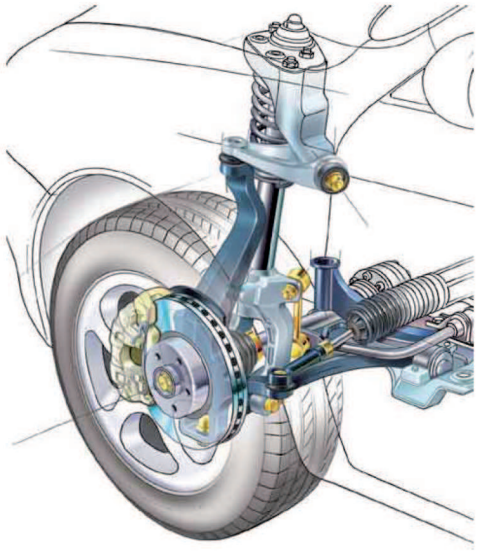
Info 16

Système linéaire d'équations : méthode du pivot de Gauss

Ce chapitre a pour objectif la résolution d'un système linéaire de n équations à p inconnues, grâce à la méthode du pivot de Gauss. On s'intéressera exclusivement au cas des systèmes de Cramer, correspondant au cas d'un système de n équations à n inconnues possédant une et une seule solution.

1. Mise en situation

Lorsque l'on cherche à déterminer les actions mécaniques s'exerçant sur un système matériel en équilibre, il est d'usage d'appliquer le principe fondamental de la statique (PFS). Cela conduit alors à établir un système d'équations linéaire. Il existe de nombreux logiciels permettant de résoudre ce genre de problème.



On se propose ici de programmer la méthode de Gauss de résolution d'un système linéaire de n équations à n inconnues.

$$(S)\begin{cases} a_{11}x_1 + \ldots + a_{1n}x_n = b_1 \\ \ldots \\ a_{n1}x_1 + \ldots + a_{nn}x_n = b_n \end{cases} \qquad n \in \mathbb{N}$$

où

- (a_{ij}) sont les coefficients du système
- $(b_i) \in \mathbb{R}^n$ sont les seconds membres du système
- $(x_i) \in \mathbb{R}^n$ sont les inconnues du système

Rq La méthode s'applique aussi dans $\mathbb{C}^n$

2. Méthode du pivot de Gauss

2.1. Résolution d'un système triangulaire

Les systèmes triangulaires sont très simples à résoudre :

$$\begin{cases} 2x + 2y - 3z = 2 \\ y - 6z = -3 \\ z = 4 \end{cases} \Leftrightarrow \begin{cases} 2x + 2y - 3z = 2 \\ y = 6z - 3 = 21 \\ z = 4 \end{cases} \Leftrightarrow \begin{cases} x = \frac{1}{2}(-2y + 3z + 2) = -14 \\ y = 21 \\ z = 4 \end{cases}$$

Lors de la résolution du système, on raisonne toujours par équivalence. C'est une condition indispensable pour garantir l'unicité de la solution trouvée.

La première étape de résolution d'un système consiste à le mettre sous forme triangulaire en gardant l'équivalence avec le système initial.

La deuxième étape de résolution du système correspond à la phase de remontée du système triangulaire : on résout les équations de bas en haut à partir de l'avant-dernière ligne, en substituant aux inconnues les valeurs obtenues dans les lignes inférieures.

2.2. Vocabulaire et généralités

On considère le système linéaire (S) : $\begin{cases} a_{11}x_1 + \ldots + a_{1n}x_n = b_1 \\ \ldots \\ a_{n1}x_1 + \ldots + a_{nn}x_n = b_n \end{cases}$ $n \in \mathbb{N}$

On définit alors la matrice A du système, le vecteur inconnu X et le second membre B :

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix} \in M_n(\mathbb{R}), \quad X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in M_{n,1}(\mathbb{R}) \quad \text{et} \quad B = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \in M_{n,1}(\mathbb{R}).$$

L'écriture matricielle du système est alors :

$$A.X = B$$

Définition

La matrice augmentée du système (S) est la matrice

$$(A|B) = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} & b_1 \\ a_{21} & a_{22} & \ldots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} & b_n \end{pmatrix} \in M_{n+1}(\mathbb{R}),$$

Les lignes L_i $(1 \leq i \leq n)$ de la matrice augmentée $(A|B)$ sont celles du système (S).
Les colonnes C_j $(1 \leq j \leq n+1)$ de la matrice augmentée $(A|B)$ sont celles du système (S).
La dernière colonne de la matrice augmentée $(A|B)$ correspond au vecteur B : $C_{n+1} = B$.

