الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'enseignement supérieur et de la recherche scientifique

Université Mohamed Khider – Biskra
Faculté des Sciences et de la technologie
Département : Génie civil et hydraulique

جامعة محمد خيضر بسكـــــرة
كلية العلوم و التكنولوجيـــــــا
قسم: الهندسة المدنية والـــري
Ref :.....
المرجع : .....

Thèse présentée en vue de l'obtention du diplôme de

Doctorat en Science

Spécialité : Génie Civil

Option : Structure

Approche Orientée Objet de la Méthode des Eléments Finis et des Eléments Finis Etendus en utilisant le Langage de programmation C++

Présentée par

Moussaoui Sabah

Soutenue publiquement le : **30/09/2021**

Devant le jury composé de

M^r BENMEBAREK Sadok	Professeur	Président	Université de Biskra
M^r BElOUNAR Lamine	Professeur	Examinateur	Université de Biskra
M^me BAHLOUL Ouassila	MCA	Examinatrice	Université de Batna 2
M^r BELGASMIA Mourad	Professeur	Rapporteur	Université de Sétif 1

REMERCIMENTS

***E**n premier lieu, je tiens à remercier notre seigneur Allah de m'avoir aidé à accomplir ce travail de recherche.*

***Je** remercie également mes très chers parents pour leurs encouragements, soutiens et surtout leur patience durant toutes mes études.*

***Je** voudrais témoigner toute ma gratitude et reconnaissance aux personnes qui m'ont aidé à réaliser cette thèse.*

***Q**ue mon directeur de thèse Monsieur le Professeur Mourad Belgasmia trouve ici l'expression de toute ma gratitude pour son aide inestimable, sa compréhension et ses encouragement, je suis très reconnaissante à son égard pour l'orientation et le savoir si important que j'ai trouvé en sa noble personne.*

***M**es remerciements les plus ardents aux membres du jury, Professeur Benmebarek Sadok , Professeur Belounar Lamine et Docteur Bahloul Ouassila, d'avoir accepté d'évaluer mon travail et de l'intérêt qu'ils ont porté à cette thèse .*

***J**e tiens à remercier également l'université de Biskra d'avoir pris en charge mon dossier d'inscription en thèse.*

Résumé

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 ».

La manière traditionnelle des programmeurs de tenter d'améliorer la maintenance du code s'effectue par le biais de l'amélioration de la modularité du code. Exemple en structurant un code comme étant un assemblage des modules les plus indépendants possible ; le but est impérativement le bon mais l'approche est fausse. L'approche courante (celles des langages tels que Fortran, Pascal, et C) est par conséquent la programmation procédurale.

Dans la programmation procédurale le programme est une séquence d'appel aux procédures (subroutines) qui sont munies de données adéquates (leurs arguments). La modularité de ces procédures est sévèrement limitée par la forte faiblesse de la structure de donnée et la séquentialité des opérations.

La véritable modularité exige que quelques arguments doivent être transmis, et le module doit être capable d'obtenir par soit même le reste des données dont il a besoin. Ceci est assuré pleinement par la programmation orientée objet.

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.

Le but de ce travail, est double, d'abord l'implémentation d'un élément plan de coque dans un code élément finis orienté objet en utilisant le langage C++, tout en détaillant les limites de la méthode des éléments finis dans le cas des structures discontinues, pour aborder la méthode des éléments finis étendus. Puis l'utilisation d'un logiciel écrit à base de l'approche objet, à fin de comparer la méthode statique non-linéaire (push-over) à la méthode dynamique non-linéaire (time history) sur un bâtiment réel en 3D. Et ce en utilisant l'élément de Spacone ; (élément de flexibilité) qui est basé sur les forces ; après l'avoir vérifié en le comparant à la méthode classique basée sur les déplacements.
Mots clés : Approche Orientée Objet, Langage C++, Eléments finis, Eléments finis étendus, Méthode statique non-linéaire.

Abstract

Numerical analysis is going through a triple revolution. First, its field of action is increasingly vast and complex. Fortran, a traditional numerical tool (software) designed over 30 years ago, is struggling to keep up with this development. The second mutation that strikes numerical analysis is called "automatic adaptivity".

The traditional way for programmers to attempt to improve code maintenance is through improving the modularity of the code. Example by structuring a code as being an assembly of the most independent modules as possible; the goal is absolutely the right one, but the approach is wrong. The current approach (that of languages such as FORTRAN, Pascal, and C) is therefore procedural programming.

In procedural programming, the program is a sequence of calls to procedures (subroutines) which are provided with adequate data (their arguments). The modularity of these procedures is severely limited by the strong weakness of the data structure and the sequentiality of operations.

True modularity requires that some argument must be passed, and the module must be able to obtain by itself the rest of the data it needs. This is fully ensured by object-oriented programming.

Part of this work highlights certain shortcomings of modeling software in terms of their both data structures and architecture. These limitations are a growing handicap as the codes become more complex, and will soon become very penalizing brakes for research.

The goal of this work is twofold, firstly the implementation of a shell element in an object oriented finite element code using the C ++ language, while detailing the limits of the finite element method in the case of discontinuous structure, to present extended finite element method. Then the use of software written based (written) on the object-oriented approach, to compare the non-linear static method (Push-over) with the dynamic non-linear method (time history) on a real building in 3D. By using Spacone element (flexibility element) which is based on the forces after having checked, it by comparing it to the classical method based on displacements.

Keywords: Object oriented approach, Finite element method, C++ language, Extended finite element method, Static nonlinear method.

ملخص

يمر التحليل الرقمي بثورة ثلاثية. أولاً، مجال عملها واسع ومعقد بشكل متزايد. تكافح. Fortran أداة رقمية (برمجية) تقليدية تم تصميمها منذ أكثر من 30 عامًا، لمواكبة هذا التطور. الطفرة الثانية التي تصيب التحليل الرقمي تسمى "التكيف التلقائي"الطريقة التقليدية للمبرمجين لمحاولة تحسين صيانة البرنامج هي من خلال تحسين نمطية البرنامج.

مثال من خلال هيكلة رمز باعتباره تجميعًا للوحدات النمطية الأكثر استقلالية الممكنة؛ الهدف هو الصحيح تمامًا، لكن النهج خاطئ. النهج الحالي (نهج لغات مثل Fortran وPascal وC) هو بالتالي البرمجة الإجرائية.

في البرمجة الإجرائية، البرنامج عبارة عن سلسلة من المكالمات إلى الإجراءات التي يتم تزويدها بالبيانات الكافية. إن نمطية هذه الإجراءات محدودة للغاية بسبب الضعف الشديد لهيكل البيانات وتسلسل العمليات.

تتطلب الوحدة النمطية الحقيقية تمرير بعض الحجة، ويجب أن تكون الوحدة قادرة على الحصول بنفسها على بقية البيانات التي تحتاجها. يتم ضمان ذلك بشكل كامل عن طريق البرمجة الشيئية.

يسلط جزء من هذا العمل الضوء على بعض أوجه القصور في برمجيات النمذجة من حيث هياكل البيانات وبنيتها. تشكل هذه القيود عائقًا متزايدًا حيث تصبح البرامج أكثر تعقيدًا، وستصبح قريبًا بمثابة عقبة كبيرة أمام البحث.

الهدف من هذا العمل هو ذو شقين، أولاً تنفيذ عنصر shell في برنامج العناصر المحدودة الموجهة للكائنات باستخدام لغة C++، مع توضيح حدود طريقة العناصر المحدودة في حالة هيكل متقطع لمقاربة طريقة العناصر الممتدة. ثم استخدام البرنامج المكتوب على أساس نهج الكائن، لمقارنة الطريقة الثابتة غير الخطية (الدفع) بالطريقة الديناميكية غير الخطية (التاريخ الزمني) على مبنى حقيقي ثلاثي الأبعاد. وذلك باستخدام عنصر Spacone (عنصر المرونة) الذي يعتمد على القوى بعد التحقق منه عن طريق مقارنته بالطريقة الكلاسيكية القائمة على الإزاحة.

الكلمات المفتاحية: منهج موجه للكائن، لغة C++، عنصر محدود، عنصر محدود ممتد، الطريقة الثابتة غير الخطية.

TABLE DES MATIERES

Chapitre 2

Efficacité de la programmation en C++

Chapitre 3

Rôle des classes et leurs appels au sein du code éléments finis orienté objet

Chapitre 4

Théorie des plaques et plaques membranes

Chapitre 5

Formulation Eléments Finis Des Plaques et Plaques Membranes

Chapitre 6

Application de la méthode statique non-linéaire en utilisant un logiciel basé sur l'approche orientée objet

Chapitre 7

Méthode des éléments finis étendus (XFEM)

Chapitre 8

Validation des résultats

LISTE DES FIGURES

CHAPITRE 2

CHAPITRE 3

CHAPITRE 4

CHAPITRE 5

CHAPITRE 6

CHAPITRE 7

CHAPITRE 8

LISTE DES TABLEAUX

CHAPITRE 2

CHAPITRE 3

CHAPITRE 5

CHAPITRE 6

CHAPITRE 8

Symboles

$\{\ \}$	vecteur colonne
< >	vecteur ligne
$[\ldots]$	matrice (utilisé aussi pour les références)
$[\ldots]^T$	matrice transposée de la matrice $[\ldots]$
d.d.l.	degrés de liberté
$\sum$	Sommation
δ	symbole de calcul des variations
x, y, z	coordonnées cartésiennes

Notations

h	épaisseur de la plaque
L	longueur d'une plaque
L/h	facteur d'élancement de la plaque
θ_x , θ_y	rotations autour des axes x et y
β_x , β_y	rotations de la normale dans les plans (x-z), (y-z) respectivement
u , v	déplacements suivant x,y
w	déplacement transversal suivant z
$\{\varepsilon\}$	vecteur des déformations
$\{\varepsilon_c\}$, $\{\gamma\}$	vecteur des déformations de cisaillement transversal
$\{\varepsilon_m\}$, $\{\varepsilon_f\}$	vecteur de déformations de membrane et de flexion
$\{\chi\}$	vecteur des variations des courbures
$[C]$	la matrice constitutive
$[C_c]$, $[C_m]$	matrice constitutive de cisaillement transversal et de membrane.
$\{\sigma\}$	vecteur des contraintes
$\{\sigma_f\}$, $\{\sigma_m\}$	vecteur des contraintes de flexion et de flexion
$\{\sigma_c\}$	vecteur des contraintes de CT
$[C]$	la matrice constitutive

$[C_c]$, $[C_m]$	matrice constitutive de cisaillement transversal et de membrane.
E	module d'Young
υ	coefficient de poisson
k	coefficient de correction de cisaillement transversal
$[D_m], [D_f], [D_c]$	matrices des caractéristiques élastiques de membrane, flexion et cisaillement transversal
N_x , N_y , N_{xy}	efforts résultants de membrane
M_x, M_y, M_{xy}	moments de flexion
T_x, T_y	efforts résultants de cisaillement ou effort tranchants.
$\{E\}$	vecteur des efforts résultants
v	volume du corps
S	surface du corps où les forces surfaciques sont appliquées
dS, dv	éléments d'aire et de volume
σ_{ij}	tenseur des contraintes ;
ε_{ij}	tenseur des déformations infinitésimales.
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	potentiel des forces conservatives de surface et de volume,
$\{u\}^e$	vecteur des déplacements en un point M de l'élément e
$\{\dot{u}\}^e$	vecteur des vitesses
$[N]^e$	matrice des fonctions d'interpolation élémentaire
$\{q\}^e$	vecteur des déplacements nodaux de l'élément e
$\{q\}$	vecteur des déplacement nodaux
$\{\dot{q}\}$	vecteur des vitesses nodales

$\{\ddot{q}\}$	vecteur des accélérations nodales
$[D]$	matrice d'opérateurs différentielles
$[B]$	matrice reliant les déformations aux variables nodales
$[B_f]$, $[B_\gamma]$	matrices reliant les courbures ou déformations de CT aux variables nodales
$[B_m]$	matrice reliant les déformations de membrane aux variables nodales de l'élément
$[\mathrm{B}]^e$	matrice de localisation de l'élément
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
$\{F\}^e$	vecteur des forces équivalentes élémentaires.
$[J]$	matrice Jacobienne de la transformation géométrique
$[j]$	matrice inverse de la matrice $[J]$
$[K_f]$, $[K_c]$	matrice de rigidité de flexion et de CT
$[K_m]$	matrice de rigidité de membrane
$[M]$	matrice de masse globale
$[K]$	matrice de rigidité globale
$[C]$	matrice d'amortissement
$\{F(t)\}$	vecteur global des sollicitations extérieures
t	temps
Δt	incrément de temps
γ, β	paramètres de Newmark
ρ	masse volumique
ϕ	le mode
a	l'accélération du sol en fonction du temps
R	vecteur d'influence
m^*	masse équivalente
Γ	facteur de participation modale
V_b	effort tranchant à la base
DDL	degré de liberté

SDOF	single degree of freedom (système à un seul degré de liberté)
MDOF	multi degree of freedom (système à plusieurs degrés de liberté)
POO	programmation orientée objet
K_I, K_{II}, K_{III}	facteur d'intensité de contrainte en mode I ,II, III respectivement
N	ensemble des nœuds du maillage
N_{er}	l'ensemble des nœuds enrichis par la discontinuité
N_{tip}	l'ensemble des nœuds enrichis par branch function
u_I	degrés de libertés classiques pour le nœud i
a_J	degrés de libertés supplémentaires pour la fonction de discontinuité
$b_{\alpha K}$	degrés de libertés supplémentaires pour branch function
$H(x)$	heaviside function
$B_\alpha(x)$	branch function
$N_I(x), \tilde{N}_J(x), \tilde{N}_K(x)$	fonctions de forme standard d'élément fini

Introduction Générale

Le langage de programmation a son importance. Le Fortran est le premier langage de programmation scientifique procédural à apparaître ; l'existence de nombreuses bibliothèques mathématiques et des compilateurs très efficaces font qu'aujourd'hui encore, le Fortran est utilisé dans bon nombre de modules d'analyse des programmes d'éléments finis.

Les langages orientés-objet, apparus plus tard, au cours des années 1980 (d'abord Smalltalk, puis Java, Python, etc.) fournissent une alternative élégante aux langages procéduraux pour les développeurs de programmes d'éléments finis. Les informations (variables ou méthodes) sont stockées (encapsulées) au niveau des différents objets (élément, noeud, ...) organisés sous forme de classes formant une hiérarchie.

Le fait d'avoir recours à des objets correspondant à des entités réelles rend la programmation plus aisée, et plus sûre. La réutilisation de parties de code en est également facilitée.

Enfin, le langage C et son dérivé orienté-objet C++ sont utilisés dans de nombreux logiciels, en particulier lorsqu'il s'agit de programmer leurs ressources graphiques (pré- ou post-processing).

Il est clair pendant plusieurs années que l'amélioration de la modularité du code en élément 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.

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 étudié ressemant. Des chercheurs révèlent 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.

Le but de ce travail, est double, d'abord l'implémentation d'un élément plaque membrane appelé aussi élément plan de coque dans un code élément finis orienté objet en utilisant le langage C++, tout en détaillant les limites de la méthode des éléments finis pour

aborder la méthode des éléments finis étendus. Puis l'utilisation d'un logiciel écrit à base de l'approche objet, à fin de comparer la méthode statique non-linéaire (push-over) à la méthode dynamique non-linéaire (time history) sur un bâtiment réel en 3D. Et ce en utilisant l'élément de Spacone (élément de flexibilité) qui est basé sur les forces après l'avoir vérifier en le comparant à la méthode classique basée sur les déplacements.

Pour ce faire, cette thèse se présente en huit chapitres comme suit :

Le premier chapitre est consacré à l'historique de la programmation depuis l'avènement de l'ordinateur, jusqu'aux caractéristiques de l'approche orientée objet.

Le chapitre deux, est consacré à l'explication et à la comparaison entre l'approche orientée objet et la programmation procédurale, une comparaison en terme d'allocation de mémoire préprocesseur et post processeur.

Le chapitre trois consiste en la présentation des classes les plus imposantes et leurs sous classes avec leurs objets et méthodes.

Les détails de la théorie des plaques et des plaques membranes ont fait l'objet du chapitre quatre.

Quant aux détails de la formulation éléments finis des plaques et élément plan de coque ont été esquissés au chapitre cinq.

Le chapitre six regroupe toutes les informations concernant l'utilisation d'un logiciel conçu à base de l'approche orienté objet, afin de faire une comparaison entre l'étude statique non-linéaire et la dynamique non-linéaire d'un bâtiment existant et ce en utilisant l'élément de flexibilité.

Le chapitre sept porte sur l'implémentation en C++ avec une approche orientée objet de la méthode XFEM.

Le chapitre huit est consacré à la validation des résultats.

Enfin, une conclusion générale, qui nous permet d'avoir une vue d'ensemble sur les résultats obtenus à travers cette étude.

Chapitre 1

Notion de la programmation orientée objet

1.1. Introduction

La programmation orienté objet en éléments finis a reçu récemment une attention ascendante, dans des articles récents les chercheurs ont développé une méthodologie d'implémenter les éléments finis à l'intérieur du contexte d'orienté objet en utilisant différents langages successivement : Smaltalk [25], Ctalk et finalement C++.

La manière traditionnelle de programmer pour tenter d'améliorer la maintenance des codes par le biais de l'amélioration de la modularité, en concevant le code comme un assemblage de modules aussi indépendant que possible ; l'objectif est exactement le bon mais l'approche est mauvaise. Cette approche (utilisée dans pas mal de langages de programmation Fortran, Pascal ou C) est la programmation procédurale où le programme est une séquence d'appel de procédures (subroutine) qui sont munies de donnée adéquates (leurs arguments).

La modularité de ces procédures est sévèrement limitée par deux grandes déficiences à savoir : La faiblesse des structures de données et la séquentialité des opérations.

Une structure de donnée qui est faible veut dire que les données (variables) portent peu d'information et ont peu de capacité. Typiquement en Fortran un tableau de variable ne porte pas leurs propre taille et sa seule capacité est d'accéder à ces coefficients, mais il est incapable d'effectuer une vérification hors limites comme conséquence pratique il y'aura trop d'interface des sous-programme avec le reste du programme ; l'interface de la subroutine consiste principalement dans sa liste d'arguments, c'est souvent très large, le sous-programme doit être alimenté avec beaucoup de données sur lesquelles il agit [34]. Puisque toutes ces données doivent être fournies par l'extérieur, le module ne peut pas exercer de contrôle sur eux, une vrais modularité exige très peu d'arguments (idéalement aucun) à transmettre et que le module soit capable par soit même d'obtenir le reste des données dont il a besoin.

La séquentialité veut dire que le sous-programme dépend fortement de l'ordre des opérations dont il est introduit. Par exemple une subroutine qui calcul la longueur d'un élément barre souvent possède comme donnée les cordonnées nodales dont elle a besoin. Cependant cette routine peut être appelée seulement après celle qui lui délivre les coordonnées nodales.

Une telle forte supposition dans la conception du sous-programme oblige le programmeur d'avoir une large connaissance du reste du programme, dans ce cas peut-on appelé la subroutine un module ? Une vraie modularité exige que la subroutine soit indépendante des opérations passées.

La section suivante prouve que cette déficience peut être surmontée en se déplaçant vers la programmation orientée objet.

Les caractéristiques clés de l'approche orienté objet sont ; l'encapsulation des données, communication par message, organisation de la hiérarchie du code, héritage et polymorphisme tous ces ingrédients font que le code devient d'une modularité élevée et une réutilisabilité et maintenance améliorée chose qui va mener à un prototypage rapide et un débogage facile.

1.2. Types de langages et leurs évolutions

Depuis l'avènement des ordinateurs, au cours des années 50, leur évolution n'a cessé d'accélérer. On peut en dire autant du coté logiciel.

1.2.1. Le langage machine (spécifique à une architecture donnée)

- excessivement compliqué pour les humains,
- n'a pas été utilisé très longtemps,
- très petits programmes (<1000 instructions).

1.2.2. Le langage d'assemblage (spécifique à une famille d'ordinateurs donnée)

- problèmes relativement simples et/ou courts,
- a évolué énormément au cours des ans,
- il était surtout utilisé pour les programmes de système.

1.2.3. Les langages de haut niveau (FORTRAN, COBOL, ...)

- problèmes longs mais peu complexes,
- pas de structure (à l'époque) des instructions ni des variables,
- étaient spécialisés à des domaines spécifiques.

1.2.4. La programmation structurée (ALGOL, Pascal, C, Modula, ...)

- problèmes assez complexes et/ou longs,
- bonnes structure des énoncés et des variables,
- assez rigides et lourds dans leur utilisation.

1.2.5. La programmation orientée objet (C++, Pascal OO, Simula, ...)

- problèmes très complexes,
- traitement simplifié d'objets physiques ou non,
- augmentation de la productivité des utilisateurs.

1.2.6. L'ingénierie du logiciel

- problèmes très gros et très complexes,
- participation de grosses équipes de travail,
- plusieurs outils de conception de logiciels.

1.3. Historique du C++ [20], [54]

La programmation Orientée Objet (en abrégé P.O.O.) est dorénavant universellement reconnue pour les avantages qu'elle procure. Notamment, elle améliore largement la productivité des développeurs, la robustesse, la portabilité et l'extensibilité de leurs programmes. Enfin et surtout, elle permet de développer des composants logiciels entièrement réutilisables.

Un certain nombre de langages dits ″Langages Orientés Objet″ (L.O.O.) ont été définis de toutes pièces pour appliquer les concepts de P.O.O. C'est ainsi que sont apparus dans un premier temps des langages comme Smaltalk, Simula ou Eiffel puis, plus récemment, Java. Le langage C++, quant à lui, a été conçu à partir de 1982 suivant une démarche quelque peu différente par Bjarne Stroustrup (AT & T Bell laboratoires) ; son objectif a été, en effet, d'adjoindre au langage C un certain nombre de spécificités lui permettant d'appliquer les concepts de P.O.O. l'originalité d'être fondé sur un langage répandu. Ceci laisse au programmeur toute liberté d'adopter un style plus ou moins orienté objet, en se situant entre les deux extrêmes que constituent la poursuite d'une programmation classique d'une part, une pure P.O.O. d'autre part. Si une telle liberté présente le risque de céder, dans un premier temps, à la facilité en mélangeant les genres (la P.O.O. ne renie pas la programmation classique- elle l'enrichit), elle permet également une transition en douceur vers la P.O.O. pure [21], avec tout le bénéfice qu'on peut en escompter à terme.

De sa conception jusqu'à sa normalisation, le langage C++ a quelque peu évolué. Initialement, un certain nombre de publications de AT & T ont servi de référence du langage. Les dernières en date sont : la version 2.0 en 1989, les versions 2.1 et 3 en 1991. C'est cette dernière qui a servi de base au travail du comité ANSI lequel, sans la remettre en cause, l'a

enrichie de quelques extensions et surtout de composant standard originaux se présentant sous forme de fonctions et de classes génériques qu'on désigne souvent par le sigle S.T.L. La norme définitive de C++ a été publiée en Juillet 1998.

1.4. Caractéristiques de l'approche Orientée Objet [20], [54]

La programmation orientée objet est une autre façon de structurer un programme et de décomposer un problème.

En programmation procédurale on essaie de séparer totalement les données et les traitements, alors qu'en POO on regroupe dans la même entité (classe) des données et les traitements qui s'y rapportent. Cette façon de faire est plus proche du fonctionnement humain, elle a de plus l'avantage de permettre :

- une meilleure réutilisation du code car la récupération d'une ancienne classe transporte avec elle tous les traitements associés aux données.
- une meilleure évolutivité, car une modification d'une structure de donnée n'affecte que le code de la classe modifiée.

De plus, la technique de l'héritage permet de développer une nouvelle classe en partant d'une classe existante, et en y ajoutant les données et les traitements manquants, ce qui évite de réécrire plusieurs fois le même code ou de "copier-coller" du code.

1.5. Les apports de la programmation orientée objet [20], [54]

1.5.1. Objet

C'est là qu'intervient la programmation Orientée Objet (en abrégé P.O.O.), fondée justement sur le concept d'**objet**, à savoir une association des données et des procédures (qu'on appelle alors méthodes) agissant sur ces données. On pourrait dire que l'équation de la P.O.O. est :

Méthodes + Données = Objet

Cette notion se rapproche de celle d'une structure : c'est un agrégat de différents types représentant une entité réelle.

1.5.1.1. États (attributs)

Les variables qui décrivent (indiquent) les états actuels d'un objet.

1.5.1.2. Méthodes

Procédures et fonctions qui décrivent le comportement d'un objet en modifiant ou traitant les états de façon à refléter l'effet des événements sur cet objet.

1.5.1.3. Message

Signal envoyé à un objet qui déclenche l'exécution d'une de ses méthodes. Dans ce cas on parle d'envoyeur et de récepteur. Un message contient le nom du récepteur, la méthode à appliquer et les paramètres requis par cette méthode.

1.5.2. Classe

En P.O.O. apparait généralement le concept de classe, qui correspond simplement à une généralisation de la notion de type que l'on rencontre dans les langages classiques. En effet, une classe n'est rien d'autre que la description d'un ensemble d'objets ayant une structure de données commune et disposant des mêmes méthodes. Les objets apparaissent alors comme des variables d'un tel type classe (en P.O.O., on dit aussi qu'un objet est une ″instance″ de classe).

1.5.3. Encapsulation [20], [54]

En effet, dans ce que l'on pourrait qualifier de P.O.O. ″pure″, on réalise ce que l'on nomme une **encapsulation des données** (terme technique utilisé pour représenter le masquage d'information). Cela signifie qu'il n'est pas possible d'agir directement sur les données d'un objet ; il est nécessaire de passer par l'intermédiaire de ses méthodes, qui jouent ainsi le rôle d'interface obligatoire. On traduit parfois cela en disant que l'appel d'une méthode est en fait l'envoi d'un ″message″ à l'objet.

Le grand mérite de l'encapsulation est que, vu de l'extérieur, un objet se caractérise uniquement par les spécifications de ses méthodes, la manière dont sont réellement implantées les données étant sans importance. On décrit souvent une telle situation on disant qu'elle réalise une ″abstraction des données″ (ce qui exprime bien que les détails concrets d'implémentation sont cachés).

L'encapsulation des données présente un intérêt manifeste en matière de qualité de logiciel. Elle facilité considérablement la maintenance : une modification éventuelle de la structure des données d'un objet n'a d'incidence que sur l'objet lui-même ; les utilisateurs de l'objet ne seront pas concernés par la teneur de cette modification ce qui n'était bien sûr pas le cas avec la programmation structurée. De la même manière l'encapsulation des données facilite grandement la réutilisation d'un objet.

1.5.4. Héritage [20], [54]

Un autre concept important en P.O.O. est celui d'héritage. Il permet de définir une nouvelle classe à partir d'une classe existante (qu'on réutilise en bloc !), à laquelle on ajoute de nouvelles données et de nouvelles méthodes. La conception de la nouvelle classe, qui ″hérite″ des propriétés et des aptitudes de l'ancienne, peut ainsi s'appuyer sur des réalisations antérieures parfaitement au point et les ″spécialiser″ à volonté. Comme on peut s'en douter, l'héritage facilite largement la réutilisation de produits existant, d'autant plus qu'il peut être réitéré autant de fois que nécessaire.

1.5.4.1. Hiérarchie de classes [20], [54]

L'héritage peut se faire sur plusieurs niveaux (générations). La structure arborescente qui en résulte est appelée hiérarchie de classes. Une classe hérite de tous ses ancêtres dans la hiérarchie.

1.5.5. Polymorphisme [20], [54]

Concept qui permet à différentes classes d'objets de répondre à un même message selon sa propre définition.

Chapitre 2

Efficacité de la programmation en C++

2.1. Introduction

Le programme élément fini en C++, orienté objet (FEMOBJ) est évalué en termes d'efficacité numérique. Ses performances sont comparées à celles du code élément finis procédural en Fortran (Dlearn). Ces derniers sont brièvement présentés dans ce chapitre en décrivant un exemple faisant office d'un test de comparaison entre les deux codes à savoir FEMOBJ [35] écris en C++ et Dlearn [47] écris en Fortran.

2.2. Dlearn programme en éléments finis écris en Fortran

Dlearn [47] est un programme de base pour résoudre les problèmes de l'élasto-dynamique linéaire, comme logiciel didactique son objectif est de procurer aux débutants une première approche de la programmation classique de la méthode des éléments finis. Une attention particulière est accordée à la rédaction et l'extensibilité du code, le tableau 2.1 suivant résume le domaine d'application ainsi que la technique de programmation de Dlearn comparé à ceux du code en C++. Les deux programmes englobent à peu près les mêmes fonctionnalités et constituent donc une excellente base de comparaison.

2.2.1. Exemple de comparaison

Les performances des deux codes sont comparées pour le problème statique linéaire de la déformation plane esquissée dans la figure 2.1. Le domaine est rectangulaire, avec des appuis simples le long des deux bords. Le côté droit est soumis à une charge uniformément répartie dont la résultante est 8000N. Le domaine est discrétisé avec un maillage uniforme quadrilatère à quatre nœuds, (huit degrés de liberté, par élément).

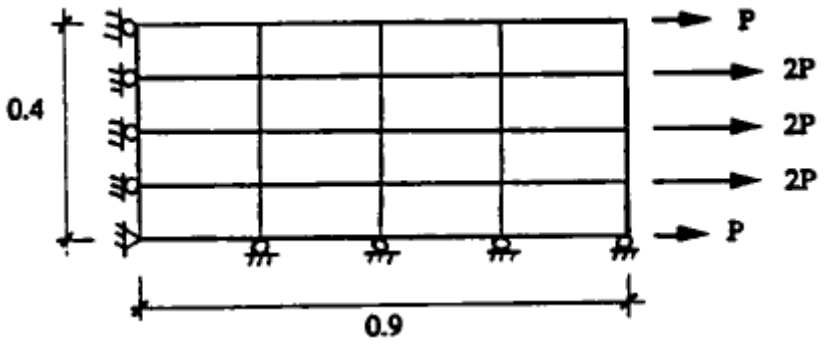


Figure 2.1. Exemple de comparaison [24].

