Jacquier Pierre
Blondé-Weinmann Cyril

SYSTÈME DISCRET BIDIMENSIONNEL

Semestre Printemps 2017

Université de Technologie de Belfort-Montbéliard

Table des matières

Ce rapport détaille la démarche suivie lors de l'élaboration d'un programme en langage *C#* permettant la résolution de systèmes physiques 2D discrétisés. Ce type de problème étant facilement traduisable en algorithme, il est alors opportun d'utiliser l'outil informatique afin d'optimiser la procédure de calcul. Après une présentation des enjeux, ce rapport développera les étapes de résolutions en réalisant une analogie avec la méthode classique.

Le language *C#* étant orienté-objet par nature, il nous a paru tout à fait évident de se tourner vers cette façon de programmer. La Programmation Orientée Objet (POO) permet en effet une abstraction et modularité qui est un réel atout dans ce type de projet. Au cours de ce rapport, nous allons en détailler quelques parties, notamment l'environnement mathématique que nous avons créé, qui pourra être facilement réutilisable dans d'autres projets nécessitant un formalisme matriciel.

En termes de pratiques, le choix a été fait de se tourner vers des constructeurs vides et des méthodes WithProperty() retournant l'objet de travail, permettant le chainage. Cette convention, adoptée du language Java, permet d'obtenir un code beaucoup plus agréable à l'usage qu'une liste infinie de paramètres sur un constructeur unique classique.

1 Description du problème et objectifs de programmation

On considère un système bidimensionnel mécanique simple constitué d'un assemblages de barres en caoutchouc. Ce type de systèmes permet d'introduire les rouages principaux de la méthode des éléments finis appliquée aux calculs de résistance mécaniques pour lesquels les matrices de rigidité locales et vecteurs forces locaux peuvent être déterminés sans formulation intégrale.

L'objectif de ce projet est alors de constuire un algorithme de résolution du problème aux paramètres modifiables afin de s'adapter à différentes situations envisageables dans le cadre des résolutions de l'UV MN41.

1.1 Élément de barre

Comme son nom l'indique, les systèmes bidimensionnels présentent des variables d'évolution dans deux direction. On considérera alors que la barre b_i du système global composé de i barres (présenté dans les sous-sections suivantes) est définie le long des axes $\overrightarrow{x}$ et $\overrightarrow{y}$.

Les paramètres caractérisant le matériau (caoutchouc) et la géométrie de la barre sont les suivants :

- E est le module d'Young de la barre b_i ;
- A est la section de la barre b_i ;
- L_i est la longueur de la barre b_i ;
- Un chargement F_i aux extremités de la barre b_i.

1.2 Matrice de rigidité

Afin de modéliser le système, il est nécessaire de définir les lois de comportement des éléments constitutifs. Cette description permettra alors de discretiser le problème en matrices élémentaires. Celles-ci découlent du tableau de connectivité qui décrit la localisation des éléments par rapport aux noeuds.

La loi de comportement d'une barre compris entre deux noeuds N_1 et N_2 est définie par :

$$\frac{EA}{L}\begin{pmatrix} cos^2(\theta) & cos(\theta)sin(\theta) & -cos^2(\theta) & -cos(\theta)sin(\theta) \\ cos(\theta)sin(\theta) & sin^2(\theta) & -cos(\theta)sin(\theta) & -sin^2(\theta) \\ -cos^2(\theta) & -cos(\theta)sin(\theta) & cos^2(\theta) & cos(\theta)sin(\theta) \\ -cos(\theta)sin(\theta) & -sin^2(\theta) & cos(\theta)sin(\theta) & sin^2(\theta) \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{pmatrix}$$

Avec θ l'inclinaison de la barre par rapport à l'horizontale.

R La matrice globale étant constituée d'un assemblage de matrices élémentaires symétriques, il est possible de résoudre le système sans conditions aux limites.

1.3 Systèmes étudiés

Nous souhaitons, à travers ce rapport, présenter la résolution de d'un système dans deux configurations différentes. Afin de mieux cerner les concepts mathématiques et méthodologiques de résolution et de programmation, on propose de traiter ces cas en parallèle d'un cas plus simple qui sera détaillé tout au long du rapport.

On considère l'assemblage de trois barres suivant. Le système possède une articulation en A et un appui plan en B. Les barres ont pour module d'Young E, section A et longueur L_i. La structure subit une force F ($F = 100$ N) verticale descendante appliquée au point C noté le noeud 3.

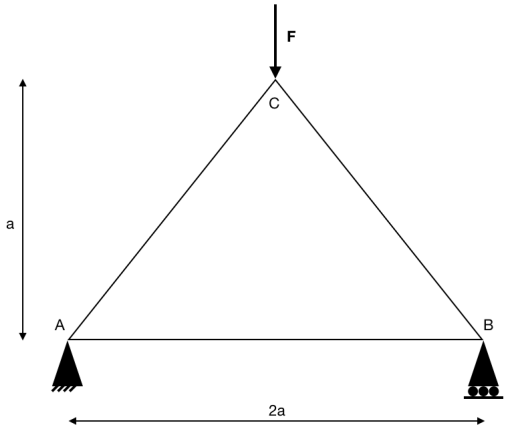

Figure 1 - Représentation schématique du cas 0

On définit à présent le système global dont la résolution présente la finalité de ce rapport. Il est constitué d'un assemblage de barres aux propriétés définies précédemment selon un ordre défini dans le schéma subséquent en prenant $a = 1$.