2.3. Inversibilité

Afin de simplifier la mise en œuvre de la méthode du pivot de Gauss, on fait l'hypothèse que la matrice A est inversible. Le système (S) a alors une unique solution :

$$X = A^{-1}B.$$

On ne souhaite pas vérifier si la matrice A est inversible ou non. On pourra donc prévoir une interruption de programme avec un renvoi de message d'erreur s'il s'avère que la matrice A n'est pas inversible (on précisera dans quel cas on rencontre ce problème).

2.4. Opérations sur les lignes du système

Pour se ramener, à partir d'un système initial, à un système triangulaire, on ajoute à une ligne donnée une combinaison linéaire des autres lignes, de telle sorte que l'on fasse disparaître des inconnues dans le bon ordre.

Les opérations élémentaires sur les lignes du sytème (S) sont :

- ◇ les échanges de lignes $L_i \leftrightarrow L_j$
- ◇ la multiplication d'une ligne par une constante non nulle λ : $L_i \leftarrow \lambda L_i$
- ◇ l'ajout de λL_j à une ligne L_i : $L_i \leftarrow L_i + \lambda L_j$

Tout système (S') obtenu à partir de (S) après un nombre fini d'opérations élémentaires admet les mêmes solutions que le système (S). Les systèmes (S') et (S) sont équivalents.

3. Algorithme du pivot de Gauss-Jordan

L'algorithme du pivot de Gauss-Jordan permet de résoudre le système (S) par une suite finie d'opérations élémentaires sur les lignes. Il procède en deux étapes principales :

- ◇ La première qui consiste à échelonner le système c'est-à-dire le rendre triangulaire.
- ◇ La deuxième qui consiste à réduire le système.

3.1. Echelonnement du système (descente)

- ◇ **Première étape** :
  - — On suppose que le premier coefficient de la première ligne a_{11} est non nul. Ce premier coefficient s'appelle le premier pivot.
  - — On effectue des opérations $L_i \leftarrow L_i + \lambda L_j$ pour $2 \leq i \leq n$ de manière à faire disparaître x_1 des équations 2 à n.
- ◇ **Deuxième étape** : on recommence l'étape précédente avec les lignes 2 à n : on cherche un pivot pour la deuxième ligne et on cherche à faire disparaître x_2 des lignes 3 à n.
- ◇ **Etapes 3 à n** : on réitère le processus jusqu'à la n^{ieme} étape.

Rq

- Si la matrice A est inversible, il est toujours possible à l'étape i de trouver un pivot sur la i^{ieme} colonne dans les lignes L_i à L_n.
- Si la matrice A n'est pas inversible : on obtient une équation du type :

$$0x_1 + 0x_2 + \cdots + 0x_n = b$$

  - — si $b = 0$: on perd une équation (matrice de rang $n-1$). Le système n'a pas de solution unique.
  - — si $b \neq 0$: le système n'admet pas de solution.

A l'issue de ces opérations, on aboutit à un système échelonné à n équations et n inconnues :

$$\left\{\begin{array}{ccccccccc} a'_{11}x_1 & + & a'_{12}x_2 & + & \dots & + & a'_{1n}x_n & = & b'_1 \\ 0 & + & a'_{22}x_2 & + & \dots & + & a'_{2n}x_n & = & b'_2 \\ \vdots & + & & + & & + & \vdots & = & \vdots \\ \vdots & + & \vdots & + & \ddots & + & a'_{in}x_n & = & b'_i \\ \vdots & + & & + & & + & \vdots & = & \vdots \\ 0 & + & \dots & + & 0 & + & a'_{nn}x_n & = & b'_n \end{array}\right. \qquad n \in \mathbb{N}$$

Exemple

On peut remplacer L_2 par $L_2 - \frac{a_{21}}{a_{11}}L_1$ et L_3 par $L_3 - \frac{a_{31}}{a_{11}}L_1$ pour obtenir un système équivalent au premier dans lequel x n'apparaît plus dans les deux dernières équations.

