Projet MN41 Automne 2020

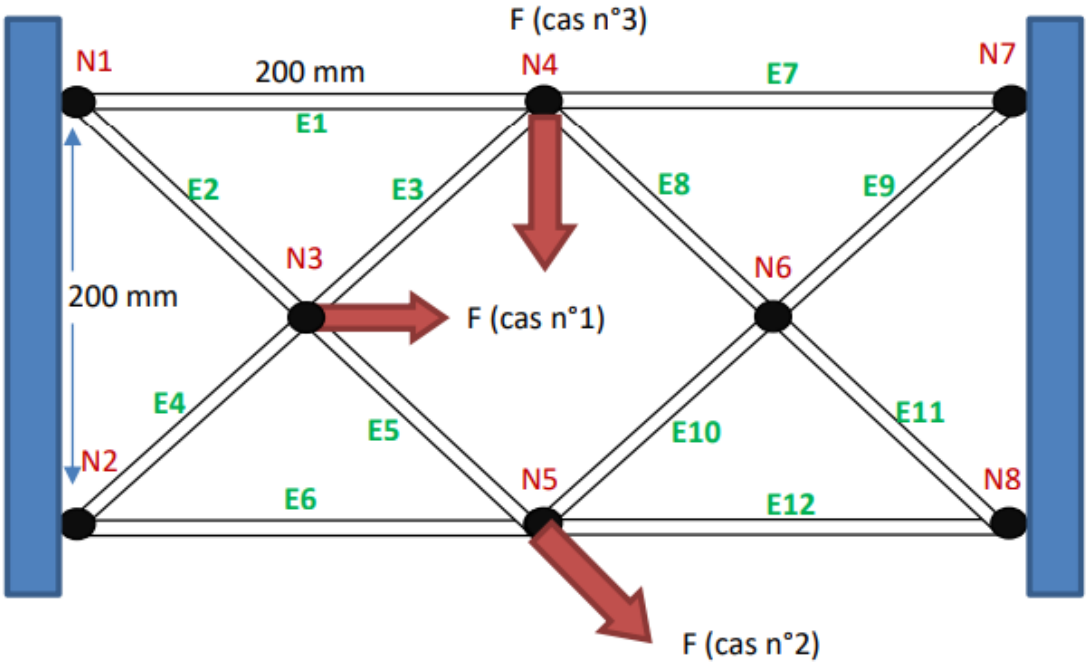
ETUDE DE L'EQUILIBRE D'UN SYSTEME DISCRET BIDIMENSIONNEL

Axel Refalo – Nicolas Meyer

Table des matières

Introduction

Dans ce rapport nous allons vous expliquez la démarche que nous avons effectuée dans un projet dont le but était de mettre au point un programme de calcul de l'équilibre d'un système discret bidimensionnel.



Nous avons du analyser le système ci-dessus afin de créer un programme capable de le résoudre en partant de ses caractéristiques pour les 3 cas différents indiqués sur le shéma

1. Structure du code

La fonction principale de notre programme est

```
static void Main(string[] args)
```

Elle exécute les étapes suivantes, chaque étape est affichée dans le terminal.

1. Lire les données du problème à partir d'un fichier .txt
2. Étude de l'équilibre
   a) Calcul des longueurs des angles et des raideurs de chaque barre
   b) Création des matrices élémentaires et assemblage dans la matrice assemblée
   c) Réduction de la matrice assemblée
   d) Résolution par la méthode choisie et vérification des résultats
      - Méthode de factorisation LU
      - Méthode de Gauss
      - Méthode de Thomas
   e) Calcul des réactions des supports
3. Calcul des tensions de chaque barre
4. Export de la solution dans le fichier .txt

2. Structure des données

La constante correspondante à une instance de problème à résoudre sont :

- Le nombre d'éléments ou de barres

```
int NbElement;
```

- Le nombre de nœuds auquel sont relié les barres

```
int NbNoeud;
```

- Le module de Young des barres exprimé en Pascal

```
double E;
```

- Le diamètre des barres exprimé en mètres

```
double D;
```

- Le tableau de connexion où chaque ligne correspond à un élément (barre) allant du nœud A au nœud B, la colonne 1 contenant l'index du nœud A et la colonne 2, celle du nœud B.

```
int[,] TabConnexion = new int[NbElement, 2];
```

- Le tableau de coordonnée des nœuds où chaque ligne correspond à une coordonnée de nœud (abscisse X en colonne 1 et ordonnée Y en colonne 2).

```
double[,] TabCoordonnee = new double[NbNoeud, 2];
```

- Le tableau des longueurs initiales de chaque barre

```
double[] LongeurB = new double[NbElement];
```

- Le tableau des angles de chaque barre

```
double[] Angle = new double[NbElement];
```

- Le tableau des raideurs de chaque barre

```
double[] TabRaideur = new double[NbElement];
```

- Le tableau des contraintes imposé à chaque nœuds

```
double[,] Contrainte = new double[NbNoeud, 4];
```

Pour chaque nœud, la colonne 1 indique s'il y a contrainte sur X (1 = oui, 0 = non), la colonne 2 contient la valeur de cette contrainte; de la même manière, la colonne 3 indique s'il y a contrainte sur Y et la colonne 4 contient la valeur de cette contrainte.

- Le tableau des contraintes sur chaque déplacement

```
double[,] ContrainteXY = new double[NbNoeud * dim, 2];
```

Pour chaque déplacement, la colonne 1 indique s'il y a contrainte sur déplacement (1 = oui, 0 = non), la colonne 2 contient la valeur de cette contrainte. Ce tableau n'est rien d'autre que le tableau précédent « déplié »

- Le nombre de déplacement non contraint

```
int NbDeplacement;
```

- Le tableau représentant les matrices élémentaires

```
double[,] MElementaire = new double[4, 4]
```

- Le tableau représentant la matrice la matrice assemblée

```
double[,] MAssemble = new double[NbNoeud * dim, NbNoeud * dim];
```

- Le tableau représentant la matrice réduite

```
double[,] MReduite = new double[NbDeplacement, NbDeplacement];
```

- Le tableau des forces appliquées aux nœuds,

```
double[,] Force = new double[NbNoeud, 2];
```

Pour chaque nœud, la colonne 1 contient la valeur de la force, la colonne 2, l'angle de cette force

- Le tableau des composantes de forces appliquées aux nœuds

```
double[] ForceXY = new double[NbNoeud * dim];
```

Pour chaque nœud, la colonne 1 contient la composante X de la force et la colonne 2 contient la composante Y de la force