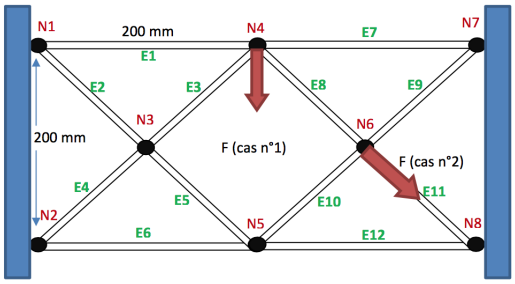

Figure 2 - Représentation schématique du cas 1 et 2

Nous étudierons deux configurations :

- Cas 1 : $F = 5N$ et $\theta = 270^{\circ}$ au noeud $N4$
- Cas 2 : $F = 10N$ et $\theta = 315^{\circ}$ au noeud $N6$

Ces situations ainsi que les conditions aux limites peuvent être modifiables facilement grâce aux documents textes détaillés dans le paragraphe 2.

1.4 Structure du programme

Nous utilisons la programmation orientée objet permettant de créer différentes classes dont chacune a sa propre fonction. Afin de mieux cerner la structure et le découpage de notre programme, on propose de le présenter sous la forme du schéma suivant :

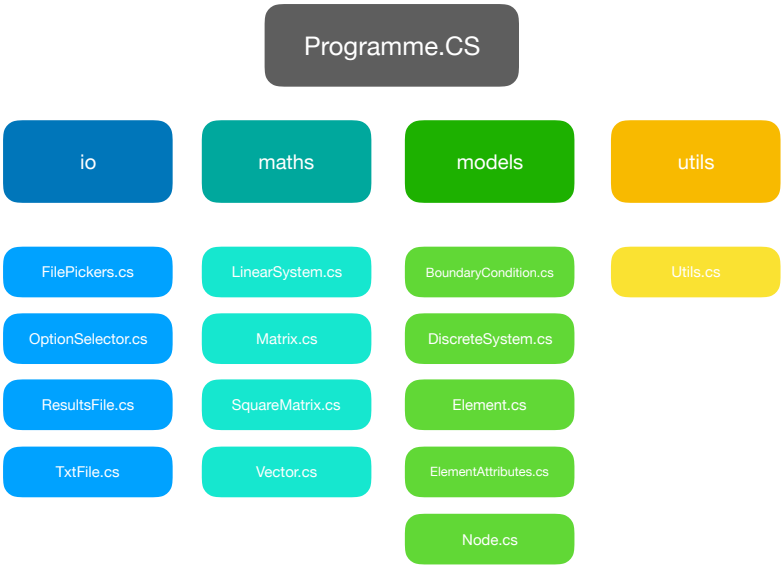

Figure 3 - Structure modulaire du programme

Nous ne développerons pas la catégorie "utils" qui constitue des outils de simplifications pour l'évolution du programme et non des fonctions indispensables à la résolution.

2 IO : fichier de données et informations fournies par l'utilisateur

La fonction de la catégorie *io* n'est pas que de lire le fichier. Elle possède aussi entre autre la classe *FilePicker* qui permet de choisir le cas d'étude et *ResultsFiles* pour l'écriture des résultats dans un document texte.

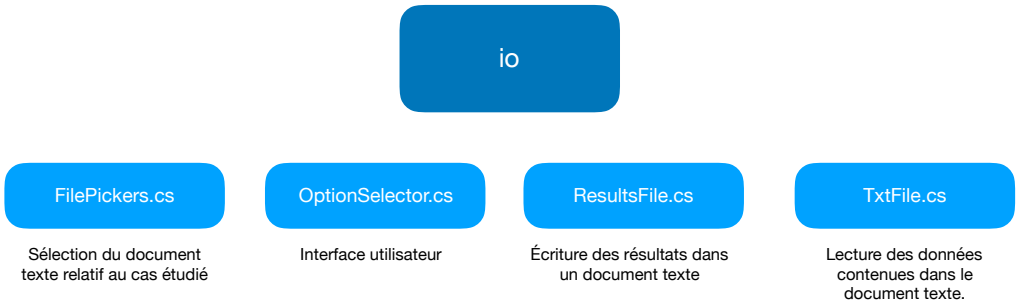

Figure 4 - Présentation de la catégorie "io"

2.1 Structure du fichier et lecture

Afin de résoudre le problème, il est demandé à l'utilisateur de fournir les données sous la forme d'un fichier texte : "$cas_i.txt$" afin que l'algorithme puisse effectuer la démarche de résolution. Afin d'optimiser le programme, on demande à l'utilisateur de placer celui-ci dans le dossier "$Data$" situé dans le fichier racine du projet. Ce document est structuré et doit respecter la trame détaillé ci-dessous :

Nombre de noeuds :

La ligne 2 indique le nombre de noeuds N. Cette information donne la taille de la matrice de rigidité globale et donc du système à résoudre.

Coordonnées des noeuds :

Les lignes 5 à $5+N-1$ informent les coordonnées sous la forme d'un couple (x,y) de chaque noeuds du système. Cette information est essentielle dans la détermination des longueurs L_i des éléments.

Nombre d'éléments :

La ligne $N+7$ renseigne le nombre d'éléments n et par extension le nombre de matrices élémentaires que l'algorithme devra par la suite assembler.

Localisation des éléments :

Les lignes $N+10$ à $N+10+n$ précisent la localisation des éléments ce qui permettra par suite l'assemblage de la matrice globale mais aussi de calculer la longueur des barres par recoupement avec les lignes 5 à $5+N-1$.

Norme et orientation des forces appliquées :

Les lignes $N+n+13$ à $2N+n+13$ renseignent sur la norme et l'orientation des forces appliquées sur les noeuds. Ces informations seront utiles à la construction du second membre.

Conditions aux limites :

Les lignes $2N+n+16$ à $3N+n+13$ correspondent aux conditions aux limites et permettront de réduire la matrice globale afin d'optimiser le processus de résolution.

