République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique.
Université Abderrahmane Mira – Bejaia
Faculté de Technologie
Département de Génie Civil

Cours
Méthode des Éléments Finis

Préparé et présenté par

Abdelghani SEGHIR

Docteur en Sciences de l'université A. Mira, Béjaia, Algérie
Docteur en Génie Civil de l'université Paris-Est, Marne-la-Vallée, France

2005-2014

Table des matières

chapitre 1

Présentation de MATLAB

1.1 Introduction

MATLAB est un logiciel interactif basé sur le calcul matriciel (MATrix LABoratory). Il est utilisé dans les calculs scientifiques et les problèmes d'ingénierie parce qu'il permet de résoudre des problèmes numériques complexes en moins de temps requis par les langages de programmation courant, et ce grâce à une multitude de fonctions intégrées et à plusieurs programmes outils testés et regroupés selon usage dans des dossiers appelés boites à outils ou "toolbox".

Au démarrage de MATLAB sur un PC, l'interface se présente comme suit :

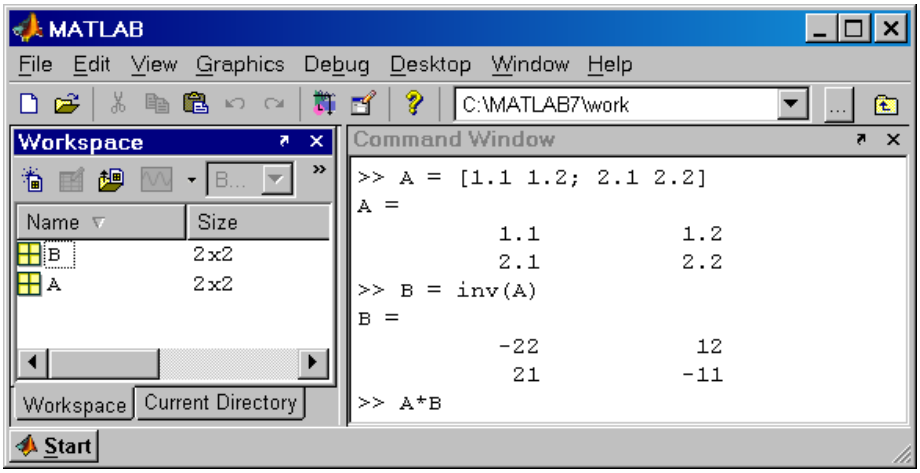


Figure 1.2 : fenêtre principale de MATALB (version 7.0)

La fenêtre principale de MATLAB contient deux fenêtres secondaires pouvant être déplacées ou fermées.

A droite la fenêtre des commandes permet à la fois d'introduire les commandes ligne par ligne et d'afficher les résultats. Le symbole >> indique l'attente d'une commande.

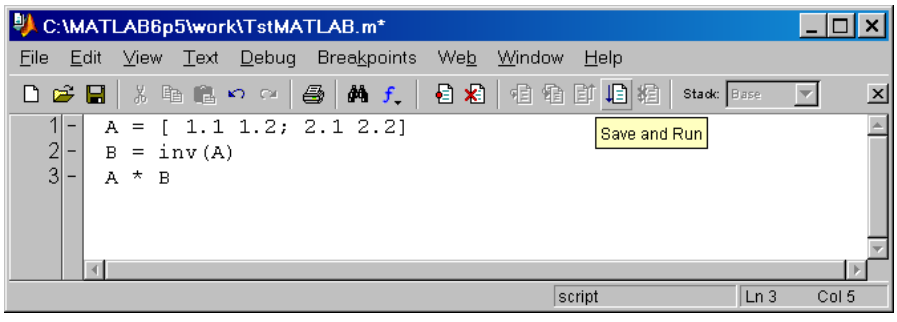
A gauche, sont imbriquées en volets les fenêtres `Workspace`, `Current Directory` et parfois `Command History`.

- `Workspace` permet d'afficher les variables utilisées avec leurs tailles.
- `Current Directory` affiche le chemin d'accès ou le répertoire en cours avec tous les fichiers et les sous répertoires.
- `Command History` affiche les commandes ayant été saisies.

Les déclarations et les commandes peuvent aussi être introduites sous forme d'un *script* dans un fichier texte d'extension "`.m`". MATLAB est équipé d'un éditeur de texte permettant de saisir les fichiers script. La commande `edit prog1` ouvre l'éditeur et charge le fichier `prog1.m` s'il existe, sinon l'éditeur s'ouvre sur un fichier vide.

La figure suivante

L'exécution du script (les commandes une après une) se fait à l'aide du bouton `Save and Run` ou avec le menu `debug/Save and Run` ou bien, simplement, en appuyant sur la touche fonction `F5`. Les résultats d'exécution sont affichés dans la fenêtre des commandes de la même manière que si les commandes sont entrées dans cette fenêtre.



Remarques :

Une ligne de commande peut contenir plusieurs instructions séparées par des virgules ou

par des points-virgules. Le résultat d'une instruction suivie par un point-virgule ne sera pas affiché.

1.2 Introduction des matrices

MATLAB fonctionne essentiellement avec des matrices. Les composantes peuvent être réelles, complexes ou symboliques. Les scalaires sont interprétés comme des matrices 1×1 !!

Il existe plusieurs façons pour introduire une matrice dans MATLAB :

1- Entrée explicitement par liste des éléments

```
>> A = [ 1.1 1.2
        2.1 2.2 ]
>> A = [ 1.1, 1.2 ; 2.1, 2.2 ]
```

Les colonnes de la matrice sont séparés par des virgules (`,`) ou des espaces et les lignes par des points-virgules (`;`) ou des sauts de ligne (RETURN).

2- Générée par une fonction interne (built-in function)

```
>>A = eye(3)                    % matrice identité
A =
    1     0     0
    0     1     0
    0     0     1

>>B = ones(3)                   % matrice carrée contenant des 1
B =
    1     1     1
    1     1     1
    1     1     1
>>C = zeros(2,3)                % matrice rectangulaire nulle
C =
    0     0     0
    0     0     0
>>D = diag([1:3])               % matrice dont la diagonale varie de 1
à 3
D =
    1     0     0
    0     2     0
```

```
    0     0     3
>> A = rand(2,3)                % matrice aléatoire, un 2eme appel de
rand
A =                             % renvois une autre matrice 2×3
    0.7271    0.8385    0.3704
    0.3093    0.5681    0.7027
```

3- Entrée à partir d'un fichier script avec l'éditeur

4- Chargée à partir d'un fichier de données ou importée d'une application.

```
>>load matrice.txt
>>A = load('matrice.txt')
```

Le fichier `matrice.txt` contient des valeurs numériques disposées en tableau.

La première commande charge les valeurs de la matrice et affecte le nom `matrice` au résultat par contre la seconde permet de choisir un nom pour la matrice chargée.

La structure du fichier `matrice.txt` est :

```
1 5 7
5 4 2
4 2 1
```

Les espacements entre les composantes peuvent être irréguliers. Des virgules ou points-virgules peuvent être utilisés pour séparer les colonnes et les lignes. Une ligne incomplète provoque une erreur d'exécution.

1.3 Opérations sur les matrices

Les opérations matricielles suivantes sont disponibles sous MATLAB

```
+         Addition              C = A + B
–         Soustraction          C = A - B
*         Multiplication        C = A * B
^         Puissance             C = A^2 ou C = A * A
'         Transposée            C = A'  ou C = transpose(A)
\         division gauche       x = A\b
/         division droite       x = b/A
```

Ces opérations matricielles s'appliquent aussi sur des scalaires. Une erreur est provoquée dans le cas où les rangs des matrices sont incompatibles.

Remarque 1

La division matricielle implique la résolution de systèmes d'équations linéaires. Si `A` est une matrice carrée inversible :

`x = A\b` est la solution du système `A*x = b`, avec `x` et `b` sont des vecteurs colonne

`x = b/A` est la solution du système `x*A = b`, avec `x` et `b` sont maintenant des vecteurs lignes

```
>> A = [3 2; 4 7]
A =
     3     2
     4     7
>> b = [2;3]
b =
     2
     3
>> x = A\b
x =
    0.6154
    0.0769
>> A * x
ans =
     2
     3
y = b'
y =
     2     3
>> z = y/A
z =
    0.1538    0.3846
>> z * A
ans =
     2     3
```

Remarque 2

Si un point est ajouté à gauche des opérateurs `* ^ \` et `/` ils seront appliqués sur les éléments des matrices avec correspondance d'indices :

```
>> A * A       % équivalent à A ^ 2
ans =
    17    20
```

```
    40    57
>> A .* A      % équivalent à A .^ 2
ans =
     9     4
    16    49
```

1.4 Opérations sur les vecteurs

Les vecteurs sont considérés comme des matrices rectangulaires d'une ligne ou d'une colonne. Toutes les opérations matricielles s'appliquent sur les vecteurs mais il faut faire attention à la correspondance des tailles. Les opérateurs précédés d'un point sont particulièrement utiles dans les calculs de vecteurs et dans les représentations graphiques.

Exemples de manipulation de vecteurs :

```
>> a = [2 1 3]          % vecteur ligne
a =
     2     1     3

>> b = [1;3;2]         % vecteur colonne
b =
     1
     3
     2

>> a * b               % produit scalaire
ans =
    11

>> a .* b'             % produit de composantes (remarquer b')
ans =
     2     3     6

>> x = 0:0.2:1         % vecteur à pas fixe
x =
         0    0.2000    0.4000    0.6000    0.8000    1.0000

>> y = x .^ 2 .* cos(x)  % y = x^2 cos(x) (noter les points)
y =
```

```
          0    0.0392    0.1474    0.2971    0.4459    0.5403

>> plot(x,y)   % affiche une fenêtre du graphe y(x)
```

1.5 Fonctions sur les matrices

MATLAB contient plusieurs fonctions qui s'appliquent sur les matrices. Celles qui peuvent nous intéresser actuellement sont les suivantes, pour en savoir plus taper `help` ou `demo` sur la ligne de commande.

```
det     déterminant
        >> d = det(A)

inv     inverse
        >> B = inv(A)        (I = B * A)

eig     valeurs et vecteurs propres
        >> [Vects,Vals] = eig(A)

chol    décomposition de Cholesky
        >> C = chol(A)       (A = C'*C

lu      décomposition LU
        >> [L,U] = lu(A)     A = L *U

qr      décomposition QR
        >> [Q,R] = qr(A)     A = Q *R

svd     décomposition en valeurs singulières
        >> [U,S,V] = svd(A)  A = U*S*V'

norm    norme
        Plusieurs définitions (taper help norm)
```

Les fonctions scalaires telles que cos, sin, exp, log … etc. s'appliquent sur les éléments des matrices

Exemple de calcul de valeurs et vecteurs propres :

```
>> A = [5 2 1; 2 7 3; 1 3 8] ;
>> [vects,vals] = eig(A)
```

```
vects =
    0.7040    0.6349    0.3182
   -0.6521    0.4005    0.6437
    0.2812   -0.6607    0.6959
vals =
    3.5470         0         0
         0    5.2209         0
         0         0   11.2322
>> vects' * A * vects
ans =
    3.5470   -0.0000   -0.0000
   -0.0000    5.2209   -0.0000
   -0.0000   -0.0000   11.2322
```

1.6 Accès aux éléments des matrices

L'accès aux éléments d'une matrice se fait à l'aide de leurs indices ligne et colonne mis, après le nom de la matrice, entre parenthèses : `A(i,j)` est l'élément de ligne *i* et de colonne *j*.

Il est possible d'obtenir une sous matrice en spécifiant une liste d'indices et deux points à la place d'un indice ou d'une liste d'indice sont interprétés comme ligne ou colonne entière :

```
>> A = [1.1 1.2 1.3 1.4
        2.1 2.2 2.3 2.4
        3.1 3.2 3.3 3.4
        4.1 4.2 4.3 4.4]

>> A(2,2:4)
ans =
    2.2000    2.3000    2.4000

>> A(3,:)
ans =
    3.1000    3.2000    3.3000    3.4000

>> A(:,3)
ans =
```

```
    1.3000
    2.3000
    3.3000
    4.3000
>> t = [1,4]
t =
     1     4
>> A(t,t)      % Cette écriture est utile en MEF
ans =
    1.1000    1.4000
    4.1000    4.4000
```

1.7 Les boucles dans MATLAB

La structure des boucles dans MATLAB ne diffère pas de celle d'autres langages de programmation. Il existe deux type de boucles : `for` et `while`.

La boucle `for` a la structure suivante :

```
for valeur = valeur_initiale : pas : valeur_finale
    instruction 1
    instruction 2
    . . . . . .
end
```

La boucle `while` a la structure suivante :

```
initialisations
while expression logique
    instruction 1
    instruction 2
    . . . . . .
end
```

La boucle `for` fait varier une variable d'une valeur initiale à une valeur finale avec un pas d'incrémentation constant qui est pris égale à un s'il est omis. Les instructions à l'intérieur de la boucle sont exécutées à chaque valeur de la variable. La boucle `while` s'exécute tant qu'une expression logique est vraie elle nécessite souvent des initialisations avant d'être lancer et peut se dérouler indéfiniment si l'expression logique de contrôle reste toujours vraie.

Les deux boucles peuvent être rompues par la commande `break`. On insère généralement `break` avec un teste de condition `if`.

Les deux boucles suivantes donne un même vecteur : `f = w = 2 4 6 8 10`

```
for i = 1:5          % boucle for à pas unité
    f(i) = 2*i;      % l'incrémentation de i est automatique
end;

i = 1;               % initialisation de i
while i <= 5         % boucle tant que i est inférieur ou égal à 5
    w(i) = 2*i;
    i = i+1;         % l'incrémentation est importante sinon la
end;                 % boucle est infinie
```

1.8 Les testes logiques

Avant les boucles

Les testes logiques dans MATLAB se font à l'aide des mots clés `if`, `else` et `end`. Les opérateurs relationnels utilisés dans les instructions logiques sont :

```
< inférieur      <=  inférieur ou égal
> supérieur     >=  supérieur ou égal
== égal           ~=   non égal (différent)
```

et les opérateurs logiques sont :

```
& et  exemple : c = a & b  ou bien c = and(a,b)
| ou  exemple : c = a | b  ou bien c = or(a,b)
~ non exemple : c =~a      ou bien c = not(a)
```

Exemple de teste `if`

```
if a < b
    m = a
else
    m = b
end
```

1.9 Les Fichiers fonctions

Les fichiers scripts permettent d'écrire et de sauvegarder les commandes dans un fichier disque. Les fichiers fonctions ont plus d'importance, ils offrent une extensibilité à MATLAB. On peut créer une fonction spécifique à un problème donné et l'utiliser de la même manière que les autres fonctions offertes par MATLAB. Un fichier fonction possède la structure suivante :

```
function [res1, res2, ...] = nomfonction(var1, var2, ...)
% mettre ici les lignes commentaire à afficher
% dans la fenêtre de commandes suite à une demande d'aide
% avec help nomfonction
instruction
instruction
```

le mot clé `function` est indispensable. Les variables d'entrée sont `var1`, `var2` ... et les résultats sont `res1`, `res2`, ... Autant de variables et de résultas que nécessaire peut être utilisé. Le fichier doit être sauvegarder sous le nom `nomfonction.m` et peut contenir plus d'une fonction.

Un mot clé `return` peut être ajouté en fin de chaque fonction.

```
function R = rot3dz(theta)
% R = rot3dz(theta)
% retourne une matrice de rotation 3d
% d'un angle autour theta de l'axe z
% theta en degrés mesuré à partir de l'axe x

theta = theta * pi/180;          % transformation en radian
c = cos(theta);                  % cos et sin pour plus de rapidité
s = sin(theta);
R = [ c  s  0
     -s  c  0
      0  0  1
     ];
return
```

Pour avoir une rotation de 30° autour de z, on peut appeler la fonction comme suit :

```
>> r  = rot3dz(30)
r =
    0.8660    0.5000         0
   -0.5000    0.8660         0
```

```
     0          0    1.0000
```

1.10 Application 1

1) Ouvrir MATLAB
2) Créer deux matrices `A` et `B` de 3×3 dans la fenêtre de commande
3) Créer un vecteur colonne `a` et un vecteur ligne `b` de longueur 3
4) Calculer les transposées des matrices `A`, `B` et des vecteurs `a`, `b`.
5) Calculer `A + B` et `A - B`.
6) Calculer `A * B` et `A .* B`, commenter
7) Calculer les vecteurs `x = A \ b` et `r = A * x`.
8) Calculer les vecteurs `x = a / A` et `r = x * A`.
9) Calculer les matrices `X = A / B` et `X = A \ B`, vérifier `A*X`, `X*A`, `B*X` et `X*B`
10) Calculer les matrices `X = A ./ B` et `X = A .\ B`, vérifier `A*X`, `X*A`, `B*X` et `X*B`
11) Calculer les matrices `R = A ^ 2`, `R = A .^2` et `R = A .^B`, commenter
12) Calculer `(A'\a')'` et `R = (b'\A')'`, commenter
13) Calculer `inv(A)*b` et `a*inv(A)` , commenter
14) Calculer `a*b` , `b*a`, `b'*a`, `a'*b`, `b'*a'`, commenter
15) Calculer `a.*b` , `b.*a`, `b'.*a`, `a'.*b`, `b'.* a'`, commenter
16) Créer trois fonctions pour les trois rotations 3D autour des axes *x*, y et *z*. Calculer une matrice de rotation `R` composée d'une rotation de 30° autour de *x* de 60° autour de *y* et de 45° autour de *z*. Calculer ensuite `Ar = R'*A*R`.
17) Comparer `trace(A)`, `trace(Ar)`, et `eig(A)`, `eig(Ar)`. Commenter
18) Créer un fichier de commandes à l'aide de l'éditeur MATLAB et refaire 2 à 10.
19) Créer un fichier texte contenant les éléments d'une matrices 3×3 et charger la matrice à l'aide d'une ligne de commande.

1.11 Exercice

Créer la matrice *A* en utilisant la fonction `diag` et un vecteur *b* avec un pas fixe

```
A =
     1     1     0
     1     2     2
     0     2     3
b =
     1
     2
     3
```

Résoudre le système d'équations $A\,x = b$ en utilisant :

1) L'opérateur de division,
2) La décomposition LU,
3) La décomposition QR.

Utiliser `help` pour les fonctions `diag`, `LU` et `QR`. Exemple `help lu`.

1.12 Devoir - TP

Ecrire des scripts MATLAB qui permettent de :

1) Calculer le tenseur des déformations en fonction du tenseur des contraintes en utilisant la relation $\varepsilon = (1 + \nu)/E\ \sigma - \nu/E\ s\ I$.
2) Calculer le tenseur des contraintes en fonction du tenseur de déformation en utilisant la relation $\sigma = 2\ \mu\ \varepsilon + \lambda\ e\ I$.
3) Calculer les relations $\sigma = D\ \varepsilon$ et $\varepsilon = C\ \sigma$ en utilisant l'écriture vectorielle et la matrice d'élasticité *D*, vérifier que $C^{-1} = D$
4) Calculer les contraintes et directions principales
5) Vérifier qu'une rotation de 60° autour de *y* donne les directions principales du tenseur des contraintes σ pris en exemple ci-dessous
6) Vérifier les invariants des contraintes et les invariants des déformations

7) Calculer le vecteur contrainte et le vecteur déformation sur une normale *n*.
8) Calculer la contrainte normale et la dilatation unitaire suivant la normale *n*.
9) Proposer une structure de plusieurs fichiers fonctions pour ce type de problème

Prendre comme exemple, le tenseur des contraintes : $\sigma = \begin{bmatrix} 7 & 0 & -3\sqrt{3} \\ 0 & 25 & 0 \\ -3\sqrt{3} & 0 & 13 \end{bmatrix} MPa$,

Le module d'élasticité : $E = 200\ 000$ MPa, Le module de Young : $\nu = 0.25$ et

Le vecteur normale : $n = [\ {}^{1}/_{\sqrt{3}}\ \ {}^{1}/_{\sqrt{3}}\ \ {}^{1}/_{\sqrt{3}}\ ]^{T}$

Rappel :

$$D = \begin{bmatrix} \lambda+2\mu & \lambda & \lambda & 0 & 0 & 0 \\ & \lambda+2\mu & \lambda & 0 & 0 & 0 \\ & & \lambda+2\mu & 0 & 0 & 0 \\ & & & 2\mu & 0 & 0 \\ & Sym & & & 2\mu & 0 \\ & & & & & 2\mu \end{bmatrix};$$

$$C = \frac{1}{E}\begin{bmatrix} 1 & -\nu & -\nu & 0 & 0 & 0 \\ & 1 & -\nu & 0 & 0 & 0 \\ & & 1 & 0 & 0 & 0 \\ & & & 1+\nu & 0 & 0 \\ & Sym & & & 1+\nu & 0 \\ & & & & & 1+\nu \end{bmatrix}$$

$I_1 = \sigma_{ii} = \sigma_I + \sigma_{II} + \sigma_{III}$;

$I_2 = \frac{1}{2}(\sigma_{ii}\,\sigma_{jj} - \sigma_{ij}\,\sigma_{ji}) = \sigma_I\sigma_{II} + \sigma_{II}\sigma_{III} + \sigma_{III}\sigma_I$;

$I_3 = \det(\sigma) = \sigma_I\sigma_{II}\sigma_{III}$.

$\lambda = \nu E/(1-2\nu)(1+\nu)$; $2\mu = E/(1+\nu)$

chapitre 2

Présentation générale de la Méthode des Eléments Finis

2.1 Introduction

Pour analyser un phénomène naturel en générale ou un problème d'ingénierie en particulier, on est souvent amené à développer un modèle mathématique pouvant décrire d'une manière aussi fiable que possible le problème en question.

Le développement d'un modèle mathématique s'appuis généralement sur quelques postulats de base et plusieurs hypothèses simplificatrices pour aboutir à des équations gouvernantes qui sont souvent des équations différentielles auxquelles sont ajoutées des conditions aux limites. Exemple, la théorie d'élasticité s'appuis sur le postula fondamental de l'existence du vecteur contrainte et les équations générales d'élasticité linéaire isotrope sont obtenues avec les hypothèses de petites déformations, d'homogénéité et d'isotropie des matériaux ainsi que la linéarité des relations liants les contraintes et les déformations.

La résolution analytique d'équations différentielles pose parfois des difficultés insurmontables, et une solution exacte décrivant bien le problème étudié n'est pas toujours facile à trouver. Le recours aux modèles physiques et à la simulation expérimentale pour la recherche d'une solution analogue à la solution recherchée peut s'avérer coûteux en temps et en moyens.

Avec les progrès enregistrés dans le domaine de l'informatique et les performances des ordinateurs de plus en plus grandes, il est devenu possible de résoudre des systèmes d'équations différentielles très complexes. Plusieurs techniques de résolution numérique

ont été ainsi développées et appliquées avec succès pour avoir des solutions satisfaisantes à des problèmes d'ingénierie très variés.

La méthode des éléments finis est l'une des techniques numériques les plus puissantes. L'un des avantages majeurs de cette méthode est le fait qu'elle offre la possibilité de développer un programme permettant de résoudre, avec peu de modifications, plusieurs types de problèmes. En particulier, toute forme complexe d'un domaine géométrique où un problème est bien posé avec toutes les conditions aux limites, peut être facilement traité par la méthode des éléments finis.

Cette méthode consiste à diviser le domaine physique à traiter en plusieurs sous domaines appelés éléments finis à dimensions non infinitésimales. La solution recherchée est remplacée dans chaque élément par une approximation avec des polynômes simples et le domaine peut ensuite être reconstitué avec l'assemblage ou sommation de tous les éléments.