On recommence ensuite pour le deuxième pivot.
Ainsi, à la première étape, on laisse la première équation inchangée. A la deuxième étape, on laisse les deux premières équations inchangées, et on procéderait de même sur un système de Cramer comportant davantage d'équations.

$$\left\{\begin{array}{rcl} 2x+2y-3z & = & 2 \\ -2x-y-3z & = & -5 \\ 6x+4y+4z & = & 16 \end{array}\right. \xrightleftharpoons[L_3 \leftarrow L_3-3L_1]{L_2 \leftarrow L_2+L_1} \left\{\begin{array}{rcl} 2x+2y-3z & = & 2 \\ y-6z & = & -3 \\ -2y+13z & = & 10 \end{array}\right. \xrightleftharpoons{L_3 \leftarrow L_3+2L_2} \left\{\begin{array}{rcl} 2x+2y-3z & = & 2 \\ y-6z & = & -3 \\ z & = & 4 \end{array}\right.$$

Nous avons étudié ici le cas le plus favorable. Toutefois, il peut arriver pour un système de Cramer que le pivot ne soit pas au bon endroit. Il se peut par exemple que le coefficient en x de la première ligne soit nul ($a_{11} = 0$). Dans ce cas, il faut, pour appliquer la méthode précédente, échanger la première équation avec l'une des deux suivantes (dans l'une des deux au moins, le coefficient devant x est non nul). A la seconde étape, si le coefficient en y est nul dans la seconde équation, il faut permuter celle-ci avec la troisième (mais, bien sûr, laisser la première en place !).

Rq

Une des questions cruciales qui se pose est le test de nullité d'un coefficient. Si cette opération peut paraître évidente au mathématicien, elle s'avère complexe pour l'informaticien. En effet, dès lors que le système a des solutions non entières, on est obligé de représenter les nombres sous la forme de flottants. Cela pose immanquablement la difficulté de comparer à zéro ce type de données (voir paragraphe 3.3).

3.2. Réduction d'un système échelonné : remontée

On dispose du système échelonné :

$$\left\{\begin{array}{ccccccccc} a'_{11}x_1 & + & a'_{12}x_2 & + & \dots & + & a'_{1n}x_n & = & b'_1 \\ 0 & + & a'_{22}x_2 & + & \dots & + & a'_{2n}x_n & = & b'_2 \\ \vdots & + & & + & & + & \vdots & = & \vdots \\ \vdots & + & \vdots & + & \ddots & + & a'_{in}x_n & = & b'_i \\ \vdots & + & & + & & + & \vdots & = & \vdots \\ 0 & + & \dots & + & 0 & + & a'_{nn}x_n & = & b'_n \end{array}\right.$$

◇ On multiplie chaque ligne i par $\dfrac{1}{a'_{ii}}$ pour chaque pivot a'_{ii}. Chaque pivot est alors égal à 1.

$$\left\{\begin{array}{ccccccccc} x_1 & + & \frac{a'_{12}}{a'_{11}}x_2 & + \ldots + & \frac{a'_{1n}}{a'_{11}}x_n & = & \frac{b'_1}{a'_{11}} \\ 0 & + & x_2 & + \ldots + & \frac{a'_{2n}}{a'_{22}}x_n & = & \frac{b'_2}{a'_{22}} \\ \vdots & + & & + \quad + & \vdots & = & \vdots \\ \vdots & + & \vdots & + \ddots + & \frac{a'_{in}}{a'_{ii}}x_n & = & \frac{b'_i}{a'_{ii}} \\ \vdots & + & & + \quad + & \vdots & = & \vdots \\ 0 & + & \ldots & + \ 0 \ + & x_n & = & \frac{b'_n}{a'_{nn}} \end{array}\right.$$

◇ La dernière ligne donne alors la valeur de x_n. On utilise le pivot valant 1 sur la dernière ligne et la dernière colonne pour éliminer x_n des équations (L_{n-1}) à (L_1) :

$$\text{pour } i \neq n \qquad L_i \leftarrow L_i + \lambda_{in} L_n \qquad \text{avec} \qquad \lambda_{in} = -\frac{a'_{in}}{a'_{ii}}$$

La ligne (L_{n-1}) donne alors la valeur de x_{n-1}.

◇ On recommence alors l'étape précédente sur les $n-1$ premières lignes, et ainsi de suite jusqu'à la ligne L_1. On obtient ainsi un système échelonné réduit de la forme :

$$(S) \quad \left\{\begin{array}{ll} x_1 = b''_1 & \\ \vdots & n \in \mathbb{N} \\ x_n = b''_n & \end{array}\right.$$

3.3. Erreurs d'arrondis et problème de la comparaison à zéro

Nous avons décrit les 2 étapes permettant de résoudre un système linéaire d'équations. Il nous reste maintenant à prendre en compte la spécificité d'une résolution numérique due à la représentation des nombres.