Dans le cas 2, les données sont présentées de la manière suivante :

```
Nombre de noeuds:
8

Cordonnees de noeuds en metre (x, y):
0 0,2
0 0
0,1 0,1
0,2 0,2
0,2 0
0,3 0,1
0,4 0,2
0,4 0

Nombre d'elements:
12

Connexion d'elements (No. noeud 1, No. noeud 2):
1 4
1 3
3 4
2 3
3 5
2 5
4 7
4 6
6 7
5 6
6 8
5 8

Force sur noeuds (Force, Angle d'orientation en degree):
0 0
0 0
0 0
0 0
0 0
10 315
0 0
0 0

Condition limite sur noeuds (X (1=oui,0=non), contrainte x, y (1=oui,0=non), contrainte y)
1 0 1 0
1 0 1 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
1 0 1 0
1 0 1 0
```

Figure 4 - Structure du document "$cas2.txt$"

La lecture du fichier selon la méthode vue en TP permet alors de mettre à disposition de l'algorithme l'ensemble des données nécessaire à la constitution des matrices élémentaires, de la matrice globale réduite et à la résolution.

2.2 Interface utilisateur

Afin de rendre le programme interactif et facile d'utilisation, on propose d'instaurer une interface utilisateur qui permet de jouir des fonctionnalités programmées.

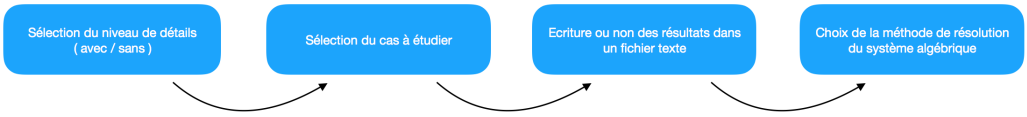

Figure 5 - Structure de l'interface utilisateur

2.3 Détection d'éventuelles erreurs

Une erreur dans le document étant possible, il est nécessaire d'instaurer une procédure permettant de gérer une telle inconvenue. D'une manière générale, l'interface utilisateur empêche de débuter les calculs sur un document inexistant (seuls les documents présents sont affichés dans le menu).

Afin de valider le fichier, on appelle la méthode *validateFile()* de la classe *TxtFile*, qui parcourt le fichier choisi en relevant les informations, et rejette une *Exception*, composant très classique des langages orientés objet, lorsqu'on rencontre un problème. En appelant la validation du fichier au sein d'un bloc *try/catch*, on peut ainsi procéder à la récupération des données, en se protégeant des éventuels problèmes sur le fichier, et, le cas échéant, en informant l'utilisateur de la nature du problème pour qu'il puisse le corriger aisément.

Code 1

```
file.ReadLine();
this.NodesCount = Convert.ToInt32(file.ReadLine());
if (!(this.NodesCount > 0))
{throw new Exception("Nodes count has
to be bigger than 0");}
file.ReadLine();
this.Nodes = new List<Node>();
for (var i = 0; i < NodesCount; i++)
{
    double[] pos = Utils.LineToDoubles(file.ReadLine());
    if (pos.Length != 2){
    throw new Exception("Wrong number of coordinates");}
    this.Nodes.Add(new     Node().WithId(i).WithPosX(pos[0]).
    WithPosY(pos[1]));
}
```

Par exemple, on reconnait dans le code ci-contre la validation du nombre de noeuds, puis des coordonnées de chacun d'entre-eux

2.4 Écriture des résultats dans un fichier

Enfin, une dernière étape consiste à écrire les résultats dans un document texte situé dans le dossier "Results" à la racine du projet. Celui-ci contient :

- la matrice assemblée ;
- le second membre ;
- le système global simplifié (après prise en compte des conditions limites) ;
- les résultats.

3 *Maths* : définition d'un nouvel environnement mathématique

Afin d'optimiser le processus de résolution, on propose de définir un environnement mathématique de résolution en définissant et redéfinissant certaines propriétés traditionnelles utilisées par le logiciel afin de s'adapter aux conditions de résolution du problème.

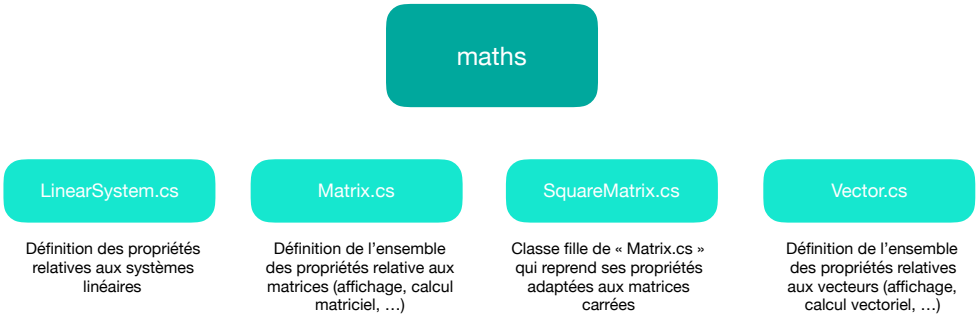

Figure 6 - Présentation de la catégorie "maths"

3.1 Classe *Matrix*

Cette classe sert avant tout à concevoir un environnement mathématique propice à la résolution du problème tels que les opérateurs d'additivité, soustractivité matriciel et vectoriel mais aussi le déterminant et des fonctions d'affichage des matrices. Nous proposons de détailler la fonction d'affichage utile dans le cadre du programme.

A noter également l'utilisation de la surcharge de paramètres, fonctionnalité commune des languages avancés, de nombreuses fois dans le code source. Que l'on retrouve ici pour la méthode *Display(string name)*, également rapidement utilisable sans demander l'affichage d'un nom, grâce à la surcharge *Display()*.