- Le tableau des composantes de forces appliquées aux nœuds non contraint

```
double[] ForceXYRed = new double[NbDeplacement];
```

Pour chaque nœud non contraint, la colonne 1 contient la composante X de la force et la colonne 2 contient la composante Y de la force

- La variable définissant la méthode de Résolution contient la méthode de résolution choisi (1 : méthode LU, 2 : méthode de Gauss, 3 : méthode de Thomas).

```
int methode;
```

- Le tableau contenant les déplacements des nœuds pas contraints

```
double[] X = new double[NbDeplacement];
```

- Le tableau contenant le déplacement de chaque nœud

```
double[] Deplacement = new double[NbNoeud * dim];
```

- Le tableau répertoriant les tensions de chaque barre

```
double[] Tension = new double[NbElement];
```

- La chaine de caractère définissant le chemin du fichier « Données.txt »

```
string CheminDonnee = @"Donnee.txt";
```

  le fichier « Données.txt » sera placer dans le dossier « Debug » pour pouvoir y accéder peu importe l'emplacement du projet

- La chaine de caractère définissant le chemin du fichier « Résultats.txt »

```
string CheminResultat = @"Resultat.txt";
```

  le fichier « Résultats.txt » sera placer dans le dossier « Debug » pour pouvoir y accéder peu importe l'emplacement du projet

3. Structure des fonctions

- Les fonctions de résolutions par différentes méthodes

```
static double[] MethodeLU(double[,] A, double[] B)
static double[] MethodeGauss(double[,] A, double[] B)
static double[] MethodeThomas(double[,] A, double[] B)
```

- Prennent comme paramètre d'entrée la matrice A et le vecteur B et qui retourne X tel que $A \times B = X$
- Les fonctions d'affichage pour les matrices, les vecteurs et de matrices avec vecteur

```
static void AffichageM(double[,] M, bool Couleur)
static void AffichageV(double[] V)
static void AffichageMV(double[,] M, double[] V, bool Couleur)
```

- Les fonctions permettant de faire des produits de deux matrices et le produit d'une matrice par un vecteur

```
static double[,] ProduitM(double[,] M1, double[,] M2)
static double[] ProduitV(double[,] M, double[] V)
```

- Les fonctions qui permettent de vérifier que deux matrices ou deux vecteurs sont égaux

```
static bool VerificationM(double[,] M1, double[,] M2)
static bool VerificationV(double[] V1, double[] V2)
```

- La fonction qui permet de vérifier qu'une matrice est tridiagonale

```
static bool VerifMTridiagonale(double[,] M
```

4. Lecture des données à partir d'un fichier .txt

Pour lire les données du problème on utilise le fichier « donnee.txt ». La structure du fichier doit rester constante pour le code que nous avons écrit retrouve l'emplacement des données. Pour pouvoir calculer les différents cas il faut changer les angles et les forces dans ce fichier.

Les données sont lues depuis le fichier d'entrée ligne à ligne avec la fonction ci-dessous. Cette fonction est aussi utilisée pour sauter certaines lignes du fichier

```
string ligne = Donnee.ReadLine();
```

puis la chaîne lue est éclatée en ses sous chaînes séparées par des espaces avec la fonction

```
string[] souschaines = texte.Split(' ');
```

les valeurs numériques sont obtenues en convertissant chaque chaîne en un entier (Int32) ou un flottant (Double)

Voici par exemple le code pour récupérer les coordonnes des nœuds :

```
for (int i = 0; i < NbNoeud; i++)
{
    string ligne = Donnee.ReadLine();
    string[] souschaines = ligne.Split(' ');
    for (int j = 0; j < dim; j++)
    {
        TabCoordonnee[i, j] = Convert.ToDouble(souschaines[j + 1]);
    }
}
```

Afin de vérifier que les données lues sont exactes, nous affichons celles-ci avant de commencer les calculs.

```
...Recupération des données...
 0,015
 Nombre d'elements :   12,000
 Module de Young : 1000000,000
 Diamètre :    0,015
 Nombre de noeud :        8
 Tableau de coordonnées :
    0,000     0,200
    0,000     0,000
    0,100     0,100
    0,200     0,200
    0,200     0,000
    0,300     0,100
    0,400     0,200
    0,400     0,000

 Tableau de contrainte :
    1,000     0,000     1,000     0,000
    1,000     0,000     1,000     0,000
    0,000     0,000     0,000     0,000
    0,000     0,000     0,000     0,000
    0,000     0,000     0,000     0,000
    0,000     0,000     0,000     0,000
    1,000     0,000     1,000     0,000
    1,000     0,000     1,000     0,000

 Tableau 'Force' :
    0,000     0,000
    0,000     0,000
    5,000     0,000
    0,000     0,000
    0,000     0,000
    0,000     0,000
    0,000     0,000
    0,000     0,000
```

5. Étude de l'équilibre

a. Calcul des longueurs de angles et des raideurs pour chaque barre

Pour calculer l'angle et la raideur de chaque barre, on considère la barre comme un vecteur pour lequel on veut connaître les coordonnées sur X et Y. Pour calculer X et Y on accède aux nœuds auxquels la barre est reliée grâce à `TabConnexion`

```
int Ni = TabConnexion[i, 0]; // Noeud d'une extrémité de la barre
int Nj = TabConnexion[i, 1]; // Noeud de l'autre extrémité
```

Puis on accède aux coordonnées de ces nœuds avec `TabCoordonnee` et on fait la différence des coordonnées sur x et des coordonnées sur y

```
double x = TabCoordonnee[Nj - 1, 0] - TabCoordonnee[Ni - 1, 0];
double y = TabCoordonnee[Nj - 1, 1] - TabCoordonnee[Ni - 1, 1];
```

Pour calculer les Longueurs qui seront plus tard utile pour le calcul des raideurs on utilise la formule : $L = \sqrt{x^2 + y^2}$ et on place les résultats dans un tableau

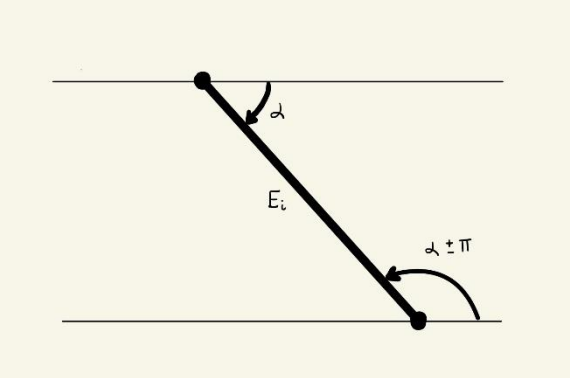
```
LongeurB[i] = Math.Sqrt(Math.Pow(x, 2.0) + Math.Pow(y, 2));
```

