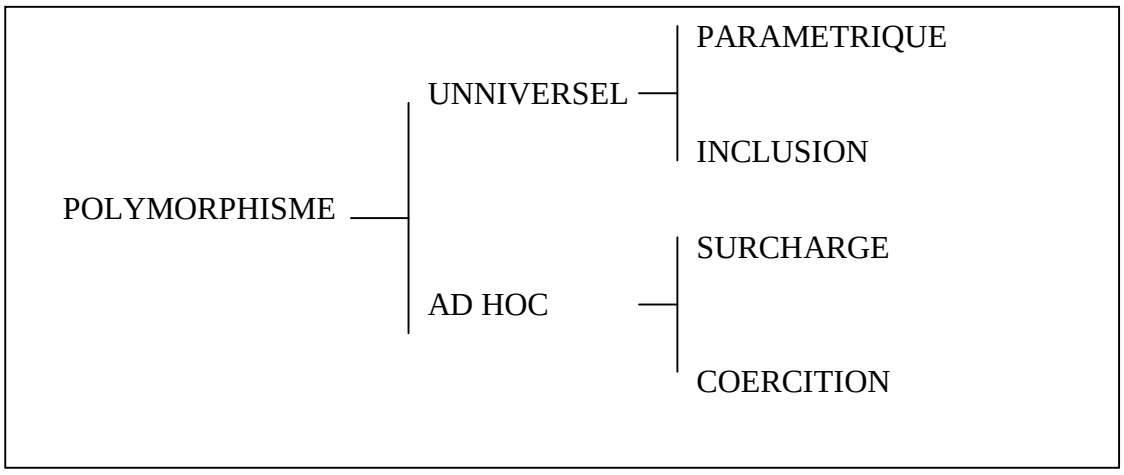


Figure 2.6 différentes formes de polymorphisme

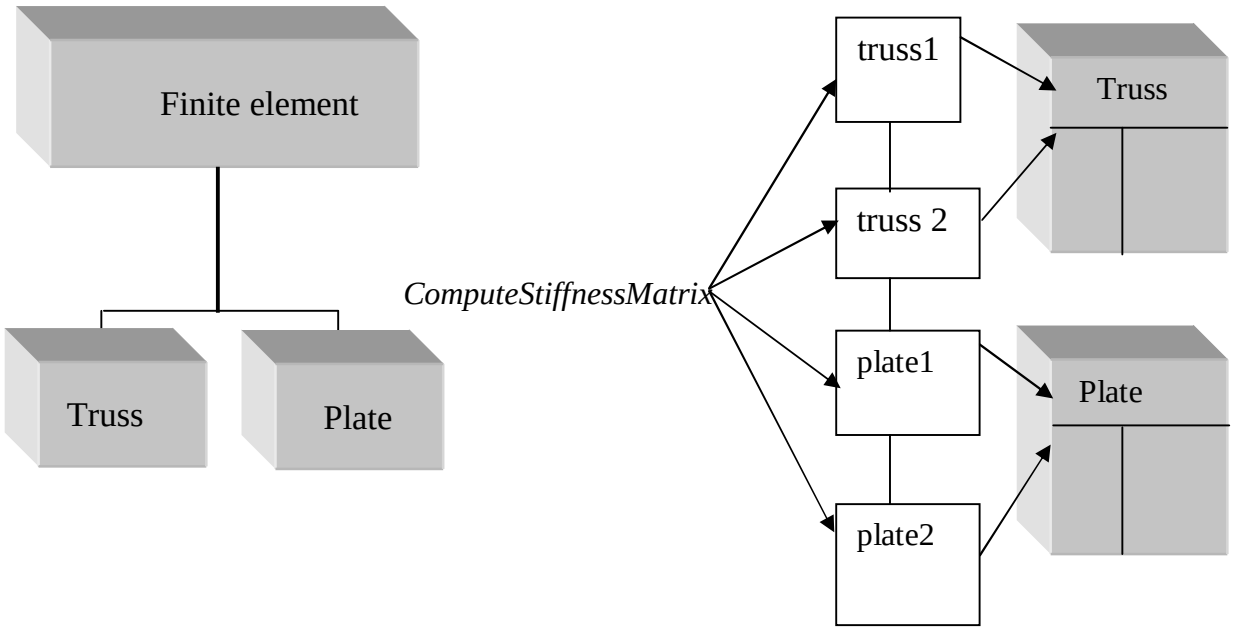


Figure 2.7 Polymorphisme et l'appel calculé

Le polymorphisme est aussi la possibilité de voir par exemple une variable (instance) de la classe *matrix_sym_skyline comme également appartenant à la classe matrix-sym puisqu'elle en possède toutes les caractéristiques.* Il y a inclusion dans classes dans leurs classes parente. Un objet de la classe *matrix_sym_skyline* peut être affecté à toute variable de classe *matrix_sym*. Inversement, toute variable de type *matrix_sym* peut être désignée comme une instance de *matrix_sym_skyline* ou de *matrix_sym_sparse*. Héritage et polymorphisme sont donc des notions étroitement liées.

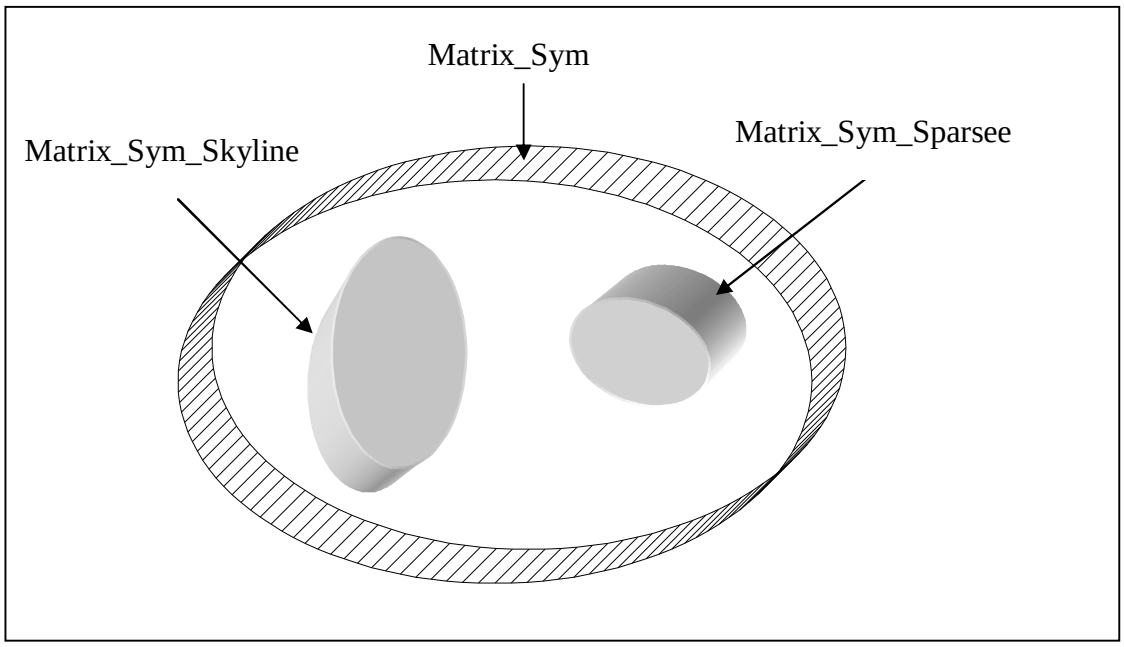


Figure 2.8 inclusions des sous-classes dans leur classe parente

Ce pendant un problème se pose : nous venons de voir d'une part qu'un objet peut être en fait classe variable et inconnue à la compilation et que d'autre part, on peut surcharger les méthodes héritées. Comment dans ces conditions pouvons nous faire pour qu'un appel Add de *matrix_sym_skyline* si nous ayons à faire à une matrice profil ou à Add de *matrix_sym_sparse* si c'est une matrice à stockage éparse ? C'est ce qu'on appelle la liaison dynamique (ou encore méthode virtuelle, late bending).

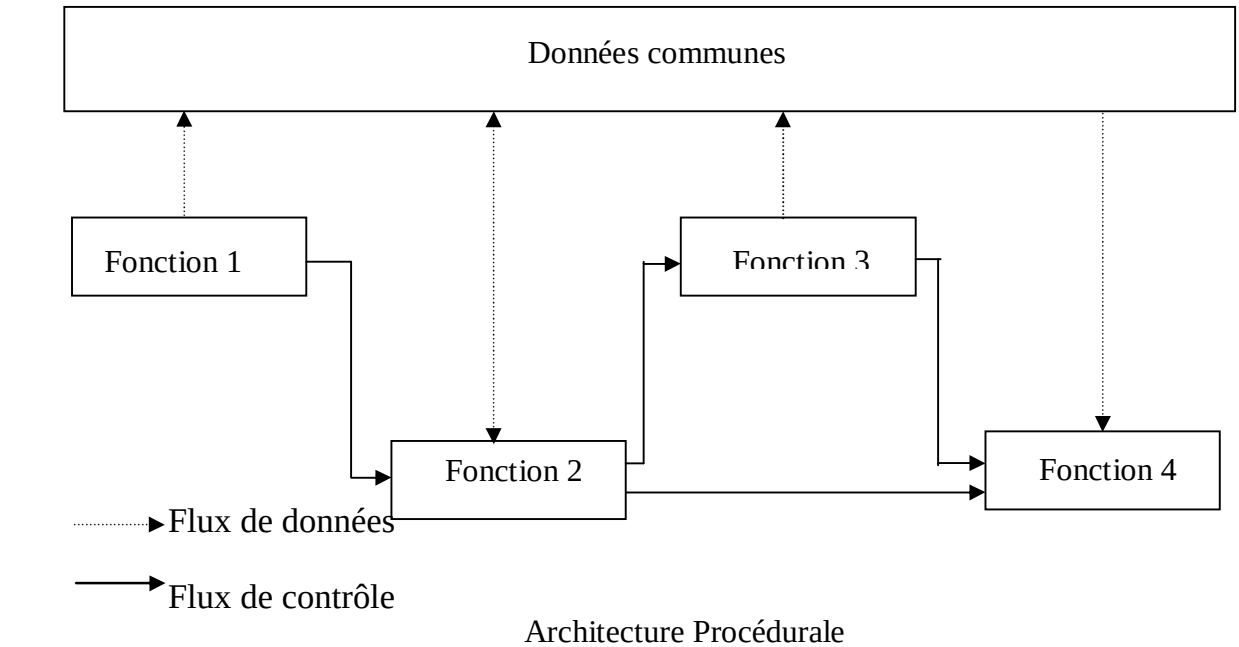
2.2.4 Liaison dynamique

On se rend compte que la liaison ne peut être réalisée qu'à l'exécution, au dernier moment, l'orque la classe vraie de l'objet est connue (et pas seulement la classe générale). On ne détaillera pas la façon avec laquelle ce mécanisme est implémenté dans les LOO mais ils convient de savoir que chaque instance possède un attribut caché qui indique sa classe vraie ainsi que la table des adresses de ses méthodes virtuelle (TMV). Un appel à Add déclenchera un appel indirect à la méthode dont l'adresse dépend de la classe vraie de l'objet.

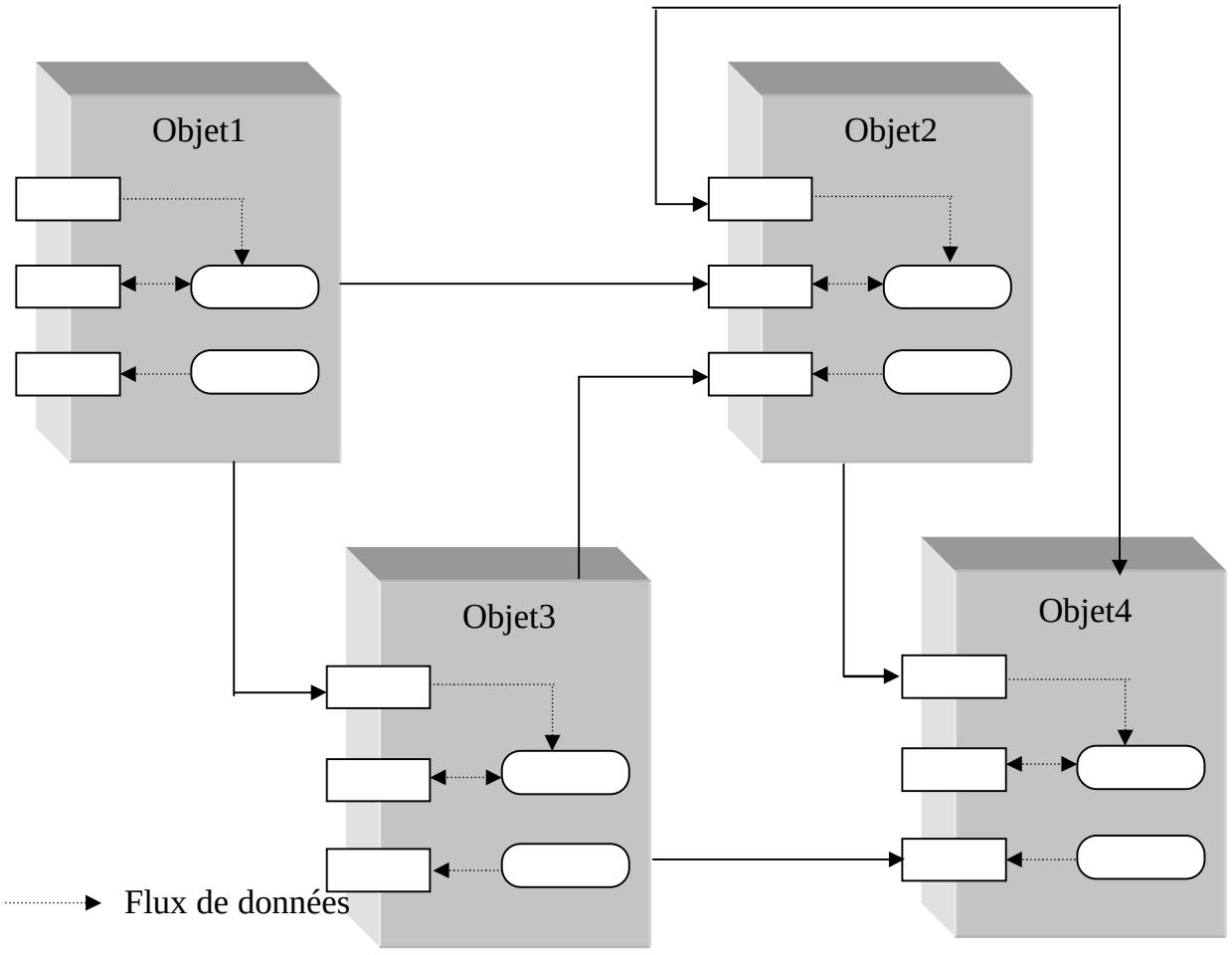
Au prix d'un léger surcoût dû à l'adressage indirect, la liaison dynamique offre une souplesse de programmation incomparable. On arrive ainsi à un bon compromis entre la liberté totale du Fortran où l'on peut mettre toute donnée *(integer, real, double, character)* dans toute variable et le typage strict du C Pascal ou Ada.

