Projet d'Informatique et Mathématiques Appliquées
en
Algèbre Linéaire Numérique

RÉDUCTION D'UNE MATRICE CREUSE SOUS FORME TRIANGULAIRE PAR BLOCS

Table des matières

1 Introduction

L'objectif de ce projet est le développement d'un algorithme permettant de découvrir la forme triangulaire par blocs d'une matrice non-symétrique carrée $\mathbf{A}$ d'ordre N.

S'il existe deux matrices de permutation $\mathbf{P}$ et $\mathbf{Q}$ telles que

$$\mathbf{PAQ} = \begin{pmatrix} \mathbf{B}_{11} & & & & & & \\ \mathbf{B}_{21} & \mathbf{B}_{22} & & & & & \\ \mathbf{B}_{31} & \mathbf{B}_{32} & \mathbf{B}_{33} & & & & \\ . & . & . & . & & & \\ . & . & . & & . & & \\ . & . & . & & & . & \\ \mathbf{B}_{K1} & \mathbf{B}_{K2} & \mathbf{B}_{K3} & . & . & . & \mathbf{B}_{KK} \end{pmatrix}$$

et si $K > 1$ alors on dira que $\mathbf{A}$ est **réductible** (**irréductible** sinon).

On considérera que la matrice $\mathbf{A}$ a été permutée en forme triangulaire par bloc (**BTF**) lorsque chaque $\mathbf{B}_{ii}$ est irréductible (sinon une décomposition plus fine peut en effet être trouvée).

Toute matrice n'est pas réductible, mais il existe de grandes classes de problèmes conduisant à des matrices réductibles. Lorsqu'une matrice est réductible sa forme BTF peut être utilisée pour résoudre de façon efficace le système $\mathbf{Ax} = \mathbf{b}$. En effet seules les matrices $\mathbf{B}_{ii}$ doivent être factorisées. Pour résoudre le système linéaire il suffit alors d'exploiter la factorisation des blocs diagonaux $\mathbf{B}_{ii}$ pour résoudre dans l'ordre croissant des indices i, $1 \leq i \leq K$, les équations suivantes

$$\mathbf{B}_{ii} y_i = \begin{cases} (\mathbf{Pb})_i & \text{si} \quad i = 1 \\ (\mathbf{Pb})_i - \sum_{j=1}^{i-1} \mathbf{B}_{ij} \mathbf{y}_j & \text{si} \quad i > 1, \end{cases}$$

où $\mathbf{y} = \mathbf{Q}^{\mathrm{T}}\mathbf{x}$.

Les algorithmes pour découvrir la forme BTF d'une matrice (calculer les permutations $\mathbf{P}$ et $\mathbf{Q}$) que nous nous proposons de développer fonctionnent en deux étapes :

- Etape (1) : calcul d'une permutation $\mathbf{Q}_1$ des colonnes de la matrice telle que $\mathbf{AQ}_1$ possède des éléments non nuls sur la diagonale.
- Etape (2) : calcul d'une permutation symétrique $\mathbf{P}$ telle que $\mathbf{PAQ}_1\mathbf{P}^t$ est en BTF.

L'étape (1) n'est pas l'objet de ce travail. On supposera donc que

la matrice initiale $\mathbf{A}$ ne possède pas d'élément nul sur la diagonale

Composants principaux des algorithmes réalisant l'étape (2)

- Objectif : Calculer la permutation $\mathbf{P}$ afin que la matrice $\mathbf{PAP}^t$ soit BTF.
- On utilisera la notion de graphe orienté pour représenter la matrice et l'on ne représentera pas les éléments diagonaux (cycles simples du graphe orienté). Il existe un arc orienté entre le nœud i et j ssi $a_{i,j} \neq 0$.