Pour calculer les angles on utilise la formule : $\alpha = Arctan\left(\frac{y}{x}\right)$ et on place les résultats dans le tableau `Angle`.

```
Angle[i] = Math.Atan(y / x);
```

Dans le cas général, pour éviter une division par 0, si la barre est verticale (ce qui ne se produit pas dans notre structure) on donne $\alpha = \frac{\pi}{2}$ si $x = 0$

Remarque : Il est possible de prendre 2 angles pour chaque barre …



… mais l'angle choisi n'exerce aucune influence sur les résultats car, plus tard, on prendra

$cos^2(\alpha) = cos^2(\alpha \pm \pi)$

$sin^2(\alpha) = sin^2(\alpha \pm \pi)$

$\sin(\alpha)\cos(\alpha) = \sin(\alpha \pm \pi)\cos(\alpha \pm \pi)$

On calcule enfin la raideur de chaque barre grâce à la formule ci-dessous

$$K_i = \frac{AE}{L_i} = \frac{\pi R^2 E}{L_i} = \frac{\pi D^2 E}{4L_i}$$

Avec :

- K_i : Raideur de la barre i

- E : module de Young
- A : Section
- D : Diamètre
- L_i : Longueur de la barre i

Les valeurs obtenues sont placées dans un tableau

```
TabRaideur[i] = (Math.PI * E * Math.Pow(D, 2.0)) / (4 *LongeurB[i]);
```

Une fois les longueurs, les angles et les raideurs de la barre calculés on les affiche pour chaque élément

```
...Calcul des angles et des raideurs de chaque barre...

 Element  1 :
 Raideur = 883.573
 Angle = 0.000

 Element  2 :
 Raideur = 1249.561
 Angle = -45.000

etc...
```

b. Création des matrices élémentaires et assemblage

Pour chaque élément on définit une matrice élémentaire :

```
double[,] MElementaire = new double[4, 4] {{  C2,  CS, -C2, -CS},
                                           {  CS,  S2, -CS, -S2},
                                           { -C2, -CS,  C2,  CS},

        { -CS, -S2,  CS,  S2}};
```

avec

```
double C2 = Math.Pow(Math.Cos(Angle[i]), 2); // cos^2(angle)
double S2 = Math.Pow(Math.Sin(Angle[i]), 2); // sin^2(angle)
double CS = Math.Cos(Angle[i]) * Math.Sin(Angle[i]); // cos*sin
```

et on multiplie tous ses coefficients par la raideur K :

```
MElementaire[l, c] *= TabRaideur[i];
```

Dans le but de minimiser le nombre d'opérations à réaliser sur la matrice assemblée (dans le cas de modèles avec beaucoup de barres), on détermine à l'avance un tableau de 4 éléments qui contient les indices de lignes de matrices assemblées qui doivent être modifies. La disposition des coefficients dans la matrice assemblée étant symétrique, ces indices sont aussi ceux des colonnes qui doivent être modifiées :

```
int[] Tab = new int[4] {(Ni-1)*2, (Ni-1)*2+1, (Nj-1)*2, (Nj-1)*2+1};
```

Pour calculer les indices il faut récupérer les nœuds auquel est relié la barre de la matrice élémentaire

```
int Ni = TabConnexion[i, 0]; // Noeud d'une extrémité
int Nj = TabConnexion[i, 1]; // Noeud de l'autre extrémité
```

Comme le tableau commence à l'indice zéro on déduit 1 à Ni et Nj, le déplacement V étant une case à droite du déplacement U on ajoute 1 à l'emplacement de V et enfin on multiplie par 2 car la longueur de la matrice est 2 fois plus grande que le nombre de nœuds ce qui nous donne les formules suivantes :

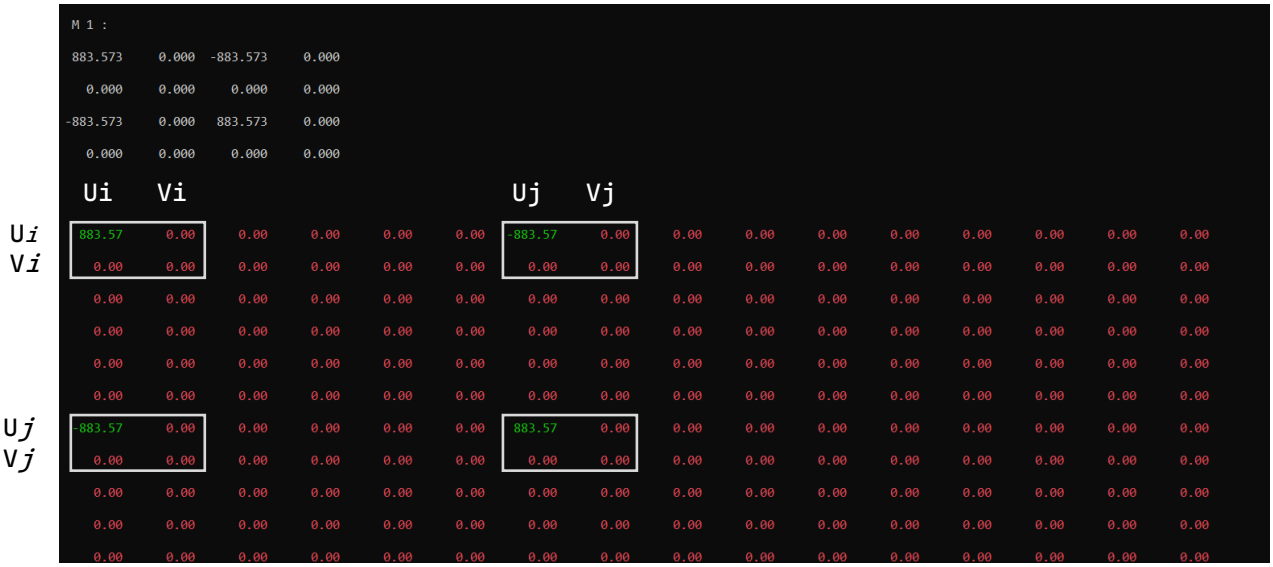
$Indice(U_i) = (N_i - 1) \times 2$
$Indice(V_i) = Emplacement(U_i) + 1 = (N_i - 1) \times 2 + 1$

$Indice(U_j) = (N_j - 1) \times 2$
$Indice(V_j) = Emplacement(U_i) + 1 = (N_j - 1) \times 2 + 1$