2.2 Historique de la méthode

La méthode des éléments finis est le fruit de deux domaines de recherche : Les mathématiques et les sciences de l'ingénieur. [26]

Mathématique : Outils qui remontent jusqu'aux résidus pondérés de Gauss (1775), Galerkin (1915) et Biezenokoch (1923), ainsi qu'aux méthodes variationnelles de Rayleigh (1870) et Ritz (1909).

Sciences de l'ingénieur : Dont la contribution a débuté dans les années quarante avec Hrenikoff (1941), Henry (1943) et Newmark (1949) qui touchèrent pour la première fois aux structures continues, en faisant une approximation sur des portions de petites dimensions dans un problème continue d'une barre longue. D'où l'idée de base des éléments finis.

Argyris (1955), Turner (1956), Glough (1956) et Martin (1956) ont fait une analogie directe en adoptant un comportement simplifié pour des petites portions : ils représentent un milieu continu élastique à deux dimensions par un assemblage de panneaux triangulaires, sur lesquels les déplacements sont supposés variés linéairement comme pour chaque barre ou poutre du système discret : chaque panneau est décrit par une

matrice de rigidité et l'assemblage donnait la rigidité globale du milieu continu. D'où la naissance d'éléments finis avec "panneaux" comme nom.

Argyris et Kelsy (1960) utilisent la notion d'énergie dans l'analyse des structures et font appel à des méthodes mathématiques (résidus pondérés, principes variationnels ...).

Le terme " élément fini " est utilisé pour la première fois par Glough (1960), et dés lors, il y a un développement rapide de la méthode.

Dans les années soixante; Zienkiwicz (1965), De Arante (1968), Oliviera (1968), Green (1969), Tones (1969), Lay (1969), Storne (1969), et Finlayson (1975) ont reformulé la méthode à partir de considérations énergétiques et variationnelles sous forme générale de résidus pondérés, d'où le modèle mathématique de la MEF.

En 1969 la MEF est reconnue comme un outil général de résolution d'EDP, et utilisée pour résoudre des problèmes non linéaires et non stationnaires dans plusieurs domaines.

En mécanique des fluides, la résolution des équations de Navier Stokes incompressibles par éléments finis en utilisant la formulation vitesse – pression a commencé dans les années 1970.

2.3 Les grandes lignes de la méthode

Dans ce paragraphe, nous essayerons de présenter d'une manière simplifiée, les étapes d'application de la méthode des éléments finis et les outils nécessaires à sa mise en œuvre. La résolution d'un problème physique par éléments finis suit grosso modo les étapes suivantes (Fig. 1.1) :

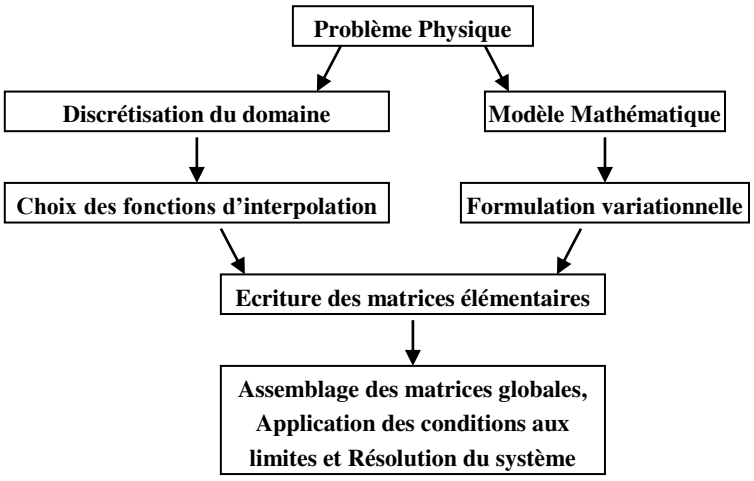


Figure 1.1 : Etapes générales de la méthode des éléments finis

Etape 1 : *Formulation des équations gouvernantes et des conditions aux limites.*

La majorité des problèmes d'ingénierie sont décrits par des équations différentielles aux dérivées partielles associées à des conditions aux limites définies sur un domaine et son contour. L'application de la MEF exige une réécriture de ces équations sous forme intégrale. La formulation faible est souvent utilisée pour inclure les conditions aux limites.

Etape 2 : *Division du domaine en sous domaines.*

Cette étape consiste à discrétiser le domaine en éléments et calculer les connectivités de chacun ainsi que les coordonnées de ses nœuds. Elle constitue ainsi la phase de préparation des données géométriques.

Etape 3 : *Approximation sur un élément.*

Dans chaque élément la variable tel que le déplacement, la pression, la température, est approximée par une simple fonction linéaire, polynomiale ou autres. Le degré du polynôme d'interpolation est relié au nombre de noeuds de l'élément. L'approximation nodale est appropriée. C'est dans cette étape que se fait la construction des matrices élémentaires.

Etape 4 : *Assemblage et application des conditions aux limite.*

Toutes les propriétés de l'élément (masse, rigidité,...) doivent être assemblées afin de former le système algébrique pour les valeurs nodales des variables physiques. C'est à ce niveau qu'on utilise les connectivités calculées à l'étape 2 pour construire les matrices globales à partir des matrices élémentaires.

Etape 5 : *Résolution du système global :*

Le système global peut être linéaire ou non linéaire. Il définit soit un problème d'équilibre qui concerne un cas stationnaire ou statique ou un problème de valeurs critiques où il faut

déterminer les valeurs et vecteurs propres du système qui correspondent généralement aux fréquences et modes propres d'un système physique.

Un problème de propagation qui concerne le cas transitoire (non stationnaire) dans lequel il faut déterminer les variations dans le temps des variables physiques et la propagation d'une valeur initiale. Les méthodes d'intégration pas à pas sont les plus fréquentes telles que, méthode des différences finies centrales, méthode de Newmark, méthode de Wilson.

A ces méthodes doivent être associées des techniques d'itération pour traiter le cas non linéaire. La plus célèbre est la méthode de Newton Raphson.

2.4 Formulation variationnelle

Actuellement, le principe des travaux virtuels est bien connu et très répandu. Il est souvent formulé en termes d'égalité des travaux effectués par les forces extérieures et intérieures lors d'un déplacement virtuel quelconque. Ce concept est essentiel pour la résolution des équations aux dérivées partielles. En effet, les déplacements sont remplacés par une fonction arbitraire continue sur le domaine et l'équation est réécrite sous forme intégrale

2.4.1 Forme forte

Un problème classique d'équations différentielles gouvernant un système physique s'énonce comme suit :

Trouver une fonction $u \in V$; V espace des fonctions, telle que :

$$A(u) = 0 \mid \Omega \quad ; \quad B(u) = 0 \mid \Gamma \qquad (2.1)$$

où $A(u)$ est l'ensemble d'équations gouvernantes définies sur le domaine Ω et $B(u)$ est l'ensemble des conditions aux limites que les fonctions u doivent vérifier sur le contour Γ (fig.3.2). La fonction u peut être un scalaire tel que la température, la pression, … ou un vecteur tel que le déplacement, la vitesse, …

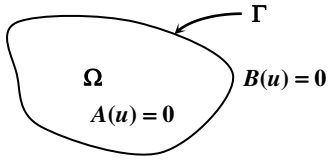


Figure 2.2 : **domaine géométrique et contour**

Le problème variationnel associé au système (2.1) s'écrit en prenant l'intégrale du système d'équations gouvernantes pondérées par des fonctions poids, l'énoncé devient :

Trouver $u \in V$ tel que :

$$\forall w \in V \ : \ \int_{\Omega} w\, A(u)\, d\Omega = 0 \qquad (2.2)$$

Cette équation est appelée forme intégrale forte de l'équation différentielle (ou du système d'équations différentielles). Elle est analogue à l'expression des travaux virtuels. En fait la solution de (2.2) a encore plus de portée, on peut affirmer que si elle est satisfaite pour toute fonction poids w, alors l'équation différentielle (2.1) est satisfaite en tout point du domaine Ω.

(i) Exemple

On considère l'équation différentielle du second ordre suivante :

$$A(u) \ = \ \frac{d^2u}{dx^2} + 1 - x = 0 \qquad ; \qquad 0 \le x \le 1 \qquad (2.3)$$

Définie dans le domaine unidimensionnel $\Omega = [0, L]$ avec les conditions aux limites :

$$u(\, x = 0\,) = \ 0 \quad \text{et} \quad u(\, x = 1\,) = \ 0 \qquad (2.4)$$

Dans ce cas $B(u)$ est l'ensemble des valeurs imposées aux deux bords du domaine. En unidimensionnel, Γ se réduit à deux points. Le forme intégrale associée à l'équation $A(u)$ s'écrit :

$$\int_{\Omega} w\,(\frac{d^2u}{dx^2} + 1 - x)\, d\Omega = 0 \ ; \quad \int_0^1 w\, \frac{d^2u}{dx^2}\, dx = \int_0^1 w\,(x-1)\, dx \qquad (2.5)$$

Avec la forme des termes à intégrer, on voit que la recherche de solutions approximatives pour la fonction inconnue u, requiert l'utilisation de polynômes ou de fonctions d'interpolation dérivables au moins deux fois. De plus les conditions aux limites doivent être vérifiées à part puisqu'elles n'interviennent pas dans l'équation intégrale ci-dessus, d'où l'introduction de la forme intégrale faible.

2.4.2 Forme faible

Pour satisfaire les conditions aux limites nous avons deux manières de procéder; soit par le choix de la fonction de pondération, soit vérifier que :

$$\int_{\Gamma} w\, B(u)\, d\Gamma = 0 \tag{2.6}$$

Dans la pratique, il est possible d'intégrer (2.2) par partie et de la remplacer par :

$$\int_{\Omega} C(w)\, D(u)\, d\Omega + \int_{\Gamma} E(w)\, F(u)\, d\Gamma = 0 \tag{2.7}$$

Les opérateurs *C*, *D*, *E* et *F* contiennent des dérivées d'ordre moins élevé, d'où un choix de fonctions d'approximation de *u* plus large.

Cette équation est la formulation faible de l'équation différentielle, elle forme la base de l'approximation par éléments finis.

Remarque : Pour obtenir de telle formulation intégrales, nous disposons de deux procédés : le premier est la méthode des résidus pondérés connue sous le nom de la méthode de Galerkin, le deuxième est la détermination de fonctionnelles variationnelles que l'on cherche à rendre stationnaire.

Dans la pratique, il n'est pas toujours facile de trouver une fonctionnelle. Le premier procédé est plus utilisé; il consiste à choisir $w = \delta u$ fonction de Dirac (une perturbation de la fonction cherchée) et d'utiliser l'approximation nodale pour la discrétisation. *w* s'appelle aussi fonction poids d'où le mot : "pondéré".

(i) Exemple

Pour obtenir la forme variationnelle faible de l'exemple précédent (équation 2.5) on intègre par partie le premier terme.

$$-\int_0^1 \frac{dw}{dx}\frac{du}{dx}\,dx + \left[w\frac{du}{dx} \right]_0^1 = \int_0^1 w\,(x-1)\,dx \tag{2.8}$$

On voit maintenant que les conditions aux limites notamment sur les dérivées sont explicitement prises en compte dans l'équation. Les valeurs imposées à la fonction elle-même seront traitées lors de la résolution des systèmes discrets.

2.5 Discrétisation du domaine

La méthode des éléments finis est une méthode d'approximation par sous domaines, donc avant toute application il faut diviser le domaine à étudier en éléments. Chaque élément est défini géométriquement par un nombre de noeuds bien déterminé qui constituent en général ses sommets. (Fig. 2.3)

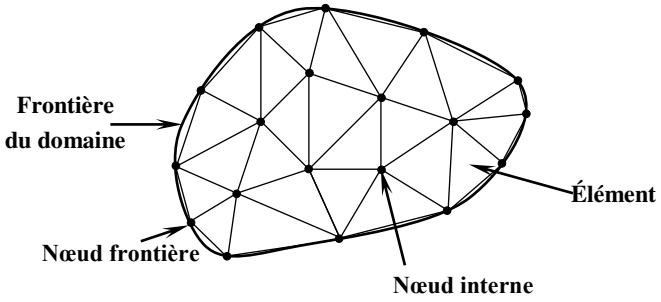


Figure 2.3 : **Discrétisation du domaine – éléments triangulaires**

La discrétisation géométrique doit respecter les règles suivantes :

20) Un nœud d'un élément ne doit pas être intérieur à un côté d'un autre du même type. (Fig. 2.4 a)

21) Aucun élément bidimensionnel ne doit être plat, éviter les angles proches de 180° ou de 0°. (Fig. 2.4 b)

22) Deux éléments distincts ne peuvent avoir en commun que des points situés dans leurs frontières communes ; le recouvrement est exclu. (Fig. 2.4 c)

23) L'ensemble de tous éléments doit constituer un domaine aussi proche que possible du domaine donné ; les trous entre éléments sont exclus. (Fig. 2.4 d)

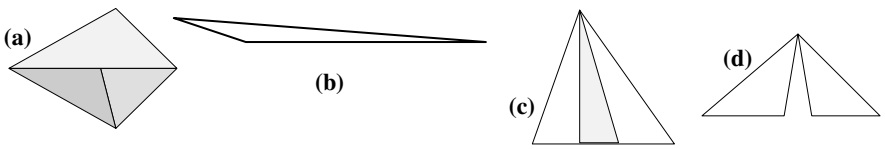


Figure 2.4 : **Règles de discrétisation**

Le résultat du procédé de discrétisation doit contenir deux données essentielles qui sont les coordonnées des noeuds et les connectivités des éléments. On doit numéroter tous les noeuds et les éléments de façon à avoir des matrices globales à petite largeur de bande, pour cela, la numérotation se fait selon la plus petite largeur du domaine.

2.6 Approximation sur l'élément

Après avoir défini l'élément, on peut remplacer la fonction exacte par une approximative. On utilise souvent des polynômes ou des fonctions faciles à mettre en oeuvre sur ordinateur.

2.6.1 Approximation polynomiale et approximation nodale

La fonction approchée est exprimée, dans le cas unidimensionnel, par :

$$u = \sum_{i=0}^{n} a_i x^i \tag{2.9}$$

Qu'on peut écrire sous la forme matricielle suivante :

$$u = <1 \;\; x \;\; x^2 \;\; \cdots > \begin{Bmatrix} a_0 \\ a_0 \\ \vdots \end{Bmatrix} \equiv < P(x) > \{A\} \tag{2.10}$$

Cette forme d'approximation est appelée interpolation polynomiale. Si on exprime la fonction sur tous les noeuds on obtient pour chaque nœud *i* de coordonnée x_i :

$$u_i = < P(x_i) > \{A\} \equiv \sum_j P_{ij} \; a_j \tag{2.11}$$

Soit pour tous les nœuds :

$$\{U\} = \begin{bmatrix} <P_{1j}> \; \{a_j\} \\ <.....> \; \{..\} \\ <F_{nj}> \; \{a_j\} \end{bmatrix} \equiv U_n = P_n \, a_n \tag{2.12}$$

U_n : représente les valeurs aux nœuds de la fonction.
P_n : valeurs des polynômes aux nœuds de coordonnées x_i.
a_n : variables généralisées qui sont les facteurs des polynômes.

L'inconvénient de cette interpolation réside dans l'utilisation des paramètres a_i comme variable de base, des calculs supplémentaires sont nécessaires pour calculer la fonction recherchée u. Afin d'éviter ces calculs, on peut mettre les valeurs de la fonction u au nœuds comme variables de base en procédant comme suit :

A partir de l'équation (3.12), on peut tirer les a_n en fonction des u_n et on les remplace dans l'équation (3.10). Ce qui donne :

$$u = < P(x) > P_n^{-1} U_n \equiv < N(x) > U_n \tag{2.13}$$

C'est la forme la plus utilisée par le fait que ses variables sont les valeurs de la fonction aux noeuds, la résolution donne directement ces valeurs.

Ce type d'approximation est appelée interpolation nodale, les fonctions N_i sont appelées fonction de forme, elles sont fonction du type d'élément utilisé pour la discrétisation géométrique.

chapitre 3

Equations différentielles et problèmes aux limites

Problème d'évolution et problème aux limites

3.1 Equation différentielles du premier ordre

3.2 Etapes à suivre

Dans le cas des équations différentielles, le problème est formulé directement sous forme mathématique et revient à déterminer une fonction inconnue *u* définie dans un domaine Ω et régie par une équation différentielle avec des conditions aux limites

3.2.1 Formulation du problème

On prends comme exemple l'équation différentielle ordinaire d'ordre un suivante :

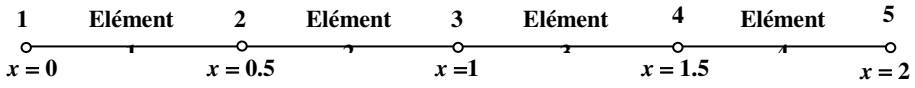
$$\frac{du}{dx} + 2x(u-1) = 0 \quad \text{avec} \quad u(0) = 0$$

Dans ce problème, $\Omega = [0, 2]$ est un domaine de dimension 1 ; sa frontière $\partial\Omega$ se réduit à deux points : 0 et 2.

La solution exacte de cette équation est : $u_e = 1 - e^{-x^2}$

3.2.2 Discrétisation du domaine

Le domaine Ω est divisé en n segments (appelés éléments) de taille 1/n. Chaque élément contient deux nœuds sur lesquelles la fonction *u* est interpolée.



La division du domaine Ω en plusieurs éléments est appelée maillage. On utilise deux tableaux pour la description du maillage : tableau de connectivités des éléments et tableau des coordonnées des nœuds. Pour une exemple de trois éléments on obtient les deux tableaux comme suit :

Tableau des connectivités		
Elément	Nœud 1 (début)	Nœud 2 (fin)
1	1	2
2	2	3
3	3	4
4	4	5

Tableau des coordonnées	
Nœud	Coordonnée (*x*)
1	0.0
2	0.5
3	1.0
4	1.5
5	2.0

Les lignes de commandes MATLAB qui permettent d'obtenir les deux tableaux sont :

```
n = 4;                    % nombre d'éléments
x = 0:2/n:2;              % coordonnées des nœuds

for i = 1:n               % début de boucle sur les éléments
    t(i,:) = [i, 1+i];    % connectivite de chaque élément
end;                      % fin de boucle
```

ou bien, sous forme plus compacte :

```
x = [0:2/n:2]'
t([1:n],:) = [[1:n]',[2:n+1]']
```

Remarque

Pour un nombre plus élevé d'éléments il suffit d'augmenter la valeur de `n` .

`x(i)` donne la coordonnée du nœud `i` ; `x(2)` affiche : `0.5`

`t(i,:)` donne les deux nœuds dé l'élément `i` ; `t(2,:)` affiche : `2 3`

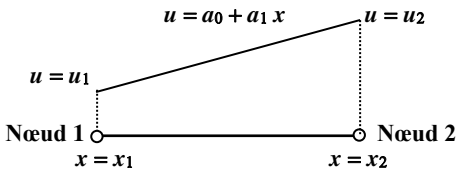
`t(:,i)` donne la colonne `i` du tableau `t` ; `t(:,2)` affiche :

```
2
3
4
5
```

3.2.3 Discrétisation et interpolation sur l'élément

On peut interpoler la fonction u recherchée dans un élément par un polynôme. L'ordre du polynôme conditionne la précision de la solution approchée. Pour un élément à deux nœuds on peut prendre :

$$u = a_0 + a_1 x \tag{3.}$$



soit sous forme vectorielle :

$$u = < 1 \quad x > \begin{Bmatrix} a_0 \\ a_1 \end{Bmatrix} \equiv u = p\, a_n$$

avec p vecteur ligne contenant les monômes x^n

et a_n vecteur colonne contenant les facteurs du polynôme.

Cette approximation de la fonction inconnue u est appelée interpolation polynomiale, elle est fonction de a_0 et a_1 qui sont des coefficients sans valeur physique. Pour utiliser les valeurs de u aux nœuds on cherche une interpolation en fonction de u_1 et u_2. L'interpolation polynomiale aux nœuds s'écrit :

$$\begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \end{Bmatrix} \quad \equiv \quad u_n = P_n \; a_n$$

L'inverse de ce système d'équations donne les paramètres a_n.

$$a_n = P_n^{-1} \, U_n \quad \equiv \quad \begin{Bmatrix} a_0 \\ a_1 \end{Bmatrix} = \frac{1}{x_2 - x_1} \begin{bmatrix} x_2 & -x_1 \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix}$$

En remplaçant les a_n on peut maintenant approcher la fonction u par :

$$u = <1 \quad x> \frac{1}{x_2 - x_1} \begin{bmatrix} x_2 & -x_1 \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = < \frac{x_2 - x}{x_2 - x_1} \quad \frac{x - x_1}{x_2 - x_1} > \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix}$$

$$u = N \; U_n$$

avec N est un vecteur ligne contenant des fonctions de x appelées fonctions de forme.

Cette interpolation est appelée interpolation nodale puisqu'elle dépend des valeurs aux nœuds de la fonction inconnue u.

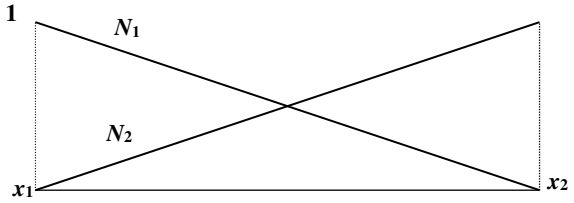
3.2.4 Propriétés des fonctions de forme

Il est intéressant de relever les propriétés suivantes pour les fonctions de forme N :

$$N_1(x) = \frac{x_2 - x}{x_2 - x_1} = \begin{cases} 1 & x = x_1 \\ 0 & x = x_2 \end{cases} \; ; \quad N_2(x) = \frac{x - x_1}{x_2 - x_1} = \begin{cases} 1 & x = x_2 \\ 0 & x = x_1 \end{cases}$$

$$N_1(x) + N_2(x) = 1 \quad \forall x \in [x_1 \, ; x_2]$$

1) Elles prennent la valeur unité aux nœuds de même indice et la valeur nulle aux autre nœuds.
2) Leur somme est égale à l'unité sur tout l'intervalle de l'élément :



Remarque

Les deux fonctions de forme peuvent s'écrire sous forme des polynômes de Jacobi :

$$N_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

3.2.5 Elément de type Lagrange

Ainsi les éléments pour lesquels les fonctions de forme sont des polynômes de Jacobi sont appelés éléments de type Jacobi. On peut construire directement, sans passer par l'interpolation polynomiale, des fonctions de forme pour des éléments d'ordre supérieur (plus de deux nœuds).

