INVERSION DE MATRICE VIA DECOMPOSITION LU MATHÉMATIQUE

Résumé

Ce rapport traite d'un programme réalisé en C# permettant l'inversion d'une matrice en utilisant sa décomposition LU, à condition que celle-ci soit inversible.

Adrien Mousty

moustyadrien@hotmail.com

DIAGRAMME DE CLASSE

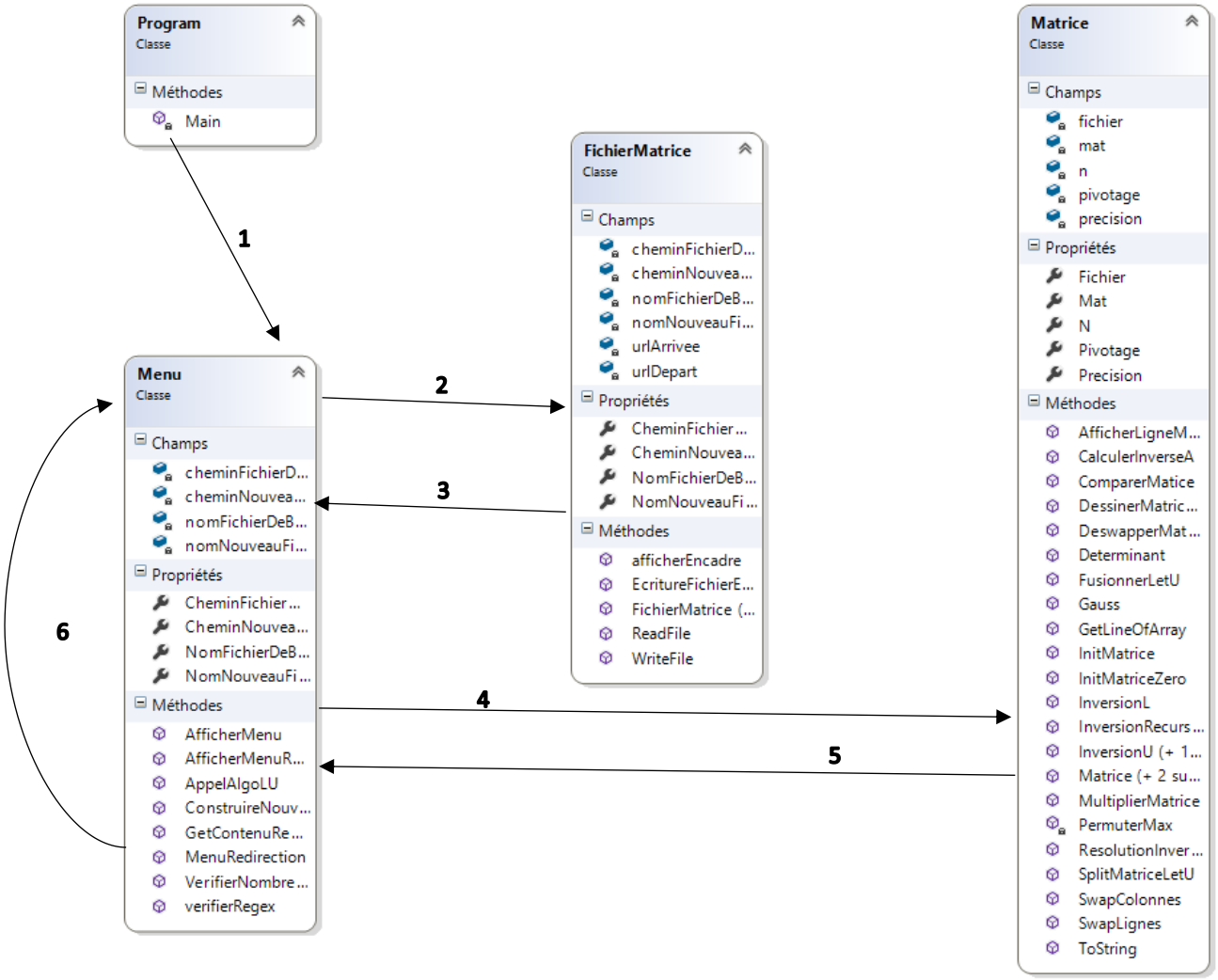


Figure 1 Diagramme de classe

Le diagramme de classe généré par Visual Studio 2015 permet d'avoir une vision d'ensemble du projet. En effet, il montre toutes les classes existantes ainsi que leurs champs, propriétés et fonctions.