Dès lors il suffit de parcourir toutes les combinaisons d'indices issus de ce tableau et d'ajouter les coefficients de la matrice élémentaire à la matrice assemblée :

```
for (int l = 0; l < 4; l++)
{
                    for (int c = 0; c < 4; c++)
            {
                        MAssemble[Tab[l], Tab[c]] += MElementaire[l, c];
            }
}
```

Exemple avec le placement de la matrice élémentaire de E1 (élément 1) :



Ici on a :

$Indice(U_i) = (N_i - 1) \times 2 = (1 - 1) \times 2 = 0$
$Indice(V_i) = Emplacement(U_i) + 1 = 0 + 1 = 1$
$Indice(U_j) = (N_j - 1) \times 2 = (4 - 1) \times 2 = 6$
$Indice(V_j) = Emplacement(U_i) + 1 = 6 + 1 = 7$

A chaque matrice élémentaire qu'on ajoute dans la matrice assemblée on affiche la matrice élémentaire et la matrice assemblée. L'affichage est en couleur pour pouvoir plus simplement vérifier que la matrice est symétrique. Lors de la dernière opération on obtient la matrice assemblée suivante :

```
M12 :
 883.573     0.000  -883.573     0.000
   0.000     0.000     0.000     0.000
-883.573     0.000   883.573     0.000
   0.000     0.000     0.000     0.000


1508.35   -624.78      0.00      0.00   -624.78    624.78   -883.57      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
-624.78    624.78      0.00      0.00    624.78   -624.78      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
   0.00      0.00   1508.35    624.78   -624.78   -624.78      0.00      0.00   -883.57      0.00      0.00      0.00      0.00      0.00      0.00      0.00
   0.00      0.00    624.78    624.78   -624.78   -624.78      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
-624.78    624.78   -624.78   -624.78   2499.12      0.00   -624.78   -624.78   -624.78    624.78      0.00      0.00      0.00      0.00      0.00      0.00
 624.78   -624.78   -624.78   -624.78      0.00   2499.12   -624.78   -624.78    624.78   -624.78      0.00      0.00      0.00      0.00      0.00      0.00
-883.57      0.00      0.00      0.00   -624.78   -624.78   3016.71      0.00      0.00      0.00   -624.78    624.78   -883.57      0.00      0.00      0.00
   0.00      0.00      0.00      0.00   -624.78   -624.78      0.00   1249.56      0.00      0.00    624.78   -624.78      0.00      0.00      0.00      0.00
   0.00      0.00   -883.57      0.00   -624.78    624.78      0.00      0.00   3016.71      0.00   -624.78   -624.78      0.00      0.00   -883.57      0.00
   0.00      0.00      0.00      0.00    624.78   -624.78      0.00      0.00      0.00   1249.56   -624.78   -624.78      0.00      0.00      0.00      0.00
   0.00      0.00      0.00      0.00      0.00      0.00   -624.78    624.78   -624.78   -624.78   2499.12      0.00   -624.78   -624.78   -624.78    624.78
   0.00      0.00      0.00      0.00      0.00      0.00    624.78   -624.78   -624.78   -624.78      0.00   2499.12   -624.78   -624.78    624.78   -624.78
   0.00      0.00      0.00      0.00      0.00      0.00   -883.57      0.00      0.00      0.00   -624.78   -624.78   1508.35    624.78      0.00      0.00
   0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00   -624.78   -624.78    624.78    624.78      0.00      0.00
   0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00   -883.57      0.00   -624.78    624.78      0.00      0.00   1508.35   -624.78
   0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00    624.78   -624.78      0.00      0.00   -624.78    624.78
```

c. Réduction de la matrice assemblée

Pour réduire la matrice, la méthode consiste à barrer les colonnes et lignes correspondantes à un déplacement contraint et créer la matrice élémentaire avec les coefficients non barré. Comme ceci :

```
...Reduction de la matrice assemblée...
 1508.35   -624.78      0.00      0.00   -624.78    624.78   -883.57      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      1.00
 -624.78    624.78      0.00      0.00    624.78   -624.78      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      1.00
    0.00      0.00   1508.35    624.78   -624.78   -624.78      0.00      0.00   -883.57      0.00      0.00      0.00      0.00      0.00      0.00      0.00      1.00
    0.00      0.00    624.78    624.78   -624.78   -624.78      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      1.00
 -624.78    624.78   -624.78   -624.78   2499.12      0.00   -624.78   -624.78   -624.78    624.78      0.00      0.00      0.00      0.00      0.00      0.00      0.00
  624.78   -624.78   -624.78   -624.78      0.00   2499.12   -624.78   -624.78    624.78   -624.78      0.00      0.00      0.00      0.00      0.00      0.00      0.00
 -883.57      0.00      0.00      0.00   -624.78   -624.78   3016.71      0.00      0.00      0.00   -624.78    624.78   -883.57      0.00      0.00      0.00      0.00
    0.00      0.00      0.00      0.00   -624.78   -624.78      0.00   1249.56      0.00      0.00    624.78   -624.78      0.00      0.00      0.00      0.00      0.00
    0.00      0.00   -883.57      0.00   -624.78    624.78      0.00      0.00   3016.71      0.00   -624.78   -624.78      0.00      0.00   -883.57      0.00      0.00
    0.00      0.00      0.00      0.00    624.78   -624.78      0.00      0.00      0.00   1249.56   -624.78   -624.78      0.00      0.00      0.00      0.00      0.00
    0.00      0.00      0.00      0.00      0.00      0.00   -624.78    624.78   -624.78   -624.78   2499.12      0.00   -624.78   -624.78   -624.78    624.78      0.00
    0.00      0.00      0.00      0.00      0.00      0.00    624.78   -624.78   -624.78   -624.78      0.00   2499.12   -624.78   -624.78    624.78   -624.78      0.00
    0.00      0.00      0.00      0.00      0.00      0.00   -883.57      0.00      0.00      0.00   -624.78   -624.78   1508.35    624.78      0.00      0.00      1.00
    0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00   -624.78   -624.78    624.78    624.78      0.00      0.00      1.00
    0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00   -883.57      0.00   -624.78    624.78      0.00      0.00   1508.35   -624.78      1.00
    0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00    624.78   -624.78      0.00      0.00   -624.78    624.78      1.00
```

Les lignes et les colonnes barrés sont en rouge, les coefficients restants sont en blanc. Avec cet affichage on peut facilement identifier la matrice réduite

Dans le programme, l'obtention de la matrice réduite est plus compliquée. Pour réduire la matrice assemblée on se déplace sur ses colonnes. Si la colonne « c » correspond à un déplacement contraint (c'est-à-dire si `ContrainteXY[c, 0] == 1`) on ne la prend pas en compte pour le moment, à l'inverse si la colonne « c » correspond à un déplacement non contraint (c'est-à-dire si `ContrainteXY[c, 0] == 0`) on la prend en compte.