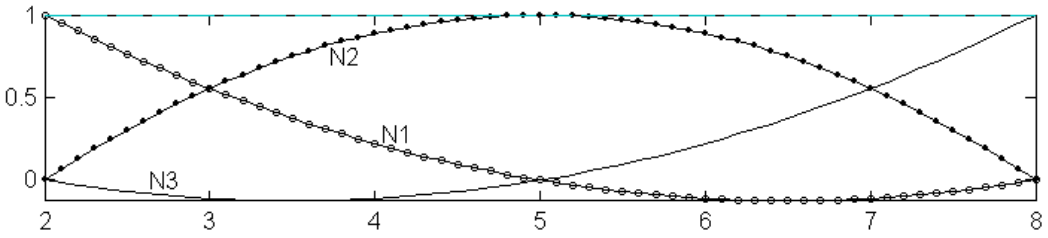
Par exemple les fonctions de forme de l'élément à trois nœuds sont :

$$N_1(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} \ ;$$

$$N_2(x) = \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} \ ;$$

$$N_3(x) = \frac{(x - x_1)(x - x_2)}{(x_2 - x_1)(x_3 - x_2)}$$

On peut vérifier aisément sur la figure ci-dessous les propriétés aux nœuds des trois fonctions ainsi que leur somme sur l'élément.



$$N_1(x)=\begin{cases}1 & x=x_1\\ 0 & x=x_2\\ 0 & x=x_3\end{cases};\ N_2(x)=\begin{cases}0 & x=x_1\\ 1 & x=x_2\\ 0 & x=x_3\end{cases};\ N_3(x)=\begin{cases}0 & x=x_1\\ 0 & x=x_2\\ 1 & x=x_3\end{cases}$$

Le scripte MATLAB qui permet de calculer et de tracer ces trois fonctions est le suivant :

```
x1 = 2;
x2 = 5;
x3 = 8;
x = x1:0.1:x3;
N1 = (x-x2).*(x-x3)/(x1-x2)/(x1-x3);
N2 = (x-x1).*(x-x3)/(x2-x1)/(x2-x3);
N3 = (x-x1).*(x-x2)/(x3-x1)/(x3-x2);
S = N1+N2+N3;
plot(x,N1,x,N2,x,N3,x,S)
```

Noter qu'il préférable d'utiliser un fichier fonction d'usage plus général.

3.2.6 Matrices élémentaires

Le calcul des matrices élémentaires passe par la réécriture du problème sous forme intégrale :

$$\int_\Omega \delta u\left[\frac{du}{dx}+2x(u-1)\right]d\Omega=0$$

avec δu est une fonction de pondération prise égale à une perturbation de la fonction inconnue u.

Le domaine Ω comprend l'intervalle 0 à 2, $d\Omega = dx$ et avec l'interpolation nodale on a :

$$\frac{du}{dx}=\frac{dN}{dx}U_n\ ;\ \text{et}\ \delta u = N\,\delta U_n$$

Puisque seules les fonctions N dépendent de x et les perturbations ne touchent que les valeurs de u.

Pour commodité on écrit : $\delta u = (\delta N_n)^T N^T$

L'intégrale de 0 à 2 peut être remplacée par la somme des intégrales de x_i à x_{i+1} (ou bien : l'intégrale sur Ω est la somme des intégrales sur Ω_e, avec Ω_e est le domaine de chaque élément)

$$\int_{\Omega} = \sum_{éléments} \int_{\Omega_e} \equiv \int_0^2 = \sum_{i=1,n} \int_{x_i}^{x_{i+1}} = \sum_{éléments} \int_{x_1}^{x_2}$$

La forme intégrale de l'équation différentielle devient alors pour chaque élément :

$$\int_{x_1}^{x_2} \left[\delta u \frac{du}{dx} \right] dx + \int_{x_1}^{x_2} \left[\delta u \, 2x \, u \right] dx - \int_{x_1}^{x_2} \left[\delta u \right] dx = 0$$

$$\int_{x_1}^{x_2} \left[\delta U_n^T N^T \frac{dN}{dx} U_n \right] dx + \int_{x_1}^{x_2} \left[\delta U_n^T N^T 2x \, N \, U_n \right] dx - \int_{x_1}^{x_2} \left[\delta U_n^T N^T \right] 2x \, dx = 0$$

Cette écriture discrétisée est valable pour tous les types d'éléments. Dans le cas particulier d'un élément linéaire à deux nœuds, elle s'écrit comme suit :

$$< \delta u_1 \quad \delta u_2 > \int_{x_1}^{x_2} \begin{Bmatrix} N_1 \\ N_2 \end{Bmatrix} < dN_1 \quad dN_2 > dx \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} +$$

$$< \delta u_1 \quad \delta u_2 > \int_{x_1}^{x_2} \begin{Bmatrix} N_1 \\ N_2 \end{Bmatrix} 2x < N_1 \quad N_2 > dx \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} -$$

$$< \delta u_1 \quad \delta u_2 > \int_{x_1}^{x_2} \begin{Bmatrix} N_1 \\ N_2 \end{Bmatrix} 2x \, dx = 0$$

On voit qu'il est possible de simplifier $(\delta u_n)^T$ puisqu'il n'est pas nul et revient à chaque terme. Au fait c'est les autres termes qui doivent être nuls ! on le voit bien avec cet exemple :

$$< \delta u_1 \quad \delta u_2 > \begin{Bmatrix} v_1 \\ v_2 \end{Bmatrix} + < \delta u_1 \quad \delta u_2 > \begin{Bmatrix} w_1 \\ w_2 \end{Bmatrix} = 0$$

donne : $\delta u_1 \, v_1 + \delta u_2 \, v_2 + \delta u_1 \, w_1 + \delta u_2 \, w_2 = 0$; soit : $v_1 + w_1 = 0$

et $v_2 + w_2 = 0$; ou : $\begin{Bmatrix} v_1 \\ v_2 \end{Bmatrix} + \begin{Bmatrix} w_1 \\ w_2 \end{Bmatrix} = 0$.

Finalement l'équation intégrale discrétisée se met sous la forme matricielle :

$$\left(\int_{x_1}^{x_2} \begin{Bmatrix} N_1 \\ N_2 \end{Bmatrix} < dN_1 \quad dN_2 > dx + \int_{x_1}^{x_2} \begin{Bmatrix} N_1 \\ N_2 \end{Bmatrix} 2x < N_1 \quad N_2 > dx \right) \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} -$$

$$\int_{x_1}^{x_2} \begin{Bmatrix} N_1 \\ N_2 \end{Bmatrix} 2x\, dx \;=\; 0$$

$$\left(\int_{x_1}^{x_2} \begin{bmatrix} N_1 dN_1 & N_1 dN_2 \\ N_2 dN_1 & N_2 dN_2 \end{bmatrix} dx + \int_{x_1}^{x_2} \begin{bmatrix} N_1 2xN_1 & N_1 2xN_2 \\ N_2 2xN_1 & N_2 2xN_2 \end{bmatrix} dx \right) \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \int_{x_1}^{x_2} \begin{Bmatrix} 2x\, N_1 \\ 2x\, N_2 \end{Bmatrix} dx$$

Qui est un système d'équations linéaires *Ke Ue = Fe* ; avec *Ke* et *Fe* sont appelés matrice et vecteur élémentaires du système d'équation. Dans le cas de la présente équation différentielle *K* est la somme de deux matrices : $Ke = Ke_1 + Ke_2$ tel que :

$$Ke_1 = \int_{x_1}^{x_2} \begin{bmatrix} N_1 dN_1 & N_1 dN_2 \\ N_2 dN_1 & N_2 dN_2 \end{bmatrix} dx \;;\quad Ke_2 = \int_{x_1}^{x_2} \begin{bmatrix} N_1 2xN_1 & N_1 2xN_2 \\ N_2 2xN_1 & N_2 2xN_2 \end{bmatrix} dx$$

En remplaçant les fonctions de forme et leurs dérivées par leurs expressions respectives on obtient :

$$Ke_1 = \int_{x_1}^{x_2} \begin{bmatrix} \dfrac{x-x_2}{x_1-x_2}\dfrac{1}{x_1-x_2} & \dfrac{x-x_2}{x_1-x_2}\dfrac{1}{x_2-x_1} \\ \dfrac{x-x_1}{x_2-x_1}\dfrac{1}{x_1-x_2} & \dfrac{x-x_2}{x_2-x_1}\dfrac{1}{x_2-x_1} \end{bmatrix} dx\,;$$

$$Ke_2 = \int_{x_1}^{x_2} \begin{bmatrix} \dfrac{x-x_2}{x_1-x_2}2x\dfrac{x-x_2}{x_1-x_2} & \dfrac{x-x_2}{x_1-x_2}2x\dfrac{x-x_1}{x_2-x_1} \\ \dfrac{x-x_1}{x_2-x_1}2x\dfrac{x-x_2}{x_1-x_2} & \dfrac{x-x_1}{x_2-x_1}2x\dfrac{x-x_1}{x_2-x_1} \end{bmatrix} dx$$

Soit après intégration des composantes des deux matrices :

$$Ke_1 = \frac{1}{2}\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix};$$

$$Ke_2 = \frac{x_2 - x_1}{6}\begin{bmatrix} 3x_1 + x_2 & x_1 + x_2 \\ x_1 + x_2 & x_1 + 3x_2 \end{bmatrix}$$

Le vecteur *F* est donné par :

$$Fe = \int_{x_1}^{x_2} \begin{Bmatrix} 2x\,(x - x_2)/(x_1 - x_2) \\ 2x\,(x - x_1)/(x_2 - x_1) \end{Bmatrix} dx$$

soit :

$$Fe = \frac{x_2 - x_1}{3}\begin{Bmatrix} 2x_1 + x_2 \\ x_1 + 2x_2 \end{Bmatrix}$$

3.2.7 Remarque

Il est possible d'utiliser MATLAB pour intégrer analytiquement les matrices élémentaires, Le script peut être le suivant :

```
syms x x1 x2 real                          % déclaration de variables
symboliques
N   = [(x-x2)/(x1-x2)  (x-x1)/(x2-x1)] % fonctions de forme
dN  = simple(diff(N,x))                    % dérivées des fonctions de
forme
Ke1 = simple( int(N' * dN      , x, x1, x2) )   % matrice Ke1
Ke2 = simple( int(N' * 2*x * N , x, x1, x2) )   % matrice Ke2
Fe  = simple( int(N' * 2*x     , x, x1, x2) )   % vecteur Fe
```

3.3 Etape 4 : Assemblage

Le calcul des matrices élémentaires permet d'obtenir pour les quatre éléments les systèmes d'équations élémentaires suivants :

Elément 1 : $x_1 = 0$; $x_2 = ½$;

$$Ke_1^{(1)} = \begin{bmatrix} -½ & ½ \\ -½ & ½ \end{bmatrix} ;\ Ke_2^{(1)} = \frac{½ - 0}{6}\begin{bmatrix} 3\times 0 + ½ & 0 + ½ \\ 0 + ½ & 0 + 3\times ½ \end{bmatrix} ;$$

$$Ke^{(1)} = Ke_1^{(1)} + Ke_2^{(1)} = \frac{1}{24}\begin{bmatrix} -11 & 13 \\ -11 & 15 \end{bmatrix} = \begin{bmatrix} -0.4583 & 0.5417 \\ -0.4583 & 0.6250 \end{bmatrix} ;$$

$$Fe^{(1)} = \frac{½ - 0}{3}\begin{Bmatrix} 2\times 0 + ½ \\ 0 + 2\times ½ \end{Bmatrix} = \frac{1}{12}\begin{Bmatrix} 1 \\ 2 \end{Bmatrix} = \begin{Bmatrix} 0.0833 \\ 0.1667 \end{Bmatrix}$$

$$Ke^{(1)}\, Ue^{(1)} = Fe^{(1)} \; ; \; \frac{1}{24}\begin{bmatrix} -11 & 13 \\ -11 & 15 \end{bmatrix}\begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \frac{1}{12}\begin{Bmatrix} 1 \\ 2 \end{Bmatrix}$$

On notant par U les valeurs de la fonction u aux cinq nœuds, les valeurs du vecteur élémentaire $Ue^{(1)}$ de l'élément 1 correspondent aux composantes u_1 et u_2 vecteur global U, celles de $Ue^{(2)}$ de l'élément 2 correspondent à la seconde et troisième du vecteur global U, celles de $Ue^{(3)}$ à la troisième et quatrième et celles de $Ue^{(4)}$ à la quatrième et cinquième.

Elément 2 : $x_1 = ½$; $x_2 = 1$;

$$Ke^{(2)} = \frac{1}{24}\begin{bmatrix} -7 & 15 \\ -9 & 19 \end{bmatrix} = \begin{bmatrix} -0.2917 & 0.6250 \\ -0.3750 & 0.7917 \end{bmatrix} \; ; \; Fe^{(2)} = \frac{1}{12}\begin{Bmatrix} 4 \\ 5 \end{Bmatrix} = \begin{Bmatrix} 0.3333 \\ 0.4167 \end{Bmatrix}$$

$$Ke^{(2)}\, Ue^{(2)} = Fe^{(2)} \; ; \; \frac{1}{24}\begin{bmatrix} -7 & 15 \\ -9 & 19 \end{bmatrix}\begin{Bmatrix} u_2 \\ u_3 \end{Bmatrix} = \frac{1}{12}\begin{Bmatrix} 4 \\ 5 \end{Bmatrix}$$

Elément 3 : $x_1 = 1$; $x_2 = {}^3/_2$;

$$Ke^{(3)} = \frac{1}{24}\begin{bmatrix} -3 & 17 \\ -7 & 23 \end{bmatrix} = \begin{bmatrix} -0.1250 & 0.7083 \\ -0.2917 & 0.9583 \end{bmatrix} \; ; \; Fe^{(3)} = \frac{1}{12}\begin{Bmatrix} 7 \\ 8 \end{Bmatrix} = \begin{Bmatrix} 0.5833 \\ 0.6667 \end{Bmatrix}$$

$$Ke^{(3)}\, Ue^{(3)} = Fe^{(3)} \; ; \; \frac{1}{24}\begin{bmatrix} -3 & 17 \\ -7 & 23 \end{bmatrix}\begin{Bmatrix} u_3 \\ u_4 \end{Bmatrix} = \frac{1}{12}\begin{Bmatrix} 7 \\ 8 \end{Bmatrix}$$

Elément 4 : $x_1 = {}^3/_2$; $x_2 = 2$;

$$Ke^{(4)} = \frac{1}{24}\begin{bmatrix} 1 & 19 \\ -5 & 27 \end{bmatrix} = \begin{bmatrix} 0.0417 & 0.7917 \\ -0.2083 & 1.1250 \end{bmatrix} \; ; \; Fe^{(4)} = \frac{1}{12}\begin{Bmatrix} 10 \\ 11 \end{Bmatrix} = \begin{Bmatrix} 0.8333 \\ 0.9167 \end{Bmatrix}$$

$$Ke^{(4)}\,Ue^{(4)} = Fe^{(4)} \;;\; \frac{1}{24}\begin{bmatrix} 1 & 19 \\ -5 & 27 \end{bmatrix}\begin{Bmatrix} u_4 \\ u_5 \end{Bmatrix} = \frac{1}{12}\begin{Bmatrix} 10 \\ 11 \end{Bmatrix}$$

En réécrivant les systèmes élémentaires en fonction de toutes les composantes de U on obtient :

Elément 1 :
$$\frac{1}{24}\begin{bmatrix} -11 & 13 & 0 & 0 & 0 \\ -11 & 15 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}\begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{Bmatrix} = \frac{1}{12}\begin{Bmatrix} 1 \\ 2 \\ 0 \\ 0 \\ 0 \end{Bmatrix}$$

Elément 2 :
$$\frac{1}{24}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -7 & 15 & 0 & 0 \\ 0 & -9 & 19 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}\begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{Bmatrix} = \frac{1}{12}\begin{Bmatrix} 0 \\ 4 \\ 5 \\ 0 \\ 0 \end{Bmatrix}$$

Elément 3 :
$$\frac{1}{24}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -3 & 17 & 0 \\ 0 & 0 & -7 & 23 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}\begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{Bmatrix} = \frac{1}{12}\begin{Bmatrix} 0 \\ 0 \\ 7 \\ 8 \\ 0 \end{Bmatrix}$$

Elément 4 :
$$\frac{1}{24}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 19 \\ 0 & 0 & 0 & -5 & 27 \end{bmatrix}\begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{Bmatrix} = \frac{1}{12}\begin{Bmatrix} 0 \\ 0 \\ 0 \\ 10 \\ 11 \end{Bmatrix}$$

En prenant maintenant la somme (≡ somme des intégrales), le système global s'écrit enfin :

$$\frac{1}{24}\begin{bmatrix} -11 & 13 & 0 & 0 & 0 \\ -11 & 15-7 & 15 & 0 & 0 \\ 0 & -9 & 19-3 & 17 & 0 \\ 0 & 0 & -7 & 23+1 & 19 \\ 0 & 0 & 0 & -5 & 27 \end{bmatrix}\begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{Bmatrix} = \frac{1}{12}\begin{Bmatrix} 1 \\ 2+4 \\ 5+7 \\ 8+10 \\ 11 \end{Bmatrix} \equiv K\,U = F$$

On appel cette opération *assemblage*. En pratique, on ne procède pas de cette manière pour des raisons d'économie de mémoire et de temps de calcul mais on fait un assemblage des matrices élémentaires en utilisant les connectivités des éléments. La matrice globale *K* est d'abord initialisée à la matrice nulle, ensuite à chaque construction de matrice élémentaire, on localise avec une table là où il faut l'ajouter à la matrice globale. Dans le cas d'un degré de liberté par nœud (ce cas présent) la table de localisation correspond à la table des connectivités. Exemple ; pour l'élément 1 la table de localisation est t = [1 2] on ajoute donc la matrice $Ke^{(1)}$ à $K(1,1)$, $K(1,2)$, $K(2,1)$ et $K(2,2)$; et pour l'élément 2 la table de localisation est t = [2 3] on ajoute la matrice $Ke^{(2)}$ à $K(2,2)$, $K(2,3)$, $K(3,2)$ et $K(3,3)$. Et ainsi de suite.

Le script MATLAB qui permet de faire ce type d'assemblage est simple :

```
K = zeros(n+1);                          % initialisation de la matrice
globale (n+1 nœuds)
F = zeros(n+1,1) ;                       % initialisation du vecteur
global (remarquer ,1)
for i = 1:n                              % boucle sur les éléments
    t = [i i+1];                         % table de localisation de
chaque élément
    x1 = x(i); x2 = x(i+1);              % coordonnées des deux nœuds
de l'élément
    [Ke,Fe] = MatElt2Nd(x1,x2);          % Fonction pour calculer la
mat. et vect. élementairs
    K(t,t)  = K(t,t) + Ke;               % Assemblage de la matrice
globale
    F(t)    = F(t)   + Fe;               % Assemblage du vecteur global
end;                                     % fin de boucle
```

3.4 Etape 5a : Résolution – Application des CAL

Avant de résoudre le système il faut appliquer les conditions aux limites de la fonction u. Au nœud 1 de coordonnée $x = 0$, $u = 0$; ce qui se traduit par $u_1 = 0$. Ceci induit la réduction du nombre d'équations total à résoudre. Le système global devient alors :

$$\frac{1}{24}\begin{bmatrix} 15-7 & 15 & 0 & 0 \\ -9 & 19-3 & 17 & 0 \\ 0 & -7 & 23+1 & 19 \\ 0 & 0 & -5 & 27 \end{bmatrix}\begin{Bmatrix} u_2 \\ u_3 \\ u_4 \\ u_5 \end{Bmatrix} = \frac{1}{12}\begin{Bmatrix} 2+4 \\ 5+7 \\ 8+10 \\ 11 \end{Bmatrix}$$

La ligne et la colonne d'indice 1 qui correspondent à la valeur u_1 ont été supprimées de la matrice K et du vecteur F. Si $u(0) = a \neq 0$ ($u_1 = a$) alors on retranche de F le produit de la 1ère colonne de K par a.

La commande MATLAB qui permet de supprimer une ligne ou une colonne d'une matrice consiste à lui affecter un vecteur nul (vide) :

```
A(i, :) = []                          % supprime la ligne i de la
matrice A
A(:, i) = []                          % supprime la colonne i
V(i) =[]                              % supprime la composante i du
vecteur V
```

Pour appliquer la condition aux limites $u_1 = 0$, il suffit d'écrire :

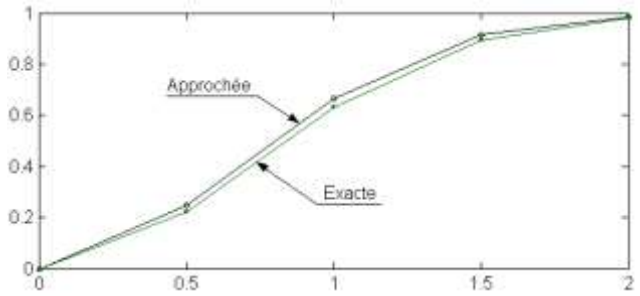
```
K(1,:) = [];                          % suppression de la 1ere ligne
de K
K(:,1) = [];                          % suppression de la 1ere
colonne de K
F(1,:) = [];                          % suppression de la 1ere
composante de F
```

Remarque : pour $u(0) = a$, on insère, avant les suppressions, la commande : `F = F - K(:,1)*a`.

3.5 Etape 5b : Résolution – Calcul de la solution

La solution peut être maintenant calculer par un des algorithme de résolution de système linéaires. Notons que dans ce cas la matrice K est tridiagonale, mais avec MATLAB il suffit d'utiliser la division gauche `U = K \ F`. Les résultats sont donnés dans le tableau et la figure suivants :

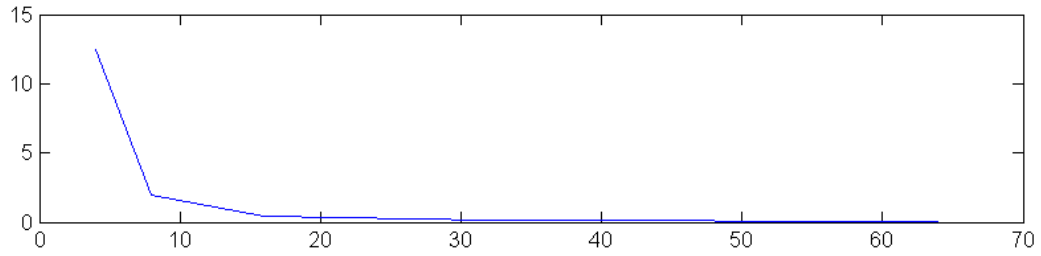
Coord. x	Solution Exacte	Solution MEF	Erreur (%)
0.5000	0.2212	0.2488	12.4555
1.0000	0.6321	0.6673	5.5705
1.5000	0.8946	0.9154	2.3225
2.0000	0.9817	0.9843	0.2694



3.6 Etude de la convergence

Toute étude qui passe par une solution numérique approchée doit faire l'objet d'un examen de convergence. Dans les études par éléments finis la convergence peut être atteinte en augmentant le degré des polynômes d'interpolation (nombre de nœuds par élément), en utilisant un nombre d'éléments assez grand avec un raffinement du maillage ou bien en cherchant une bonne adaptation du maillage au problème traité. La figure si après donne le pourcentage d'erreur relative au point $x = 0.5$ en fonction du nombre n d'éléments utilisés. On voit que le décroissement de l'erreur relative est une fonction de

type $^1/_n$ et tends à s'annuler d'une manière monotone. Un nombre d'éléments $n \approx 30$ suffit pour avoir une erreur acceptable.