1. Le programme appelle directement le menu.
2. Le menu va proposer des choix simples à l'utilisateur :

```
################################################
# Menu.                                        #
################################################

1. Matrice par défaut
2. Choisir parmi les matrices existantes (.txt uniquement)
3. Matrice entrée par l'utilisateur
0. Quitter

################################################
# Choix : _
```

Figure 2 Menu principal

3. Selon le choix entré par l'utilisateur la classe « FichierMatrice » va :
   a. Lire un fichier texte contenant une matrice existante pour la retourner sous forme d'objet « Matrice » à la classe « Menu ».
   b. Transformer les données entrées par l'utilisateur en fichier « .txt » pouvant être lu par le programme. Ensuite il effectuera l'étape a.
4. La classe « Menu » va appeler la classe « Matrice » qui, elle, effectuera toutes les opérations sur les matrices.
5. Après avoir effectué les calculs (ou avoir déclenché une exception) , le programme retourne dans la classe « Menu » afin de proposer à l'utilisateur de recommencer ou non le calcul sur une autre matrice.

```
################################################
# Menu.                                        #
################################################

Voulez-vous recommencer ?

1. Oui
2. Non

################################################
# Choix : _
```

Figure 3 Menu recommencer

6. Si l'utilisateur veut, il peut recommencer ou quitter le programme.

GAUSS

```
// Resolution de la matrice avec Gauss.
public Matrice Gauss()
{
    try
    {
        Matrice U = new Matrice(n, this.fichier.CheminNouveauFichier, this.fichier.NomNouveauFichier, this.precision,
        Matrice L = InitMatrice(n);
        U.Mat = this.Mat;
        // la valeur qui sera insérée dans la matrice
        double m = 0;
        // Si le pivot est égal à true à un moment donné, alors on arrête tout ça ne sert à rien.
        bool pivotnull = false;
        //-> Affichage à l'écran
        fichier.afficherEncadre(2, "Matrice de départ :");
        fichier.EcritureFichierEtAffichage("Precision : " + precision);
        fichier.EcritureFichierEtAffichage(U.ToString());
        fichier.afficherEncadre(3, "Début de Gauss.");
        for (int k = 0; k < n-1 && !pivotnull; k++)
        {
            // Le pivot vaut zéro => on permute
            if (U.mat[k][k] == 0)
            {
                fichier.EcritureFichierEtAffichage("/!\\ Nécéssité de pivoter la ligne " + (k + 1) + "/!\\");
                // Si la permutation n'a pas fonctionné le pivot reste nul et on ne peut pas continuer.
                if (!PermuterMax(U, L, k))
                {
                    fichier.EcritureFichierEtAffichage("Le pivot est null");
                    pivotnull = true;
                }
            }

            for (int i = 0; i < n && !pivotnull; i++)
            {
                // On ne s'occupe que du triangle inférieur car nous faisons Gauss
                if (i > k)
                {
                    // On ne sait pas diviser par 0, donc si le dénominateur = 0 on ne change rien
                    // Le cas échéant, M vaut la matrice U

                  m = U.mat[k][k] == 0 ? 0 : U.mat[i][k] / U.mat[k][k];
                  fichier.EcritureFichierEtAffichage("m[" + (i + 1) + "][" + (k + 1) + "] = " + m);
                  // On inscrit cette partie dans la matrice L
                  L.mat[i][k] = m;
                  for (int j = k; j < n && m != 0; j++)
                  {
                      // ~> application de Gauss
                      U.mat[i][j] = U.mat[i][j] - (m * U.mat[k][j]);
                  }
              }
          }
          fichier.EcritureFichierEtAffichage("\nMatrice U après l'étape " + (k + 1) + " (" + (n - k - 2) + " étapes restant
          fichier.EcritureFichierEtAffichage(U.ToString());
      }
      fichier.afficherEncadre(4, "Fin de Gauss.");
      fichier.EcritureFichierEtAffichage("Affichage des résultats :");
      // Affiche si'l y a eu ou non des permutations
      if (U.pivotage.Count != 0) { fichier.EcritureFichierEtAffichage("-> Nombre de permutations : " + U.pivotage.Count); }
      else { fichier.EcritureFichierEtAffichage("-> Il n'y a pas eu de permutation."); }
      return FusionnerLetU(n, L, U);
  }
  catch (Exception) { throw; }
}
```