Dès lors, on se déplace sur les lignes de la colonne pas contrainte et avec le même principe que précédemment on prendra en compte seulement les lignes qui correspondent à un déplacement non contraint (c'est-à-dire `ContrainteXY[l, 0] == 0`).

Les coefficients se trouvant à la fois sur une colonne et une ligne non contrainte seront ensuite recopiés dans la matrice réduite

```
int colonneMReduite = 0;
for (int c = 0; c < NbNoeud * dim; c++)
{
  if (ContrainteXY[c, 0] == 0) // le Noeud 'c' n'est pas contraint
  {
     int ligneMReduite = 0;
     // Recopie les colonnes dans la matrice reduite
     for (int l = 0; l < NbNoeud * dim; l++)
     {
        if (ContrainteXY[l, 0] == 0)
        {
               MReduite[ligneMReduite, colonneMReduite] = MAssemble[c, l];
               ligneMReduite++;
        }
     }
     colonneMReduite++;
  }
```

Dans le cas où la colonne est contrainte (c'est-à-dire `ContrainteXY[c, 0] == 1`) alors on doit prendre en compte la valeur de la contrainte pour changer le second membre, `ForceXY`. Pour le modifier on se déplace sur la colonne contrainte est on identifie les lignes non contrainte (c'est-à-dire `ContrainteXY[l, 0] == 0`) et on réalise l'opération ci-dessous sur `ForceXY[l]`

```
else // si le Noeud 'c' est contraint on change le second membre 'ForceXY'
{
  for (int l = 0; l < NbNoeud * dim; l++)
  {
    if (ContrainteXY[l, 0] == 0)
    {
      ForceXY[l] = ForceXY[l] - (ContrainteXY[c, 1] * MAssemble[l, c]);
    }
  }
}
```

Remarque : pour savoir où placer les coefficients dans la matrice réduite il faut définir l'entier `colonneMReduite qui` est incrémenté à chaque passage sur une colonne non contrainte et l'entier `ligneMReduite` qui est incrémenté à chaque passage sur une ligne non contrainte

d. Reduction du second membre

Pour réduire le second membre `ForceXY` on enlève les coefficients se situant sur la même ligne qu'un déplacement contraint et on place le résultat dans `ForceXYRed`. Et on affiche la matrice `MReduite` avec son second membre (pour le cas 1)

2499.12	0.00	-624.78	-624.78	-624.78	624.78	0.00	0.00	5.000
0.00	2499.12	-624.78	-624.78	624.78	-624.78	0.00	0.00	0.000
-624.78	-624.78	3016.71	0.00	0.00	0.00	-624.78	624.78	0.000
-624.78	-624.78	0.00	1249.56	0.00	0.00	624.78	-624.78	0.000
-624.78	624.78	0.00	0.00	3016.71	0.00	-624.78	-624.78	0.000
624.78	-624.78	0.00	0.00	0.00	1249.56	-624.78	-624.78	0.000
0.00	0.00	-624.78	624.78	-624.78	-624.78	2499.12	0.00	0.000
0.00	0.00	624.78	-624.78	-624.78	-624.78	0.00	2499.12	0.000

A la fin de cette opération on obtient donc l'équation matricielle suivante :

$$MReduite \times X = ForceXY$$

Il ne reste plus qu'à trouver X en calculant les déplacements

e. Résolution par trois méthodes différentes

Pour résoudre un système du type $A = XB$ nous avons mis dans notre programme les 3 types de résolutions vu en TP. Pour pouvoir choisir le type de résolution il suffit de changer la valeur de la méthode de résolution dans le fichier texte :

```
Méthode de résolution (LU = 1 / Gauss = 2 / Thomas = 3)
2
```

Ces méthodes de résolution ayant déjà été vues en TP nous n'en expliquerons que les grandes étapes. A noter que dans cette partie les valeurs montrées en exemple ont été prises pour le cas 1 du problème à résoudre.

Méthode LU

Comme vu en TP cette méthode est la plus efficace pour résoudre plusieurs fois avec la même matrice *A* car une fois la décomposition de A en LU effectué il suffit de faire deux opérations efficace pour trouver X.

La décomposition en L et U nous donne :

```
L :

      1.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
      0.00      1.00      0.00      0.00      0.00      0.00      0.00      0.00
     -0.25     -0.25      1.00      0.00      0.00      0.00      0.00      0.00
     -0.25     -0.25     -0.12      1.00      0.00      0.00      0.00      0.00
     -0.25      0.25      0.00      0.00      1.00      0.00      0.00      0.00
      0.25     -0.25      0.00      0.00      0.12      1.00      0.00      0.00
      0.00      0.00     -0.23      0.61     -0.23     -0.61      1.00      0.00
      0.00      0.00      0.23     -0.61     -0.23     -0.61      0.00      1.00

U :
   2499.12      0.00   -624.78   -624.78   -624.78    624.78      0.00      0.00
      0.00   2499.12   -624.78   -624.78    624.78   -624.78      0.00       0.0
      0.00      0.00   2704.32   -312.39      0.00      0.00   -624.78    624.78
      0.00      0.00      0.00    901.08      0.00      0.00    552.61   -552.61
      0.00      0.00      0.00      0.00   2704.32    312.39   -624.78   -624.78
      0.00      0.00      0.00      0.00      0.00    901.08   -552.61   -552.61
      0.00      0.00      0.00      0.00      0.00      0.00   1532.64      0.00
      0.00      0.00      0.00      0.00      0.00      0.00      0.00   1532.64
```

On calcule d'abord Y tel que $L \times Y = B$.

L Étant une matrice triangulaire cette étape est efficace, il suffit de commencer par calculer le dernier coefficient de Y et de remonter le résultat jusqu'au calcul du premier coefficient

```
Y :

  5.00000  0.00000  1.25000  1.39439  1.25000 -1.39439 -1.13270  0.00000
```

Après on calcule X tel que $U \times X = Y$ et on obtient la solution. Comme précédemment, U étant une matrice triangulaire, le calcul de X et très efficace. Il commencer par calculer le dernier coefficient de X et on remonte ainsi jusqu'à obtenir le premier.

```
X :

0.326  0000  0.00052  0.00200  0.00052 -0.00200 -0.00074  0.00000
```

Méthode de Gauss

Cette méthode est moins efficace que la méthode LU mais elle est plus facile à mettre en œuvre.