2.2.5 Différent type de mode de programmation

Il y a deux modes de programmation, programmation procédurale et programmation orientée objet la figure 2.9 illustre la différence entre les deux.



Architecture Procédurale



Architecture Orientée Objets

Figure 2.9 différent type de mode de programmation

CHAPITRE 3
LA PROGRAMMATION ORIENTEE OBJET EN ELEMENTS FINIS EXECUTION EFFICACE EN C++

3.1 INTRODUCTION [12] :

Il est clair que pendant plusieurs années, l'amélioration de la modularité du code en éléments finis demande une organisation des données, plusieurs auteurs ont fait de leur mieux pour améliorer l'arrangement des données dans le contexte du langage fortran [13],[21],[30],[36],[37].

La suffisance du concept de la Programmation Orientée Objet dans l'efficacité et la compréhension de l'organisation des données dans les programmes d'analyse numérique ont été seulement étudiés récemment. Baugh et Rehak [3] révèle la séquentialité ainsi que la nature contraignante de l'exécution classique des algorithmes, et ils ont introduit plus de flexibilité dans l'organisation du champ de données.

Les premiers efforts qui touchent a l'architecture des logiciel en élément finis ont été présenté par forde et al [16] et d'autre auteurs [11],[38], les autres progrès sont dus à Fenves [15], Miller [28] et Desjardin et Fafard [9].

Les principes fondamentaux de l'application de la technique d'Orientée Objet à la méthode des éléments finis ont été présentés par [38], ces principes sont implémentés dans un programme éléments finis écrits en smalltalk [11]. Le désigne de ce programme résulte sur une haute modularité, facilité de compréhension, et un programme extensible ces qualités sont basées sur :

- l'encapsulation des données :

Un objet est une variable munie d'une série de données de type protected (attribut) qui permet d'exécuter ses méthodes et de gérer ses opérations d'une manière autonome.

- l'héritage des classes :

Les différents types d'objets (ou classes) sont organisés en une simple structure hiérarchique qui évite la duplication des codes.

- La non-anticipation :

C'est complètement nouveau comme concept de programmation destiné à la programmation en éléments finis. C'est ce qu'on appelle la nouvelle modularité dans le temps; les méthodes doivent être indépendantes des contextes. Les deux concepts de base fournissent une forte structure de donnée ; cependant ils ne règlent pas le problème de la séquentialité des

opérations, mais le concept de la non anticipation le fait. C'est complètement nouveau comme concept de programmation destinée à la programmation en élément finis, il affirme ceci :
Le contenu d'une méthode ne doit pas se poser sur une quelconque supposition de l'état de la variable.

Exemple : la classe Element est en possession de l'attribut de locationArray, c'est là où on stocke la numérotation de l'équation de ses degrés de libertés, l'obtention de ce vecteur de la classe Element peut ce faire par deux approches :

a) - L'approche standard ; le programmeur envoie à la classe Element (disant element1)

Le message : loc=element1⟶ formlocationArray( ) ceci dans le cas ou on sais que l'allocation d'un tableau n'a pas été formé.

Ou bien le message : loc=element1 ⟶ returnlocationArray( ) ceci dans le cas ou on sait que l'allocation d'un tableau a été formée (le tableau existe).

Le trouble dans cette approche, c'est qu'il est très difficile de connaître quant est ce qu'une variable ou un objet doit être initialisé, ceci entraîne souvent le programmeur à une anticipation d'opération très délicates, avec une pensée en tête « je la calcule maintenant du moment que je sais que je vais en avoir besoin après ». Cette attitude tente d'organiser les codes classiques en des véritables séquences d'opérations rigides empêchant ainsi la facilité d'extension future.

b) – L'approche non anticipation : le programmeur ne doit pas faire aucune supposition sur l'état de l'attribut locationArray ceci pour tous les cas en envoyant juste le message
loc=element1 ⟶ givelocationArray( ).
La classe Element est munie d'une méthode qui est givelocationArray( ) cette dernière est extrêmement simple, elle est utilisée dans toutes les situations.

```
IntergeArray * Element : : givelocationArry ( )
{
 if (location Array == NULL)
        return this ⟶ formLocation Array () ;
else
        return locationArray ;
}
```

Figure 3.1 non anticipation

Cette méthode consiste en un test : l'élément vérifie si l'attribut locationArray est nul (il n'existe pas) ou non (il existe), s'il n'existe pas l'élément exécute la méthode formlocationArray( ),puis renvoie le tableau. Si par contre il existe, le tableau est renvoyé, clairement le message ci-dessous : Element1⟶ givelocationArray( ), peut être inséré dans n'importe quel endroit dans le programme et l'élément va toujours répondre d'une manière adéquate.

Les méthodes d'un objet peuvent être exécutées dans n'importe quelle place dans le programme, sans aucune opération séquentielle contraignante du code classique. Dans [14] la modularité des logiciels est établie à partir du point de vue de maintenance du code comparativement à l'approche procédurale où l'amélioration est reportée sur toutes les parties du code.

Premièrement : dans la compréhension du code: le code est facile à lire, plus homogène et beaucoup plus auto-descriptif.

Deuxièmement : dans l'extensibilité du code, des caractéristiques additionnelles sont faciles à introduire

Troisièmement : dans le débogage du code, l'approche favorise un test local des nouvelles parties du code et elles sont moins disposées à l'erreur (un nombre minimum est transmis à la subroutine)

Le langage Smalltalk met à la porte un outils parfait pour un programme prototype en élément finis, il a aidé les auteurs à se débarrasser des habitudes de la programmation procédurale (Fortran), de comprendre le potentiel du concept de la Programmation Orientée Objet, le choix adéquats des objets à introduire, assigner les bonnes opérations aux objets et de gravir avec des règle profitable de programmation (principalement le principe de la non-anticipation).

Cependant le Smalltalk est un langage très long en exécution par exemple un maillage de 25 éléments bar d'un treillis, est solutionné à peut près 100 fois plus lent qu'un code en Fortran pour un même type de maillage, et ce facteur augmente de plus en plus qu'on raffine le maillage.

Le gain de l'efficacité numérique demande un passage à un autre langage. Le C++ a été choisi pour deux raisons :

Premièrement, pour son efficacité intrinsèque ; c'est à dire qu'il est conçu avec l'usage combiné des avantages de la Programmation Orientée Objet et avec l'efficacité de calcul du langage C.

La seconde raison, c'est que le C++ est vraisemblablement au moment de notre recherche le langage Orienté Objet le plus utilisé dans les années à venir ceci est du à sa compatibilité avec la forte utilisation du langage C et du fait que c'est un langage qui est dans le domaine public.

3.2 L'architecture du concept de la programmation Orientée Objet dans un langage classique [12]

La programmation Orientée objet est intrinsèquement moins efficace que la programmation procédurale. La programmation orientée objet facilite le design de haut niveau du logiciel : L'organisation du code est attendue à adhérer le plus possible à la manière humaine de penser. La programmation procédurale est beaucoup plus orientée exécution des calculs, exactement comme la machine qui organise les opérations et les données ceci généralement plus efficace point de vue calcul.

Cette section explique comment un programmeur en c++ peut néanmoins gravir avec efficacité d'exécution comparable à celle des programmes en fortran. Pendant l'utilisation extensive des caractéristiques de l'orienté objet (haut niveau) du langage, la base qui nous permet d'atteindre ce niveau d'efficacité est le signe le plus classique du c++ qui est du à la similarité de ce langage avec le langage procédurale.

L'efficacité numérique dans le langage c++ peut être atteint en deux manières complémentaires.

- en bénéficiant de l'existence des différents niveaux d'abstraction (section 2.1)
- en organisant les moules en des librairies (section 2.2)

3.2.1 Les niveaux d'abstraction en c++

La programmation orientée objet favorise une haute abstraction comme clés de la modularité et la réutilisabilité du code cependant dans certain cas, l'efficacité peut être obtenue seulement par une personnalisation d'une portion du code, pour une utilisation spatiale, utilisant ainsi le bas niveau et ceci est au frais de la réutilisabilité du code. Cette demande est strictement rigoureuse pour les opérations impliquant les matrices et les tableaux.

Dans le but d'être efficace, le langage doit se munir de la programmation bas niveau ; C++ le fait comparativement au fortran, le c++ enveloppe entièrement le haut, égal et bas niveau d'abstraction

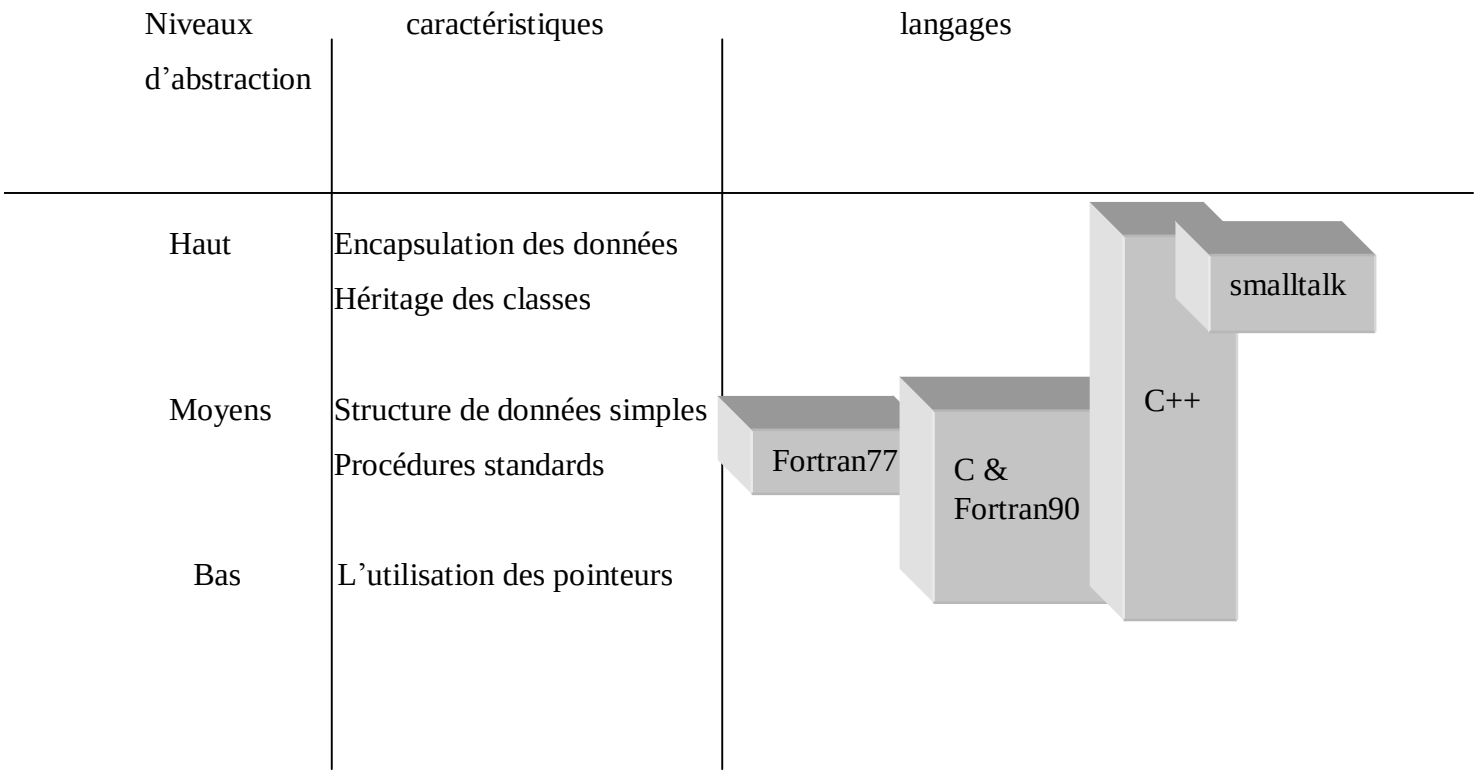


Figure 3.2 les niveaux d'abstraction en programmation

Au haut niveau d'abstraction, les caractéristiques de l'orientée objet comme l'encapsulation des données et l'héritage des classes sont introduit. Le niveau d'abstraction moyen correspond à la programmation procédure (appelle des procédures plutôt que l'envois des messages à des objets). Le bas niveau d'abstraction conduit vers la possibilités de pointer directement à la mémoires du micro (par exemple les indices des tableaux).