	Code en fortran	Code en C++
Organisaion de la mémoire d'allocation	Pseudo allocation dynamique dans le common blanc	Allocation et désallocation dynamique
Stokage du système lineaire	Skyline symmétrique	Skyline symmétrique
Type d'élément finis	Elément quadrilatère	Elément quadrilatère
Algorithme de résolution	Factorisation U^tDU	Factorisation U^tDU
Préprocesseur Donnée graphique	Non	Non
Post processeur	Les résultats écrits dans un fichier	Les résultats écrits dans un fichier
Occupation du disk Code source Code exécutable (3) Code exécutable (4)	 185 kB 180 kB + common blanc 520 kB + common blanc	 300 kB (1) 100 kB (2) 350 kB (2)

Tableau 2.1. Comparaisons des caractéristiques des deux codes [24].

(1) Commentaires inclus
(2) Espace mémoire occupé par l'allocation dynamique des objets n'est pas inclus
(3) sur un PC
(4) sur Unix

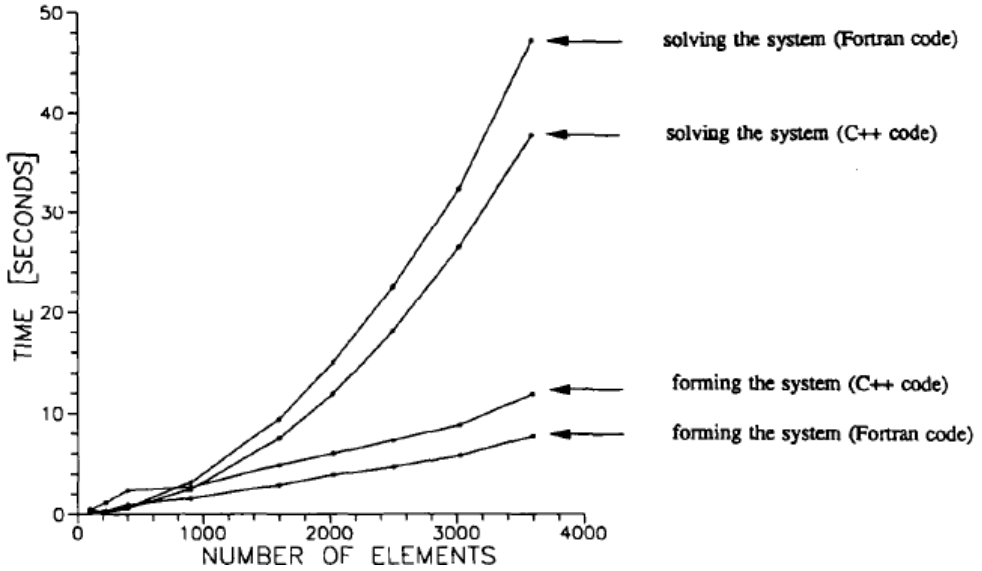


Figure 2.2. Temps consommé pour la formation et résolution du système en fonction du maillage [24].

2.3. Bénéfices de la POO VS la programmation procédurale

Voici quelques statistiques provenant de certains départements de la compagnie IBM.

	Langage C	**Langage C++**
Ligne de codes	5827	1059
Variables globales	245	5
Nombre de ligne moyenne d'une fonction	39	6

Tableau 2.2. Différences entre la POO et la programmation procédurale.

2.4. Efficacité en programmation [24], [26]

2.4.1. Rapidité en programmation

Le code C++ est bien plus adapté qu'un programme Fortran dans le sens de la rapidité d'extensibilité du programme. Les raisons en sont la modularité améliorée induite par l'approche orientée objet et le soin apporté à la sélection des classes d'éléments finis.

2.4.2. Maintenabilité du code

La capacité de programmation rapide bénéficie de tous les composants de la maintenabilité du code : compréhensibilité, extensibilité, débogage facile. Le programme éléments finis C + + est maintenant évalué en respect de ces aspects.

2.4.2.1. Compréhensibilité (lisibilité, homogénéité, auto-description)

En sous-entend par compréhensibilité ; la lisibilité, l'homogénéité et l'auto-description du code, par le fait que le programme en C++ en sa globalité est facilement lisible comme l'illustre la figure 2.3.

Un sacrifice sur la lisibilité pour l'intérêt de l'efficacité de calcul (souvent au bénéfice de la manipulation avec les pointeurs autrement dit programmation bas niveau) est de temps à autre accordé mais ce genre de comportement en programmation affecte seulement une petite portion du code, dans les environs de 10%. Cette partie optimisée (utilisation des pointeurs) du code est concentrée dans un nombre extrêmement réduit de classes telles que FloatArray et FloatMatrix.

Une partie du programme est dite **homogène** si toutes les lignes du code sont d'une même complexité, cette qualité est bien préservée en C++. Methode computestiffnessMatrix de la classe Element de la figure 2.3. témoigne de ça : les ingrédients pour K^e (B, D, DB) sont obtenue un par un puis sommé.

La compréhensibilité, d'un code écrit en C++ est complétée par une auto-description, cette auto-description est atteinte lorsqu'un étranger au programme peut facilement le comprendre sans avoir recours aux documents externes tels que le manuel de programmation. Cela est réalisé dans une large mesure, grâce à un aperçu progressif du programme, en trois étapes à savoir :

1- une vue d'ensemble du logiciel est fournie par la hiérarchie des classes voir figure 3.1. du chapitre 3 ;
2- la description de chaque classe répertoriée dans la hiérarchie est contenue dans un document de programmation appelé : fichier en-tête ;
3- Enfin, les détails de mise en œuvre de chaque méthode déclarés dans le fichier Header sont accessibles dans le code source.

Mais une parfaite auto-description n'est pas atteinte car un étranger ne peut pas comprendre le déroulement de la chose en détail ceci viens du fait que les classes sont conçues comme outils de programmation non pas comme driver et c'est pour ça qu'elles sont flexibles et réutilisables.

```
FloatMatrix*  Element :: computeStiffnessMatrix ( )
  // Computes numerically the stiffness matrix of the receiver.
{
  int              i ;
  double           dV ;
  FloatMatrix   *b,*db,*d ;
  GaussPoint    *gp ;
  stiffnessMatrix = new FloatMatrix( ) ;
  for (i=0 ; i<numberOfGaussPoints ; i++) {
    gp = gaussPointArray[i] ;
    b  = this -> ComputeBmatrixAt(gp) ;
    d  = this -> giveConstitutiveMatrix() ;
    dV = this -> computeVolumeAround(gp) ;
    db = d -> Times(b) ;
    stiffnessMatrix -> plusProduct(b,db,dV) ;
    delete b ;
    delete db ;}
    return stiffnessMatrix -> symmetrized( ) ;
}
```

Figure 2.3. Programme source de la méthode computestiffnessmatrix de la classe Element.

2.4.2.2. Extensibilité

Le manque de l'évolutivité facile de l'approche procédurale est la malédiction des programmeurs de ce type d'approche, en revanche la rapidité de programmation point de vue extensibilité (voir figure 2.2.) démontre à quel point le programme C++ se prête bien à l'introduction de nouvelles fonctionnalités.

```
Element*  Element  :: ofType (char*   aClass)

  {
      Element*   newElement ;
      if (! strncmp(aClass, "SH" ,2))
            newElement = new SHELLS (number , domain) ;
     else if (! strncmp(aClass,"OP",2))
           newElement = new Orplate (number,domain) ;

     else {
            printf ("%s : unknown element type \n",aClass) ;
            exit (0) ;  }

     return    newElement ;
   }
```

Figure 2.4. Creation d'une novelle classe : Shell type element.

2.4.2.3. Facilité de débogage

Comparativement au Fortran, le langage de programmation C++ introduit deux améliorations majeures ; premièrement le programmeur en C++ est amené à faire moins d'erreurs grâce à l'encapsulation de l'objet car il possède et cache ses données ; qui est un moyen de gérer ses opérations d'une manière autonome. Ces opérations peuvent être activés avec sécurité par un simple message, par exemple : element $->$ giveConstitutiveMatrix ( ) ;.

Le programmeur détecte les erreurs rapidement du moment que dans la programmation orientée objet les variables sont regroupées dans une structure compréhensible (objet), leurs exactitude est facile à vérifier les débuggeurs se révèlent être plus utile qu'en Fortran.

2.5. Eléments finis en Oriente Objet

Cette section montre les notions par lesquelles une forte amélioration de la modularité est atteinte pour une programmation en éléments finis qui sont l'encapsulation des données, l'héritage des classes et la non-anticipation.

2.5.1. Une nouvelle modularité dans l'espace

2.5.1.1. Encapsulation des données

L'orientation objet concentre l'intérêt sur les données plutôt que sur les opérations. Les données sont appelées objet. Le but de ce dernier est d'accomplir une séries d'opération attribuée appelées méthode. La propriété fondamentale de l'objet qui est à la fois posséder et cacher ces attributs (données) est connues sous le nom de l'encapsulation des données.

2.5.1.2. Héritage des classes [56], [58], [77]

La classe doit partager ces attributs et méthodes communes. La programmation orientée objet regroupes des objets de même genre appelée classe ; L'implémentation consiste à créer de nouvelles classes, à leur attribuer une série d'attributs et à mettre en œuvre leur série de méthodes, deux classes ou plus présentent souvent des similitudes. Le mécanisme d'héritage permet de définir leurs attributs et méthodes communs une seule fois, dans une superclasse partagée. Prenons comme exemple les deux classes Plate et Shell (qui implémente l'élément quadrilatérale), qui ont un attribut stiffnessMatrix dans le but de former la matrice de rigidité, les deux classes ont besoin exactement de la même méthode FormStiffnessMatrix.

Cependant définir une superclasse appelée classe Element est une bonne pratique surtout en lui assignant l'attribut stiffnessMatrix et la méthode FormStiffnessMatrix. Par héritage, les classes Plate et Shell possèdent automatiquement ces fonctionnalités en évitant la duplication du code, l'héritage favorise fortement la modularité.

2.5.2. Une nouvelle modularité dans le temps

2.5.2.1. Non anticipation

Les deux concepts de base de la programmation orientée objet (l'encapsulation des données et l'héritage des classes) fournit une forte structure de données. Cependant il ne résout pas le problème de séquentialité des opérations, mais le concept de la non anticipation le fait. Ceci est complètement nouveau comme concept de programmation destinée à la programmation en élément finis, il affirme ce qui suit :

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 massMatrix, c'est là où on stocke notre matrice de masse; l'obtention de cette matrice de la classe Element peut se faire par deux approches :

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

Le message : Mtr = element1 −> computeMassMatrix( ) ; ceci dans le cas où l'on sait que la matrice masse n'a pas été formée.

Ou bien le message : Mtr = element1 −> return massMatrix( ); ceci dans le cas où l'on sait que la matrice masse a été formée (la matrice existe).

Le trouble dans cette approche, c'est qu'il est très difficile de connaître quand est-ce qu'une variable ou un objet doit être initialisé ; ceci entraîne souvent le programmeur à une anticipation d'opération très délicate, 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 faire aucune supposition sur l'état de l'attribut massMatrix, ceci dans tous les cas en envoyant juste le message
Mtr = element1 −> giveMassMatrix( ) ;.

La classe Element est munie d'une méthode qui est giveMassMatrix( ); cette dernière est extrêmement simple, elle est utilisée dans toutes les situations.

```
FloatMatrix*    Element : : giveMassMatrix ( )
{
        if  (! massMatrix)
                this −> computeMassMatrix( ) ;
        return   massMatrix ;
}
```

Figure 2.5. Non anticipation.

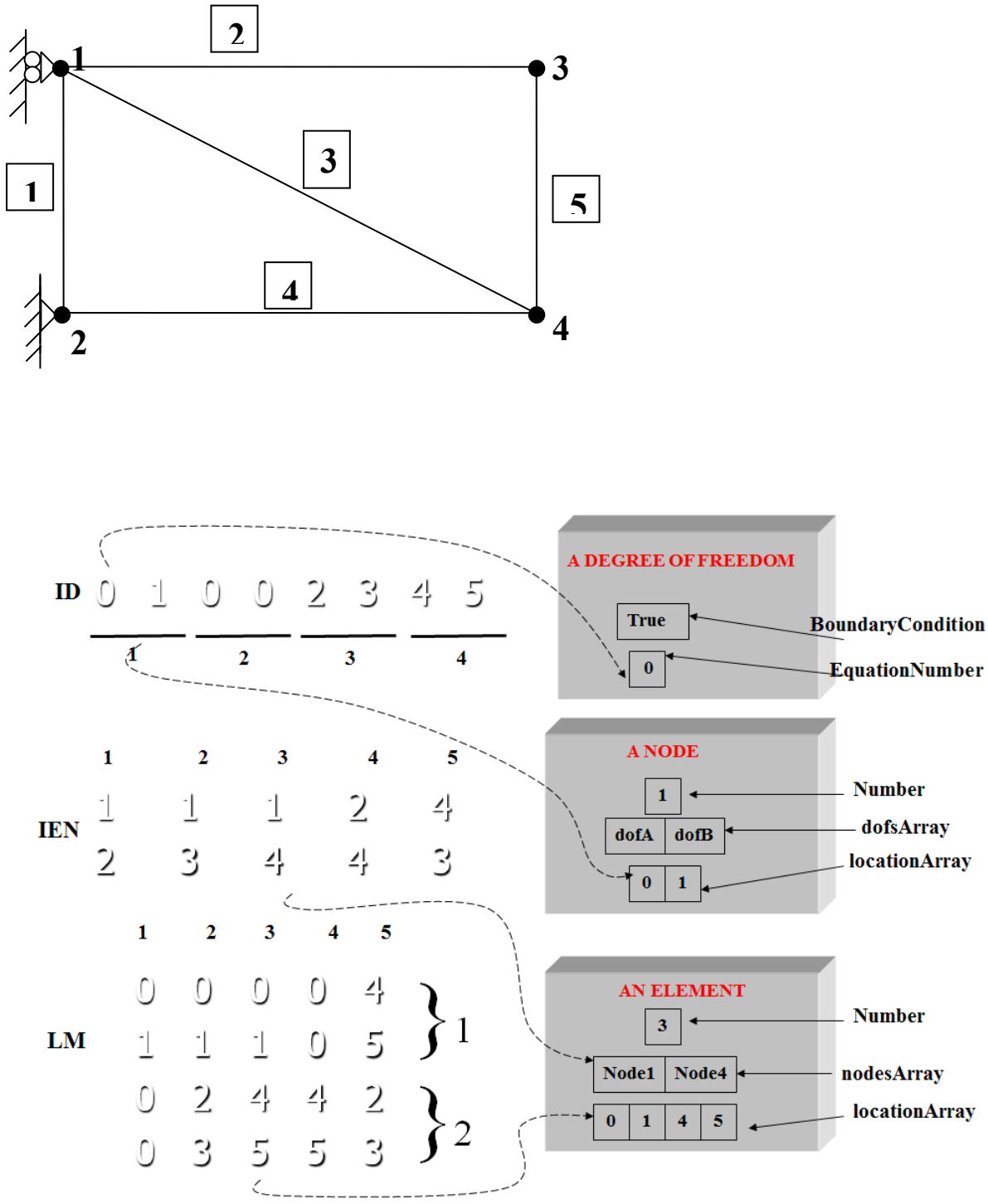
Cette méthode consiste en un test : l'élément vérifie si l'attribut massMatrix est nul (il n'existe pas) ou non (il existe); s'il n'existe pas l'élément exécute la méthode

computeMassMatrix( ), puis renvoie la matrice. Si par contre l'attribut existe, la matrice est renvoyée. Clairement le message ci-dessus « Element1 −> giveMassMatrix( ); » peut être inséré dans n'importe quel endroit dans le programme et l'élément va toujours répondre d'une manière adéquate.

2.6. Modularisation du processus de numérotation d'équations

La localisation dans le système d'équations linéaires est une tâche qui se prête à une comparaison entre l'approche orientée objet et l'approche procédurale (Fortran). Elle montre comment la nature décentralisée des variables aide à produire un code plus modulaire, donc plus compréhensible et plus réutilisable.

L'emplacement englobe toutes les fonctionnalités liées à la numérotation de l'équation des degrés de libertés, y compris la mise en place des tableaux nodaux et élémentaux qui donnent la correspondance entre la numérotation locale et globale, la figure 2.6 affiche une structure de treillis et ses caractéristiques d'emplacement associées. Sur la gauche, une gestion classique est ; À droite, la gestion orientée objet.



Approche procedurale Approche orientee objet

Figure 2.6. Localisation Approche procédural VS approche orientée objet.

2.7. Conclusion

Ce chapitre laisse apparaitre l'efficacité du langage de programmation C++ en le comparant à celui du FORTRAN, par le biais de deux codes qui font la même chose en termes de résolution d'un même problème. Ce qu'il faut retenir de ce chapitre est que le langage C++ est d'attrait rivalisant avec le FORTRAN en termes de calcul.

Chapitre 3

Rôle des classes et leurs appels au sein du code éléments finis orienté objet

3.1. Introduction

Ce chapitre est consacré aux différentes classes fars sur lesquelles repose le code FEMOBJ [35] ; qui est un code en élément finis basé sur l'approche orientée objet en utilisant le langage de programmation C++. Une attention particulière a été donnée aux classes les plus importantes (Element, GaussPoint, Material, Domain, NLSolver et leurs sous classes, dont la dernière classe permet de résoudre le problème non-linéaire en utilisant l'algorithme de retour radial [15], [17], [27], [48], [57].

3.2. Hiérarchie des classes et organigramme du code FEMOBJ

On commence d'abord par donner un schéma qui permet de représenter la hiérarchie des classes ainsi que l'organigramme du code :

Dictionary
Dof
Domain
FemComponent
- Element
  - PlaneStrain
  - Truss2D
  - **Plate**
  - **Shell**
- Load
  - BodyLoad
- DeadWeight
- BoundryCondition
- NodalLoad
- LoadTimeFunction
  - ConstantFunction
  - PeakFunction
- Material
- Node
- TimeIntegrationScheme
  - Newmark
  - Static
- Timestep

FileReader
FloatArray
- Column

GaussPoint
IntArray
LinearSystem
- BandSystem
- SkylineSystem

List
Matrix
- FloatMatrix
  - BandMatrix
- PolynomialMtrix

Polynomial
- PolynomialXY

Skyline
String

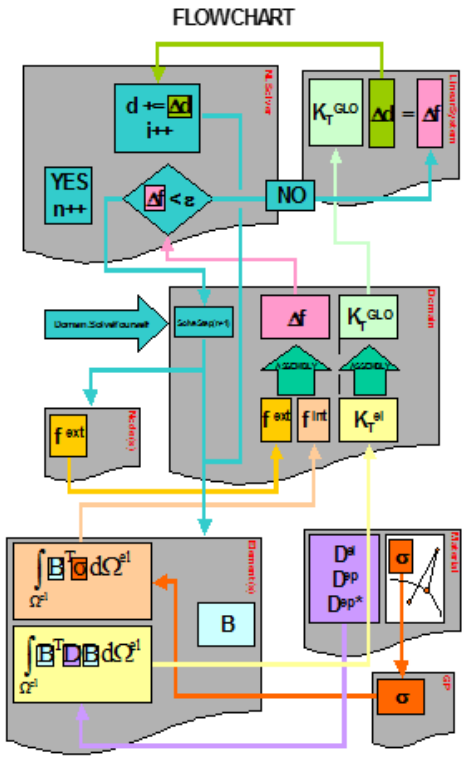


Figure 3.1. Hiérarchie des classes et organigramme du code éléments finis en C++ , [15],[17],[24],[27].

3.3. Classe Domain

C'est la super classe de la classe Element ses taches, ses attributs ainsi que ses méthodes sont détaillés sur le tableau 3.1 ci-dessous :

Classe Domain		
Taches	***Attributs***	***Méthodes***
1) Gérer les composants du problème	*elementList* *nodeList* materailList loadList loadTimeFunctionlist Nlsolver timeIntegrationScheme numberOfElement numberOfNodes numberOfFreeDofs	giveElement(i) giveNode(i) giveMaterial(i) giveLoad(i) giveLoadTimeFunction(i) giveNLsolver(i) givetimeIntegrationScheme( ) givenumberOfElement( ) givenumberOfNodes( ) givenumberOfFreeDofs( )
2) Résoudre le problème	*unknownsArray*	give*unknownsArray* ( ) solveYourself( ) formTheSystemAt(astep) solveYourselfAt(astep)
3) Interaction avec le solver nonlinéaire	-	computeRHSAT(aΔd) computeInternalForce(aΔd) computeLoadvecteorAT(aΔd) computeTangetStiffnessMatrix(aΔd)
4) input/output	DataFileName inputScream outputScream	giveDataFileName( ) giveInputScream( ) giveoutputScream( ) readNumberOf(aComponent)

Tableau 3.1. Rôle de la classe Domain.

3.4. Classe FemComponent

Cette classe, qui est une superclasse des classes Element, Load, LoadTimeFunction, Material, Node, TimeIntegrationScheme and TimeStep, regroupe tous les attributs et les méthodes qui sont communs à toutes ses sous classes.

3.4.1. Classe Element

Regroupe les attributs et les méthodes qui sont communs à chaque élément. Ses taches principales sont :

1- Le calcul de la matrice de masse (dans le cas dynamique), matrice de rigidité et le vecteur de charge.
2- Donner sa contribution aux membres gauches et droits du système (à travers l'opération d'assemblage exécutée par la classe Domain).
3- Lire, stocker et rendre ses données.

3.4.2. Classe Node

Le nœud est un attribut de un ou plusieurs éléments. Il a les taches suivantes :

- Retourner ses coordonnées.
- Arranger (création et stockage) ses dégrée de libertés (class Dof).
- Calculer et assembler ses vecteurs de charge nodale (class NodalLoad).
- Initialise ses attributs à la fin de chaque itération.

Classe Node Hérite de :FemComponent		
Taches	***Attributs***	***Méthodes***
1) création	-	Node(aDomain, aNumber) InstanciateYourself()
2)Positionnement dans l'espace	coordinates	getCoordinates() giveCoordinate(i)
3)Arrangement des degrees de libertes	dofArray NumberOfDofs	giveDof(i) giveNumberOfDofs()
4) Gerer le vecteur de charge nodal **a) Calcul** **b) Assemblage**	loadArray locationArray	computeLoadVectorAT(aStep) giveLoadArray() assembleYourLoadsAT(aStep) giveLocationArray()
5) affichage		

Tableau 3.2. Rôle de la classe Node.

3.4.3. Classe TimeIntegrationScheme

Cette classe (et ses sous classes Newmark et Static) définit la time history du problème. Ses tâches sont :

- Gérer la time history du problème (exemple : Le pas de temps (class TimeStep)).
- Retourner ses coefficients (β et γ).

3.4.4. Classe Load

Cette superclasse implémente plusieurs actions appliquées à l'élément, neoud et DDL.

3.4.4.1. Classe DeadWeight

Cette classe permet l'implémentation de la force de gravite et de volume.

3.4.4.2. Classe NodalLoad

La charge nodale est une charge concentrée qui acte directement sur le neoud. C'est un attribut de un ou plusieurs nœuds. Sa principale tâche est de retourner la valeur de ses composantes au pas de temps donné.

3.4.4.3. Classe BoundaryCondition

Les conditions aux limites est une restriction imposée au DDL. Elle définit la valeur prescrite de l'inconnue et c'est un attribut d'un ou plusieurs DDL (class Dof).

3.5. Classe GaussPoint

Un point de Gauss est un attribut d'un élément. Sa tâche est de regrouper les données spécifiques au point de Gauss : les coordonnées et les valeurs des points en intégration numérique, les déformations et les contraintes. Il doit stocker l'état de contrainte et de déformation à l'itération actuelle.

Class GaussPoint **Inherits from : -**		
Tasks	**Attributes**	**Methods**
1) creation	number element coordinates weight	*GaussPoint(aNumber, anElement, x, y, w)* giveNumber() giveCoordinates() giveCoordinate(i) giveWeight()
2) stresses / strains handling	stressVector previousStressVector strainVector	giveStressVector() givePreviousStressVector() giveStrainVector() letStressVectorBe(aStressState) letPreviousStressVectorBe(aStressState) letStrainVectorBe(aStrainState)
3) stress return computation	deltaGamma plasticCode stressLevel	setDeltaGamma(aDeltaGamma) giveDeltaGamma() isPlastic() givePlasticCode() computeStressLevel() giveStressLevel()
4) output	-	printOutput(aFile) printBinaryResults(aFile)
5) internal handling	-	updateYourself()

Tableau 3.3. Rôle de la classe GuassPoint.

3.6. Classe Shell

La classe Shell est une classe qu'on a implémentée dans le code FEMOBJ [35], cette dernière fait appel à une série de classe comme le montre le schéma ci-dessous, suivi par le schéma d'héritage.

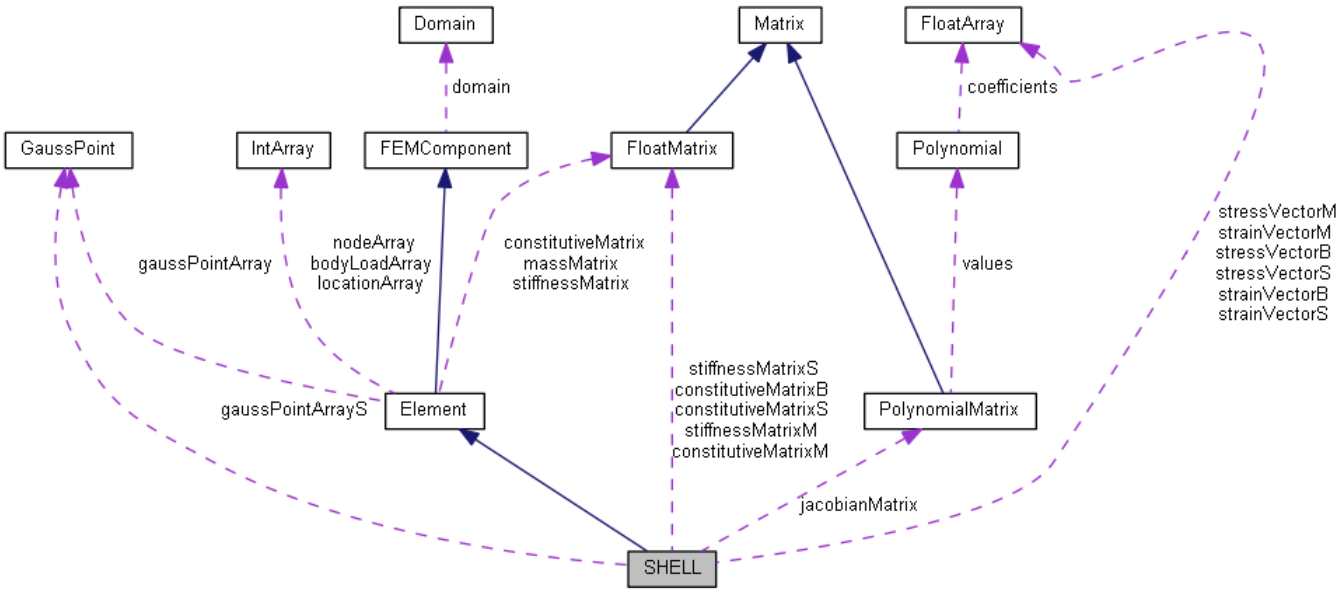


Figure 3.2. Classes appelées par la classe Shell.

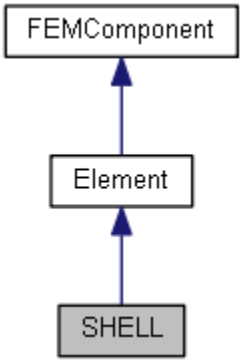


Figure 3.3. Schémas d'héritage de la classe Shell.

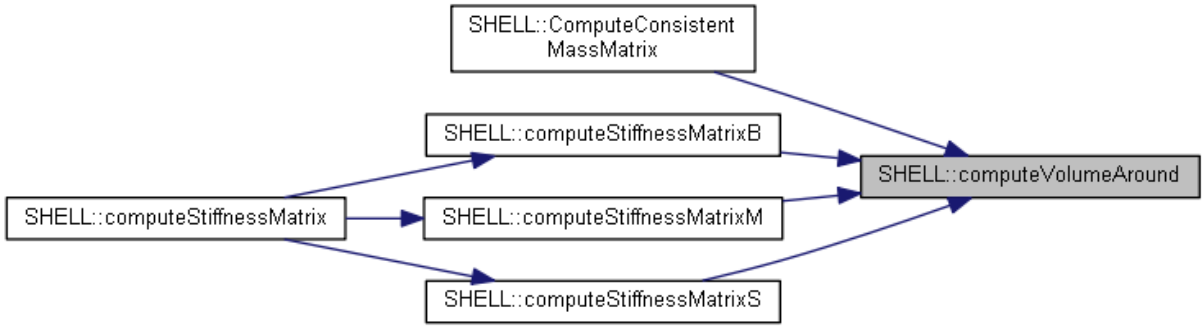


Figure 3.4. Schémas des méthodes déclenchées lors de l'appel de la méthode computeStiffnessMatrix de la classe Shell pour le calcul de la matrice de rigidité.