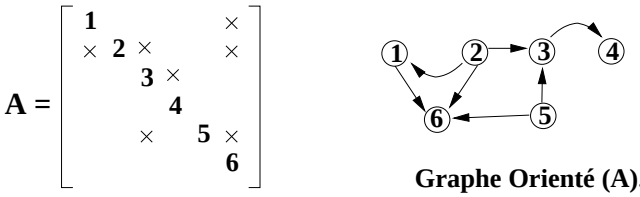


FIG. 1 – Correspondance entre matrice non-symétrique et graphe orienté

- Une permutation symétrique de la matrice correspond à une renumérotation des nœuds du graphe.
- S'il n'existe pas de chemin fermé (cycle) passant par tous les nœuds du graphe alors le graphe peut être partitionné en au moins deux groupes de nœuds S_1 et S_2 tels qu'il n'existe pas de chemin d'un nœud quelconque de S_1 vers un nœud de S_2. Si l'on numérote les nœuds de S_1 avant ceux de $S2$ alors la matrice permutée commence à révéler une forme BTF partielle : le même processus peut alors être appliqué récursivement à chacune des partitions (bloc de la matrice).

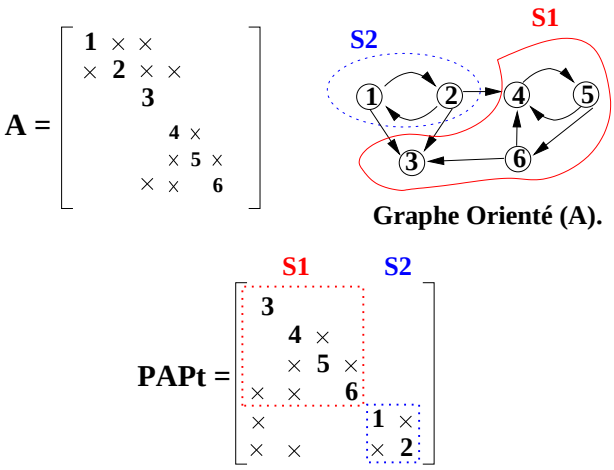


FIG. 2 – Permutation sous forme BTF partielle d'une matrice réductible

- On désigne par **composante forte** du graphe, l'ensemble des nœuds appartenant à un chemin fermé (cycle) de taille maximale.
- Les composantes fortes d'un graphe correspondent aux blocs diagonaux $\mathbf{B}_{ii}$ de la matrice au format BTF.

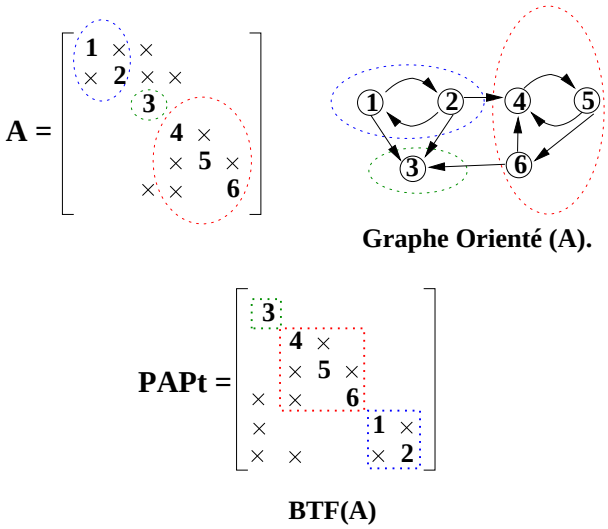


FIG. 3 – Illustration du lien entre composantes fortes du graphe et blocs diagonaux de la BTF

2 L'algorithme de Sargent et Westerberg

L'algorithme de Sargent et Westerberg part de l'observation suivante : *"une matrice triangulaire possède une forme particulière de BTF possédant des blocs diagonaux de taille 1"*. La forme BTF peut ainsi être considérée comme une généralisation de la forme triangulaire pour laquelle chaque entrée sur la diagonale correspondrait en fait à une composante forte du graphe.

L'algorithme de Sargent et Westerberg peut ainsi être vu comme une généralisation d'un algorithme de recherche de la forme triangulaire d'une matrice.

2.1 Préambule : Découvrir la forme triangulaire d'une matrice

On suppose dans cette section que la matrice $\mathbf{A}$ possède une forme triangulaire (il existe une permutation $\mathbf{P}$, en général non unique, telle que $\mathbf{PAP}^{\mathrm{T}}$ soit triangulaire).

On notera que le graphe orienté associé à la matrice $\mathbf{A}$ est sans cycle et qu'il existe un nœud ne possédant pas d'arc sortant. Ces deux observations constituent la base de l'algorithme de Sargent et Westerberg.

