TP Fortran90
Résolution d'un système linéaire.

1 Problème à résoudre

La discrétisation du problème:

$$-\frac{d^2}{dx^2}u(x) = 1 \text{ sur }]0,1[,\ u(0) = u(1) = 0 \tag{1}$$

par différences finies sur n points régulièrement espacés, conduit à résoudre le système linéaire $Ax = b$. La matrice A a la forme:

$$\frac{1}{(n+1)^2}\begin{pmatrix} 2 & -1 & 0 & ... & & & \\ -1 & 2 & -1 & 0 & ... & & \\ 0 & -1 & 2 & -1 & 0 & & \\ & & & ... & & & \\ & ... & 0 & -1 & 2 & -1 & 0 \\ & & ... & 0 & -1 & 2 & -1 \\ & & & ... & 0 & -1 & 2 \end{pmatrix}$$

et le vecteur b a toutes ses composantes égales à 1.

2 Utilisation explicite d'une matrice

- Écrire une sous-routine fortran 90 qui initialise une matrice de taille $n \times n$ et un vecteur de taille n de la forme ci-dessus.

  Les paramètres de la sous-routine sont :

  - La matrice A en sortie,
  - Le vecteur b en sortie.

- Écrire une sous-routine fortran 90 qui calcule la solution d'un système linéaire par la méthode du gradient conjugué :

$$\begin{aligned} &d_{(0)} = r_{(0)} = b - Ax_{(0)} \\ &\text{boucle tant que } |r_{(k)}| > \epsilon \\ &\quad \alpha_{(k)} = \frac{r_{(k)}^T r_{(k)}}{d_{(k)}^T A d_{(k)}} \\ &\quad x_{(k+1)} = x_{(k)} + \alpha_{(k)} d_{(k)} \\ &\quad r_{(k+1)} = r_{(k)} - \alpha_{(k)} A d_{(k)} \\ &\quad \beta_{(k+1)} = \frac{r_{(k+1)}^T r_{(k+1)}}{r_{(k)}^T r_{(k)}} \\ &\quad d_{(k+1)} = r_{(k+1)} + \beta_{(k+1)} d_{(k)} \\ &\text{fin de la boucle} \end{aligned}$$

Remarque

Autant que possible, on utilisera les opérations vectorielles offertes par fortran 90.

L'algorithme n'a besoin que de

- une seule variable réelle pour contenir successivement tous les $\alpha_{(k)}$,
- une seule variable réelle pour contenir successivement tous les $\beta_{(k)}$,
- un seul vecteur réel pour contenir successivement tous les $x_{(k)}$,
- un seul vecteur réel pour contenir successivement tous les $d_{(k)}$,
- un seul vecteur réel pour contenir successivement tous les $r_{(k)}$,
- deux variables réelles pour contenir 2 produits scalaires successifs $r_{(k)}^T r_{(k)}$ et $r_{(k+1)}^T r_{(k+1)}$.

Les paramètres de la sous-routine sont :

- la matrice A en entrée,
- le vecteur b en entrée,
- le vecteur x en entrée/sortie,
- le réel ϵ en entrée,
- le réel $|r_{(k)}|$ (dernière valeur de la norme de $|r_{(k)}|$) en sortie.

- Écrire un programme principal qui
  - lit une valeur de n au clavier
  - alloue dynamiquement la mémoire pour la matrice A et les vecteurs b et x
  - appelle la sous-routine qui initialise A et b
  - met toutes les composantes de x à 0, valeur initiale de la solution ($x_{(0)} = 0$)
  - appelle la sous-routine de résolution,
  - affiche le résultat.

  Tester ce programme pour plusieurs valeurs de n (on pourra mettre les résultats dans un fichier et tracer le graphe du résultat avec `gnuplot` par exemple).

3 Utilisation d'une matrice implicite

On remarquera que l'algorithme ci-dessus utilise la matrice A uniquement pour calculer le produit de cette matrice par des vecteurs.
La matrice du système contient beaucoup de coefficients nuls et elle a une structure particulièrement simple.
On peut éviter de stocker en mémoire cette matrice en écrivant une sous-routine qui calcule le produit matrice - vecteur $v = Au$ par la formule spécifique :

$$\begin{aligned} v_1 &= 2u_1 - u_2 \\ v_i &= -u_{i-1} + 2u_i - u_{i+1} \quad (i = 2, \ldots, n-1) \\ v_n &= -u_{n-1} + 2u_n \end{aligned}$$

Écrire une autre version du code dans laquelle :

- La matrice complète n'est plus générée.
- La sous-routine ci-dessus est appelée dans le code du gradient conjugué à la place du produit matrice - vecteur standard.

Comparer les performances des deux versions (temps d'exécution en fonction de n).