**Classe Matrix**

Cette documentation permet de définir plus précisément les différentes ressources offerte par la classe Matrix. Il s’agit d’un objet C++ permettant de réaliser des outils matriciels de manière simple et efficace

1. **Présentation globale**

Matrix est en fait simplement une bibliothèque organisée sous la forme d’une classe C++. L’objectif est simplement d’instancier un objet Matrice et de réaliser des opérations basiques très facilement sans avoir à en connaitre les algorithmes complexes.

Cette bibliothèque a été construite a un niveau d’abstraction assez bas et par conséquent n’utilise que des bibliothèques généralement incluse de base dans les compilateur (iostream par exemple). La seule bibliothèque qu’il est nécessaire d’ajouter est la bibliothèque d’erreur implémenter par ailleurs. Elle permet de définir les classes d’exception qui sont lancée lors de réalisation d’opération non valide.

La classe est conçue pour permettre une utilisation la plus intuitive possible, ainsi les opérateurs courants sont surchargés, et de nombreuses opérations basiques sont implémentée (transposée, inverse).

En plus de la classe quelques fonction simple sont créées permettant de générer des objets de type matrice assez particulier, tel l’identité, ou des matrice diagonale. Ils permettent ainsi de créer des matrices sans passer directement par un constructeur qui aurait rendu la syntaxe moins intuitive.

1. **Présentation des fichiers de base**

La classe Matrix utilise seulement 3 fichiers :

* Le fichier d’en-tête global.h. Il contient principalement les include nécessaires pour la classe, et les paramètres de configuration. En réalité la seule configuration que l’on peut faire est de déclarer le type de valeur qu’on les matrice. Les test et la programmation ont été réalisé pour des matrice réelle, dont les valeur sont des double, mais en réalité, le programme peut s’exécuter avec tout type de donnée, la seule restriction étant de pouvoir réaliser les opérations élémentaire (+, -, \*, /) sur ces données. Il suffit pour cela de redéfinir le type val\_matrix pour changer le type des valeurs stockées dans la matrice.
* Le fichier d’en-tête Matrix.h. Il contiend simplement la déclaration des fonctions de la matrice et des fonctions simple extérieures. Chaque fonction est brièvement commentée afin de savoir ce qu’elle réalise, et ainsi en regardant simplement ce fichier, on peut utiliser sans difficulté le programme.
* Le fichier source Matrix.cpp. C’est là que réside réellement le code source de toutes les fonctions.

Sans faire réellement partie de la bibliothèque, on peut toutefois noter la présence d’un fichier main.cpp qui contient simplement un code permettant de tester les différentes fonctions. Lors de son exécution, il se contente de proposer un menu en mode console qui permet d’exécuter chaque fonction que l’on désire à partir de matrice pré générée. En cas d’erreur il affiche également les exceptions reçues sans arrêter le programme.

1. **Présentation des différentes fonctions**
   1. *Les constructeurs :*

Pour créer un objet Matrix, trois types de constructeurs sont disponibles.

* Tout d’abord, en précisant les dimensions de la matrice. Si l’on précise le nombre de ligne et le nombre de colonnes, une matrice **initialisée à 0** sera créée correspondant aux dimensions souhaitées. Si seulement une dimension est donnée, cela créera une matrice carrée. Si une dimension a une valeur négative ou nulle, une exception sera lancée
* Une autre possibilité est de faire une copie de la Matrice (constructeur par copie). Dès lors le nouvel objet copiera les dimensions et les valeurs de l’objet source, sans créer de lien par adresse. C’est également ce constructeur qui est indirectement utilisée lorsqu’on crée une matrice à partir d’une fonction extérieure comme décris plus bas
* La dernière possibilité est un constructeur sans argument, qui crée une Matrice de dimension nulle. L’objet ainsi créer ne peut servir a rien excepté créer une copie plus tard ou être redimensionné par la suite.
  1. *Les opérateurs d’accès aux valeurs :*

Pour accéder à une valeur, les opérateurs de parenthèse « () » ont été surchargés. On peut ainsi facilement accéder à n’importe quel élément en positionnant ses coordonnées sous la forme (ligne, colonne) dans la matrice. Les indice commencent a partir de 0 et se termine par la dimension-1. Si l’indice demandé sort de la matrice, une exception est lancée.