Algorithme 1 Forme triangulaire (Sargent et Westerberg)

-Sélectionner un nœud non encore visité et tracer un chemin jusqu'à atteindre un nœud sans arc sortant.
-Numéroter ce dernier nœud en premier (avant tous les autres nœuds non encore éliminés).
-Éliminer ce nœud (et tous les arcs pointant vers lui) du graphe.
-Continuer à partir du nœud précédent dans le chemin courant ou à partir d'un nouveau nœud de départ si le chemin est vide.

L'algorithme 1 est illustré sur la Figure 4. On pourra noter que la permutation sous forme triangulaire n'est pas unique.

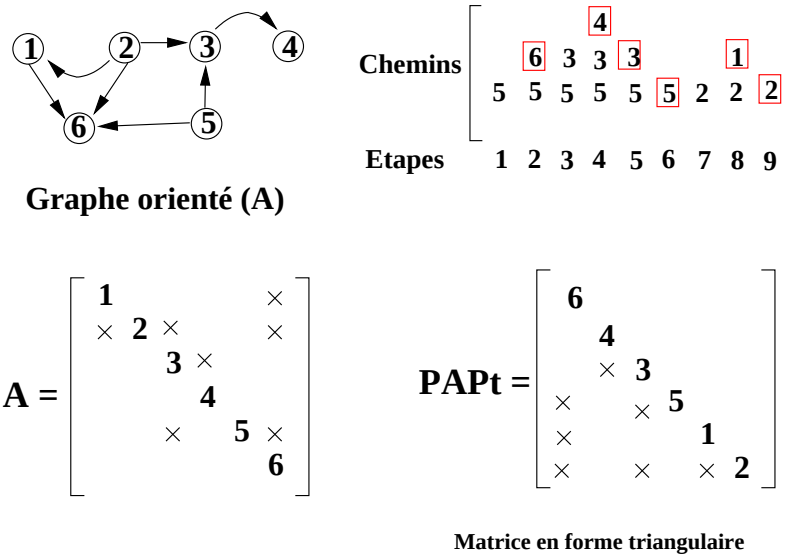


FIG. 4 – A chaque étape on indique le chemin courant. $\boxed{i}$ indique que le nœud i n'a plus d'arc sortant dans le graphe réduit à cette étape et sera donc éliminé.

2.2 Généralisation de l'algorihme de Sargent et Westerberg pour calculer la forme BTF d'une matrice

On notera **nœud composite** un ensemble de nœuds appartenant à un chemin fermé (cycle).

Algorithme 2 BTF (Sargent et Westerberg)

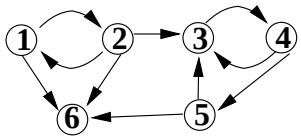
Sélectionner un nœud de départ.
Suivre un chemin jusqu'à :
(1) Trouver un chemin fermé (cycle) :
-agréger alors tous les nœuds du cycle en un nœud composite,
-mettre à jour le graphe
(le nœud composite remplace tous les nœuds du cycle,
-le chemin continue à partir de ce nœud (composite).

(2) Atteindre un nœud (composite) ne possédant plus d'arc sortant :
-ce nœud (composite) est alors le suivant dans la numérotation,
-ce nœud est alors supprimé du graphe et l'algorithme se poursuit à partir du nœud précédant
ou à partir d'un nouveau point de départ si le chemin est vide.

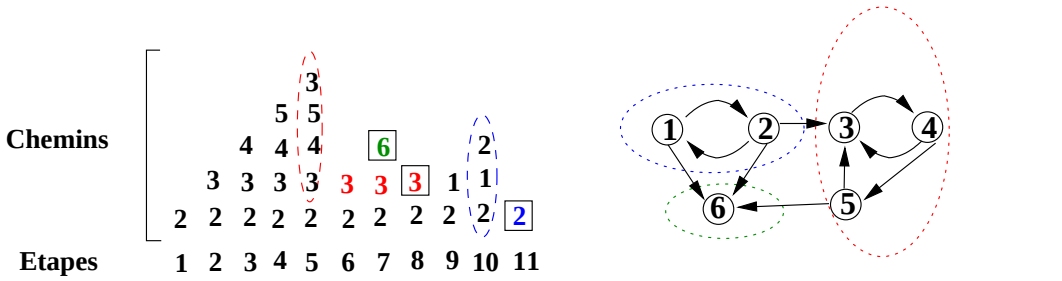
L'algorithme 2 est illustré sur la Figure 5. On notera la détection à l'étape 5 du cycle $3 \rightarrow 4 \rightarrow 5 \rightarrow 3$ conduisant à l'agrégation à l'étape 6 en un nœud composite numéroté ici 3. Le premier nœud éliminé est le nœud (simple) 6 à l'étape 7. A l'étape 8 le nœud composite 3 associé à $(3, 4, 5)$ est éliminé. Le nœud composite 2 associé à $(2, 1)$ est éliminé à l'étape 11. L'ordonnancement ainsi obtenu est donc $(6, 3, 4, 5, 2, 1)$

