Chapitre 8

RÉSOLUTION DE SYSTÈMES LINÉAIRES

1 Introduction

La résolution de systèmes d'équations linéaires est peut-être le problème le plus connu et le mieux étudié de l'analyse numérique. Ces études s'appuient sur l'ensemble des connaissances en algèbre linéaire. Malgré ces atouts, la résolution pratique d'un système linéaire peut poser des difficultés, soit pour certains choix des coefficients (systèmes «mal conditionnés»), soit lorsque le nombre d'inconnues devient grand (plus de quelques milliers), comme c'est le cas lors de la résolution d'équations aux dérivées partielles ou dans certains domaines de l'économétrie. Il peut aussi arriver que le système soit «surdéterminé» (plus d'équations que d'inconnues).

Quelles sont les méthodes que propose l'arsenal mathématique ? Tout d'abord une méthode que je qualifierai de formelle : la solution du problème

$$\mathbf{Ax} = \mathbf{b} \tag{1}$$

(où $\mathbf{A}$ est une matrice $n \times n$, $\mathbf{b}$ et $\mathbf{x}$ des vecteurs à n coordonnées) s'écrit $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$, puis la méthode de Cramer, qui se révèle inutilisable en pratique dès qu'il y a plus de trois inconnues, (le calcul d'un déterminant $n \times n$ nécessite $n!$ opérations, ce qui est prohibitif). Je rencontre ensuite un groupe de méthodes qui procèdent par éliminations successives des inconnues : elles sont associées aux noms de Gauss et de Gauss-Jordan et sont d'application aisée. D'autres algorithmes «factorisent» la matrice des coefficients ($\mathbf{A} \Longrightarrow \mathbf{FG}$) avant de pratiquer des éliminations successives qui sont alors très rapides à cause de la forme particulière des facteurs bfF, G : ce sont les méthodes de Crout, Doolittle ou Cholesky. En dépit des apparences, ces deux ensembles sont mathématiquement équivalents.

Dans un esprit bien différent, je pourrai faire appel à des procédés itératifs de résolution, dont la convergence n'est pas toujours assurée : ce sont les méthodes associées aux noms de Jacobi et Gauss-Seidel.

La méthode des gradients conjugués (non abordée ici) est exacte, mais peut-être interrompue en cours de calcul.

Comme vous le verrez, le calcul de l'inverse $\mathbf{A}^{-1}$d'une matrice $\mathbf{A}$ est plus long et moins stable que le calcul direct de la solution du système linéaire (1). Une remarque analogue s'applique en algèbre. Pour résoudre $10x = 3$, je peux faire $10^{-1} = 0,1; 0,1 \times 3 = 0,3$ ou $x = 3/10 = 0.3$. On évite donc de calculer l'inverse de la matrice des coefficients, à moins que le problème posé ne le demande explicitement. Si tel est le cas, je calculerai $\mathbf{A}^{-1}$ en résolvant n systèmes linéaires particuliers, dont les seconds membres sont respectivement égaux à chacun des vecteurs de base.

Dans la mise en oeuvre des méthodes d'élimination ou de factorisation, l'ordinateur ne commet que des erreurs d'arrondi. Dans le cas d'une méthode itérative, je devrais tenir compte aussi d'une erreur de troncation, puisque l'algorithme s'arrête au bout d'un nombre fini d'opérations.

Pour l'analyse numérique, la difficulté de résolution d'un système linéaire tient aux effets conjugués de sa taille et de son «conditionnement». Voici une interprétation graphique de cet anglicisme. Soit le système linéaire :

$$\begin{aligned} ax + by &= m, \\ cx + dy &= n. \end{aligned}$$

Je divise la première équation par un $\sqrt{a^2+b^2}$ et la seconde par $\sqrt{c^2+d^2}$, pour obtenir

$$n_{1x}x + n_{1y}y = d_1, n_{2x}x + n_{2y}y = d_2.$$

Les coefficients satisfaisant maintenant aux relations $n_{ix}^2 + n_{iy}^2 = 1$, $i = 1, 2$, les vecteurs $\mathbf{n}_i$ de coordonnées $\{n_{ix}, n_{iy}\}$ sont unitaires.

Le système linéaire ainsi écrit admet l'interprétation géométrique suivante. Chaque équation est celle d'une droite D_i du plan (x, y). La droite D_i est perpendiculaire au vecteur $\mathbf{n}_i$; la distance de la droite à l'origine est d_i. La solution cherchée est représentée par le point d'intersection des deux droites. Dans les cours de mathématiques, on distingue trois cas : D_1confondue avec D_2 (système indéterminé), D_1 parallèle à D_2 (système impossible), et le cas normal où D_1 coupe D_2.

Numériquement, il faut distinguer ici deux possibilités. Si D_1 et D_2 font entre elles un angle notable, le point d'intersection est bien défini et la solution du système sera facile à trouver. Si D_1 et D_2 se coupent sous un angle très faible, le système est dit «mal conditionné». Dans ce cas, la moindre variation de l'un des d_i (les seconds membres) ou de l'un des $n_{\alpha,i}$ (les coefficients, $\alpha = x, y$) entraînera un déplacement très important du point d'intersection. Autrement dit, la solution est extrêmement sensible aux perturbations des coefficients ou des seconds membres. Un déterminant très petit est une indication qu'un système est mal conditionné ; ce n'est pas une condition nécessaire surtout lorsque le nombre de variables devient grand.

Je peux donner un aspect plus quantitatif aux considérations précédentes. J'admets qu'il existe une norme vectorielle $\| \mathbf{x} \|$ et une norme matricielle $\| \mathbf{A} \|$, cohérente avec la précédente (c'est-à-dire que, pour tout $\mathbf{x}$, $\| \mathbf{A} \| \cdot \| \mathbf{x} \| \geq \| \mathbf{Ax} \|$). En plus du système (1), je considère un système «perturbé» ; le second membre a subi une petite modification $\Delta\mathbf{b}$, ce qui a entraîné une petite variation $\Delta\mathbf{x}$ de la solution

$$\mathbf{A}(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{b} + \Delta\mathbf{b}$$

d'où $\Delta\mathbf{x} = \mathbf{A}^{-1}\Delta\mathbf{b}$ et :

$$\| \Delta\mathbf{x} \| \leq \| \mathbf{A}^{-1} \| \| \Delta\mathbf{b} \| .$$

J'en tire une majoration de la variation relative :

$$\frac{\| \Delta\mathbf{x} \|}{\| \mathbf{x} \|} = \| \mathbf{A} \| \| \mathbf{A}^{-1} \| \frac{\| \Delta\mathbf{b} \|}{\| \mathbf{b} \|} = cond(\mathbf{A}) \frac{\| \Delta\mathbf{b} \|}{\| \mathbf{b} \|}.$$

Le nombre $cond(\mathbf{A}) = \| \mathbf{A} \| \| \mathbf{A}^{-1} \|$ est une mesure de la sensibilité de l'erreur relative (sur la solution) aux variations du second membre.