La représentation d'un nombre réel sous la forme d'une grandeur de type `float` conduit à une erreur (hormis les nombres qui peuvent s'écrire sous la forme $\frac{N}{2^k}$ où N et k sont des entiers).

Ces approximations successives (le bruit de calcul) conduisent à des imprécisions sur les résultats numériques des calculs.

Python
```
>>>12*(1./3-1./4)-1
        -2.220446049250313e-16
```

En pratique, lors de la résolution d'un système, cela peut avoir deux conséquences néfastes :

◇ Le coefficient de l'une des inconnues, dans une équation, est non nul alors qu'il devrait l'être. Si cela se produit, il se peut que cet élément soit alors pris comme pivot, ce qui conduit à un résultat du système complètement faux (puisqu'on multipliera une ou plusieurs lignes par un nombre flottant très grand, ce qui fera perdre tout sens au système).

◇ Inversement, dans des cas plus rares, un coefficient qui ne devrait pas être nul apparaîtra comme tel.

Exemple

On peut ainsi mettre en évidence facilement des systèmes qui ne sont pas de Cramer, pour lesquels Python donne un résultat numérique alors qu'ils n'ont pas de solution :

$$\begin{cases} x + 1/4y + z = 0 \\ x + 1/3y + 2z = 0 \\ y + 12z = 1 \end{cases}$$

On peut aussi illustrer l'incidence du choix du pivot sur les erreurs d'arrondis en étudiant l'exemple suivant :

Exemple

$$\begin{cases} 10^{-4}\,x + y = 1 \\ x + y = 2 \end{cases}$$

- ◇ Solution : $x = 1,0001$ et $y = 0,9999$.
- ◇ Résolution en virgule flottante avec 3 chiffres significatifs en considérant $a_{11} = 10^{-4}$ comme pivot et compte tenu des erreurs d'arrondi donne

$$0,999.10^3\, y = 0,999.10^3 \qquad \text{donc} \qquad y = 1$$

En reportant le résultat dans (L_1) donne $x = 0$.

Ainsi considérer des petits pivots peut engendrer des erreurs d'arrondi qui lors des additions et soustractions provoquent des catastrophes.

On dit qu'un algorithme est numériquement instable si les erreurs d'arrondis sont considérablement grossies par l'application de l'algorithme.

Pour remédier à ce problème, la méthode la plus simple est la méthode du **pivot partiel**. Cette méthode n'est pas infaillible (on peut construire des systèmes qui la mettent en défaut) mais résout dans la plupart des cas la difficulté évoquée ci-dessus.

Il s'agit, à chaque fois que l'on choisit un coefficient pour pivot, de prendre le coefficient le plus grand parmi ceux disponibles. Cela revient à considérer, parmi toutes les lignes inférieures à la ligne courante, quelle ligne présente le coefficient le plus élevé (en valeur absolue) devant la variable que l'on cherche à éliminer.

On déplace ensuite cette ligne afin que son coefficient serve de pivot. Ainsi, on évite le doute qui peut subsister lorsqu'on cherche à savoir si un flottant très petit est rigoureusement nul ou non. Puisque ce flottant est très petit, il est très peu probable, grâce à cette méthode, qu'il serve effectivement de pivot. De plus, cette méthode minimise les erreurs numériques en travaillant toujours, à chaque étape, sur des nombres plus grands, et dont la représentation en mémoire est donc plus précise.

Rq

Pour avoir une plus grande stabilité, on aurait pu choisir la méthode du **pivot total** qui consiste à autoriser des échanges de lignes et de colonnes et à prendre comme pivot à l'étape i le coefficient $a_{i'j'}$ ($i \leq i' \leq n$ et $j \leq j' \leq n$) ayant la plus grande valeur absolue. Cette méthode présente une difficulté : l'échange de colonne entraîne une modification de l'ordre des inconnues.

3.4. Mise en œuvre de l'algorithme du pivot partiel

On considère le système $Ax = B$. La ligne L_i désigne à la fois les coefficients de A (qui sont dans une matrice, c'est-à-dire un tableau, une liste de listes) et les seconds membres, qui sont dans une matrice colonne B. On numérote les lignes de 0 à $n-1$ si le système comporte n lignes. On obtient donc l'algorithme en « pseudo-code » suivant :

Pseudo Code

```
pour i de 0 à n-2
        trouver j entre i et n-1 tel que |a_(j,i)| soit maximale
        échanger (L_i) et (L_j)
        pour k de i+1 à n-1
                L_k <- L_k - a_(k,i)/a_(i,i) L_i
```

Cette étape étant réalisée, on est certain que le système est sous forme triangulaire, et il ne reste qu'à programmer la phase de remontée évoquée au début de ce cours.

