TELECOM Nancy *(ESIAL)*
Maths Numériques

feuille 3 : normes vectorielles et matricielles

Exercice 1 *Boules unités*

Dessiner la boule unité fermée dans $\mathbb{R}^2$ (c'est à dire l'ensemble des points $z = [x, y]^\top \in \mathbb{R}^2$ tels que $\|z\| \leq 1$) pour les 3 normes suivantes :

1. $\|z\| := \|z\|_2 = \sqrt{x^2 + y^2}$ (la norme euclidienne) ;
2. $\|z\| := \|z\|_1 = |x| + |y|$, (la norme 1) ;
3. $\|z\| := \|z\|_\infty = Max\{|x|, |y|\}$, (la norme infinie).

Exercice 2 *normes vectorielles*

1. Montrer que les normes 1, 2 et ∞ définissent bien des normes, c'est à dire vérifient les propriétés attendues : application de $\mathbb{R}^n$ dans $\mathbb{R}^+$, vérifiant (i) $\|x\| = 0 \iff x = 0$, (ii) $\|\alpha x\| = |\alpha|\|x\|$ et (iii) $\|x + y\| \leq \|x\| + \|y\|$. Aide : seule l'inégalité triangulaire pour la norme 2 présente des difficultés ; pour la montrer on utilisera l'inégalité de Cauchy-Schwartz :

$$|(x|y)| := |\sum_i x_i y_i| \leq \|x\|_2 \|y\|_2$$

   Pour démontrer l'inégalité de Cauchy-Schwartz, on peut partir de : $\|x - \lambda y\|_2^2 = (x - \lambda y | x - \lambda y) \geq 0$ et développer le produit scalaire par bilinéarité.

2. D'une manière générale, étant donné $p \in \mathbb{N}^*$ on définit la norme p d'un vecteur de $\mathbb{R}^n$ par :

$$\|x\|_p = \left(\sum_i |x_i|^p \right)^{1/p}$$

   (la norme 1 et la norme 2 en sont des cas particuliers). On admettra qu'on définit bien ainsi une norme (l'inégalité triangulaire, qui porte le nom d'inégalité de Minkowski pour les sommes, est plus difficile à montrer). Par contre montrer que pour tout vecteur x on a :

$$\lim_{p \to +\infty} \|x\|_p = \|x\|_\infty$$

   ce qui justifie le terme "norme infinie".

3. codage en machine d'un vecteur : soit x un vecteur de $\mathbb{R}^n$ et $\hat{x}$ son codage en machine supposé sans dépassement de capacité (on a donc $\hat{x}_i = fl(x_i) = x_i(1 + \epsilon_i)$ avec $|\epsilon_i| \leq \mathbf{u}$). On pose $\delta x = \hat{x} - x$ montrer que pour toute p norme vectorielle on a :

$$\frac{\|\delta x\|_p}{\|x\|_p} \leq \mathbf{u}, \forall x \neq 0$$

Exercice 3 *normes matricielles*

On se place dans l'espace vectoriel des matrices à coefficients réels (matrices carrées ou rectangulaires).

1. Montrer que :

$$\|A\|_\infty = \max_i \sum_j |a_{i,j}|, \quad \|A\|_1 = \max_j \sum_i |a_{i,j}|$$

   Aide : dans un premier temps par majoration on obtient assez facilement $\|A\|_\infty \leq \max_i \sum_j |a_{i,j}|$ et $\|A\|_1 \leq \max_j \sum_i |a_{i,j}|$, puis en choisissant un bon vecteur (différent pour les 2 cas) on montre que ces bornes sont atteintes.

2. Application : calculer la norme 1 (ou infinie...) de la matrice A suivante et de son inverse :

$$A = \begin{bmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix} \quad A^{-1} = \begin{bmatrix} 25 & -41 & 10 & -6 \\ -41 & 68 & -17 & 10 \\ 10 & -17 & 5 & -3 \\ -6 & 10 & -3 & 2 \end{bmatrix}$$

et en déduire son conditionnement relativement à cette norme (réponse $\kappa_1(A) = 4488$).