Le programme MATLAB suivant regroupe toutes les étapes précédentes.

```
function [U, Ue, Err, x] = ExempleEquaDiff(n)
%
% du/dx - 2*x*(1-u) = 0
% u(0) = 0
%
%-------------------------------------------------------
% Solution avec des fonctions d'interpolation à 2Nds
%-------------------------------------------------------
 x = [0:2/n:2]';                        % vecteur des coordonnées
 K = zeros(n+1  ) ;                     % initialisations
 F = zeros(n+1,1) ;

 for i = 1:n                            % boucle sur les éléments
     j = i+1;
     t = [i j];
     x1 = x(i);
     x2 = x(j);
     [Ke, Fe] = MatElt2Nd(x1,x2);
     K(t,t)   = K(t,t) + Ke;
     F(t)     = F(t)   + Fe;
 end;
```

```
 K(1,:) = [];                          % application des CAL
 K(:,1) = [];
 F(1,:) = [];

 U = K\F;                              % résolution

 U =[0;U];                             % incorporation de la valeur
initiale pour plot
 Ue = 1-exp(-x.^2);                    % calcul de la solution exacte
 Err = 100*(U-Ue)./Ue;                 % calcul de l'erreur relative
 plot(x,U,x,Ue)                        % trace les deux solutions
return

%-----------------------------------------
% Calcul de la matrice Ke et du vecteur Fe
%-----------------------------------------
function [Ke, Fe] = MatElt2Nd(x1,x2)   % déclaration de la fonction
 Ke1 = 1/2*[ -1   1                      % matrice Ke1
             -1   1 ] ;

 Ke2 = (x2-x1)/6* [ 3*x1+x2   x1+x2    % matrice Ke2
                    x1+x2     x1+3*x2];

 Ke = Ke1 + Ke2;                          % matrice Ke
 Fe = (x2-x1)/3 * [2*x1+x2 ; x1+2*x2]; % vecteur Fe
return
```

La commande : `[U, Ue, Err, x] = ExempleEquaDiff(30)` permet d'avoir à la fois la solution par éléments finis `U`, la solution exacte `Ue`, l'erreur relative `Err` et les coordonnées `x` avec 30 éléments.

Remarque :

Noter que ce programme permet pratiquement de résoudre toutes les équations différentielles d'ordre 1 avec la condition aux limites $u(0) = 0$. Seule la fonction `MatElt2Nd(x1,x2)` nécessite un changement pour les matrices élémentaires.

3.7 Devoir N°2

1) Modifier le programme `ExempleEquaDiff(n)` pour résoudre la même équation en utilisant des fonctions de forme quadratiques (élément à trois nœuds)
2) Ecrire une fonction MATLAB qui permet d'avoir les expressions des fonctions de forme *1D*.
3) Ecrire un programme MATLAB d'éléments finis qui permet de résoudre dans le domaine [0, 3] l'équation : $du/dx + (2x-1)\,u = 0$ avec $u(0) = 1$. Comparer avec la solution exacte : $u_e = e^{x-x^2}$ et examiner la convergence au point $x = 0.5$.

3.8 Equations différentielles du 2nd ordre

Comme exemple d'équation du 2nd ordre, on va traiter dans le domaine Ω = [0 , 1], l'équation :

$\frac{d^2u}{dx^2} + 6\frac{du}{dx} + 9\,u = x\,(1-x)$ avec les conditions aux limites : $u(0) = 0$ et $\left[du/dx\right]_{x=1} = 0$.

La solution exacte est : $u_e = \frac{1}{27}(1-x)(3x+4) + \frac{1}{54}e^{-3x}(8 + xe^3 - 12x)$

Le domaine Ω sera subdivisé (maillé) en *n* éléments linéaires à 2 nœuds. La forme intégrale forte de cette équation s'écrit :

$$\int_0^1 \delta u \frac{d^2u}{dx^2} dx + 6\int_0^1 \delta\, u \frac{du}{dx} dx + 9\int_0^1 \delta\, u\, u\; dx = \int_0^1 \delta\, u\, x(1-x)\, dx$$

Lorsqu'il est question d'équations différentielles du second ordre et plus, on est confronté à deux difficultés. L'interpolation des dérivées et la satisfaction des conditions aux limites. En effet plus l'ordre des dérivées est grand, plus fort doit être le degrés des polynômes ou des fonctions d'interpolation à utiliser. Un élément linéaire à deux nœud ne peut donc être utilisé pour ce type d'équation. De plus, la forme intégrale forte ne fait pas apparaître la condition $\left[du/dx\right]_{x=1} = 0$.

Ces deux difficultés ont conduit à l'utilisation de la forme intégrale faible qu'on peut obtenir, dans le cas présent avec intégration par partie du premier terme :

$$\int_0^1 \delta u \frac{d^2u}{dx^2} dx = -\int_0^1 \frac{\delta du}{dx}\frac{du}{dx} dx + \left[\delta u \frac{du}{dx}\right]_0^1$$

On voit immédiatement l'avantage majeur qu'offre cette expression. Elle réduit l'ordre des dérivées (d'où le terme faible) et permet de prendre en compte la condition aux limite $[du/dx]_{x=1} = 0$.

De plus puisque u (0) = 0, le terme δu s'annule aussi à x = 0 (il ne peut y avoir de perturbations dans des valeurs connues ou nulles).

Ainsi la forme intégrale s'écrit avec la prise en compte des conditions aux limites comme suit :

$$-\int_0^1 \frac{\delta du}{dx}\frac{du}{dx} dx + 6\int_0^1 \delta u \frac{du}{dx} dx + 9\int_0^1 \delta u\, u\ dx = \int_0^1 \delta u\, x(1-x)\, dx$$

Sa discrétisation donne pour les intégrales sur les éléments :

$$-\int_{x_1}^{x_2} \frac{dN^T}{dx}\frac{dN}{dx} dx + 6\int_{x_1}^{x_2} N^T \frac{dN}{dx} dx + 9\int_{x_1}^{x_2} N^T\, N\, dx = \int_{x_1}^{x_2} N^T\, x(1-x)\, dx$$

Avec cette écriture il est possible d'utiliser un élément linéaire ; la matrice élémentaire Ke est la somme de trois matrices : $Ke = Ke_1 + Ke_2 + Ke_3$, avec :

$$Ke_1 = -\int_{x_1}^{x_2} \begin{Bmatrix} 1/(x_1 - x_2) \\ 1/(x_2 - x_1) \end{Bmatrix} < 1/(x_1 - x_2) \quad 1/(x_2 - x_1) > dx \quad ; \quad Ke_1 = \frac{1}{x_2 - x_1}\begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$Ke_2 = 6\int_{x_1}^{x_2} \begin{Bmatrix} (x - x_2)/(x_1 - x_2) \\ (x - x_1)/(x_2 - x_1) \end{Bmatrix} < 1/(x_1 - x_2) \quad 1/(x_2 - x_1) > dx \quad ; \quad Ke_2 = \begin{bmatrix} -3 & 3 \\ -3 & 3 \end{bmatrix}$$

$$Ke_3 = 9\int_{x_1}^{x_2} \begin{Bmatrix} (x - x_2)/(x_1 - x_2) \\ (x - x_1)/(x_2 - x_1) \end{Bmatrix} < (x - x_2)/(x_1 - x_2) \quad (x - x_1)/(x_2 - x_1) > dx \quad ;$$

$$Ke_3 = \frac{x_2 - x_1}{2}\begin{bmatrix} 6 & 3 \\ 3 & 6 \end{bmatrix}$$

et le vecteur élémentaire Fe est :

$$Fe = \int_{x_1}^{x_2} \begin{Bmatrix} (x-x_2)/(x_1-x_2) \\ (x-x_1)/(x_2-x_1) \end{Bmatrix} x(1-x)\,dx \ ;$$

$$Fe = \frac{x_2-x_1}{12} \begin{Bmatrix} (4-3x_1-2x_2)x_1+(2-x_2)x_2 \\ (4-3x_2-2x_1)x_2+(2-x_1)x_1 \end{Bmatrix}$$

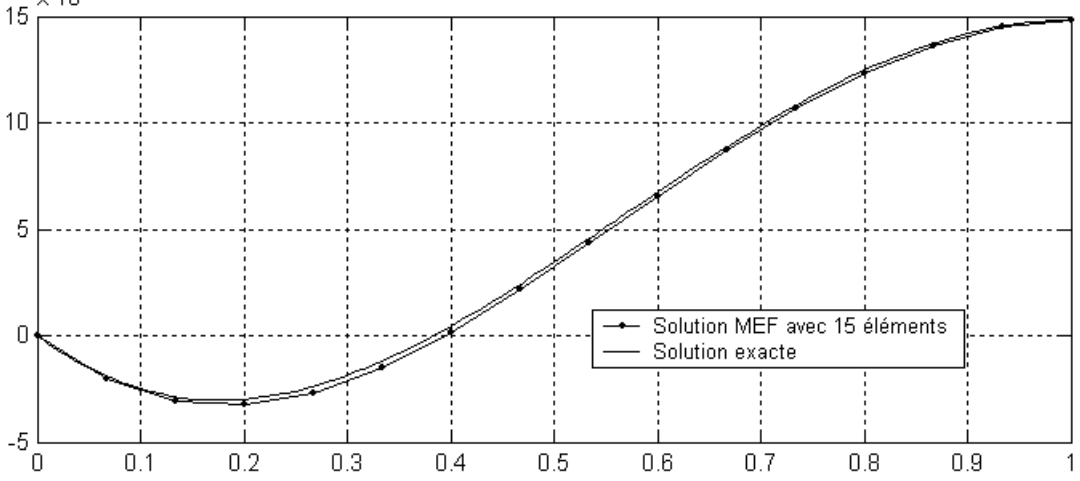
3.9 Programme MATLAB

Remarquer les modifications introduites dans le programme précédent pour traiter cette équation.

```
function [U, Ue] = EquaDiff2(n)
%----------------------------------
% d²u/dx² + 6 du/dx + 9 u = x(1-x)
% u(0) = 0   du(1)= 0
%----------------------------------
x = [0:1/n:1]';                          % modification d'1 borne
d'intégration
K = zeros(n+1  ) ;
F = zeros(n+1,1) ;
for i = 1:n
    j = i+1;
    t = [i j];
    x1 = x(i);
    x2 = x(j);
    [Ke,Fe] = MatElt2Nd(x1,x2);
    K(t,t)  = K(t,t) + Ke;
    F(t)    = F(t)   + Fe;
end;
K(1,:) = [];
K(:,1) = [];
F(1)   = [];

U = K\F;
U = [0.0;U];

t = 0:0.01:1;
Ue = 1/27*(1-t).*(3*t-4)+1/54*exp(-3*t).*(8+t*exp(3)-12*t);
```

```
plot(x,U,'-.',t,Ue)
return

%-----------------------------------------
% Calcul de la matrice Ke et du vecteur Fe
%-----------------------------------------
function [Ke,Fe] = MatElt2Nd(x1,x2)

 Ke1 = 1/(x2-x1)*[ -1    1               % les modifications ne
touchent
                   1   -1 ] ;           % essentiellement que les
matrices
 Ke2 = [ -3   3                         % élémentaires
         -3   3 ] ;
 Ke3 = (x2-x1)* [ 3   3/2
                  3/2   3];
 Ke = Ke1 + Ke2 + Ke3;
 Fe = (x2-x1)/12* [(4-3*x1-2*x2)*x1+(2-x2)*x2;
                   (4-3*x2-2*x1)*x2+(2-x1)*x1 ];
return
```

Un appel de ce programme avec la commande `EquaDiff2(15)` permet d'avoir la figure suivante :



3.10 Comparaison avec l'intégration pas à pas

Comparativement aux méthodes d'intégration pas à pas qui traitent les problèmes aux valeurs initiales (ou problèmes d'évolution), la méthode des éléments finis traite les problèmes aux valeurs aux limites. Ceci se traduit par les conditions aux limites ou conditions initiales exigées par les deux méthodes.

A titre de comparaison nous allons traiter l'équation différentielle

$$\frac{d^2u}{dx^2} + u - x = 0$$

avec les conditions aux limites : $u(0) = 0$ et $\frac{du}{dx}(4\pi) = 0$ pour la *MEF*

et les conditions initiales :

$u(0) = 0$, $\frac{du}{dx}(0) = 0$ et $\frac{d^2u}{d^2x}(0) = 0$ pour l'intégration *Pas à Pas*

La solution exacte étant $u_e = x - \sin x$. La commande MATLAB qui permet d'avoir cette solution est :

```
>> ue = simple(dsolve('D2u + u -t','u(0)=0','Du(4*pi)=0'))
```

La fonction MATLAB d'intégration pas à pas des systèmes à un degré de liberté avec la méthode d'accélération linéaire est utilisée pour résoudre numériquement l'équation. Un exposé détaillé de cette méthode peut être trouvé dans l'ouvrage de dynamique des structures de Penzien et Clough.

```
function [u,v,a] = SdofTH(m,c,k,acc,dt)
% [u,v,a] = SdofTH(m,c,k,acc,dt)
% u, v, a : histoire de la réponse en déplacements u,
%           vitesses v et accélérations a d'un système à 1DDL
% m, c, k : masse, amortissement et rigidité
% acc     : accélération à la base en vecteur
% dt      : pas de temps
%
% A. Seghir, 19/08/04

  u(1) = 0.0;
  v(1) = 0.0;
  a(1) = - acc(1);
```

```
n = length(acc);
K = k + 6/dt^2 * m + 3/dt * c;
for i = 1:n-1
   A = 6/dt * v(i) + 3.0  * a(i);
   B = 3.0  * v(i) + dt/2 * a(i);
   P = m * ( acc(i) - acc(i+1) + A ) + B * c;
   du = P/K;
   dv = 3/dt * du - 3.0 * v(i) - dt/2 * a(i);
   u(i+1) = u(i) + du;
   v(i+1) = v(i) + dv;
   P = - c * v(i+1) - k * u(i+1) - m * acc(i+1);
   a(i+1) = P/m;
end
return
```

Les commandes MATLAB qui permettent de tracer les deux solutions sont les suivantes :

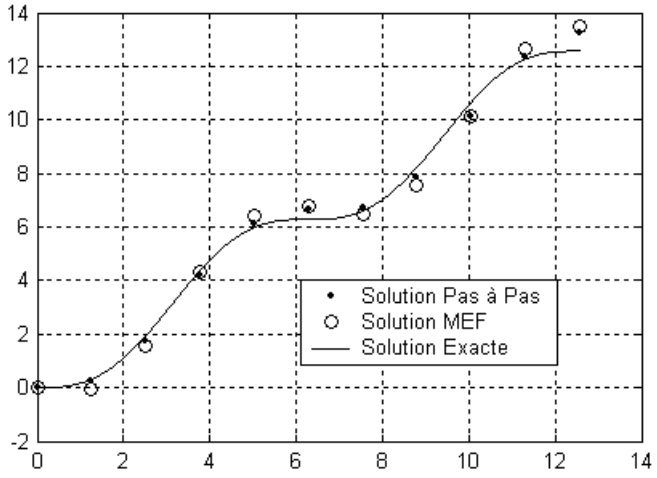
```
clear
clc
n = 20                                  % nombre de pas d'intégration
ou d'éléments finis
dt = 4*pi/n;                            % pas de temps d'intégration
t = 0:dt:4*pi;                          % discrétisation du temps
acc = -t;                               % force d'inertie Fi = - m *
acc
up = sdofth(1,0,1,acc,dt);              % appel de la fonction
d'intégration pas à pas

[um,xm] = EquadiffMEF(n);               % solution par éléments finis
x = 0:0.01:4*pi;                        % coordonnées x pour tracer la
solution exacte
ue= x-sin(x);                           % solution exacte
plot(t,up,'.',xm,um,'o', x,ue)          % trace les trois solutions
```

Remarque : La fonction `EquadiffMEF(n)` a la même structure que `EquaDiff2(n)`. La différence réside seulement dans le calcul des matrices et vecteur élémentaires. Remarquer aussi les variables de sortie !

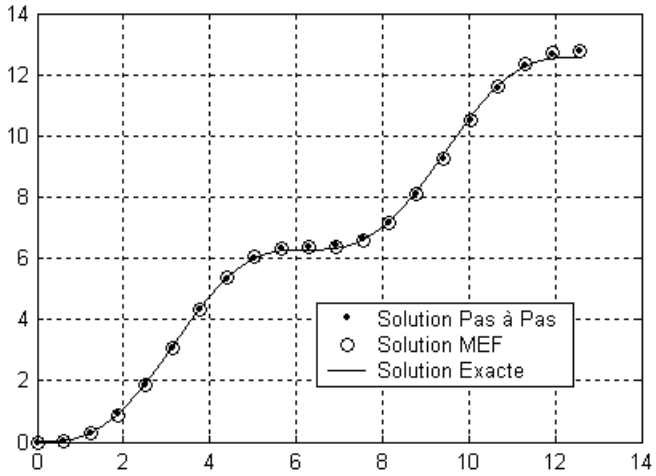
Le tracé des trois solutions avec un nombre de pas d'intégration égal au nombre d'éléments montre sur les deux figures ci-dessous que l'intégration pas à pas est un peu plus proche de la solution exacte quand le nombre d'éléments est petit et les deux

solutions coïncident presque parfaitement avec la solution exacte lorsque le nombre d'éléments est grand.

La différence dans le cas d'un petit nombre d'éléments est due au fait que l'élément utilisé est linéaire, donc le polynôme d'interpolation est du première ordre. Par contre, l'intégration pas à pas utilise une approximation avec un polynôme cubique puisque la



10 éléments et 10 pas d'intégration



20 éléments et 20 pas d'intégration

3.11 Devoir N°3

1) Modifier le programme `EquaDiff2(n)` pour obtenir `EquadiffMEF(n)`. Calculer avec MATLAB les matrices et vecteur élémentaires et changer la fonction `MatElt2Nd(x1,x2)` pour calculer les nouvelles matrices et le nouveau vecteur.
2) Introduire dans MATLAB la fonction `SdofTH` et retracer les deux figures précédentes
3) Utiliser la fonction `FF1D(n)` ci-dessous pour calculer les fonctions de forme d'un élément quadratique (élément à trois nœuds) et calculer les expressions des

matrices et vecteur élémentaires de l'équation précédente : $\frac{d^2u}{dx^2}+u-x=0$ avec les mêmes CAL.

```
function [N,dN] = FF1D(n)
%-------------------------------------------------------------------
-
% [N,dN] = FFDim1(n)
% N  : fonction de forme d'élément unidimensionnel de type Jacobi
% dN : dérivés des fonctions de forme
% n  : nombre de noeuds de l'élément
% la fonction retourne l'expression symbolique des fonctions de
forme
% A. Seghir, 12/09/04
%-------------------------------------------------------------------
-
 x = sym('x','real');                % caractère x
 for i=1:n                           % boucle pour ajouter les
caractères 1 2 3 ...
     c = char(48+i);                 % attention à n > 9
     xn(i) = sym(['x',c],'real');
 end;

 for i=1:n                           % boucle pour les N(i)
     N(i) = sym(1);                  % initialisation à 1
     for j=1:n                       % boucle pour le produit
         if j~=i
            N(i) = N(i)*(x-xn(j))/(xn(i)-xn(j));
         end
     end
 end
 N  = expand(N); N = simple(N);      % réarrangement des termes de
N
 dN =  simple(diff(N,x));            % calcul des dérivée
return
```

3.12 Programme général pour les équations du 2nd ordre

La structure des programmes MATLAB réalisés jusqu'ici ne permet pas de traiter toutes les équations différentielles du second ordre. Des modifications plus ou moins légères sont nécessaires pour chaque cas, elles touchent essentiellement les matrices élémentaires et les conditions aux limites.

Il est possible de penser à mettre la fonction de calcul des matrices élémentaires dans un autre fichier et de créer une autre fonction de traitement des conditions aux limites pour garder le reste du programme dans une fonction qui ne sera pas modifiée. Mais, en plus du fait que le type d'élément reste fixé, les expressions des matrices élémentaires elles mêmes posent toujours problème puisqu'il faut les calculer par intégration sur l'élément. Or, dans certains cas justement, l'intégration peut être difficile ou quasiment impossible, l'intégration numérique est une solution qui convient parfaitement notamment si on l'associe à un élément à bornes fixes appelé élément parent ou élément de référence .

Par ailleurs, les coefficients de l'équation peuvent être eux aussi des fonctions de la variable x. Pour mettre en œuvre un programme qui traite sans modifications des équations très diversifiées, on doit donc en premier lieu caractériser les variables qui sont des données du problème.

Un problème aux valeurs limites défini dans un domaine $\Omega = [x_0, x_\ell]$ et régie par une équation différentielle linéaire du 2nd ordre s'écrit comme suit :

$$\frac{d^2u}{d^2x} + c(x)\frac{du}{dx} + k(x)u = f(x), \text{ avec } u(x_0) = u_0 \text{ et } \frac{du}{dx}(x_\ell) = du_\ell$$

Si le terme du second ordre de l'équation est multiplié par une fonction $m(x)$ alors il est préférable de diviser toute l'équation par ce terme sinon la formulation variationnelle faible fera intervenir un terme intégrale de plus contenant la dérivée de $m(x)$ par rapport à x.

Les paramètres de l'équations sont les fonctions k, c et f ainsi que les valeurs u_0 et du_ℓ, ceux du domaine Ω (de la géométrie) sont x_0 et x_ℓ et ceux propres à la méthode sont le type de l'élément caractérisé par le nombre de nœud qu'il contient `nne` et le nombre

d'éléments total `net`. Il es préférable de regrouper tous ces paramètres dans une structure de données appelée `Eq` comme Equation.

Le programme d'éléments finis qu'on veut obtenir peut être appelé avec une préparation de données de la manière suivante :

```
function TstMefEqDiff;
%-------------------------------
% Test du programme MefEquaDiff
%
% A. Seghir, 14/10/04
%-------------------------------

 Eq.c = @coef_c ;                    % les coefficients de
l'équation pointent
 Eq.k = @coef_k ;                    % sur des noms de fonctions
implémentées
 Eq.f = @func_f ;                    % plus bas

 Eq.nne = 3; Eq.net = 20;            % nbr noeuds par elt et nbr.
elt. total

 Eq.xi  = 0; Eq.xf  = pi;            % bornes d'intégration et CAL
 Eq.u0  = 1; Eq.Du  = 1;

 [U,x] = MefEquaDiff(Eq);            % appel au programme

 t = Eq.xi:0.01:Eq.xf;               % solution exacte
 Ue = 1/3*cos(t)+2/3*cos(2*t)+1/2*sin(2*t);

 plot(x,U,'.',t,Ue);                 % tracé des deux solutions

return

function c = coef_c(x); c = 0; return;% déclaration des coefficients
function k = coef_k(x); k = 4; return;% de l'équation
function f = func_f(x); f = cos(x); return;
```

On voit clairement qu'avec ce type de script, il est facile de traiter des équations très diverses ; il suffit de donner l'expression des coefficients dans les fonctions `coef_k`, `coef_c` et `func_f`, les bornes du domaine ainsi que les conditions aux limites peuvent être

changées à l'aide des valeurs de `Eq.xi`, `Eq.xf`, `Eq.u0` et `Eq.Du`. De plus nous avons maintenant le choix du type d'éléments et de leur nombre avec les variables `Eq.nne` et `Eq.net`.