Je montre maintenant que $cond(\mathbf{A})$ représente aussi la sensibilité aux variations des coefficients des premiers membres. Je considère un nouveau système perturbé

$$(\mathbf{A} + \delta\mathbf{A})(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b}.$$

En retranchant membre à membre (1), je trouve

$$\delta\mathbf{x} = -\mathbf{A}^{-1}\delta\mathbf{A}(\mathbf{x} + \delta\mathbf{x}),$$

ce qui implique l'inégalité

$$\| \delta\mathbf{x} \| \leq \| \mathbf{A}^{-1} \| \| \delta\mathbf{A} \| \| (\mathbf{x} + \delta\mathbf{x}) \|,$$

soit encore

$$\frac{\| \delta\mathbf{x} \|}{\| \mathbf{x} + \delta\mathbf{x} \|} = cond(\mathbf{A}) \frac{\| \delta\mathbf{A} \|}{\| \mathbf{A} \|}.$$

2 Méthode de Gauss

2.1 Exemple

Je vais traiter un exemple qui, mieux qu'un long discours, fera saisir l'esprit de la méthode d'élimination de Gauss et son équivalence avec une factorisation de la matrice des coefficients. Il s'agit du système

$$\mathbf{Ax} = \begin{bmatrix} 2 & 1 & 0 & 4 \\ -4 & -2 & 3 & -7 \\ 4 & 1 & -2 & 8 \\ 0 & -3 & -12 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 2 \\ -9 \\ 2 \\ 2 \end{bmatrix} = \mathbf{x}. \tag{2}$$

Je vais procéder par éliminations successives en utilisant trois théorèmes bien connus d'algèbre linéaire. La solution du système linéaire reste inchangée si :

* on multiplie une équation (une ligne de $\mathbf{A}$ et la coordonnée correspondante de $\mathbf{b}$) par une constante ;

* on ajoute une équation à une autre (une ligne de $\mathbf{A}$ à une autre et la coordonnée correspondante de $\mathbf{b}$ à une autre) ;

* on permute deux équations (deux lignes de $\mathbf{A}$ et les coordonnées correspondantes de $\mathbf{b}$).

Je peux traiter $\mathbf{A}$ et $\mathbf{b}$ en parallèle ou construire une matrice $\mathbf{A}'$ "augmentée" dont la $n+1$-ième colonne est identique à $\mathbf{b}$.

Je commence par éliminer x_1. Pour cela :
je multiplie la ligne (1) par 2 et je l'ajoute à la ligne (2),
je multiplie la ligne (1) par -2 et je l'ajoute à la ligne (3).
Il vient :

$$\mathbf{A}^{(1)} = \begin{bmatrix} 2 & 1 & 0 & 4 \\ 0 & 0 & 3 & 1 \\ 0 & -1 & -2 & 0 \\ 0 & -3 & -12 & -1 \end{bmatrix} \quad ; \quad b^{(1)} = \begin{bmatrix} 2 \\ -5 \\ -2 \\ 2 \end{bmatrix}$$

On remarque que le multiplicateur relatif à la ligne i est $-a_{i,1}/a_{11}$; a_{11} est appelé le "pivot" ; il a intérêt à n'être pas nul ! Beaucoup d'auteurs emploient la convention opposée : ils multiplient la ligne (1) par $+a_{i,1}/a_{11}$ et la retranchent de la ligne (i) Le résultat est donné par la matrice $\mathbf{A}^{(1)}$ et le vecteur $\mathbf{b}^{(1)}$. Je dois maintenant éliminer x_2 ; problème : le pivot correspondant ($a_{22}^{(1)}$) est nul ! Mon élan parait devoir être brisé près du départ. Mais, heureusement, je peux permuter les lignes (2) et (3) ; le nouveau pivot est maintenant -1 ; j'ajoute -3 fois (2) à (4) pour trouver :

$$\mathbf{A}^{(3)} = \begin{bmatrix} 2 & 1 & 0 & 4 \\ 0 & -1 & -2 & 0 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & -6 & -1 \end{bmatrix} \quad ; \quad \mathbf{b}^{(3)} = \begin{bmatrix} 2 \\ -2 \\ 5 \\ 8 \end{bmatrix}$$

Finalement, en utilisant le pivot 3, j'ajoute 2 fois (3) à (4) :

$$\mathbf{A}^{(4)} = \begin{bmatrix} 2 & 1 & 0 & 4 \\ 0 & -1 & -2 & 0 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} ; \mathbf{b}^{(4)} = \begin{bmatrix} 2 \\ -2 \\ -5 \\ -2 \end{bmatrix}$$

La matrice $\mathbf{A}^{(4)}$, équivalente à $\mathbf{A}$, a la forme « triangulaire supérieure ». Elle rassemble les coefficients d'un système triangulaire facile à résoudre par substitution à partir de la dernière ligne :

$$\begin{array}{lll} (4) & \Rightarrow & x_4 = -2, \\ (3) & \Rightarrow & 3x_3 - 2 = -5 \qquad\quad \Rightarrow x_3 = -1, \\ (2) & \Rightarrow & -x_2 - 2(-1) = -2 \quad \Rightarrow x_2 = 4, \\ (1) & \Rightarrow & 2x_1 + 4 + 4(-2) = 2 \quad \Rightarrow x_1 = 3. \end{array}$$

Je gagne en prime la valeur de $\det \mathbf{A} = (-1)^p \prod_i a_{ii}^{(4)}$, où p est le nombre de permutations de lignes, ici 1, d'où $\det \mathbf{A} = 6$. Si l'un des éléments diagonaux de $\mathbf{A}^{(4)}$ (l'un des pivots de l'élimination de Gauss) est nul, le déterminant est nul ; réciproquement, si $\det \mathbf{A} \neq 0$, je suis assurés de trouver des pivots non nuls (quitte à permuter des lignes).