Code 2

```
public void Display(string name)
{
            int i, j;

            string spacer = "   ";
            if (name != "")
            {
                for (i=0; i <  name.Length; i++)
                {
                    spacer += " ";
                }
            }

            Console.WriteLine();
            for (i = 0; i < this.VSize; i++)
            {
                if (name !=  "") {
                    if (i == Math.Floor(Convert.ToDouble
                    (this.VSize / 2)))
                    {

                        Console.Write(" {0} =", name);
                    }
                    else
                    {
                        Console.Write(spacer);
                    }
                }
                Console.Write(" |");
                for (j = 0; j < this.HSize; j++)
                {
                    Console.Write("{0,8:f1} ",
                    this.GetValue(i, j));
                }
                Console.Write("| \n");
            }
            Console.WriteLine();
}
```

3.2 Classe *Vector*

Cette classe pose les bases du calcul vectoriel classique. La fonction *WithPolarValues* convertit un vecteur à coordonnées polaires en vecteur à coordonnées cartésiennes. Rappelons que l'on travaille par rapport à l'origine du repère.

Code 3

```
public Vector WithPolarValues(double r, double angle)
        {
            base.WithValues(new double[2] { r * Math.Cos
            (angle / 360 * 2 * Math.PI), r * Math.Sin
            (angle / 360 * 2 * Math.PI) });
            this.Size = this.VSize;
            this.Display();
            return this;
        }
```

3.3 Classe *SquareMatrix*

Cette classe est une extension de la classe *Matrix* et définit un environnement mathématique adéquat aux matrices carrées. Celle-ci reprend les propriétés existantes en justifiant entre autre que la taille verticale est égale à la taille horizontale.

4 Models : construction du model et résolution

Dans cette partie, nous utilisons les données fournies par l'utilisateur afin de construire le modèle mathématique et le résoudre.

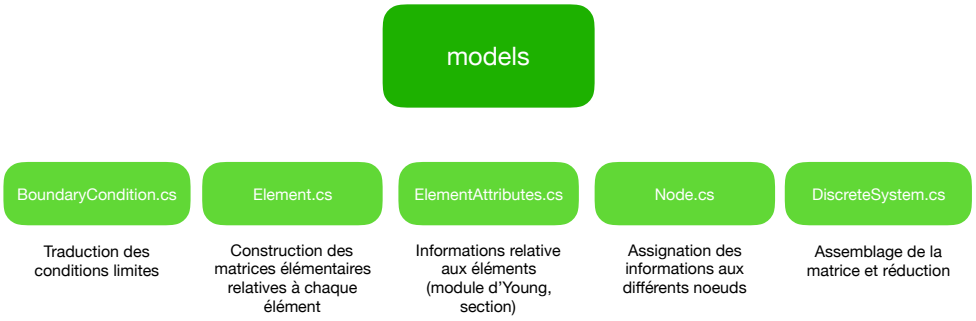

Figure 7 - Présentation de la catégorie "models"

4.1 Classe *BoundaryCondition*

Dans cette classe sont renseignées les conditions aux limites pour chaque noeud.

Code 4

```
public class BoundaryCondition
        {
                public BoundaryCondition ()
                {
                        this.HasOnX = false;
                        this.HasOnY = false;
                }
                public BoundaryCondition WithX(double x)
                {
                        this.HasOnX = true;
                        this.XValue = x;
                        return this;
                }
                public BoundaryCondition WithY(double y)
                {
                        this.HasOnY = true;
                        this.YValue = y;
                        return this;
                }
        }
```

Par défaut, aucune condition aux limites n'est appliquée au noeud. Si la référence du fichier indique qu'une condition est appliquée, alors cette classe applique sur X ou sur Y la condition indiquée.

4.2 Classe *Node*

Cette classe renseigne :
- L'identifiant du noeud $(i = 1,2,...,n)$;
- La position du noeud i (coordonnées en x et en y) ;
- Les conditions aux limites (noeuds fixes, noeuds mobiles) ;
- Les éventuelles forces appliquées et réactions d'appuis.

Code 5

```
public Node ()
{
        this.BoundaryCondition = new BoundaryCondition();
        this.Force = new Vector().WithZeroes(2);
}
public Node WithId(int id) {
        this.Id = id;
        return this;
}
public Node WithPosX(double x) {
        this.PosX = x;
        return this;
}
public Node WithPosY(double y) {
        this.PosY = y;
        return this;
}
public Node WithForce(Vector f){
        this.Force = f;
        return this;
}
public Node WithBoundaryCondition(BoundaryCondition bc) {
        this.BoundaryCondition = bc;
        return this;
}
```

4.3 Classe *ElementAttributes*

Dans cette classe, on assigne les informations relatives à l'élément :
- La section ;
- Le module d'Young.

R Cette classe centralise donc les informations statiques.

4.4 Classe *Element*

En utilisant les informations suivantes :
- La localisation (entre deux noeuds) ;
- La longueur ;
- Le chargement ;

— Les propriétés physiques.

Il est alors possible de construire la matrice élémentaire de l'élément. Cette matrice, de dimension $(4,4)$ respecte la loi de comportement énoncée en 1.2.

Reprenons l'exemple du cas simple. Il est facile de déterminer les matrices élémentaires de trois barres.

Barre 1 :

Les propriété physiques sont : E, A, $L = 2$.
Remarques mathématiques : $cos(\theta) = 1$ et $sin(\theta) = 0$.

Ainsi :

$$\frac{EA}{2}\begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \end{pmatrix}$$

Barre 2 :

Les propriété physiques sont : E, A, $L = \sqrt{2}$.
Remarques mathématiques : $cos(\theta) = -\sqrt{2}/2$ et $sin(\theta) = \sqrt{2}/2$.

Ainsi :

$$\frac{EA}{\sqrt{2}}\begin{pmatrix} 1/2 & -1/2 & -1/2 & 1/2 \\ -1/2 & 1/2 & 1/2 & -1/2 \\ -1/2 & 1/2 & 1/2 & -1/2 \\ 1/2 & -1/2 & -1/2 & 1/2 \end{pmatrix}\begin{pmatrix} u_2 \\ v_2 \\ u_3 \\ v_3 \end{pmatrix}$$

Barre 3 :

Les propriété physiques sont : E, A, $L = \sqrt{2}$.
Remarques mathématiques : $cos(\theta) = \sqrt{2}/2$ et $sin(\theta) = \sqrt{2}/2$.