Remarque

La solution exacte de l'équation différentielle traitée comme exemple peut être obtenue avec ces commandes :

```
>>ue = dsolve('D2u + 4* u = cos(t)','u(0)=1','Du(pi)=1');
>>ue = simple(ue)
ue =
1/3*cos(t)+2/3*cos(2*t)+1/2*sin(2*t)
```

Pour arriver à une telle flexibilité, le programme doit être assez complet. Sa structure générale est :

```
Début du programme
  Géométrie (coordonnées et connectivités)
  Initialisation des matrices
  Début de boucle sur les éléments
      Appel de fonction pour calculer ∫ dN^T dN dx
      Appel de fonction pour calculer ∫ N^T c(x) dN dx
      Appel de fonction pour calculer ∫ N^T k(x) N dx
      Appel de fonction pour calculer ∫ N^T f(x) dx
      Assemblage de la matrice K
      Assemblage du vecteur F
  Fin de boucle sur les éléments
  Application des conditions aux limites
  Résolution
Fin du programme
```

Le fichier fonction `MefEquaDiff` est le suivant :

```
function [U,p] =  MefEquaDiff(Eq)
%-------------------------------------------------------------
% [U,p] =  MefEquaDiff(Eq)
%  U :  vecteur contenant les valeurs de la solution aux noeuds
%  p :  coordonnées des noeuds
%  Eq:  structure contenant les paramètres de l'équation
```

```
%
% A. Seghir, 16/10/04
% ------------------------------------------------------------

[t,p] = maillage1d(Eq.net,Eq.nne,Eq.xi,Eq.xf);  % connectivités et
coordonnées
 nnt = length(p);                        % nombre de noeuds total
 K = zeros(nnt);                         % initialisation de la matrice
K
 F = zeros(nnt,1);                       % initialisation du vecteur F
 for ie = 1:Eq.net                       % boucle sur les éléments
     te = t(ie,:);                       % connectivité de l'élément
actuel
     X = p(te)';                         % coordonnées de l'élément
actuel
     Me = -intdNTdN(X);                  % calcul de l'intégrale ∫ dN^T
dN dx
     Ce =  intNTcxdN(Eq.c,X);            % calcul de l'intégrale ∫ N^T c
dN dx
     Ke =  intNTkxN (Eq.k,X);            % calcul de l'intégrale ∫ N^T
k(x) N dx
     Fe =  intNTfx   (Eq.f,X);           % calcul de l'intégrale ∫ N^T
f(x) dx
     K(te,te) = K(te,te) + Me + Ce + Ke; % assemblage de la matrice
K
     F(te  )  = F(te  ) + Fe ;           % assemblage du vecteur F
 end;                                    % fin de boucle
  F = F -K(:,1)*Eq.u0;                   % application des CAL
  F(nnt) = F(nnt) - Eq.Du;
  K(1,:) = [];
  K(:,1) = [];
  F(1)   = [];
  U = K\F;                               % résolution
  U = [Eq.u0;U];
return
```

La première chose qu'on remarque dans cette version du programme est que le domaine est maillé avec la fonction `maillage1d` qui permet de calculer à la fois les coordonnées des nœuds et les connectivités des éléments. Cette façon de faire est particulièrement utile pour les cas de deux ou trois dimensions, notamment lorsqu'on veut utiliser un fichier de

maillage résultat d'un programme de maillage automatique. Pour ce cas simple, on propose cette fonction comme suit :

```
function [t,p] = maillage1d(net,nne,xi,xf)
%-------------------------------------------------------
% t,p] = maillage1d(net,nne)
% t   : table des connectivités des éléments linéiques
% p   : table des coordonnées des noeuds
% net : nombre d'éléments total
% nne : nombre de noeuds par élément
% xi,xf : bornes du domaine d'intégration
%
% A. Seghir, 14/10/04
%-------------------------------------------------------
 t(1,:) = 1:nne;
 for i = 2:net
     t(i,:) = t(i-1,:)+nne-1;
 end;
 n  = (nne-1)*(net);
 dx = (xf-xi)/n;
 p  = xi:dx:xf;
return
```

La deuxième remarque est plus importante, elle concerne le calcul des matrices élémentaires. La fonction `MatElt2N` utilisée dans les versions précédentes est maintenant remplacée par quatre fonctions dont trois : `intdNTdN`, `intNTcxdN`, `intNTkxN` retournent les matrices élémentaires après intégration et la quatrième `intNTfx` fait le calcul du vecteur force élémentaire. Les arguments d'entrée sont un vecteur `X` des coordonnées de l'élément, les pointeurs `Eq.k`, `Eq.c` et `Eq.f` qui prennent les adresses (ou handle) des fonctions $k(x)$, $c(x)$ et $f(x)$. La taille du vecteur `X` n'étant pas spécifiée, les fonctions prennent en charge tout type d'élément linéique.

Il est à rappeler que c'est dans ces fonctions que s'effectue l'intégration des matrices, et c'est justement ce qui a limité les versions précédentes, donc c'est ici qu'on s'attend à des modifications. En effet, l'intégration exacte des composantes des matrices est remplacée par une somme pondérée des valeurs de ces composantes en des points spécifiques selon la méthode d'intégration numérique de Gauss. Cette méthode qui sera exposée plus loin, nous permet d'évaluer l'intégrale d'une fonction à l'aide d'une somme, soit :

$$\int_a^b f(x)\,dx = \sum_{i=1}^{n} w_i f(x_i)$$

Avec f une fonction quelconque de x, n est le nombre de points x_i dans lesquels on évalue la fonction et w_i sont des valeurs de pondération. Les coordonnées x_i ainsi que les poids w_i sont calculés et la précision de cette intégration numérique dépend du nombre de points n, le résultat est exacte si la fonction f est un polynôme d'ordre $2n$. On donne ci-dessous les points et poids pour l'intégration sur l'intervalle [−1,+1] :

```
n = 3
  x   -0.34641016151378    0                   0.34641016151378
  w    0.55555555555556    0.88888888888889    0.55555555555556
n = 4
  x   -0.861136311594053   0.861136311594053  -0.339981043584856
  0.339981043584856
  w    0.347854845137454   0.347854845137454   0.652145154862546
  0.652145154862546
n = 6
  x   -0.932469514203152   0.932469514203152  -0.661209386466265
  0.661209386466265
  w    0.171324492379170   0.171324492379170   0.360761573048139
  0.360761573048139
  x   -0.238619186083197   0.238619186083197
  w    0.467913934572691   0.467913934572691
```

La fonction MATLAB implémentée pour le programme est basée sur un nombre de point n = 6, ce qui nous permet d'utiliser pratiquement sans perte de précision des éléments ayant jusqu'à 11 nœuds.

```
function [xsi,w] = wpgL6p
%-----------------------------------------------------------------
---
%  [xsi,w] = wpgL6p
%  xsi, w : points et poids de Gauss entre -1 et +1 en vecteurs
lignes
%           nombre de poins : 6 points
%  A.Seghir, 03/08/04
```

```
%-------------------------------------------------------------------
---
xsi = [ -0.932469514203152 , 0.932469514203152 , ...
        -0.661209386466265 , 0.661209386466265 , ...
        -0.238619186083197 , 0.238619186083197 , ...
      ];

w   = [ 0.171324492379170 ,  0.171324492379170 , ...
        0.360761573048139 ,  0.360761573048139 , ...
        0.467913934572691 ,  0.467913934572691 , ...
    ];
return
```

Avec cette méthode d'intégration, les expressions exactes des matrices élémentaires ne sont plus nécessaires, d'autant plus qu'on ne connaît pas, à priori, les fonctions à intégrer ; les coefficients de l'équation peuvent être quelconques.

Le dernier point qui reste consiste à faire le passage entre les bornes d'intégration de l'élément réel et les bornes -1 et $+1$ de l'élément de référence. La transformation géométrique :

$$x = N\ \xi$$

avec ξ la coordonnée dans l'élément de référence, offre la meilleure solution du fait qu'elle utilise les fonctions de forme N définies aussi dans l'élément de référence. On peut vérifier aisément que si $\xi = \pm 1$, $x = x_1$, x_2 pour un élément linéaire et $x = x_1$, x_3 pour un élément quadratique, … etc.

Avec cette transformation géométrique, les dérivées des fonctions de forme dans les deux éléments sont liées par la relation :

$$\frac{dN}{d\xi} = \frac{dN}{dx}\frac{dx}{d\xi}\ ;$$

Soit sous forme plus générale $dN/d\xi = J\ dN/dx$ avec J devient une matrice (Jacobien) dans le cas de deux ou trois dimensions : $J = \frac{dx}{d\xi} = \frac{d}{d\xi}(N\ x_n) = dN\ x_n$.

Exemple d'un élément linéaire : $N = \frac{1}{2} < 1-\xi\ ,\quad 1+\xi >$; $dN = \frac{1}{2} < -1\ ,\quad 1 >$; et $J = \frac{1}{2}(x_2 - x_1)$

Les bornes des intégrales sont ainsi transformées = ±1 :

$$\int_{x_1}^{x_2} f(x)dx = \int_{-1}^{+1} f(N(\xi))\, J\, d\xi$$

Finalement, avec ces deux modifications majeures qui concernent la transformation géométrique et l'intégration numérique, les quatre fonctions des matrices et vecteur élémentaires auront la structure suivante :

```
Début de fonction
  Points et poids d'intégration de Gauss
  Initialisation de la matrice (vecteur) élémentaire
  Début de boucle sur les points de Gauss
      Evaluation des fonctions de forme au point de Gauss
      Calcul de J = dN * Xn
      Calcul de dN/dx = J-1 dN
      Somme des produits dNT.dN, dNT.N, NT.N avec multiplication par
les poids et J
  Fin de boucle sur les points de Gauss
Fin de la fonction
```

Le code source des quatre fonctions est le suivant :

```
function Me = intdNTdN(Xn)
%-----------------------------------
% integration de : dN' * dN
%-----------------------------------
 [xsi, w] = wpgL4p;
 npg = length(xsi);
 nne = length(Xn);
 Me  = zeros(nne);
 for i =1:npg
     [N, dN] = FF1D(xsi(i), nne);
     J = dN * Xn;
     dN = dN * inv(J);
     Me = Me + det(J) * w(i) * ( dN' * dN);
 end
return
```

```
function Ce = intNTcxdN(cx,Xn)
%---------------------------------
% integration de : N' * c(x) * dN
%---------------------------------
 [xsi, w] = wpgL4p;
 npg = length(xsi);
 nne = length(Xn);
 Ce  = zeros(nne);
 for i =1:npg
     [N, dN] = FF1D(xsi(i), nne);
     x = N * Xn;
     J = dN * Xn;
     dN = dN * inv(J);
     c = feval(cx,x);
     Ce = Ce + det(J) * w(i) * c * ( N' * dN);
 end
return

function Ke = intNTkxN(kx,Xn)
%---------------------------------
% integration de : N' * k(x) * N
%---------------------------------
 [xsi, w] = wpgL4p;
 npg = length(xsi);
 nne = length(Xn);
 Ke  = zeros(nne);
 for i =1:npg
     [N, dN] = FF1D(xsi(i), nne);
     x = N * Xn;
     J = dN * Xn;
     k = feval(kx,x);
     Ke = Ke + det(J) * w(i) * k * ( N' * N);
 end
return

function Fe = intNTfx(fx,Xn)
%-----------------------------
% integration de : N' * f(x)
%-----------------------------
 [xsi, w] = wpgL4p;
 npg = length(xsi);
```

```
 nne = length(Xn);
 Fe  = zeros(nne,1);
 for i =1:npg
     [N, dN] = FF1D(xsi(i), nne);
     x = N * Xn;
     J = dN * Xn;
     f = feval(fx,x);
     Fe = Fe + det(J) * w(i) * f * N';
 End
return
```

Les fonctions de forme ainsi que leurs dérivées pour des éléments de référence d'ordre *n* sont :

```
function [N,dN] = FF1D(x,n)
%-------------------------------------------------
% [N,dN] = FF1D(x,n)
% N  : fonctions de forme d'éléments linéiques 1D
% dN : dérivées des fonctions de forme
% x  : point d'evaluation des fonctions
% n  : nombre de noeuds de l'élément
%
%A. Seghir 14/10/04
%-------------------------------------------------
 if isa(x,'sym')                       % détermine si x est
symbolique
     one  = sym(1);
     zero = sym(0);
 else
     one  = 1.0;
     zero = 0.0;
     if (x < -1) | (x > 1); error('coordonné x erronée'); end;
 end;

 xn =-1:2/(n-1):1;

 for i=1:n
     p(i) = one;
     q(i) = one;
     s(i) = zero;
```

```
        for j=1:n
            if j~=i
               p(i) = p(i) * (x     - xn(j));
               q(i) = q(i) * (xn(i) - xn(j));
               r  = one;
               for k =1:n
                   if (k~=i & k~=j); r = r*( x - xn(k) ); end;
               end;
               s(i) = s(i) + r;
            end;
        end;
     end;
     N  = p./q;
     dN = s./q;
    return
```

3.13 Devoir N°4

1) Reprendre l'équation du devoir N°3, question 3 : $d^2u/dx^2 + u - x = 0$ avec ce programme. Modifier les fonctions `coef_k`, `coef_c` et `func_f`, ainsi que les bornes du domaine et les conditions aux limites. Faire une étude de convergence en type et en nombre d'éléments.
2) Résoudre avec le programme `MefEquaDiff` l'équation :

$$\frac{d^2u}{dx^2} + \frac{1}{x}\frac{du}{dx} = \frac{1+x^2}{x}$$

avec les conditions aux limites : $u(0) = 1$ et $\frac{du}{dx}(4\pi) = -2$

intégration numérique

```
function TstIntNum
clear
clc
xn  =[-2 , 2];
[xsi,w, npg] = wpgL6p;
```

```
sn = 0;
for i=1:npg
    [N,dN]  = FF2Nd(xsi(i));
    x  = xn*N;
    dx = xn*dN;
    f  = func(x);
    sn = sn + dx* w(i) * f;
end;
x  = sym('x','real');
se = int(func(x),xn(1),xn(2))
int(func(x))
sn
se = double(se)
err = sn-se

function f = func(x)
 f = x.*exp(-x);
return

function [N,dN] = FF2Nd(xsi)
  N  = 0.5*[1-xsi; 1+xsi];
  dN = 0.5*[-1; 1];
return;
```

chapitre 4

Elément Barre

4.1 Equation gouvernante

L'élément barre est utilisé dans les assemblages de barres ou de tiges travaillant en traction ou compression. On les trouves surtout en charpente métallique et dans les systèmes à treillis.

Pour formuler cette élément, on considère une barre de section A de longueur L soumise à une traction $P(x)$ variant de P_0 à P_L.

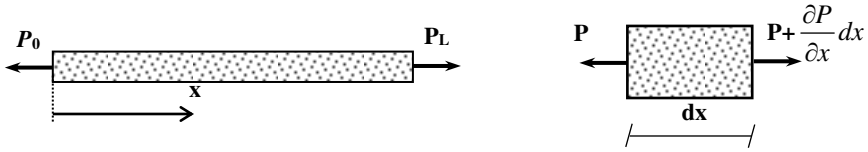


Fig. 2.1 : Equilibre élémentaire

Une portion infinitésimale de longueur dx située à la coordonnée x le long de la barre est en équilibre dynamique sous le système de forces suivant :

$$\sum F = m\gamma$$

$$\left[P + \frac{\partial P}{\partial x} dx\right] - P = (\rho\, A\, dx) \frac{\partial^2 u}{\partial t^2} \tag{2.1}$$

Dans cette expression u désigne le déplacement longitudinal, x est la cordonnée et t le temps.

Si on désigne par E le module d'élasticité du matériau avec lequel est faite la barre, la loi de Hooke donne la contrainte axiale en fonction de la déformation longitudinale :

$$\frac{P}{A} = \sigma_x = E\,\varepsilon_x \tag{2.2}$$

La déformation est liée au déplacement par la dérivée par rapport à x :

$$\varepsilon_x = \frac{\partial u}{\partial x} \tag{2.3}$$

En substituant (2.3) dans (2.2) on obtient l'expression de la traction P en fonction du déplacement u :

$$P = EA\frac{\partial u}{\partial x} \tag{2.4}$$

D'où l'équation différentielle de l'équilibre de l'élément dx qui s'obtient en remplaçant (2.4) dans (2.1) :

$$\frac{\partial}{\partial x}\left(EA\frac{\partial u}{\partial x}\right) = \rho A\frac{\partial^2 u}{\partial t^2} \tag{2.5}$$

Les conditions aux limites de cette équation dépendent de l'encastrement ou des déplacements imposés pour u, et du chargement aux nœuds pour les dérivées de u (Equation 2.4).

4.2 Formulation de l'élément

En prenant δu comme fonction de pondération, la formulation variationnelle forte de l'équation (2.5) s'écrit :

$$\int_0^L \delta u\left[\frac{\partial}{\partial x}\left(EA\frac{\partial u}{\partial x}\right)\right] - \delta u\,\rho A\frac{\partial^2 u}{\partial t^2}\,dx = 0 \tag{2.6}$$

La formulation faible s'écrit en prenant l'intégration par parties du premier terme :

$$\int_0^L \delta\frac{\partial u}{\partial x}\, EA\frac{\partial u}{\partial x}\, dx + \int_0^L \delta u\, \rho A\frac{\partial^2 u}{\partial t^2}\, dx - \left[\delta u\, AE\frac{\partial u}{\partial x}\right]_0^L = 0 \tag{2.7}$$

Le dernier terme n'est que la différence des forces appliquées aux extrémités de la barre : $\delta u|_{x=L} P_L - \delta u|_{x=0} P_0$.

Pour la discrétisation de l'équation (2.7) en prends des fonctions de forme linéaires avec $x_1 = 0$ et $x_2 = L$. Les expressions de ces fonctions et leurs dérivées sont :

$$N(x) = < \frac{x-L}{0-L} \quad \frac{x-0}{L-0} > = \frac{1}{L} < L-x \quad x > \; ; \; dN(x) = \frac{1}{L} < -1 \quad 1 > \tag{2.8}$$

Ainsi en remplaçant pour les différent opérateurs on obtient :

$$\delta u = < \delta u_0 \quad \delta u_L > \begin{Bmatrix} (L-x)/L \\ x/L \end{Bmatrix} \qquad \delta\frac{\partial u}{\partial x} u = < \delta u_0 \quad \delta u_L > \begin{Bmatrix} -1/L \\ 1/L \end{Bmatrix} \tag{2.9a}$$

$$\delta u\big|_{x=0} = < \delta u_0 \quad \delta u_L > \begin{Bmatrix} 1 \\ 0 \end{Bmatrix} \qquad \delta u\big|_{x=L} = < \delta u_0 \quad \delta u_L > \begin{Bmatrix} 0 \\ 1 \end{Bmatrix} \tag{2.9b}$$

$$\frac{\partial u}{\partial x} = < -1/L \quad 1/L > \begin{Bmatrix} u_0 \\ u_L \end{Bmatrix} \qquad \frac{\partial^2 u}{\partial t^2} = \frac{1}{L} < L-x \quad x > \begin{Bmatrix} \ddot{u}_0 \\ \ddot{u}_L \end{Bmatrix} \tag{2.9c}$$

où : $\ddot{u}$ représente la seconde dérivée par rapport au temps des déplacements ; c'est-à-dire l'accélération.

Avec ces expressions l'équation intégrale faible (2.7) devient après simplification de $< \delta u_0 \quad \delta u_L >$:

$$\begin{aligned} &\int_0^L \begin{Bmatrix} -1/L \\ 1/L \end{Bmatrix} EA < -1/L \quad 1/L > dx \begin{Bmatrix} u_0 \\ u_L \end{Bmatrix} + \\ &\int_0^L \frac{1}{L}\begin{Bmatrix} L-x \\ x \end{Bmatrix} \rho A \frac{1}{L}\begin{Bmatrix} L-x \\ x \end{Bmatrix} dx \begin{Bmatrix} \ddot{u}_0 \\ \ddot{u}_L \end{Bmatrix} = \begin{Bmatrix} -P_0 \\ P_L \end{Bmatrix} \end{aligned} \tag{2.10}$$

soit sous forme matricielle : $K_e\, U_e + M_e \ddot{U}_e = F_e$, avec :

$$K_e = \int_0^L \begin{Bmatrix} -1/L \\ 1/L \end{Bmatrix} EA < -1/L \quad 1/L > dx$$
$$= \frac{1}{L^2}\int_0^L EA \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} dx = \frac{EA}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{2.12}$$

$$M_e = \int_0^L \frac{1}{L}\begin{Bmatrix} L-x \\ x \end{Bmatrix} \rho A \frac{1}{L} < L-x \quad x > dx$$
$$= \frac{\rho A}{L^2}\int_0^L \begin{bmatrix} (L-x)^2 & x(L-x) \\ x(L-x) & x^2 \end{bmatrix} dx = \frac{\rho AL}{6}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \tag{2.13}$$

$$F_e = \begin{Bmatrix} 0 \\ 1 \end{Bmatrix} P_L - \begin{Bmatrix} 1 \\ 0 \end{Bmatrix} - P_0 = \begin{Bmatrix} -P_0 \\ P_L \end{Bmatrix} \tag{2.14}$$

Les matrices élémentaires K_e et M_e sont appelées respectivement matrice de rigidité et matrice masse. F_e est le vecteur chargement de l'élément. Ces expressions sont valables si la section A, la densité ρ et le module de Young E ne varient pas le long de toute la barre. Dans le cas où ces quantités varient, il est possible de subdiviser la barre en plusieurs éléments et de prendre des valeurs moyenne pour chaque l'élément.

L'expression de la matrice masse telle qu'elle est obtenue en (2.13) est appelée masse cohérente ou répartie. Il est possible de concentrer la masse de l'élément en ses extrémité. On attribue à chacun des deux nœuds la moitié de la masse totale de l'élément soit :

$$M_e^c = \frac{\rho AL}{2}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.15}$$

4.3 Exemple d'une tige encastrée

Une tige de longueur $L = 3m$ et de section $A = 25cm^2$ est encastrée à une de ses extrémité et soumise à une tension $P = 250KN$ à l'autre extrémité. La densité et le module de Young du matériau de la tige sont : ρ =7800 Kg/m^3et $E = 210\,000\ MPa$.

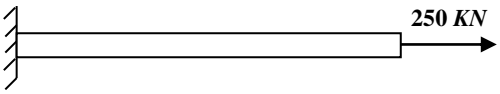


***Fig.* 2.2 : Barre encastrée**

Les matrices masse et rigidité ainsi que le vecteur chargement sont :

$$K = 175\cdot 10^6 \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} N/m \; ; \; M = 29.25 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} Kg \; ; \; F_e = \begin{Bmatrix} 0 \\ 25\cdot 10^4 \end{Bmatrix} N$$

Compte tenue de la condition $u(0) = 0$. La résolution donne un déplacement $u(L) = 1.4mm$

et une période $T = 0.0026\ sec$.

4.4 Structures planes à treillis

Les structures à treillis sont constituées par des assemblage de barres liées par des joints de telle sorte que le chargement extérieur soit repris uniquement par des forces axiales dans les barres. La figures 2.3 montre un exemple de système à treillis composé d'un assemblage de 13 barres et soumis à un chargement de deux forces.

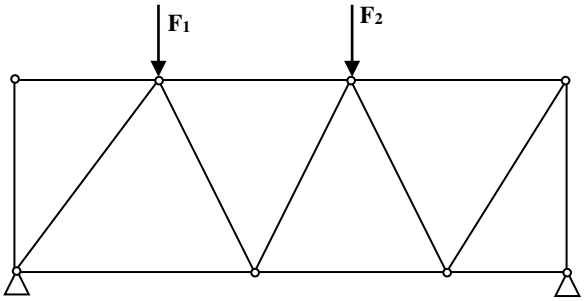


***Fig.* 2.3 : Système à treillis**

Les barres composant ce système subissent deux déplacements à leurs extrémités ; une composante horizontale et une autre verticale. Cependant, seule le déplacement axiale à la barre donne naissance à la force axiale. Ainsi, les matrices élémentaires d'une barres

bidimensionnelle (fig.2.4) deviennent des matrices 4×4 puisque le vecteur des déplacements élémentaires devient :

$$U_e = < u_1 \quad v_1 \quad u_2 \quad v_2 >^{\mathrm{T}} \tag{2.16}$$

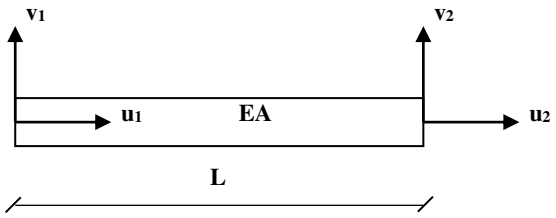


***Fig.* 2.4 : Elément barre bidimensionnel**

Toutes les composantes de la matrice de rigidité associées à la composante *v* du déplacement sont nulles. Celles de la matrice masse, par contre, ne le sont pas toutes. Les effets de l'inertie existes aussi lorsque les extrémités de la barre se déplacent dans la direction orthogonale, seulement, il n'y aucun couplage entre les deux composantes *u* et *v* du déplacement. Les deux matrices s'écrivent ainsi :

$$K_e = \frac{AE}{L}\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}; \qquad M_e = \frac{\rho AL}{6}\begin{bmatrix} 2 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \\ 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 2 \end{bmatrix} \tag{2.17}$$

La matrice masse concentrée s'écrit de la même manière qu'en (2.15) en attribuant la moitié de la masse de l'élément pour toutes les composantes du déplacement :

$$M_e = \frac{\rho AL}{2}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.18}$$

Les expressions des matrices élémentaires telles qu'elles sont écrites par les équations (2.17) et (2.18) sont valables dans un système d'axes confondu avec l'axe longitudinal de l'élément barre. Dans les cas pratiques, la barre peut être incliné par rapport aux axes de référence. Il convient alors de faire une rotation d'axes pour revenir aux axes de la barres.