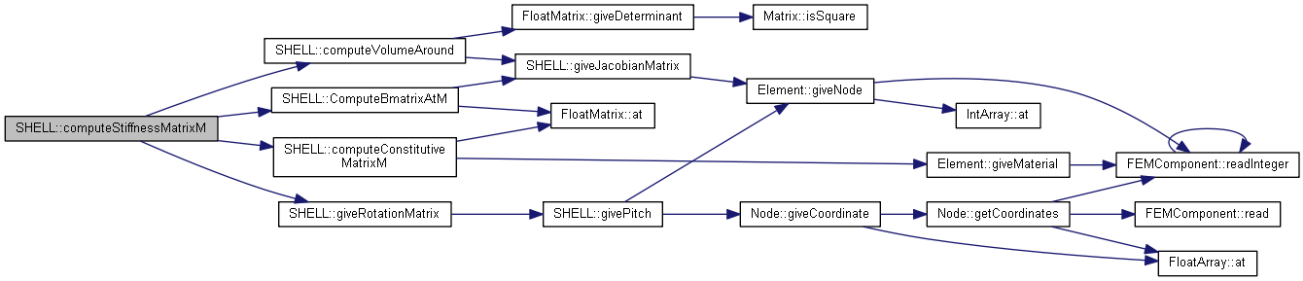


Figure 3.5. Schémas des méthodes déclenchées lors de l'appel de la méthode computeStiffnessMatrixM de la classe Shell pour le calcul de la matrice de rigidité membranaire.

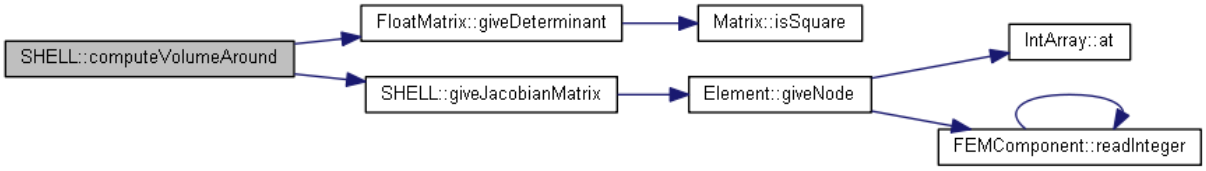


Figure 3.6. Schémas des méthodes déclenchées lors de l'appel de la méthode computeVolumeAround de la classe Shell pour le calcul de l'intégration numérique.

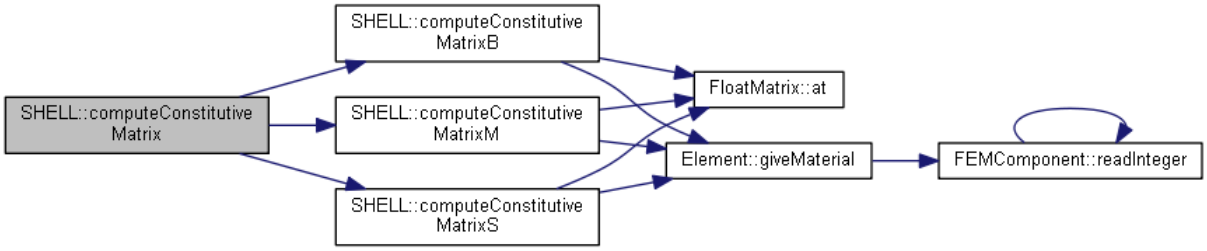


Figure 3.7. Schémas des méthodes déclenchées lors de l'appel de la méthode computeConstitutiveMatrix de la classe Shell pour le calcul de la matrice constitutive.

3.7. Conclusion

Ce chapitre est destiné à illustrer la hiérarchie des classes et les supers classes du code utilisé afin d'implémenter notre classe Shell qui incarne l'élément plaque membrane (coque plane). Ce chapitre nous permet de faire la lumière sur les différentes méthodes déclenchées lors de l'appel de la méthode essentielle ComputeStiffnessMatrix permettant le calcul de la matrice de rigidité.

Chapitre 4
Théorie des plaques et plaques membranes

4.1. Introduction

Ce chapitre est consacré à la présentation des théories des plaques en flexion et plaques membranes dans le cas des petits déplacements. En détaillant les relations cinématiques à savoir le champ de déplacement et le champ de déformation, on passe par la suite aux relations contraintes déformations et on termine par une conclusion.

4.2. Théorie des plaques en flexion

4.2.1. 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 appellerons surface moyenne. Par convention, cette surface moyenne sera le plan XY, l'axe OZ correspond à l'axe transverse selon l'épaisseur [49].

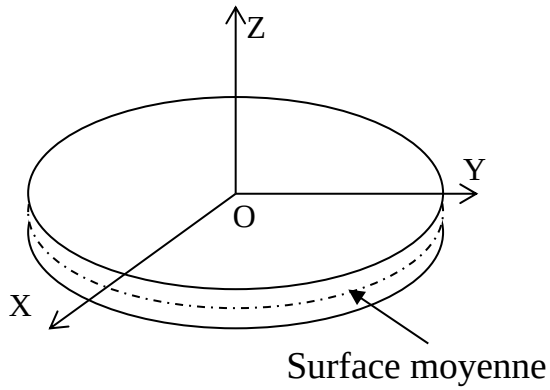


Figure 4.1. Portion d'une Plaque.

4.2.2. Hypothèses

La théorie des plaques [5], [49] repose sur les hypothèses suivantes :

• les contraintes normales σ_{zz} sont négligeables par rapport aux autres composantes de contraintes $(\sigma_{zz} = 0)$;

• les pentes de la surface moyenne après déformation, sont supposées petites par rapport à l'unité (petite déflexion du plan moyen) ;

• l'hypothèse dite des sections droites (ou planes) : les points matériels situés sur une normale à la surface moyenne non déformée restent sur une droite dans la configuration déformée. Cette hypothèse a été proposé par plusieurs hauteurs dont les plus connus sont Reissner , Hencky, Mindlin et permet de prendre en compte l'influence des déformations de CT.

Les deux théories des plaques les plus importantes pour l'analyse linéaire des structures :

• La théorie de Hencky, Reissner, Mindlin ou théorie linéaire des plaques avec cisaillement transversal.

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

Pour les plaques homogènes isotropes, la validité de la théorie de plaque retenue dépend des caractéristiques géométriques [5].

On admet généralement les hypothèses de Mindlin si : $4 \leq L/h \leq 20$

et celles de Kirchoff si : $L/h > 20$

Où : L : est la longueur de la plaque dans le plan (x-y).

h : est l'épaisseur de la plaque.

L/h : est le facteur d'élancement de la plaque.

4.2.3. Conventions de signe pour déplacements et rotations

Les notations adoptées ci-après [49] pour les déplacements sont définies à la figure ci-dessous :

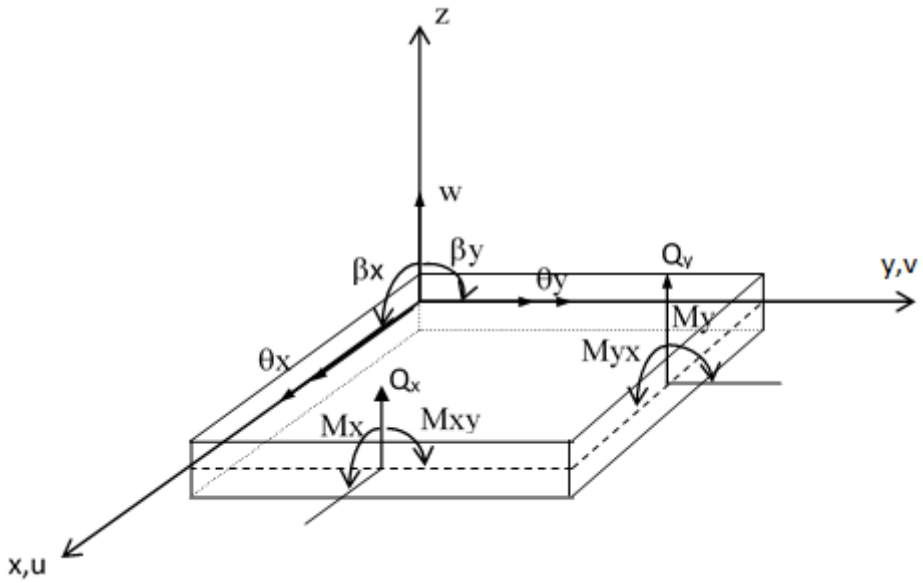


Figure 4.2. Conventions générales.

Soient les déplacements dans le plan u et v, le déplacement transversale w et les rotations β_x et β_y , ou θ_x et θ_y. On a évidemment :

$$\beta_x = \theta_y \qquad\qquad \beta_y = -\theta_x \tag{4.1}$$

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

4.2.4. Relations cinématiques [5], [49]

4.2.4.1. 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 $\beta_x(x, y)$ et $\beta_y(x, y)$.

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

$$\begin{aligned} u(x,y,z) &= z\,\beta_x(x,y) \\ v(x,y,z) &= z\,\beta_y(x,y) \\ w(x,y,z) &= w(x,y) \end{aligned} \tag{4.2}$$

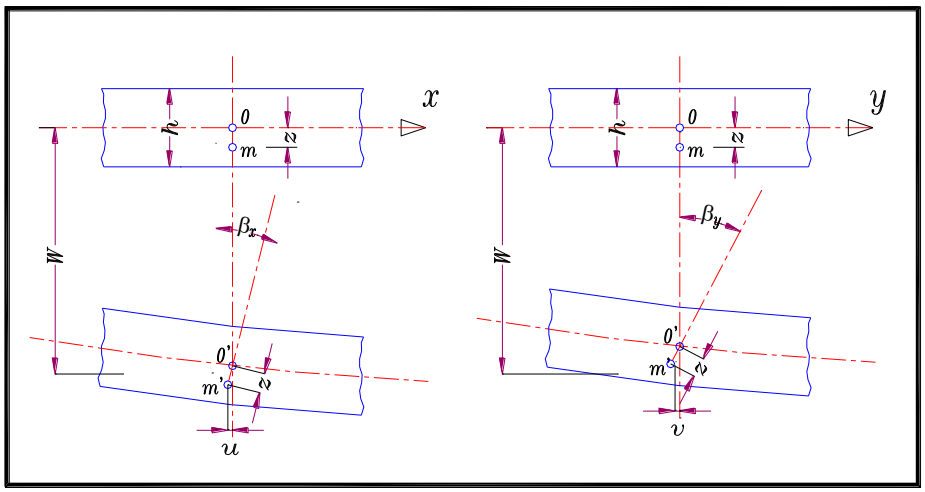


Figure 4.3. Rotations β_x et β_y.

4.2.4.2. Champ de déformations

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

$$\varepsilon_x = \frac{\partial u}{\partial x}$$

$$\varepsilon_y = \frac{\partial v}{\partial y}$$

$$2\varepsilon_{xy} = \gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \tag{4.3}$$

$$2\varepsilon_{xz} = \gamma_{xz} = \frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}$$

$$2\varepsilon_{yz} = \gamma_{yz} = \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}$$

Après substitution des déplacements dans (4.3), nous obtenons les composantes du tenseur de déformations en fonction des trois degrés de libertés w, β_x, β_y :

$$\varepsilon_x = z\frac{\partial \beta_x}{\partial x}$$

$$\varepsilon_y = z\frac{\partial \beta_y}{\partial y}$$

$$\gamma_{xy} = z\left(\frac{\partial \beta_x}{\partial y} + \frac{\partial \beta_y}{\partial x}\right) \tag{4.4}$$

$$\gamma_{xz} = \beta_x + \frac{\partial w}{\partial x}$$

$$\gamma_{yz} = \beta_y + \frac{\partial w}{\partial y}$$

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

$$\{\varepsilon\} = \left\{\{\varepsilon_f\}^T, \{\varepsilon_c\}^T\right\}^T = \left\{z\{\chi\}^T, \{\gamma\}^T\right\}^T \tag{4.5}$$

$$\{\varepsilon_f\} = \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = z\{\chi\} \qquad \{\varepsilon_c\} = \{\gamma\} = \begin{Bmatrix} \gamma_{xz} \\ \gamma_{yz} \end{Bmatrix} = \begin{Bmatrix} \beta_x + \frac{\partial w}{\partial x} \\ \beta_y + \frac{\partial w}{\partial y} \end{Bmatrix} \tag{4.6}$$

avec :

$$\{\chi\}=\begin{Bmatrix} \dfrac{\partial \beta_x}{\partial x} \\ \dfrac{\partial \beta_y}{\partial y} \\ \dfrac{\partial \beta_x}{\partial y}+\dfrac{\partial \beta_y}{\partial x} \end{Bmatrix} \tag{4.7}$$

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

D'où on peut écrire :

$$\{\bar{\varepsilon}\}=\left\{\{\chi\}^T,\{\gamma\}^T\right\}^T \tag{4.8}$$

4.2.5. Relation contraintes-déformations [5], [49]

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

$$\{\sigma\}=[C]\{\varepsilon\} \tag{4.9}$$

telle que :

$[C]$: étant la matrice constitutive pour un état de contrainte plane.

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

$$\begin{Bmatrix} \{\sigma_f\} \\ \{\sigma_c\} \end{Bmatrix}=\begin{bmatrix} [C_f] & [0] \\ [0] & [C_c] \end{bmatrix}\begin{Bmatrix} \varepsilon_f \\ \gamma \end{Bmatrix} \tag{4.10}$$

avec :

$$\{\sigma_f\} = \{\sigma_x,\ \sigma_y,\ \sigma_{xy}\}^T \qquad \{\sigma_c\}=\{\sigma_{xz},\sigma_{yz}\}^T \tag{4.11}$$

$$[C_f]=\frac{E}{1-\nu^2}\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \dfrac{1-\nu}{2} \end{bmatrix} \qquad [C_c]=\frac{E}{2(1+\nu)}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4. 12}$$

Tels que :

$[C_c]$: matrice constitutive de cisaillement transversal.

E : Module de Young.

ν : Coefficient de Poisson.

4.2.6. Relation efforts résultants – déformations

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 :

$$\{M\} = \begin{Bmatrix} M_x \\ M_y \\ M_{xy} \end{Bmatrix} = \int_{-h/2}^{+h/2} z \left\{\sigma_x , \sigma_y , \sigma_{xy}\right\}^T dz = \left[D_f\right] \{\chi\} \qquad (4.13)$$

Les efforts tranchants sont :

$$\{T\} = \begin{Bmatrix} T_x \\ T_y \end{Bmatrix} = \int_{-h/2}^{+h/2} \left\{\sigma_{xz} , \sigma_{yz}\right\}^T dz = \left[D_c\right] \{\gamma\} \qquad (4.14)$$

Avec :

h : L'épaisseur de la plaque.

$$\left[D_f\right] = \frac{Eh^3}{12(1-\nu^2)} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \qquad \left[D_c\right] = \frac{Ehk}{2(1+\nu)} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad (4.15)$$

k : facteur de correction de cisaillement.

4.3. Théorie des plaques membranes

4.3.1. Etat membranaire et état flexionnel [38]

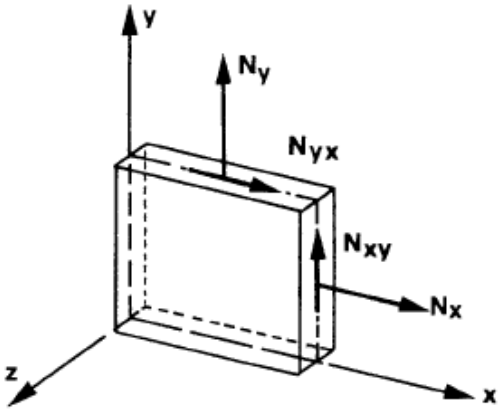
L'état de contrainte, par lequel l'élément structural résiste aux actions extérieures, est caractérisé par des *efforts intérieurs* définis au niveau de la surface moyenne. La trace de cette surface dans une section droite s'appelle la *ligne moyenne*. Les efforts intérieurs sont décrits par unité de longueur de ligne moyenne dans les sections droites.

L'***état membranaire*** s'associe aux efforts intérieurs de type force agissant dans la surface moyenne, à savoir les *efforts normaux* et les *efforts tangentiels*. L'***état flexionnel***

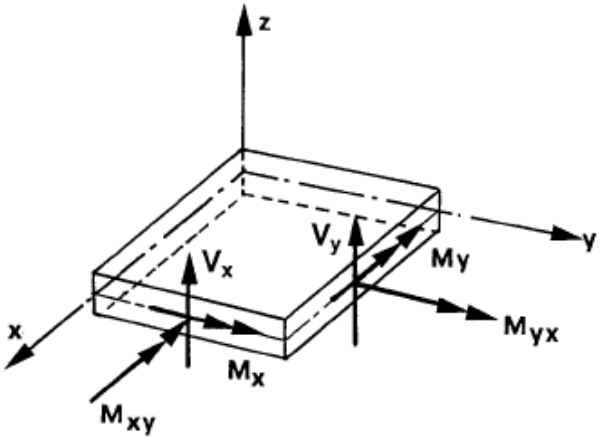
regroupe les efforts intérieurs de caractère flexionnel, soit *les moments de flexion*, les *moments de torsion* et les *efforts tranchants*.

Selon le mode de travail, on peut distinguer quatre types d'éléments structuraux :

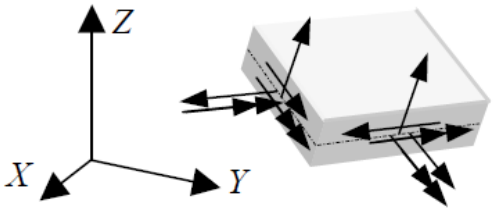
- **l'élément de *paroi*** : est défini par la géométrie plane de sa surface moyenne (plan moyen) et par son épaisseur ; sollicité par des charges agissant dans son plan moyen, il y résiste par un *état membranaire* (figure 4.4a); les efforts normaux et tangentiels résultent d'ailleurs de l'état plan de contrainte ;
- **l'élément de *plaque*** : est défini par la géométrie plane de sa surface moyenne (plan ou feuillet moyen) ; il résiste aux charges agissant normalement à son plan moyen par un *état flexionnel* (figure 4.4b) ;
- **l'élément de *plaque-membrane*** : est la superposition des deux cas précédents et réunit donc l'état *membranaire* de paroi et l'état *flexionnel* de plaque (figure 4.4c) ; bien que plan, il se comporte de manière spatiale, pouvant être soumis à des charges quelconques, tant parallèles que perpendiculaires à son plan moyen ; il constitue la base des *structures plissées* ;
- Enfin, l'élément structural de ***coque*** : est, par nature, courbe ; il utilise les deux états d'efforts intérieurs, *membranaires et flexionnels*, pour s'opposer aux actions arbitraires pouvant le solliciter (figure 4.4d).



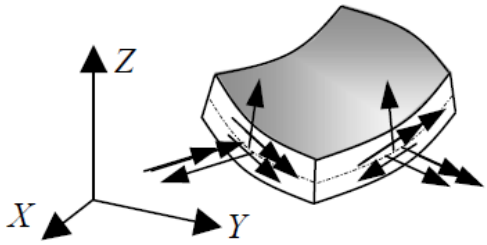
(a) Etat membranaire de paroi :
efforts normaux *Nx* et *Ny* (*Nx* = *t* σ_x *; Ny* = *t* σ_y)
efforts tangentiels : *Nxy* = *Nyx* (*Nxy* = *t* τ_{xy}).



(b) Etat flexionnel de plaque :
moments de flexion *Mx* et *My*
moments de torsion *Mxy* = *Myx*
efforts tranchants *Vx* et *Vy* .



(c) Plaque-membrane : superposition des états membranaire et flexionnel.



(d) Coque : cinq efforts intérieurs par section droite.

Figure 4.4. Efforts intérieurs [38].

L'élément structural plaque-membrane est aussi appelé *élément plan de coque*.

4.3.2. Utilité de l'élément plaque membrane ou élément plan de coque [38],[39]

Il est très simple dans un élément fini plan, de combiner un champ membranaire à un champ flexionnel. On obtient un élément plaque membrane, aussi appelé élément plan de coque. L'état flexionnel de cet élément est basé sur la théorie des plaques minces (Kirchoff) ou d'épaisseur modérée (Mindlin,)[38], qui fait objet de notre étude. Il permet de résoudre les structures tridimensionnelles [39] telles que montré dans figure 4.5 :

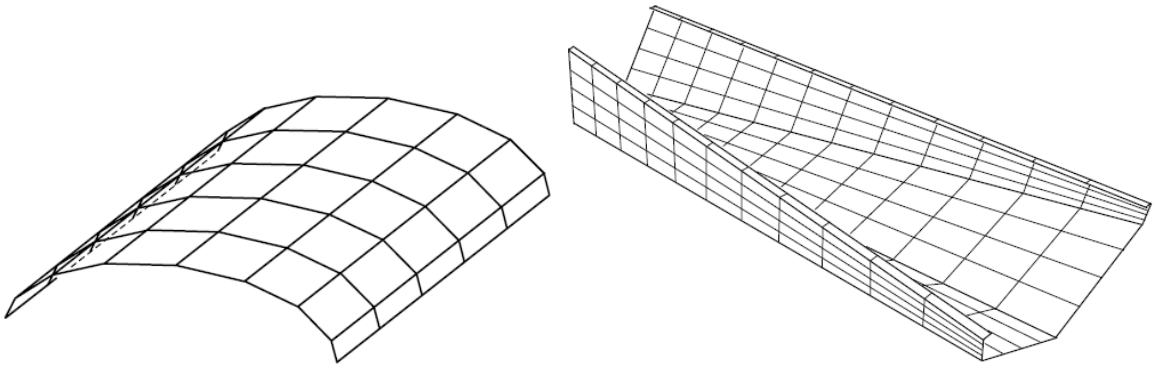


Figure 4.5. Structure plissée et coque modélisées par des éléments plans de coque [38].

• les structures plissées qui sont autre qu'un assemblage de panneaux plans, comme les ponts a caisson, toiture plissés, murs de soutènement etc ;

• les coques, surfaces courbes approchées par un assemblage de facettes planes.

4.3.3. Cinématique des plaques membranes

4.3.3.1. Champ de déplacements

Le champ de déplacement d'un point quelconque $q(x, y, z)$ est défini avec l'hypothèse des sections planes ou droites (hypothèse de Mindlin, prise en compte du cisaillement transversal) [5], [38] :

$$\begin{aligned} u(x,y,z) &= u(x,y) + z\,\beta_x(x,y) \\ v(x,y,z) &= v(x,y) + z\,\beta_y(x,y) \\ w(x,y,z) &= w(x,y) \end{aligned} \tag{4.16}$$

$$\begin{Bmatrix} u(x,y,z) \\ v(x,y,z) \\ w(x,y,z) \end{Bmatrix} = \begin{Bmatrix} u(x,y) \\ v(x,y) \\ 0 \end{Bmatrix} + z \begin{Bmatrix} \beta_x(x,y) \\ \beta_y(x,y) \\ w(x,y) \end{Bmatrix} \tag{4.17}$$

Le champ des déplacements est défini par les cinq degrés de libertés suivants :

$\beta_x$et β_y : sont les rotations de la normale dans les plans (x,z) et (y,z) respectivement ;

u et v : sont les déplacements de membrane dans le plan (x,y);

w(x,y) : est le déplacement transversal.

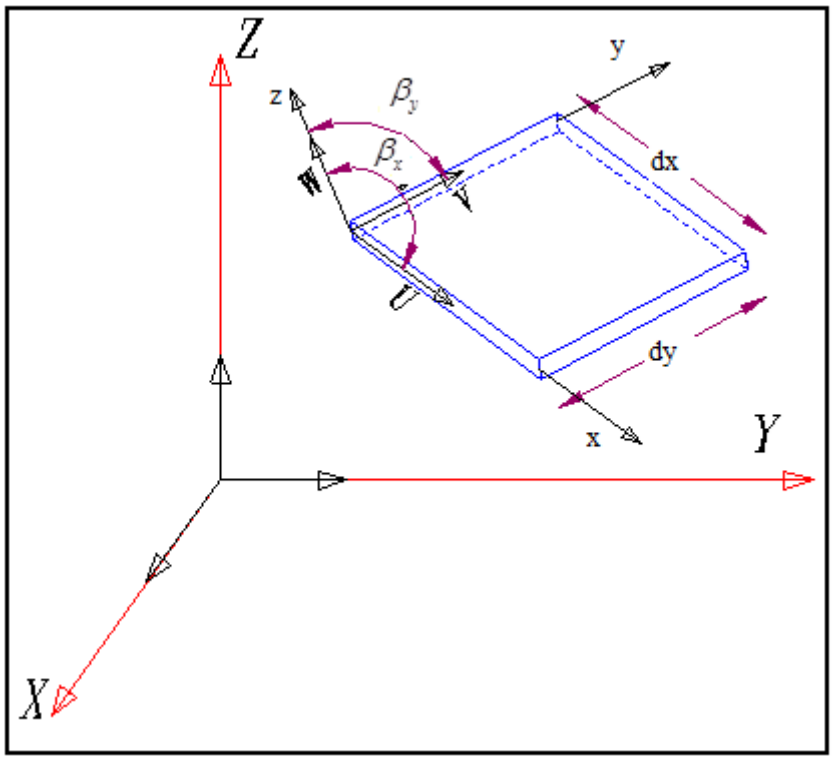


Figure 4.6 Portion d'une plaque membrane.

4.3.3.2. Champ de déformations

L'état de déformation peut être défini en fonction des cinq variables par les relations cinématiques suivantes :

$$\begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \\ \gamma_{xz} \\ \gamma_{yz} \end{Bmatrix} = \begin{Bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \\ \frac{\partial u}{\partial z} + \frac{\partial w}{\partial x} \\ \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} \end{Bmatrix} = \begin{Bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \\ 0 \\ 0 \end{Bmatrix} + \begin{Bmatrix} z\frac{\partial \beta_x}{\partial x} \\ z\frac{\partial \beta_y}{\partial y} \\ z(\frac{\partial \beta_x}{\partial y} + \frac{\partial \beta_y}{\partial x}) \\ \beta_x + \frac{\partial w}{\partial x} \\ \beta_y + \frac{\partial w}{\partial y} \end{Bmatrix} \quad (4.18)$$

Il peut être décomposé en :

$$\{\varepsilon\} = \begin{Bmatrix}\{\varepsilon_m\}\\ \{0\}\end{Bmatrix} + \begin{Bmatrix}z\{\chi\}\\ \{\varepsilon_c\}\end{Bmatrix} \tag{4.19}$$

Avec :

$$\{\varepsilon_m\} = \begin{Bmatrix}\dfrac{\partial u}{\partial x}\\ \dfrac{\partial v}{\partial y}\\ \dfrac{\partial u}{\partial y}+\dfrac{\partial v}{\partial x}\end{Bmatrix} \quad \{\chi\} = \{\bar{\varepsilon}_f\} = \begin{Bmatrix}\dfrac{\partial \beta_x}{\partial x}\\ \dfrac{\partial \beta_y}{\partial \beta_y}\\ \dfrac{\partial \beta_x}{\partial y}+\dfrac{\partial \beta_y}{\partial x}\end{Bmatrix} \quad \{\varepsilon_c\} = \begin{Bmatrix}\beta_x+\dfrac{\partial w}{\partial x}\\ \beta_y+\dfrac{\partial w}{\partial y}\end{Bmatrix} \tag{4.20}$$

$\{\varepsilon_m\}, \{\varepsilon_f\}, \{\varepsilon_c\}$: sont respectivement les vecteurs de déformations de membrane, de flexion et de cisaillement.

$\{\chi\}$: est le vecteur des courbures.

4.3.4. Relation contraintes-déformations [5], [38]

En plus des contraintes de flexion et de cisaillement, nous avons aussi la contribution des contraintes de membrane dans la structure.

La contrainte normale σ_z , est considérée comme négligeable.

Suite à cette hypothèse, on peut relier les composantes du tenseur de contraintes en un point à celle du tenseur de déformation. La relation contrainte déformations, d'après la loi de Hooke, généralisée s'écrit :

$$\{\sigma\} = [C]\{\varepsilon\}$$

$$\{\sigma\} = \left\{\{\sigma_m\}^T, \{\sigma_f\}^T, \{\sigma_c\}^T\right\}^T \tag{4.21}$$

Pour un matériau isotrope, elles peuvent être réécrites sous la forme suivante :

$$\begin{Bmatrix}\{\sigma_m\}\\ \{\sigma_f\}\\ \{\sigma_c\}\end{Bmatrix} = \begin{bmatrix}[C_m] & [0] & [0]\\ [0] & [C_f] & [0]\\ [0] & [0] & [C_c]\end{bmatrix}\begin{Bmatrix}\{\varepsilon_m\}\\ \{\varepsilon_f\}\\ \{\varepsilon_c\}\end{Bmatrix} \tag{4.22}$$

Avec :

$$[C_m] = \frac{E}{1-\nu^2}\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \tag{4.23}$$

$$[C_f] = \frac{E}{1-\nu^2}\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \qquad [C_c] = \frac{E}{2(1+\nu)}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.24}$$

4.3.5. Contraintes et efforts intérieurs [38]

Les efforts intérieurs sont subdivisés en efforts membranaires et efforts flexionnels (voir figure 4.4c):

- ***les efforts normaux :***

$$N_x = \int_{-h/2}^{h/2} \sigma_x \, dz \qquad N_y = \int_{-h/2}^{h/2} \sigma_y \, dz \tag{4.25}$$

- ***les efforts tangentiels :***

$$N_{xy} = \int_{-h/2}^{h/2} \sigma_{xy} \, dz \qquad N_{yx} = \int_{-h/2}^{h/2} \sigma_{yx} \, dz \tag{4.26}$$

- ***les moments de flexions :***

$$M_x = \int_{-h/2}^{h/2} \sigma_x \, z \, dz \qquad M_y = \int_{-h/2}^{h/2} \sigma_y \, z \, dz \tag{4.27}$$

- ***les moments de torsion :***

$$M_{xy} = \int_{-h/2}^{h/2} \sigma_{xy} \, z \, dz = M_{yx} \tag{4.28}$$

- ***les efforts tranchants :***

$$T_x = \int_{-h/2}^{h/2} \sigma_{xz} \, dz \qquad T_y = \int_{-h/2}^{h/2} \sigma_{yz} \, dz \tag{4.29}$$

N_x , N_y , N_{xy} : efforts résultants de membrane ;

M_x , M_y , M_{xy} : efforts résultants de flexion ;

T_x , T_y : efforts résultants de cisaillement ou effort tranchants.