La bonne règle pour un design de haute qualité du code en c++ est de s'appuyer d'une manière extensive sur les caractéristiques de la haute abstraction de la programmation orientée objet cependant accepter d'optimiser l'implantation de quelques opérations (bas niveau) une telle tactique s'accorde bien à la Méthode des éléments finis.

3.2.2 Programmations en complétant les librairies

Comme les programmeurs en smalltalk, les programmeurs en c++ peuvent décider de s'appuyer forttement sur l'héritage dans le but d'améliorer la génèrecité et d'éviter la duplication des codes; cette approche est bonne pour un prototypage fort, et elle est aussi retenue par quelques programmeurs en c++ pour construire une large programmation d'environnement [15].

Compter systématiquement sur l'héritage n'est probablement pas la nature du c++ ; l'héritage favorise le polymorphisme; c'est la raison d'envoyer le même message à des objets différents (de types différents) le polymorphisme exige la liaison dynamique; qui est la possibilité de retarder jusqu'au temps d'exécution la série d'édition de liens par un message entre la méthode appelante et la méthode appelée ? Du moment que la liaison dynamique est une opération au temps d'exécution et non pas au temps d'édition de liens, elle introduit une augmentation au temps d'exécution, plus encore, l'implémentation du polymorphisme en c++ est souvent à la porte de la difficulté. Pour cette raison dans les codes c++ standard, l'héritage des classes est utilisé seulement quand il y a un besoin très fort plutôt que systématiquement ceci conduit à une programmation de classes plus autonomes, plus compacte et plus efficace, quoique moins générique que dans le langage smalltalk. Ainsi le résultat de l'arborescence des classes ont peut niveau d'héritage que celui du langage smalltalk.

Moralité en c++, on programme en élargissant les librairies plutôt que l'environnement (les librairies consistes en des classes non pas en des subroutines).

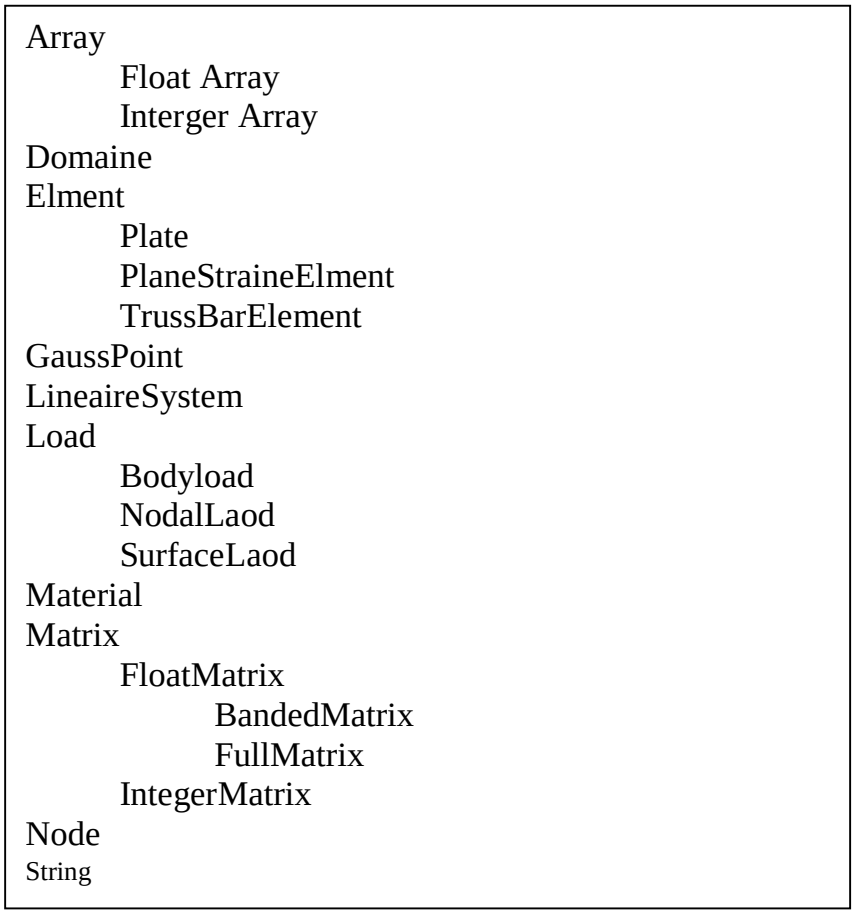


Figure 3.3 la hiérarchie des classes élément finis en C++

Conclusion

La conception orientée objet avec le langage C++ est une bonne solution car il enveloppe entièrement le haut, égal et bas niveau d'abstraction, Nous proposant dans le prochain chapitre la théorie des plaques isotropes et orthotropes. Quelques classes essentielles qui décrivent la manière dont on a implémenté l'élément de plaque en c++ sont décrites en annexe.

CHAPITRE IV THEORIE D'ELASTICITE ANISOTROPE ET DES PLAQUES EN FLEXION

Introduction :

La théorie de l'élasticité isotrope couramment utilisée, ne permet pas de caractériser pleinement le comportement des matériaux composites présentant des réponses différentes suivant les directions d'actions des efforts. Il est donc nécessaire d'entrer dans le détail de l'élasticité anisotrope.

4.1 Notion de continuité et d'homogénéité : [7]

La continuité:[7][17]

un corps est dit continu s'il occupe complètement tout son volume, et si deux points jointifs de sa matière le restent indéfiniment. Ceci s'exprime mathématiquement par le fait qu'en tout point M du milieu, il existe une masse volumique (la limite existe)

$$r(M) = \lim_{\Delta V \to 0} \frac{\Delta m}{\Delta V} \tag{4.1}$$

Δm **:** étant la masse du petit élément de volume $\mathbf{\Delta v}$ entourant le point M.

L'homogénéité : un corps est homogène si la masse volumique $r(M)$ est identique en tout point de son volume.

4.2 Notion d'anisotropie [7]: L'anisotropie influe sur les propriétés associées à une direction, et en particulier, sur le nombre des caractéristiques mécaniques entrants dans la relation contraintes-déformations.

4.3 Etats de contraintes et de déformations:

Définition d'un état de contrainte: l'état de contrainte en un point d'un corps, repéré dans un trièdre Oxyz direct est parfaitement déterminé si les valeurs des composantes du tenseur des contraintes sont connues en ce point.

$$\|s\| = \begin{Vmatrix} s_x & t_{xy} & t_{xz} \\ t_{yx} & s_y & t_{yz} \\ t_{zx} & t_{zy} & s_z \end{Vmatrix}$$

Par des équations de moments, on trouve: $t_{yx} = t_{xy}$, $t_{zx} = t_{xz}$, $t_{zy} = t_{yz}$, ce qui réduit à six le nombre de composantes indépendantes. Celles-ci peuvent alors prendre la forme matricielle suivante : $[s]^T = [s_x \quad s_y \quad s_z \quad t_{yz} \quad t_{zx} \quad t_{xy}]$.

La connaissance de six contraintes permet, par des formules de rotation d'axes, de déterminer l'état de contrainte dans une direction quelconque, autour du même point.

Définition d'un état de déformation : l'état de déformation en un point d'un corps est parfaitement déterminé si les valeurs des composantes du tenseur de déformations sont connues en ce point :

$$\|e\| = \begin{Vmatrix} e_x & 1/2g_{xy} & 1/2g_{xz} \\ 1/2g_{yx} & e_y & 1/2g_{yz} \\ 1/2g_{zx} & 1/2g_{zy} & e_z \end{Vmatrix}$$

Ce tenseur se réduit également à six composantes indépendantes, qui s'écrivent sous la forme matricielle ci-après :

$$[e]^T = [e_x \quad e_y \quad e_z \quad g_{yz} \quad g_{zx} \quad g_{xy}]$$

4.4 Relation contraintes - déformations :[24] [35]

Toute structure soumise à un système de forces extérieures, est le siége d'un état de contraintes, qui engendre un certain état de déformation, et inversement. Ainsi, dans le système d'axes de référence choisi, et en utilisant la notation matricielle, chaque composante du vecteur $[s_{ij}]$ des contraintes, est en fonction des composantes du vecteur e_{ij} des déformations :

$$s_{lk} = f\ (e_{ij}) \tag{4.2}$$

HYPOTHESE 1 : en faisant l'hypothèse que le matériau est continu, cette fonction se développe par la formule de TAYLOR :

$$s_{lk} = f_{lk}(0) + e_{ij}\frac{\partial f_{lk}}{\partial e_{ij}}(0) + \frac{e_{ij}^2}{2}\frac{\partial^2 f_{lk}}{\partial e_{ij}^2}(0) + \mathbf{L} \qquad (4.3)$$

Les termes différentiels sont les coefficients caractéristiques du matériau.

HYPOTHESE 2 : en admettant l'existence d'un état neutre, qu'un état de déformations nulles correspond à un état de contraintes nulles, hypothèse qui, physiquement, est toujours valable, les coefficients $f_{lk}(0)$ disparaissent.

Les termes différentiels de premier ordre sont généralement les seuls connus.

Ils forment une famille des caractéristiques du matériau, à laquelle appartiennent, par exemple, le module longitudinal E, et le coefficient de Poisson υ .

Les termes du second ordre représentent les variations des coefficients définis ci – dessus, en fonction des déformations. Ils disparaissent, ainsi que les termes différentiels d'ordre supérieur, en admettant le principe de linéarité interne, c'est-à-dire que le matériau obéit à la loi de HOOKE généralisée.

La relation contrainte – déformation s'écrit dans ce cas : $s_{lk} = e_{ij}\frac{\partial f_{lk}}{\partial e_{ij}}(0)$

Et en développant :

$$\begin{bmatrix} s_x \\ s_y \\ s_z \\ t_{yz} \\ t_{zx} \\ t_{xy} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} \end{bmatrix} * \begin{bmatrix} e_x \\ e_y \\ e_z \\ g_{yz} \\ g_{zx} \\ g_{xy} \end{bmatrix}$$

Soit encore : $[s] = [D] * [e]$

$[D]$: Est la matrice des rigidités, ou rigidité du matériau. La matrice inverse $[D]^{-1}$, permettant de calculer les déformations en fonction des contraintes porte le nom de matrice de complaisance ou de souplesse.

HYPOTHESE 3 : en s'appuyant sur l'hypothèse que le matériau est homogène , il est possible d'affirmer que la matrice de rigidité est identique en tout point du corps, et qu'il en est de même pour la matrice de souplesse du matériau considéré .

Symétrie de la matrice $[D]$: par des considérations thermodynamiques, la matrice $[D]$ se révèle symétrique : et par conséquence, elle ne comporte que 21 coefficients indépendants parmi les 36 la composant.

En effet, en thermodynamique des milieux continus, il est possible de montrer que les contraintes dérivent du potentiel élastique, c'est – à – dire que :

$$s_x = \frac{\partial \bar{V}}{\partial e_x} \qquad s_y = \frac{\partial \bar{V}}{\partial e_y} \quad \text{L L L} \quad t_{xy} = \frac{\partial \bar{V}}{\partial g_{xy}} \tag{4.4}$$

Le potentiel élastique étant l'énergie de déformation par unité de volume :

$$\bar{V} = \frac{dU}{dV} = 1/2\,[s]^T\,[e] \tag{4.5}$$

$$\bar{V} = 1/2\,\left(s_x.e_x + s_y.e_y + s_x.e_z + t_{yz}.g_{yz} + t_{zx}.g_{zx} + t_{xy}.g_{xy} \right) \tag{4.6}$$

Avec la relation : $[s] = [D] * [e]$

$$s_x = a_{11}\,e_x + a_{12}\,e_y + a_{13}\,e_z + a_{14}\,g_{yz} + a_{15}\,g_{zx} + a_{16}\,g_{xy} \tag{4.7}$$

$$s_y = a_{21}\,e_x + a_{22}\,e_y + a_{23}\,e_z + a_{24}\,g_{yz} + a_{25}\,g_{zx} + a_{26}\,g_{xy} \tag{4.8}$$

En calculant : $\dfrac{\partial s_x}{\partial e_y} = a_{12} = \dfrac{\partial^2 \bar{V}}{\partial e_x \partial e_y}$

De même $\dfrac{\partial s_y}{\partial e_x} = a_{21} = \dfrac{\partial^2 \bar{V}}{\partial e_y \partial e_x}$

Etant donné la neutralité de l'ordre de dérivation : $a_{12} = a_{21}$

La même symétrie se démontre d'une façon semblable pour tous les coefficients a_{ij} $(i \neq j)$.

4.5 Symétrie élastique *:*

Principe : si la composition interne du matériau possède une symétrie, de quelques formes que ce soit, cette même symétrie s'observe dans ses propriétés élastiques ; ceci va simplifier les équations de la loi de HOOKE, et réduire le nombre de composantes indépendantes de la matrice $[D]$.