Dans un premier temps on triangularise la matrice A. Cette partie nécessite 7 étapes. A chaque étape la matrice est recalculée. Une colonne partie de la colonne correspondante à l'étape est remplie de 0 jusqu'à obtenir une matrice triangulaire.

```
...Triangularisation...

 Etape    1 :

 2499.12      0.00   -624.78   -624.78   -624.78    624.78      0.00      0.00          5.000
    0.00   2499.12   -624.78   -624.78    624.78   -624.78      0.00      0.00          0.000
    0.00   -624.78   2860.51   -156.20   -156.20    156.20   -624.78    624.78          1.250
    0.00   -624.78   -156.20   1093.37   -156.20    156.20    624.78   -624.78          1.250
    0.00    624.78   -156.20   -156.20   2860.51    156.20   -624.78   -624.78          1.250
    0.00   -624.78    156.20    156.20    156.20   1093.37   -624.78   -624.78         -1.250
    0.00      0.00   -624.78    624.78   -624.78   -624.78   2499.12      0.00          0.000
    0.00      0.00    624.78   -624.78   -624.78   -624.78      0.00   2499.12          0.000
```

…

```
Etape    7 :

  2499.12      0.00   -624.78   -624.78   -624.78    624.78      0.00      0.00          5.000
     0.00   2499.12   -624.78   -624.78    624.78   -624.78      0.00      0.00          0.000
     0.00      0.00   2704.32   -312.39      0.00      0.00   -624.78    624.78          1.250
     0.00      0.00      0.00    901.08      0.00      0.00    552.61   -552.61          1.394
     0.00      0.00      0.00      0.00   2704.32    312.39   -624.78   -624.78          1.250
     0.00      0.00      0.00      0.00      0.00    901.08   -552.61   -552.61         -1.394
     0.00      0.00      0.00      0.00      0.00      0.00   1532.64      0.00         -1.133
     0.00      0.00      0.00      0.00      0.00      0.00      0.00   1532.64          0.000
```

Dans un deuxième temps on calcule X tel que $A' \times X = Y$ (A' représente la matrice triangulaire). Cette étape est appelée « la remontée » on commence par calculer le dernier coefficient de X et on remonte jusqu'au premier avec un calcul relativement simple. On obtient, bien sûr, les mêmes résultats qu'avec la méthode LU.

Méthode de Thomas

La méthode de Thomas est la méthode la plus efficace mais elle nécessite que la matrice soit tridiagonale. On vérifie que la matrice est tridiagonale en appelant la fonction `VerifMTridiagonale` une fonction booléenne qui retourne `true` si la matrice est tridiagonale est `false` si ce n'est pas le cas

Pour vérifier que la matrice est bien tridiagonale on se déplace dans un premier temps sur toutes les colonnes et on vérifie qu'il y a seulement 3 coefficients sur chaque colonne. Si ce n'est pas le cas pour une des colonnes la fonction retourne `false` : la matrice n'est pas tridiagonale.
Dans un deuxième temps on se déplace sur toutes les lignes et de la même manière on vérifie qu'il a seulement 3 coefficients sur chaque ligne, si ce n'est pas le cas pour une ligne, la fonction retourne `false`, la matrice n'est pas tridiagonale.

```
for (int c = 1; c < TAILLE - 1; c++)
		{
			int compte = 0;
			for (int l = 0; l < TAILLE; l++)
			{
				if (M[l, c] != 0)
				{
					compte++;
				}
			}
			if (compte != 3)
			{
				return false;
			}
		}
```

Cette vérification astucieuse permet de vérifier si la matrice est bien tridiagonale mais aussi si la matrice est bien symétrique

Remarque : la première et la dernière colonne d'une matrice tridiagonale ont 2 coefficients. De même pour la première et la dernière ligne d'une matrice diagonale. Pour ces colonnes et ces lignes il faut donc effectuer un traitement spécial vérifiant qu'il n'y a que 2 coefficients. Pour ce faire il suffit de reprendre la partie ci-dessus en changeant la deuxième condition

```
if (compte != 2)
	{
		return false;
	}
```

Une fois qu'on est sûr que la matrice est tridiagonale on identifie ces 3 diagonales et on stock chaque diagonale dans `CoeffA` pour la première diagonale, dans `CoeffB` pour la diagonale et dans `CoeffC` pour la troisième diagonale

```
for (int i = 0; i < TAILLE; i++)
{
	for (int j = 0; j < TAILLE; j++)
	{
		if (A[i, j] != 0)
		{
			if (i - j == 1) // diagonale 1
			{
				CoeffA[i - 1] = A[i, j];
			}

			if (i == j) // diagonale 2
			{
				CoeffB[i] = A[i, j];
			}
			if (i - j == -1) // diagonale 3
			{
				CoeffC[j - 1] = A[i, j];
			}
		  }
	}
}
```

Une fois les diagonales identifiées, on utilise l'algorithme de Thomas vu en TP pour résoudre le système. La résolution avec Thomas n'a pas été utilisé car dans les 3 cas étudié la matrice n'était pas tridiagonale

f. Calcul des réactions des supports

Enfin, il faut calculer les réactions des supports, pour ce faire il faut identifier les lignes de la matrice assemblée qui correspondent à un déplacement contraint (c'est-à-dire `ContrainteXY[l, 0] == 1`). Lorsqu'une ligne est identifiée on effectue le produit scalaire de la ligne par le `Deplacement`

```
for (int l = 0; l < NbNoeud * dim; l++)
     {
         if (ContrainteXY[l, 0] == 1)
         {
             double Somme = 0;

             for (int c = 0; c < NbNoeud * dim; c++)
             {
                 if (ContrainteXY[c, 0] == 0)
                 {
                     Somme += MAssemble[l, c] * Deplacement[c];

                 }
             }

             ForceXY[l] = Somme;
         }
```

On affiche ensuite les résultats

```
Forces appliqués aux noeuds :

 N1 :  Fx =  -2.500000 N  Fy =   2.038252 N
 N2 :  Fx =  -2.500000 N  Fy =  -2.038252 N
 N3 :  Fx =   5.000000 N  Fy =   0.000000 N
 N4 :  Fx =   0.000000 N  Fy =   0.000000 N
 N5 :  Fx =   0.000000 N  Fy =   0.000000 N
 N6 :  Fx =   0.000000 N  Fy =   0.000000 N
 N7 :  Fx =   0.000000 N  Fy =   0.461748 N
 N8 :  Fx =   0.000000 N  Fy =  -0.461748 N

  Somme des Forces = 0.00
```