Le vecteur des efforts résultants $\{E\}$ est donné par :

$$\{E\} = \begin{bmatrix} [D_m] & [0] & [0] \\ [0] & [D_f] & 0 \\ [0] & [0] & [D_c] \end{bmatrix} \begin{Bmatrix} \{\varepsilon_m\} \\ \{\bar{\varepsilon}_f\} \\ \{\varepsilon_c\} \end{Bmatrix} \tag{4.30}$$

$$\{N\} = \begin{Bmatrix} N_x \\ N_y \\ N_{xy} \end{Bmatrix} = [D_m]\{\varepsilon_m\} = h\,[C_f]\{\varepsilon_m\} \tag{4.31}$$

$$\{M\} = \begin{Bmatrix} M_x \\ M_y \\ M_{xy} \end{Bmatrix} = [D_f]\{\bar{\varepsilon}_f\} = \frac{h^3}{12}\,[C_f]\{\chi\} \tag{4.32}$$

$$\{T\} = \begin{Bmatrix} T_x \\ T_y \end{Bmatrix} = [D_c]\{\gamma\} = h\,k[C_c]\{\gamma\} \tag{4.33}$$

Tel que :

$$[D_m] = \frac{Eh}{(1-\nu^2)} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \dfrac{1-\nu}{2} \end{bmatrix} \tag{4.34}$$

4.4. Conclusion

Dans ce chapitre la théorie des plaques en flexion, et des plaques membranes est survolée, car il est impératif de faire un rappel théorique pour faciliter sa compréhension, afin de passer à une formulation élément fini de ces derniers, qui fera l'objet du chapitre suivant.

Chapitre 5
Formulation Eléments Finis Des Plaques et Plaques Membranes

5.1. Introduction

Dans ce chapitre, nous abordons la formulation élément fini pour l'analyse linéaire des problèmes en statique et dynamique. A partir du principe des travaux virtuels et d'Hamilton nous formulons l'équation du mouvement en statique et l'équation différentielle du second ordre en dynamique respectivement. Enfin, nous procédons à la construction des matrices de rigidité de masse ainsi que le vecteur charge équivalent.

5.2. Formulation en statique linéaire [49]

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

$$[K]\{q\}=\{F\} \tag{5.1}$$

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

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

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

Le théorème qui conduit à la relation (5.1), dans le cas d'une approche de type « déplacement », est celui des travaux virtuels.

5.2.1. 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 [5], [49], exprime le bilan des travaux virtuels interne et externe, lorsque le corps est en équilibre :

$$\int_v \sigma_{ij}\ \delta\varepsilon_{ij}\ dv \quad = \quad \int_v f_i^v\ \delta u_i\ dv \quad + \quad \int_S f_i^S\ \delta u_i\ dS \quad + \quad Q_i\ \delta u_i \tag{5.2}$$

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 ;

σ_{ij} : Tenseur des contraintes ;

ε_{ij} : Tenseur des déformations.

Sous forme matricielle, nous avons :

$$\int_V <\delta\varepsilon>\{\sigma\}dV = \int_V <\delta u>\{f_V\}dV + \int_S <\delta u>\{f_S\}dS + \sum_i <\delta u_i>\{Q_i\} \qquad (5.3)$$

$<\varepsilon>$: Vecteur des déformations, transposé ;

$\{\sigma\}$: 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.

On peut introduire la fonctionnelle énergie potentielle totale V , et le principe du travail virtuel s'écrit :

$$\delta U = \delta W$$

Ou : $$\delta V = \delta(U - W) = 0 \qquad (5.4)$$

Avec :

V : énergie potentielle totale.

U : énergie de déformation.

W : travail des forces appliquées.

5.2.2. Principe de la méthode des éléments finis en statique [4], [40], [49]

La méthode des éléments finis de type déplacement permet de ramener les problèmes de milieux continus à des problèmes discrets.

En statique, nous considérons successivement :

- La discrétisation du domaine en éléments finis.
- La formulation élémentaire au niveau de l'élément.
- La formulation globale après assemblage.

5.2.2.1. Discrétisation de la structure

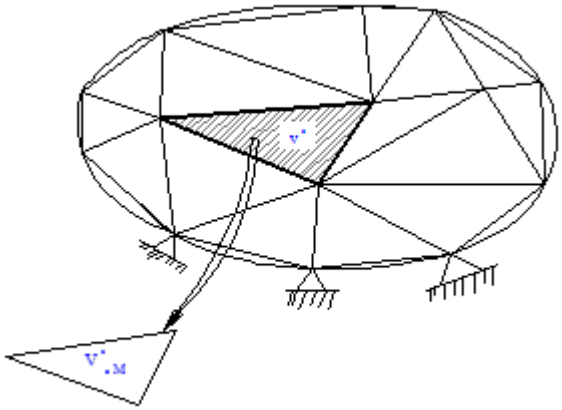


Figure 5.1. Discrétisation de la structure.

La phase de discrétisation, consiste à découper la structure (domaine continu V), en sous-domaines V^e, de forme géométrique simple que l'on appelle « éléments finis », interconnectés en des points remarquables appelés « nœuds ». Dans chaque élément, on définit une approximation des déplacements en fonction des déplacements aux nœuds, soit :

$$\{u(x,y,z)\}^e = [N(x,y,z)]^e \{q\}^e \tag{5.5}$$

avec :

$\{u\}^e$: vecteur des déplacements en un point M de l'élément e.

$[N(x,y,z)]^e$: matrice d'interpolation pour l'élément e.

$\{q\}^e$: vecteur des déplacements aux nœuds de l'élément e.

5.2.2.2. Formulation élémentaire

La formulation au niveau de l'élément, consiste à rechercher pour chaque élément des expressions matricielles d'énergie de déformation et du travail des forces appliquées en fonction des déplacements aux nœuds [49]. Ceci nécessite le calcul de matrices caractéristiques de l'élément : matrice de rigidité, et du vecteur de forces équivalentes.

L'expression de l'énergie potentielle totale en fonction des déplacements aux nœuds de l'élément, est :

$$V^e = U^e - W^e = \frac{1}{2}\{q\}^{eT}[K]^e\{q\}^e - \{q\}^{eT}\{F\}^e \tag{5.6}$$

Avec :

$[K]^e$: matrice de rigidité élémentaire ;

$\{F\}^e$: vecteur des forces équivalentes élémentaires.

$$[K]^e = \int_{V^e} [B]^T [C][B]\, dV \tag{5.7}$$

$$\{F\}^e = \int_{V^e} [N]^T \{f_V\}\, dV + \int_{S^e} [N]^T \{f_S\}\, dS \tag{5.8}$$

$\{f_V\}$et $\{f_S\}$: sont respectivement les vecteurs des forces de volume et de surface.

On rappelle que l'on a :

$$\{\sigma\} = [C]\{\varepsilon\} \tag{5.9}$$

$$\{\varepsilon\} = [D]\{u\} = [B]\{q\} \tag{5.10}$$

avec :

$[D]$: Matrice d'opérateurs différentiels.

$[B]$**:** Matrice reliant les déformations aux variables nodales.

5.2.2.3. Formulation globale

La formulation globale du problème, consiste à rechercher pour la structure complète l'expression matricielle d'énergie de déformation et du travail des forces appliquées en fonction des déplacements en tous les nœuds de la structure. Ceci nécessite l'assemblage des caractéristiques élémentaires (matrice de rigidité, vecteur des forces équivalentes) pour tous les éléments [49].

L'énergie potentielle totale de la structure peut être obtenue par sommation des énergies potentielles totales élémentaires, soit :

$$V = \sum_{éléments} V^e = \sum_{éléments} \left\{ \frac{1}{2} \{q\}^{eT} [K]^e \{q\}^e - \{q\}^{eT} \{F\}^e \right\} \tag{5.11}$$

Soit $\{q\}^T$ le vecteur ligne des déplacements aux nœuds de la structure, soit pour une structure à m nœuds :

$$\{q\}^T = \{q_1^T \ldots q_i^T \ldots q_m^T\} \tag{5.12}$$

avec :

$\{q_i\}$: sous-vecteur des déplacements au nœud i.

On peut définir pour chaque élément une relation matricielle permettant d'établir une correspondance entre les déplacements aux nœuds de l'élément $\{q\}^e$ et les déplacements aux nœuds de la structure $\{q\}$, soit :

$$\{q\}^e = [\mathrm{B}]^e \{q\} \qquad (5.13)$$

$$(n_e \times 1) = (n_e \times N).(N \times 1)$$

avec :

$[\mathrm{B}]^e$: Matrice de localisation de l'élément.

n_e : Nombre de degrés de liberté de l'élément.

N : Nombre de degrés de liberté de la structure.

Chaque relation (5.13) permet de repérer ou de localiser les d.d.l. de chaque élément dans l'ensemble des d.d.l. de la structure.

En utilisant les relations (5.11) et (5.13), on peut écrire :

$$V = \sum_{\text{éléments}} \left\{ \frac{1}{2} \{q\}^T [\mathrm{B}]^{eT} [K]^e [\mathrm{B}]^e \{q\} - \{q\}^T [\mathrm{B}]^{eT} \{F\}^e \right\} \qquad (5.14)$$

D'où :

$$V = \frac{1}{2} \{q\}^T [K] \{q\} - \{q\}^T \{F\} \qquad (5.15)$$

avec :

$$[K] = \sum_{\text{éléments}} [\mathrm{B}]^{eT} [K]^e [\mathrm{B}]^e \qquad (5.16)$$

$$\{F\} = \sum_{\text{éléments}} [\mathrm{B}]^{eT} \{F\}^e \qquad (5.17)$$

$[K]$: Matrice de rigidité de la structure.

$\{F\}$: Vecteur des forces équivalentes pour la structure complète.

Dans le cas de forces ponctuelles appliquées aux nœuds de la structure (vecteur {P}), l'expression de $\{F\}$ devient :

$$\{F\} = \{P\} + \sum_{\text{éléments}} [\mathrm{B}]^{eT} \{F\}^e \qquad (5.18)$$

Ces expressions permettent d'obtenir par application directe du principe des travaux virtuels, le système des équations d'équilibre des nœuds. En effet, on a :

$$\delta U = \delta W$$

$$\{\delta q\}^T [K] \{q\} = \{\delta q\}^T \{F\} \qquad (5.19)$$

d'où :

$$[K]\{q\} = \{F\} \tag{5.20}$$

5.3. Formulation en dynamique linéaire [45]

De nombreux problèmes d'analyse des structures peuvent être traités par les méthodes d'analyse statique. Cependant, il existe également de nombreux cas où l'on ne peut négliger les forces d'inertie et d'amortissement résultant de la variation des forces appliquées, on utilisera alors les méthodes d'analyse dynamique.

5.3.1. Formulation des équations de mouvement [14], [49], [64]

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

Soit le Lagrangien : $L = T - V$

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

$$T = \frac{1}{2}\int_V \rho\, \dot{u}_i \dot{u}_i\, dV \tag{5.21}$$

$$V = U - W = \frac{1}{2}\int_V \sigma_{ij}\varepsilon_{ij} dV - \int_V f_i^V u_i\, dV - \int_S f_i^S u_i\, dS \tag{5.22}$$

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 extérieures 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 dérivés :

$$L = T(\dot{q}_i) - V(q_i) \tag{5.23}$$

Pour une structure sans amortissement nous avant les équations d'Euler-Lagrange [49], suivante :

$$\frac{\partial}{\partial t}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = 0 \tag{5.24}$$

ou encore :

$$\frac{\partial}{\partial t}\left(\frac{\partial T}{\partial \dot{q}_i}\right) - \frac{\partial T}{\partial q_i} + \frac{\partial U}{\partial q_i} = F_i(t) \tag{5.25}$$

avec :

$F_i(t)$: force définie par le travail virtuel des forces extérieures : $\delta W = F_i \delta q_i$

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

$$T = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \dot{q}_i M_{ij} \dot{q}_j = \frac{1}{2}\dot{q}^T M \dot{q} \tag{5.26}$$

$$U = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} q_i K_{ij} q_j = \frac{1}{2} q^T K q \tag{5.27}$$

Les équations de Lagrange deviennent alors :

$$[M]\{\ddot{q}\} + [K]\{q\} = \{F(t)\} \tag{5.28}$$

Dans le cas d'un système avec amortissement visqueux, les équations de Lagrange s'écrivent :

$$\frac{\partial}{\partial t}\left(\frac{\partial T}{\partial \dot{q}_i}\right) - \frac{\partial T}{\partial q_i} + \frac{\partial U}{\partial q_i} - F_i^* = F_i(t) \tag{5.29}$$

Le terme additionnel $(-F_i^*)$ représente la force de dissipation visqueuse : $F^* = -C\dot{q}$

D'où les équations du mouvement :

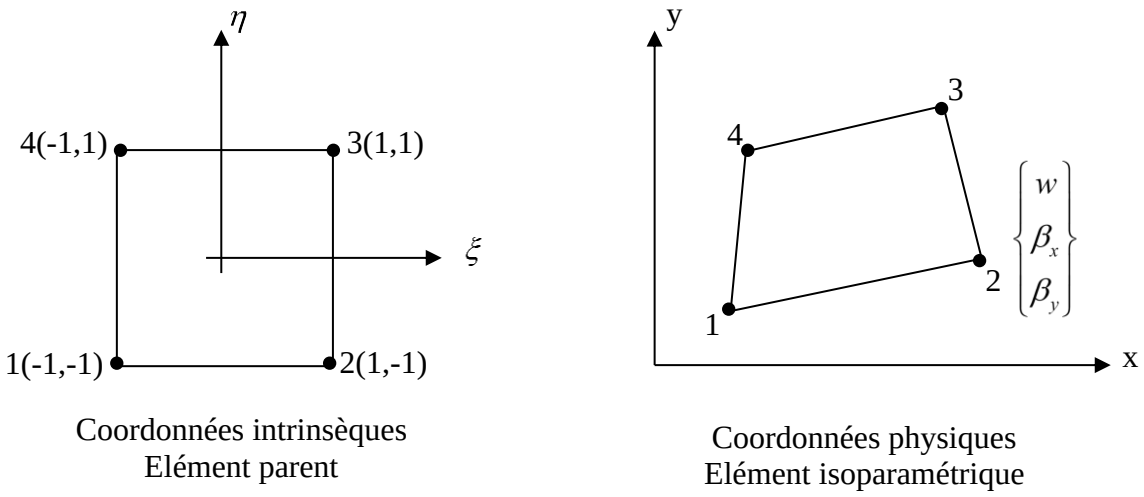
$$[M]\{\ddot{q}\} + [C]\{\dot{q}\} + [K]\{q\} = \{F(t)\} \tag{5.30}$$

5.4. Eléments finis de l'élément plaque avec cisaillement transversal [5] [49]

La formulation des éléments basés sur la théorie de Hencky-Mindlin avec prise en compte du cisaillement transversal 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 $w, \beta_x, et\ \beta_y$ [5], [49].

5.4.1. Discrétisation du champ de déplacements

Nous considérons des éléments de type quadrilatère auxquels nous appliquons la formulation isoparamétrique [5], [22], [49].



Coordonnées intrinsèques
Elément parent

Coordonnées physiques
Elément isoparamétrique

Figure 5.2. Elément isoparamétrique de plaque avec cisaillement transversal.

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

$$\begin{aligned} w &= N^T(\xi,\eta)W \\ \beta_x &= N^T(\xi,\eta)\,\widehat{\beta}_x \\ \beta_y &= N^T(\xi,\eta)\widehat{\beta}_y \end{aligned} \qquad (5.31)$$

Ou :

$$\begin{aligned} w &= \sum_{i=1}^{n_e} N_i w_i \\ \beta_x &= \sum_{i=1}^{n_e} N_i \beta_{xi} \\ \beta_y &= \sum_{i=1}^{n_e} N_i \beta_{yi} \end{aligned} \qquad (5.32)$$

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 :

$$N^T = \begin{bmatrix} N_1 & N_2 & N_3 & N_4 \end{bmatrix} \tag{5.33}$$

Avec :
$$N_i(\xi,\eta) = \frac{1}{4}(1+\xi\xi_i)(1+\eta\eta_i) \qquad i = 1,4 \tag{5.34}$$

$\xi_i ou \eta_i$: prenant les valeurs (+1) ou (-1) suivant le nœud considéré.

$$W = \begin{Bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{Bmatrix} \qquad \widehat{\beta}_x = \begin{Bmatrix} \beta_{x1} \\ \beta_{x2} \\ \beta_{x3} \\ \beta_{x4} \end{Bmatrix} \qquad \widehat{\beta}_y = \begin{Bmatrix} \beta_{y1} \\ \beta_{y2} \\ \beta_{y3} \\ \beta_{y4} \end{Bmatrix} \tag{5.35}$$

5.4.2. Discrétisation du champ de déformations [5], [49]

Par substitution de (5.31), dans les relations de déformations (4.6) et (4.7), on obtient les matrices d'interpolation des déformations de flexion et de cisaillement :

$$\{\chi\} = \{\bar{\varepsilon}_f\} = \begin{Bmatrix} \frac{\partial \beta_x}{\partial x} \\ \frac{\partial \beta_y}{\partial y} \\ \frac{\partial \beta_x}{\partial y} + \frac{\partial \beta_y}{\partial x} \end{Bmatrix} = \begin{bmatrix} 0 & \frac{\partial N^T}{\partial x} & 0 \\ 0 & 0 & \frac{\partial N^T}{\partial y} \\ 0 & \frac{\partial N^T}{\partial y} & \frac{\partial N^T}{\partial x} \end{bmatrix} \begin{Bmatrix} W \\ \widehat{\beta}_x \\ \widehat{\beta}_y \end{Bmatrix} \tag{5.36}$$

$$\{\varepsilon_c\} = \{\gamma\} = \begin{Bmatrix} \beta_x + \frac{\partial w}{\partial x} \\ \beta_y + \frac{\partial w}{\partial y} \end{Bmatrix} = \begin{bmatrix} \frac{\partial N^T}{\partial x} & N^T & 0 \\ \frac{\partial N^T}{\partial y} & 0 & N^T \end{bmatrix} \begin{Bmatrix} W \\ \hat{\beta}_x \\ \hat{\beta}_y \end{Bmatrix} \tag{5.37}$$

Soit :

$$\{\chi\} = \{\bar{\varepsilon}_f\} = [B_f]\{q\} \qquad \{\gamma\} = [B_\gamma]\{q\} \tag{5.38}$$

Où :

$$\left[B_f\right] = \begin{bmatrix} 0 & \dfrac{\partial N^T}{\partial x} & 0 \\ 0 & 0 & \dfrac{\partial N^T}{\partial y} \\ 0 & \dfrac{\partial N^T}{\partial y} & \dfrac{\partial N^T}{\partial x} \end{bmatrix} \qquad \left[B_\gamma\right] = \begin{bmatrix} \dfrac{\partial N^T}{\partial x} & N^T & 0 \\ \dfrac{\partial N^T}{\partial y} & 0 & N^T \end{bmatrix} \tag{5.39}$$

5.4.3. Matrice de rigidité

L'expression de l'énergie de déformation [5], [44], [49] permet de calculer la matrice de rigidité, soit :

$$U = U_F + U_C$$

$$U = \frac{1}{2}\int_{V^e} \left\{\varepsilon_f\right\}^T \left\{\sigma_f\right\} dV + \frac{1}{2}\int_{V^e} \{\gamma\}^T \{\sigma_c\} dV = \frac{1}{2}\{q\}^T [K]\{q\} \tag{5.40}$$

$$U = \frac{1}{2}\int_{S^e} \{\chi\}^T \left[D_f\right]\{\chi\} dx dy + \frac{1}{2}\int_{S^e} \{\gamma\}^T \left[D_c\right]\{\gamma\} dx\, dy \tag{5.41}$$

Après substitution des expressions de déformations (5.38) dans l'énergie de déformation, nous obtenons :

$$U = U_F + U_C = \frac{1}{2}\{q\}^T \int_{S^e} \left[B_f\right]^T \left[D_f\right]\left[B_f\right] dx\, dy\, \{q\} + \frac{1}{2}\{q\}^T \int_{S^e} \left[B_\gamma\right]^T \left[D_c\right]\left[B_\gamma\right] dx\ dy \{q\} \tag{5.42}$$

D'où :

$$[K]^e = \left[K_f\right] + \left[K_c\right] = \int_{S^e} \left[B_f\right]^T \left[D_f\right]\left[B_f\right]\ dS + \int_{S^e} \left[B_\gamma\right]^T \left[D_c\right]\left[B_\gamma\right] dS \tag{5.43}$$

$$[K]^e = \int_{-1}^{+1}\int_{-1}^{+1} \left[B_f\right]^T \left[D_f\right]\left[B_f\right] \det\left[J\right]\, d\xi\, d\eta + \int_{-1}^{+1}\int_{-1}^{+1} \left[B_\gamma\right]^T \left[D_c\right]\ \left[B_\gamma\right] \det\ \left[J\right]\, d\xi\, d\eta \tag{5.44}$$

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

La matrice jacobienne $\left[J(\xi, \eta)\right]$ [5], [22], [49] est :

$$\left[J\right] = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial N^T}{\partial \xi} X & \dfrac{\partial N^T}{\partial \xi} Y \\ \dfrac{\partial N^T}{\partial \eta} X & \dfrac{\partial N^T}{\partial \eta} Y \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} \tag{5.45}$$

où :

$$x = \sum_{i=1}^{4} N_i(\xi,\eta)x_i \qquad y = \sum_{i=1}^{4} N_i(\xi,\eta)y_i \tag{5.46}$$

$$\begin{Bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{Bmatrix} = [j] \begin{Bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{Bmatrix} \qquad [j] = [J]^{-1} \tag{5.47}$$

$$[j] = \begin{bmatrix} j_{11} & j_{12} \\ j_{21} & j_{22} \end{bmatrix} = \frac{1}{\det[J]} \begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{22} \end{bmatrix} \tag{5.48}$$

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

$$\{\chi\} = \lfloor B_f \rfloor \{q\} \qquad ; \qquad \{\gamma\} = \lfloor B_\gamma \rfloor \{q\}$$

avec :

$$[B_f] = \begin{bmatrix} 0 & \frac{\partial N^T}{\partial x} & 0 \\ 0 & 0 & \frac{\partial N^T}{\partial y} \\ 0 & \frac{\partial N^T}{\partial y} & \frac{\partial N^T}{\partial x} \end{bmatrix} \qquad [B_\gamma] = \begin{bmatrix} \frac{\partial N^T}{\partial x} & N^T & 0 \\ \frac{\partial N^T}{\partial y} & 0 & N^T \end{bmatrix} \tag{5.49}$$

$$\frac{\partial N^T}{\partial x} = j_{11}\frac{\partial N^T}{\partial \xi} + j_{12}\frac{\partial N^T}{\partial \eta} \qquad ; \qquad \frac{\partial N^T}{\partial y} = j_{21}\frac{\partial N^T}{\partial \xi} + j_{22}\frac{\partial N^T}{\partial \eta} \tag{5.50}$$

La matrice de rigidité $[K]$ est obtenue par intégration numérique de (5.44) de type Gauss [5], [22], [49].

L'intégrale peut être évaluée en utilisant la formule :

$$\int_{-1}^{+1}\int_{-1}^{+1} f(\xi,\eta)\,d\xi\,d\eta = \sum_{i=1}^{r_1}\sum_{j=1}^{r_2} w_i\, w_j\, f(\xi_i,\eta_j) \tag{5.51}$$

Où :

ξ_i, η_j : sont les coordonnées des points d'intégration.

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

r_1 , r_2 : nombre de points d'intégration dans le sens ξ et η.

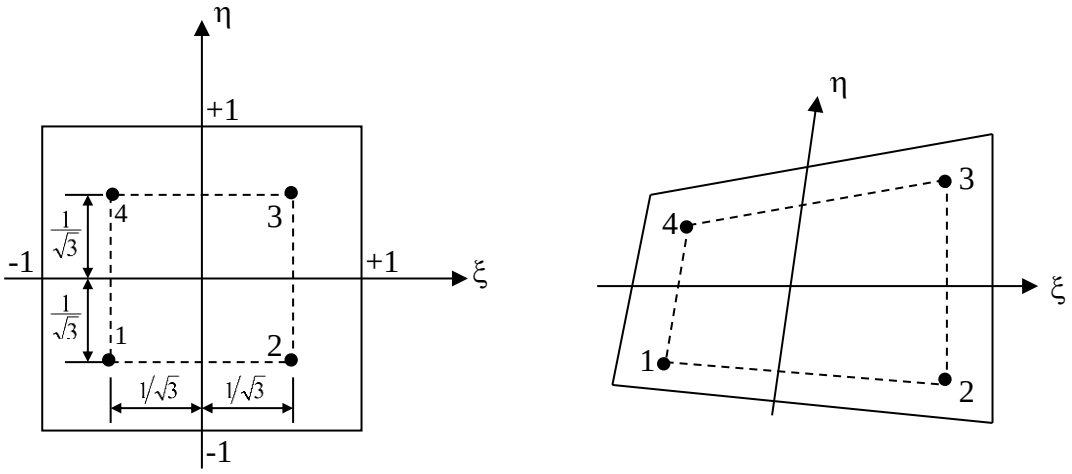


Figure 5.3. Intégration de Gauss (2×2) pour le quadrilatère.

5.4.4. Matrice de masse élémentaire

L'expression de l'énergie cinétique [5], [19], [49], [55], [65] permet de calculer la matrice de masse :

$$T = \frac{1}{2}\int_{v^e} \rho\{\dot{u}\}^{e^T}\{\dot{u}\}^e dv \;\; = \frac{1}{2}\{\dot{q}\}^{e^T}[M]^e\{\dot{q}\}^e \qquad (5.52)$$

On a : $$\{\dot{u}\}^e = [N]\,\{\dot{q}\}^e \qquad (5.53)$$

d'où :

$$T = \frac{1}{2}\;\{\dot{q}\}^{e^T}\int_{v^e} \rho[N]^T[N]dv\;\{\dot{q}\}^e \qquad (5.54)$$

avec :

$$[M]^e = \int_{v^e} \rho[N]^T[N]\,dv \qquad (5.55)$$

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

La matrice de masse élémentaire est calculée par intégration numérique de type Gauss.

5.4.5. Vecteur charge équivalent

L'énergie potentielle des forces extérieures [49], exprimée à l'aide des forces de surface et de volume, s'écrit :

$$W = \int_{V^e} \{u\}^{eT} \{f_v\} dV + \int_{S^e} \{u\}^{eT} \{f_s\} dS = \{q\}^{eT} \{F\}^e \tag{5.56}$$

Nous avons : $$\{u\}^e = [N]^e \{q\}^e \tag{5.57}$$

$$W = \langle q \rangle^e \int_{V^e} [N]^{eT} \{f_V\} dV + \langle q \rangle^e \int_{S^e} [N]^{eT} \{f_S\} dS = \langle q \rangle^e \{F\}^e \tag{5.58}$$

D'où : $$\{F\}^e = \int_{V^e} [N]^{eT} \{f_V\} dV + \int_{S^e} [N]^{eT} \{f_S\} dS \tag{5.59}$$

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

$$\{F\}^e = \int_{S^e} [N]^{eT} f_z \, dS \tag{5.60}$$

5.5. Eléments finis de plaque membrane (élément plan de coque) [5], [38], [39], [42], [49], [61]

Considérons, un élément fini plan quadrilatère. On peut définir, pour cet élément :

• un état plan de contrainte, d'inconnues cinématiques u, v en chaque nœud ; cet état est dit membranaire, car tout se passe dans le seul plan (x, y) de l'élément ;

• un état flexionnel de plaque, de degrés de liberté w, β_x, β_y en chaque nœud ; où tout dépend du comportement transversal (hors plan) de l'élément.

L'état membranaire étant indépendant de l'état flexionnel (Les phénomènes de membrane et de flexion sont découplés), on peut combiner ces deux états afin de créer un élément fini spatial, simultanément plaque et membrane, appelé élément plaque membranaire ou élément plan de coque.

L'état flexionnel de cet élément est basé sur la théorie des plaques d'épaisseur modérée (Mindlin,) [38].

5.5.1. Discrétisation du champ de déplacement

Dans le cas de l'élément coque plane le champ de déplacement est à cinq degrés de liberté par nœud ; à savoir : u_i , v_i, w_i, β_{xi} et β_{yi}.