Simplification : pour trouver les simplifications résultantes d'une symétrie géométrique quelconque de constitution, la méthode suivante est souvent utilisée :

Le corps est repéré dans un premier système de coordonnées XYZ, puis dans un second système X'Y'Z', déduit du précédent par la symétrie envisagée.

Les directions des axes correspondants des deux systèmes doivent être équivalentes vis - à-vis des propriétés élastiques, ce qui implique que les expressions de la loi de HOOKE, et du potentiel élastique, ont la même forme dans les deux systèmes, ce qui implique également que les constantes élastiques, correspondant dans ces expressions, sont égales.

Par identification des termes correspondants dans les expressions de la lois de HOOKE, ou du potentiel élastique (l'une dans XYZ, l'autre dans X'Y'Z') des relations supplémentaires sont obtenues entre les composantes de la matrice $[D]$, ce qui diminue le nombre de constantes indépendantes.

Cas particuliers :

Matériau Orthotrope : un tel matériau possède des propriétés équivalentes au point de vue élastique dans des directions symétriques par rapport à trois plans orthogonaux.

Soit XOY, XOZ et YOZ ces plans, avec la méthode déjà citée, la matrice

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{yz} \\ \tau_{zx} \\ \tau_{xy} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 & 0 \\ & a_{22} & a_{23} & 0 & 0 & 0 \\ & & a_{33} & 0 & 0 & 0 \\ & & & a_{44} & 0 & 0 \\ & \mathbf{s} & \mathbf{y} & \mathbf{m} & a_{55} & 0 \\ & & & & & a_{66} \end{bmatrix} * \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{yz} \\ \gamma_{zx} \\ \gamma_{xy} \end{bmatrix}$$

se simplifie et devient :

Les axes X, Y, Z intersection des 3 plans de symétrie, sont dits **axes principaux de l'orthotropie.**

La relation inverse : $[e]=[D]^{-1}*[s]$, s'écrit :

$$\begin{cases} e_x = \frac{1}{E_x} s_x - \frac{u_{xy}}{E_y} s_y - \frac{u_{xz}}{E_z} s_z \\ e_y = \frac{u_{yx}}{E_x} s_x + \frac{1}{E_y} s_y - \frac{u_{yz}}{E_z} s_z \\ e_z = -\frac{u_{zx}}{E_x} s_x - \frac{u_{zy}}{E_y} s_y + \frac{1}{E_z} s_z \\ g_{yz} = \frac{1}{G_{yz}} t_{yz} \\ g_{zx} = \frac{1}{G_{zx}} t_{zx} \\ g_{xy} = \frac{1}{G_{xy}} t_{xy} \end{cases} \quad (4.9)$$

Les matrices de rigidité et de souplesse ne comportent que 9 coefficients indépendants (4 dans le cas de problèmes plan) pour un materiau orthotrope.

Matériau isotrope : dans ce cas, toutes les directions, et tous les plans sont élastiquement équivalents. Le nombre de constantes élastiques est alors égal à 2 :

E et u avec $G = \frac{E}{2(1+u)}$

$$\begin{aligned} e_x &= \frac{1}{E}\left(s_x - u s_y - u s_z\right) & g_{yz} &= \frac{t_{yz}}{G} \\ e_y &= \frac{1}{E}\left(-u s_x + s_y - u s_z\right) & g_{zx} &= \frac{t_{zx}}{G} \\ e_z &= \frac{1}{E}\left(-u s_x - u s_y + s_z\right) & g_{xy} &= \frac{t_{xy}}{G} \end{aligned} \quad (4.10)$$

4.6 Théories des plaques en flexion

4.6.1 Définition d'une plaque

Une plaque est un solide élastique dont une dimension, selon l'épaisseur, est petite en comparaison des deux autres, et qui généralement comporte un plan de symétrie au milieu de l'épaisseur que nous appelons surface moyenne. Par convention, cette surface sera le plan (x-y), l'axe (o-z) correspond à l'axe transversal selon l'épaisseur [23].

Une plaque peut être constituée d'un matériau homogène, ou être obtenue par l'empilement de différentes couches de matériaux orthotropes [2].

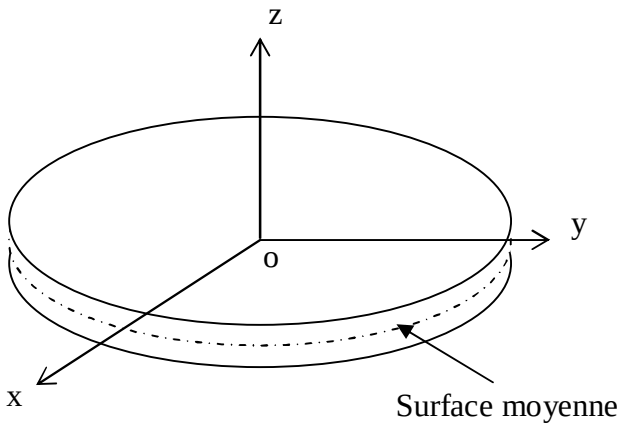


Figure 4.1 Portion d'une Plaque.

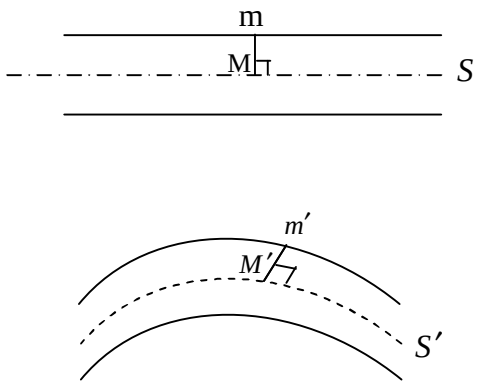
4.6.2 Hypothèses :

La théorie des plaques [2][23] repose sur les hypothèses suivantes :

H.1 : Les contraintes normales s_{zz} sont négligeables par rapport aux autres composantes de contraintes : $s_{zz} = 0$.

H.2 : Les pentes de la surface moyenne après déformation, sont supposées petites par rapport à l'unité (Petite déflexion du plan moyen).

H.3 : Les points situés sur une normale à la surface moyenne avant déformation, restent sur cette normale au cours de la déformation.



Les hypothèses H.1 et H.3 correspondent dans le cas bidimensionnel aux hypothèses classiques de la résistance des matériaux avec conservation des sections droites, elles correspondent aussi à la définition d'un état plan de contrainte.

La théorie des plaques dans laquelle on néglige les effets de cisaillement transversal est due à Kirchoff (cas des plaques minces).

Il existe des théories des plaques qui permettent la prise en compte du cisaillement transversal : ce sont les théories de Hencky, Reissner et Mindlin ; Dans ce cas, il faut prendre en compte les déformations de cisaillement transversal et alors les fibres normales à la surface moyenne avant déformation ne le restent pas au cours de la déformation [23].

Pour les plaques orthotropes et composites, le rôle des déformations de cisaillement transversal dépend non seulement des caractéristiques géométriques (l'élancement L/h), mais également des caractéristiques mécaniques représentées par le rapport E/kG .

Où :

E : module de Young.

G : module de cisaillement transversal (CT).

k : facteur de correction de cisaillement transversal.

La théorie de Kirchoff (théorie classique des plaques minces), peut être interprétée comme un cas particulier de la théorie de Reissner/Mindlin, ainsi un bon modèle élément fini basé sur la théorie de Reissner/Mindlin devra donner des résultats en accord avec la théorie de Kirchoff si l'influence du cisaillement transversal est faible [2].

4.6.3 Conventions de signe pour les déplacements et rotations

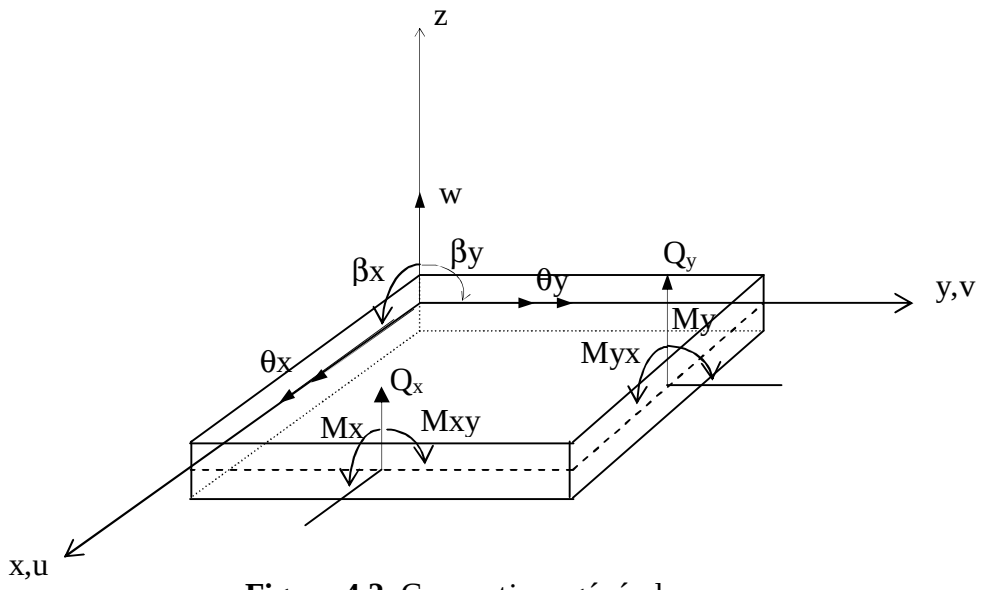


Figure 4.2 Conventions générales.

Soient les déplacements dans le plan u et v, le déplacement transversal w et les rotations b_x et b_y, ou q_x et q_y. On a évidemment

$$b_x = q_y \qquad b_y = -q_x \tag{4.11}$$

b_x et b_y : les rotations de la normale à la surface moyenne dans les plans (x-z) et (y-z) respectivement.

4.6.4 Relations cinématiques

4.6.4.1 Champ de déplacements

Dans la théorie de Hencky-Mindlin, (prise en compte du cisaillement transversal), on se donne un modèle de déplacements basé sur trois variables indépendantes :

le déplacement transversal $w(x, y)$ et les deux rotations $b_x(x, y)$ et $b_y(x, y)$.

Le champ des déplacements s'exprime alors en fonction de ces trois variables par la relation suivante :

$$\begin{aligned} u(x,y,z) &= z\, b_x(x,y) \\ v(x,y,z) &= z\, b_y(x,y) \\ w(x,y,z) &= w(x,y) \end{aligned} \tag{4.12}$$

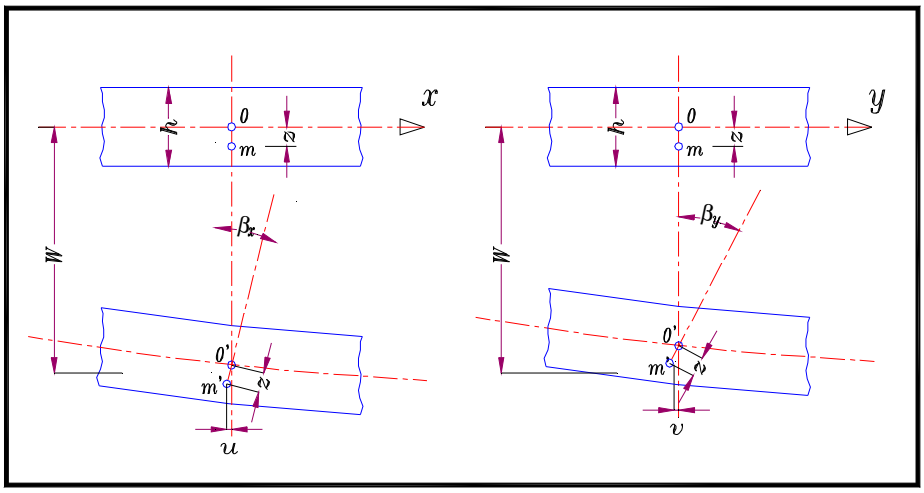


Figure 4.3 Rotations b_x et b_y

4.6.4.2 Champ de déformations

L'état de déformation en coordonnées cartésiennes est défini par les expressions suivantes :

$$\begin{aligned} e_x &= \frac{\partial u}{\partial x} \\ e_y &= \frac{\partial v}{\partial y} \\ 2e_{xy} &= g_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \\ 2e_{xz} &= g_{xz} = \frac{\partial u}{\partial z} + \frac{\partial w}{\partial x} \\ 2e_{yz} &= g_{yz} = \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} \end{aligned} \qquad (4.13)$$

Après substitution des déplacements dans (4.13), nous obtenons les composantes du tenseur de déformations en fonction des trois degrés de liberté w, b_x, b_y :

$$\begin{aligned} e_x &= z\frac{\partial b_x}{\partial x} \\ e_y &= z\frac{\partial b_y}{\partial y} \\ g_{xy} &= z\left(\frac{\partial b_x}{\partial y} + \frac{\partial b_y}{\partial x}\right) \\ g_{xz} &= b_x + \frac{\partial w}{\partial x} \\ g_{yz} &= b_y + \frac{\partial w}{\partial y} \end{aligned} \qquad (4.14)$$

Le vecteur de déformation peut être décomposé en deux parties, l'une indépendante de z traduisant les déformations de cisaillement notée $\{e_c\}$ ou $\{g\}$, et l'autre partie $\{e_f\}$ dépendante de z représente les déformations de flexion :

$$\{e\} = \left\{\{e_f\}^T, \{e_c\}^T\right\}^T = \left\{z\{c\}^T, \{g\}^T\right\}^T \qquad (4.15)$$

$$\{e_f\} = \begin{Bmatrix} e_{xx} \\ e_{yy} \\ g_{xy} \end{Bmatrix} = z\{c\} \qquad \{e_c\} = \{g\} = \begin{Bmatrix} g_{xz} \\ g_{yz} \end{Bmatrix} = \begin{Bmatrix} b_x + \frac{\partial w}{\partial x} \\ b_y + \frac{\partial w}{\partial y} \end{Bmatrix} \qquad (4.16)$$

Avec :

$$\{c\}=\begin{Bmatrix} \dfrac{\partial b_x}{\partial x} \\ \dfrac{\partial b_y}{\partial y} \\ \dfrac{\partial b_x}{\partial y}+\dfrac{\partial b_y}{\partial x} \end{Bmatrix} \tag{4.17}$$

$\{c\}$: est le vecteur des variations de courbure.