Pour s'assurer de l'équilibre du système on effectue la somme des forces (c'est-à-dire la somme des valeurs de « ForceXY ») et on affiche le résultat. Le système est équilibré si la somme des forces est nulle.

g. Calcul des tensions de chaque barre

Pour calculer les tensions de chaque barre on se base sur la loi de Hooke qui nous dit que :

$$T = K \times \Delta x$$

Avec :

- T : la tension de la barre (en N)
- K : la raideur de la barre (en N/m)
- Δx : l'allongement de la barre (en m)

Pour connaitre l'allongement de la barre il faut connaitre la longueur initiale de la barre qui a été précédemment stocké dans le tableau « LongueurB » et la longueur finale, après le déplacement. Pour calculer les longueurs finales comme précédemment on identifie les nœud Ni et Nj entourant la barre et on calcule les nouvelles positions des nœuds :

```
// Nj
double Xi = TabCoordonnee[Ni - 1, 0] + Deplacement[(Ni - 1) * 2]; // X
double Yi = TabCoordonnee[Ni - 1, 1] + Deplacement[(Ni - 1) * 2 + 1]; // Y
// Nj
```

```
double Xj = TabCoordonnee[Nj - 1, 0] + Deplacement[(Nj - 1) * 2]; // X
double Yj = TabCoordonnee[Nj - 1, 1] + Deplacement[(Nj - 1) * 2 + 1]; // Y
```

On calcule ensuite la nouvelle longueur de chaque barre :

```
double Longueur = Math.Sqrt(Math.Pow(Xi - Xj, 2.0) + Math.Pow(Yi - Yj, 2));
```

Et l'allongement

```
double Allongement = Longueur - LongeurB[i];
```

Et enfin on utilise la formule de Hooke pour obtenir les tensions de chaque barre

```
Tension[i] = Allongement * TabRaideur[i];
```

Une fois les tensions déterminées on les affiche :

```
...Calcul des tensions de chaque barre...

 Tension de chaque barre :

 E1 :   0.470566 N (traction)
 E2 :   2.905655 N (traction)
 E3 :  -0.603194 N (compression)
 E4 :   2.905655 N (traction)
 E5 :  -0.603194 N (compression)
 E6 :   0.470566 N (traction)
 E7 :  -0.452883 N (compression)
 E8 :   0.676431 N (traction)
 E9 :   0.654212 N (traction)
 E10 :  0.676431 N (traction)
 E11 :  0.654212 N (traction)
 E12 : -0.452883 N (compression)
```

h. Vérification du bon fonctionnement de la résolution du système

Pour s'assurer du bon fonctionnement du programme et pour pouvoir le debugger plus facilement nous avons fait des vérifications à chaque fois qu'une matrice ou un vecteur été calculé.

Par exemple dans la Methode LU, après la décomposition de A en LU on vérifie que le produit de L par U est bien égale à A. Pour ce faire on commence par faire le produit de L par U en appelant la fonction « ProduitM »

```
double[,] LU = ProduitM(L, U);
```

Et puis on vérifie que la matrice LU est bien égale à A en appelant la fonction « VerificationM ». Si les deux matrices sont égales on affiche

```
========> L x U = MReduite
```

à l'inverse si elles ne sont pas égales on affiche

```
========> ERREUR : L x U pas égale à MReduite
```

La fonction « VérificationM » est une fonction booléenne qui renvoie **true** si la valeur absolue de la différence entre deux coefficients des matrices en paramètres d'entrée est inférieure à 0.000001, si cette différence est supérieure à 0.000001 la fonction renvoie **false**. Il n'est pas possible de prendre 0 comme valeur minimale de différence car en raison des calculs de la machine sur des nombres flottants a précision finie, les valeurs dans les matrices sont très rarement égales. Nous avons donc arbitrairement choisi 10^{-6} comme valeur de seuil.

Il existe les des fonctions similaires («ProduitV » et « VerficationV ») pour vérifier que le produit de 2 vecteurs est égale à une matrice.

i. Ecriture des résultats dans « Resutat.txt»

Cette partie est homologue à la lecture des données vue en première partie. Dans un premier temps on ouvre le fichier grâce au chemin spécifié

```
StreamWriter Resultat = new StreamWriter(CheminResultat);
```

Et grâce à la fonction dans l'exemple ci-dessous on écrit successivement les déplacements X et Y de chaque nœud, les composantes X et Y de chaque Force appliqué à chaque nœud et les tensions de chaque barre

```
// ecriture des déplacement
Resultat.WriteLine("Deplacement de chaque noeuds (en mètre)");
Resultat.WriteLine("    X            Y");
```

Ce qui nous le fichier « Resultat.txt » suivant :

```
Deplacement de chaque noeuds (en mètre)
   X           Y
N1    0.00000000  0.00000000
N2    0.00000000  0.00000000
N3    0.00326235  0.00000000
N4    0.00052259  0.00200070
N5    0.00052259 -0.00200070
N6   -0.00073906  0.00000000
N7    0.00000000  0.00000000
N8    0.00000000  0.00000000

Force appliqués a chaque noeuds (en newton)
   Fx          Fy
N1   -2.500000  2.038252
N2   -2.500000  -2.038252
N3    5.000000  0.000000
N4    0.000000  0.000000
N5    0.000000  0.000000
N6    0.000000  0.000000
N7    0.000000  0.461748
N8    0.000000  -0.461748

Tension de chaque barre (en newton)
E1    0.4706 N  (traction)
E2    2.9057 N  (traction)
E3   -0.6032 N  (compression)
E4    2.9057 N  (traction)
E5   -0.6032 N  (compression)
E6    0.4706 N  (traction)
E7   -0.4529 N  (compression)
E8    0.6764 N  (traction)
E9    0.6542 N  (traction)
E10   0.6764 N  (traction)
E11   0.6542 N  (traction)
E12  -0.4529 N  (compression)
```