2.2 Gauss-Jordan

Pour ceux qui auraient pris goût à l'élimination, je propose une variante. Plutôt que de substituer dans le système triangulaire, je poursuis l'élimination au-dessus de la diagonale principale : c'est la méthode de Gauss-Jordan. Pour changer, je travaille sur la matrice augmentée $\mathbf{A}'$, réunion de $\mathbf{A}$ et de $\mathbf{b}$. J'ajoute (2) à (1) :

$$\mathbf{A}'^{(5)} = \begin{bmatrix} 2 & 0 & -2 & 4 & 0 \\ 0 & -1 & -2 & 0 & -2 \\ 0 & 0 & 3 & 1 & -5 \\ 0 & 0 & 0 & 1 & -2 \end{bmatrix}$$

puis 2/3 de (3) à (1) et à (2) :

$$\mathbf{A}'^{(6)} = \begin{bmatrix} 2 & 0 & 0 & 14/3 & -20/3 \\ 0 & -1 & 0 & 2/3 & -16/3 \\ 0 & 0 & 3 & 1 & -5 \\ 0 & 0 & 0 & 1 & -2 \end{bmatrix}$$

Pour finir, j'ajoute -14/3 (4) à (1), -2/3 (4) à (2) et -(4) à (3) ; il vient :

$$\mathbf{A}'^{(8)} = \begin{bmatrix} 2 & 0 & 0 & 0 & 6 \\ 0 & -1 & 0 & 0 & -2 \\ 0 & 0 & 3 & 0 & -3 \\ 0 & 0 & 0 & 1 & -2 \end{bmatrix}$$

écriture matricielle d'un système dont la résolution est immédiate.

2.3 Représentation matricielle de l'élimination

Multiplier la coordonnée 1 d'un vecteur par le coefficient m, puis ajouter le résultat à la coordonnée 2 revient à multiplier ce même vecteur par une matrice particulière :

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ m & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ mb_1 + b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

Plus généralement, pour ajouter m fois la ligne i à la ligne j, il suffit de disposer le nombre m à la place de l'élément j, i de la matrice unité ($i < j$). Ce qui est vrai pour un vecteur l'est également pour une matrice. La prémultiplication de $\mathbf{A}$ carrée quelconque par la matrice précédente revient à multiplier successivement chaque colonne de $\mathbf{A}$, donc à ajouter la première ligne de $\mathbf{A}$ à la deuxième. Encore un exemple ; je veux ajouter -3 fois la 3-ième ligne à la 4-ième de la matrice générale : -3 est en ligne 4, colonne 3 :

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -3 & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} =$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ -3a_{31} + a_{41} & -3a_{32} + a_{42} & -3a_{33} + a_{43} & -3a_{34} + a_{44} \end{bmatrix}$$

Les matrices qui viennent d'être définies (dites matrices de Frobenius) jouissent de propriétés intéressantes. Si $\mathbf{f}(i, j, m)$ désigne la matrice de Frobenius contenant m en ligne i et colonne j, alors $\mathbf{f}^{-1} = \mathbf{f}(i, j, -m)$: soustraire m fois la ligne i de la ligne j est bien l'opération inverse d'ajouter m fois (i) à (j). $\mathbf{f}(i, j, m)\mathbf{f}(i', j, m') = \mathbf{f}(i', j, m')\mathbf{f}(i, j, m)$ contient m et m' en colonne j, aux lignes i et i', ce qui représente bien le fait qu'ajouter un multiple de la ligne j aux lignes i et i' sont des actions indépendantes.

Dans l'exemple précédent, j'ai obtenu la matrice $\mathbf{A}^{(2)}$ à partir de la matrice $\mathbf{A}^{(1)} = \mathbf{A}$, en utilisant deux multiplicateurs (2 et -2), ce qui peut s'écrire :

$$\mathbf{A}^{(2)} = \mathbf{M}_1\mathbf{A}^{(1)}; \mathbf{b}^{(2)} = \mathbf{M}_1\mathbf{b}^{(1)},$$

avec

$$\mathrm{M}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ -2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Je pose, en vue d'une utilisation future, $\mathbf{L}_1 \equiv \mathbf{M}_1^{-1}$. Revenant à l'exemple, j'ai ensuite permuté les lignes (2) et (3) pour faire apparaître un pivot non nul, ce que je représente par une prémultiplication par la matrice de permutation $\mathbf{P}$:

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Cette matrice est symétrique et orthogonale : $\mathbf{P}^{-1} = \mathbf{P}^T = \mathbf{P}$ et, par conséquent, $\mathbf{P}^2 = \mathbf{I}$, ce qui traduit le fait qu'échanger deux fois les mêmes lignes revient à ne rien faire. J'ai ensuite éliminé x_2 avec l'aide implicite de la matrice $\mathbf{M}_2$.

$$\mathbf{M}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -3 & 0 & 1 \end{bmatrix}$$

Je définis au passage son inverse, noté $\mathbf{L}_2 \equiv \mathbf{M}_2^{-1}$.

Le système linéaire s'écrit maintenant : $\mathbf{A}^{(3)}x = \mathbf{b}^{(3)}$, avec $\mathbf{A}^{(3)} = \mathbf{M}_2\mathbf{P}\mathbf{A}^{(2)}$ et $\mathbf{b}^{(3)} = \mathbf{M}_2\mathbf{P}b^{(1)}$. Pour éliminer x_3, j'ai utilisé $\mathbf{M}_3$:

$$\mathbf{M}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 1 \end{bmatrix}$$

La dernière forme du système s'écrit : $\mathbf{A}^{(4)}\mathbf{x} = \mathbf{b}^{(4)}$. Je pose encore $\mathbf{A}^{(3)} = \mathbf{L}_3\mathbf{A}^{(4)}$ et $\mathbf{b}^{(3)} = \mathbf{L}_3\mathbf{b}^{(4)}$.