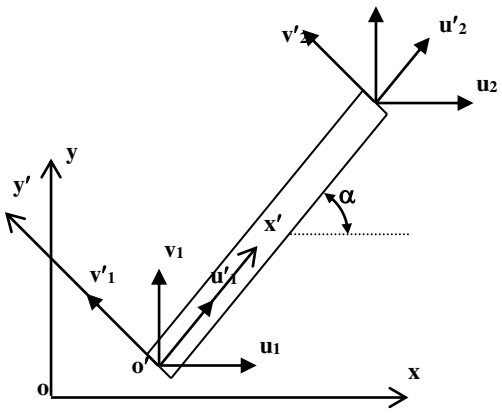


Fig. 2.5 : Elément barre incliné

La figure 2.5 montre une barre inclinée d'un angle α par rapport à l'axe horizontal du repère (oxy). On note (u,v) les composantes du déplacement dans ce système et (u',v') celles du déplacement dans le repère (o$'x'y'$) lié à la barre. On note aussi R la matrice de rotation de repère qui permet de faire le passage du système (oxy) au nouveau système (o$'x'y'$). On peut ainsi écrire les relations suivantes :

$$\begin{Bmatrix} u'_1 \\ v'_1 \end{Bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \end{Bmatrix} \; ; \; \begin{Bmatrix} u'_2 \\ v'_2 \end{Bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{Bmatrix} u_2 \\ v_2 \end{Bmatrix}$$

avec $c = \cos(\alpha)$ et $s = \sin(\alpha)$.

Soit avec la totalité des vecteurs des déplacements élémentaires U_e et U'_e :

$$\begin{Bmatrix} u_1' \\ v_1' \\ u_2' \\ v_2' \end{Bmatrix} = \begin{bmatrix} c & s & 0 & 0 \\ -s & c & 0 & 0 \\ 0 & 0 & c & s \\ 0 & 0 & -s & c \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \end{Bmatrix} \; ; \quad U_e' = R\, U_e \tag{2.19}$$

Il reste maintenant de lier le chargement aux déplacements dans le repère(*oxy*), on a la relation dans le repère lié à la barre $F_e' = K_e'\, U_e'$. Le relation entre les deux vecteurs forces dans les deux repères est :

$F_e' = R\, F_e$, ou bien : $F_e = R^T\, F_e'$; avec : $Fe = <\ F_{x1} \quad F_{y1} \quad F_{x2} \quad F_{y2}>^{\mathrm{T}}$ (2.20)

d'où il en résulte :

$$F_e = R^T\, K_e'\, U_e' = R^T\, K_e'\, R\, U_e = K_e\, U_e \tag{2.21}$$

On obtient ainsi la relation entre les expressions de la matrice de rigidité dans les deux repères :

$$K_e = R^T\, K_e'\, R \tag{2.22}$$

Soit :

$$K_e = \begin{bmatrix} c & -s & 0 & 0 \\ s & c & 0 & 0 \\ 0 & 0 & c & -s \\ 0 & 0 & s & c \end{bmatrix} \frac{AE}{L} \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} c & s & 0 & 0 \\ -s & c & 0 & 0 \\ 0 & 0 & c & s \\ 0 & 0 & -s & c \end{bmatrix}$$

$$= \frac{AE}{L} \begin{bmatrix} c^2 & cs & -c^2 & -cs \\ cs & s^2 & -cs & -s^2 \\ -c^2 & -cs & c^2 & cs \\ -cs & -s^2 & cs & s^2 \end{bmatrix}$$

la matrice de rigidité, la matrice masse cohérente ou concentrée reste inchangée à la rotation de repère. En effet du point de vue de la physique, la masse est indépendante de l'orientation du repère.

Le script MATLAB qui permet de faire les vérifications par calcul des matrice élémentaires est :

```
%-----------------------------------------
% Matrices élémentaires de l'élément barre
%
% A. Seghir, 10/08/04
%-----------------------------------------
clear                                  % efface toutes les variables
dans la mémoire globale
clc                                    % efface la fenêtre des
commandes
syms s c real                          % déclare les variables
symboliques

R = [ c  s  0 0                        % matrice de rotation
     -s  c  0 0
      0  0  c s
      0  0 -s c
    ]

K = [ 1 0 -1 0                         % matrice de rigidité dans le
repère de la barre
      0 0  0 0                         % le facteur A*E/L est
intentionnellement omis
     -1 0  1 0
      0 0  0 0
     ]

M  = [ 2  0  1  0                      % matrice masse dans le repère
lié à la barre
       0  2  0  1                      % ale facteur rho*A*L/6 est
aussi omis
       1  0  2  0
       0  1  0  2
   ]

Ke = R' * K * R                        % matrice de rigidité dans le
repère global (oxy)
Me = R' * M * R                        % matrice masse dans le repère
global (oxy)
```

4.5 Exemple de deux barres

La figure 2.6 montre une structure composée de deux barres encastrées à leurs extrémités supérieures et assemblées à leurs extrémités inférieures avec un joint rotule auquel est suspendu un poids $P = 3400\ KN$.

Les barres ont les mêmes caractéristiques :

Longueur : $L = 1\ m$; section : $A = 25\ cm^2$

et module de Young $E = 210\,000\ MPa$.

L'angle à l'encastrement est : $\alpha = 36.87°$.

Ce qui donne pour la barre N° 1 : $c = -0.8$ et $s = 0.6$

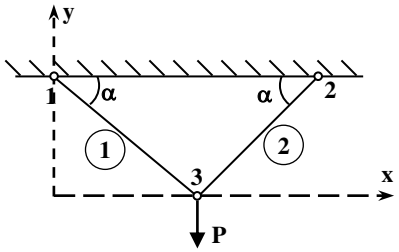


***Fig.* 2.5 : Treillis à deux barres**

Et le premier système élémentaire : $Ke_1\ Ue_1 = Fe_1$

$$525\cdot 10^6 \begin{bmatrix} 0.64 & -0.48 & -0.64 & 0.48 \\ -0.48 & 0.36 & 0.48 & -0.36 \\ -0.64 & 0.48 & 0.64 & -0.48 \\ 0.48 & -0.36 & -0.48 & 0.36 \end{bmatrix} \begin{Bmatrix} u_1^{e1} \\ v_1^{e1} \\ u_2^{e1} \\ v_2^{e1} \end{Bmatrix} = \begin{Bmatrix} F_{x1}^{e1} \\ F_{y1}^{e1} \\ F_{x2}^{e1} \\ F_{y2}^{e1} \end{Bmatrix}$$

En faisant intervenir le vecteur déplacent de tout les nœuds (de 1 à 3) le système devient :

$$525\cdot 10^6 \begin{bmatrix} 0.64 & -0.48 & 0 & 0 & -0.64 & 0.48 \\ -0.48 & 0.36 & 0 & 0 & 0.48 & -0.36 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -0.64 & 0.48 & 0 & 0 & 0.64 & -0.48 \\ 0.48 & -0.36 & 0 & 0 & -0.48 & 0.36 \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{Bmatrix} = \begin{Bmatrix} F_{x1}^{e1} \\ F_{y1}^{e1} \\ 0 \\ 0 \\ F_{x2}^{e1} \\ F_{y2}^{e1} \end{Bmatrix}$$

Pour la barre N°2 : $c = 0.8$ et $s = 0.6$, le système avec toutes les composantes du vecteur déplacements est :

$$525\cdot10^6\begin{bmatrix}0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&0.64&0.48&-0.64&-0.48\\0&0&0.48&0.36&-0.48&-0.36\\0&0&-0.64&-0.48&0.64&0.48\\0&0&-0.48&-0.36&0.48&0.36\end{bmatrix}\begin{Bmatrix}u_1\\v_1\\u_2\\v_2\\u_3\\v_3\end{Bmatrix}=\begin{Bmatrix}0\\0\\F_{x1}^{e2}\\F_{y1}^{e2}\\F_{x2}^{e2}\\F_{y2}^{e2}\end{Bmatrix}$$

L'assemblage des deux systèmes élémentaires donne le système global :

$$525\cdot10^6\begin{bmatrix}0.64&-0.48&0&0&-0.64&0.48\\-0.48&0.36&0&0&0.48&-0.36\\0&0&0.64&0.48&-0.64&-0.48\\0&0&0.48&0.36&-0.48&-0.36\\-0.64&0.48&-0.64&-0.48&1.28&-0\\0.48&-0.36&-0.48&-0.36&0&0.72\end{bmatrix}\begin{Bmatrix}u_1\\v_1\\u_2\\v_2\\u_3\\v_3\end{Bmatrix}=\begin{Bmatrix}0\\0\\0\\0\\0\\-3.4\cdot10^6\end{Bmatrix}$$

L'application des conditions aux limites (encastrement : $u_1 = v_1 = u_2 = v_2 = 0$) réduit ce système à deux inconnus :

$$525\cdot10^6\begin{bmatrix}1.28&0\\0&0.72\end{bmatrix}\begin{Bmatrix}u_3\\v_3\end{Bmatrix}=\begin{Bmatrix}0\\-3.4\cdot10^6\end{Bmatrix}$$

La solution donne : $u_3 = 0$; $v_3 = -8.99\ mm$ qui est un déplacement vers le bas.

Pour calculer les réactions, il suffit d'effectuer les produits matriciels des systèmes élémentaires avec les déplacements maintenant connus et les forces internes inconnues ; on obtient pour la barre N°1 :

$$525\cdot10^6\begin{bmatrix}0.64&-0.48&-0.64&0.48\\-0.48&0.36&0.48&-0.36\\-0.64&0.48&0.64&-0.48\\0.48&-0.36&-0.48&0.36\end{bmatrix}\begin{Bmatrix}u_1^{e1}=0\\v_1^{e1}=0\\u_2^{e1}=0\\v_2^{e1}=-8.99\cdot10^{-3}\end{Bmatrix}=\begin{Bmatrix}-2265.48\\1699.11\\2265.48\\-1699.11\end{Bmatrix}KN$$

Les réactions correspondent aux deux premières composantes du vecteur *Fe* et elles sont égales à l'opposée des forces appliquées sur l'autre extrémité et qui correspondent à la troisième et quatrième composantes.

La force axiale dans la barre s'obtient avec la même rotation de repère définie plus haut :

$Fx = c\ Fe_1 + s\ Fe_2 =$ 2831.85 KN .

Les calculs donne la même valeur pour la deuxième barre.

4.6 Techniques d'assemblage pour les éléments barres

Il y a lieu de remarquer lors de l'assemblage des deux systèmes élémentaires que les composantes 5 et 6 de la matrice globale K sont la somme des composantes 3 et 4 des matrices élémentaires. On voit donc que les composantes qui correspondes aux nœud 3 occupent les positions $2\times3 = 6$ et $2\times3-1 = 5$ dans la matrice globale.

En généralise ainsi cet assemblage en calculant une table de localisation à partir de la table des connectivités comme suit :

```
L([2*i-1 2*i])  =[2*t(i)-1    2*t(i)].
```

La fichier fonction MATLAB suivant retourne une table de localisation dans le cas de deux degrés de liberté par nœud :

```
function L = Localise(t)
% L = Localise(t)
% t : table de connectivités de l'élément
% L : table de localisation
   nne = length(t);
   for i= 1:nne
       L([2*i-1 2*i]) = [2*t(i)-1 2*t(i)];
   end
return
```

Dans le cas d'un nombre de degrés de liberté quelconque, on propose la fonction suivante qui est plus générale :

```
function L = Loc(t,n)
%-------------------------------------------------
% L = Loc(t,n)
% t : table de connectivités de l'élément
```

```
% L : table de localisation
% n : nombre de degrés de liberté par noeud
%
%  A. Seghir, 24/08/04 modifié 23/10/04
%-------------------------------------------------
 e = eye(n);
 L = kron(n*t,e(n,:));
 for i=1:n-1
     L = L + kron(n*t-i,e(n-i,:));
 end
return
```

4.7 Programme d'éléments finis pour l'élément barre

La mise en œuvre d'un programme d'éléments finis pour les structures d'assemblage de barres nécessite en premier lieu une bonne organisation du fichier de données. Le programme doit avoir une structure qui permet d'effectuer trois grandes tâches : lecture de données, calcul de la solution et affichage des résultas.

4.7.1 Saisie des données

La partie du programme destinée à la lecture des données doit pouvoir construire des variables pour la géométrie des éléments, les caractéristiques du matériau et le chargement extérieur.

- La géométrie concerne les coordonnées des nœuds `p`, les connectivités des éléments `t` et leurs sections droites `A` ainsi que les conditions d'encastrement qu'on regroupe dans un vecteur `e`.
- Les caractéristiques du matériau sont le module d'élasticité `E` et la masse volumique `rho`.
- Le chargement extérieur est mis sous forme d'un vecteur colonne `F`.

Ces données peuvent être saisie dans un fichier texte mais pour des raisons de simplicités offertes par les fonctions MATLAB, ou utilise un fichier fonction.

Dans le cas de l'exemple de deux barres précédent (§2.5), ce fichier est le suivant :

```
function [t,p,e,A,E,rho,F] = expl2barres;
```

```
t = [1  3              % connectivité de l'élément 1
     2  3              % connectivité de l'élément 2
    ];
p = [  0.0  0.6        % coordonnées du noeud 1
       1.6  0.6        % coordonnées du noeud 2
       0.8  0.0        % coordonnées du noeud 3
    ];
e = [ 1; 2             % noeud d'encastrement 1
      3; 4             % noeud d'encastrement 2
    ] ;
A = [1 1] * 25e-4;     % sections des éléments barres
E = [1 1] * 210e9;     % module d'élasticité des éléments
rho=[1 1] * 2800 ;     % masse volumique 2800 kg/m3

F = [ 0;  0            % noeud 1 non chargé (encastrement)
      0;  0            % noeud 2 non chargé (encastrement)
      0; -3.4e6        % noeud 3 chargé dans la direction verticale
    ];
```

Si on veut afficher la structure pour une vérification visuelle, on utilise les commandes suivantes :

```
>> [t,p,b,A,E,rho,F] = expl2barres;
>> plotmesh(t,p,1,1,'b');
```

La fonction `plotmesh` affiche une figure du maillage, elle permet donc, avec le tracer la structure saisie, une vérification visuelle des éléments et des nœuds. Son script est :

```
function plotmesh(T,P, NodeLabels, EltLabels,color)
% plotmesh(T, P, NodeLabels, EltLabels, color)
%
%  T = Table des connectivités (fem.mesh.t)
%  P = Table des coordonnées   (fem.mesh.p)
%  NodeLabels : vaut true pour afficher les éléments
%  EltLabels  : vaut true pour afficher les nœuds
%  color      : couleur des lignes
%
% A. Seghir, 02/08/04 modifié 07/08/04
Net = size(T,1);
```

```
for ie = 1:Net
    XY = P(T(ie,:),:);
    X = [[XY(:,1)]; XY(1,1)];
    Y = [[XY(:,2)]; XY(1,2)];
    line(X,Y,'color',color)
    if(EltLabels)
        x =  mean(XY(:,1));
        y =  mean(XY(:,2));
        text(x,y,num2str(ie),'color','b')
    end
end
if(NodeLabels)
    Np = size(P,1);
    for i=1:Np
        text(P(i,1),P(i,2),num2str(i),'color','m')
    end
end
```

4.7.2 Calcul des matrices de rigidité et de masse

Une fois la géométrie est saisie, on peut calculer les matrices de rigidité et de masse assemblées. On crée une fonction `truss2dKM` pour les deux matrices *K* et *M* à la fois.

Les variables d'entrée communes aux deux matrices sont : la table des connectivités des éléments `t`, la table des coordonnées des nœuds `t` et les sections des éléments `A`. On ajoute les modules d'élasticité `E` pour la matrice de rigidité *K* et les masses volumiques `rho` pour la matrice masse *M*. On prends des caractéristiques différentes pour chaque élément (`A`, `E` et `rho` des vecteurs) pour pouvoir calculer des structures d'assemblage de barres de plusieurs types de matériau et de différentes sections.

Le script de la fonction `truss2dKM` est :

```
function [K,M] = truss2dM(t,p,A,E,rho)
% [K,M] = truss2dKM(t,p,A, rho)
% K  : matrice de rigidité assemblée
% M  : matrice masse assemblée
% t  : table de connectivités des éléments
% p  : table des coordonnées des noeuds
% A  : sections des éléments (des barres)
% E  : modules d'élasticité en vecteur des éléments
```

```
% rho: masse volumique
% A. Seghir, le 07/08/04, modifié : 26/10/04

  net = size(t,1);      % nombre d'éléments total
  nnt = 2*size(p,1);    % nombre de noeuds total

  K = sparse(nnt,nnt);  % initialisation des matrices K et M
  M = sparse(nnt,nnt);  % par des matrices creuses nulles

  for i = 1:net           % début de boucle sur les éléments
      ti = t(i,:);        % tables de connectivité et de
      Li = localise(ti);  % localisation de l'élément courant
      Ke = truss2dKe(p(ti,:),A(i),E(i));   % matrices de rigidité
      Me = truss2dMe(p(ti,:),A(i),rho(i)); % et masse élémentaires
      K(Li,Li) = K(Li,Li) + Ke;        % assemblage de la matrice K
      M(Li,Li) = M(Li,Li) + Me;        % assemblage de la matrice M
  end                                  % fin de boucle
return                                 % fin de fonction
```

Il est à remarquer qu'on a initialisé les matrices *K* et *M* avec des matrices creuses (`sparse`). L'avantage qu'offre les matrices creuses est que seul les éléments non nuls sont sauvegarder en mémoire. MATLAB est équipé d'une structure de données et de plusieurs fonctions pour prendre en charge toutes les opérations matricielles sur ce type de matrices et d'une façon pratiquement imperceptible pour l'utilisateur.

La construction des deux matrices *K* et *M* n'est pas très différente de ce qui a été présenté pour les équations différentielles. On a introduit ici une table de localisation `Li` qu'on calcul avec la fonction `localise` (§2.6) et qu'on utilise à la place de la table des connectivités `t` pour l'assemblage de *K* et *M*.

Pour le calcul des matrices élémentaires *Ke* et *Me* on a besoin pratiquement que saisir les composantes. La longueur et les cosinus directeurs de l'élément sont calculés à partir des coordonnées de ses nœuds :

$$L=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}\ ;\ \cos(\alpha)=(x_2-x_1)/L\ ;\ \sin(\alpha)=(y_2-y_1)/L \qquad (2.24)$$

Les fichiers fonctions des deux matrices élémentaires sont :

```
function ke = truss2DKe(XY,A,E)
```

```
% ke = truss2dKe(XY,A,E)
%      Calcul de la matrice elementaire pour un
%      élément barre à deux noeuds (x1,y1) (x2,y2)
% A : section de la barre
% E : module d'élasticité du matériau
% XY: cordonnées des noeuds  XY = [x1,y1; x2,y2]
% A. Seghir, 06/08/04

  [c,s,L] = EltLen(XY);  % longueur et orientation de l'élément

  cc = c*c;              % cos(angle)^2
  cs = c*s               % cos(angle)*sin(angle)
  ss = s*s;              % sin(angle)^2

  ke = (A*E/L) * [  cc  cs -cc -cs   % matrice élémentaire Ke
                    cs  ss -cs -ss
                   -cc -cs  cc  cs
                   -cs -ss  cs  ss
                 ];
return

function Me = truss2dMe(XY,A,rho)
% Me = truss2dMe(XY,A,rho)
%      Calcul de la matrice masse elementaire Me pour un
%      élément barre à deux noeuds (x1,y1) (x2,y2)
% A   : section de l'élément
% rho : masse volumique de l'élément
% XY  : cordonnées des noeuds  XY = [x1,y1; x2,y2]
% A. Seghir, 07/08/04

  L = EltLen(XY);
  Me = 0.5*rho*A*L*eye(4);
return
```

Pour la masse repartie remplacer la ligne `Me = 0.5*rho*A*L*eye(4);` par :

```
  Me =(rho*A*L/6) * [ 2  0  1  0
                      0  2  0  1
                      1  0  2  0
                      0  1  0  2
```

```
                    ];
```

La matrice masse peut être calculée en considérant une masse répartie ou une masse concentrée. Pour changer il faut mettre la première expression en ligne commentaire et enlever les `%` de la seconde expression. Au besoin, il est préférable d'introduire une variable option `op` dans l'entête de la fonction : `truss2dMe(XY,A,rho,op)` et de choisir ensuite avec un test `if` ce qu'il faut retourner comme résultat. Il faut tout de même noter que dans la plus part des cas le résultat est sensiblement le même et c'est pour sa configuration de matrice diagonale qu'on préfère la masse concentrée.

Enfin, on donne le code source de la fonction `EltLen` :

```
function [ds,c,s] = EltLen(XY)
% [ds,c,s] = Getcosine(XY)
%       Calcul la longueur de l'élément, cos et sin pour la matrice
%       de rotation d'un élément barre à deux noeuds (x1,y1) (x2,y2)
% ds: longueur de l'élément
% c : cosinus de l'angle entre l'axe de l'élément et l'axe x
% s : sinus de l'angle
% XY: cordonnées des noeuds  XY = [x1,y1; x2,y2]
%
% A. Seghir, 06/08/04

  dx = XY(2,1) - XY(1,1);
  dy = XY(2,2) - XY(1,2);
  ds = sqrt(dx^2 + dy^2);
  c = dx/ds;
  s = dy/ds;

return
```

4.7.3 Application des conditions d'appuis

Les conditions d'appuis sont l'ensemble des déplacements nuls aux niveaux des appuis. Le vecteur `e` dans le fichier de données est utilisé pour spécifier les degrés de liberté à bloquer. Pour appliquer cette condition on élimine les lignes et les colonnes des matrices

K et M ainsi que du vecteur F puisque le chargement est saisi pour tous les nœuds y compris ceux d'appuis.

Il faut faire attention à l'application des commandes MATLAB `A(i,:)=[]` et `A(:,i)=[]` qui permettent de supprimer des lignes et des colonnes d'une matrice. A chaque colonne ou ligne supprimée la taille de la matrice est réduite et les indices sont décalés vers la gauche dans la cas d'une colonne et vers le haut dans le cas d'une ligne.

Ainsi, avant de procéder à la suppression d'une liste de degrés de liberté, il faut d'abord la trier selon un ordre croissant ou décroissant pour ensuite supprimer les lignes et les colonnes en la parcourant du plus grand au plus petit indice. De cette façon on a pas à se soucier des décalages dans les valeurs de la liste.

La commande MATLAB qui permet d'ordonner une liste ou un vecteur `L` est : `sort(L)`.