Interprétation des résultats

1er cas : Force de 5N à 0° sur le nœud N3

On observe que quand on applique une force horizontale sur le nœud 3 le nœuds 4 et 5 s'écarte de la structure de façon symétrique ce qui ramène le nœud 6 vers le nœud 3. On remarque aussi que la force étant horizontale il n'y a pas de déplacement sur horizontale sur les nœuds 3 et 6. De plus les déplacements des nœuds 1, 2, 7 et 8 sont bien nul. On est déduit que ces déplacements sont cohérents au problème mécanique posé.

```
Deplacement des noeuds :

U1  =   0.000000 mm  V1  =   0.000000 mm
U2  =   0.000000 mm  V2  =   0.000000 mm
U3  =   3.262350 mm  V3  =   0.000000 mm
U4  =   0.522591 mm  V4  =   2.000703 mm
U5  =   0.522591 mm  V5  = -2.000703 mm
U6  = -0.739056 mm  V6  =   0.000000 mm
U7  =   0.000000 mm  V7  =   0.000000 mm
U8  =   0.000000 mm  V8  =   0.000000 mm
```

```
...Calcul des reactions des supports...

 Forces appliqués aux noeuds :

 N1 :  Fx =  -2.500000 N  Fy =   2.038252 N
 N2 :  Fx =  -2.500000 N  Fy =  -2.038252 N
 N3 :  Fx =   5.000000 N  Fy =   0.000000 N
 N4 :  Fx =   0.000000 N  Fy =   0.000000 N
 N5 :  Fx =   0.000000 N  Fy =   0.000000 N
 N6 :  Fx =   0.000000 N  Fy =   0.000000 N
 N7 :  Fx =   0.000000 N  Fy =   0.461748 N
 N8 :  Fx =   0.000000 N  Fy =  -0.461748 N

  Somme des Forces = 0.00

...Calcul des tensions de chaque barre...

 Tension de chaque barre :

 E1 :  0.470566 N (traction)
 E2 :  2.905655 N (traction)
 E3 :  -0.603194 N (compression)
 E4 :  2.905655 N (traction)
 E5 :  -0.603194 N (compression)
 E6 :  0.470566 N (traction)
 E7 :  -0.452883 N (compression)
 E8 :  0.676431 N (traction)
 E9 :  0.654212 N (traction)
 E10 :  0.676431 N (traction)
 E11 :  0.654212 N (traction)
 E12 :  -0.452883 N (compression)
```

2eme cas : Force de 10N à 315° sur le nœud N5

```
 Deplacement des noeuds :

 U1  =  0.000000 mm  V1  =  0.000000 mm
 U2  =  0.000000 mm  V2  =  0.000000 mm
 U3  =  3.568477 mm  V3  = -3.568477 mm
 U4  =  0.000000 mm  V4  =  0.000000 mm
 U5  =  2.956223 mm  V5  =-11.317685 mm
 U6  = -2.090365 mm  V6  = -2.090365 mm
 U7  =  0.000000 mm  V7  =  0.000000 mm
 U8  =  0.000000 mm  V8  =  0.000000 mm

...Calcul des reactions des supports...

 Forces appliqués aux noeuds :

 N1 :  Fx =  -4.459029 N  Fy =   4.459029 N
 N2 :  Fx =  -2.612039 N  Fy =   0.000000 N
 N3 :  Fx =   0.000000 N  Fy =   0.000000 N
 N4 :  Fx =   0.000000 N  Fy =   0.000000 N
 N5 :  Fx =   7.071068 N  Fy =  -7.071068 N
 N6 :  Fx =   0.000000 N  Fy =   0.000000 N
 N7 :  Fx =   2.612039 N  Fy =   2.612039 N
 N8 :  Fx =  -2.612039 N  Fy =   0.000000 N

  Somme des Forces = 0.00

...Calcul des tensions de chaque barre...

 Tension de chaque barre :

 E1 :  0.000000 N (compression)
 E2 :  6.306019 N (traction)
 E3 :  0.112479 N (traction)
 E4 :  0.112479 N (traction)
 E5 :  6.455073 N (traction)
 E6 :  2.890643 N (traction)
 E7 :  0.000000 N (compression)
 E8 :  0.038605 N (traction)
 E9 :  3.693981 N (traction)
 E10 : 4.134286 N (traction)
 E11 : 0.038605 N (traction)
 E12 :-2.325088 N (compression)

...Ecriture des résultats dans un fichier texte...
```

3eme cas : Force à 20N à 270° sur N4

On observe que quand on applique une force verticale sur le nœud 4 la structure se déforme de façon symétrique (d'axe N4N5) à la même manière que dans le cas 1. Les nœuds 3 et 4 sont expulsés vers la droite et la gauche de structure de façon symétrique et les nœuds 4 et 5 n'ont pas de déplacement sur X. On remarque que le nœud 4 se déplace beaucoup sur Y ce qui nous indique que c'est un point faible de la structure et qu'en revanche le nœud 5 ne se déplace pas du tout

```
 Deplacement des noeuds :

 U1  =  0.000000 mm  V1  =  0.000000 mm
 U2  =  0.000000 mm  V2  =  0.000000 mm
 U3  = -8.002812 mm  V3  = -8.002812 mm
 U4  =  0.000000 mm  V4  =-32.011247 mm
 U5  =  0.000000 mm  V5  =  0.000000 mm
 U6  =  8.002812 mm  V6  = -8.002812 mm
 U7  =  0.000000 mm  V7  =  0.000000 mm
 U8  =  0.000000 mm  V8  =  0.000000 mm

...Calcul des reactions des supports...

 Forces appliqués aux noeuds :

 N1 :  Fx =   0.000000 N  Fy =   0.000000 N
 N2 :  Fx =  10.000000 N  Fy =  10.000000 N
 N3 :  Fx =   0.000000 N  Fy =   0.000000 N
 N4 :  Fx =   0.000000 N  Fy = -20.000000 N
 N5 :  Fx =   0.000000 N  Fy =   0.000000 N
 N6 :  Fx =   0.000000 N  Fy =   0.000000 N
 N7 :  Fx =   0.000000 N  Fy =   0.000000 N
 N8 :  Fx = -10.000000 N  Fy =  10.000000 N

  Somme des Forces = 0.00

...Calcul des tensions de chaque barre...

 Tension de chaque barre :

 E1 :  2.249223 N (traction)
 E2 :  0.564981 N (traction)
 E3 :  -11.700036 N (compression)
 E4 :  -14.142136 N (compression)
 E5 :  0.564981 N (traction)
 E6 :  0.000000 N (compression)
 E7 :  2.249223 N (traction)
 E8 :  -11.700036 N (compression)
 E9 :  0.564981 N (traction)
 E10 :  0.564981 N (traction)
 E11 :  -14.142136 N (compression)
 E12 :  0.000000 N (compression)

...Ecriture des résultats dans un fichier texte...
```

Conclusion

Ce projet nous a permis de découvrir de nouvelles applications pour la programmation en C# et nous avons pu expérimenter et nous entraîner à répondre à un problème mathématique lors de la création d'un programme de calcul. Nous avons rencontré quelques problèmes que nous avons pu résoudre grâce au travail de groupe et aux critiques positives de chacun. En ce qui concerne le code en général, nous avons choisi de donner la priorité à la lisibilité et à la facilité de compréhension plutôt que d'essayer de créer des formules trop complexes.