Figure 4 Gauss

Inversion de matrice via décomposition LU

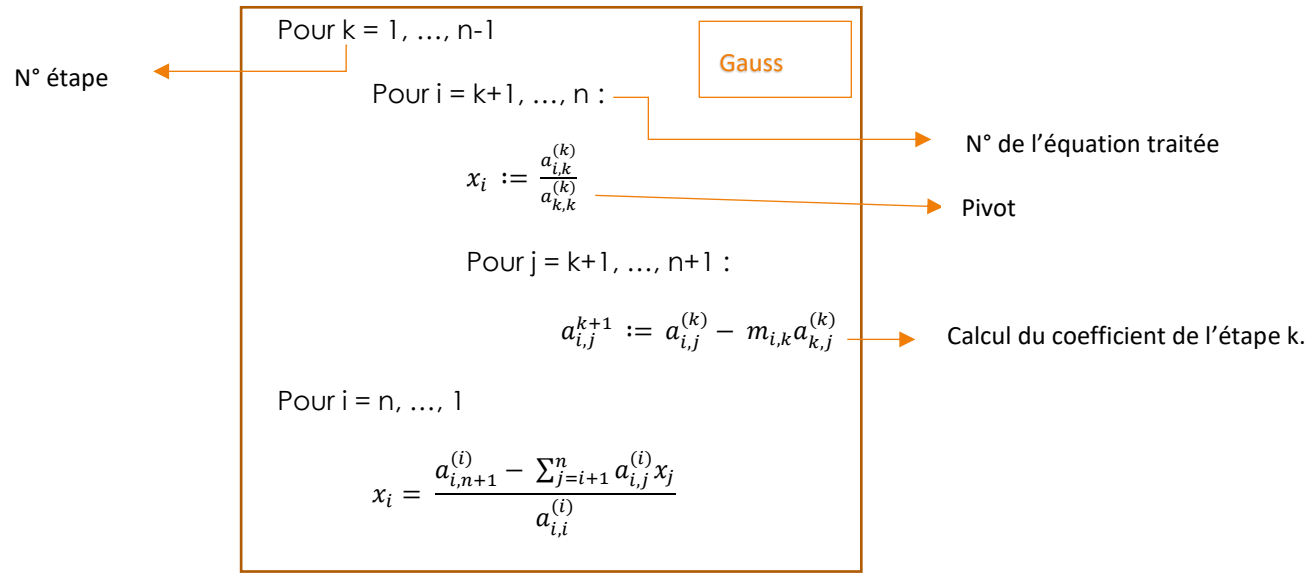
La matrice **U** est initialisée avec comme matrice, la matrice par défaut. Tandis que la matrice **L** est initialisée à une matrice identité[1].

À chaque ligne, nous vérifions si le pivot[2] est null. S'il l'est alors nous tentons de pivoter[3] cette ligne avec une ligne inférieure. Si ça n'est pas possible alors la matrice n'est pas triangulable.