En résumé, en ce qui concerne le tableau des coefficients, j'ai

$$\mathbf{A}^{(4)} = \mathbf{M}_3\mathbf{A}^{(3)} = \mathbf{M}_3\mathbf{M}_2\mathbf{P}\mathbf{A}^{(2)} = \mathbf{M}_3\mathbf{M}_2\mathbf{P}\mathbf{M}_1\mathbf{A}^{(1)}$$

Le vecteur $\mathbf{b}$ a subi les mêmes transformations, résumées par :

$$\mathbf{b}^{(4)} = \mathbf{M}_3\mathbf{M}_2\mathbf{P}\mathbf{M}_1\mathbf{b}^{(1)}.$$

Ces formules résument bien la «triangularisation» du système, mais c'est la relation inverse qui est vraiment intéressante. Pour le voir, j'introduis les notations $\mathbf{U} = \mathbf{A}^{(4)}$ et $\mathbf{c} = \mathbf{b}^{(4)}$ (le système linéaire est donc équivalent à : $\mathbf{U}\mathbf{x} = \mathbf{c}$). Je calcule ensuite

$$\mathbf{A} = \mathbf{A}^{(1)} = \mathbf{L}_1\mathbf{P}\mathbf{L}_2\mathbf{L}_3\mathbf{A}^{(4)} = \mathbf{L}_1\mathbf{P}\mathbf{L}_2\mathbf{L}_3\mathbf{U}.$$

Si je pose $\hat{\mathbf{L}} \equiv \mathbf{L}_1\mathbf{P}\mathbf{L}_2\mathbf{L}_3$, je mets la matrice de départ sous la forme $\mathbf{A} = \hat{\mathbf{L}}\mathbf{U}$: c'est un produit de facteurs, dont l'un est triangulaire supérieur ($\mathbf{U}$) ; malheureusement, à cause du facteur $\mathbf{P}$, la matrice $\hat{\mathbf{L}}$ n'a pas de forme simple. Pour obtenir un résultat plus élégant, je commence par définir $\mathbf{L}_1' \equiv \mathbf{P}\mathbf{L}_1\mathbf{P}^T$; cette nouvelle matrice s'obtient en permutant les lignes (2) et (3) de $\mathbf{L}_1$ (prémultiplication par $\mathbf{P}$) et en permutant les colonnes (2) et (3) de la matrice obtenue (postmultiplication par $\mathbf{P}^T$; le résultat s'écrit :

$$\mathbf{L}_1' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ -2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Comme vous le constatez, $\mathbf{L}_1'$ est à nouveau triangulaire inférieure : la perturbation apportée par $\mathbf{P}$ a été «absorbée» dans $\mathbf{L}_1'$. Je définis ensuite $\mathbf{L} = \mathbf{L}_1'\mathbf{L}_2\mathbf{L}_3$:

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ -2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 3 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -2 & 1 \end{bmatrix}$$

Le calcul explicite donne

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ -2 & 0 & 1 & 0 \\ 0 & 3 & -2 & 1 \end{bmatrix}$$

Il suffit de recopier à leurs places dans $\mathbf{L}$ les éléments sous la diagonale des facteurs $\mathbf{L}'_1, \mathbf{L}_2$ et $\mathbf{L}_3$, comme pour les matrices élémentaires de Frobenius. Je forme ensuite le produit $\mathbf{LU}$:

$$\mathbf{LU} = \mathbf{L}'_1\mathbf{L}_2\mathbf{L}_3\mathbf{A}^{(4)} = \mathbf{L}'_1\mathbf{L}_2\mathbf{A}^{(3)} = \mathbf{L}'_1\mathbf{P}\mathbf{A}^{(2)} = \mathbf{P}\mathbf{L}_1\mathbf{A}^{(2)} = \mathbf{P}\mathbf{A}^{(1)} = \mathbf{PA}.$$

La matrice $\mathbf{PA}$ a été décomposée en un produit de deux facteurs, l'un ($\mathbf{L}$, lower) triangulaire inférieur, avec des éléments diagonaux égaux à un, l'autre ($\mathbf{U}$, upper) triangulaire supérieur. Vous pouvez aussi effectuer numériquement le produit $\mathbf{LU}$ pour vous convaincre du résultat (matrice de départ sauf permutation des lignes (2) et (3)).

Pour trouver la solution de (1), il suffit maintenant de résoudre successivement deux systèmes triangulaires. D'abord :

$$\mathbf{L}'\mathbf{y} = \mathbf{Pb},$$

dont la solution est le vecteur $\mathbf{y}$, puis :

$$\mathbf{Ux} = \mathbf{y}$$

dont la solution est le vecteur $\mathbf{x}$ cherché. Vous pourrez vérifier que $\mathbf{y} = \mathbf{b}^{(4)}$.

D'autre part, j'ai toujours sous la main le déterminant de $\mathbf{A}$:

$$(-1)^p \det(\mathbf{A}) = \det(\mathbf{PA}) = \det(\mathbf{LU}) = \det(\mathbf{L})\det(\mathbf{U}) = \prod_i u_{ii}.$$

2.4 Généralisation

L'exemple précédent se généralise sans difficulté au cas où $\mathbf{A}$ est une matrice carré, régulière, $n \times n$. On peut démontrer l'existence et l'unicité de la factorisation. Je me contente de démontrer, par l'absurde, cette dernière propriété. Supposons qu'il existe deux décompositions d'une même matrice régulière :

$$\mathbf{A} = \mathbf{L}_1\mathbf{U}_1 = \mathbf{L}_2\mathbf{U}_2,$$

d'où :

$$\mathbf{X} = \mathbf{L}_2^{-1}\mathbf{L}_1 = \mathbf{U}_2\mathbf{U}_1^{-1}$$

Or, l'inverse d'une matrice triangulaire inférieure (supérieure) est elle-même triangulaire inférieure (supérieure). Le produit de deux matrices triangulaires inférieures est une matrice de même forme ; de même, le produit de deux matrices triangulaires supérieures est encore triangulaire supérieur. $\mathbf{X}$, qui doit réunir ces deux propriétés, ne peut être que la matrice unité, ce qui entraîne $\mathbf{L}_1 = \mathbf{L}_2$ et $\mathbf{U}_1 = \mathbf{U}_2$.

2.5 Mise en oeuvre pratique

Si la matrice $\mathbf{A}$ est régulière, je suis assuré de toujours pouvoir trouver un pivot non nul : l'élimination de Gauss est toujours possible en théorie. On montre qu'elle est encore utile pour des matrices de rang $< n$ ou pour des matrices rectangulaires. Je n'aborderais pas ces points ici.