Ainsi :

$$\frac{EA}{\sqrt{2}}\begin{pmatrix} 1/2 & 1/2 & -1/2 & -1/2 \\ 1/2 & 1/2 & -1/2 & -1/2 \\ -1/2 & -1/2 & 1/2 & 1/2 \\ -1/2 & -1/2 & 1/2 & 1/2 \end{pmatrix}\begin{pmatrix} u_2 \\ v_2 \\ u_3 \\ v_3 \end{pmatrix}$$

Le programme utilise ces propriétés pour construire une matrice symétrique constituées de coefficients flottants. Afin d'optimiser les calculs, les propriétés de symétrie sont utilisées et seuls 3 coefficients sont calculés puis transposés avec le signe adéquat sur les colonnes et lignes.

Code 6

```
double xLength = this.SecondNode.PosX - this.FirstNode.PosX;
double yLength = this.SecondNode.PosY - this.FirstNode.PosY;

this.Length = Math.Sqrt(xLength * xLength + yLength * yLength);
this.Angle = Math.Atan2(yLength, xLength);

SquareMatrix matrix = new SquareMatrix().
WithValues(new double[4, 4]);
SquareMatrix subMatrix = this.GetSubMatrix();
double coeff = this.GetCoeff();

for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
                if (i < 2 && j < 2)
                {
                        matrix.SetValue (i, j, coeff *
                        subMatrix.GetValue (i, j));
                }
                else if (i >= 2 && j >= 2)
                {
                        matrix.SetValue (i, j, coeff *
                        subMatrix.GetValue (i - 2, j - 2));
                }
                else if (i < 2)
                {
                    matrix.SetValue(i, j, -coeff *
                    subMatrix.GetValue(i, j - 2));
                }
                else if (j< 2)
                {
                        matrix.SetValue(i, j, -coeff *
                        subMatrix.GetValue(i - 2, j));
                }
        }
}

this.ElementaryMatrix = matrix;
```

R Une boucle permet alors de calculer la matrice élémentaire de chaque élément et de procéder à l'assemblage de la matrice de rigidité globale.

4.5 Assemblage de la matrice globale

Le module *DiscreteSystem* permet d'assembler les différentes matrices élémentaires en fonction de la localisation des éléments. Rappelons que le système global est de la forme :

$$\mathbb{K}\mathbb{U} = \mathbb{F}$$

Avec $\mathbb{K}$ la matrice globale de rigidité, $\mathbb{U}$ le vecteur déplacements des noeuds et $\mathbb{F}$ le vecteur des forces extérieures.

R Il s'agit d'une application du théorème de superposition.

Code 7

```
int size = this.NodesCount * 2;
this.AssembledMatrix = new SquareMatrix().WithZeroes(size);
foreach (Element element in this.Elements) {
int posUi = 2 * element.FirstNode.Id;
int posUj = 2 * element.SecondNode.Id;
for (int i  = 0; i < 4; i++)
{
    for (int j = 0; j < 4; j++)
{
if (i < 2 && j < 2)
    {
    this.AssembledMatrix.AddToValue(posUi + i, posUi + j,
     element.ElementaryMatrix.GetValue(i, j));
    }
else if (i < 2 && j >= 2)
    {
    this.AssembledMatrix.AddToValue(posUi + i, posUj + j - 2,
     element.ElementaryMatrix.GetValue(i, j));
    }
else if (i >= 2 && j < 2)
    {
    this.AssembledMatrix.AddToValue(posUj + i-2, posUi + j,
     element.ElementaryMatrix.GetValue(i, j));
    }
else if (i >= 2 && j >= 2)
    {
    this.AssembledMatrix.AddToValue(posUj + i - 2, posUj + j - 2,
     element.ElementaryMatrix.GetValue(i, j));
    }
```

Appliquons le théorème de superposition au système simple dont nous avons déjà calculé les matrices élémentaires. On pose $k_1 = EA/2$, $k_2 = EA/2\sqrt{2}$. On a alors :

$$\begin{pmatrix} k_1+k_2 & k_2 & -k_1 & 0 & -k_2 & -k_2 \\ k_2 & k_2 & 0 & 0 & -k_2 & -k_2 \\ -k_1 & 0 & k_1+k_2 & -k_2 & -k_2 & k_2 \\ 0 & 0 & -k_2 & k_2 & -k_2 & -k_2 \\ -k_2 & -k_2 & -k_2 & k_2 & 2k_2 & -k_2+k_2 \\ -k_2 & -k_2 & k_2 & -k_2 & -k_2+k_2 & k_2+k_2 \end{pmatrix} \begin{pmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{pmatrix}$$

Les applications numériques donnent :

$$k_1 = 88{,}4 \qquad k_2 = 62{,}5$$

Soit la matrice de rigidité :

$$\begin{pmatrix} 150{,}9 & 62{,}5 & -88{,}4 & 0 & -62{,}5 & -62{,}5 \\ 62{,}5 & 62{,}5 & 0 & 0 & -62{,}5 & -62{,}5 \\ -88{,}4 & 0 & 150{,}9 & -62{,}5 & -62{,}5 & 62{,}5 \\ 0 & 0 & -62{,}5 & 62{,}5 & -62{,}5 & -62{,}5 \\ -62{,}5 & -62{,}5 & -62{,}5 & 62{,}5 & 125 & 0 \\ -62{,}5 & -62{,}5 & 62{,}5 & -62{,}5 & 0 & 125 \end{pmatrix} \begin{pmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{pmatrix}$$