Suite à cela, nous allons maintenant appliquer la méthode de Gauss qui consiste à n'avoir que des zéros dans le triangle inférieur de la matrice.

$$\begin{pmatrix}1 & 1 & 1\\0 & 1 & 0\\1 & 2 & 3\end{pmatrix} \Longrightarrow \begin{pmatrix}1 & 1 & 1\\0 & 1 & 0\\0 & 1 & 2\end{pmatrix} \Longrightarrow \begin{pmatrix}1 & 1 & 1\\0 & 1 & 0\\0 & 0 & 2\end{pmatrix}$$

La méthode est relativement facile à comprendre, pour avoir les zéros en position [n][1][4], il suffit de diviser la ligne n par la valeur de n-1. Ou en d'autres termes plus simples :



[1] Matrice ne contenant que des zéros, excepté dans sa diagonale, qui content la valeur 1.

[2] Le pivot se situe toujours en [i][i], soit toujours en diagonale.

[3] Voir la méthode de pivotage de ligne.

[4] Les indices en informatique commencent à 0, ici ils commencent à 1 comme en mathématique

INVERSION U

La formule pour l'inversion de U est un peu plus contraignante, celle-ci utilise notamment une division (il faut donc prévoir un cas ou le déterminant vaut zéro) et se fait en partant de la fin.

Le principe est le même que pour l'inversion de la matrice L. La seule chose qui change est la formule.

```
    for (int i = n - 1; i >= 0; i--)
    {
        for (int j = 0; j < n; j++)
        {
            // Le dernier élément on ne fait que recopier de la matrice de Kronecker
            // /!\ on ne peut pas diviser par zero
            if (i == n - 1) { y.mat[i][j] = u.mat[i][i] == 0 ? 0 : kronecker.mat[i][j] / u.mat[i][i]; }
            // Les autres il faut appliquer la formule page 28 (explication dans la fonction correspondante)
            else { y.mat[i][j] = InversionU(kronecker, u, y, i, j); }
            fichier.EcritureFichierEtAffichage(lettr + ((i + 1) + "") + ((j + 1) + " = ") + y.mat[i][j]+ "\n");
        }
    }

    fichier.afficherEncadre(8, "Matrice " + lettr + " après inversion : ");
    fichier.EcritureFichierEtAffichage(y.ToString());
    return y;
}

public double InversionU(Matrice kr, Matrice u, Matrice y, int i, int j)
{
    // Si le num = 0 alors le résultat vaut zero.
    if (u.mat[i][i] == 0) { return 0; }
    else
    {
        // On commence par la fin, k = la dimension de la matrice -1
        int k = n - 1;
        // Au début le résultat vaut la valeur dans Kronecker en position [i][j] divisé par U[i][i]
        double doublY = kr.mat[i][j] / u.mat[i][i];

        while (k > i)
        {
            fichier.EcritureFichierEtAffichage("    ~> Calcul de [" + (i + 1) + "," + (j + 1) + "] =>> (" + doublY
            // Utilisation de l'algorithme donné page 28 du cours.
            // Ensuite, de cette valeur on soustrait u.mat[i][k] * y.mat[k][j] toujours divisé par u[i][i]
            // Qu'importe le nombre d'itérations, il n'y a que K qui évolue jusqu'à ce qu'il soit = à i.
            doublY -= u.mat[i][k] * y.mat[k][j] / u.mat[i][i];
            k--;
        }
        return doublY;
    }
}
```

Figure 5 Inversion U

La formule est la suivante

$$Y_{n,j} = \frac{\delta_{n,j}}{u_{n,n}}$$

$$Y_{n-1,j} = \frac{\delta_{n-1,j} - u_{n-1,n}\, y_{n,j}}{u_{n-1,n-1}}$$

…

$$Y_{1,j} = \frac{\delta_{1,j} - u_{1,2}\, y_{2,j} - \cdots - u_{1,n}\, y_{n,j}}{u_{1,1}}$$

RESUME

- Projet effectué seul
- Langages utilisés : C#
- Utilisation de nouveautés du C#6
- Utilisation de fichiers pour la lecture ainsi que le stockage du résultat.
- Ce projet fut plus mathématique que les autres, troquant les interfaces graphiques contre des algorithmes d'inversion de matrice.
- Note finale : 16/20