Les calculs réels sont en fait entachés d'erreurs d'arrondi. Ceci a pour conséquence que le résultat final dépendra fortement du choix des pivots (ou encore de l'ordre des variables éliminées). On peut voir en gros ce qui se passe. Je suppose que les variables $x_1, x_2, \ldots, x_{k-1}$ ont été éliminées. Pour éliminer l'inconnue x_k, je multiplie la ligne k par a_{ik}/a_{kk}. Si le pivot a_{kk} est petit, toutes les erreurs qui affectent les coefficients a_{ik} seront amplifiées. En conséquence, on adopte toujours la stratégie de «recherche du pivot partiel» ; pour éliminer la variable x_k et quelque soit la valeur de $|a_{kk}|$, on recherche l'élément de plus grand module dans la colonne k et c'est lui que l'on prendra comme pivot effectif, après avoir fait la permutation de lignes nécessaire.

Certains auteurs recommandent la stratégie de la recherche complète du pivot maximal («pivot total»). Pour éliminer x_k, on cherche pour $i \geq k$ et pour $j \geq k$ l'élément a_{ij} de plus grand module. On l'amène en position de pivot par des permutations de lignes **et** de colonnes. Permuter les colonnes n'est pas neutre, c'est équivalent à permuter les inconnues. Il faut donc garder trace des ces permutations pour restaurer l'ordre initial à la fin. La programmation est un peu plus compliquée. Il semble par ailleurs que l'analyse précise des erreurs d'arrondi ne démontre pas une supériorité nette de la recherche globale sur la recherche partielle.

Les erreurs d'arrondi sont plus faibles si les éléments des matrices successives restent à peu près du même ordre de grandeur. Pour cela, on peut, avant de résoudre le système, multiplier certaines lignes par des facteurs tels que le plus grand coefficient de chaque ligne soit compris entre 0,5 et 1 (équilibrage).

Le décompte détaillé des opérations arithmétiques nécessaires fait apparaître le résultat suivant :

factorisation	$n(n-1/2)(n-1)/3$	additions et multiplications
	$n(n-1)/2$	divisions
substitution	$n(n-1)/2$	additions et multiplications
	n	divisions

Vous retiendrez les valeurs asymptotiques : pour l'élimination : $O(n^3/3)$ opérations et pour la résolution du système linéaire : $O(n^2)$ opérations.

L'algorithme de Gauss est assez économe en mémoire, si l'on accepte de détruire $\mathbf{A}$. En effet, à chaque étape, je range les éléments de $\mathbf{L}$ à la place que devraient occuper les zéros de $\mathbf{A}$ après élimination (dans le triangle inférieur). Les 1 de la diagonale de $\mathbf{L}$ sont sous-entendus. Le triangle supérieur de $\mathbf{A}$ est progressivement occupé par les éléments de $\mathbf{U}$.

2.6 Calcul de l'inverse de A

Pour certaines applications, il faut calculer $\mathbf{A}^{-1} = \mathbf{X}$ explicitement, soit encore trouver $\mathbf{X}$ telle que $\mathbf{AX} = \mathbf{I}$. La colonne j de $\mathbf{X}$ est solution du système linéaire :

$$\mathbf{A}\mathbf{x}_j = \mathbf{e}_j.$$

Je vais avoir à résoudre n systèmes linéaires dont les premiers membres sont identiques, mais les seconds membres différents. Il est commode d'utiliser la décomposition $\mathbf{PA} = \mathbf{LU}$ effectuée une seule fois, puis de résoudre les systèmes triangulaires $\mathbf{L}\mathbf{y}_j = \mathbf{P}e_j$ puis $\mathbf{U}\mathbf{x}_j = \mathbf{y}_j$. On sait que les coordonnées de $\mathbf{y}_j$ sont nulles jusqu'à un certain rang, ce qui permet de gagner un peu de temps. Le nombre d'opérations nécessaires est $O(n^3)$, pas plus que pour calculer $\mathbf{A}^2$ mais trois fois plus que pour résoudre un système linéaire de même n.

3 Factorisation directe

Je sais que la matrice régulière **A** admet une représentation sous forme d'un produit de matrices triangulaires **LU**. Je peux alors me poser la question suivante : est-il possible de trouver les éléments de **L** et **U** sans utiliser l'algorithme de Gauss ? La réponse est affirmative et c'est même assez simple. Cependant, comme la décomposition **LU** est unique, ce nouvel algorithme est certainement équivalent à celui de Gauss.

Soient $\ell_{i,j}$ les éléments de **L**, avec $\ell_{i,j} = 0$ si $i < j$; de même, les éléments de **U** sont tels que $u_{i,j} = 0$ si $i > j$; enfin $\ell_{i,i} = 1$. Je procéderai par identification.

$$\begin{aligned} a_{11} &= \sum \ell_{1k} u_{k1} = \ell_{11} u_{11} = u_{11}, \\ a_{12} &= \sum \ell_{1k} u_{k2} = u_{12}, \\ &\cdots\cdots\cdots\cdots\cdots\cdots \\ a_{1n} &= \sum \ell_{1k} u_{kn} = u_{1n}. \end{aligned}$$

En utilisant la première ligne de **A**, j'ai trouvé la première ligne de **U**. L'astuce pour continuer (due à Crout, d'où le nom de l'algorithme), consiste à identifier maintenant la première colonne de **A**.

$$\begin{aligned} a_{21} = \sum \ell_{2k} u_{k1} = \ell_{21} u_{11} + \ell_{22} u_{21} &\Rightarrow \ell_{21} = a_{21}/u_{11} \\ a_{31} = \sum \ell_{3k} u_{k1} = \ell_{31} u_{11} + \ell_{32} u_{21} + \ell_{33} u_{31} &\Rightarrow \ell_{31} = a_{31}/u_{11} \\ &\cdots\cdots\cdots\cdots \end{aligned}$$

Je détermine ainsi la première colonne de **L**. Vous pouvez imaginer que la deuxième ligne de **U** s'obtient à partir de la deuxième ligne de **A**, puis que la deuxième colonne de **L** découle de la deuxième colonne de **A**, etc... Les formules générales s'écrivent :

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} \ell_{ik} u_{kj}; j = i, i+1, ...n;$$

$$\ell_{i,j} = \frac{1}{u_{jj}} \left[a_{ij} - \sum_{k=1}^{j-1} \ell_{ik} u_{kj} \right]; i = j+1, j+2, ...n \tag{3}$$

$$\tag{4}$$

avec toujours $\ell_{ii} = 1$. Il y a une petite difficulté : cet algorithme est très sensible aux erreurs d'arrondis, et ne fonctionnera pas sans recherche du pivot maximal dans chaque colonne. D'autre part, en général, je forme la décomposition **LU** pour résoudre un système linéaire ; ici, je n'ai permuté que les lignes de **A** et non celles de **b** (qui n'apparaît pas) ; il faut donc garder trace des permutations pour pouvoir retrouver les inconnues par la suite.