Pour une interpolation linéaire du champ des déplacements, on a la relation :

$$\begin{Bmatrix} u \\ v \\ w \\ \beta_x \\ \beta_y \end{Bmatrix} = \sum_{i=1}^{n_e} N_i \begin{Bmatrix} u_i \\ v_i \\ w_i \\ \beta_{xi} \\ \beta_{yi} \end{Bmatrix} \tag{5.61}$$

avec :

n_e : indique le nombre de nœuds par élément.

$$N_i(\xi,\eta)=\frac{1}{4}(1+\xi\xi_i)(1+\eta\eta_i) \tag{5.62}$$

$$\begin{Bmatrix} u \\ v \\ w \\ \beta_x \\ \beta_y \end{Bmatrix} = \left[\begin{array}{cccccccccc:ccccc} N_1 & 0 & 0 & 0 & 0 & N_2 & 0 & 0 & 0 & 0 & Nn_e & 0 & 0 & 0 & 0 \\ 0 & N_1 & 0 & 0 & 0 & 0 & N_2 & 0 & 0 & 0 & 0 & N_{ne} & 0 & 0 & 0 \\ 0 & 0 & N_1 & 0 & 0 & 0 & 0 & N_2 & 0 & 0 & 0 & 0 & N_{ne} & 0 & 0 \\ 0 & 0 & 0 & N_1 & 0 & 0 & 0 & 0 & N_2 & 0 & 0 & 0 & 0 & N_{ne} & 0 \\ 0 & 0 & 0 & 0 & N_1 & 0 & 0 & 0 & 0 & N_2 & 0 & 0 & 0 & 0 & N_{ne} \end{array}\right] \begin{Bmatrix} u_1 \\ v_1 \\ w_1 \\ \beta_{x1} \\ \beta_{y1} \\ : \\ : \\ : \\ : \\ : \\ u_{ne} \\ v_{ne} \\ w_{ne} \\ \beta_{xne} \\ \beta_{yne} \end{Bmatrix}$$

5.5.2. Discrétisation du champ de déformation

Par substitution de (5.61) dans les relations de déformations (4.20), on obtient les matrices reliant les déformations de membrane, flexion et cisaillement aux variables nodales :

$$\{\varepsilon_m\} = \begin{Bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial v}{\partial y} \\ \dfrac{\partial u}{\partial y} + \dfrac{\partial v}{\partial x} \end{Bmatrix} = \sum_{i=1}^{4} \begin{bmatrix} \dfrac{\partial N_i}{\partial x} & 0 & 0 & 0 & 0 \\ 0 & \dfrac{\partial N_i}{\partial y} & 0 & 0 & 0 \\ \dfrac{\partial N_i}{\partial y} & \dfrac{\partial N_i}{\partial x} & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} u_i \\ v_i \\ w_i \\ \beta_{xi} \\ \beta_{yi} \end{Bmatrix} \tag{5.63}$$

$$\{\chi\} = \{\bar{\varepsilon}_f\} = \begin{Bmatrix} \dfrac{\partial \beta_x}{\partial x} \\ \dfrac{\partial \beta_y}{\partial y} \\ \dfrac{\partial \beta_x}{\partial y} + \dfrac{\partial \beta_y}{\partial x} \end{Bmatrix} = \sum_{i=1}^{4} \begin{bmatrix} 0 & 0 & 0 & \dfrac{\partial N_i}{\partial x} & 0 \\ 0 & 0 & 0 & 0 & \dfrac{\partial N_i}{\partial y} \\ 0 & 0 & 0 & \dfrac{\partial N_i}{\partial y} & \dfrac{\partial N_i}{\partial x} \end{bmatrix} \begin{Bmatrix} u_i \\ v_i \\ w_i \\ \beta_{xi} \\ \beta_{yi} \end{Bmatrix} \tag{5.64}$$

$$\{\varepsilon_c\} = \begin{Bmatrix} \beta_x + \dfrac{\partial w}{\partial x} \\ \beta_y + \dfrac{\partial w}{\partial y} \end{Bmatrix} = \sum_{i=1}^{4} \begin{bmatrix} 0 & 0 & \dfrac{\partial N_i}{\partial x} & N_i & 0 \\ 0 & 0 & \dfrac{\partial N_i}{\partial y} & 0 & N_i \end{bmatrix} \begin{Bmatrix} u_i \\ v_i \\ w_i \\ \beta_{xi} \\ \beta_{yi} \end{Bmatrix} \tag{5.65}$$

Soit :

$$\{\varepsilon_m\} = [\beta_m]\{q\} \qquad \{\chi\} = \{\bar{\varepsilon}_f\} = [\beta_f]\{q\} \qquad \{\varepsilon_c\} = \{\gamma\} = [\beta_c]\{q\} \tag{5.66}$$

On peut écrire les matrices d'interpolation des déformations comme suit :

$$[B_m] = \begin{bmatrix} \dfrac{\partial N^T}{\partial x} & 0 & 0 & 0 & 0 \\ 0 & \dfrac{\partial N^T}{\partial y} & 0 & 0 & 0 \\ \dfrac{\partial N^T}{\partial y} & \dfrac{\partial N^T}{\partial x} & 0 & 0 & 0 \end{bmatrix} \quad [B_f] = \begin{bmatrix} 0 & 0 & 0 & \dfrac{\partial N^T}{\partial x} & 0 \\ 0 & 0 & 0 & 0 & \dfrac{\partial N^T}{\partial y} \\ 0 & 0 & 0 & \dfrac{\partial N^T}{\partial y} & \dfrac{\partial N^T}{\partial x} \end{bmatrix} \tag{5.67}$$

$$[B_\gamma] = \begin{bmatrix} 0 & 0 & \dfrac{\partial N^T}{\partial x} & N^T & 0 \\ 0 & 0 & \dfrac{\partial N^T}{\partial y} & 0 & N^T \end{bmatrix}$$

5.5.3. Matrice de rigidité

L'expression de l'énergie de déformation [5], [49] permet de calculer la matrice de rigidité, soit :

$$U = U_f + U_c + U_m$$

$$U = \frac{1}{2}\int_{V^e} \{\varepsilon_f\}^T \{\sigma_f\} dV + \frac{1}{2}\int_{V^e} \{\gamma\}^T \{\sigma_c\} dV + \frac{1}{2}\int_{V^e} \{\varepsilon_m\}^T \{\sigma_m\} dV = \frac{1}{2}\{q\}^T [K]\{q\} \quad (5.68)$$

$$U = \frac{1}{2}\int_{S^e} \{\chi\}^T [D_f]\{\chi\} dx dy + \frac{1}{2}\int_{S^e} \{\gamma\}^T [D_c]\{\gamma\} dx dy + \frac{1}{2}\int_{S^e} \{\varepsilon_m\}^T [D_m]\{\varepsilon_m\} dx dy \quad (5.69)$$

d'où :

$$[K] = [K_f] + [K_c] + [K_m] = \int_{S^e} [B_f]^T [D_f][B_f] dS + \int_{S^e} [B_\gamma]^T [D_c][B_\gamma] dS + \int_{S^e} [B_m]^T [D_m][B_m] dS \quad (5.70)$$

5.6. Implémentation du calcul de la matrice de rigidité de l'élément plan de coque [61]

```
FloatMatrix*  SHELLS :: computeStiffnessMatrix ( )
  // Computes numerically the stiffness matrix of the receiver.
{
   this->computeStiffnessMatrixB( );
   this->computeStiffnessMatrixS( );
   this->computeStiffnessMatrixM( );
   stiffnessMatrix = stiffnessMatrix ->GiveCopy( ) ;
   stiffnessMatrix->plus(stiffnessMatrixS);
   stiffnessMatrix->plus(stiffnessMatrixM);
   return stiffnessMatrix ;
}
```

Tableau 5.1. Méthode permettant le calcul de la matrice de rigidité de l'élément plaque membrane.

```
FloatMatrix*  SHELLS :: computeStiffnessMatrixB ( )
  // Computes numerically the stiffness matrix of the receiver.
{
  int             i ;
  double          dV ;
  FloatMatrix   *b,*db,*d ;
  GaussPoint    *gp ;

  stiffnessMatrix = new FloatMatrix( ) ;
  for (i=0 ; i<4 ; i++) {
    gp = gaussPointArray[i] ;
    b  = this -> ComputeBmatrixAtB(gp) ;
    d  = this -> computeConstitutiveMatrixB( ) ;
    dV = this -> computeVolumeAround(gp) ;
    db = d -> Times(b) ;
    stiffnessMatrix -> plusProduct(b,db,dV) ;
   this -> giveRotationMatrix() ;

stiffnessMatrix -> rotatedWith(rotationMatrix) ;

    delete b ;
    delete db ;}
return stiffnessMatrix -> symmetrized( ) ;
}
```

Tableau 5.2. Méthode permettant le calcul de la matrice de rigidité flexionnelle de l'élément plaque membrane.

```
FloatMatrix*  SHELLS :: computeStiffnessMatrixS ( )
  // Computes numerically the stiffness matrix of the receiver.
{
  int             i ;
  double          dV ;
  FloatMatrix   *b,*db,*d ;
  GaussPoint    *gp  ;

  stiffnessMatrixS = new FloatMatrix( ) ;
  for (i=0 ; i<4 ; i++)  {
    gp = gaussPointArray[i] ;
    b  = this -> ComputeBmatrixAtS(gp) ;
    d  = this -> computeConstitutiveMatrixS( ) ;
    dV = this -> computeVolumeAround(gp) ;
    db = d -> Times(b) ;
    stiffnessMatrixS -> plusProduct(b,db,dV) ;
   this -> giveRotationMatrix() ;

stiffnessMatrixS -> rotatedWith(rotationMatrix) ;

    delete b ;
    delete db ;}
  return stiffnessMatrixS -> symmetrized( ) ;
}
```

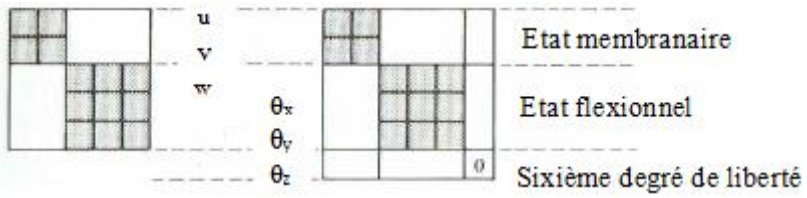
Tableau 5.3. Méthode permettant le calcul de la matrice de rigidité de cisaillement de l'élément plaque membrane.

```
FloatMatrix*  SHELLS :: computeStiffnessMatrixM ( )
  // Computes numerically the stiffness matrix of the receiver.
{
  int              i ;
  double           dV ;
  FloatMatrix  *b,*db,*d ;
  GaussPoint   *gp ;
  stiffnessMatrixM = new FloatMatrix( ) ;
  for (i=0 ; i<4 ; i++)
{
     gp = gaussPointArray[i] ;
     b  = this -> ComputeBmatrixAtM(gp) ;
     d  = this -> computeConstitutiveMatrixM() ;
     dV = this -> computeVolumeAround(gp) ;
     db = d -> Times(b) ;
     stiffnessMatrixM -> plusProduct(b,db,dV) ;

   this -> giveRotationMatrix() ;

    stiffnessMatrixM -> rotatedWith(rotationMatrix) ;
     delete b ;
     delete db ;
}
  return stiffnessMatrixM -> symmetrized( ) ;
}
```

Tableau 5.4. Méthode permettant le calcul de la matrice de rigidité membranaire de l'élément plaque membrane.

5.7. Sixième degré de liberté [39]

Dans les axes locaux de l'élément il y a cinq inconnues par nœud : un degré de liberté n'est pas utilisé, la rotation de la normale au plan de l'élément autours d'elle-même θ_z. Dans la matrice de rigidité $[K]$ d'un tel élément, le bloc matriciel du nœud 1est à l'image de la figure 5.4a.



(a) (b)

Figure 5.4. Aspect d'un bloc matriciel d'un élément fini plaque-membrane.

Pour pouvoir travailler en 3D, où il y a 6 degré de liberté nodaux dans le cas général, il faut faire une expansion de chaque bloc matriciel par l'adjonction d'une sixième ligne et d'une sixième colonne de zéros (figure 5.4b). Après rotation des axes locaux aux axes globaux, les zéros disparaissent et chaque bloc matriciel semble plein, mais il est évident qu'une des composantes de rotation reste linéairement dépendante des deux autres. C'est le problème du sixième degré de liberté qui se présente en pratique, comme suit.

Examinons le maillage d'une structure plissée (figure 5.5). Deux cas peuvent se produire :

- Tous les éléments entourant un nœud sont dans le même plan (nœuds A par exemple) ou ont un plan tangent commun : alors, en ce nœud, il n'y a pas de rigidité autours de la normale à ce plan et il n'y a que cinq inconnues ;
- Les éléments joignant un nœud ne sont pas tous dans un même plan (nœuds B) ; dans ce cas, il y a trois rotations indépendantes en un tel nœud, et six degré de liberté.

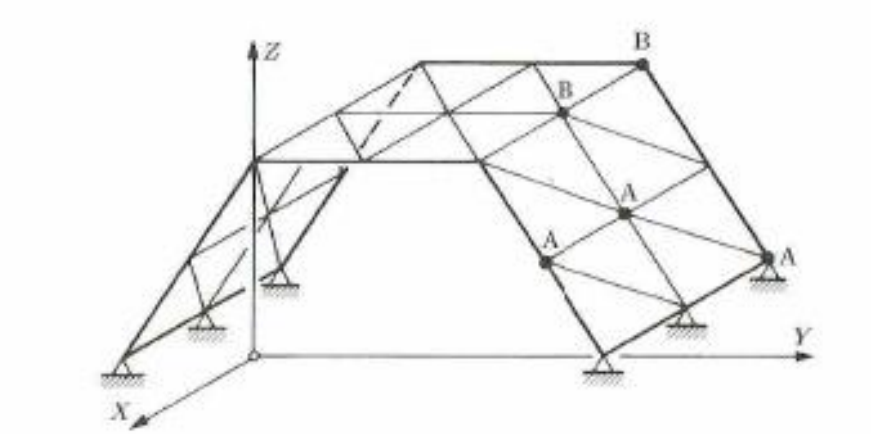


Figure 5.5. Nœuds situés ou non dans un même plan ; A : nœuds a cinq degrés de liberté ; B : nœuds à six degrés de liberté [39].

Dans le premier cas, le vecteur rotation se situe dans le plan du nœud et deux composantes suffisent à le définir ; dans le second cas, le vecteur rotation est quelconque (il est la résultante d'au moins deux vecteurs provenant de deux plans) et nécessite trois composantes pour être défini.

5.8. Conclusion

Ce chapitre permet aux lecteurs de bien comprendre la formulation élément finis des plaques et plaques membranes en se basant sur la théorie développée dans le chapitre précédant, et ce pour pouvoir appréhender l'implémentation de ces dits éléments.

Chapitre 6

Application de la méthode statique non-linéaire en utilisant un logiciel basé sur l'approche orientée objet

6.1. Introduction

Les logiciels de modélisation ont déjà subi de profondes modifications pour intégrer les évolutions des modèles et des méthodes numériques mais 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 que se confirmer ; la réutilisabilité devient de plus en plus difficile à obtenir.

6.2. 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 résolution ont été bouleversées : méthodes directes, méthodes itératives, méthodes multi-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 à savoir :

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

Par-dessus tout :

- ✓ Les Non linéarités apparaissent simultanément et donc interagissent.
- ✓ Les Imperfections précipitent le comportement non linéaire en compliquant d'avantage le calcul.

6.2.1. Orientation souhaitable pour la nouvelle génération de logiciels

- ✓ 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 industriels
- ✓ Un logiciel de recherche doit être dès 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 calculs déjà effectués.
- ✓ Il serait intéressant de classifier les modèles éléments finis dans le logiciel, C'est-à-dire définir des relations de parenté entre modèles ; on gagnera beaucoup en clarté dans l'organisation du code.
- ✓ On pourrait aussi écrire des procédures valables pour toute une classe d'élément donc gagné en réutilisabilité.

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.

6.2.2. Evolution des méthodes avec l'évolution du comportement des structures

Les codes de conception sismiques modernes permettent aux ingénieurs d'utiliser des analyses linéaires ou non linéaires pour calculer les forces et les déplacements de conception.

Eurocode 8, [29] contient notamment quatre méthodes d'analyse : analyse statique simplifiée, analyse modale spectrale, analyse statique non linéaire et analyse dynamique non linéaire voir tableau ci-dessous :

Action / Structure	Statique	Dynamique
Elastique Linéaire	Force de remplacement	Spectre de réponse
Non-linéaire	Poussée progressive (pushover)	Non-linéaire dynamique

Tableau 6.1. Résumé des méthodes et leurs domaines d'action.

Ces méthodes font référence à la conception et à l'analyse de structures à ossature, principalement des bâtiments et des ponts. Les deux méthodes non linéaires nécessitent des modèles avancés et des procédures non linéaires avancées pour être pleinement applicables par les ingénieurs de conception.

6.3. Analyse statique non linéaire (pushover) selon l'Eurocode 8 [29]

6.3.1. But et principe de l'analyse statique non-linéaire

L'analyse de poussée progressive est décrite en détail dans l'Eurocode 8 [29] ; push-over peut être utilisé pour vérifier la performance structurelle des bâtiments nouvellement conçus et des bâtiments existants. En particulier, l'analyse de poussée progressive peut être utilisée pour les buts suivants :

a- Evaluer les mécanismes plastiques attendus ainsi que la distribution de dommages ;

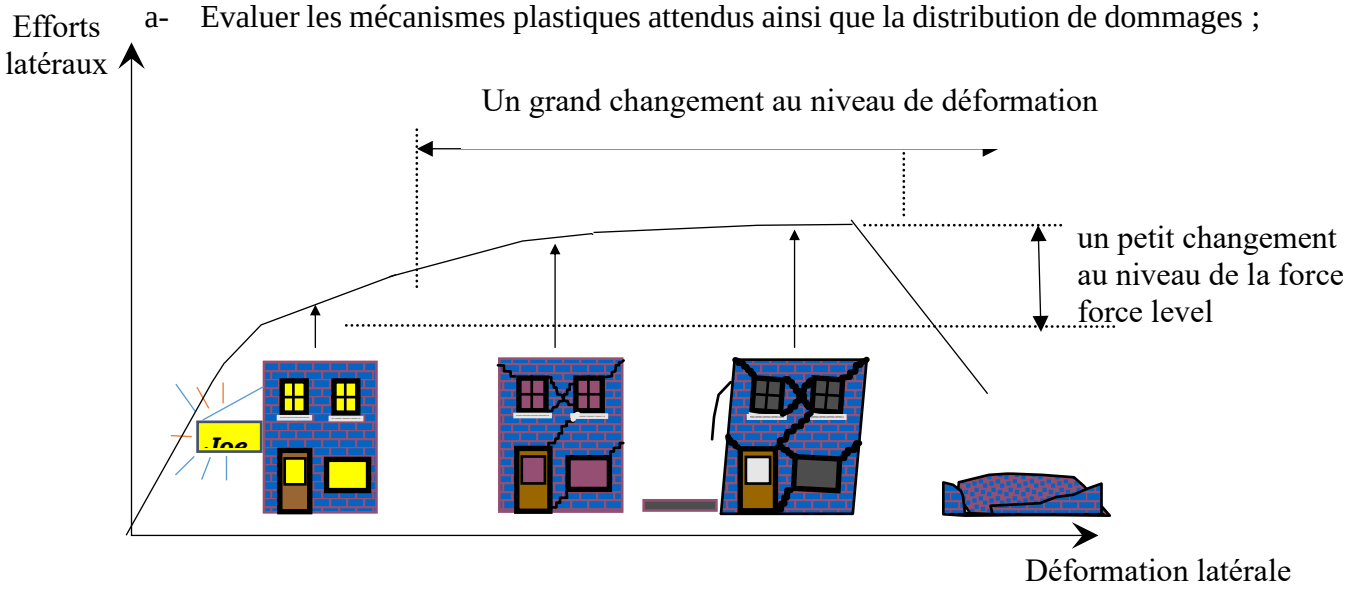


Figure 6.1. Distribution des dommages.

b- Evaluer la performance structurelle des bâtiments existants ou leurs rénovation ;

c- Comme une alternative à la conception basée sur l'analyse élastique-linéaire qui utilise le coefficient de comportement q. Dans ce cas, le déplacement cible à partir de l'analyse trouvé de poussée progressive doit être utilisé comme base pour la conception [3].

Aucune indication n'est donnée dans l'Eurocode 8 [29] ou dans la littérature publiée sur la façon de réaliser une analyse statique non-linéaire avec deux charges appliquées simultanément dans deux directions orthogonales, par conséquent, on suppose ici que dans une analyse de push-over la structure est poussée (pilotée) avec des charges appliquées dans une direction horizontale uniquement (la sismique vertical est généralement négligée dans les bâtiments).

Des indications sont données sur la façon de combiner les effets des actions appliquées séparément dans deux directions horizontales (l'Eurocode 8 [29] Partie 1 § 4.3.3.5). Pour les bâtiments conformes aux critères de régularité de l'Eurocode 8 [29], l'analyse peut être effectuée en utilisant deux modèles plans (2D), un pour chaque direction principale horizontale.

La méthode consiste à appliquer des formes de charge constante au modèle de construction. Ces formes de charge représentent les charges latérales appliquées par le mouvement du sol. L'intensité de charge est augmentée d'une manière pseudo-statique. Le modèle de structure peut être plan (2D) ou spatial (3D), en fonction des caractéristiques de régularité de l'immeuble. D'autre part, Le modèle de charge est toujours appliqué dans une seule direction.

L'analyse de la poussée progressive consiste à appliquer une charge latérale croissante de façon uniforme.

Étant donné que les méthodes non linéaires sont particulièrement intéressantes pour les bâtiments existants, qui sont rarement régulière, un modèle 3D est nécessaire dans la plupart des cas [70], [72], [74], [75].

La procédure push-over statique (non-linéaire) dans l'EC8 suit la méthode N2 développé par [31], [32], [33] de l'Université de Ljubljana, Slovénie. La méthode N2 a été développée en utilisant un modèle de bâtiment sous cisaillement (shear building model), à savoir un modèle en portique avec des planchers rigides dans leurs plans. En outre, le déplacement vertical est généralement négligé dans la méthode, et que les deux composantes du mouvement du sol horizontal, x et y, sont considérés.

On peut encore longer la vision en appliquant la méthode pushover sur des structures existantes avec prise en compte du sol (interaction sol-structure) [9], [16], [41].

La méthode N2 consiste à appliquer deux distributions de charge à la structure :

a- un modèle de charge "triangulaire" "modale", qui est une forme de la charge proportionnelle à la matrice de masse multiplié par le premier mode élastique $\mathbf{P}^1 = \mathbf{M}\boldsymbol{\Phi}_1$

b- un modèle «uniforme», qui est une forme de la charge proportionnelle à la masse $\mathbf{P}^2 = \mathbf{MR}$.

avec :

$\mathbf{M}$: est la matrice de masse ;

$\boldsymbol{\Phi}_1$: est le premier mode ;

R : vecteur ou la valeur unitaire 1 représente les degrés de libertés correspondant à l'application du mouvement du sol et la valeur 0 correspond aux autres ddl.

Dans la méthode N2 $\boldsymbol{\Phi}_1$ est normalisée de sorte que le déplacement du plancher du haut est 1, c'est à dire $\boldsymbol{\Phi}_{1,n} = 1$.

Les deux distributions de charges sont représentées schématiquement dans la figure 6.2. les charges latérales sont appliquées d'une manière croissante, et la réponse est tracer dans une courbe effort tranchant à la base en fonction du déplacement au sommet de la structure (par exemple au centre de masse du dernier plancher) , il s'agit de la courbe dite de poussée progressive ou bien courbe de capacité qui est aussi représentée dans la figure 6.2.

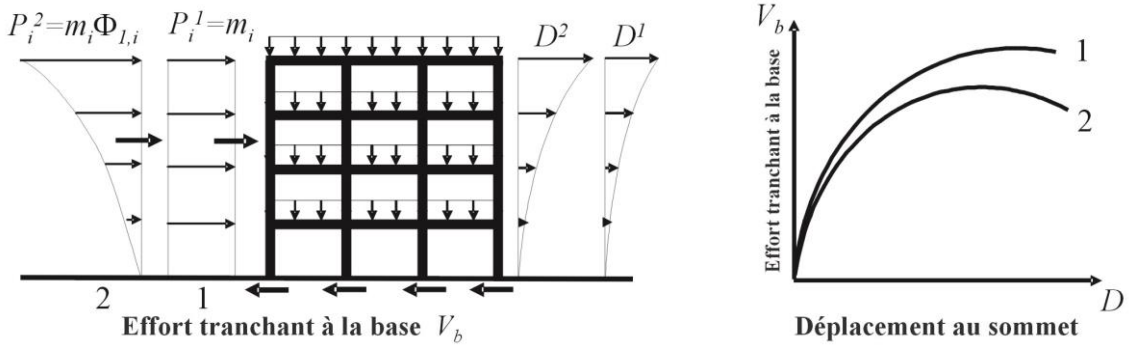


Figure 6.2 Distribution de charge pour l'analyse de poussée progressive selon l'Eurocode [29] et les courbes de capacité de réponse.

La procédure N2 transforme la réponse du système MDOF (plusieurs degrés de libertés) en une réponse d'un système SDOF équivalent (un seul degré de liberté). Cela est nécessaire afin de comparer la courbe de capacité du bâtiment de la figure 6.2 à la demande, exprimée dans les codes de conception par les spectres de calcul, qui se réfèrent à des systèmes à un seul degré de liberté (SDOF). La transformation MDOF–SDOF tel qu'elle a été développé par [31], [32], [33] est traitée plus en détails dans la section suivante.

6.3.2. Système à un seul degré de liberté équivalent et diagramme de capacité [6], [7], [8]

Le calcul théorique de la procédure de transformation est le suivant : Le point de départ est l'équation d'un système à plusieurs degrés de libertés (MDOF) soumis à un mouvement du sol à la base, l'équation de mouvement peut être écrite sous cette forme :

$$M\,\ddot{U} + F(U) = -M\,R\,a \tag{6.1}$$

Où l'amortissement est négligé,

M : est la matrice de masse (supposé diagonale dans la méthode N2),

U et **F** : sont des vecteurs de déplacement et de force internes respectivement,

R : est le vecteur d'influence,

a : est l'accélération du sol en fonction du temps (a = a(t)), a est donnée dans une seule direction seulement.

Dans le cas linéaire élastique : $F = KU$ (K est la matrice de rigidité), dans le cas non linéaire F dépend de l'histoire du déplacement.

Pour un mouvement du sol uni-directionnel par exemple la direction x, le vecteur d'influence R pend la valeur de 1 qui correspond au degré des libertés dans la direction x et prend la valeur 0 pour les autres degrés de libertés.

Pour le cadre de la figure 6.3 ci-dessous en considérant seulement 12 degrés de libertés des 4 étages (3 degrés de liberté par étage), $\mathbf{R}^T = \{\mathbf{1} \quad \mathbf{1} \quad \mathbf{1} \quad \mathbf{1}\}$, avec $\mathbf{1} = \{1 \quad 0 \quad 0\}$

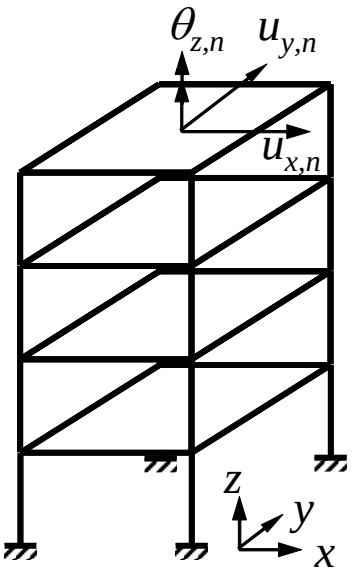


Figure 6.3. Degrés de libertés d'un portique en 3D.

La première supposition (approximation) de la méthode N2 est que le déplacement « U » a une forme constante qui ne changera pas durant la réponse à l'accélération du sol.

Le vecteur de déplacement de U est défini comme :

$$U = \phi\, D_t \tag{6.2}$$

Ou bien : $U(x,t) = \phi(x)\, D_t(t)$

Dans l'analyse de la poussée progressive des portiques avec des planchers rigides, le vecteur de charge latérale P est appliqué au centre de masse de chaque étage et il est déterminé comme suit :

$$P = p\, M\, \phi \tag{6.3}$$

La répartition de la charge latérale est liée à la forme du déplacement supposé ϕ (c'est la deuxième hypothèse de la procédure). Notez que la forme de déplacement ϕ (matrice modale) n'est nécessaire que pour la transformation du système de MDOF au système SDOF équivalent dans la procédure statique non linéaire.