3. Soit A une matrice réelle non nulle et son codage en machine $\hat{A}$ (supposé sans dépassement de capacité) on pose $\delta A = \hat{A} - A$. Déduire de la question 1 que :

$$\frac{\|\delta A\|_1}{\|A\|_1} \leq \mathbf{u}, \quad \text{et } \frac{\|\delta A\|_\infty}{\|A\|_\infty} \leq \mathbf{u}$$

Exercice 4 *système linéaire perturbé*

Soit une matrice $A \in \mathcal{M}_{n,n}(\mathbb{R})$ inversible, et un vecteur $b \in \mathbb{R}^n$ donnés ($b \neq 0$). On considère le système linéaire $Ax = b$ ainsi que le système linéaire $(A + \delta A)(x + \delta x) = b + \delta b$ où δA et δb sont des perturbations sur les données, peut être simplement dues au codage en machine de A et b. On utilise une norme vectorielle $\|.\|$ (par exemple la norme 1 ou 2 ou infinie) et la norme matricielle $\|.\|$ induite par la norme vectorielle choisie. Le but de l'exercice est de montrer que si $\|A^{-1}\delta A\| < 1$ alors on a l'inégalité :

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A)}{1 - \|A^{-1}\delta A\|} \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right)$$

(où $\kappa(A) = \|A\|\|A^{-1}\|$ est le conditionnement de la matrice A). La condition $\|A^{-1}\delta A\| < 1$ est assez naturelle et vérifiée si la perturbation sur A est suffisamment petite. On rappelle le théorème suivant vu en cours : soit $B \in \mathcal{M}_{n,n}(\mathbb{R})$ telle que $\|B\| < 1$ alors (i) la matrice $I - B$ est inversible, (ii) $(I - B)^{-1} = I + B + B^2 + B^3 + \ldots$ et (iii) $\|(I - B)^{-1}\| \leq 1/(1 - \|B\|)$.

1. En utilisant le théorème précédent, montrer que la condition $\|A^{-1}\delta A\| < 1$ implique l'inversibilité de la matrice $A + \delta A$ et que :

$$\|(A + \delta A)^{-1}\| \leq \frac{\|A^{-1}\|}{1 - \|A^{-1}\delta A\|}$$

Montrer que si l'on suppose $\|\delta A\| < 1/\|A^{-1}\|$ alors nécessairement $\|A^{-1}\delta A\| < 1$.

2. Montrer que $\delta x = (A + \delta A)^{-1}(\delta b - \delta A x)$ puis en déduire que :

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\|A^{-1}\|}{1 - \|A^{-1}\delta A\|} \left(\frac{\|\delta b\|}{\|x\|} + \|\delta A\| \right)$$

3. A partir de l'égalité $Ax = b$ montrer que $\|x\| \geq \|b\|/\|A\|$ et en déduire le résultat cherché.

Ce qu'il faut retenir de ce résultat : malgré les majorations successives qui ont été faites pour l'obtenir, l'inégalité est quand même "presque" optimale car elle devient "presque" une égalité dans certains cas. Si on suppose la perturbation assez petite de sorte que $\|A^{-1}\delta A\|$ soit négligeable devant 1, alors l'idée est que les erreurs relatives sur les données risquent d'entrainer une erreur relative sur le résultat amplifiée d'un facteur égal au conditionnement de la matrice A. Exemples :

— sur la matrice A de l'exercice précédent, pour $b = [32, 23, 33, 31]^\top$ on obtient la solution $x = [1, 1, 1, 1]^\top$; maintenant si on considère $b + \delta b = [32.1, 23.1, 33.1, 31.1]^\top$, on obtient $x + \delta x = [-0.2, 3, 0.5, 1.3]^\top$ assez différente de x ce qui est expliqué par le conditionnement déjà élevé de cette matrice.

— en supposant juste des erreurs sur le codage en machine, on introduit des erreurs relatives de l'ordre de $\mathbf{u}$, il ne faut donc pas espérer résoudre correctement des systèmes linéaires avec des matrices qui ont un conditionnement de l'ordre de $1/\mathbf{u}$.

Exercice 5 *Estimation du conditionnement d'une matrice (extrait de l'examen de 2013)*

Soit A une matrice carrée $n \times n$ inversible et admettant une factorisation $A = LU$. Le but de ce problème est d'écrire un code python permettant d'estimer le conditionnement de A en norme 1 avec l'algorithme de Boyd et Hager qui est utilisé dans la bibliothèque Lapack. On rappelle que :

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1, \quad \|A\|_1 = \sup_{x \neq 0} \frac{\|Ax\|_1}{\|x\|_1} = \max_{j \in [\![1,n]\!]} \sum_{i=1}^{n} |a_{i,j}|$$

La partie facile consiste à calculer $\|A\|_1$. Calculer $\|A^{-1}\|_1$ (par la définition de $\|.\|_1$ en fonction des coefficients de la matrice) requiert le calcul de A^{-1} et ce coût est jugé souvent prohibitif lorsque l'on veut juste résoudre quelques systèmes linéaires de matrice A tout en donnant à l'utilisateur une idée de la précision des calculs. Dans ce problème presque toutes les questions sont indépendantes [1] et il n'y aucune question théorique sur la précision et/ou la convergence de l'algorithme.