La fonction qui permet d'appliquer les conditions d'appuis est appelée `DelDOFs` et attend en arguments d'entré une variable `A` qui peut être une matrice carrée ou un vecteur colonne, et une liste `L` des degrés de liberté à supprimer. Son script est le suivant :

```
function A = DelDOFs(A,L)
% A = DelDOFs(A,L)
%
% A : matrice globale après assemblage
% L : liste des degrés de liberté à éliminer
%
% A. Seghir, le 01/08/04 modifié le 08/08/04

 L = sort(L);                          % tri de la liste des DDL
 n = length(L);                        % longueur de L

 if (size(A,2) == 1)                   % cas d'un vecteur
     for i = n:-1:1                    % parcourir L à partir du plus
grand indice
         A(L(i)) =[] ;                 % suppression de la composante
associée au DDL courant
     end
 else                                  % cas d'un matrice
     for i = n:-1:1
         A(L(i),:) =[] ;               % suppression de la ligne L(i)
         A(:,L(i)) =[] ;               % suppression de la colonne
L(i)
     end
```

```
 end
return
```

4.7.4 Résolution du système discret

Une fois les conditions aux limites sont appliquées, il ne reste qu'à résoudre le système discret. Nous avons deux types d'analyse :

1) Analyse statique qui consiste à déterminer les déplacements et les efforts internes dans les barres sous l'effet du chargement statique
2) Analyse dynamique qui consiste à déterminer le comportement dynamique de la structure et sa réponse à un spectre de réponse ou à un accélérogramme.

Dans le cas présent nous nous limiterons à l'analyse statique et au simple calcul des modes propres. La réponse dynamique fera l'objet d'un chapitre à part. La solution en déplacement s'obtient avec la commande `U = K\F` et les forces axiales dans les barres (forces internes) sont obtenues avec les systèmes élémentaires *Fe = Ke U*. Pour chaque élément il faut recalculer la matrice de rigidité *K*, extraire les déplacements élémentaires à partir de la solution *U* et effectuer le produit *Ke Ue* et enfin appliquer une rotation de repère pour revenir au repère local de la barre. Le vecteur *Ue* doit comprendre aussi les valeurs nulles des déplacements bloqués au niveaux des appuis (voir l'exemple). Enfin, les réactions et le chargement sont la somme des forces élémentaires de tous les éléments liées à l'appuis. La fonction MATLAB qui permet d'effectuer ces opérations est :

```
function [F,R] = TrussForces(t,p,A,E,U)
% [F,R]  = TrussForces(t,p,A,E,U)
% F : forces axiales dans les barres
% R : forces aux niveaux des nœuds = réactions dans le cas d'appuis
% t : table de connectivités des éléments
% p : table des coordonnées des noeuds
% A : sections des éléments
% E : modules d'élasticité
% U : solution en déplacement
% A. Seghir, le 07/08/04 modifié : 27/08/04

  net = size(t,1);                    % nombre d'éléments total
  nnt = 2*size(p,1);                  % nombre de noeuds total
  R = zeros(2*nnt,1);                 % Forces aux noeuds
  for ie = 1:net                      % boucle sur les éléments
```

```
      L  = localise(t(ie,:));            % table de localisation
      ke = truss2dKe(p(t(ie,:),:),A(ie),E(ie)); % matrice
élémentaire
      ue = U(L);                         % déplacements des noeuds
      fe = ke*ue;                        % forces élémentaires dans
(oxy)
      [Len,c,s] = EltLen(p(t(ie,:),:));  % les cosinus directeurs
      F(ie,:) = -(c*fe(1)+s*fe(2));    % rotation au repère local
      R(L) = R(L) + fe;                  % tous les éléments liés au
noeud
  end                                    % fin de boucle
return
```

On peut calculer les modes propres de la structure avec la fonction `eigs` qui résout le système $[K - \omega^2 M] \phi = 0$. Elle prend en argument d'entrée les deux matrice K et M et retourne comme résultat deux matrices : l'une orthogonale pour les vecteurs propres et l'autre diagonale pour les valeurs propres. Les modes propres sont donc les colonnes de la matrice des vecteurs propres et les périodes propres sont données par : $T = 2\pi/\omega$.

4.7.5 Programme principal

A présent toutes les fonctions nécessaires à la mise en œuvre d'un programme de calcul des structures à treillis ont été présentées. Il reste à les assemblées de préférence dans un fichier fonction comme celui-ci :

```
function [U,P,R, T,phi]= trussfem(ffd)
% [U,P,T,phi] = trussfem(ffd)
% résolution des systèmes d'assemblage de barres bidimensionnels
% U   : solution en déplacements nodaux
% P   : forces axiales dans les barres
% R   : réactions aux appuis
% T   : périodes propres de la structure
% phi : modes propres de la structure
% ffd : fichier fonction de données du problème
% A. Seghir, le 06/08/04 modifié le 27/10/04

[t,p,e,A,E,rho,F] = feval(str2func(ffd));      % fait appel au
fichier fonction
                                       % contenant les données
plotmesh(t,p,1,1,'b');                 % affiche la structure
chargée
```

```
[K,M] = truss2dKM(t,p,A,E,rho);          % calcul des matrices K et
M
K = DelDOFs(K,e);                        % Application des
conditions
M = DelDOFs(M,e);                        % aux limites
F = DelDOFs(F,e);
U = K \ F;                               % solution statique
U = AddDOFs(U,e);                        % ajout des DDL des noeuds
encastré
[P,R] = TrussForces(t,p,A,E,U);          % forces dans les barres et
les noeuds
[phi, omega2] = eigs(K,M);               % modes propres
omega = sqrt(diag(omega2));              % pulsations
T = 2*pi./sort(omega);                   % périodes propres par
ordre croissant

                                         % affichage des résultats
net = size(t,1); nnt = size(p,1);
disp(' Résultats de calcul de la structure d''assemblage de barres')
disp([' fichier de données : ',ffd])
disp([' Nombre de barres : ',num2str(net)])
disp([' Nombre de noeuds : ',num2str(nnt)])

disp(sprintf('\n Déplacements aux noeuds :'))
disp(sprintf(' Noeud\t\t Ux\t\t\t\t Uy '))
for i=1:nnt
    disp(sprintf(' %d\t\t\t%+5.4f\t\t\t%+5.4f',i,U(localise(i))))
end

disp(sprintf('\n Efforts dans les barres :'))
disp(sprintf(' Barre \t\t P'))
for i=1:net
    disp(sprintf(' %d\t\t\t%+1.4E',i,P(i)))
end

disp(sprintf('\n Réactions aux appuis :'))
disp(sprintf(' Appuis\t\t Rx\t\t\t\t\t Ry '))
for i=1:nnt
    L = localise(i);
    if find(e==L(1))
```

```
        disp(sprintf(' %d\t\t\t%+1.4E\t\t%+1.4E',i,R(L)))
    end
end

disp(sprintf('\n Périodes propres de la structure :'))
disp(sprintf(' mode\t\tT'))
for i=1:size(T)
    disp(sprintf(' %d\t\t\t%5.4f',i,T(i)))
end
plotdeforme(U,p,t,10)                  % dessine la structure
déformée
```

Remarquer comment le fichier de données est appelé par cette fonction. La variable `ffd` doit être de type caractère (`string`), `str2func` permet de construire une adresse de fonction à partir d'un nom et `feval` la fait évaluer. Pour utiliser `trussfem`, il faut donc préparer un fichier de données semblable à `expl2barres.m` et utiliser la commande :

```
>> [U,P,T] = trussfem('expl2barres') ;
```

Son exécution affiche les lignes suivantes comme résultats :

```
Résultats de calcul de la structure d'assemblage de barres
fichier de données : expl2barres
Nombre de barres : 2
Nombre de noeuds : 3

Déplacements aux noeuds :
Noeud       Ux           Uy
1          +0.0000      +0.0000
2          +0.0000      +0.0000
3          +0.0000      -0.0090

Efforts dans les barres :
Barre         P
1            +2.8333E+006
2            +2.8333E+006

Réactions aux appuis :
Appuis      Rx              Ry
1          -2.2667E+006    +1.7000E+006
2          +2.2667E+006    +1.7000E+006
```

```
Périodes propres de la strucutre :
Mode       T
1          0.0011
2          0.0014
```

4.8 TP N°3

Modéliser et calculer la structure ci-dessous en utilisant le programme `trussfem` et le logiciel `SAP2000`. Toutes les barres de la structure ont une section droite $A = 6cm \times 6cm$ et sont constituées d'un matériau de masse volumique $\rho = 7800\ Kg/m^3$ et de module d'élasticité $E = 210000\ MPa$. Les distances sont données en mètre et les chargements en *KN* . Donner les déplacements des nœuds, les réactions aux appuis et les forces internes dans les barres.

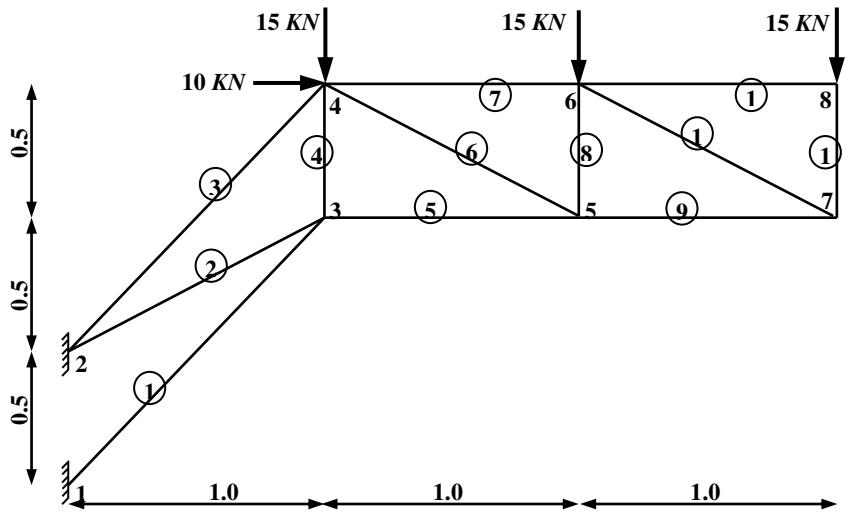


***Fig.* 2.6 : Structure à calculer en TP**

chapitre 5

Elément Poutre

5.1 Equation gouvernante

L'élément poutre est utilisé pour reprendre, en plus de l'effort axial comme l'élément barre, un chargement perpendiculaire à son axe. On retrouve les poutres dans beaucoup de structures de génie civil et de constructions mécaniques. Les cas les plus fréquents sont les portiques constituant les bâtiments d'habitation, les ponts … etc.

On considère comme poutres les pièces élancées (en béton armé ou en acier), qui ont une dimension très grande par rapport aux deux autres et qui travaillent généralement en flexion.

La formulation de l'élément poutre peut être obtenue en se basant sur le théorie de la résistance des matériaux ; on considère une poutre de section A et de longueur L soumise à un chargement $q(x)$ variant le long de son axe longitudinal tel que montrée sur la figure 3.1 ci-dessous :

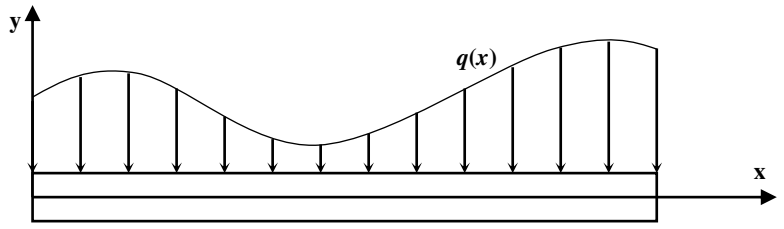


***Fig.* 3.1 : poutre chargée**

Sous l'effet du chargement la poutre fléchie et se déplace verticalement d'un déplacement $v(x)$. On suppose qu'après cette déformation, les sections droites restent planes et

perpendiculaires à la ligne moyenne ; elles subissent de ce fait une petite rotation d'angle θ dans le plan (*oxy*). Considérons un élément *dx* de la poutre délimité par deux sections voisines, l'une droite et l'autre inclinée.

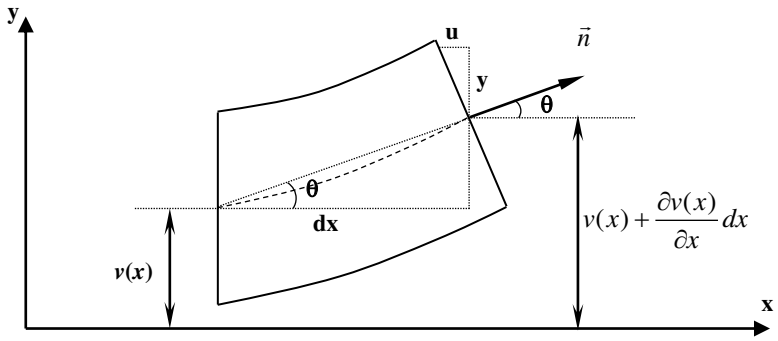


Fig. 3.2 : déformation d'une section

La rotation de la section déformée est la tangente de la ligne moyenne courbée :

$$\theta = \frac{\partial v}{\partial x} \tag{3.1}$$

A cause de la rotation, les points de la section subissent un déplacement horizontal *u* variant linéairement de la fibre inférieure à la fibre supérieure. En un point de la section ce déplacement vaut :

$$u = -\theta\ y = -y\frac{\partial v}{\partial x} \tag{3.2}$$

où *y* désigne la distance à partir de la ligne moyenne (centre de la section)

Dans le cadre de l'hypothèse des petites déformations, la déformation axiale suivant *x* le long de la section est :

$$\varepsilon_x = \frac{\partial u}{\partial x} \tag{3.3}$$

Si on désigne par E le module d'élasticité du matériau de la poutre, la loi de Hooke donne la répartition des contraintes le long de la section :

$$\sigma_x = E\,\varepsilon_x = E\frac{\partial u}{\partial x} = -E\,y\,\frac{\partial^2 v}{\partial x^2} \tag{3.4}$$

Le moment créé par ces contraintes doit équilibrer le moment de flexion M créé par le chargement extérieur :

$$M - \int_s \sigma.y.ds = 0 - E\frac{\partial^2 v}{\partial x^2}\int_s y^2\,ds \quad M = E\frac{\partial^2 v}{\partial x^2}\int_s y^2\,ds \tag{3.5}$$

avec s désigne l'aire de la section droite. On posant I_z le moment d'inertie par rapport à l'axe z perpendiculaire au plan (xy) $I = \int_s y^2\,ds$, l'expression du moment devient :

$$M = EI\frac{\partial^2 v}{\partial x^2} \tag{3.6}$$

Considérons maintenant l'équilibre statique d'un élément dx.

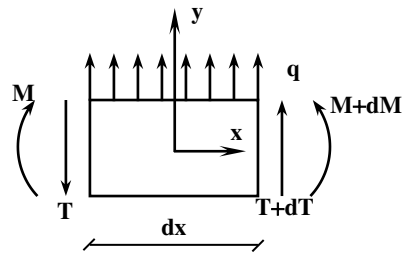


Fig. 3.3 : Equilibre statique

La somme des moments par rapport à son centre de gravité donne :

$$-M + T\frac{dx}{2} + (T + dT)\frac{dx}{2} + (M + dM) = 0$$

$$Tdx + dT\frac{dx}{2} + dM = 0$$

En négligeant les termes du second ordre, on obtient la relation entre l'effort tranchant et le moment fléchissant :

$$T = -\frac{dM}{dx} = -\frac{\partial}{\partial x}(EI\frac{\partial^2 v}{\partial x^2}) \tag{3.7}$$

Equilibre des forces verticales pour un chargement positif dans le sens de l'axe *y* (figure 3.3):

$$T + dT + q\,dx - T = 0 \tag{3.8}$$

donne la relation entre le chargement *q* et l'effort tranchant *T* et relie le chargement au déplacement *v* par :

$$q = -\frac{dT}{dx} = \frac{\partial^2}{\partial x^2}(EI\frac{\partial^2 v}{\partial x^2}) \tag{3.9}$$

Cette équation traduit l'équilibre statique de la poutre. Dans le cas d'un mouvement dynamique, il faut ajouter dans l'équation (3.8) un terme traduisant les forces d'inertie :

$$F_i = m\gamma = \rho\, A\, dx\frac{\partial^2 u}{\partial t^2} \tag{3.10}$$

avec ρ est la masse volumique du matériau, *A* la section de la poutre et t représente le temps.

Les équations (3.8) et (3.9) deviennent :

$$T + dT + q\,dx - T = Fi \tag{3.11}$$

$$\rho A\frac{\partial^2 v}{\partial t^2} + \frac{\partial}{\partial x^2}(EI\frac{\partial^2 v}{\partial x^2}) = q(x) \tag{3.12}$$

L'équation (3.11) est l'équation d'Euler-Bernoulli pour la flexion des poutres. Le déplacement *v* est fonction de la coordonnée *x* le long de l'axe de la poutre et du temps *t*.

5.2 Formulation de l'élément

5.2.1 Formulation variationnelle

En désignant par L la longueur de la poutre et en prenant δv la fonction poids, la formulation variationnelle forte associée à l'équation (3.12) s'écrit :

$$\int_0^L \delta v\, \rho A \frac{\partial^2 v}{\partial t^2} dx + \int_0^L \delta v \frac{\partial^2}{\partial x^2}(EI \frac{\partial^2 v}{\partial x^2})\, dx = \int_0^L \delta v\, q(x)\, dx \tag{3.13}$$

La forme intégrale faible s'obtient avec deux intégrations par parties du second terme.

$$\int_0^L \delta v \frac{\partial^2}{\partial x^2}(EI \frac{\partial^2 v}{\partial x^2})\, dx = -\int_0^L \frac{\partial \delta v}{\partial x} \frac{\partial}{\partial x}(EI \frac{\partial^2 v}{\partial x^2})\, dx + \left[\delta v \frac{\partial}{\partial x}(EI \frac{\partial^2 v}{\partial x^2}) \right]_0^L \tag{3.14a}$$

$$\int_0^L \frac{\partial \delta v}{\partial x} \frac{\partial}{\partial x}(EI \frac{\partial^2 v}{\partial x^2})\, dx = -\int_0^L \frac{\partial^2 \delta v}{\partial x^2} EI \frac{\partial^2 v}{\partial x^2}\, dx + \left[\frac{\partial \delta v}{\partial x} EI \frac{\partial^2 v}{\partial x^2} \right]_0^L \tag{3.14b}$$

Compte tenu des expressions (3.6) et (3.7), les seconds termes représentent la différence des chargements en forces (T_0 et T_L) et en moments (M_0 et M_L) appliqués aux extrémités de la poutre. De plus, en peut remplacer la dérivée des perturbations des déplacements par une perturbation des rotions (équation 3.1) : $\partial \delta v / \partial x = \delta\theta$.

On écrit ainsi les conditions aux limites comme suit :

$$\left[\frac{\partial \delta v}{\partial x} EI \frac{\partial^2 v}{\partial x^2} \right]_0^L = \delta\theta \big|_{x=0} M_0 - \delta\theta \big|_{x=L} M_L \tag{3.15a}$$

$$\left[\delta v \frac{\partial}{\partial x}(EI \frac{\partial^2 v}{\partial x^2}) \right]_0^L = \delta v \big|_{x=L} T_L - \delta v \big|_{x=0} T_0 \tag{3.15b}$$

En substituant maintenant (3.15) dans (3.14) et le résultat dans (3.13) on obtient l'expression de la forme variationnelle faible :

$$\int_0^L \delta v\, \rho A \frac{\partial^2 v}{\partial t^2} dx + \int_0^L \frac{\partial^2 \delta v}{\partial x^2} E\, I \frac{\partial^2 v}{\partial x^2} dx + \delta\theta \big|_{x=L} M_L + \delta v\big|_{x=L} T_L - \delta v\big|_{x=0} T_0 - \delta\theta \big|_{x=0} M_0 = \int_0^L \delta v\, q(x)\, dx \tag{3.16}$$

5.2.2 Discrétisation

Pour la discrétisation de cette équation on considère un élément à deux nœuds : un nœud à chaque extrémité de la poutre. La présence de dérivées d'ordre deux impose l'utilisation de polynômes quadratiques ou plus. En outre, on voit que l'expression des conditions aux limites fait intervenir la rotation aux extrémité, il est donc plus intéressant de prendre deux degrés de liberté par nœuds dans le but d'assurer en même temps la continuité des déplacements et de leurs dérivées qui sont les rotations. Le nombre de degrés de liberté atteint ainsi quatre et le polynôme d'interpolation doit être cubique (quatre constantes). Le vecteur des déplacements et rotations élémentaires s'écrit donc comme suit :

$$U_n = < v_1 \quad \theta_1 \quad v_2 \quad \theta_2 >^T \tag{3.17}$$

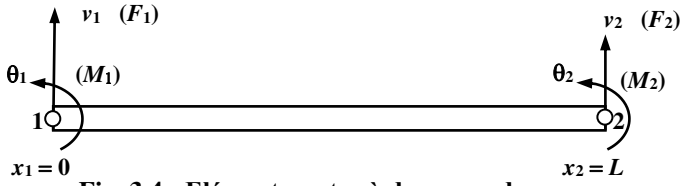


Fig. 3.4 : Elément poutre à deux nœuds

Les déplacements et les rotations le long de la poutres sont approximés par :

$$v(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 \tag{3.18a}$$

$$\theta(x) = a_1 + 2a_2 x + 3a_3 x^2 \tag{3.18b}$$

L'évaluation de ces polynômes aux nœuds donne :

$$v(0) = a_0 = v_1 \ ; \qquad \theta(0) = a_1 = \theta_1 \tag{3.19a}$$

$$v(L) = v_2 = v_1 + \theta_1 L + a_2 L^2 + a_3 L^3 \;;\; \theta(L) = \theta_2 = \theta_1 + 2a_2 L + 3a_3 L^2 \tag{3.19b}$$

La résolution de (3.19b) pour a_2 et a_3 donne :

$$a_2 = \frac{3}{L^2}(v_2 - v_1) - \frac{1}{L}(2\theta_1 - \theta_2) \;;\quad a_3 = \frac{2}{L^3}(v_1 - v_2) + \frac{1}{L^2}(\theta_1 + \theta_2) \tag{3.20}$$

En remplace ces paramètres dans l'équation (3.18) et après arrangement des termes on écrit l'interpolation nodale des déplacements sous la forme :

$$v(x) = N_1(x)\, v_1 + N_2(x)\, \theta_1 + N_3(x)\, v_2 + N_4(x)\, \theta_2 \tag{3.21}$$

Les fonctions de forme N_i sont appelées polynômes d'Hermite, leurs expressions sont :

$$N_1(x) = 1 - \frac{3x^2}{L^2} + \frac{2x^3}{L^3} \;;\quad N_2(x) = x - \frac{2x^2}{L} + \frac{x^3}{L^2} \;;$$

$$N_3(x) = \frac{3x^2}{L^2} - \frac{2x^3}{L^3} \;;\quad N_4(x) = \frac{x^3}{L^2} - \frac{x^2}{L} \tag{3.22}$$