La Figure 6 montre le lien entre les nœuds composites du graphe orienté associés aux composantes fortes du graphe et les blocs diagonaux de la matrice au format BTF.

Une implantation normale de l'algorithme de Sargent et Westerberg conduit à un nombre de renumérotations (dus aux nœuds composites) pouvant être très élevé. Tarjan propose une variante de cet algorithme décrite dans la section suivante qui résout ce problème.



Graphe orienté (A)



Composantes fortes du graphe

FIG. 5 – Algorithme de Sargent et Westerberg pour le calcul d'une BTF.

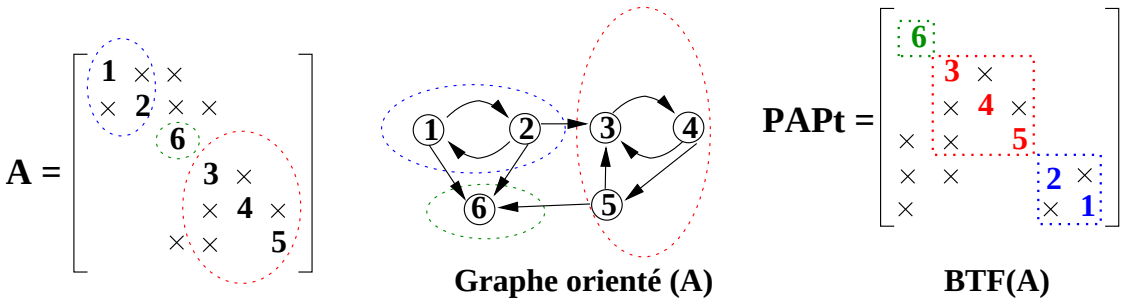
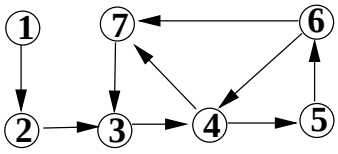


FIG. 6 – Illustration des composantes fortes du graphe associées aux blocs diagonaux de la matrice au format BTF.

3 L'algorithme de Tarjan

L'algorithme de Tarjan suit les mêmes idées que celui de Sargent et Westerberg dans le sens où il parcourt des chemins dans le graphe réduit et identifie les composantes fortes du graphe. Il utilise en plus de façon astucieuse un mécanisme de pile. Pour mieux comprendre l'algorithme nous allons dans un premier temps l'illustrer sur deux exemples. Nous le décrirons dans un cadre général ensuite.



Graphe orienté (A)

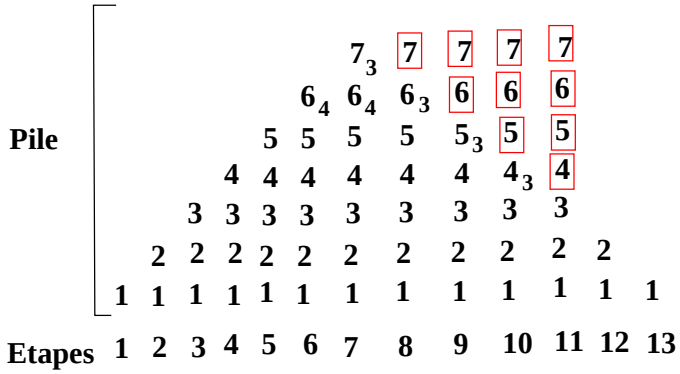
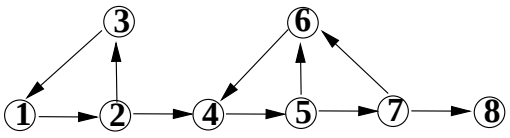


FIG. 7 – Exemple 1 : illustration du mécanisme de pile utilisé dans l'algorithme de Tarjan. Les nœuds ne faisant plus partie du chemin sont entourés d'une boite. Les indices correspondent à des liens vers un noeud d'un cycle.