D'où on peut écrire :

$$\{\bar{e}\}=\{\{c\}^T,\{g\}^T\}^T \tag{4.18}$$

4.6.5 Relations contraintes-déformations [2][23][35]

Nous considérons les relations linéaires entre les contraintes et les déformations (loi de Hooke généralisée). Pour les matériaux orthotropes, la relation liant les contraintes aux déformations s'écrit :

$$\{s\}=[C]\{e\} \tag{4.19}$$

telle que :

$[C]$: matrice de constantes élastiques.

Lorsque le cisaillement transversal est pris en considération, le vecteur de contraintes peut être également décomposé en une contribution de cisaillement $\{s_c\}$, et une contribution de flexion $\{s_f\}$:

$$\begin{Bmatrix}\{s_f\}\\ \{s_c\}\end{Bmatrix}=\begin{bmatrix}[C_f] & [0]\\ [0] & [C_c]\end{bmatrix}\begin{Bmatrix}e_f\\ g\end{Bmatrix} \tag{4.20}$$

$$\{s_f\}=\{s_{xx}, s_{yy}, s_{xy}\}^T \qquad \{s_c\}=\{s_{xz}, s_{yz}\}^T \tag{4.21}$$

avec :

$$[C_f]=\begin{bmatrix} Q_{11} & Q_{12} & 0\\ Q_{12} & Q_{22} & 0\\ 0 & 0 & Q_{66}\end{bmatrix} \qquad [C_c]=\begin{bmatrix} Q_{44} & 0\\ 0 & Q_{55}\end{bmatrix} \tag{4.22}$$

$$Q_{11}=\frac{E_1}{1-n_{21}n_{12}} \qquad Q_{12}=\frac{n_{12}E_2}{1-n_{21}n_{12}}$$

$$Q_{22} = \frac{E_2}{1-n_{21}n_{12}} \qquad Q_{66} = G_{12} \qquad (4.23)$$

$$Q_{44} = G_{13} \qquad Q_{55} = G_{23}$$

tels que :

E_1 : Module de Young dans la direction (x).

E_2 : Module de Young dans la direction (y).

n_{12} ,n_{21} : Coefficients de poisson.

G_{12}, G_{13}, G_{23} : Modules de cisaillement.

4.6.6 Relations efforts résultants-déformations [2][18][23]

Pour la conception et le calcul des éléments de la mécanique, il est souvent intéressant de connaître les efforts de résistance des matériaux.

Les moments résultants de flexion sont :

$$\{M\} = \begin{Bmatrix} M_{xx} \\ M_{yy} \\ M_{xy} \end{Bmatrix} = \int_{-h/2}^{+h/2} z \{s_{xx}, s_{yy}, s_{xy}\}^T dz = [D_f]\{c\} \qquad (4.24)$$

Les efforts tranchants sont :

$$\{Q\} = \begin{Bmatrix} Q_x \\ Q_y \end{Bmatrix} = \int_{-h/2}^{+h/2} \{s_{xz}, s_{yz}\}^T dz = [D_c]\{g\} \qquad (4.25)$$

avec :

h : L'épaisseur de la plaque.

$$[D_f] = \frac{h^3}{12} \begin{bmatrix} \frac{E_1}{1-n_{21}n_{12}} & \frac{E_2 n_{12}}{1-n_{21}n_{12}} & 0 \\ \frac{E_2 n_{12}}{1-n_{21}n_{12}} & \frac{E_2}{1-n_{21}n_{12}} & 0 \\ 0 & 0 & G_{12} \end{bmatrix} ; \quad [D_c] = hk \begin{bmatrix} G_{13} & 0 \\ 0 & G_{23} \end{bmatrix} \qquad (4.26)$$

k : Coefficient de correction de cisaillement transversal.

$[D_f]$: Matrice de rigidité à la flexion.

$[D_c]$: Matrice de rigidité cisaillement transversal.

4.6.7 Formulation en statique linéaire [20]

On aboutit, dans le cas de la formulation en statique et élasticité linéaire, au système d'équations :

$$[K]\{q\}=\{F\} \tag{4.27}$$

$[K]$: représente la matrice de rigidité de la structure.

$\{q\}$: vecteur de déplacements nodaux.

$\{F\}$: vecteur chargement extérieur.

Le théorème qui conduit à la relation (4.27), dans le cas d'une approche de type « déplacement », est celui des travaux virtuels.

4.6.7.1 Principe des travaux virtuels :

Soit un corps solide en équilibre sous l'action de forces : de volume f_i^V, de surface f_i^S, et des forces concentrées. Considérons un champ de déplacement virtuel δu_i cinématiquement admissible.

Le théorème des travaux virtuels [2], [17], [20], [23] exprime le bilan des travaux virtuels interne et externe, lorsque le corps est en équilibre :

$$\int_V s_{ij}\, de_{ij}\, dV = \int_V f_i^V\, du_i\, dV + \int_S f_i^S\, du_i\, dS + Q_i\, du_i \tag{4.28}$$

V : est le volume du corps ;

S : la surface extérieure du corps où les forces surfaciques sont appliquées ;

f_i^V : Forces volumiques ;

f_i^S : Forces surfaciques appliquées à la surface extérieure S du corps ;

Q_i : Forces concentrées ;

s_{ij} : Tenseur des contraintes ;

e_{ij} : Tenseur des déformations infinitésimales.

Sous forme matricielle, nous avons :

$$\int_V <de>\{s\}dV = \int_V <du>\{f_V\}dV + \int_S <du>\{f_S\}dS + \sum_i <du_i>\{Q_i\} \qquad (4.29)$$

$<e>$: Vecteur des déformations, transposé ;

$\{s\}$: Vecteur des contraintes ;

$<u>$: Vecteur des déplacements, transposé ;

$\{f_V\}$: Vecteur des forces volumiques ;

$\{f_S\}$: Vecteur des forces surfaciques ;

$\{Q_i\}$: Vecteur des forces concentrées.

CHAPITRE V : ELEMENTS FINIS DE PLAQUE EN STATIQUE ET EN DYNAMIQUE LINÉAIRE

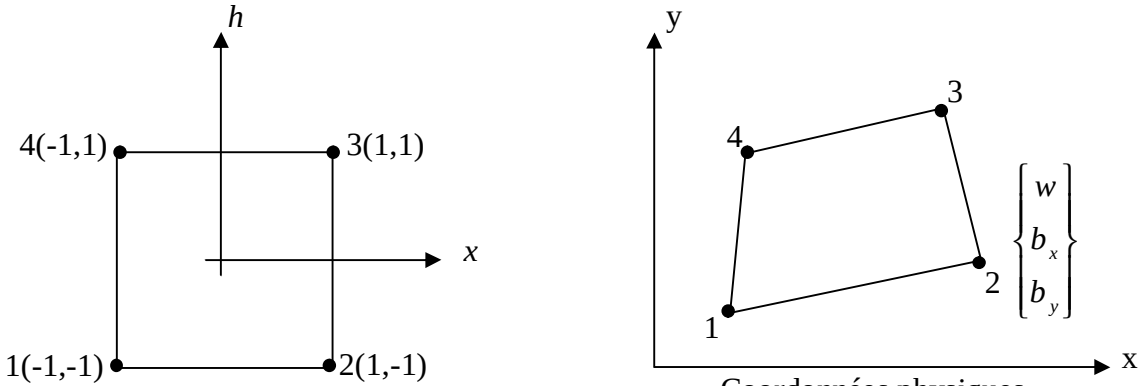
Dans ce chapitre nous exposons la formulation de l'élément quadrilatère isoparamétrique permettant l'analyse linéaire statique et dynamique des plaques. En outre, sont exposés les calculs de la matrice de rigidité, vecteur charge équivalent ainsi que la matrice de masse.

Introduction

La prise en compte du cisaillement transversal modifie largement les principes d'élaboration des éléments de plaque. En effet, leur formulation est basée sur l'approximation de trois champs indépendants : le déplacement transversal et les deux rotations. Par ailleurs, leur conformité ne requiert que la continuité C^0 de $w, b_x, et\ b_y$ [2],[23].

5.1 Discrétisation du champ de déplacements

Nous considérons des éléments de type quadrilatère aux quels nous appliquons la formulation isoparamétrique [2],[10],[23].



Coordonnées intrinsèques
Elément parent

Coordonnées physiques
Elément réel

Figure 5.1 Elément isoparamétrique de plaque avec cisaillement transversal.

Pour tout élément isoparamétrique quadrilatéral à quatre nœuds nous avons les approximations suivantes :

$$
\begin{aligned}
x &= [N]^T (x\ ,\ h\ )\{X\} \\
y &= [N]^T (x\ ,\ h\ )\{Y\} \\
w &= N^T(x,h)W \\
b_x &= N^T(x,h)\ b_x' \\
b_y &= N^T(x,h)\ b_y'
\end{aligned}
\qquad (5.1)
$$

Ou : $$w = \sum_{i=1}^{n_e} N_i w_i$$

$$b_x = \sum_{i=1}^{n_e} N_i b_{xi} \qquad (5.2)$$

$$b_y = \sum_{i=1}^{n_e} N_i b_{yi}$$

n_e : indique le nombre de nœuds par élément.

Les fonctions d'interpolation utilisées sont les fonctions d'interpolation habituelles des quadrilatères isoparamétriques.

Dans le cas du quadrilatère linéaire, on a :

$$N^T = \begin{bmatrix} N_1 & N_2 & N_3 & N_4 \end{bmatrix} \qquad (5.3)$$

avec : $$N_i(x,h) = \frac{1}{4}(1+x x_i)(1+h h_i) \qquad (5.4)$$

$x_i ou h_i$: prenant les valeurs (+1) ou (-1) suivant le nœud considéré.

$$W = \begin{Bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{Bmatrix} \qquad b_x = \begin{Bmatrix} b_{x1} \\ b_{x2} \\ b_{x3} \\ b_{x4} \end{Bmatrix} \qquad b_y = \begin{Bmatrix} b_{y1} \\ b_{y2} \\ b_{y3} \\ b_{y4} \end{Bmatrix} \qquad (5.5)$$

$$X = \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} \qquad Y = \begin{Bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{Bmatrix}$$

5.2 Discrétisation du champ de déformations [2],[23]

Par substitution de (5.1), dans les relations de déformations (4.16) et (4.17), on obtient les matrices d'interpolation des déformations de flexion et de cisaillement :

$$\{c\} = \{\bar{e}_f\} = \begin{Bmatrix} \frac{\partial b_x}{\partial x} \\ \frac{\partial b_y}{\partial y} \\ \frac{\partial b_x}{\partial y} + \frac{\partial b_y}{\partial x} \end{Bmatrix} = \begin{bmatrix} 0 & \frac{\partial N^T}{\partial x} & 0 \\ 0 & 0 & \frac{\partial N^T}{\partial y} \\ 0 & \frac{\partial N^T}{\partial y} & \frac{\partial N^T}{\partial x} \end{bmatrix} \begin{Bmatrix} W \\ b_x \\ b_y \end{Bmatrix} \qquad (5.6)$$

$$\{e_c\} = \{g\} = \begin{Bmatrix} b_x + \dfrac{\partial w}{\partial x} \\ b_y + \dfrac{\partial w}{\partial y} \end{Bmatrix} = \begin{bmatrix} \dfrac{\partial N^T}{\partial x} & N^T & 0 \\ \dfrac{\partial N^T}{\partial y} & 0 & N^T \end{bmatrix} \begin{Bmatrix} W \\ \hat{b}_x \\ \hat{b}_y \end{Bmatrix} \qquad (5.7)$$

Soit :

$$\{c\} = \{\bar{e}_f\} = [\bar{b}_f]\{q\} \qquad \{g\} = [b_g]\{q\} \qquad (5.8)$$

Où :

$$[\bar{b}_f] = \begin{bmatrix} 0 & \dfrac{\partial N^T}{\partial x} & 0 \\ 0 & 0 & \dfrac{\partial N^T}{\partial y} \\ 0 & \dfrac{\partial N^T}{\partial y} & \dfrac{\partial N^T}{\partial x} \end{bmatrix} \qquad [b_g] = \begin{bmatrix} \dfrac{\partial N^T}{\partial x} & N^T & 0 \\ \dfrac{\partial N^T}{\partial y} & 0 & N^T \end{bmatrix} \qquad (5.9)$$

5.3 Matrice de rigidité

L'expression de l'énergie de déformation [2], [17], [23] permet de calculer la matrice de rigidité, soit :

$$U = U_F + U_C$$

$$U = \frac{1}{2}\int_{V^e} \{e_f\}^T \{s_f\} dV + \frac{1}{2}\int_{V^e} \{g\}^T \{s_c\} dV = \frac{1}{2}\{q\}^T [K]\{q\} \qquad (5.10)$$

$$U = \frac{1}{2}\int_{S^e} \{c\}^T [D_f]\{c\} dxdy + \frac{1}{2}\int_{S^e} \{g\}^T [D_c]\{g\} dx\,dy \qquad (5.11)$$

Après substitution des expressions de déformations (5.8) dans l'énergie de déformation, nous obtenons :

$$U = U_F + U_C = \frac{1}{2}\{q\}^T \int_{S^e} [\bar{b}_f]^T [D_f][\bar{b}_f] dxdy \{q\} + \frac{1}{2}\{q\}^T \int_{S^e} [b_g]^T [D_c][b_g] dx\,dy \{q\} \qquad (5.12)$$

D'où :

$$[K] = [K_f] + [K_c] = \int_{S^e} [\bar{b}_f]^T [D_f][\bar{b}_f] dS + \int_{S^e} [b_g]^T [D_c][b_g] dS \qquad (5.13)$$

$$[K] = \int_{-1}^{+1}\int_{-1}^{+1} [\bar{b}_f]^T [D_f][\bar{b}_f] \det[J] dx\, dh + \int_{-1}^{+1}\int_{-1}^{+1} [b_g]^T [D_c][b_g] \det[J] dx\, dh \qquad (5.14)$$

$[J]$: est la matrice Jacobienne de la transformation géométrique.