On rencontre fréquemment deux cas particuliers. La premier est celui d'une matrice définie positive, le second celui d'une matrice tridiagonale. Je vais les examiner brièvement.

3.1 Matrice tridiagonale

Soit **A** une matrice tridiagonale ; Je pose $a_{i,i-1} = x_i$, $a_{i,i} = y_i$ et $a_{i,i+1} = z_i$. J'admets en plus que **A** possède une décomposition **LU** sans permutation de lignes. Par identification, on vérifie que **L** et **U** sont bidiagonales :

$$\begin{bmatrix} y_1 & z_1 & 0 & \dots & \dots & 0 \\ x_2 & y_2 & z_2 & 0 & \dots & 0 \\ 0 & \dots & \dots & \dots & \dots & 0 \\ 0 & \dots & 0 & x_{n-1} & y_{n-1} & z_{n-1} \\ 0 & \dots & \dots & 0 & x_n & y_n \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ l_2 & 1 & 0 & \dots & 0 \\ 0 & \dots & 1 & \dots & 0 \\ 0 & \dots & l_{n-1} & 1 & 0 \\ 0 & \dots & 0 & l_n & 1 \end{bmatrix} \begin{bmatrix} d_1 & u_1 & 0 & \dots & 0 \\ 0 & d_2 & u_2 & \dots & 0 \\ 0 & \dots & \dots & \dots & 0 \\ 0 & \dots & 0 & d_{n-1} & u_{n-1} \\ 0 & \dots & 0 & 0 & d_n \end{bmatrix}$$

Vous prouverez sans peine les relations suivantes :

$$y_1 = d_1 \quad ; \quad ; z_1 = u_1 \quad ; \quad x_i = l_i d_{i-1};$$

$$y_i = l_i u_{i-1} + d_i \quad ; \quad ; z_i = u_i.$$

Vous pourrez finalement établir les formules très simples de résolution d'un système linéaire dont las coefficients seraient les x_i, y_i, z_i. Dans le cas où l'on doit pratiquer une recherche du pivot partiel, le pivot de l'étape k se trouve nécessairement dans l'une des lignes k ou $k+1$ (matrice tridiagonale). Il en résulte une décomposition $\mathbf{PA} = \mathbf{LU}$, où **L** est toujours bidiagonale inférieure, alors que **U** est tridiagonale supérieure.

3.2 Matrice symétrique définie positive

Si **A** est symétrique, sa décomposition s'écrit : $\mathbf{A} = \mathbf{L}\mathbf{L}^T$. Je suppose en plus que **A** est définie positive. Je rappelle deux définitions équivalentes : $\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$ quelque soit le vecteur $\mathbf{x}$; toutes les sous-matrices principales ont des déterminants positifs). On montre alors que les pivots de l'élimination de Gauss sont strictement positifs et donc que la factorisation $\mathbf{L}\mathbf{L}^T$ se fait sans permutation (décomposition régulière de Cholesky). On détermine les éléments de **L** ($\ell_{ij} = 0 si j > i$) par identification :

$$\begin{aligned} \ell_{11}^2 &= a_{11} > 0, \\ \ell_{11}\ell_{21} &= a_{21}, \\ \ell_{21}^2 + \ell_{22}^2 &= a_{22} \\ &\dots\dots\dots\dots \\ \ell_{i,j} &= \frac{1}{\ell_{jj}} \left[a_{ij} - \sum_{k=1}^{j-1} \ell_{ik}\ell_{jk} \right], \\ \ell_{i,j} &= \frac{1}{\ell_{jj}} \left[a_{ij} - \sum_{k=1}^{j-1} \ell_{ik}\ell_{jk} \right] \end{aligned}$$

3.3 Variantes

J'ai montré que la matrice régulière $\mathbf{A}$ admet la factorisation $\mathbf{LU}$ en un produit de matrices triangulaires, inférieure pour $\mathbf{L}$ à diagonale unitaire, supérieure pour $\mathbf{U}$. Cette répartition des rôles n'est pas la seule possible. On peut concevoir la décomposition $\mathbf{A} = \mathbf{L'U'}$ où $\mathbf{U'}$ est maintenant triangulaire à diagonale unitaire, $\mathbf{L'}$ étant simplement triangulaire ; un algorithme du à Doolittle et très voisin des précédents permet d'obtenir $\mathbf{L'}$ et $\mathbf{U'}$.

Plutôt que de favoriser l'un des facteurs grâce à la possession d'une diagonale unitaire, on peut imposer que chaque facteur triangulaire présente des éléments diagonaux égaux à 1. On a alors :

$$\mathbf{A} = \mathbf{LDU} \quad , \quad \ell_{ii} = u_{ii} = 1$$

où $\mathbf{D}$ est une matrice diagonale. Dans le cas particulier où $\mathbf{A}$ est symétrique, il vient :

$$\mathbf{A} = \mathbf{LDL}^T$$

4 Méthodes itératives de résolution des systèmes linéaires

Les très grands systèmes linéaires posent un problème pratique : il peut être impossible de caser en mémoire centrale l'ensemble des coefficients. D'autre part, on rencontre, surtout lorsque le système linéaire provient de la discrétisation d'une équation aux dérivées partielles, une structure particulière des coefficients : les éléments diagonaux sont plus grands que tous les autres. Dans ces cas là, on peut utiliser avec profit des méthodes itératives de résolution, beaucoup moins gourmandes en mémoire et plus rapides.