La Figure 7 montre l'état de la pile à chaque étape de l'algorithme de Tarjan. Durant les six premières étapes la pile mémorise simplement le chemin $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$. A l'étape 6 une arête reliant le noeud 6 au noeud 4 plus bas dans la pile est trouvée. Comme, par construction il existe un chemin remontant la pile ($4 \rightarrow 5 \rightarrow 6$) nous avons donc détecté un cycle. Ce cycle est mémorisé à l'étape 6 en indiquant un lien représenté sur la figure par l'indice 4 adossé au noeud 6. De façon similaire à l'étape 7, l'arête $7 \rightarrow 3$ correspond à la fermeture du chemin $3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7$ et sera mémorisée par un lien vers 3 sur le noeud 7. A l'étape 8, comme il n'y a plus d'arête non visitée partant de 7, ce noeud est retiré du chemin de recherche mais il est gardé dans la pile car il fait partie du cycle $(3, 4, 5, 6, 7)$. Pour indiquer cela le noeud 7 est entouré d'une boite. Le noeud 6 qui précédait le noeud 7 se retrouve à la fin du chemin et il hérite du lien qu'avait 7 vers le noeud 3. A l'étape 9, on s'aperçoit qu'il n'y a pas non plus d'arc non visité partant de 6 et donc on supprime 6 du chemin mais on le garde dans la pile (comme pour 7). Le lien vers le noeud 3 se propage vers le noeud 5. De façon similaire on élimine, à l'étape 10, le noeud 5 du chemin. A l'étape 11 le noeud 4 est alors enlevé du chemin (il est inutile de propagé le lien vers 3 pour le noeud 3). A l'étape 12 on s'aperçoit qu'il n'y a pas non plus de chemin sortant du noeud 3. Comme ce noeud ne possède pas de lien plus bas dans la pile, il ne peut pas y

avoir d'arc entre lui (ou un noeud plus haut dans la pile) avec un noeud plus bas dans la pile. Les nœuds $(3, 4, 5, 6, 7)$ constituent une composante forte et peuvent donc être enlevés de la pile. Les composantes fortes réduites aux singletons 2 puis 1 sont ensuite détectées lors des étapes 12 puis 13.

On peut noter que l'exemple 1 a été choisi de façon à garantir que le chemin courant soit toujours composé de nœuds adjacents dans la pile. L'exemple 2 (voir Figure 8) relâche cette contrainte.



Graphe orienté (A)

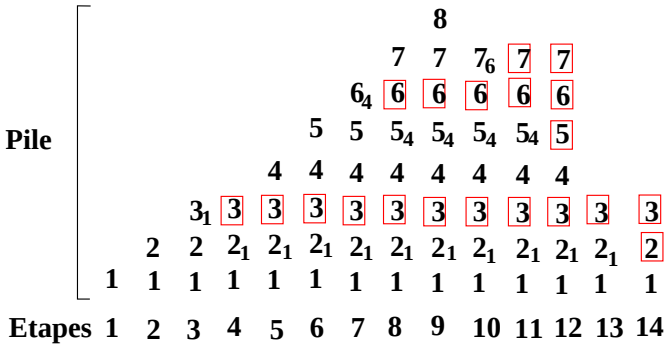


FIG. 8 – Exemple 2 : illustration du mécanisme de pile utilisé dans l'algorithme de Tarjan. Les nœuds ne faisant plus partie du chemin sont entourés d'une boite, les indices correspondent à des liens vers un noeud d'un cycle.