La matrice Jacobienne $[J(x,h)]$ [2],[10],[23] est :

$$[J]=\begin{bmatrix}\frac{\partial x}{\partial x} & \frac{\partial y}{\partial x}\\ \frac{\partial x}{\partial h} & \frac{\partial y}{\partial h}\end{bmatrix}=\begin{bmatrix}\frac{\partial N^T}{\partial x}X & \frac{\partial N^T}{\partial x}Y\\ \frac{\partial N^T}{\partial h}X & \frac{\partial N^T}{\partial h}Y\end{bmatrix}=\begin{bmatrix}J_{11} & J_{12}\\ J_{21} & J_{22}\end{bmatrix} \tag{5.15}$$

$$\left\{\frac{\partial}{\partial x}\right\}=[j]\left\{\frac{\partial}{\partial x}\right\} \qquad [j]=[J]^{-1} \tag{5.16}$$

$$[j]=\begin{bmatrix}j_{11} & j_{12}\\ j_{21} & j_{22}\end{bmatrix}=\frac{1}{\det[J]}\begin{bmatrix}J_{22} & -J_{12}\\ -J_{21} & J_{11}\end{bmatrix} \tag{5.17}$$

Les déformations $\{\chi\}$ et $\{\gamma\}$ sont définies en fonction des variables nodales :

$$\{c\}=\left[\bar{b}_f\right]\{q\} \qquad ; \qquad \{g\}=\left[b_g\right]\{q\}$$

avec :

$$\left[\bar{b}_f\right]=\begin{bmatrix}0 & \frac{\partial N^T}{\partial x} & 0\\ 0 & 0 & \frac{\partial N^T}{\partial y}\\ 0 & \frac{\partial N^T}{\partial y} & \frac{\partial N^T}{\partial x}\end{bmatrix}; \qquad \left[b_g\right]=\begin{bmatrix}\frac{\partial N^T}{\partial x} & N^T & 0\\ \frac{\partial N^T}{\partial y} & 0 & N^T\end{bmatrix} \tag{5.18}$$

$$\frac{\partial N^T}{\partial x}=j_{11}\frac{\partial N^T}{\partial x}+j_{12}\frac{\partial N^T}{\partial h} \quad ; \qquad \frac{\partial N^T}{\partial y}=j_{21}\frac{\partial N^T}{\partial x}+j_{22}\frac{\partial N^T}{\partial h} \tag{5.19}$$

La matrice de rigidité $[K]$ est obtenue par intégration numérique de (5.14) de type Gauss [2],[10],[23].

L'intégrale peut être évaluée en utilisant la formule :

$$\int_{-1}^{+1}\int_{-1}^{+1} f(x,h)\,dx\,dh=\sum_{i=1}^{r_1}\sum_{j=1}^{r_2} w_i\,w_j\,f(x_i,h_j) \tag{5.20}$$

Où :

x_i, h_j : sont les coordonnées des points d'intégration.

w_i , w_j : sont les coefficients de pondération (ou poids) correspondants.

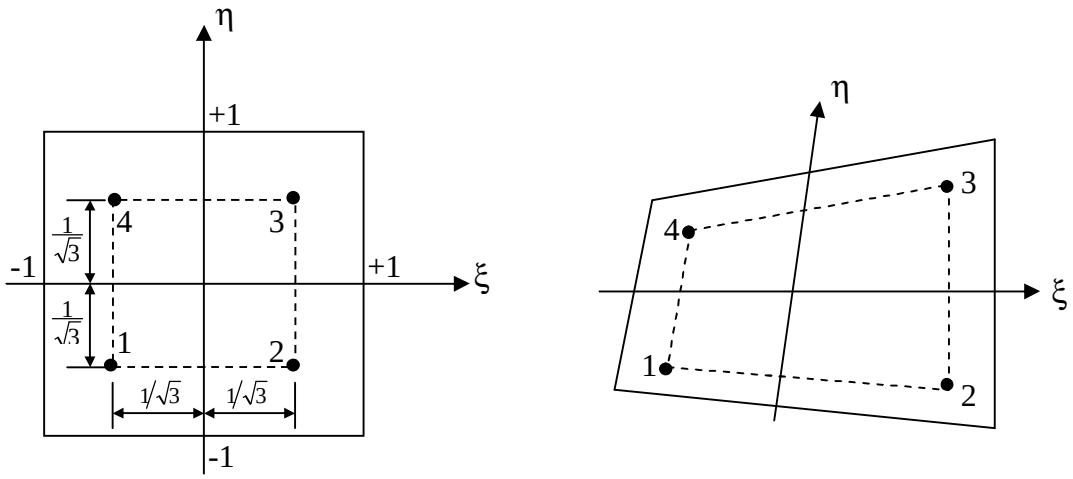


Figure 5.2 Intégration de Gauss (2×2) pour le quadrilatère.

5.4 Vecteur charge équivalent

L'énergie potentielle des forces extérieures [17],[23] exprimée à l'aide des forces de surface et de volume, s'écrit :

$$W = \int_{V^e} \{u\}^{eT} \{f_v\} dV + \int_{S^e} \{u\}^{eT} \{f_s\} dS = \{q\}^{eT} \{F\}^e \qquad (5.21)$$

Nous avons : $$\{u\}^e = [N]^e \{q\}^e \qquad (5.22)$$

$$W = \langle q \rangle^e \int_{V^e} [N]^{eT} \{f_V\} dV + \langle q \rangle^e \int_{S^e} [N]^{eT} \{f_S\} dS = \langle q \rangle^e \{F\}^e \qquad (5.23)$$

D'où : $$\{F\}^e = \int_{V^e} [N]^{eT} \{f_V\} dV + \int_{S^e} [N]^{eT} \{f_S\} dS \qquad (5.24)$$

Pour une charge uniforme répartie f_z suivant z, le vecteur des charges équivalentes associées aux variables W, a la forme habituelle :

$$\{F\}^e = \int_{S^e} [N]^{eT} f_z \, dS \qquad (5.25)$$

5.5 Formulation des équations de mouvement en dynamique linéaire [23]

A partir du principe de Hamilton nous formulons l'équation différentielle du second ordre, caractérisant le mouvement de la structure au cours du temps, ensuite nous procédons à la construction des matrices de masse. Enfin, nous exposons la méthode de calcul utilisée (méthode d'intégration directe de Newmark) pour la résolution du système d'équations différentielles du second ordre.

5.6 Formulation des équations de mouvement [23]

Les équations de Lagrange permettent d'obtenir les équations du mouvement du système discret à partir des expressions des énergies cinétiques, potentielle et de dissipation.

Soit le Lagrangien : $L = T - V$

On a respectivement pour l'énergie cinétique T et l'énergie potentielle totale V :

$$T = \frac{1}{2}\int_V r\, \dot{u}_i \dot{u}_i\, dV \tag{5.26}$$

$$V = U - W = \frac{1}{2}\int_V s_{ij} e_{ij} dV - \int_V f_i^V u_i\, dV - \int_S f_i^S u_i\, dS \tag{5.27}$$

U : Énergie de déformation.

W : Potentiel des forces conservatives de surface et de volume.

f_i^V : Forces de volume.

f_i^S : Forces surfaciques appliquées à la surface extérieure du corps.

L'approximation nodale pour le déplacement $\{u(t)\}$ d'un point quelconque d'un élément a pour expression : $\{u(x,y,z,t)\}^e = [N(x,y,z)]^e \{q(t)\}^e$

On a de même pour la composante de vitesse : $\{\dot{u}(x,y,z,t)\}^e = [N(x,y,z)]^e \{\dot{q}(t)\}^e$

On peut ainsi exprimer le Lagrangien à l'aide des déplacements aux nœuds q_i et de leurs dérivées :

$$L = T(\dot{q}_i) - V(q_i) \tag{5.28}$$

Pour une structure sans amortissement nous avons les équations d'Euler Lagrange [23], [27], suivante :

$$\frac{\partial}{\partial t}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = 0 \tag{5.29}$$

ou encore :

$$\frac{\partial}{\partial t}\left(\frac{\partial T}{\partial \dot{q}_i}\right) - \frac{\partial T}{\partial q_i} + \frac{\partial U}{\partial q_i} = F_i(t) \tag{5.30}$$

avec :

$F_i(t)$: force définie par le travail virtuel des forces extérieures : $dW = F_i\, dq_i$

Pour les petits mouvements des systèmes élastiques, les énergies cinétiques et de déformation s'expriment comme suit :

$$T = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\dot{q}_i\, M_{ij}\,\dot{q}_j = \frac{1}{2}\dot{q}^T M\,\dot{q} \tag{5.31}$$

$$U = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} q_i\, K_{ij}\, q_j = \frac{1}{2}\, q^T K\, q \tag{5.32}$$

Les équations de Lagrange deviennent alors :

$$[M]\{\ddot{q}\} + [K]\{q\} = \{F(t)\} \tag{5.33}$$

5.7 Matrice de masses élémentaires :

L'expression de l'énergie cinétique [8], [23],[26] permet de calculer la matrice masse :

$$T = \frac{1}{2}\int_{V^e} r\,\{\dot{U}\}^{eT}\{\dot{U}\}^e dV = \frac{1}{2}\{\dot{q}\}^{eT}[M]^e\{\dot{q}\}^e \tag{5.34}$$

On a : $$\{\dot{U}\}^e = [N]\{\dot{q}\}^e \tag{5.35}$$

D'où : $$T = \frac{1}{2}\{\dot{q}\}^{eT}\int_{V^e} r\,[N]^T[N]dV\{\dot{q}\}^e \tag{5.36}$$

avec : $$[M]^e = \int_{V^e} r\,[N]^T[N]dV \tag{5.37}$$

$[M]^e$: Matrice de masse cohérente de l'élément e.

5.8 Intégration numérique :

La formulation en éléments finis des matrices de rigidité et de masse, ainsi que le vecteur des sollicitations, nous ramène à des expressions intégrales bidimensionnelles. Leur intégration explicite n'est facile que pour les éléments les plus simples. Il est donc préférable d'utiliser une méthode d'intégration numérique.

5.8.1 Méthode de Gauss :

L'intégration numérique à deux dimensions consiste à utiliser dans chaque direction x et h une intégration numérique à une dimension. Si nous utilisons r_1 points dans le sens x et r_2 points dans le sens h, la méthode de Gauss intègre exactement le produit d'un polynôme en x d'ordre $2r_1$-1 et d'un polynôme en h d'ordre $2r_2$-1. Dans cette méthode on ne se donne pas a priori la position des points d'intégration mais on détermine cette position de façon à minimiser l'erreur

L'intégrale peut être évaluée en utilisant la formule :

$$\int_{-1}^{+1}\int_{-1}^{+1} f(x,h)\,dx\,dh = \sum_{i=1}^{r_1}\sum_{j=1}^{r_2} W_i W_j f(x_i,h_j) \qquad (5.38)$$

Où :

x_i, h_j : sont les coordonnées des points d'intégration.

W_i, W_j : sont les coefficients de pondérations (ou poids) correspondants.

5.8.2 Intégration réduite, intégration sélective :

Les éléments de plaque avec cisaillement transversal sont limités à la modélisation des plaques relativement épaisses, en effet, leur précision se dégrade très rapidement lorsque le paramètre d'allongement de la plaque l (l = L/h, longueur caractéristique sur épaisseur) devient grand. Ainsi de tels éléments sont incapables de simuler correctement le comportement de plaque de type Kirchoff.

Pour améliorer cette situation on utilise la technique d'intégration numérique sélective ; en effet, la matrice de rigidité peut être considérée comme la somme d'une matrice de type flexion et d'une matrice de type cisaillement, le choix d'une intégration réduite à un point pour les termes de cisaillement, alors que les termes de flexion sont intégrés avec (2×2) points de Gauss, améliore sensiblement cet élément.

5.9 Méthodes de résolution des systèmes du second ordre [23]:

Deux approches fondamentales sont envisageables pour la résolution d'un système d'équations différentielles du second ordre, l'une d'entre elles consiste à résoudre ce système par intégration directe, l'autre méthode consiste à définir la solution dans la base des modes propres de vibration de la structure, qui est appelée méthode de superposition modale. Le choix entre ces deux stratégies dépend de la nature du problème (linéaire ou non linéaire) et du contenu fréquentiel de l'excitation.