D'après l'équation (6.3), il s'ensuit que dans un cadre la force latérale de cisaillement au i^{ème} étage est proportionnelle à la composante ϕ_i, pondérée par la masse d'étage m_i. Si le mouvement du sol est appliqué dans la direction x :

$$P_i = p m_i \phi_{x,i} \tag{6.4}$$

De la statique découle : $$P = F \tag{6.5}$$

En introduisant les équations (6.2), (6.3) et (6.5) dans l'équation (6.1) et après la multiplication par ϕ^t on trouve :

$$\phi^t\, M\, \phi\, \ddot{D}_t + \phi^t\, M\, \phi\, p = \phi^t\, M\, R a \tag{6.6}$$

Après la multiplication et la division de l'équation par $\phi^t\, M\, R$, le côté gauche de l'équation (6.6) on trouve :

$$\underbrace{\phi^t\, M\, R}_{m^*} \underbrace{\frac{\phi^t\, M\, \phi}{\phi^t\, M\, R}}_{\frac{1}{\Gamma}} \ddot{D}_t + \underbrace{\frac{\phi^t\, M\, \phi}{\phi^t\, M\, R}}_{\frac{1}{\Gamma}} \underbrace{\phi^t\, M\, R}_{V_d}\, p = \underbrace{\phi^t\, M\, R}_{m^*} a \tag{6.7}$$

Où : m^* est la masse équivalente du (SDOF) équivalent au bâtiment (MDOF).

$$m^* = \phi^t M R \tag{6.8}$$

Pour le bâtiment sous un tremblement de terre (mouvement du sol) appliqué dans la direction x :

$$m^* = \sum m_i \phi_{i,x} \tag{6.9}$$

La constante Γ contrôle la transformation d'un modèle à plusieurs dégrée de liberté (MDOF) vers un modèle à un seul dégrée de liberté (SDOF) et vice-versa.

$$\Gamma = \frac{\phi^t M R}{\phi^t M \phi} \tag{6.10}$$

Pour le bâtiment sous un tremblement de terre (mouvement du sol) appliqué dans la direction x :

$$\Gamma = \frac{\sum m_i \phi_{x,i}^i}{\sum m_i \phi_{x,i}^2} \tag{6.11}$$

Γ : est un facteur de participation modale, où ϕ est égale à la forme de un des modes du bâtiment,

V_b : est l'effort tranchant à la base du bâtiment MDOF dans la direction du mouvement du sol,

de l'équation (6.7) s'ensuit :

$$V_b = \phi^t M R p \tag{6.12}$$

Pour un bâtiment sous un mouvement du sol dans la direction x :

$$V_b = p\sum m_i \phi_{x,i} = \sum P_{x,i} \tag{6.13}$$

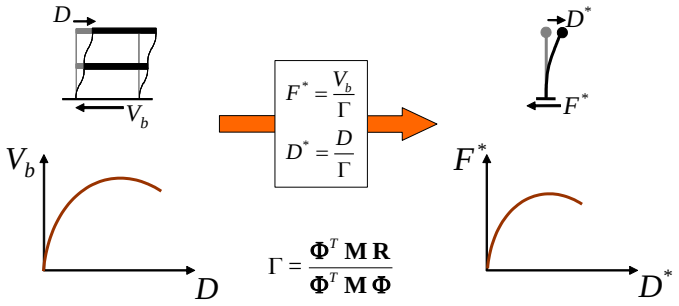


Figure 6.4. Transformation de la courbe de capacité en une réponse d'un système SDOF.

L'équation (6.7) ainsi devient l'équation du SDOF équivalent du bâtiment MDOF

$$m^* \ddot{D}^* + F^* = m^* a \tag{6.14}$$

$$D^* = \frac{D_t}{\Gamma} \quad \text{et} \quad F^* = \frac{V_b}{\Gamma} \tag{6.15}$$

Le calcul ci-dessus permet la transformation de la courbe de capacité du système MDOF de la figure 6.2, en une courbe de capacité du système SDOF équivalent, comme le représente la figure6.4.

La force et les déplacements sont à la fois mis à l'échelle par le même facteur Γ. A noter que le facteur de transformation Γ dépend de la forme du déplacement supposé, et il est ainsi différent selon le choix de ϕ. Deux formes de ϕ sont suggérées dans l'Eurocode [29].

Pour : $\Phi = \Phi_1$ $\Gamma = \dfrac{\phi^t MR}{\phi^t M\phi}$;

Pour : $\Phi = R$ $\Gamma = 1$

6.3.3. Linéarisation de la courbe de capacité et comparaison au spectre de demande

Afin de comparer la courbe de capacité à la courbe de demande donnée par le spectre de calcul :

• Premièrement, les courbes de poussée non linéaires du SDOF sont approximées par des courbes élastiques-parfaitement plastiques (ou bilinéaires).

Selon l'Eurocode [29], cette transformation peut être basée sur le principe d'égalité d'énergie. Une énergie égale est supposée entre les courbes de poussée bilinéaires et non linéaires. Cette procédure simple est illustrée à la figure 6.5.

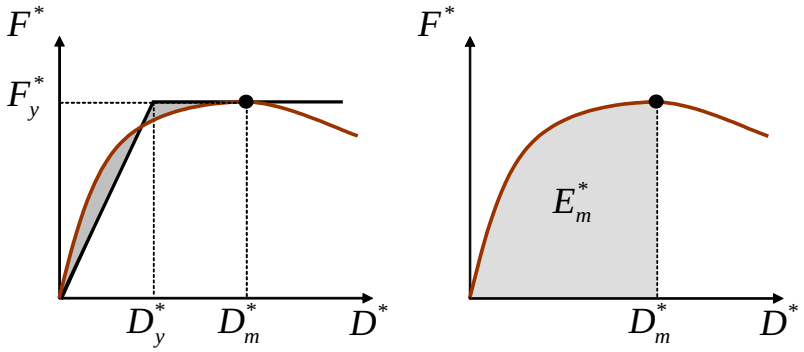


Figure 6.5. Bilinéarisation de la courbe de capacité du SDOF.

• Deuxièmement, la courbe de capacité est transformée en un spectre de capacité en normalisant la force par rapport au poids SDOF. Le spectre de capacité résultant est illustré à la figure 6.6.

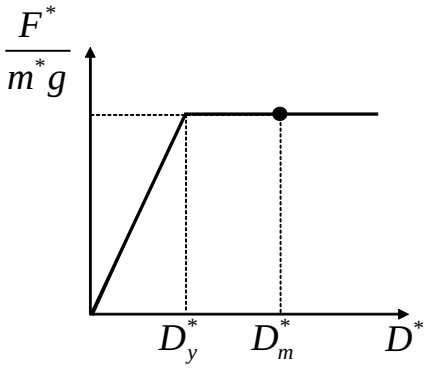


Figure 6.6. Spectre de capacité en SDOF.

La demande sur le bâtiment est donnée par le spectre de conception fourni par les codes de conception. Afin de comparer la capacité et la demande, la première étape consiste à

transformer le format du spectre de conception du format classique Accélération (A) vs Période T au format ADRS, c'est-à-dire Accélération (A) vs Déplacement (D). La procédure est assez simple, comme l'accélération et le déplacement sont liés par :

$$S_D = \left(\frac{T}{2\pi}\right)^2 S_A \tag{6.16}$$

La transformation vers le spectre ADRS est illustrée à la figure 6.7 :

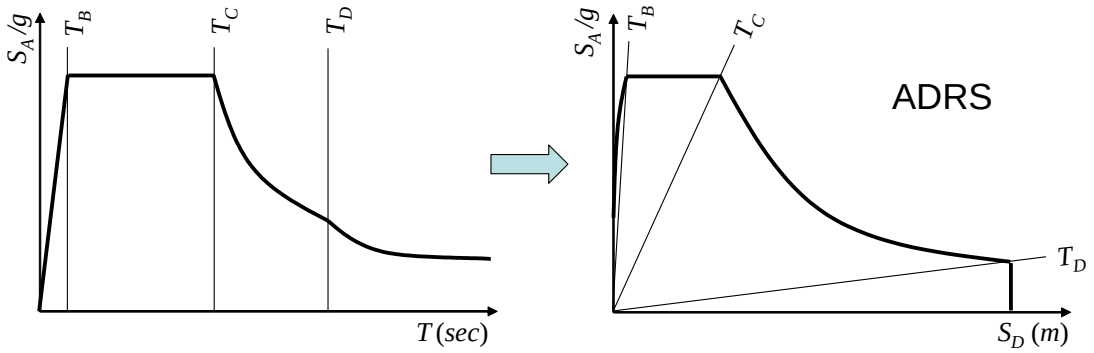


Figure 6.7. Transformation du spectre en format ADRS.

Le spectre de capacité de la figure 6.6 est maintenant comparé au spectre de demande ADRS de la figure 6.7. La comparaison n'est pas immédiate, car le spectre de capacité est non linéaire, tandis que le spectre ADRS donné par les codes de conception est linéaire.

Pour un système SDOF à comportement plastique bilinéaire, le spectre d'accélération S_A et le spectre de déplacement S_D peuvent être déterminés comme :

$$S_A = \frac{S_{Ae}}{R_\mu} \tag{6.17}$$

$$S_D = \frac{\mu}{R_\mu} S_{De} = \frac{\mu}{R_\mu} \frac{T^2}{4\pi^2} S_{Ae} = \mu \frac{T^2}{4\pi^2} S_{Ae} \tag{6.18}$$

où :

e : l'indice d'élasticité,

$\boldsymbol{\mu}$: est le facteur de ductilité = déplacement inélastique maximum / déplacement élastique,

$\boldsymbol{R_\mu}$: est le facteur de réduction dû à la ductilité.

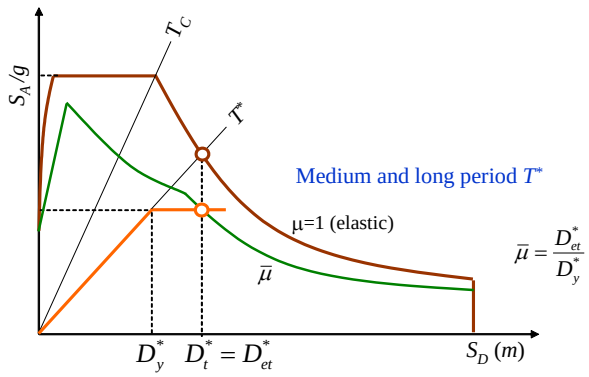


Figure 6.8. Spectres de capacité et de demande et détermination du déplacement cible pour SDOF.

6.4. Conclusion

Ce chapitre lève le voile sur trois points essentiels à savoir : l'évolution des techniques numériques et des modèles physiques, l'orientation souhaitable pour la nouvelle génération des logiciels ; (le centre d'intérêt de la thèse est l'utilisation d'un logiciel écris à base de la notion d'objet), et enfin la théorie derrière la méthode statique non linéaire.

Chapitre 7
Méthode des éléments finis étendus (XFEM)

7.1. Introduction

Les méthodes des éléments finis ont été largement utilisées depuis leur apparition au début des années 1960. Cette popularité est due à leur capacité à traiter de nombreux problèmes en raison de leur robustesse. La méthode a été utilisée avec grand succès dans des domaines tels que la mécanique des solides, la mécanique des fluides, le transfert de chaleur, l'électromagnétisme, etc.

Malgré ce succès, il existe plusieurs domaines dans lesquels les méthodes actuelles par éléments finis sont moins que l'idéal. En raison du fait que les méthodes standard d'éléments finis sont basées sur des approximations polynomiales, elles ne sont pas bien adaptées aux problèmes de discontinuités. En réponse à cette lacune des méthodes éléments finis standard, des éléments finis étendus ont été développés.

Avec des espaces d'approximation enrichis, X-FEM est capable de reproduire les caractéristiques problématiques, c'est-à-dire les discontinuités, et de manière spectaculaire.

Des résultats prouvés sont obtenus. Récemment, [10], [59], ont proposé une méthodologie systématique pour incorporer des fonctions arbitraires dans l'espace d'approximation en éléments finis.

Depuis sa première apparition [10], [59], La méthode des éléments finis étendue (X-FEM) a été appliquée avec succès à de nombreux problèmes de mécanique tels que les problèmes de discontinuité en statiques bidimensionnels, problème d'évolution de la propagation des fissures ainsi que l'extension aux trois dimensions qui a été présenté par [71].

La méthode des éléments finis étendue (X-FEM) est utilisée pour simuler la croissance des fissures. Une bibliothèque C ++ pour X-FEM est implémentée qui permet une extension facile pour le développement ultérieur de la méthode. Dans X-FEM, l'espace d'approximation élément finis standard est enrichi avec des fonctions spécialement conçues pour aider à saisir les caractéristiques complexes d'un problème. Les fonctions d'enrichissement sont nécessaires pour modéliser les discontinuités dans le champ [12], [28], [68].

Le principal avantage de l'adoption d'une approche orientée objet est que l'extension du programme est plus simple et plus naturel, car les nouvelles applications ont peu d'effet sur le code existant. Ainsi, le code est réutilisé. De plus, en le comparant à la programmation classique on remarque que le code orienté objet est mieux organisée.

7.2. Avantage de la programmation orientée objet et les éléments finis étendus (XFEM)

L'application de concepts orientés objet à la programmation en éléments finis a fait l'objet d'une grande attention ces dernières années [36], [37], [56]. Le principal avantage d'adopter une approche orientée objet est que l'extension du programme est plus simple et naturel, car les nouvelles implémentations ont peu d'impact sur le code existant. Ainsi, la réutilisation du code est maximisée. De plus, par rapport aux programmations classique et structurée, l'utilisation de la programmation orientée objet à une intégration plus étroite entre la théorie et la mise en œuvre informatique. La programmation orientée objet est particulièrement utile dans le développement de programmes volumineux et complexes, comme les codes à éléments finis sans fin destinés à gérer différents types d'éléments, modèles constitutifs, algorithmes d'analyse et fonction enrichie etc.

Les langages de programmation orientés objet ont été présentée comme outils robustes pour développer une variété d'applications. La fiabilité et la facilité de la réutilisation de l'approche orientée objet ont généré des logiciels qui ont donné naissance à plusieurs paquets d'éléments finit orientés objet. Par conséquent, il est naturel d'utiliser cette notion pour la mise en œuvre de la méthode des éléments finis étendu X-FEM qui est applicable dans le domaine de la mécanique linéaires de la rupture élastiques [1], [30].

Conjugué cette manière de programmer en utilisant le langage C++ avec la méthode des éléments finis étendu X-FEM était notre centre d'intérêt.

L'idée centrale de ce chapitre est d'utiliser une bibliothèque C ++ flexible, nommée OpenXFEM ++. Cette bibliothèque a été construite sur la base de [35] un logiciel fini orienté objet.

Le code actuel est utilisé pour résoudre divers problèmes liés à la mécanique de la rupture élastique linéaire, notamment le calcul des paramètres de fracture et la simulation de la croissance des fissures [12], [60], [63]. Avec l'approche orientée objet, OpenXFEM ++ est un code informatique extensible et facile à gérer, on arrive à implémenter la résolution de nouveaux problèmes sans obstacles majeurs.

7.3. Mécanique de rupture

La fissuration se manifeste par la séparation irréversible d'un milieu continu en deux parties, appelées lèvres de la fissure, ce qui introduit une discontinuité. Les mouvements possibles des lèvres de chaque fissure sont des combinaisons de trois modes indépendants, comme illustré dans la figure 7.1. :

- Mode d'ouverture (mode I).
- Mode coulissant ou cisaillant (mode II).
- Mode déchirement (mode III).

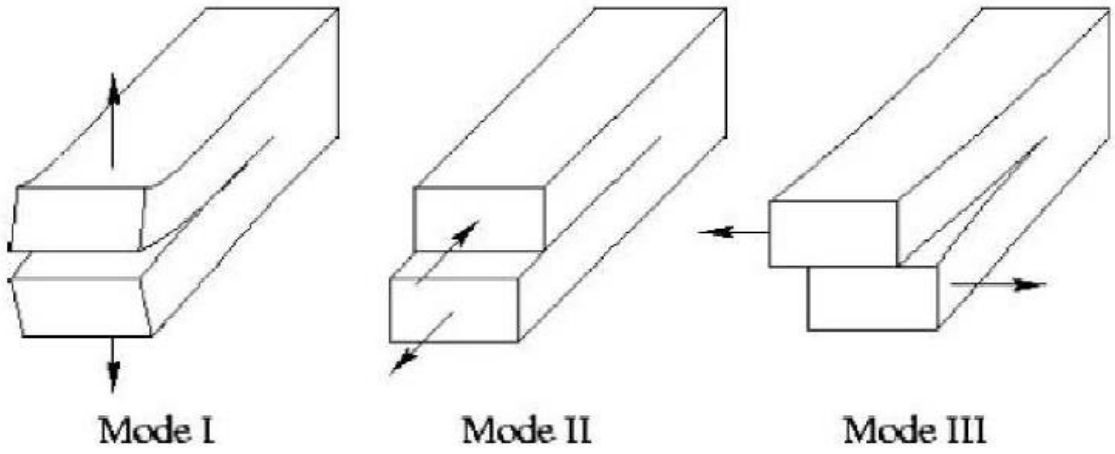


Figure 7.1. Modes d'ouverture des fissures.

Les champs de contrainte, de déformation et de déplacement de la fissure peuvent être représentés sous forme de combinaison linéaire pour chaque mode individuel via des fonctions de contrainte appropriées [53], on peut développer des expressions pour chacun des trois modes présentés ci-dessus.

Pour chaque mode de chargement, la magnitude du champ de contrainte est définie par un coefficient scalaire appelé le facteur d'intensité de contrainte. Il y a un facteur d'intensité de contrainte pour chaque mode de chargement que nous appellerons : K_I, K_{II} et K_{III} pour les modes : I, II et III respectivement [50], [51]. Une fissure est dite chargée en mode mixte lorsque plusieurs facteurs d'intensité de contrainte sont nécessaires pour représenter les champs de la pointe de fissure. (Peut faire l'objet d'un travail futur).

$$K_I = (2\pi r)^{\frac{1}{2}} \lim_{r\to 0} \sigma_{yy}(r,0)$$

$$K_{II} = (2\pi r)^{\frac{1}{2}} \lim_{r\to 0} \sigma_{xy}(r,0) \qquad (7.1)$$

$$K_{III} = (2\pi r)^{\frac{1}{2}} \lim_{r\to 0} \sigma_{zy}(r,0)$$

Pour de plus amples informations de la provenance de l'équation (7.1) voir [23], [53]. Le champ de déplacement approximatif pour la modélisation en 2D est donné par l'expression suivante :

$$u^h(x) = \sum_{I\in \mathrm{N}} N_I(x)u_I + \sum_{J\in \mathrm{N}_{er}} \tilde{N}_J(x)(H(x)-H(x_J))a_J + \sum_{K\in N_{tip}} \tilde{N}_K(x)\sum_{\alpha=1}^{4}(B_\alpha(x)-B_\alpha(x_K))b_{\alpha K} \quad (7.2)$$

N : est l'ensemble des nœuds du maillage,

N_{er} : est l'ensemble des nœuds enrichis par la discontinuité (l'ensemble des nœuds appartenant aux éléments divisées par la fissure),

N_{tip} : est l'ensemble des nœuds enrichis par branch function (l'ensemble des nœuds appartenant à l'élément contenant la pointe de la fissure),

u_I : Degrés de libertés classiques pour le nœud i,

a_J : Degrés de libertés supplémentaires pour la fonction de discontinuité,

$b_{\alpha K}$: Degrés de libertés supplémentaires pour branch function,

$N_I(x), \tilde{N}_J(x), \tilde{N}_K(x)$: Les fonctions de forme standard d'élément fini.

$H(x)$: heaviside function qui prend les valeurs :

$$H(x) = \begin{cases} +1 & au-dessus\ \ de\ \ la\ \ fissure \\ -1 & au-dessous\ \ de\ \ la\ \ fissure \end{cases}$$

$B_\alpha(x)$: Branch function

Les fonctions d'enrichissement de la fissure pour un matériau isotopique élastique sont données par :

$$B = [B_1, B_2, B_3, B_4] = \left[\sqrt{r}\sin\frac{\theta}{2}, \sqrt{r}\cos\frac{\theta}{2}, \sqrt{r}\sin\frac{\theta}{2}\cos\theta, \sqrt{r}\cos\frac{\theta}{2}\cos\theta \right] \qquad (7.3)$$

Où : r et θ : sont les coordonnées polaires dans le système des coordonnées locales du fond de la fissure.

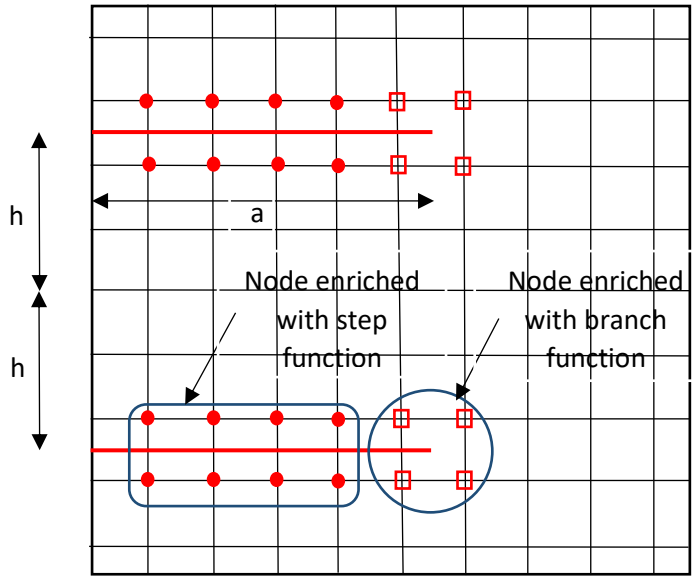


Figure 7.2. Discrétisation en XFEM autour du font de la fissure.

L'équation qui gouverne dans le cas statique est :

$$KU = F \tag{7.4}$$

Où U : représente les vecteurs des inconnues nodales, et K et F représentent respectivement la matrice de rigidité globale et les vecteurs de forces externes.

$$K^e = \int_{\Omega^h} \overline{B}^t C \overline{B} d\Omega \tag{7.5}$$

$$\overline{B} = \left[B_I^U \quad B_J^a \quad B_K^{b1} \quad B_K^{b2} \quad B_K^{b3} \quad B_K^{b3} \right]$$

Avec :

$$B_I^U = \begin{bmatrix} N_{I,x} & 0 \\ 0 & N_{I,y} \\ N_{I,y} & N_{I,x} \end{bmatrix} \qquad B_j^a = \begin{bmatrix} (\widehat{N}_J(H - H(x_J)))_{,x} & 0 \\ 0 & (\widehat{N}_J(H - H(x_J)))_{,y} \\ (\widehat{N}_J(H - H(x_J)))_{,y} & (\widehat{N}_J(H - H(x_J)))_{,x} \end{bmatrix} \tag{7.6}$$

$$B_k^{bl} = \begin{bmatrix} (\widehat{N}_K(B_K^l - B_K^l(x_K)))_{,x} & 0 \\ 0 & (\widehat{N}_K(B_K^l - B_K^l(x_K)))_{,y} \\ (\widehat{N}_K(B_K^l - B_K^l(x_K)))_{,y} & (\widehat{N}_K(B_K^l - B_K^l(x_K)))_{,x} \end{bmatrix}$$

Le vecteur de force élémentaire F^e est :

$$f^e = \left\{ f_I^U, f_J^a, f_K^{b1}, f_K^{b2}, f_K^{b3}, f_K^{b4} \right\} \tag{7.7}$$

$$f_I^U = \int_\Gamma N_I \bar{t} d\Gamma + \int_\Omega N_I b d\Omega$$

$$f_J^a = \int_\Gamma \widehat{N}_J(H - H(x_J))\bar{t} d\Gamma + \int_\Omega \widehat{N}_J(H - H(x_J)) b d\Omega \tag{7.8}$$

$$f_K^{bl} = \int_\Gamma \widehat{N}_K(B_K^l - B_K^l(x_K))\bar{t} d\Gamma + \int_\Omega \widehat{N}_K(B_K^l - B_K^l(x_K)) b d\Omega$$

7.4. Conclusion

Ce chapitre a pour but de montrer les limites de la méthode des éléments finis, et en même temps d'expliquer les principes de la méthode venant à la rescousse de la MEF, celle-ci n'est autre que la méthode des éléments fins étendus, qui elle aussi s'apprête bien à être implémenter en utilisant l'approche orientée objet.

Chapitre 8

Validation des résultats

8.1. Introduction

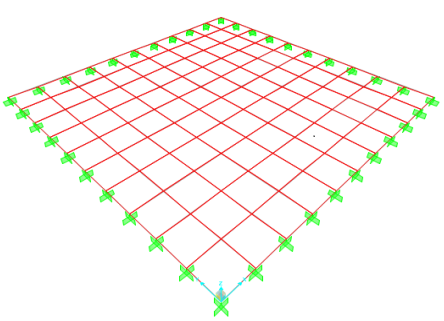
Dans le but de valider et de donner plus de fiabilité à la démarche élaborée dans cette thèse, nous avons procédé à une série de tests numériques, de divers problèmes en génie civil qui se divisent en trois grandes sections à savoir :

- Eléments finis des plaques membranes,
- Evaluation du bâtis existant, en utilisant la méthode statique non linéaire,
- Modélisation des fissures en utilisant la méthode des éléments finis étendus.

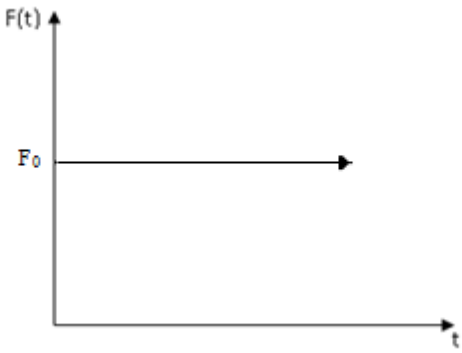
Les résultats obtenus de ces dits problèmes sont assez exhaustifs, et ils sont comparés à des résultats numériques, ou analytiques déjà trouvés dans la littérature.

8.2. Validation de l'élément plaque membrane implémenté dans un code orienté objet en C++

Validation numérique d'une plaque membranaire qui a les caractéristiques suivantes :



Data
E=210 × 10⁹ N/m²
a =2m
h = 0.03m
ν =0.3
ρ=7800 kg/m³
γ = 0.5 β=0.25 (newmark)
DT=0.0025 s
F = 800 N



$$F(t) = \begin{cases} F_0 \ pour & t \geq 0 \\ 0 \ pour & t < 0 \end{cases}$$

Figure 8.1. Plaque membrane carrée simplement appuyée sur ses quatre cotés.

Les résultats dynamiques donnés par le code FEMOBJ (élément finis orientée objet) d'un élément plaque membrane simplement appuyé sur ces quatre côtés ; sont comparés à ceux donnés par le logiciel Sap2000, comme le montre la figure 8.2., cette dernière laisse apparaitre une convergence quasi parfaite.

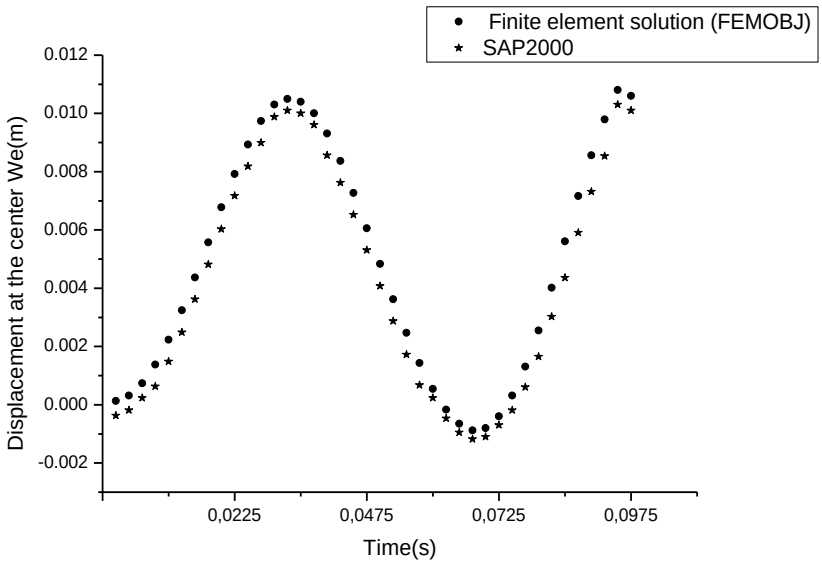


Figure 8.2. Réponse dynamique en élément fini pour une plaque membrane isotrope simplement appuyée sur ses quatre cotés.

La figure 8.3. montre clairement l'effet du type d'appuis sur la réponse de la structure :

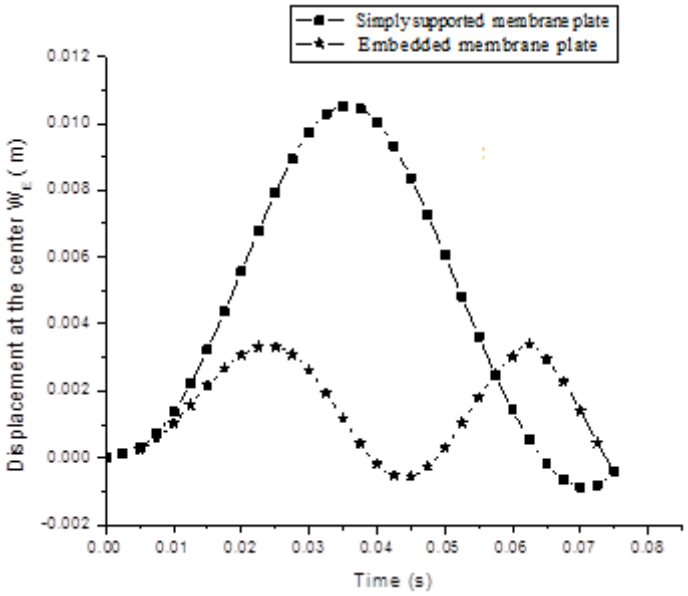


Figure 8.3. Influence du type d'appuis sur la réponse dynamique d'une plaque membrane carrée en utilisant un code C++ orientée objet.

8.3. Modélisation Numérique en utilisant la méthode de flexibilité

Avant de se lancer dans la modélisation d'un bâtiment réel 3D en utilisant la méthode de flexibilité, qui fera l'objet de la section 8.3.4, on a jugé impératif de montrer l'efficacité de la formulation force (méthode de flexibilité ou élément de Spacone) en la comparant à celle basée sur les déplacements. Cette comparaison est effectuer apriori sur des éléments simple à savoir :

- ✓ Système élémentaire (SDOF) modélisé dans le cas lineaire et non lineaire [11] ;
- ✓ Poutre en porte à faux sous charge concentrée ;
- ✓ Cadre (portique) en 2D sous un séisme d'El Centro.