La Figure 8 illustre la gestion de la pile lorsque l'on démarre du noeud 1. A l'étape 3 le noeud 3 ainsi que le lien vers le noeud 1 sont ajoutés. A l'étape 4 le noeud 3 est enlevé (entouré d'une boite sur la figure) du chemin car il ne possède aucun arc sortant non visité. Le lien vers 1 est alors propagé vers le noeud précédent dans la pile qui constitue le nouveau dernier noeud du chemin courant. Le noeud 4 est alors ajouté à l'étape 5 à cause de l'arête $(2, 4)$ et le chemin courant est $1 \rightarrow 2 \rightarrow 4$. De façon similaire le noeud 7 est ajouté à l'étape 8 à cause de l'arête $(5, 7)$. A l'étape 10, le noeud 8 est éliminé à la fois de la pile et du chemin car il ne possède pas d'arc sortant ni de lien vers un noeud plus bas dans la pile. A l'étape 10, l'arête $(6, 7)$ conduit à l'ajout du lien vers le noeud 6 (indice ajouté au noeud 7). A l'étape 11, le noeud n'ayant plus d'arc sortant est éliminé du chemin mais est gardé dans la pile à cause du lien indiquant qu'il fait partie d'une composante forte. Ce lien est propagé vers 5, le précédent dans le chemin (notons que ce n'est pas 6 car 6 a déjà été éliminé à l'étape 8). 5 possédant déjà un lien vers un noeud plus bas dans la pile ce lien est donc conservé à l'étape 11. A l'étape 13, le noeud 4 n'ayant pas de lien sortant et ne possédant pas de lien, il peut être enlevé du chemin. Tous les nœuds au dessus du noeud 4 dans la pile à l'étape 12 constituent donc une composante forte et sont enlevés de la pile à l'étape 13. La composante forte $(1, 2, 3)$ est ensuite détectée de façon similaire à l'étape 14.

Fort de ces deux exemples nous pouvons maintenant décrire de façon générale l'algorithme de Tarjan en décrivant les étapes de mise à jour de la pile à chaque étape. On supposera pour simplifier la description de l'algorithme que chaque noeud pointe (notion de lien utilisé dans les exemples) par défaut vers lui même. A une étape donnée, l'algorithme cherche une arête non encore parcourue adjacente au dernier noeud du chemin courant (soit p_i ce noeud p d'indice i, i pouvant être égal à p, i étant sinon par construction toujours un noeud situé plus bas que p dans la pile). Notons que si le chemin courant est vide alors un noeud quelconque non encore traité est considéré. Les cas suivants peuvent alors se présenter :

1. L'arête (p_i, j) pointe vers un noeud j qui n'est pas dans la pile. j_j est alors ajouté en sommet de pile et au chemin courant (voir Figure 8, étape 6).
2. L'arête (p_i, j) pointe vers un noeud j plus bas dans la pile que i. Le lien est dans ce cas mis à jour pour pointer vers ce nouveau lien (p_i devient p_j) (voir Figure 8, étape 7).
3. L'arête (p_i, j) pointe vers un noeud j plus haut dans la pile que i ou déjà éliminé. Ne rien faire dans ce cas.
4. Il n'y a pas d'arête non parcourue adjacente à p_i et i est plus bas dans la pile que p. Dans ce cas p est laissé dans la pile mais est enlevé du chemin courant. Soit k_r le noeud k de lien r précédant p dans le chemin courant. Ce noeud existe et devient donc le dernier noeud du chemin. Le lien du noeud k devient soit r (si le noeud r est plus bas dans la pile que le nœud i), soit i (si le noeud i est plus bas dans la pile que le nœud r) (voir Figure 8, étape 8).
5. Il n'y a pas d'arête non parcourue adjacente à p_i et $i = p$. Dans ce cas le noeud p est enlevé du chemin courant. De plus p ainsi que tous les nœuds au dessus de lui dans la pile constituent une composante forte que l'on peut enlever de la pile (voir Figure 8, étape 13).

L'ordre dans lequel les nœuds sont enlevés de la pile décrit la permutation des indices conduisant à une forme BTF. L'algorithme permet aussi d'identifier les composantes fortes du graphe associées aux blocs B_{ii} de la matrice en BTF.

4 Description du travail à effectuer

Le travail se décompose en deux parties : une partie algorithmique et une partie développement, validation et expérimentation. A chacune sera associé un rendu.