5.9.1 La méthode de superposition modale :

Convient aux structures linéaires dont les premiers modes propres sont susceptibles d'être excités.

5.9.2 La méthode de résolution directe :

Cette méthode est utilisée pour les problèmes non-linéaires ou si le contenu fréquentiel de l'excitation est susceptible d'exciter un grand nombre de modes de la structure.

L'analyse directe d'une structure en régime transitoire implique l'intégration pas à pas des équations du mouvement :

$$[M]\{\ddot{q}\} + [K]\{q\} = \{F(t)\} \qquad (5.39)$$

Dans cette méthode, les vecteurs de déplacements, vitesses et accélérations au temps

t = 0, sont connus, la période T, sur laquelle la réponse est recherchée est subdivisée en n intervals de temps égale Δt $(\Delta t = \frac{T}{n})$, et le schéma d'intégration employé établi une approximation de la solution au temps : Δt , $2\Delta t$, ..., t, $t+\Delta t$, ..., T.

Il existe plusieurs méthodes de résolution directe, mais on étudiera uniquement la méthodes de Newmark qui est la plus utilisée en dynamique des structures, et que nous avons utilisé.

5.9.3 Méthode de Newmark :

La méthode de Newmark [1],[10],[29]est une méthode implicite, qui permet de construire la solution à l'instant $t+\Delta t$ à partir des vecteurs connus $\{U_t\}, \{\dot{U}_t\}, \{\ddot{U}_t\}$. Elle utilise les hypothèses suivantes :

$$\{\dot{U}_{t+\Delta t}\} = \{\dot{U}_t\} + [(1-a)\{\ddot{U}_t\} + a\{\ddot{U}_{t+\Delta t}\}]\Delta t \qquad (5.40)$$

$$\{U_{t+\Delta t}\} = \{U_t\} + \{\dot{U}_t\}\Delta t + \left[\left(\frac{1}{2} - b\right)\{\ddot{U}_t\} + b\{\ddot{U}_{t+\Delta t}\}\right]\Delta t^2 \qquad (5.41)$$

Où : a et b sont des paramètres permettant d'obtenir l'exactitude et la stabilité du schéma d'intégration.

En fait, le paramètre b contrôle la variation de l'accélération pendant l'incrément de temps Δt .

A l'origine Newmark propose comme un schéma inconditionnellement stable, la règle trapézoïdale, dans laquelle $a = \frac{1}{2}$ et $b = \frac{1}{4}$ (voir fig 5.3).

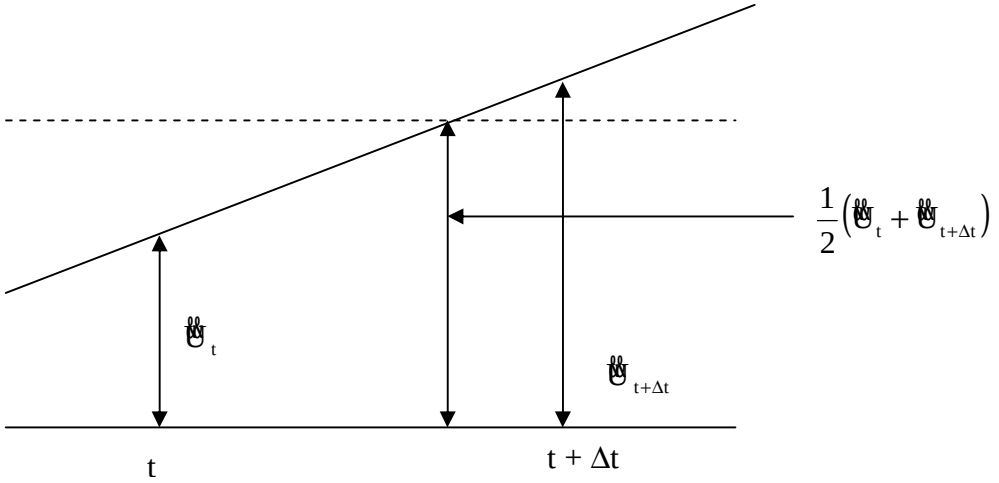


Figure 5.3 Schéma de Newmark.

En plus des équations (5.40) et (5.41), pour la solution des déplacements, vitesses, et accélérations aux temps $t+\Delta t$, les équations de l'équilibre dynamique au temps $t+\Delta t$, sont aussi considérées :

$$[M]\{\ddot{U}_{t+\Delta t}\} + [K]\{U_{t+\Delta t}\} = \{F_{t+\Delta t}\} \qquad (5.42)$$

En utilisant l'équation (5.40) et (5.41), on peut tirer des équations pour $\{\ddot{U}_{t+\Delta t}\}$ et $\{\dot{U}_{t+\Delta t}\}$, en fonction des déplacements inconnus $\{U_{t+\Delta t}\}$; après substitution de ces deux relations dans l'équation du mouvement au temps $(t+\Delta t)$ On obtient :

$$[\overline{K}]\{U_{t+\Delta t}\} = \{\overline{F}_{t+\Delta t}\} \qquad (5.43)$$

Où : $[\overline{K}] = [K] + a_0[M]$

$$\{\overline{F}_{t+\Delta t}\} = \{F_{t+\Delta t}\} + [M]\left[a_0\{U_t\} + a_1\{\dot{U}_t\} + a_2\{\ddot{U}_t\}\right]$$

Les constantes d'intégrations sont données par :

$$a_0 = \frac{1}{b\,\Delta t^2} \qquad a_1 = \frac{1}{b\,\Delta t} \qquad a_2 = \frac{1}{2b} - 1$$

L'équation (5.43) est en fait dans une forme statique. On vient de transformer la résolution d'un problème dynamique on une résolution d'un problème statique.

CONCLUSION GENERALE

Une partie de ce travail met en évidence certaines rigidités et insuffisances des logiciels de modélisation tant au niveau de leurs structures de données que de leur architecture. Ces limitations constituent un handicap croissant à mesure que l'on complexifie les codes, et deviendront bientôt pour la recherche des freins très pénalisants.

Nous avons mis en avant quelques orientations qui nous paraissent indispensables à l'écriture des logiciels modernes, la programmation orientée objet (POO) nous semble apporter des voies prometteuses. Il est reconnu qu'elle améliore grandement la fiabilité des programmes, facilitant le développement et la maintenance.

C++ est actuellement la meilleure solution pour faire de la programmation orientée objet. Notre choix était uniquement guidé par des soucis de fiabilité, de lisibilité, et de modularité ainsi que le besoin d'héritage et de liaison dynamique.

Nous proposons une implémentation de la MEF plus innovante que celle des codes dont nous avons parlé. Nous avons profité du passage aux objets pour concrétiser beaucoup d'idées nouvelles sur des entités qui interviennent dans la MEF. Cela nous a permis de mieux les formaliser et d'obtenir des objets très proches de leur spécification mécanique ou mathématique d'origine. Il nous est apparu indispensable, au cours de notre développement, que tout soit traité en objet. Les approches procédurales et orientée objet sont trop différentes pour pouvoir cohabiter harmonieusement dans un même programme.

Le fait de donner un statut d'objet aux fonctions de base permet, d'une part de leur délégué le travail d'évaluation des valeurs et des dérivées , et d'autre part de pouvoir les manipuler très simplement comme des objets mathématiques.

Les degrés de liberté forment des entités informatiques complètement autonomes. Ils possèdent une valeur, un type de champ d'inconnues, « savent » s'ils sont bloqués ou non et connaissent les parties d'éléments finis sur lesquels ils s'appliquent.

On a pu alors définir les types « vecteur réel indicé par des ddls » et « matrice symétrique réelle indicés par des ddls ». Les matrices de rigidité de masse et le second membre élémentaire des éléments finis seront de ces types.

Un « point de Gauss » est vu comme un objet uniquement capable de donner son poids et sa position dans le repère de référence, c'est avec des objets de cette classe que seront effectués les intégrations numériques (une extension de cette classe suivant la topologie du domaine est possible).

La définition soignée de ces objets de bas niveau est fondamentale pour l'implémentation objet de la MEF. Ils sont mieux formalisés que dans les codes classiques, permettant une meilleure abstraction et offrent d'intéressantes possibilités.

L'approche orientée objet offre d'indéniable avantages par rapport à la programmation classique structurée. L'encapsulation des données améliorent grandement la modularité, et par là également la fiabilité et la lisibilité. L'héritage permet une réutilisabilité automatique des méthodes déjà développées, et le polymorphisme est un puissant moyen pour lever le niveau d'abstraction. Nous profitons de l'approche objet pour proposer des structures plus originales permettant de traiter naturellement un certain nombre de problèmes des implémentations classiques de la MEF.

La programmation orienté objet offre de réelles possibilités d'évolution à la méthode des éléments finis. Par une meilleure modularité, une grande souplesse d'évolution grâce à la dérivation de classes et une abstraction élevée due au polymorphisme, la POO peut être un outil efficace pour maîtriser les codes de demain. La liaison dynamique et le choix des classes au dernier moment peuvent grandement faciliter l'adaptabilité et pilotage dynamique par un système expert.

Les codes numériques d'analyse des structures ont toujours été développés par des ingénieurs et des mécaniciens ; il nous semble que les informaticiens ont aujourd'hui leur mot à dire dans ce domaine. La programmation devient une « science » à maîtriser à part entière.

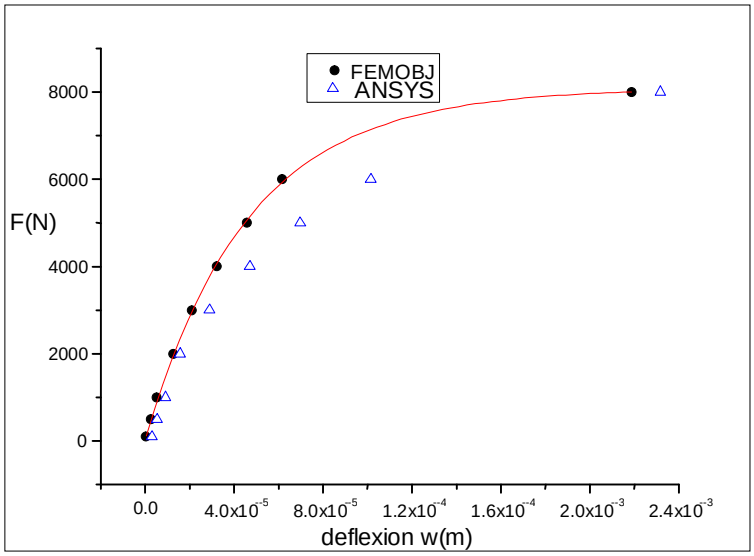


Fig 7.17 Réponse d'une Plaque carrée isotrope encastrée sur ses quatre cotés

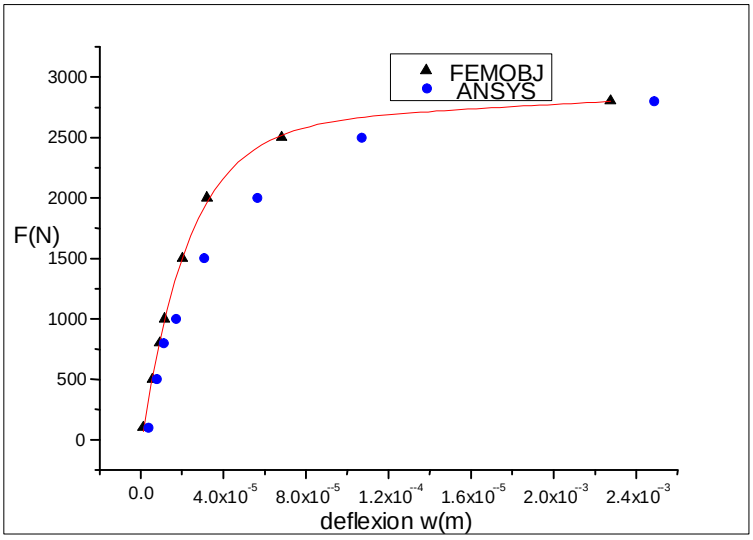


Figure 7.19 Réponse d'une Plaque carrée isotrope simplement appuyée sur ses quatre cotés

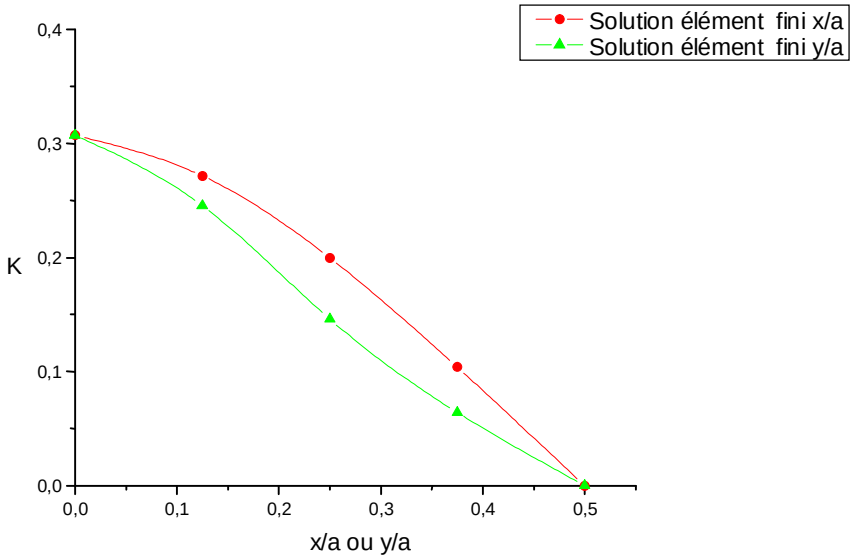


Figure 7.6 : La déflexion le long des deux demi-axes de symétrie d'une plaque orthotrope simplement appuyée avec une charge concentrée au milieu