Exemple

Système à résoudre :

$$\begin{pmatrix} 2 & 1 & 3 & 14 \\ 1 & 2 & 1 & 7 \\ 4 & 1 & 1 & 12 \end{pmatrix}$$

- Etape 1 : on échange L_1 et L_3 :

$$\begin{pmatrix} 4 & 1 & 1 & 12 \\ 1 & 2 & 1 & 7 \\ 2 & 1 & 3 & 14 \end{pmatrix}$$

- $L_2 \leftarrow L_2 - \frac{1}{4}L_1$ et $L_3 \leftarrow L_3 - \frac{1}{2}L_1$

$$\begin{pmatrix} 4 & 1 & 1 & 12 \\ 0 & \frac{7}{4} & \frac{3}{4} & 4 \\ 0 & \frac{1}{2} & \frac{5}{2} & 8 \end{pmatrix}$$

- Etape 2 : $L_3 \leftarrow L_3 - \frac{2}{7}L_2$

$$\begin{pmatrix} 4 & 1 & 1 & 12 \\ 0 & \frac{7}{4} & \frac{3}{4} & 4 \\ 0 & 0 & \frac{16}{7} & \frac{48}{7} \end{pmatrix}$$

- Solution :

$$\begin{cases} x = 2 \\ y = 1 \\ z = 3 \end{cases}$$

4. Mise en œuvre en Python

4.1. Fonctions de base

On suppose que le système à résoudre est représenté par une matrice A et un vecteur colonne B (second membre), de telle sorte que la résolution du système devient équivalente à la résolution de l'équation $AX = B$ où X est le vecteur colonne représentant les inconnues à chercher.

On propose pour réaliser cet algorithme du pivot de Gauss partiel de mettre en place quatre fonctions qui seront appelées à chaque fois qu'elles seront nécessaires.

- ◇ La fonction `Matrice_aug` qui prend pour arguments A et B et qui renvoie la matrice augmentée M.
- ◇ La fonction `chercher_pivot` prend pour argument une matrice A et un indice i. Elle doit renvoyer un indice $j \geq i$ tel que $|a_{ji}|$ soit maximale.
- ◇ La fonction `echanger_lignes` prend pour argument une matrice A et deux indices i et j. Il n'est pas nécessaire que cette fonction renvoie un résultat, puisqu'elle modifiera directement la matrice fournie en entrée.
- ◇ La fonction `Combinaison` prend pour argument une matrice A, deux indices i et j et un réel λ. Il n'est pas nécessaire que cette fonction renvoie un résultat, puisqu'elle modifiera directement la matrice fournie en entrée, en remplaçant (L_i) par $(L_i) + \lambda(L_j)$.

- Fonction `Matrice_aug`

- Fonction `chercher_pivot`

- Fonction `echanger_lignes`

- Fonction `Combinaison`

4.2. Algorithme

Le dernier point à noter tient au fait que le type `list`, en Python, est mutable. Ainsi, si l'on passe une matrice (sous la forme d'une liste de listes) en argument d'une fonction, les modifications de la matrice affectent directement la variable originale. C'est pourquoi, par précaution, pour ne pas modifier les données passées en entrée, on commence par réaliser une copie de la matrice initiale, que l'on manipule ensuite dans les fonctions évoquées ci-dessus.

Il en résulte finalement l'algorithme ci-dessous, en Python :

5. Complexité

5.1. Mise sous forme triangulaire

Lors de la mise sous forme triangulaire, pour i fixé, la recherche de j a un coût linéaire en n ($n-i$ comparaisons), les éventuels échanges de lignes également ($2n+2$ affectations), ainsi que chacune des combinaisons ($2n+2$ affectations, additions et divisions). Il y a au plus n combinaisons, donc le coût est quadratique à i fixé. Puisque i décrit $[0, n-2]$, la complexité de la mise sous forme triangulaire est cubique, c'est-à-dire en $O(n^3)$.

5.2. Phase de remontée

La phase de remontée, quant à elle, est clairement quadratique (on a une complexité linéaire à i fixé, et on réalise n itérations).

5.3. Conclusion

Au total, la complexité de l'algorithme est donc cubique, et le temps de calcul est en grande partie lié à la mise sous forme triangulaire.