4.1 Travail algorithmique (essentiellement sur l'algorithme de Tarjan)

1. Dérouler l'algorithme de Tarjan, en expliquant l'état de la pile à chaque étape (sur le modèle de la Section 3) en démarrant pour l'exemple 1 du nœud 3 et pour l'exemple 2 du nœud 2.
2. Construire quelques exemples simples supplémentaires sur lesquels vous illustrerez toutes les situations possibles de l'algorithme de Tarjan.
3. En respectant **impérativement** *les contraintes liées à l'interface utilisateur, à la représentation de la pile, et à la taille de l'espace de travail* (voir Section 4.3), décrire les structures de données que vous proposez d'utiliser dans votre algorithme. Décrire l'algorithme de Tarjan utilisant ces structures de données.
4. Vous commenterez les simplifications pouvant être effectuées sur la taille de l'espace de travail et sur l'algorithme lorsque l'on se limite à détecter la forme triangulaire d'une matrice.

5. Utiliser les exemples précédents pour illustrer l'utilisation des structures de données et leur évolution à chaque étape de l'algorithme. Ces exemples seront ensuite utilisés pour valider les procédures développées.

4.2 Développement / validations de codes / préparation oral

Les objectifs de cette partie sont indiqués si aprés.

1. Développements. Dans un premier temps de développer les deux subroutines (TRIANG et BTF) correspondant respectivement à la detection de la forme triangulaire d'une matrice (Algorithme 1) et à l'algorithme de Tarjan.
   Il est impératif de respecter l'interface proposée et les contraintes de complexité mémoire (taille maximum de l'espace mémoire utilisé) décrites en Section 4.3.
2. Documentation/Validation. Les codes seront commentés (un rappel algorithmique et un commentaire sur l'utilisation des tableaux de travail sera indiqué en debut de procedure), et validés.

   Le soin et la rigueur dans le développement des codes (commentaires de tous les codes fournis, clarté des codes rendus, et richesse des tests de validation) constituera une partie importante de l'évaluation. Les subroutines devront être robustes et traiter les erreurs possibles sur les données d'entrée. Les cas d'erreur détectés devront être documentés. Il faudra notamment vérifier que toutes les données dynamiques allouées dans les subroutines développées sont effectivement libérées (pas de fuite mémoire dans les subroutines fournies).

   Tous les programmes seront écrits en FORTRAN90. Il est demandé de fournir aussi les matrices de test et les codes de test. Les subroutines TRIANG et BTF doivent respecter strictement l'interface fournie. Votre jeu de matrices de test comportera au moins les matrices fournies dans le sujet et les matrices utilisées pour décrire le comportement de votre algorithme dans le rapport papier (voir Section 4.1).

   Le répertoire Src (voir README du répertoire) contient une version de départ du validateur qui permet de lire une matrice non symmétrique à un format SIMPLE décrit dans le répertoire MAT.simple. Ce programme permet alors de contruire le graphe associé et appelle la routine BTF. Le répertoire MAT.simple contient un jeu de matrices réelles de petites taille dont le format est décrit dans le fichier README du répertoire.
3. Préparation de l'oral. Chaque groupe préparera une présentation courte (3 planches maximum) pour l'oral. Ce fichier sera exporté au format pdf et rendu avec les codes commentés. Dans ces 3 planches, vous vous focaliserez dans un premier temps sur la description de **votre** (vous considérerez que l'algorithme général de Tarjan est connu) travail (environ 2 planches). Vous commenterez ensuite les performances (temps de calcul et mémoire utilisée) de votre code/approche algorithmique (1 planche).

4.3 Structures de données et implantation de l'algorithme de Tarjan

4.3.1 Représentation du graphe orienté

Pour implanter l'algorithme de Tarjan il vous est demandé de représenter le graphe G, associé à la matrice d'ordre N, comme un tableau de N cellules de type cell_graphe.

```
      type cell_graphe
         integer                    : : indice
         type (cell_graphe), pointer : : suivant
       END type cell_graphe
       type (cell_graphe),dimension(N) : : G
```

$\forall i \in [1, N]$ $G(i)\%suivant$ pointe sur le début de la liste d'adjacence de la ligne i. $G(i)\%indice$ contient le degré du nœud i (nombre d'arcs sortants dans le graphe orienté).