On peut vérifier que la somme des fonctions de forme associées aux déplacements est égale à l'unité : $N_1+N_3 = 1$. Elles prennent aussi des valeurs égales à un aux nœuds qui leurs correspondent et des valeurs nulles aux nœuds opposés. Cette remarque n'est pas valable pour les fonctions associées aux rotations puisque les rotations sont elles mêmes des dérivées des déplacements. Le programme MATLAB qui permet de calculer et de tracer les fonctions forme d'Hermite est le suivant :

```
clear, clc
syms x L real                    % déclarer x et L symboliques réels

P  = inline('[1  x  x^2  x^3]')  % polynôme du 3ème degré
dP = inline(diff(P(x)))          % dérivée du polynôme
Pn = [ P(0); dP(0); P(L) ; dP(L) ]    % évaluation au noeuds

N  = inline(( P(x) * inv(Pn)))   % fonctions de forme et
dN = inline((dP(x) * inv(Pn)))   % leurs dérivées
                                 % graphes de N et dN
t = 0:0.01:1;
subplot(2,1,1), plot(t, N(1,t')), title(' Fonctions de forme N1 N2 N3
N4 ')
subplot(2,1,2), plot(t,dN(1,t')), title(' Dérivées dN1 dN2 dN3 dN4 ')
```

La figure ci-dessous montre les courbes des fonctions de forme et de leurs dérivées pour un élément poutre de longueur unitaire ($L = 1$) :

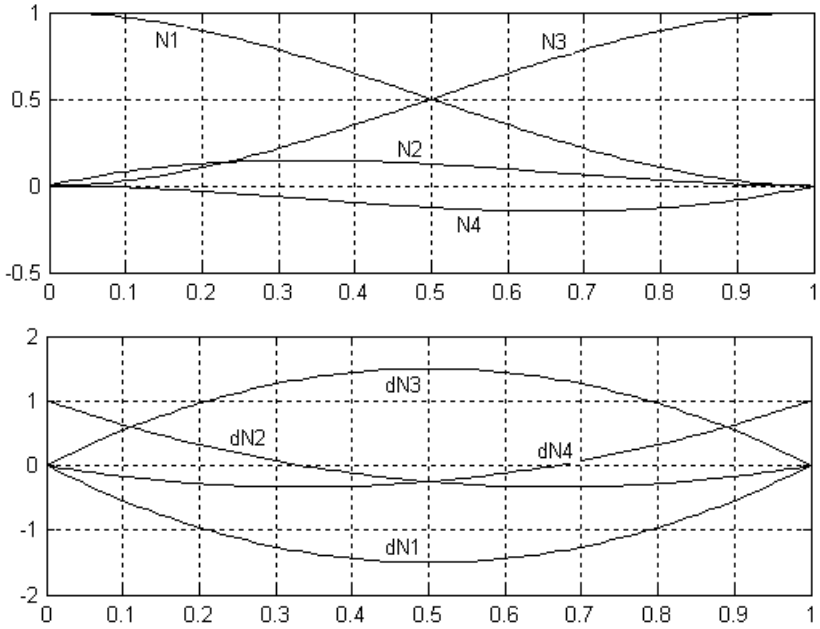


Fig. 3.5 : Fonctions de forme d'un élément poutres et leurs dérivées

5.2.3 Matrices élémentaires

On remplace maintenant dans la forme variationnelle (3.16) le déplacement v par son approximation (3.21), on obtient pour les perturbations :

$$\delta v = \delta U_n^T \, N^T \;\; ; \;\; \delta\theta = \delta U_n^T \, \frac{dN^T}{dx} \quad \text{avec} \quad \delta U_n^T = < \delta v_1 \quad \delta\theta_1 \quad \delta v_1 \quad \delta\theta_2 > \tag{3.23}$$

Les dérivées deviennent :

$$\frac{\partial^2 v}{\partial x^2} = \frac{d^2 N}{dx^2} U_n \;\; ; \;\; \frac{\partial^2 v}{\partial t^2} = N \frac{d^2 U_n}{dt^2} = N \ddot{U}_n \tag{3.24}$$

$$\text{avec : } \frac{d^2N}{dx^2}(x) = \frac{1}{L^3} < 12x - 6L, \quad 6Lx - 4L^2, \quad 6L - 12x, \quad 6Lx - 2L^2 > \tag{3.25}$$

Compte tenu des valeurs des fonctions *N* et *dN* aux nœuds (Fig. 3.5), les conditions aux limites s'écrivent :

$$\begin{aligned} \delta v\,\big|_{x=L} T_L - \delta v\,\big|_{x=0} T_0 &= \delta U_n^T (<0 \quad 0 \quad 1 \quad 0>^T T_L - <1 \quad 0 \quad 0 \quad 0>^T T_0) \\ &= \delta U_n^T <-T_0 \quad 0 \quad T_L \quad 0>^T \end{aligned} \tag{3.26a}$$

$$\begin{aligned} \delta\theta\,\big|_{x=L} M_L - \delta\theta\,\big|_{x=0} M_0 &= \delta U_n^T (<0 \quad 0 \quad 0 \quad 1>^T M_L - <0 \quad 1 \quad 0 \quad 0>^T M_0) \\ &= \delta U_n^T <0 \quad -M_0 \quad 0 \quad M_L>^T \end{aligned} \tag{3.26b}$$

Ces conditions correspondent aux chargement extérieur appliqué aux nœuds c'est-à-dire au niveau des jonctions entre les éléments telles que les jonctions poteaux–poutres par exemple. Le vecteur élémentaire F_n des forces et des moments concentrées aux nœuds s'écrit comme suit :

$$F_c = <-T_0 \quad -M_0 \quad T_L \quad M_L>^T \tag{3.26c}$$

Le chargement F_e réparti sur l'élément poutre correspond au second terme de la forme variationnelle (3.16) :

$$\int_0^L \delta v\, q(x)\, dx = \delta U_n^T \int_0^L N^T\, q(x)\, dx \;;\; F_e = \int_0^L N^T\, q(x)\, dx \tag{3.27}$$

Ce chargement est fonction de la répartition $q(x)$. Dans les cas simples on peut intégrer et donner l'expression explicite de F_e, sinon, il est possible de subdiviser la poutre (ou l'élément) en plusieurs petits éléments de telle sorte à pouvoir remplacer le chargement réparti q par deux forces équivalentes concentrées aux extrémités.

On donne ci-dessous l'expression de F_e pour un cas de chargement trapézoïdal variant de q_0 à q_L, Les chargements uniformes et triangulaires ne sont que des cas particuliers du chargement trapézoïdale.

$$q(x) = q_0 + (q_L - q_0)\frac{x}{L} \;; \tag{3.28a}$$

$$F_e = \frac{L}{60} < (9q_L + 21q_0),\quad L(2q_L + 3q_0),\quad (21q_L + 9q_0),\quad -L(3q_L + 2q_0) >^T \quad (3.28b)$$

Il reste maintenant deux termes intégrales dans l'équation (2.16) à discrétiser :

$$\int_0^L \frac{\partial^2 \delta v}{\partial x^2} E\, I \frac{\partial^2 v}{\partial x^2}\, dx = \int_0^L \delta U_n^T \frac{d^2 N^T}{dx^2} E\, I \frac{d^2 N}{dx^2} U_n^T\, dx \quad (3.29a)$$

$$\int_0^L \delta v\, \rho A \frac{\partial^2 v}{\partial t^2} dx = \int_0^L \delta U_n^T N^T \rho\, N\, \ddot{U}_n\, dx \quad (3.29b)$$

L'élimination de δU_n^T de l'équation intégrale discrète donne les deux matrices masse M_e et rigidité K_e :

$$K_e = \int_0^L \frac{d^2 N^T}{dx^2} E\, I \frac{d^2 N}{dx^2}\, dx \quad (3.29)$$

$$M_e = \int_0^L N^T \rho A\, N\, dx \quad (3.30)$$

Si la poutre est faite d'un même matériau homogène et de même section (E, I et ρ sont constantes), alors les expressions explicites des matrices élémentaires peuvent être obtenues et s'écrivent comme suit :

$$K_e = \frac{EI}{L^3}\begin{bmatrix} 12 & 6L & -12 & 6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ 6L & 2L^2 & -6L & 4L^2 \end{bmatrix};\quad M_e = \frac{\rho AL}{420}\begin{bmatrix} 156 & 22L & 54 & -13L \\ 22L & 4L^2 & 13L & -3L^2 \\ 54 & 13L & 156 & -22L \\ -13L & -3L^2 & -22L & 4L^2 \end{bmatrix}$$

(4.31)

La même remarque concernant la matrice masse de l'élément barre peut être faite pour l'élément poutre. On peut associer la moitié de la masse totale de l'élément aux degrés de translation de chaque nœud. On prends uniquement les translations parce que les rotations ne produisent pas de forces d'inertie. La matrice masse concentrée s'écrit donc comme suit :

$$M_e = \frac{\rho AL}{2}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.32}$$

Il est à noter que les deux matrices concentrée et répartie donnent la même masse totale avec la somme des composantes associées aux degrés de liberté de translation : $M(1, 1) + M(1, 3) + M(3, 1) + M(3, 3) = \rho AL$.

5.3 Exemple d'une poutre console

5.3.1 Cas d'un chargement concentré à l'extrémité libre

Une poutre en béton armé de masse volumique $\rho = 2.5\ tonnes/m^3$ et de module d'élasticité $E = 32000\ MPa$ est encastrée à l'une de ses extrémités et chargée à l'autre extrémité d'une charge concentrée $F_y = -90\ KN$ et d'un moment $M = 60\ KN\ m$

Les caractéristiques géométriques de la poutre sont :

- longueur $L = 150\ cm$
- largeur $b = 30cm$
- hauteur $h = 40\ cm$

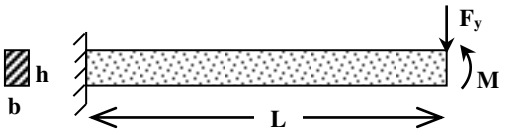


Fig. 3.6 : Console chargée à l'extrémité

L'inertie de la poutre est : $I = 3 \times 4^3 / 12 = 16\ cm^4$;

La matrice de rigidité s'écrit :

$$K_e = \frac{3.2 \times 16}{1.5^3}\frac{3}{2}\begin{bmatrix} 8 & 6 & -8 & 6 \\ 6 & 6 & -6 & 3 \\ -8 & -6 & 8 & -6 \\ 6 & 3 & -6 & 6 \end{bmatrix} MN/m$$

Le vecteur force selon l'équation (3.26) est : $F = 10^3 < 0 \quad 0 \quad -90 \quad 60 >^T$

Le système élémentaire s'écrit :

$$K = \frac{3.2\times 16}{1.5^3}\cdot 10^6 \cdot \frac{3}{2}\begin{bmatrix} 8 & 6 & -8 & 6 \\ 6 & 6 & -6 & 3 \\ -8 & -6 & 8 & -6 \\ 6 & 3 & -6 & 6 \end{bmatrix}\begin{Bmatrix} v_1 \\ \theta_1 \\ v_2 \\ \theta_2 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ -90000 \\ 60000 \end{Bmatrix}$$

Les conditions d'encastrement ($v_1 = 0$; $\theta_1 = 0$) réduit le système à deux équations :

$$K = 2.27556\cdot 10^7 \begin{bmatrix} 8 & -6 \\ -6 & 6 \end{bmatrix}\begin{Bmatrix} v_2 \\ \theta_2 \end{Bmatrix} = \begin{Bmatrix} -90000 \\ 60000 \end{Bmatrix}$$

La solution donne $v_2 = -0.6592\ mm$; $\theta_2 = -0.2197\cdot 10^{-3}$; un déplacement négatif vers le bas et une rotation négative dans le sens inverse au sens trigonométrique (Fig. 3.4)

Les efforts élémentaires sont :

$$K = 2.27556\cdot 10^7 \begin{bmatrix} 8 & 6 & -8 & 6 \\ 6 & 6 & -6 & 3 \\ -8 & -6 & 8 & -6 \\ 6 & 3 & -6 & 6 \end{bmatrix}\begin{Bmatrix} 0 \\ 0 \\ -0.6592 \\ -0.2197 \end{Bmatrix} 10^{-3} = \begin{Bmatrix} F_1 \\ M_1 \\ F_2 \\ M_2 \end{Bmatrix} = 10^3 \begin{Bmatrix} 90 \\ 75 \\ -90 \\ 60 \end{Bmatrix}$$

La deux premières composantes sont les réactions à l'appuis :

$R_y = 90\ KN$; $M_z = 90 \times 1.5 - 60 = 75\ KN\,m$.

Le moment est positif à cause de la convention de signe adoptée à la figure 3.4 : Au nœud 1, le moment positif tend la fibre supérieure pour avoir une rotation dans le sens trigonométrique. Les deux dernières composantes correspondent aux efforts nodaux appliqués à l'extrémité libre.

Il est à remarquer que la solution obtenue que ce soit pour la flèche ou les réactions est une solution exacte. Ceci est du au fait que la flèche est une fonction en x^3 et les fonctions de forme sont des polynômes du troisième degré. Il en résulte que les polynômes d'Hermite permettent d'obtenir la solution exacte.

5.3.2 Cas d'un chargement réparti

On considère maintenant la même charge précédente répartie uniformément le long de la poutre, soit :

$q = 90/1.5 = 60\ KN/\ell$. ;

et on garde le même moment à l'extrémité libre.

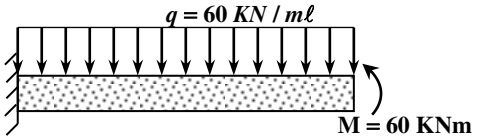


Fig. 3.7 : Chargement réparti

Le vecteur force du au chargement q, s'écrit, selon l'équation (3.28),

(avec $q_0 = q_1 = q = -60\ KN/m\ell$) :

$F_q = 10^3 < -45 \quad -11.25 \quad -45 \quad 11.25 >^T$ (vérifier la somme de F_i)

Le vecteur forces concentrées du au moment est : $F_c = 10^3 < 0 \quad 0 \quad 0 \quad 60 >^T$

Le vecteur force total est donc : $F = 10^3 < -45 \quad -11.25 \quad -45 \quad 71.25 >^T$

La matrice de rigidité étant la même, la solution du système réduit donne :

$v_2 = 0.5768\ mm$ et $\theta_2 = 1.0986 \cdot 10^{-3}$.

Le vecteur réaction élémentaire :

$R = KU = 10^3 < 45 \quad -3.75 \quad -45 \quad 71.25 >^T$

Ces efforts sont représentés dans la figure 3.8 ci-contre.

Ils sont une partie des réactions qui correspond au déplacement et à la rotation de l'extrémité libre. La présence de la force q répartie sur l'élément donne naissance à des forces et à des moments nodaux (vecteur F) auxquels correspond une seconde partie des réactions.

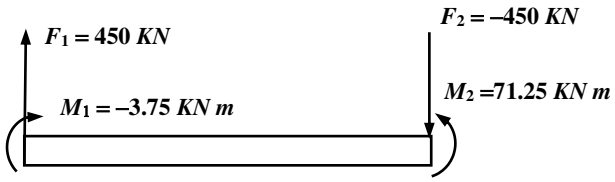


Fig. 3.8 : Efforts aux nœuds

La réaction totale correspond donc à la somme des deux type de réactions :

$R_y = R_1 + (-F_{q1}) = 90\ KN.$; $M_z = R_2 + (-F_{q2}) = 7.5\ KN\,m.$

A l'extrémité libre la somme des deux réactions s'annulent en force et leur somme égale le moment appliqué :

$M = R_4 + (-F_{q4}) = 60\ KN\,m.$

D'une façon générale, le moment s'exerçant sur une section de coupure à une distance x de l'encastrement est :

$$M(x) = -M_z + R_y\, x - \tfrac{1}{2}\, q\, x^2$$
$$= -7.5 + 90\, x - 30\, x^2 \ \ [KN\,m]$$

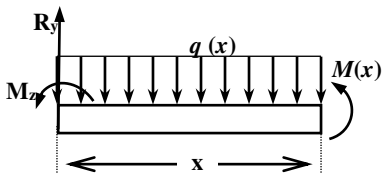


Fig. 3.9 : Le moment dans l'élément

à la distance $x = 0.5\ m$: $M(0.5) = 30.0\ KN\,m$

à la distance $x = 1.0\ m$: $M(1.0) = 52.5\ KN\,m$

à la distance $x = 1.5\ m$: $M(1.5) = 60.00\ KN\,m.$

Ce moment s'annule à la position $x = \dfrac{3}{2} - \sqrt{2} = 0.08579$

Remarque : Dans le cas d'une charge trapézoïdale (expression 3.28a), il faut remplacer $\frac{1}{2}\, q\, x^2$ par la somme du moment d'une charge uniforme q_0 et celui d'une la charge triangulaire $(q_L - q_0)\, x/L$ soit :

$$M_q = \tfrac{1}{2} q_0\, x^2 + (q_L - q_0)\frac{x^3}{6L}$$

La matrice masse concentrée et répartie de la poutre sont :

$$M_c = \begin{bmatrix} 225 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 225 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \ ; \ M_r = \frac{450}{1680}\begin{bmatrix} 624 & 132 & 216 & -78 \\ 132 & 36 & 78 & -27 \\ 216 & 78 & 624 & -132 \\ -78 & -27 & -132 & 36 \end{bmatrix}$$

Les périodes propres de la poutres sont données par $T = 2\pi/\omega$ avec ω solution de l'équation : $det(K - \omega^2 M) = 0$.

Pour une matrice masse concentrée : $T_1 = 0.01397\ s$; $T_2 = 0.0\ s$

Pour une matrice masse cohérente : $T_1 = 0.00968\ s$; $T_2 = 0.00098\ s$

5.3.3 Programme MATLAB

Si on veut faire ces calcules avec MATLAB, on peut prévoir un script semblable à celui-ci :

```
%--------------------------------------------------------------------
% console.m
% script pour résoudre le problème d'une poutre console
%--------------------------------------------------------------------
clear, clc                      % efface les variables et la fenêtre
L = 1.5;                        % Longueur de la console
b = 0.3;                        % Largeur de section
h = 0.4;                        % Hauteur de section

E   = 3.2e10;                   % Module d'élasticité
rho = 2.5e03;                   % Masse volumique

A = b*h;                        % Section
I = b*h^3/12;                   % Inertie

q = -60000;                     % Charge répartie
P = -90000;                     % Charge concentrée
Mz=  60000;                     % Moment concentré

Fq = q*L/60*[30 5*L 30 -5*L]';  % Vecteur force associé à q
Fm = [0 0 0 Mz]';               % Vecteur force associé à Mz
Fp = [0 0 P 0 ]';               % Vecteur force associé à P
F1 = Fm + Fp;                   % Premier cas de chargement
F2 = Fm + Fq;                   % Deuxième cas de chargement

% Matrice de rigidité
K = E*I/L^3 * [ 12    6*L    -12    6*L
                6*L   4*L^2  -6*L   2*L^2
               -12   -6*L     12   -6*L
                6*L   2*L^2  -6*L   4*L^2];

% Matrice masse concentrée
Mc = rho*A*L/2 * diag([1 0 1 0]);
```

```
% Matrice masse cohérente
Mr = rho*A*L/420 * [ 156    22*L   54    -13*L
                     22*L   4*L^2  13*L  -3*L^2
                     54     13*L   156   -22*L
                    -13*L  -3*L^2 -22*L   4*L^2];

U1 = K([3 4],[3 4])\F1([3 4])     % solution pour le 1er cas
U2 = K([3 4],[3 4])\F2([3 4])     % solution pour le 2em cas
R1 = K*[0;0;U1]                   % Réactions pour le 1er cas
R2 = K*[0;0;U2]-Fq                % Réactions pour le 1er cas

syms x real;                      % Moment en fonction de x
Mzx = inline(-R2(2) + R2(1) * x + 0.5* q * x.^2)
Mzx([0 0.5 1 1.5]')

% périodes propres cas d'une masse concentrée
Tc = 2*pi*eig(K([3 4],[3 4]),Mc([3 4],[3 4])).^-0.5

% périodes propres cas d'une masse répartie
Tr = 2*pi*eig(K([3 4],[3 4]),Mr([3 4],[3 4])).^-0.5
```

5.3.4 Modèle SAP

Le modèle SAP de cette poutre console peut être réalisé de la manière suivantes :

1) Créer un élément `Frame` entre les points (`X0.0,Y0.0`) et (`X1.5,Y0.0`). Voir les étapes 1 à 6 décrites pour l'exemple 2.5. Encastrer le nœud au point (`0,0`)
2) Dans la fenêtre de définition du matériau, mettre `3.2e10` pour `Elasticity Modulus`, `2500` pour `Mass per Unite Volume` et `0` dans tous les autres champ de saisie.
3) Pour la section, il faut mettre `0.3` pour `Depth (t3)` et `0.4` pour `Width (t2)` puisque les le modèle est plan et la hauteur de la poutre est dans le sens `Y` au lieu du sens `Z`.
4) Définir les trois cas de charges statiques `FP`, `FQ` et `FM` dans la fenêtre `Define Static Load Case Names` ; choisir le type `Live` et saisir `0` pour `Self Weight Multiplier`.
5) Une fois les charge définies, on peut faire les combinaisons `FM+FQ` et `FM+FP` avec le menu `Define/Load Combinations`. Cliquer sur le bouton `Add New Combo`. Dans la fenêtre `Load Combination Data`, saisir `FQM` pour les deux champs `Load Combination Name` et `Title`, laisser `ADD` pour `Load Combination Type`. Dans le

volet `Define Combination`, choisir `FQ Load Case` pour `Case Name` et garder `1` comme `Sccale Factor` puis appuyer sur le bouton `Add`. Ajouter de la même manière `FM` et valider avec `OK`. Définir maintenant de la même façon la combinaison `FPM`. Utiliser le bouton `Delete` pour effacer les charges précédentes ou le bouton `Modify` pour les modifiées. Fermer les deux fenêtres en validant dans les deux cas avec `OK`.

6) Sélectionner l'élément pour lui assigner la charge répartie avec le menu `Assign/Frame Static Loads/Point and Uniform`. Choisir `FQ` pour `Load Case Name`, `Forces` et `GlobalY` pour `Load Type and Direction` et saisir la valeur `-60000` dans le champs `Uniforme Load`.
7) Sélectionner maintenant le nœud de l'extrémité libre pour lui affecter la charge `P` avec le menu `Assign/Joint Static Loads/Forces`. Choisir `FQ` pour `Load Case Name` et saisir `-90000` pour `Force GlobalY`. Sélectionner une autre fois le nœud pour lui affecter le moment `M`. choisir maintenant `FM` pour `Load Case Name` et saisir `60000` pour `Moment GlobalZZ`. Le moment n'est visible dans la fenêtre de dessin que si elle est en mode `3D` ou dans le plan `X-Z`, changer de vue si nécessaire.
8) Dans la fenêtre `Analysis Options` cocher uniquement `UY` et `RZ` pour `Available DOF's`, activer aussi l'option `Dynamic Analysis` et appuyer sur `Set Dynamic Parameters` pour mettre `Number of Modes` à `1`.
9) Lancer l'analyse. Avant de ferme la fenêtre d'avancement des calculs, on peut lire : `Period = 0.014265`, le second mode de vibration est supprimé par le SAP puisqu'il est à périodes nulle.
10) Les résultats peuvent être affichés selon les combinaisons définies ou selon les charges seules.

Remarque :

Remarquer la définition des charges et leurs combinaisons dans les étapes 4 à 6 et comparer avec le script `console.m`. Au fait cette remarque se généralise pour toutes les déclarations de données, l'interface graphique de SAP2000 crée un fichier de données qui est sauvegardé avant l'analyse du modèle pour être chargé et lu par les modules du logiciel. Une version au format `ASCII` peut être créée avec le menu `File/Export/SAP2000.S2K`. Il est intéressant de faire cette opération et d'appeler le

fichier par exemple `console.s2k` pour le charger avec un éditeur de textes comme `WordPad` ou `NotePad` afin de l'étudier.

5.4 Devoir N°6

Retrouver les expressions des matrices de rigidité et de masse de l'élément poutre (équations : 4.31)

Force F dans le cas d'une charge repartie triangulaire Δ et