Le calcul via l'algorithme aboutit à :

```
| 150,835   62,478  -88,357    0,000  -62,478  -62,478 |
|  62,478   62,478    0,000    0,000  -62,478  -62,478 |
| -88,357    0,000  150,835  -62,478  -62,478   62,478 |
|   0,000    0,000  -62,478   62,478   62,478  -62,478 |
| -62,478  -62,478  -62,478   62,478  124,956    0,000 |
| -62,478  -62,478   62,478  -62,478    0,000  124,956 |
```

Figure 8 - Matrice de rigidité (résolution informatique)

4.6 Vecteur forces extérieures (second membre)

Dans le cas de l'exemple traité en parallèle, le vecteur second membre s'exprime alors :

$$F_{ext} = \begin{pmatrix} X_1 \\ Y_1 \\ 0 \\ Y_2 \\ 0 \\ -100 \end{pmatrix}$$

Le code associé permet de retrouver ce résultat. En utilisant les informations relatives aux noeuds, il construit le vecteur second membre.

Code 8

```
int size = this.NodesCount * 2;
this.SecondMember = new Vector().WithZeroes(size);
this.IsUnknown = new bool[size];
this.BCValues = new double[size];
for (int i = 0; i < this.NodesCount; i++)
    {
    this.IsUnknown[i] = true;
    }
foreach (Element element in this.Elements)
    {
    this.IsUnknown[2 * element.FirstNode.Id] =
        !element.FirstNode.BoundaryCondition.HasOnX;
    this.IsUnknown[2 * element.FirstNode.Id + 1] =
        !element.FirstNode.BoundaryCondition.HasOnY;
    this.IsUnknown[2 * element.SecondNode.Id] =
        !element.SecondNode.BoundaryCondition.HasOnX;
    this.IsUnknown[2 * element.SecondNode.Id + 1] =
        !element.SecondNode.BoundaryCondition.HasOnY;

this.BCValues[2 * element.FirstNode.Id] =
    element.FirstNode.BoundaryCondition.HasOnX ?
    element.FirstNode.BoundaryCondition.XValue : 0;
this.BCValues[2 * element.FirstNode.Id + 1] =
    element.FirstNode.BoundaryCondition.HasOnY ?
    element.FirstNode.BoundaryCondition.YValue : 0;
this.BCValues[2 * element.SecondNode.Id] =
    element.SecondNode.BoundaryCondition.HasOnX ?
    element.SecondNode.BoundaryCondition.XValue : 0;
this.BCValues[2 * element.SecondNode.Id + 1] =
    element.SecondNode.BoundaryCondition.HasOnY ?
    element.SecondNode.BoundaryCondition.YValue : 0;

this.SecondMember.SetValue(2 *
    element.FirstNode.Id,
    element.FirstNode.Force.GetValue(0));
this.SecondMember.SetValue(2 *
    element.FirstNode.Id + 1,
    element.FirstNode.Force.GetValue(1));
this.SecondMember.SetValue(2 *
    element.SecondNode.Id,
    element.SecondNode.Force.GetValue(0));
this.SecondMember.SetValue(2 *
    element.SecondNode.Id + 1,
    element.SecondNode.Force.GetValue(1));
```

Lorsque l'on complète le vecteur second membre, un tableau de la taille du système est déclaré *IsUnknown*. Il s'agit d'une propriété de la classe *DiscreteSystem*. Tous les éléments sont déclarés vrais au début puis la liste d'éléments du système est parcourue et une vérification est opérée sur les conditions limites pour chaque élément. Le tableau *IsUnknown* est mis à jour en fonction.

4.7 Réduction de la matrice

Cette étape utilise la connaissance des conditions aux limites. Si les déplacements sont possibles ou non (position du noeud fixée à un mur par exemple).

Si le déplacement i est imposé non nul, il est alors nécessaire de retrancher la colonne i au second membre.

Reprenons l'exemple simple. On a $u_1 = v_1 = v_2 = 0$ d'où le système réduit :

$$\begin{pmatrix} 150,9 & -62,5 & 62,5 \\ -62,5 & 125 & 0 \\ 62,5 & 0 & 125 \end{pmatrix} \begin{pmatrix} u_2 \\ u_3 \\ v_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -100 \end{pmatrix}$$

Le programme, lui, donne le système réduit suivant :

```
|150,8353 -62,4780  62,4780 |    | u2 |    |  0,0000 |
|-62,4780 124,9561   0,0000 | .  | u3 | =  |  0,0000 |
| 62,4780   0,0000 124,9561 |    | v3 |    |-100,0000 |
```

Figure 9 - Système réduit (résolution informatique)

Le processus de résolution aboutit au système théorique. On peut ainsi valider notre programme et l'appliquer à la résolution des deux situations souhaitées.

Le code associé à cette procédure se trouve ci-dessous. Dans la procédure *SimplifySystem()*, le nombre d'inconnues est compté (le nombre de *true* dans *IsUnknown*) et la matrice assemblée est décomposée en vecteurs colonnes qui sont mis dans une liste. On crée alors deux nouvelles listes vides de type Vector, simplifiedVectors et toBeRemovedVectors. En faisant ensuite varier un indice sur la taille du système, on créé des nouveaux vecteurs, de la taille du nombre d'inconnues, et on lui attribue les valeurs des colonnes correspondantes. Si l'indice de ce vecteur colonne correspond à une inconnue (test via le tableau *IsUnknown*), on ajoute à la liste *simplifiedVectors*, sinon, on multiplie par la condition limite correspondante et on l'ajoute à la liste toBeRemovedVectors. Le nouveau système simplifié est ensuite créé par les vecteurs colonnes de la liste simplifiedVectors, qui forment une matrice via la méthode *Matrix.WithVectors(Vector[])*, et par le second membre original dont seules les composantes correspondantes aux inconnues ont été gardées.