4.3.2 Interface des procédures à réaliser

```
       SUBROUTINE BTF(N, G, TRACE, PERM, NbCF, PTDebCF, IERREUR)
!   ------------------------------------------------
!   Algorithme de Tarjan :
!     Soit G le graphe orienté associé à une matrice
!     A non symétrique possédant des éléments non nuls
!     sur la diagonale.
!     L'algorithme calcule les composantes fortes du graphe
!     et renvoie la permutation permettant de mettre la
!     matrice A sous forme BTF.
! --------------------
! Paramètres d'entrée
! --------------------
!      N : Nombre de nœuds du graphe
!      G : Graphe orienté représenté comme un tableau de cellules.
!          Pour i, 1<=i<=N, liste pointée par la cellule G(i) correspond
!          aux indices de colonnes associés aux élément non nuls
!          dans la ligne i. L'élément diagonal de la matrice initiale
!          ne sera pas représenté G.
!          Le graphe G ne sera pas modifié.
!      trace : booléen. Positionné a vrai pour
!              afficher le contenu de la pile à chaque étape de l'algorithme.
       INTEGER, INTENT(in)                          : : N
       TYPE (cell_graphe), dimension(N), INTENT(IN) : : G
       LOGICAL, INTENT(IN)                          : : TRACE
!
! --------------------
! Paramètres en SORTIE
! --------------------
!      PERM    : Tableau d'ordre N d'entiers contenant la permutation :
!                PERM(i) = j si j est la iieme entrée de la matrice
!                                  permutée.
!      NbCF    : Nombre de composantes fortes détectées
!                (1 si la matrice associée est irréductible,
!                 N si la matrice associée est triangulaire)
!      PTDebCF : Tableau d'ordre N d'entiers tel que
!                Soit 1<= k <= NbCF alors PTDebCF(k) indique la position
!                dans le tableau PERM du premier indice de la
!                composante kieme composante forte du graphe.
!                (on a donc par définition toujours PTDebCF(1) = 1
!      IERREUR : Codage du retour d'erreur (0 correspond a pas d'erreur).
       INTEGER, DIMENSION(N), INTENT(OUT)   : : PERM
       INTEGER, INTENT(OUT)                 : : IERREUR
```

La subroutine TRIANG qui traite le cas particulier des matrices ayant une forme triangulaire aura la même interface que la subroutine BTF (même si une interface plus simple aurait pu être proposée).

4.3.3 Représentation de la pile et contraintes sur la taille de l'espace de travail

La complexité de l'espace de travail sera en O(N) ; Plus précisément, seulement un espace de travail de taille maximum $5 \times N$ données de type entier ou pointeur pourra être alloué. On pourra bien sur déclarer en plus des variables scalaires de type entier ou pointeur.

Pour représenter la pile de l'algorithme deux possibilités sont proposées :

1. Un tableau d'entiers de taille N. A ce tableau on pourra alors associer au maximum 4 autres tableaux de taille maximum N d'entiers ou de pointeurs.
2. Une pile permettant d'empiler une structure (contenant k champs) qu'il vous faudra définir. On pourra ajouter à cette pile s tableaux d'ordre maximum N d'entiers ou de pointeurs. La contrainte devant être respectée est alors : $k + s \leq 5$.

5 Dates importantes

– Le travail demandé en section 4.1 sera rendu sous forme d'un document au plus tard le **Vendredi 3 Avril 09** dans les boites aux lettres du département prévues à cet effet.
– Le travail demandé en section 4.2 sera rendu sous forme d'un fichier tar contenant l'ensemble de vos codes (BTF, TRIANG et les programmes de test) ainsi que les matrices et programmes de test utilisées. Le fichier au format pdf (voir section 4.2) qui vous servira pour présenter votre travail. Ce fichier tar sera déposé au plus tard le **Jeudi 7 Mai 09**.

Références

[1] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, London, 1986.

[2] I. S. Duff and J. K. Reid. An implementation of tarjan's algorithm for the block triangularizarion of a matrix. *ACM Transactions on Mathematical Software*, 4 :137–147, 1978.

[3] A. Pothen and C. J. Fan. Computing the block triangular form of a sparse matrix. *ACM Transactions on Mathematical Software*, 16 :303–324, 1990.