4.1 Méthode de Jacobi

Je cherche la solution de (1), avec $a_{ii} \neq 0$, $1 \leq i \leq n$, en connaissant une approximation de la solution : $\mathbf{x}^{[0]}$. Je vais utiliser une méthode d'approximation successive en traitant de façon équivalente chaque coordonnée de $\mathbf{x}$:

$$x_i^{[k]} = \frac{1}{a_{ii}}(b_i - \sum_{j \neq i} a_{ij} x_j^{[k-1]}) \quad ; \quad 1 \leq i \leq n. \tag{5}$$

Le calcul de $x_i^{[k]}$ n'utilise que la ligne i de $\mathbf{A}$: cette matrice peut donc être rangée sur disque et lue à la demande.

Comme pour toute méthode d'itération, je dois me préoccuper d'un critère d'arrêt. On rencontre couramment l'une des deux définitions classiques suivantes. Je définis le «résiduel» à l'itération k :

$$\mathbf{r}^{[k]} = \mathbf{b} - \mathbf{A}\mathbf{x}^{[k]}.$$

Je peux choisir la condition de convergence

$$\| \mathbf{r}^{[k]} \| \leq \epsilon \| \mathbf{b} \|$$

, ϵ étant un seuil fixé à l'avance ou la condition :

$$\| \mathbf{x}^{[k]} - \mathbf{x}^{[k-1]} \| \leq \epsilon \| \mathbf{x}^{[k-1]} \| .$$

L'algorithme (5) admet une représentation matricielle commode pour les analyses théoriques. Je décompose **A** en une somme de trois matrices (rien à voir avec les **L** et **U** des paragraphes précédents) :

$$\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U}$$

où **D** est diagonale, **E** strictement triangulaire inférieure et **F** strictement triangulaire supérieure (j'ai donc $a_{ii} = d_{ii}), \ell_{ii} = u_{ii} = 0$). J'introduis le vecteur $\delta\mathbf{x}^{[k]} \equiv \mathbf{x}^{[k+1]} - \mathbf{x}^{[k]}$. En utilisant les relations de récurrence et la définition du vecteur résiduel, vous montrerez facilement que :

$$\delta\mathbf{x}^{[k]} = \mathbf{D}^{-1}\mathbf{r}^{[k]}$$

d'où :

$$\mathbf{x}^{[k+1]} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x}^{[k]} + \mathbf{D}^{-1}\mathbf{b}.$$

Exemple. Étant donné le système linéaire ci-dessous et le vecteur initial $[0, 0]^T$:

$$\begin{aligned} 3x + y &= 2 \\ -x + 2y &= -2 \end{aligned}$$

ou

$$\begin{aligned} x &= (1/3)(2 - y) \\ y &= \frac{1}{2}(x - 2) \end{aligned}$$

je trouve la suite de solutions approchées suivante :

x	$2/3$	1	$8/9$	$5/6$	$23/27$	$31/36$	$139/162$
y	-1	$-2/3$	$-1/2$	$-5/9$	$-7/12$	$-31/54$	$-41/72$

qui converge effectivement mais lentement vers la solution exacte $[6/7, -4/7]^T$.

La méthode de Jacobi converge lentement, mais je peux facilement modifier cet algorithme pour le rendre plus rapide.

4.3. Méthode de Gauss-Seidel

L'algorithme de Jacobi me prescrit de calculer $x_i^{[k+1]}$ en utilisant les valeurs $x_1^{[k]}, x_2^{[k]}, \ldots, x_{i-1}^{[k]}, x_{i+1}^{[k]}, ...x_n^{[k]}$; il semble que je pourrais gagner du temps ou de la précision, puisque je connais déjà $x_1^{[k+1]}, x_2^{[k+1]}, \ldots, x_{i-1}^{[k+1]}$, qui sont plus précises que les valeurs des mêmes variables à l'itération précédente (k). C'est ce que fait l'algorithme de Gauss-Seidel. Les relations de récurrence s'écrivent :

$$x_i^{k+1} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{[k+1]} - \sum_{j=i+1}^{n} a_{ij} x_j^{[k]} \right] \tag{6}$$

Ces équations sont aussi plus faciles à programmer que celles qui découlent de la méthode de Jacobi. En effet, il n'est pas nécessaire de conserver l'ensemble des valeurs $x_i^{[k]}$ pendant le calcul des $x_i^{[k+1]}$: chaque nouvelle valeur remplace la précédente.

Je vous laisse le soin de montrer que l'équivalent matriciel de ces formules est :

$$\mathbf{x}^{[k+1]} = (\mathbf{D} - \mathbf{L})^{-1}\mathbf{U}\mathbf{x}^{[k]} + (\mathbf{D} - \mathbf{L})^{-1}\mathbf{b}.$$

Exemple. Appliquant l'algorithme de Gauss-Seidel à l'exemple du paragraphe précédent, je trouve la suite de valeurs :

x	2/3	8/9	23/27	139/162
y	-2/3	-5/9	-31/54	-185/324

Certains de ces nombres sont apparus lors du calcul précédent ; on observe toutefois que la convergence est beaucoup plus rapide.

4.4. Méthode de surrelaxation

Cette variante de l'algorithme de Gauss-Seidel (elle peut utiliser Jacobi aussi, mais sans avantage) procède comme suit. Connaissant un vecteur $\mathbf{x}^{[k]}$, je calcule un vecteur provisoire $\hat{\mathbf{x}}^{[k+1]}$ à l'aide de l'algorithme (6), puis je définis le vecteur $\mathbf{x}^{[k+1]}$ définitif comme :

$$\mathbf{x}^{[k+1]} = \omega\hat{\mathbf{x}}^{[k+1]} + (1-\omega)\mathbf{x}^{[k]}. \tag{7}$$

ω est un paramètre réel dont il faut optimiser la valeur pour chaque classe de problème. Le choix $\omega = 1$ me ramène à (6). On a toujours $\omega < 2$, et l'expérience montre que la meilleure valeur de ω est voisine de 1,7.