Code 9

```
private void SimplifySystem()
{ int size = this.NodesCount * 2;
int unknownsCount = size;
for (int i = 0; i < this.IsUnknown.Length; i++)
    { if (!this.IsUnknown[i])
        { unknownsCount--;} }

List<Vector> vectors = new List<Vector>();
vectors.AddRange(this.AssembledMatrix.toVectors());
List<Vector> simplifiedVectors = new List<Vector>();
List<Vector> toBeRemovedVectors = new List<Vector>();

for (int i = 0; i < size; i++)
    {Vector v = new Vector(unknownsCount);
    int count = 0;
    for (int j = 0; j < size; j++)
        {if (this.IsUnknown[j])
            {v.SetValue(count, vectors[i].GetValue(j));
            count++;}}
    if (this.IsUnknown[i])
        {simplifiedVectors.Add(v);}
    else
        {toBeRemovedVectors.Add(v * this.BCValues[i]);}}

SquareMatrix simpleMatrix =
new SquareMatrix().WithVectors(simplifiedVectors.ToArray().
ToArray());

Vector simpleSecondMember = new Vector(unknownsCount);

int sum = 0;
for (int i = 0; i < size; i++)
    {if (this.IsUnknown[i])
        {simpleSecondMember.SetValue(sum,
        this.SecondMember.GetValue(i));
        sum++;}}

    for (int i = 0; i < toBeRemovedVectors.Count; i++)
        {simpleSecondMember = simpleSecondMember
        - toBeRemovedVectors[i];}

    this.SimpleSystem = new LinearSystem(simpleMatrix,
    simpleSecondMember);
    }
```

4.8 Résolution du système matriciel

La résolution des systèmes matriciels reprend les algorithmes étudiés en TP.

Résolution par la méthode de Gauss

Cette méthode consiste à faire une suite d'opérations sur le système d'équations qui, sans changer ses solutions, le transforment en un système triangulaire de résolution triviale.

Résolution par la méthode de LU

La méthode de Lu consiste à factoriser la matrice globale en deux matrices triangulaires : l'une supérieure et l'autre inférieure puis résoudre successivement deux systèmes matriciels triangulaires.

R L'intérêt est notable : on ne touche pas au second membre ce qui est utile pour changer les conditions aux limites par exemple.

Résolution par la méthode de Thomas

Cette méthode consiste à transformer le système $\mathbb{K}\mathbb{U} = \mathbb{F}$ en un système triangulaire supérieur $\mathbb{K}'\mathbb{U} = \mathbb{F}'$ où $\mathbb{K}'$ est une matrice bidiagonale avec une diagonale principale unitaire et une diagonale supérieure définie par une relation de récurrence. Les composantes du second membre $\mathbb{F}'$ sont définis par une autre récurrence.

La méthode de Thomas ne peut pas être utilisée tout le temps. Seules les matrices tri-bandes peuvent jouir d'une telle résolution. La vérification se base alors sur une liste de coordonnées où un zero doit être présent. L'algorithme compare alors coefficient à coefficient si les coordonnées font partie de cette liste. Si un coefficient devant être nul ne l'est pas, alors la résolution par la méthode de Thomas n'est pas possible.

5 Résultats aux études de cas

Dans cette partie nous réaliserons dans un premier temps des comparaisons entre les résultats théoriques et les résultats obtenus à l'ordinateurs afin de valider le modèle (cas 0). Puis, nous lancerons les calculs pour les autres études de cas (cas 1 et cas 2).

5.1 Cas 0

On rappelle dans un premier temps le système théorique réduit aux coefficients arrondis :

$$\begin{pmatrix} 150,9 & -62,5 & 62,5 \\ -62,5 & 125 & 0 \\ 62,5 & 0 & 125 \end{pmatrix} \begin{pmatrix} u_2 \\ u_3 \\ v_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -100 \end{pmatrix}$$

La résolution aboutie à :

$$u2 = 0,5 \qquad u_3 = 0,3 \qquad v_3 = -1$$

Le programme, lui, donne les résultats suivants :

```
| u2 |     |   0,566 |
| u3 |  =  |   0,283 |
| v3 |     |  -1,083 |
```

Figure 10 - Solution du cas 0 (programme)

Les résultats issus du programme sont en adéquation avec le modèle théorique. On peut valider notre algorithme et opérer la résolution des cas 1 et 2.

5.2 Matrice de rigidité

La matrice de rigidité obtenue par le programme pour les deux cas est :

```
|  1508,4   -624,8      0,0      0,0   -624,8    624,8   -883,6      0,0      0,0      0,0      0,0      0,0      0,0      0,0      0,0      0,0 |
|  -624,8    624,8      0,0      0,0    624,8   -624,8      0,0      0,0      0,0      0,0      0,0      0,0      0,0      0,0      0,0      0,0 |
|     0,0      0,0   1508,4    624,8   -624,8   -624,8      0,0      0,0   -883,6      0,0      0,0      0,0      0,0      0,0      0,0      0,0 |
|     0,0      0,0    624,8    624,8   -624,8   -624,8      0,0      0,0      0,0      0,0      0,0      0,0      0,0      0,0      0,0      0,0 |
|  -624,8    624,8   -624,8   -624,8   2499,1      0,0   -624,8   -624,8   -624,8    624,8      0,0      0,0      0,0      0,0      0,0      0,0 |
|   624,8   -624,8   -624,8   -624,8      0,0   2499,1   -624,8   -624,8    624,8   -624,8      0,0      0,0      0,0      0,0      0,0      0,0 |
|  -883,6      0,0      0,0      0,0   -624,8   -624,8   3016,7      0,0      0,0      0,0   -624,8    624,8   -883,6      0,0      0,0      0,0 |
|     0,0      0,0      0,0      0,0   -624,8   -624,8      0,0   1249,6      0,0      0,0    624,8   -624,8      0,0      0,0      0,0      0,0 |
|     0,0      0,0   -883,6      0,0   -624,8    624,8      0,0      0,0   3016,7      0,0   -624,8   -624,8      0,0      0,0   -883,6      0,0 |
|     0,0      0,0      0,0      0,0    624,8   -624,8      0,0      0,0      0,0   1249,6   -624,8   -624,8      0,0      0,0      0,0      0,0 |
|     0,0      0,0      0,0      0,0      0,0      0,0   -624,8    624,8   -624,8   -624,8   2499,1      0,0   -624,8   -624,8   -624,8    624,8 |
|     0,0      0,0      0,0      0,0      0,0      0,0    624,8   -624,8   -624,8   -624,8      0,0   2499,1   -624,8   -624,8    624,8   -624,8 |
|     0,0      0,0      0,0      0,0      0,0      0,0   -883,6      0,0      0,0      0,0   -624,8   -624,8   1508,4    624,8      0,0      0,0 |
|     0,0      0,0      0,0      0,0      0,0      0,0      0,0      0,0      0,0      0,0   -624,8   -624,8    624,8   1508,4   -624,8    624,8 |
|     0,0      0,0      0,0      0,0      0,0      0,0      0,0      0,0   -883,6      0,0   -624,8    624,8      0,0      0,0   -624,8    624,8 |
```