Sur un ordinateur personnel, le nombre d'opérations élémentaires est de l'ordre de grandeur de 10^8 par seconde. Il est donc exclu de lancer un algorithme de complexité cubique pour $n > 1000$. Pour $n = 1000$, l'ordre de grandeur du temps de calcul serait la minute, mais pour $n = 10000$, il faudrait environ trois heures, et pour $n = 100000$, il faudrait une année.
Il arrive que, dans des problèmes de recherche, des systèmes linéaires comportant un milliard d'équations et d'inconnues nécessitent d'être résolus. Les supercalculateurs sont capables d'effectuer jusqu'à 10^{20} opérations par seconde ce qui rend le calcul presque instantané si la complexité de la résolution du système est quadratique (10 millisecondes suffisent), mais affreusement long (une année) si celle-ci est cubique. Il existe heureusement des méthodes de résolution approchée des systèmes linéaires, souvent moins précises, de complexité quadratique, utilisant des propriétés des matrices creuses. Mais cela dépasse bien sûr le cadre de cette introduction.

6. Pour aller plus loin : problème de conditionnement

On considère le système de deux équations à deux inconnues suivant :

$$\begin{cases} 3.218613\,x + 6.327917\,y &= 10.546530 \\ 3.141592\,x + 4.712390\,y &= 7.853982 \end{cases}$$

Le système est non singulier et sa solution est donnée par $x = y = 1$.
On considère maintenant un système voisin :

$$\begin{cases} 3.218611\,x + 6.327917\,y &= 10.546530 \\ 3.141594\,x + 4.712390\,y &= 7.853980 \end{cases}$$

Ce système est non singulier, et sa solution est donnée par $x = -5$ et $y = 5$.
Ces deux systèmes sont voisins, pourtant leurs solutions sont très différentes. On parle dans ce cas de systèmes mal conditionnés. Résoudre un système mal conditionné avec un ordinateur peut être problématique si l'ordinateur utilise trop peu de chiffres significatifs.
Dans l'exemple précédent, on observe que si l'ordinateur ne retient que 6 chiffres significatifs, il est complètement inespéré d'obtenir une solution raisonnablement proche de la solution.

Définition

Le conditionnement d'une matrice A non singulière est défini par :

$$\text{cond}(A) = \|A\|.\|A^{-1}\| \qquad \text{où } \|.\| \text{ est une norme matricielle.}$$

On peut remarquer que $\text{cond}(A) \geq 1$.

On considère un système linéaire $AX = b$. Si δb est une perturbation de b et si on résout $AY = b + \delta b$, on obtient par linéarité $Y = X + \delta X$ avec $A\delta X = \delta b$.

La question est de savoir s'il est possible de majorer l'erreur relative $\frac{\|\delta X\|}{\|X\|}$ sur la solution du système en fonction de l'erreur relative $\frac{\|\delta b\|}{\|b\|}$ commise sur le second membre. Il est possible de démontrer que

$$\frac{\|\delta X\|}{\|X\|} \leq \text{cond}(A)\frac{\|\delta b\|}{\|b\|} \qquad \text{où } \text{cond}(A) \text{ est le nombre de conditionnement de la matrice } A.$$

On constate alors que plus le conditionnement de la matrice est grand, plus la solution du système linéaire risque d'être sensible aux perturbations des données.

Rq Le fait qu'un système linéaire soit bien conditionné n'implique pas nécessairement que sa solution soit calculée avec précision. Il faut en plus utiliser des algorithmes stables. Inversement, le fait fait d'avoir une matrice avec un grand conditionnement n'empêche pas nécessairement le système global d'etre bien conditionné pour des choix particuliers du second membre.

Si $\frac{\|\delta b\|}{\|b\|}$ est de l'ordre de la précision relative $\eta = 10^{-p}$ du calculateur, alors $\frac{\|\delta X\|}{\|X\|}$ pourrait au pire être égal à

$$\frac{\|\delta X\|}{\|X\|} = \text{cond}(A)\eta.$$

Si on calcule la solution du système avec un ordinateur à p chiffres significatifs en valeur décimale, on ne pourra *a priori* pas garantir plus de $E\left(p - \log_{10}(\text{cond}(A))\right)$ chiffres significatifs sur la solution.

Si on applique cette règle au système linaire de l'exemple, il est facile de vérifier que $\text{cond}(A) = 10^7$, par conséquent, on peut perdre jusqu'à 7 chiffres significatifs lors de sa résolution. Il faut donc un ordinateur calculant avec 10 chiffres significatifs pour être sûr d'obtenir les 3 premiers chiffres de la solution.