LISTE DES FIGURES

CHAPITRE I

CHAPITRE II

CHAPITRE III

CHAPITRE IV

CHAPITRE V

CHAPITRE VI

CHAPITRE VIII

Symboles

$\{\ \}$	vecteur colonne
$<\ >$	vecteur ligne
$[\mathbf{K}]$	matrice (utilisé aussi pour les références)
$[\mathbf{K}]^T$	matrice transposée de la matrice $[\mathbf{K}]$
$[\mathbf{K}]^{-1}$	matrice inverse de la matrice $[\mathbf{K}]$
CT	cisaillement transversal
d.d.l.	degrés de liberté
P.O.O	programmation orientée objet
$f,x = \frac{\partial f}{\partial x}$	dérivée partielle de f par rapport à x
δ	symbole de calcul des variations
$\sum$	sommation
$\int$	intégrale
$a \rightarrow \infty$	α tend vers l'infini
x, y, z	coordonnées cartésiennes

Notations

s_{ij}	tenseur des contraintes
e_{ij}	tenseur des déformations
C_{ijkl}	tenseur des constantes élastiques
S_{ijkl}	tenseur des souplesses
E_i	module d'Young dans la direction i=1, 2, 3 avec : $i \neq j$
n_{ij}	coefficient de poisson i, j = 1,2,3 avec : $i \neq j$
G_{ij}	module de rigidité au cisaillement dans le plan (i,j) i,j = 1,2,3 avec : $i \neq j$
$[C]$	matrice des rigidités
$[S]$	matrice des souplesses

$\{s\}$	vecteur des contraintes
$\{e\}$	vecteur des déformations
$\{s_f\}$	vecteur des contraintes de flexion
$\{s_c\}$	vecteur des contraintes de CT
$\{e_f\}$	vecteur des déformations de flexion
$\{e_c\}, \{g\}$	vecteur des déformations de cisaillement transversal
u , v	déplacements suivant x,y
w	déplacement transversal suivant z
q_x , q_y	rotations autour des axes x et y
b_x , b_y	rotations de la normale dans les plans (x-z), (y-z) respectivement
h	épaisseur de la plaque
L	longueur d'une plaque
L/h	facteur d'élancement de la plaque
k	coefficient de correction de cisaillement transversal
$\{c\}$	vecteur des variations des courbures
$[C_f], [C_c]$	matrices de comportement élastique (flexion et CT)
$[D_f], [D_c]$	matrices de rigidités de : flexion, CT
M_x, M_y, M_{xy}	moments de flexion
Q_x, Q_y	efforts de cisaillement transversal parallèle à l'axe des z, des sections perpendiculaire aux axes x et y
V	volume du corps
S	surface du corps où les forces surfaciques sont appliquées
dS , dV	éléments d'aire et de volume
f_i^v	forces volumiques
f_i^s	forces surfaciques
Q_i	forces concentrées
V	énergie potentielle totale
U	énergie de déformation
T	énergie cinétique

L	le lagrangien
W	travail des forces appliquées
$\{u\}^e$	vecteur des déplacements en un point M de l'élément e
$[N]^e$	matrice des fonctions d'interpolation élémentaire
$\{q\}^e$	vecteur des déplacements nodaux de l'élément e
$\{q\}$	vecteur des déplacement nodaux
$\{\ddot{q}\}$	vecteur des accélérations nodales
$[D]$	matrice d'opérateurs différentielles
$[b]$	matrice d'interpolation des déformations
$[b_f], [b_g]$	matrices d'interpolation des déformations de flexion et cisaillement
n_e	nombre de degrés de liberté de l'élément (ou nombre de nœuds par élément)
N	nombre de degré de liberté de la structure
$[K]^e$	matrice de rigidité élémentaire
$[M]^e$	matrice de masse cohérente élémentaire
$[J]$	matrice Jacobienne de la transformation géométrique
$[j]$	matrice inverse de la matrice $[J]$
$[K]$	matrice de rigidité globale
$[K_f], [K_c]$	matrice de rigidité de flexion et de CT
$[M]$	matrice de masse globale
$\{F(t)\}$	vecteur global des sollicitations extérieures
a , b	paramètres du schéma d'intégration de Newmark
t	temps
Δt	incrément de temps
$a_0,\ldots,a_5$	constantes d'intégration
a, L	côté de la plaque
ν	coefficient de poisson
r	masse volumique
D	coefficient de rigidité de flexion
P	intensité d'une charge distribuée uniformément
W_{max}	déplacement maximal

Q4	élément quadrilatère à 4 nœuds
Γ_1	conditions aux limites de Newmann
Γ_2	conditions aux limites de Dirichlet
J_2	second invariant
S	tenseur de contrainte déviatorique
p	tenseur de contrainte hydrostatique
d	tenseur unitaire.
R	rayon du cylindre
$\boldsymbol{de}^{\boldsymbol{e}}$	incrément de déformation élastique
$\boldsymbol{de}^{\boldsymbol{p}}$	incrément de déformation plastique
ds	incrément de contrainte
$r(s)$	vecteur d'écoulement plastique
f	fonction d'écoulement
$\boldsymbol{d}g$	muliplicateur plastique
$\boldsymbol{L(U)}$	équation différentielle

REFERENCES

[1] Bathe K.J., « Finite element procedures », Prentice-Hall, Englewood Cliffs, New Jersey, 1996.

[2] Batoz J.L., Dhatt G., «Modélisation des structures par éléments finis », Vol. 2, Hermès, Paris, Octobre 1990.

[3] Baugh .JW. and Rehak.D.R. Computation abstractions for finite element programming, technical report R-89-182, departement of civil Engineering,Carnegie Mellon University, Pittsburgh, USA

[4] Claude Delannoy Programmer en langage c++ . 3éme Edition Eyrolles Paris 2002.

[5] Clough R.W., Penzien J., «Dynamique des structures », Tome 1, Editions Pluralis, 1980.

[6] Commend stéphane, Thomas Zimmermann, Object-Oriented Nonlinear Finite Element Programming Advances In Engineering Software 2001 32, 8, 611-628.

[7] Courtade R.M., Lemaire M., Cubaud J.C., « Déplacements, déformations et contraintes dans les matériaux élastiques anisotropes », Verre textile – plastiques renforcés, PP. 20-27, N° 4, Avril 1973.

[8] Craveur J.-C., « Modélisation des structures Calcul par éléments finis », Masson, Paris, 1996.

[9] Desjardins.R and Fafard.M, développement d'un logiciel pour l'analyse des structures par élementfinis utilisant l'approche de la programmation objets, Rapport GCT-92-05, Univrsité Laval, Sainte-Foy, Canada. 1992.

[10] Dhatt G., Touzot G., «Une présentation de la méthode des éléments finis », 2ème Edition, Editeur Maloine S.A., paris, 1984.

[11] Dubois-pèlerin Yves, Thomas Zimmermann, Patricia Bomme, Object-oriented finite element programming II. A prototype programming in smaltalk . Computer Methods in Applied Mechanics and Engineering 1992, 98, 361-397.

[12] Dubois-pèlerin Yves ,Thomas Zimmermann, Object-oriented finite element programming III. An efficient implementation in c++. Computer Methods in Applied Mechanics and Engineering 1993, 108, 165-183.

[13] Felippa.C.A, Database management in scientific computing-II. Data structure and programming architecture. Omput.&Structure 12 (1980) 131-145

[14] FEM-Object, www.zace.com, freeware.

[15] Fenves.G.L, Object-oriented programming for Engineering software development. Engrg.With Comput. 6 (1990) 1-15.

[16] Forde.B.W.R, Foschi.R.O and Stiemer.S.F, Object-oriented finite element analysis, Comput.& Structure 34 (1990) 355-374

[17] Frey. F., « Analyse des structures et milieux continus Mécanique des solides », Vol. 3, 1ére édition, Lausanne, 1998.

[18] Gallagher R.H., « Introduction aux éléments finis », Editions Pluralis, 1976.

[19] Geoffroy P., « Développement et évaluation d'un élément fini pour l'analyse non linéaire statique et dynamique de coques minces », Thèse de Doct.-Ing., Université de Technologie de Compiègne, Avril 1983.

[20] Han W.-S., «Analyse linéaire et non-linéaire de plaques et coques par éléments finis en statique et dynamique sur micro-ordinateur », Thèse de Doctorat, Institut National polytechnique de Lorraine, Juin 1989.

[21] Hawken.D.M, P.Townsend and Webster.M.F, The use of dynamic data strutures in finite element application, internat. J. numer. Methods.Engrg. 33 (1992) 1795-1812

[22] Hughes T.J.R., R.M. Ferencz and A.M.Raefsky, DLEARN- Alineae static dynamic finite element analysis program, in : T.J.R. Hughes , ed. The Finite Element Method (Prentice Hall, Englewood Cliffs. NJ, 1987)

[23] Imbert J.F., « Analyse des structures par éléments finis », 3ème Edition, Editions Cépaduès, Toulouse, Janvier 1991.

[24] Jones R. M., « Mechanics of composite materials », McGraw-Hill, Etats-Unis, 1975.

[25] Kris Jamsa. Lars Klander La bible du programmeur c/c++, Editions Reynauld Goulet, 1999.

[26] Lalanne M., Berthier P., Der Hagopian J., «Mécanique des vibrations linéaires », 2éme édition, Masson, Paris, 1986.

[27] Laroze S., « Résistance des matériaux et structures » , Tome 3, Masson, Paris, 1979.

[28] Miller.G.R, An object –oriented approach to structural analysis and design, comput.& Structures 40 (1991) 75-82.

[29] Newmark N.M., « A method of computation for structural dynamics », ASCE, Journal of Engineering Mechanics Division, Vol. 85, N°. EM3, PP. 67-94, 1959.

[30] Pahl.P.J, Data management in finite element analysis, in:Wunderlich, stein and bathe,eds.,Nonlineair Finite Element Analysis in strutural Mechanics (Springer, berlin, 1980) 715-741

[31] Prat M., Bisch P. et Mestat P., « La modélisation des ouvrages », Hermès, Paris, 1995.

[32] Reddy J.N., « An introduction to the finite element method », Ed. Slaughter T.M., Hazlett S., McGraw-Hill, Etats-Unis, 1984.

[33] Shi G. Bezine G., « A general boundary integral formulation for the anisotropic plate bending problems », Journal of composite materials, Vol. 22, PP. 694-716, Aôut 1988.

[34] Timoshenko S., Woinowsky-Krieger S., « Théorie des plaques et coques », Librairie Polytechnique CH. Béranger.

[35] Techniques de l'ingénieur « Comportement élastique linéaire-Loi de Hooke », A7 750, PP. 5-18.

[36] Tzong-Shuoh Yang, Yuang-Bin Chang, Array management system, Comput & Structure 33 (1989) 1507-1527.

[37] Wilson.E.L and Hoit.M.I, A computer adaptive langage for the development of structural analysis program, Comput.& Structure 19 (1984) 321-338

[38] Zimmermann, Thomas Yves Dubois-pèlerin, Patricia Bomme, Object-oriented finite element programming I. Governing principles, Computer Methods in Applied Mechanics and Engineering 1992, 98, 291-303.

RESUME

L'analyse numérique est en train de vivre une triple révolution. En premier lieu, son champ d'action est de plus en plus vaste et complexe. Fortran, outil (logiciel) traditionnel des numériciens conçu il y a plus de 30 ans, a du mal à suivre cette évolution. La seconde mutation qui frappe l'analyse numérique s'appelle « adaptativité automatique » ; nous sommes loin des données statiques entrées par l'utilisateur, la programmation structurée ne nous semble pas un outil adéquat à cette nouvelle forme d'analyse numérique.

Enfin, l'accroissement considérable de la mémoire centrale des ordinateurs permet aujourd'hui d'envisager des structures de données plus riches.

Il a été ainsi accepté jusqu'à récemment que les codes éléments finis ne pouvaient réaliser que des analyses sur des données statiques et bien définis au départ par l'utilisateur. Les résultats du calcul devaient également suivre un format déterminé. Ainsi a-t-on continué de développer les codes de modélisation avec le langage Fortran et le style procédural, bien adaptés à ce monde « de serveur numérique ». A l'opposé, en conception assistée par ordinateur, les données sont changeantes en valeur, mais aussi en forme. Elles sont modifiées par le logiciel ou, de manière itérative, par l'utilisateur. Le développement de tels logiciels de CAO se fait en plus en utilisant des langages modernes et des méthodologies logicielles fondées sur l'approche objet.

Le C++ est actuellement la meilleure solution pour faire de la programmation orientée objet. Nous proposons une implémentation de la MEF plus innovante que celle des codes dont nous avons parlés.

La programmation orientée objet (POO) nous semble être des voies prometteuses. Il est reconnu qu'elle améliore grandement la fiabilité des programmes, facilite le développement et la maintenance.

Le but de cette thèse est de proposer des « définitions objets » des entités intervenant dans la méthode des éléments finis, plus précisément dans l'implémentation de l'élément plaque isotrope et orthotrope dans le cas dynamique, statique linéaire et statique non-linéaire avec l'approche orientée objet en utilisant le langage C++.