Figure 11 - Matrice globale (informatique)

Une erreur d'assemblage étant possible, on propose de comparer les résultats pratiques aux résultats théoriques. La construction de la matrice globale *à la main* aboutit à la rigidité définie par :

$$K = EA \begin{pmatrix}
\frac{2+\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & 0 & 0 & -\frac{\sqrt{2}}{4} & \frac{\sqrt{2}}{4} & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-\frac{\sqrt{2}}{4} & \frac{\sqrt{2}}{4} & 0 & 0 & \frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{2+\sqrt{2}}{4} & \frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{\sqrt{2}}{4} & \frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-\frac{\sqrt{2}}{4} & \frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & \sqrt{2} & 0 & -\frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & +\frac{\sqrt{2}}{4} & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & 0 & \sqrt{2} & -\frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & +\frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & 0 & 0 & 0 & 0 & 0 & 0 \\
-\frac{1}{2} & 0 & 0 & 0 & -\frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & \frac{2+\sqrt{2}}{2} & 0 & 0 & 0 & -\frac{\sqrt{2}}{4} & \frac{\sqrt{2}}{4} & -\frac{1}{2} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -\frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & 0 & 0 & 0 & 0 & \frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & 0 & 0 & 0 & 0 \\
0 & 0 & -\frac{1}{2} & 0 & -\frac{\sqrt{2}}{4} & \frac{\sqrt{2}}{4} & 0 & 0 & \frac{2+\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & 0 & 0 & -\frac{1}{2} & 0 \\
0 & 0 & 0 & 0 & \frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & 0 & 0 & 0 & 0 & -\frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -\frac{\sqrt{2}}{4} & \frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & \sqrt{2} & 0 & -\frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{4} & \frac{\sqrt{2}}{4} \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & 0 & \sqrt{2} & 0 & -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} \\
0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 & -\frac{\sqrt{2}}{2} & 0 & \frac{1+\sqrt{2}}{2} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} & 0 & -\frac{\sqrt{2}}{4} & \frac{\sqrt{2}}{4} & 0 & 0 & \frac{2+\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\sqrt{2}}{4} & -\frac{\sqrt{2}}{4} & 0 & 0 & -\frac{\sqrt{2}}{4} & \frac{\sqrt{2}}{4}
\end{pmatrix}$$

Figure 12 - Matrice globale (théorique)

Afin d'optimiser la représentation, l'unité utilisée pour les longueur des barres a été le décimètre.

On remarque bien une cohérence mathématique entre les deux matrices de rigidité : la localisation des coefficients nuls et négatifs correspondent et le calcul de certains coefficients mettent en exergue une correspondance.

5.3 Situation 1

Dans ce cas, $F = 5N$ et $\theta = 270^{\circ}$ au noeud $N4$. On obtient les déplacements suivants :

```
| u3 |     |  -0,002 |
| v3 |     |  -0,002 |
| u4 |     |   0,000 |
| v4 |     |  -0,008 |
| u5 |  =  |   0,000 |
| v5 |     |   0,000 |
| u6 |     |   0,002 |
| v6 |     |  -0,002 |
```

Figure 13 - Résultat du cas 1

5.4 Situation 2

Dans ce cas, $F = 10N$ et $\theta = 315^{\circ}$ au noeud $N6$. On obtient les déplacements suivants :

```
| u3 |     |  -0,001 |
| v3 |     |  -0,001 |
| u4 |     |   0,001 |
| v4 |     |  -0,006 |
| u5 |  =  |   0,000 |
| v5 |     |   0,000 |
| u6 |     |   0,005 |
| v6 |     |  -0,005 |
```

Figure 14 - Résultat du cas 2

6 Discussions et conclusion générale

Nous avons démontré qu'il était possible d'implémenter des méthodes algorithmiques de calcul systématique pour le traitement de problèmes simples. Dans le cas des systèmes bidimensionnels 2D, la clef de voute est l'extraction et l'utilisation des données dans la construction du système global.

Il est cependant primordial de pouvoir comparer la méthode implémentée avec un élément théorique pour vérifier la cohérence des résultats. Dans cet exemple, la comparaison avec un système de 3 élements a permis de détecter d'éventuelles erreurs lors de la programmation.

Enfin, cet exercice met en lumière la puissance et l'efficacité des ordinateur dans la résolution de problèmes. Les nouveaux défis des ingénieurs est alors d'implémenter des algorithmes de résolution toujours plus complexes afin de gagner en temps, efficacité et précision dans le traitement de l'information.