Le principal avantage de la deuxième formulation basée sur les forces, est qu'elle est «exacte». Cela implique (élément de spacone), la non nécessité de discrétiser la structure, entraînant ainsi une réduction du nombre global de degrés de liberté. La théorie complète de l'élément basé sur la force se trouve dans [2], [9], [18], [46], [62], [70], [73].

8.3.1. Problème de test SDOF

8.3.1.1. Cas linéaire

On considère un cas de charge échelon ($F(t) = 0 \ \ pour\ t < 0\ et\ F(t) = F_0\ pour\ t \geq 0$) appliqué à l'oscillateur élémentaire (SDOF) linéaire [11]. L'oscillateur simple, présente les caractéristiques suivantes :

rigidité : K =1 N/m, masse : M = 1 kg, force : $F_0 = 1\ N$.

L'équation du mouvement se présente comme suit :

$$M\ddot{u} + Ku = F_0$$

La solution exacte est : $u(t) = u_{st}\ (1 - \cos \omega t\ )$.

avec : $u_{st} = F_0/K$ Pour ce cas le déplacement maximum est : $u_{max} = 2\ m$.

La figure 8.4. compare les résultats obtenus avec l'élément à base des déplacements et l'élément à base de force (la flexibilité) avec ce de [11]. Comme prévu, la solution exacte est obtenue en utilisant un seul élément dans les deux cas. L'algorithme de Newmark [64] avec : γ = 0,5 et β = 0,25 est utilisé.

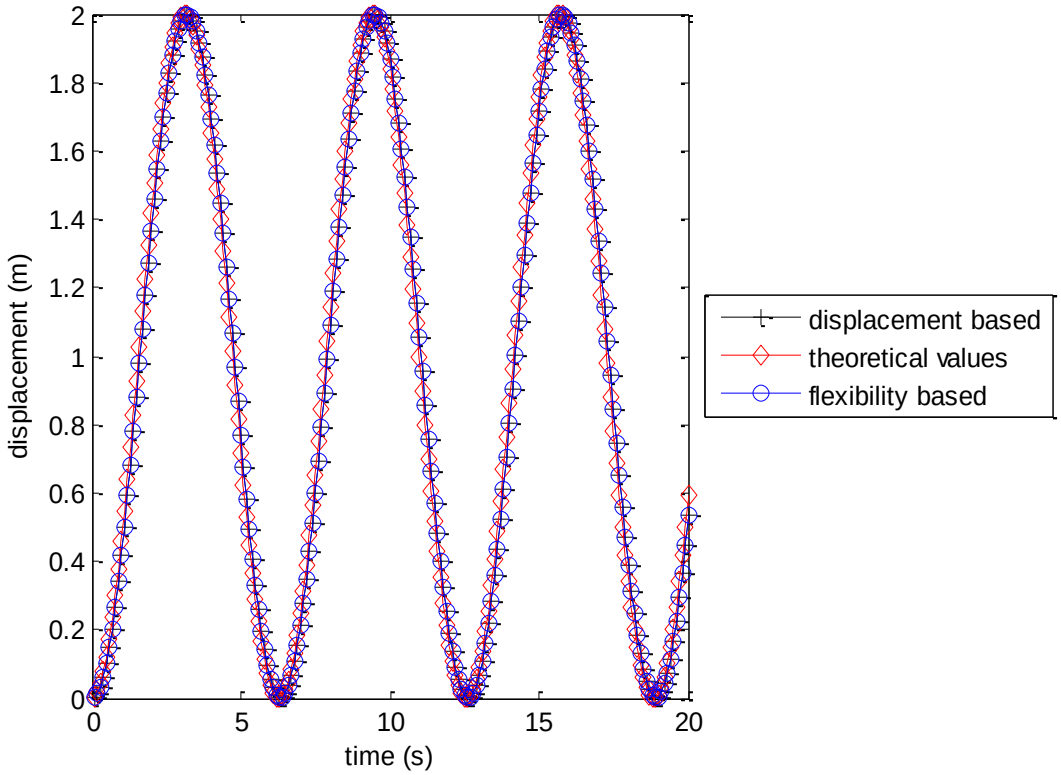


Figure 8.4. Comparaison entre la formulation à base des déplacements et celle à base de forces dans l'évolution des déplacements en temps, Δt = 0.1.

8.3.1.2. Cas non linéaire

Le même oscillateur (SDOF) est étudié mais cette fois ci avec un comportement non linéaire élasto-plastique. La non-linéarité est caractérisée par une limite d'élasticité σ_{yield} = 1,5 N/m^2. L'équation du mouvement non-linéaire est la suivante :

$$Mu\ddot{} + F(u) = F_0$$

Les figures 8.5 et 8.6 comparent les résultats obtenus avec l'élément à base des déplacements et l'élément à base de force (la flexibilité) avec ce de [11].

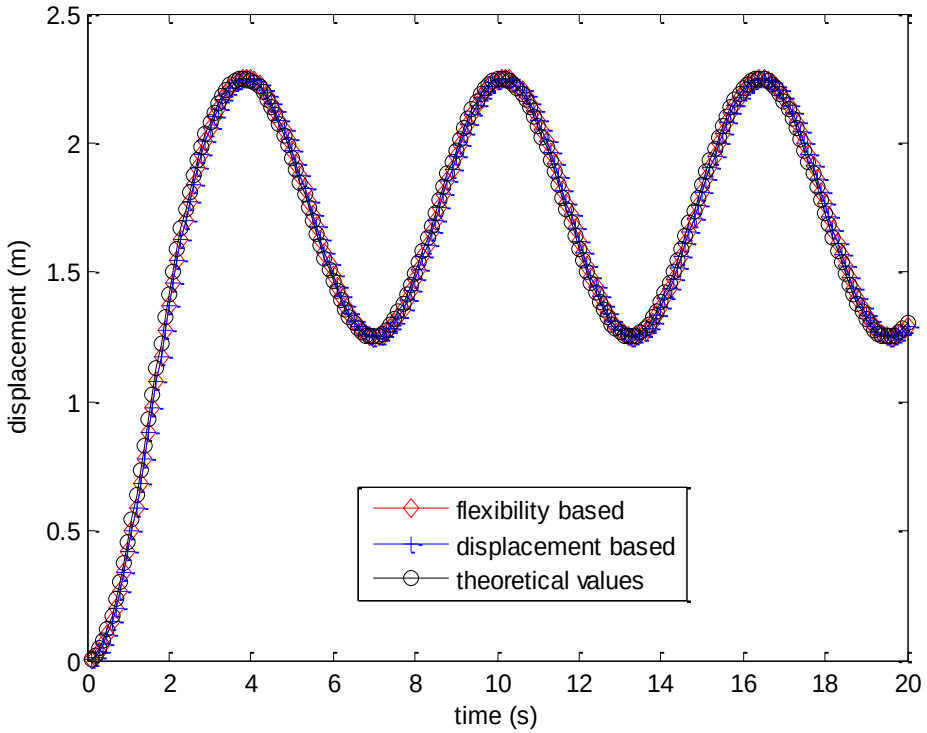


Figure 8.5. Réponse en termes de déplacements pour l'exemple de Biggs, cas non-linéaire $\Delta t = 0.1$.

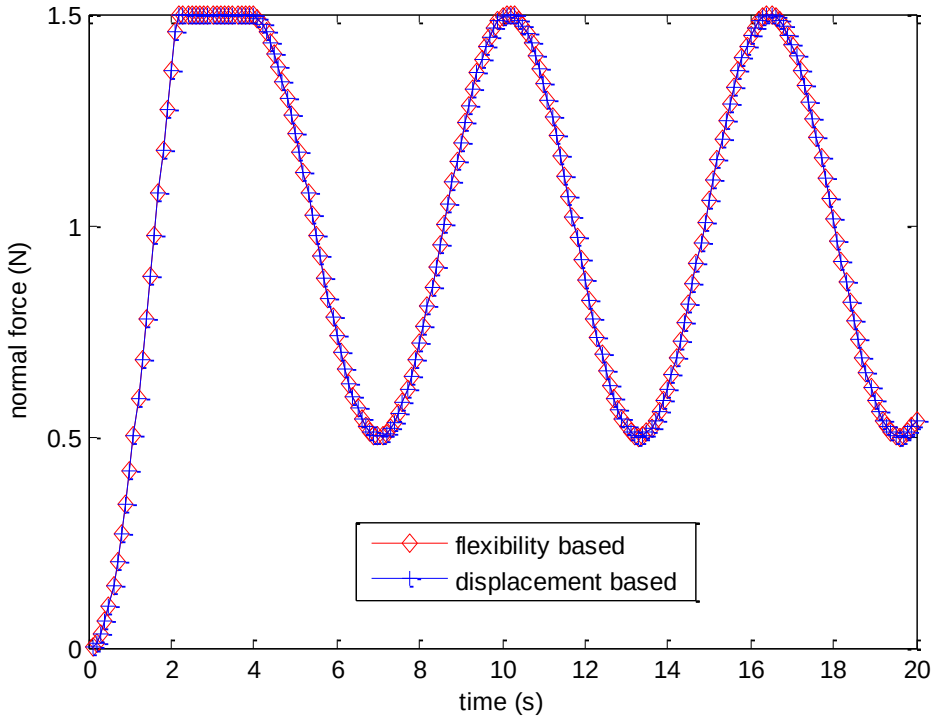


Figure 8.6. Réponse en termes de force pour l'exemple de Biggs, cas non-linéaire $\Delta t = 0.1$.

8.3.2. Chargement en flexion [7], [9]

Une poutre en porte-à-faux est utilisée dans ce deuxième exemple. Cette dernière présente les caractéristiques suivantes :

L=1 m, section rectangulaire : $b = h = 0{,}3\,m$, $E = 493{,}83\ KN/m^2$, $\nu = 0{,}16$, $f_t = f_c = 3000\ KN/m^2$.

La poutre en porte-à-faux a une masse forfaitaire de 750 kg et est chargée à l'extrémité libre d'une charge transversale de 15 N, appliquée à l'instant t = 0 (figures 8.7 et figure 8.8). La réponse en porte-à-faux en termes de déplacement de la pointe (dans la direction transversale) est illustrée dans la figure 8.9.

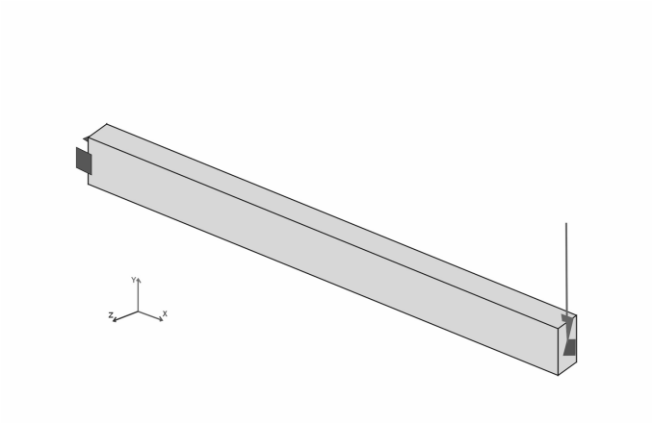


Figure 8.7. Modèle d'une poutre en porte-à-faux discrétisée avec un seul élément (flexibilité).

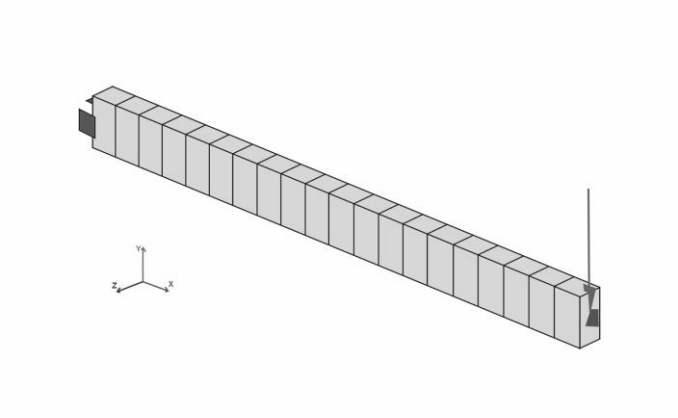


Figure 8.8. Modèle d'une poutre en porte-à-faux discrétisée avec 20 éléments (déplacement).

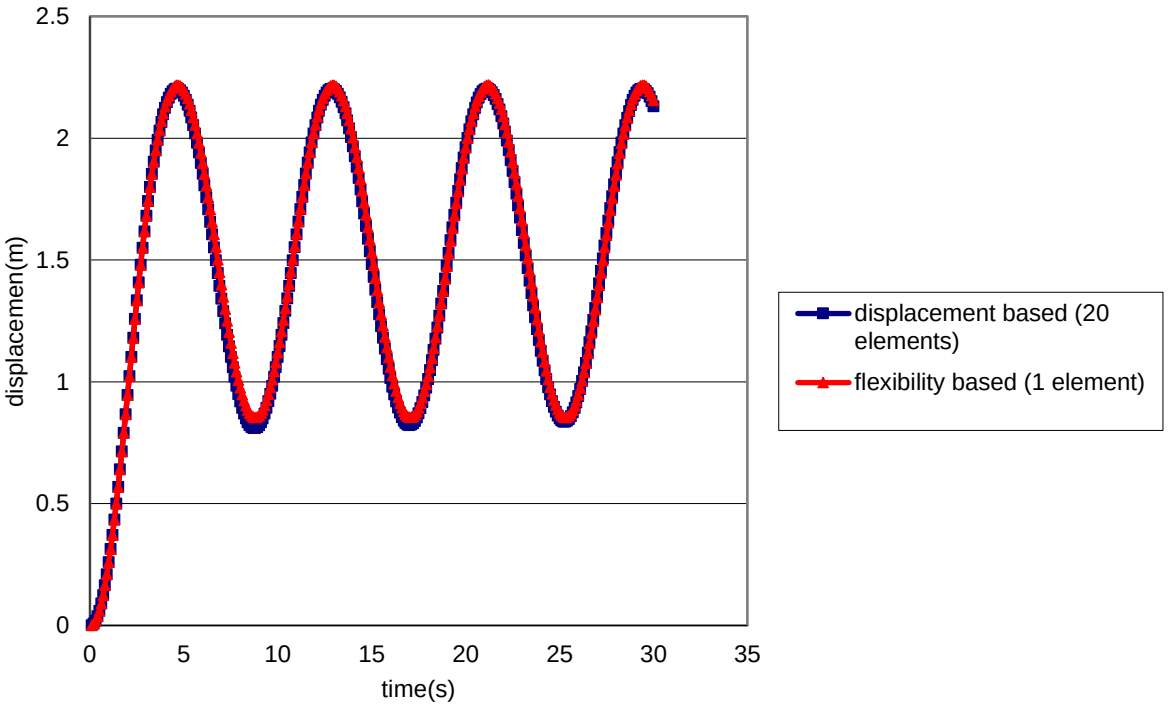


Figure 8.9. Réponse à la pointe extrême de la poutre en porte-à-faux Δt = 0.1.

Après avoir étudié ces deux exemples(cas SDOF et cas de la poutre en flexion), nous pouvons en déduire qu'il existe une nette convergence entre les valeurs données par la théorie, que ce soit en cas linéaire ou non linéaire, et celles du logiciel utilisé, pour le cas SDOF. Même constat pour le cas de la poutre en flexion où une nette convergence apparait entre l'élément fini basé sur la flexibilité et celui basé sur les déplacements.

8.3.3. Réponse d'un bâtiment à deux étages à une accélération du sol [7], [9], [13], [52]

La réponse d'un portique en 2D fait l'objet d'étude. Le portique latéral analysé est montré à la Figure 8.10.

Figure 8.10. Modèle 2D poteau poutre.

La réponse du portique est montrée dans la figure suivante (Figure 8.11) en utilisant l'élément à base de force (flexibilité) et celui à base des déplacements :

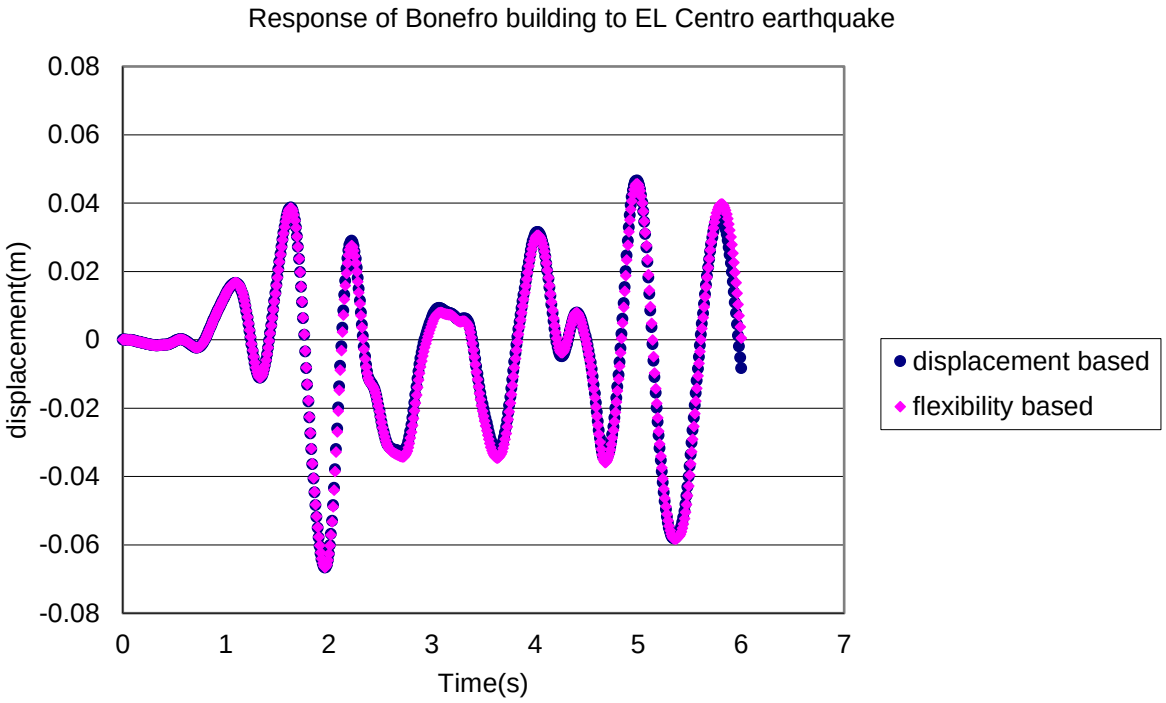


Figure 8.11. Réponse au niveau du dernier étage au séisme El Centro.

La Figure 8.11 illustre la réponse du plancher (sommet) en model 2D, au tremblement de terre d'El Centro à l'aide d'un élément basé sur à la fois la formulation déplacement et force (flexibilité). D'après la figure 8.11, les résultats sont concluants car l'élément à base des déplacements et celui à base de flexibilité coïncide parfaitement, chose qui va nous permettre de suivre notre étude qu'avec la deuxième formulation (flexibilité).

Les résultats sont concluants car après l'étude de ces trois premiers exemples l'élément en question (élément de spacone) [70] offre des avantages significatifs par rapport aux approches existantes basées sur les déplacements, car aucune erreur de discrétisation ne se produit et toutes les équations sont parfaitement satisfaites. Par conséquent, moins d'éléments sont nécessaires pour obtenir des résultats d'une précision comparable.

8.3.4. Modélisation d'un bâtiment réel en utilisant la méthode de flexibilité

8.3.4.1. Propriété des matériaux

La deuxième partie à valider est une structure modélisée par un logiciel développé à base de l'approche orienté objet, le cas en question est un bâtiment existant modéliser en 3D en utilisant l'élément de Spacone (élément de flexibilité) voir figures 8.12 et 8.13 :

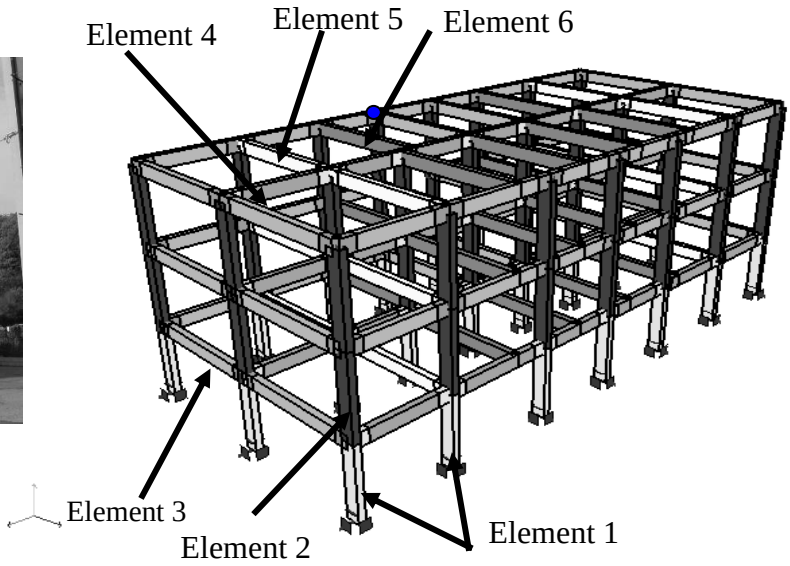


Figure 8.12. Bâtiment à deux étages utilisé dans l'analyse. **Figure 8.13.** Modèle 3D.

Un modèle 3D est présenté dans la figure 8.13, les propriétés des matériaux suivantes, ainsi que le détail d'armatures utilisées sont résumées dans ce qui suit (voir tableau 8.1).

Béton: $E = 2.7\ e7\ KN/m^2$
υ=0.16
f_t =300 KN/m^2
f_c =30000 KN/m^2

Acier : $E = 2.1\ e8\ KN/m^2$
υ=0.2
$f_t = f_c$ =500000 KN/m^2

Tableau 8.1. Détails de ferraillages des éléments.

elements & caracteristics	b(m)	h(m)	Reinforcement
element 1	0.3	0.3	2Φ16 top; 2Φ16 bottom; 2 Φ 14 middle
element 2	0.3	0.3	2Φ16 top ; 2Φ16 bottom
element 3	0.3	0.5	2Φ14 top ; 4Φ14 bottom
element 4	0.8	0.2	6Φ14 top ; 6Φ14 bottom
element 5	0.5	0.2	2Φ14 top ; 2Φ14 bottom
element 6	0.3	0.2	2Φ14 top ; 2Φ14 bottom

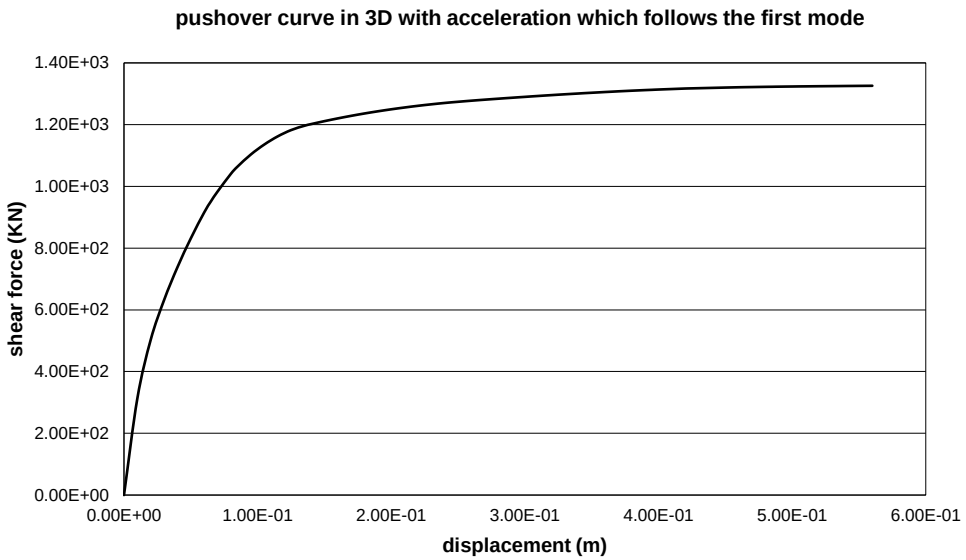


Figure 8.14. Réponse Pushover du modèle 3D sous distribution de charge modale selon la direction x.

Après avoir convertie la courbe de capacité montrée dans la figure 8.14 en une courbe en format ADRS puis la superposée avec le spectre de demande du même format on aura la figure 8.15 :

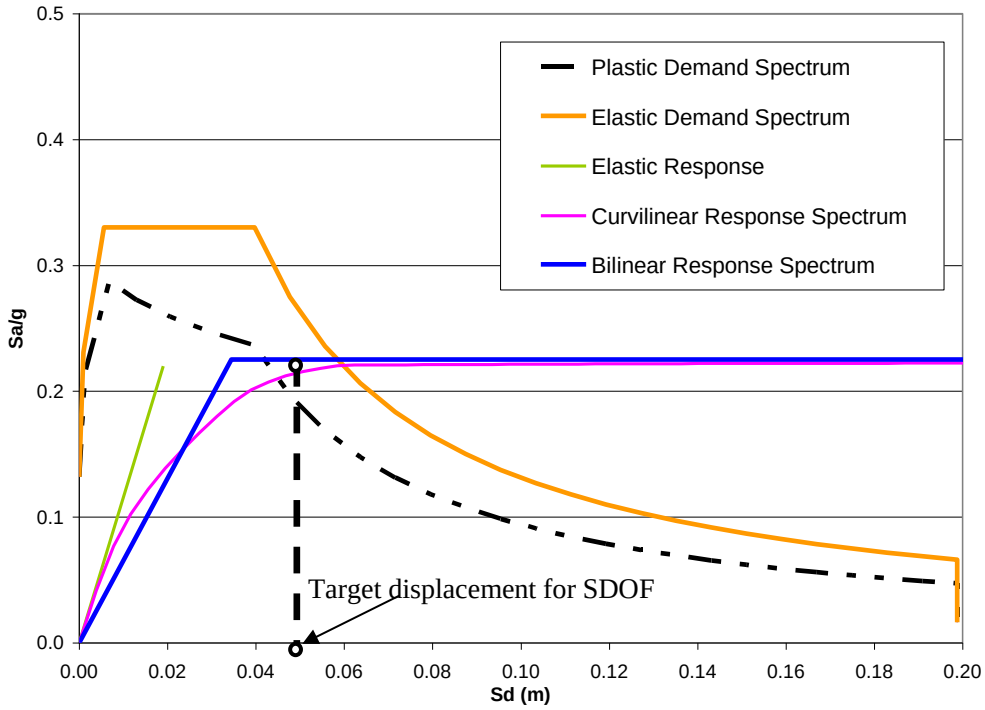


Figure 8.15. Déplacement cible pour la réponse pushover.

La réponse dynamique non-linéaire du même bâtiment est présentée après lui avoir appliqué trois séismes artificieux (figures 8.16 , 8.17 et 8.18).

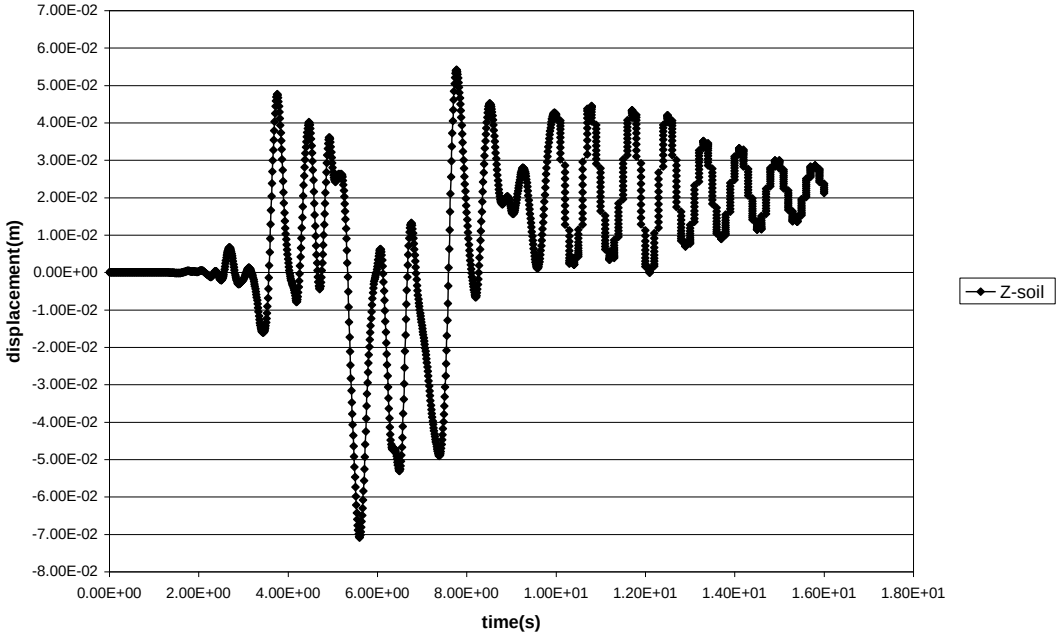


Figure 8.16. Réponse du bâtiment au premier séisme appliqué selon la direction x.