5 Un peu de programmation

Il est frustrant d'écrire un beau programme d'inversion de matrice, par exemple, et de devoir recompiler le programme pour chaque valeur de l'ordre de la matrice ; en général, les programmeurs tournent la difficulté en déclarant une taille de matrice très supérieure aux besoins prévisibles, d'où un encombrement de la mémoire qui peut devenir gênant. En Pascal, il est possible de déclarer des tableaux, arguments de sous-programmes, sans spécifier à l'avance leur taille. C'est évidemment dangereux, car si l'on s'affranchit du strict contrôle de taille effectué par le compilateur, on travaille sans filet. Je propose ci-dessous deux possibilités (tirées de la notice Borland). Dans ces ébauches de programme, les instructions d'affichage des résultats ont été omises.

```
PROGRAM dim_var_1;

{$R-}
PROCEDURE vext(n: INTEGER; VAR y; VAR ymin, ymax, dy: REAL);
TYPE tmin = ARRAY[0..0] OF REAL;
VAR i: INTEGER;
BEGIN
   ymin := tmin(y)[0]; ymax := tmin(y)[0];
   FOR i := 1 TO n{\-}1 DO BEGIN
      IF tmin(y)[i] < ymin THEN ymin := tmin(y)[i];
      IF tmin(y)[i] $>$ ymax THEN ymax := tmin(y)[i];
   END;
   dy := ymax {\-} ymin;
END;
{$R+}

CONST n = 128;
TYPE tab = ARRAY[1..n] OF REAL;
VAR x, max, min, delta: REAL; vec: tab; i: INTEGER;
BEGIN
   FOR i := 1 TO n DO
```

```
      vec[i] := x := 100.0*RANDOM - 50.0;
   END;
   WRITELN;
   vext(n,vec,max,min,delta);
END.

PROGRAM dim_var_2;

{$R-}
PROCEDURE vext(n: INTEGER; VAR y; VAR ymin, ymax, dy: REAL);
TYPE tmin = ARRAY[0..0] OF REAL;
VAR i: INTEGER; ya: tmin ABSOLUTE y;
BEGIN
   ymin := ya[0]; ymax := ya[0];
   FOR i := 1 TO n{\-}1 DO BEGIN
      IF ya[i] $<$ ymin THEN ymin := ya[i];
      IF ya[i] $>$ ymax THEN ymax := ya[i];
   END;
   dy := ymax - ymin;
END;
{$R+}

CONST n = 128;
TYPE tab = ARRAY[1..n] OF REAL;
VAR x, max, min, delta: REAL;
vec: tab; i: INTEGER;
BEGIN
   FOR i := 1 TO n DO
   vec[i] := 100.0*RANDOM - 50.0;
   vext(n,vec,max,min,delta);
END.
```

Chaque programme remplit un vecteur de nombres aléatoires, puis la procédure `vext` trouve les valeurs extrêmes et leur différence. Elle est programmée pour accueillir un argument (`y`) de taille quelconque ; en contrepartie, le contrôle de portée des indices est désactivé (commutateur de compilation `{$R-}` ; il est réactivé à la fin de la procédure (`{$R+}`). Le premier exemple utilise le transtypage pour affecter les coordonnées de `y` à celle de `tmin` (`ymin := tmin(y)[i]`). Le second utilise l'instruction `ABSOLUTE` pour parvenir au même résultat (le premier élément de `ya` coïncide avec le premier élément de `y` et tous les autres se correspondent deux à deux). Les deux procédés peuvent être employés simultanément, comme ci-dessous.

```
PROGRAM dim_var_3;

{$R-}
PROCEDURE vext(n: INTEGER; VAR y; VAR ymin, ymax, dy: REAL);
TYPE tmin = ARRAY[0..0] OF REAL;
VAR i: INTEGER; ya: tmin ABSOLUTE y;
BEGIN
```

```
   ymin := ya[0]; ymax := tmin(y)[0];
   FOR i := 1 TO n-1 DO BEGIN
      IF ya[i] < ymin THEN ymin := tmin(y)[i];
      IF ya[i] > ymax THEN ymax := ya[i];
   END;
   dy := ymax - ymin;
END;
{$R+}

CONST n = 128;
TYPE tab = ARRAY[1..n] OF REAL;
VAR x, max, min, delta: REAL;
vec: tab; i: INTEGER;
BEGIN
   FOR i := 1 TO n DO
   vec[i] := 100.0*RANDOM - 50.0;
   vext(n,vec,max,min,delta);
END.
```

En Pascal une structure (tableau, enregistrement) ne peut pas excéder 64500 octets ; de plus, le nombre de structures de grande taille utilisables simultanément est limité. Ces contraintes proviennent de ce que Pascal range les variables dans une zone de mémoire appelée la «pile» («stack»), dont la taille est de 64500 octets. On peut manipuler simultanément plusieurs grandes matrices en utilisant le «tas» («heap»), accessible uniquement par l'intermédiaire de pointeurs. Dans l'exemple suivant, j'emploie des pointeurs sur des tableaux de pointeurs pour effectuer un calcul bidon mettant en oeuvre trois matrices 100 x 100.

```
program grosse_m;
CONST max_l = 100; max_c = 100;
TYPE
   p{_reel = ^REAL; p_mat = ^mat;
   mat = ARRAY[1..max_l,1..max_c] OF p_reel;
VAR
   pa,pb,pc:p_mat;
   i,j,k: INTEGER;
   s: REAL;
BEGIN
   NEW(pa);NEW(pb);NEW(pc);
   FOR i := 1 TO max_l DO BEGIN
      FOR j := 1 TO max_c DO BEGIN
         NEW(pa^[i,j]); pa^[i,j]^ := 1/(i+j-1);
         NEW(pb^[i,j]); pb^[i,j]^ := pa^[i,j]^;
      END;
      WRITE(i:4);
   END;
   FOR i := 1 TO max_l DO BEGIN
      FOR j := 1 TO max_c DO BEGIN
         s := 0;
```

```
         FOR k := 1 TO max_l DO s := s + pa^[i,k]^*pb^[k,j]^;
         NEW(pc^[i,j]); pc^[i,j]^ := s;
      END;
      WRITE(I:4);
   END;
   FOR i := 1 TO max_l DO BEGIN
      FOR j := 1 TO max_l DO BEGIN
         WRITE(pc^[i,j]^:10:4); IF ((j+1) mod 8 = 0) THEN WRITELN;
      END;
      WRITELN; WRITELN;
   END;
END.
```

Cependant, un réel étant représenté par 6 octets, les matrices précédentes occupent chacune 60 000 octets, soit presque la taille maximale.