On peut également accéder a un élément en utilisant qu’une seule coordonnées. Dans ce cas la matrice est lue en ligne. Par exemple pour une matrice de 3 lignes et 3 colonnes, le paramètre numéro 3, donnera accès à la première valeur de la deuxième ligne. On a la représentation sur le schéma suivant des indices de la matrice :

* 1. *Les opérations avec un scalaire :*

Les opérateurs mathématiques de base (+, -, \* et /) peuvent être utilisé entre une matrice et un nombre simple. Dans ce cas l’opérateur s’appliquera sur chaque valeur de la matrice avec ce nombre. Par exemple ajouter le nombre 2 a une matrice consiste à ajouter 2 à chaque membre de la matrice. Il en est de même pour les 4 opérateurs.

On notera tout de même qu’il est indispensable que la matrice soit le premier membre de l’opération. Ainsi, si M est un objet de type matrice, l’opération 2\*M sera invalide, seule M\*2 fonctionnera. De même il est impossible de diviser un nombre terme à terme avec une matrice

* 1. *Les opérations matricielles :*

Les opérateurs de base sont également implémentés entre les matrices. Pour l’addition et la soustraction, il s’agit naturellement de l’opération terme à terme. Il est nécessaire pour se faire que les matrices aient exactement la même dimension, une exception sera lancée si cela n’est pas le cas.

La multiplication correspond au produit matriciel, et non au produit terme a terme. Les dimensions des matrices doivent évidemment être compatible, le nombre de ligne de la première égal au nombre de colonnes de la seconde, sans quoi, une exception sera lancée.

La division entre deux matrices n’existe pas et n’est donc pas implémentée.

* 1. *L’opérateur d’affectation :*

Tout comme le constructeur par copie, l’opérateur d’affectation (=), utilisé avec une matrice, permet d’obtenir une copie des valeurs de la matrice. Si la matrice cible n’est pas vide avant la copie, celle-ci sera effacé pour laisser place aux nouvelles valeurs. La matrice est également redimensionnée pour convenir à sa nouvelle forme.

* 1. *L’opérateur de comparaison :*

Il est possible de vérifier que deux matrice sont égales, grâce à l’opérateur de comparaison correspondant (==). Celui-ci renvoie vrai lorsqu’elles sont identiques en dimension et en valeur, et faux sinon.

Aucune autre comparaison n’est implémenter, la différence peut se faire simplement en prenant le complémentaire de l’opérateur d’égalité.

Du reste, il n’existe pas de relation d’ordre total dans l’espace des matrices, donc les opérateur de comparaison d’infériorité ou de supériorité ne peuvent être définis.

* 1. *Les semi-opérateurs :*

Les semi-opérateurs sont en fait des fonctions pour lequel il n’existe pas d’opérateur proprement dit. Pour les utiliser il suffit donc d’appeler la fonction de la classe, généralement sans argument.

On peut ainsi, sur un matrice réalise la transposition (Matrix ::Transpose(), calculer son inverse (Matrix ::Inverse() ), son déterminant (Matrix ::Determinant()) ou sa trace (Matrix ::Trace()).

Il est a noter que pour ces trois dernier, il est nécessaire que la matrice soit carré, et si ce n’est pas le cas, une exception sera lancée. De même si l’on tente d’inverser une matrice non inversible, une exception sera lancée.

* 1. *L’accès aux dimensions :*

Il est possible d’accéder aux dimensions d’une matrice par la fonction Dimension. Celle-ci prend en paramètre 2 pointeurs sur entier, qui s’ils sont non nul, seront mis à jour avec le nombre de lignes et de colonnes de la matrices. Ces valeurs ne seront toutefois pas liées aux dimensions de la matrice et les modifier par la suite n’aura aucun effet.

Pour redimensionner la matrice, il est nécessaire d’utiliser la fonction Redim. Celle-ci prend en paramètre le nouveau nombre de ligne et de colonnes souhaité, et redimensionne la matrice pour avoir ces valeurs. Les valeurs seront conservées dans les parties non affectées par le changement, les nouvelles valeurs seront initialisées à 0.

* 1. *Les fonctions extérieures :*

Ce ne sont pas des fonctions qui font partie de la classes, mais au contraire des fonctions qui créent un objet de type Matrix. Elles permettent ainsi d’obtenir des matrice qui ne sont pas initialisée à 0, par exemple la matrice identité, une matrice diagonale, ou encore une matrice initialisée à 1.