1. Rappeler comment on peut résoudre un système linéaire $Ax = b$ lorsqu'on connaît la factorisation $A = LU$. Ne pas rentrer dans les détails (i.e. ne pas écrire l'algorithme en détail mais soyez **précis** dans l'ordre des 2 opérations de haut niveau à effectuer). Pour des besoins ultérieurs on notera *resoud_A_x_eq_b(L,U,b)* une fonction qui retourne la solution x (équivalente à notre `lu_solve` du module `syslin` du TP2).
2. Montrer que cette factorisation permet aussi de résoudre facilement des systèmes linéaires de la forme $A^\top x = b$ en résolvant un système triangulaire inférieur puis un système triangulaire supérieur à diagonale unité. Idem ne pas rentrer dans les détails algorithmiques.
3. Ecrire l'algorithme (en pseudo-code et pas en python) permettant de résoudre $A^\top x = b$ connaissant $A = LU$. Remarque : ne pas transposer les matrices L et U mais utiliser $L_{j,i}$ lorsque l'algorithme a besoin de $(L^\top)_{i,j}$. Calculer le nombre d'opérations de cet algorithme. Pour des besoins ultérieurs on notera *resoud_AT_x_eq_b(L,U,b)* une fonction qui implémente cet algorithme.
4. Voici l'algorithme de Boyd et Hager permettant d'approcher la norme 1 d'une matrice B (de taille $n \times n$) :

   Etant donné B l'algorithme [2] calcule $\gamma \leq \|B\|_1$ et x tel que $\|Bx\|_1 = \gamma \|x\|_1$:

   $x \leftarrow ones(n)/n$ (soit $x_i \leftarrow 1/n, i = 1, \ldots, n$)
   répéter
   $y \leftarrow Bx$
   $\xi \leftarrow sign(y)$
   $z \leftarrow B^\top \xi$
   si $\|z\|_\infty \leq z^\top x$
   $\gamma \leftarrow \|y\|_1$
   retourner γ et x (sortie de la boucle **répéter** et fin de l'algorithme)
   $x \leftarrow e^j$ où j est le plus petit entier tel que $|z_j| = \|z\|_\infty$

   On montre qu'en général cet algorithme se termine en quelques itérations et donne un γ supérieur à $\|B\|_1/10$ ce qui est suffisant pour le but cherché (détecter un système linéaire instable par rapport au epsilon machine).

   (a) En supposant que l'algorithme se termine effectivement, montrer qu'on a bien $\gamma \leq \|B\|_1$.

1. Admettez le résultat d'une question et passer à la suivante si vous êtes bloqué.
2. La fonction *sign* renvoie un vecteur de même taille que son argument avec $\xi_i = 1$ si $y_i > 0$, $\xi_i = -1$ si $y_i < 0$ et $\xi_i = 0$ si $y_i = 0$. La notation e^j désigne le j ème vecteur de la base canonique.

(b) Il semble bizarre d'utiliser cet algorithme pour estimer $\|B\|_1$ puisqu'on dispose d'une formule exacte...Montrer qu'on peut l'adapter pour calculer $\|A^{-1}\|_1$ sans connaître A^{-1} mais en résolvant des systèmes linéaires de la forme $Ax = b$ et $A^\top x = b$ ce qui est facile si on connaît la factorisation LU de A. Ecrire l'adaptation de cet algorithme permettant d'approcher $\|A^{-1}\|_1$ connaissant la factorisation LU de A. Utiliser les fonctions *resoud_A_x_eq_b(L,U,b)* et *resoud_AT_x_eq_b(L,U,b)* précédentes.

(c) Ecrire maintenant une fonction python d'entête `estime_cond1(A, LU):` qui retourne une estimation du conditionnement $\kappa_1(A)$ connaissant la factorisation LU de A stockée de manière compacte dans le tableau `LU`. Cette fonction serait à rajouter dans votre module `syslin` dans lequel :

— `numpy` est importé via `import numpy as np` ;

— vous disposez dans syslin des fonctions `lu_solve(LU,b)` (qui implémente *resoud_A_x_eq_b(L,U,b)*) mais aussi de :

— `lu_solve_T(LU,b)` qui implémente *resoud_AT_x_eq_b(L,U,b)* ;

— `norm_inf(x)` et `norm_1(x)` qui calculent respectivement $\|x\|_\infty$ et $\|x\|_1$ (x est un vecteur et `x` un tableau 1d numpy) ;

— `norm_mat_1(A)` qui calcule la norme 1 d'une matrice A (tableau 2d numpy `A`) ;

— et `mon_signe` qui implémente la fonction *sign* utilisée par l'algorithme.

On rappelle que la fonction numpy `dot` permet de faire des produits matrice-vecteur mais aussi le produit scalaire de deux vecteurs et que la fonction numpy `argmax` appliquée sur un tableau 1d renvoie l'indice de l'élément maximal du tableau (le plus petit indice si le max est atteint par plusieurs composantes).