L3 - Méthodes numériques

TP 3 - Algèbre linéaire

Support de TP

Les TPs visent à appliquer de manière concrète les notions vues en cours. Il est donc conseillé de venir avec vos supports de cours ou de les télécharger sur le web.

Les TPs notés doivent être soumis sur la page Moodle du cours dans les deux semaines suivant le TP. Vous devez soumettre un fichier zip contenant le code **sans bug** avec un scripte python **TP3_1.py,TP3_2.py,..** par partie du TP ainsi qu'un court rapport de 2 pages maximum en **PDF** (sans page de titre). Le rapport ne doit contenir aucune sortie terminal ou ligne de code. Vous devez par contre pour chaque section du TP rédiger une réflexion sur les notions utilisées et ce que vous avez appris.

1 Opérations de base

Cette section du TP vise à implémenter en Python certaines multiplications matricielles vues dans l'exercice 1 (page 12) du cours d'algèbre linéaire.

Une bibliothèque d'algèbre linéaire est donnée dans le fichier `linalgutils.py`. Vous y trouverez toutes les fonctions vues en cours. Notez que les implémentations dans ce fichier ne sont particulièrement pas efficaces et sont fournies ici pour des raisons pédagogique uniquement.

1. Initialiser en mémoire les vecteurs `u,v` (`np.array,np.arange,np.ones`) et les matrices `A,B` (`np.array`).
2. Pour toutes les questions suivantes, implémenter la solution qui appeler la fonction dans `linalgutils` et celle utilisant les fonctions numpy en comparant leur sortie. les fonctions sont données entre parenthèses.
   - Coder et afficher le résultat des opérations $\mathbf{u}^\top\mathbf{u}$ et $\mathbf{u}^\top\mathbf{v}$ (fonctions `linalgutils.ddot,np.dot`).
   - Coder les multiplications matricielles $\mathbf{Au}$ et $\mathbf{Av}$ et afficher les résultats (fonction `linalgutils.dgemv,np.dot`).
   - Coder les multiplications $\mathbf{Au}$ et $\mathbf{Av}$ en utilisant BLAS (fonction `scipy.linalg.blas.dgemv`).
   - Comparer les deux implémentations et mesurer la différence entre les deux solutions (fonction `linalgutils.err`).
   - Coder la multiplication $\mathbf{uu}^\top$ et afficher le résultat (fonction `linalgutils.dger,np.outer`).

2 Résolution de système linéaire

Dans cette section nous allons chercher à résoudre un système linéaire de la forme $\mathbf{Ax} = \mathbf{b}$. Pour celà nous allons coder un pivot de Gauss et résoudre le système triangulaire puis comparer à l'implémentation de référence LAPACK.

1. Initialiser et afficher une matrice $\mathbf{A}_0$ et un vecteur $\mathbf{b}_0$ avec :

$$\mathbf{A}_0 = \begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix}, \quad \mathbf{b}_0 = \begin{bmatrix} 8 \\ -11 \\ -3 \end{bmatrix}$$

2. Copier $\mathbf{A}_0$ et $\mathbf{b}_0$ dans une matrice $\mathbf{A}$ et un vecteur $\mathbf{b}$ et appliquer le pivot de Gauss vu en cours (fonction `pivot`). Imprimer la matrice triangulaire résultante $\mathbf{A}$ et le vecteur $\mathbf{b}$.
3. Comparer le résultat au pivot illustré page 46 du cours.

4. Maintenant que la matrice $\mathbf{A}$ est triangulaire supérieure, il faut résoudre le système. Pour ce faire, compléter la fonction `tri_sup_sv` du fichier `linalgutils.y`. Vous pouvez vous inspirer de `tri_inf_sv` qui résout un système pour une matrice triangulaire inférieure.
5. Afficher la solution $\mathbf{x}$ obtenue ainsi que $\mathbf{A}_0\mathbf{x}$. Comparer à $\mathbf{b}_0$.
6. Copier $\mathbf{A}_0$ et $\mathbf{b}_0$ dans une matrice $\mathbf{A}1$ et un vecteur $\mathbf{b}1$ et résoudre le système linéaire à l'aide de la fonction numpy `np.linalg.solve` et de la fonction LAPACK (fonction `scipy.lilalg.lapack.dgesv`).
7. Comparer les solutions (`pivot+tri_sup_sv` VS LAPACK VS numpy).

3 Précision et temps de calcul

Dans cette section nous effectuerons une comparaison systématique des solutions de systèmes linéaire pour différentes tailles de problème.

1. Prendre $n = 10$.
2. Initialiser $\mathbf{A}_0, \mathbf{A}, \mathbf{A}_1$ de taille $n \times n$ et $\mathbf{b}_0, \mathbf{b}, \mathbf{b}_1$ de taille n à la valeur 1 pour toutes les composantes.
3. Initialiser $\mathbf{A}_0$ avec des valeurs aléatoires (fonction `np.random.rand`) et copier les valeurs dans $\mathbf{A}$ et $\mathbf{A}1$.
4. Résoudre le système avec `pivot` et `tri_sup_sv`. Afficher le temps de calcul (fonctions `utils.tic` et `utils.toc`).
5. Calculer l'erreur entre $\mathbf{b}_0$ et $\mathbf{A}_0\mathbf{x}$ et l'afficher (fonctions `mgemv` et `err`).
6. Résoudre le système en utilisant `np.linalg.solve`. Afficher le temps de calcul et l'erreur numérique (fonction `err`).
7. Changer la valeur de n pour 100, 500 et 1000. Conclusions ?

4 Evolution de la précision et du temps de calcul

Le but est de tracer le temps de calcul et l'erreur de la résolution en fonction de la taille du système avec Gnuplot.

1. Pour une liste de valeurs de `n` allant de 50 à 500 par pas de 50. Effectuer les étapes de la section précédente dans une boucle.
2. Tracer l'évolution du temps de calcul et de l'erreur numérique en fonction de n pour chacune des méthodes.