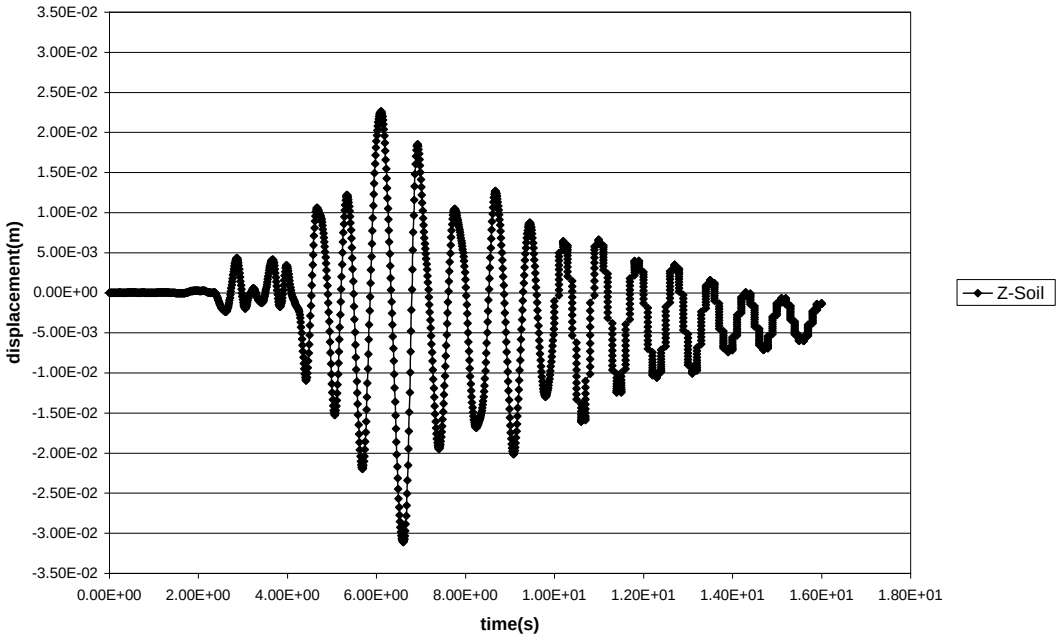


Figure 8.17. Réponse du bâtiment à un deuxième séisme appliqué selon la direction x.

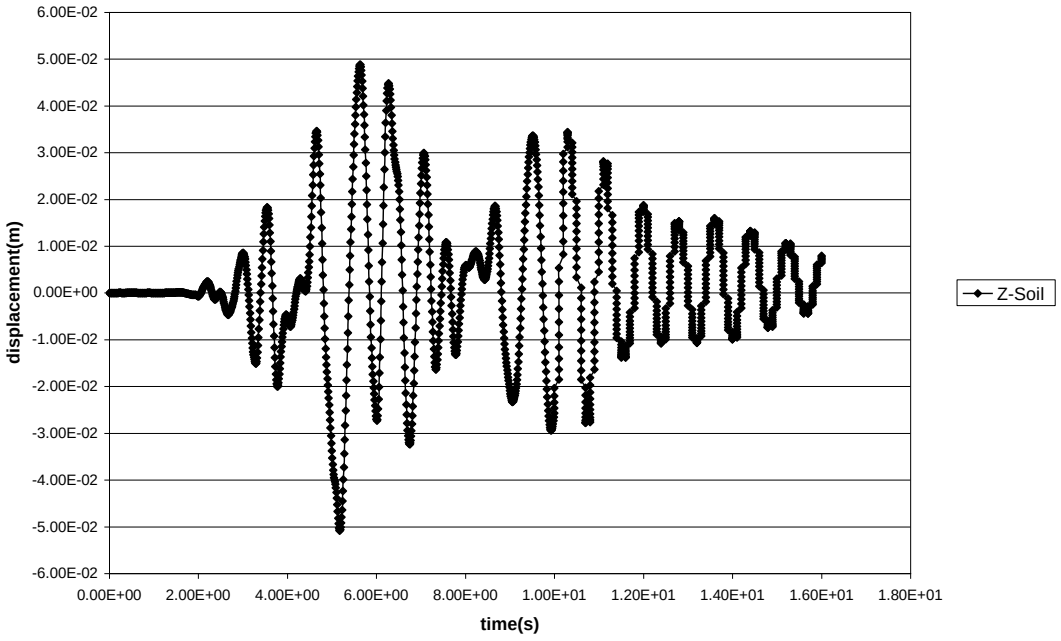


Figure 8.18. Réponse du bâtiment à un troisième séisme appliqué selon la direction x.

Afin de comparer la méthode statique non-linéaire avec celle de la dynamique non-linéaire il faut d'abord multiplier le déplacement cible par le coefficient de participation modal qui vaut : Γ=1.2 d'où les résultats 6 x1.2 =7.2 cm. Cette valeur représente le déplacement au sommet du bâtiment réel, qui va être comparé à la valeur maximale données par la dynamique non-linéaire qui est max (7.22 , 3.1 , 5.13) cm =7.22cm. On voie clairement que les résultats de la statique non-linéaire et la dynamique non-linéaire sont très proches.

8.4. Validation des résultats XFEM donnés par un code orienté objet en C++

8.4.1. Description du problème étudié

Soit une plaque rectangulaire isotrope homogène, d'une largeur : W=10 cm et hauteur : 2H=20 cm, sollicité par une charge repartie : **S** = 100 KN/m^2, dont les propriétés élastiques sont les suivantes : module d'élasticité : $E = 7.10^6\ MPA$, coefficient de poisson : $\upsilon = 0.3$.

La plaque est munies d'une fissure droite sur le côté gauche d'une longueur : a=3 cm.

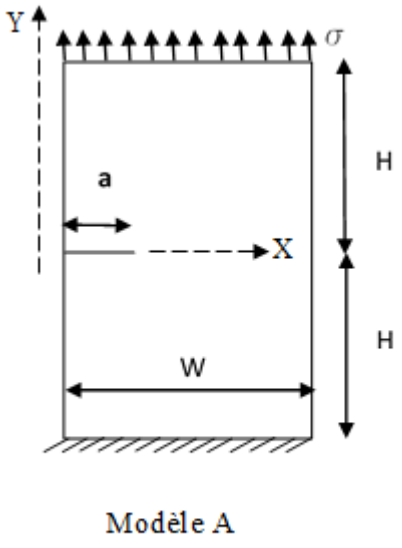
Le but est d'étudier l'influence de la position de la fissure du coté, sur la valeur du facteur d'intensité de contrainte en mode I, en présentant deux cas :

Une fissure du côté de la plaque sous tension uni axiale en variant sa position le long de l'axe y.

Deux fissures parallèles du côté sous tension bi axiale en variant leurs positions le long de l'axe y.

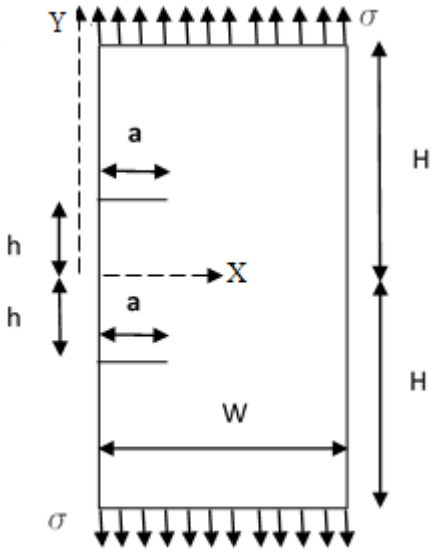
Sachant que le facteur d'intensité de contrainte d'une seule fissure est calculé par le biais de cette formule :

$$K_I = \sigma_{app} \sqrt{\pi a}\, f\left(\frac{a}{w}\right)$$



Modèle A

Figure 8.19. Modèle avec une fissure sur le côté gauche et un chargement dans un seul côté de l'axe y.



Modèle B

Figure 8.20. Modèle avec deux fissures sur le côté gauche et un chargement des deux côté de l'axe y.

8.4.2. Discussion des résultats

8.4.2.1. Présence d'une fissure du côté gauche

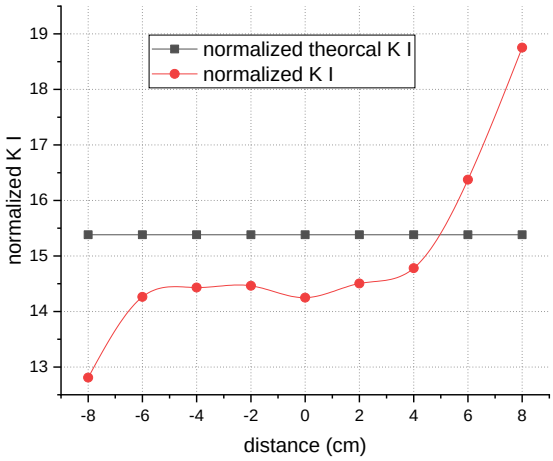
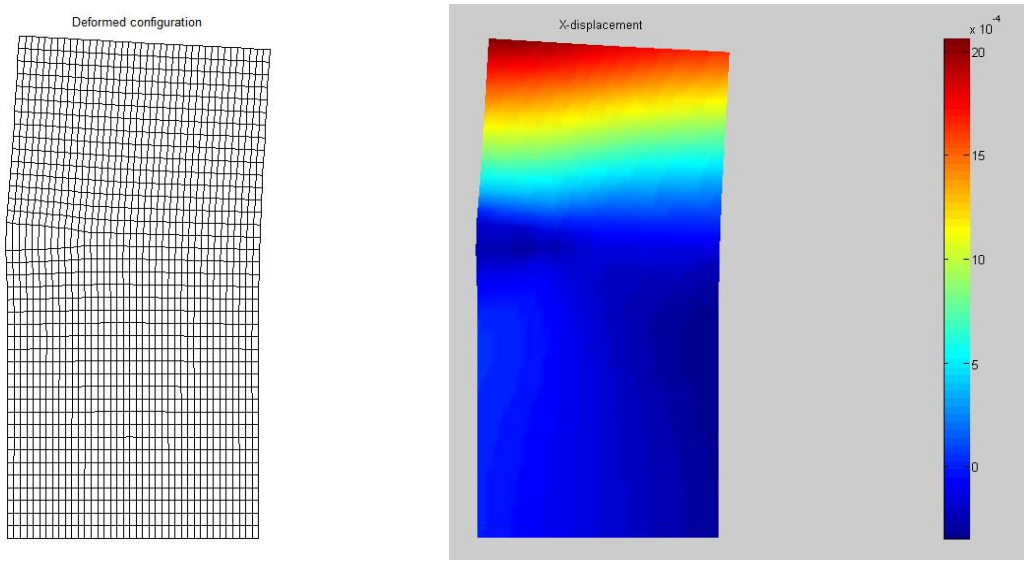


Figure 8.21. Valeur du facteur d'intensité de contrainte en mode I (K I) par rapport à l'emplacement de la fissure.

Un générateur de maillage appelé GMSH [43] est utilisé afin de faire un maillage structuré avec des éléments quadrilatéraux qui est représenté dans la figure ci-dessous.



(a) **(b)**

Figure 8.22. (a) Déformation de la plaque , (b) Déplacement de la plaque selon l'axe Y.

8.4.2.2. Présence de deux fissures du côté gauche

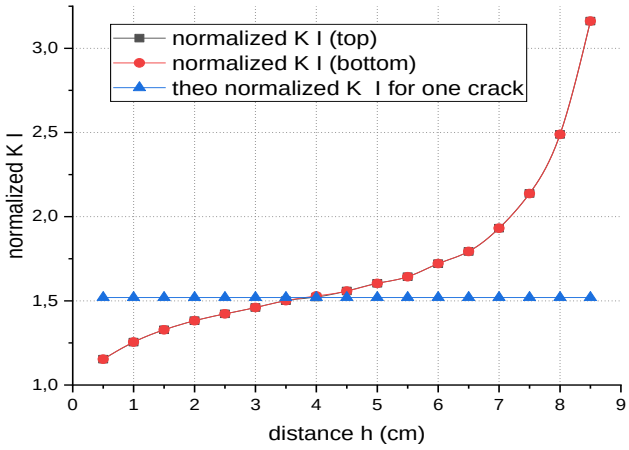


Figure 8.23. Valeur du facteur d'intensité de contrainte normalisé en mode I selon la demi distance (h) entre deux fissures par-rapport à l'axe Y.

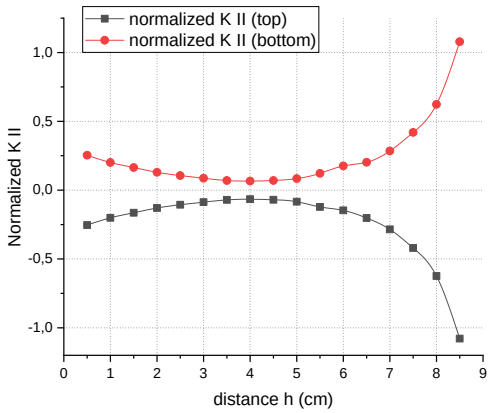
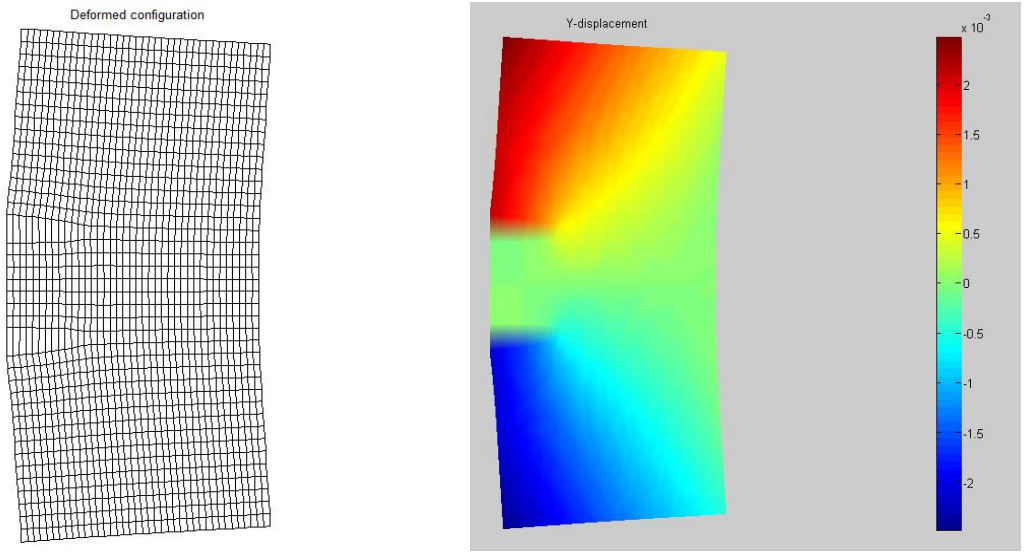


Figure 8.24. Valeur du facteur d'intensité de contrainte normalisé en mode II selon la demi distance (h) entre deux fissures par-rapport à l'axe Y.



(a) **(b)**

Figure 8.25. (a) déformation de la plaque, (b) Déplacement de la plaque selon L'axe Y.

La figure 8.21 illustre l'approximation du facteur d'intensité de contrainte en mode I par rapport à la valeur théorique de la fissure supposée au milieu du côté gauche de la plaque, on voie clairement que dès que la distance est au-delà de l'intervalle [-4, +4]cm par rapport à l'axe y les valeurs du facteur d'intensité diverge, pour ce qui est de la figure 8.22 illustre la déformation ainsi que les déplacements de cette dite plaque.

La figure 8.23 montre l'évolution du facteur d'intensité de contrainte en mode I dans le cas de deux fissures avec une position symétrique selon l'axe x, dont les valeurs converge vers la valeur théorique seulement dans l'intervalle de la demi distance (h) entre les deux fissures allant de [2cm à 6cm] , et divergent au-delà de cet intervalle, ce qui signifie que plus la fissure et plus proche du chargement plus importante est sa propagation.

La figure 8.24 laisse apparaitre l'évolution du facteur d'intensité de contrainte en mode II dans le cas de la présence de deux fissures symétriques selon l'axe x, les valeurs de ce facteur avoisine le zéro pour une demi distance (h) entre les deux fissures allant de 2 à 6 cm. Dès que cette dernière dépasse cet intervalle ; les valeurs du facteur d'intensité de contrainte en mode II prend des valeurs plus importantes symétriquement. Chose qui est prévisible à cause de la présence des efforts de cisaillement.

La figure 8.25 montre la déformation ainsi que le déplacement de la plaque contenant deux fissures.

8.5. Conclusion

Ce dernier chapitre, montre en premier lieu l'attrait rivalisant de la programmation orientée objet en utilisant le C++ par rapport à la programmation procédurale ; en terme de calcul et la suprématie de la nouvelle manière de programmer en terme d'élégance d'extensibilité etc.

En deuxième lieu l'évolutivité des logiciels avec cette nouvelle approche, chose qui a été démontré avec la validation des résultats.

Conclusion Générale

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 par le besoin d'héritage et de liaison dynamique.

Nous proposons une implémentation de la MEF/XFEM 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/XFEM. 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 car l'approches procédurale et orientée objet sont trop différentes pour pouvoir cohabiter harmonieusement dans un même programme.

On arrive a vite remarque que le langage C++ peut rivaliser avec le fortran en terme de temps de calcul tout en utilisant les principes de la notion d'objet tel que le polymorphisme, l'héritage, la classe, et surtout le principe de la non anticipation qui est d'une importance capitale.

En organisant une application en un monde d'objet, l'architecture des résultats ainsi que le mode d'opération est terriblement différent de ceux qui ont été obtenus par l'approche procédurale.

Les taches globales scissionnées en sous taches élémentaires, qui sont exécutées dans un ordre fixe, en comparant à l'approche orientée objet où l'objet encapsule les données et

communique à travers des messages avec un ordre d'activation déterminé par l'état de l'objet, l'architecture orientée objet casse la solution séquentielle du problème.

Le logiciel de modélisation utilisée nous a permis de conclure que la méthode statique non-linéaire est un outil très important pour les ingénieurs grâce à ses avantages à savoir la rapidité de diagnostiquer les structures, par le biais de la puissance graphique de la procédure, possibilité de permettre aux concepteurs de suivre de plus près la réponse non-linéaire des bâtiments et des ponts. C'est une procédure qui nous permet d'éviter le calcul fastidieux de la dynamique non-linéaire.

La mécanique des ruptures est d'une importance capitale pour les structures en génie civil ou mécanique. A cause de la fatigue, des fissures peuvent germer et créer ainsi des dommages irréversibles ; la nouvelle manière de programmer nous permet d'appréhender ce genre de problème sans aucun complexe, en modélisant les fissures et suivre leurs propagations dans la structure.

Dans l'optique d'étendre cette recherche, le développement de classes additionnelles dans le domaine de la mécanique des ruptures, qui traitent l'évolution de la fissure en dynamique pourra faire l'objet d'un axe de recherche future.

REFERENCES

[1] Anderson T.L., (2005), « Fracture Mechanics Fundamental and Applications », Third Edition Taylor and Francis Group, CRC Press.

[2] Antoniou S., and Pinho, R. (2004), « Advantages and Limitations of Adaptive and Non adaptive Force-Based Pushover Procedures ». Journal of Earthquake Engineering, Vol. 8, No. 4, pp. 497-522.

[3] Aydinoglu M. N. (2004), « An Improved Pushover Procedure for Engineering practice: Incremental Response Spectrum Analysis (IRSA) in Performance-based seismic design ». In Krawinkler & Fajfar (Eds.), Concepts and implementation PEER report, Bled, Slovenia (pp. 345-356).

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

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

[6] Belgasmia Mourad, (2021), « Structural Dynamic and Static Nonlinear Analysis from Theory to Application », IGI GLOBAL USA.

[7] Belgasmia M., Spacone E., and Zimmermann Th., (2006), « Seismic Evaluation of Constructions: Static Pushover procedure and time history analysis for Nonlinear Frames », LSC Internal Report, Swiss Federal Institute of Technology Lausanne Switzerland.

[8] Belgasmia M., (2017), **«** Definition of Static Nonlinear Procedure and Flexibility-Based model with Application on 2D Model for an Existing Structure and Comparing Results with Time History Analysis **»**, in a book titled **«** Modeling and Simulation Techniques in Structural Engineering **»**, IGI Global : 61–89.

[9] Belgasmia M., and Moussaoui S., (2015), « Comparison Between Static Nonlinear and Time History Analysis using Flexi-bility-Based Model for an Existing Structure and

Effect of Taking into Account Soil Using Domain Reduction Method for a Single Media », *KSCE* Journal of Civil Engineering, 19(3), 651-663.

[10] Belytschko T., Black T., (1999), « Elastic Crack Growth In Finite Elements With Minimal Remeshing », international journal for numerical methods in engineering; 45:601-620.

[11] Biggs J.M., (1964), « Introduction in structural dynamics », McGraw-Hill, New York.

[12] Bordas. S., Phu. N.V., Dunant. C., Dan. H.N., Guidoum A., (2006), « An Extended Finite Element Library », international journal for numerical methods in engineering; 71(6)703-732.

[13] Ciampi V., and Carlesimo L., (1986), « A nonlinear beam element for seismic analysis of structures », Proceedings of *8th* European. Conference on Earthquake Engineering, Lisbon.

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

[15] Commend S., (1998), « An object-oriented approach to nonlinear finite element programming », LSC Internal Report 98/7.

[16] Commend S., (2013), « Pushover Analysis with Zsoil Taking Soil into Account », Numerics in Geotechnics and Structures Symposium, 28 years of Zsoil and structure PC, Swiszerland.

[17] Commend Stéphane, Thomas Zimmermann, (2001), « Object-Oriented Nonlinear Finite Element Programming », Advances In Engineering Software, 32, 8, 611-628.

[18] Conte J. P., Barbato M., and Spacone E., (2004), « Finite Element Response Sensitivity Analysis Using Force-Based Frame Models », International Journal for Numerical Methods in Engineering, Vol. 59, Issue 13, pp. 1781-1820.

[19] Craveur J.C.,(1996), « Modélisation des structures Calcul par éléments finis », Masson, Paris.

[20] Delannoy C., (2002), « Programmer en C++ », Edition Eyrolles, Paris.

[21] Desjardins R., and Fafard M., (1992), « développement d'un logiciel pour l'analyse des structures par élément finis en utilisant l'approche de la programmation objet », Rapport GCT-92-05, Univrsité Laval, Sainte-Foy, Canada.

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

[23] Dolbow J., Moes N., Belytschko T., (2000), « Modeling Fracture In Mindlin-Reissner Plates With The Extended Finite Element Method », international journal for numerical methods in engineering; 37:7161-7183.

[24] Dubois-pèrlerin Y., and Zimmermann T., (1993), « Object-oriented finite element programming :III. An efficient implementation in C++ », Computer Methods in Applied Mechanics and Engineering 108,165-183, North-Holland.

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

[26] Dubois-Pèlerin Y., Th. Zimmermann, (1993), « Object-oriented finite element programming : Theory and C++ implementation for FEM_Object C++ 001 », Elmepress International.

[27] Dubois Y., Pélerin P., Pegon, (1998), « Object-oriented programming in nonlinear finite element analysis », Computers & Structures (67), pp. 225-241.

[28] Dunant C., Nguyen Vinh P., Belgasmia M., Bordas S., Guidoum A., (2007), « Architecture tradeoffs of integrating a mesh generator to partition of unity enriched objectoriented finite element software », European Journal of Computational Mechanics.

[29] Eurocode 8, (2003), « Design of Structures for Earthquake Resistance », European

Committee for Standardization.

[30] Ewalds H., and Wanhill R., (1984), « Fracture Mechanics », Edward Arnold.

[31] Fajfar P., Kilar V., Marusic D., and Perus I., (2005), « The Extension of the N2 Method to Asymmetric Buildings », Proceedings of the 4th European Workshop on the Seismic Behaviour of Irregular and Complex Structures, N°. 41, Thessaloniki, Greece.

[32] Fajfar P., (1999), « Capacity Spectrum Method Based on Inelastic Demand Spectra », Earthquake Engineering and Structural Dynamics, n°28, p 979-993.

[33] Fajfar P., (2002), « Structural Analysis in Earthquake Engineering – A Breakthrough of Simplified Non-Linear Methods », Proc. 12th European Conference on Earthquake Engineering, P 843.

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

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

[36] Fenves G.L., (1990), « Object-oriented programming for Engineering software development », Engineering With Computer. 6 1-15.

[37] Forde B.W.R., Foschi R.O., and Stiemer S.F., (1990), « Object-oriented finite element analysis », Computers.& Structures 34,3, 355-374.

[38] Frey F., (2003), « Analyse des structures et milieux continus », Coque, Vol. 5, 1ére édition, Lausanne.

[39] Frey F., (2001), « Analyse des structures et milieux continus Méthode des éléments finis », Vol. 6, 1ére édition, Lausanne.

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

[41] Gazetas G., (2006), « Seismic Design of Foundation and Soil-Structure Interaction », First European Conference on Earthquake Engineering and Seismology, Geneva, Swiszerland.

[42] Geoffroy P., (Avril 1983), « 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.

[43] Geuzaine C., remacle J.F., « GMSH An Open Source 3D Finite Element Mesh generator »**,** http://geuz.org/gmsh/.

[44] Han W.-S., (1989), « 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.

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

[46] Hjelmstad K.D., and Taciroglu E., (2002), « Mixed Methods and Flexibility Approaches for Nonlinear Frame Analysis », Journal of Constructional Steel Research., Vol. 58, No.5-8, pp. 967–993.

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

[48] Hughes T.J.R., T. Belytschko, (1997), « Nonlinear Finite Element Analysis », Course Notes, Paris.

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

[50] Jabur L.S., (2015), « Theorical and Numerical Analysis of Central Crack Plate With Different Orientation Under Tensile Load », international journal for industrial engineering and technologie; (E)2278-9456.

[51] Jabur L.S., (2015), « Stress Intensity Factor For Double Edge Cracked Finite Plate Subjected To Tensile Stress », Thi-Qar university journal for engineering science; Vol (6), N°2.

[52] Kaba S., and Mahin S.A., (1984), « Refined Modeling of Reinforced Concrete Columns for Seismic Analysis », EERC Report 84/03**,** Earthquake Engineering Research Center, University of California, Berkeley.

[53] Kanninen M. and Popelar C., (1985), « Advanced fracture mechanics », Volume Oxford Engineering Science Series, Oxford University Press.

[54] Kris Jamsa, Lars Klander, (1999), « La bible du programmeur C/C++ », Editions Reynald Goulet.

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

[56] Mackerle J., (2004), « Object-oriented programming in FEM and BEM », Advances in Engineering Software 35,325-336.

[57] Menétrey Ph., Zimmermann Th., (1993), « Object-oriented nonlinear finite element analysis : application to J2 plasticity », Computers & Structures (49), pp. 767-777.

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

[59] Moes, N., J. Dolbow, and T. Belytschko, (1999), « A finite element method for crack growth without remeshing », International Journal for Numerical Methods in Engineering 46, 131-150.

[60] Mohammadi S., (2008), « Extended Finite Element Method_ for Fracture Analysis of Structures », Wiley-Blackwell.

[61] Moussaoui S., & Belgasmia M., (2018), « Improvement of continuity between Industrial Software and research one by Object Oriented Finite element formulation of shell and plate element », Book chapter, Optimization of Design for Better Structural Capacity IGI Global USA.

[62] Neuenhofer A., and Filippou F., (1998), « Geometrically Nonlinear Flexibility-Based Frame Finite Element », Journal of Structure Engineering, Vol. 124, N°. 6, pp. 704-711.

[63] Nguyen V. P., (2005), « An Object-Oriented approach to the Extended Finite Element Method with Applications to Fracture », Mechanics thesis at Hochiminh City University of Technology.

[64] Patrick Paultre, (2005), « Dynamique des structures application aux ouvrages de génie civil », Hermes, Paris.

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

[66] Reynders E., and Roeck G., (2010), « A local Flexibility Method for Vibration-Based Damage Localization and Quantification », Journal of Sound and Vibration, Vol. 329, N°.12, pp. 2367–2383.

[67] Sabetta F., and Pugliese A., (1996), « Estimation of Response Spectra and Simulation of Non Stationary Earthquake Ground Motions », Bulletin of the Seismological Society of America, Vol. 86, N°. 2, pp. 337-352.

[68] Sharma K., (2014), « Crack Interaction Studies Using XFEM Technique », journal Of Solid Mechanics; Vol (6), N°4: pp410-421.

[69] Spacone E., (1994), « Flexibility-Based Finite Element Models for the Nonlinear Static and Dynamic Analysis of Concrete Frame Structures », Ph.D. Dissertation**,** Department of Civil Engineering, University of California, Berkeley.

[70] Spacone E., Filippou F.C., and Taucer F.F., (1996), « Fiber Beam-Column Model for Nonlinear Analysis of R/C Frames. I: Formulation, II: Applications », Earthquake Engineering and Structural Dynamics, n°25(7), p711-742.

[71] Sukumar N. and J.-H. Prevost, (2003), « Modeling quasi-static crack growth with the extended finite element method », Computer implementation. International Journal of Solids and Structures.

[72] Taucer F.F., Spacone E., and Filippou F. C., (1991), « A Fiber Beam-Column Element for Seismic Response Analysis of Reinforced Concrete Structures », *EERC Report* 91/17, Earthquake Engineering Research Center, University of California, Berkeley.

[73] Valipour H.R., and Foster S.J., (2010), « A Total Secant Flexibility-Based Formulation for Frame Elements with Physical and Geometrical Nonlinearities », Finite Elements in Analysis and Design, Vol. 46, N°. 3, pp. 288-297.

[74] Zeris C.A., and Mahin S.A., (1988), « Analysis of Reinforced Concrete Beam-Columns Under Uniaxial Excitation », Journal of Structure Engineering, Vol. 114, N°. 4, pp. 804-820.

[75] Zeris C.A., and Mahin S. A., (1991), « Behavior of Reinforced Concrete Structures Subjected to Biaxial Excitation », Journal of Structure Engineering, Vol. 117, pp. 2657-2673.

[76] Zimmermann Th., Truty A., Urbanski A., and Podles K., (2008), « Z-Soil User Manual, Zace Services », Switzerland.

[77] Zimmermann Thomas, Y. Dubois-Pélerin, P. Bomme, (1992), « Object-oriented finite element programming: I Governing principles », Computer Methods in Applied Mechanics and Engineering (98), 291-303.