Université Paris XIII
Institut Galilée
Formation MACS - 2ème année

Année 2006-2007
N. Bereux, S. Ferrand

Initiation au FORTRAN 77

Feuille n° 4

Exercice 1 Entrées/sorties : lecture et écriture de matrices

1. Recopier le fichier `mat.dat`. Ce fichier contient une matrice réelle de taille $m \times n$ sous le format suivant :

$$\begin{array}{l} m\ n \\ a(1,1) \\ a(2,1) \\ \vdots \\ a(m,1) \\ a(1,2) \\ \vdots \\ a(m,2) \\ \vdots \\ a(m,n) \end{array} \tag{1}$$

2. Écrire une routine `lit_mat.f` qui relit une matrice réelle de taille $m \times n$ depuis un fichier du disque (identifié par un numéro d'unité logique). Cette matrice sera stockée dans un tableau bidimensionnel de taille $mmax \times nmax$. Quels sont les arguments à passer à la routine de lecture ?
3. Écrire une routine `affiche_mat.f` qui affiche à l'écran une matrice de taille $m \times n$. Quels sont les arguments à passer à la routine d'écriture ? Prévoir un argument de type *character* pour le nom de la matrice de façon à ce que l'affichage resssemble à :

```
a( 1, 1)=   0.811054945
a( 2, 1)=  -0.0731715783
a( 3, 1)=  -0.749524891
```

   si l'on passe en argument `'a'`.
4. Écrire un programme principal `io.f` :
   - relit la matrice a depuis le disque,
   - l'affiche à l'écran,
   - initialise un vecteur x de taille m tel que $x = (1, \ldots, 1)^T$,
   - affiche ce vecteur à l'écran.

Exercice 2 Produit matrice-vecteur Soit a une matrice réelle de taille $m \times n$, et x un vecteur de taille n. On appelle y le vecteur obtenu en multipliant A par x. Ses composantes sont données par :

$$y(i) = \sum_{j=1}^{n} a(i,j)\, x(j), \quad i = 1, m.$$

1. Écrire une routine `prod_mv1.f` qui effectue le produit d'une matrice réelle a, de taille $m \times n$, stockée dans un tableau de taille $mmax \times nmax$ par un vecteur colonne x de taille m. Le résultat sera stocké dans un vecteur y. L'algorithme à implémenter est le suivant :

   PROD_MV1(a, x, b) :

   ```
   1  for i ← 1 to m
   2  ▷ Boucle sur les lignes
   3        do
   4           for j ← 1 to n
   5           ▷ Boucle sur les colonnes
   6                 do
   7                    y(i) ← a(i, j) x(j) + y(i)
   ```

2. Écrire un programme principal `blas2.f` qui :
   - initialise une matrice identité de taille $n \times n$
   - initialise le vecteur

   $$x \in \mathbb{R}^n, \quad x(i) = i \quad i \in [1 : n]$$

   - appelle la routine `prod_mv1.f`
   - vérifie le résultat : on utilisera la routine `comb_lin.f` et la fonction `norm2.f` pour afficher l'erreur relative :

   $$err = \frac{||x - y||_2}{||x||_2}.$$

3. Écrire un `makefile` pour compiler le programme.

4. Commencer par prendre $n = 3$ pour valider le programme. Puis, passer à $n = 3000$. Mesurer le temps CPU pris par la multiplication matrice-vecteur.
   Pour mesurer le temps, on effectuera deux appels à la fonction `etime`, avant et après le produit matrice-vecteur, sur le modèle suivant :

   ```
         real t(2)
         real t0,t1
         real etime
         external etime
         .....
   c
         t0=etime(t)
         call prod_mv_col(...
         t1=etime(t)
   c
         write(6,*) 'Tps CPU pour le produit matrice-vecteur: ',t1-t0
   c
         ......
   ```

   On utilisera l'option de compilation `-O3` qui permet d'optimiser le code obtenu.

5. Remplacer l'appel à `prod_mv1.f` par un appel à la routine BLAS `sgemv.f` dont voici les spécifications (retour de la commande `man sgemv` :

SUBROUTINE SGEMV(TRANSA, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)

PURPOSE

sgemv performs one of the matrix-vector operations y := alpha*A*x + beta*y, or y := alpha*A'*x + beta*y, where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.

ARGUMENTS

TRANSA (input)
On entry, TRANSA specifies the operation to be performed as follows:

TRANSA = 'N' or 'n' y := alpha*A*x + beta*y.

TRANSA = 'T' or 't' y := alpha*A'*x + beta*y.

TRANSA = 'C' or 'c' y := alpha*A'*x + beta*y.

Unchanged on exit.

M (input)
On entry, M specifies the number of rows of the matrix A. M >= 0. Unchanged on exit.

N (input)
On entry, N specifies the number of columns of the matrix A. N >= 0. Unchanged on exit.

ALPHA (input)
On entry, ALPHA specifies the scalar alpha. Unchanged on exit.

A (input)
Before entry, the leading m by n part of the array A must contain the matrix of coefficients. Unchanged on exit.

LDA (input)
On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA >= max(1, m). Unchanged on exit.

X (input)
(1 + (n - 1)*abs(INCX)) when TRANSA = 'N' or 'n' and at least (1 + (m - 1)*abs(INCX)) otherwise. Before entry, the incremented array X must contain the vector x. Unchanged on exit.

INCX (input)

```
          On entry, INCX specifies the increment for the
          elements of X. INCX <> 0.  Unchanged on exit.

BETA (input)
          On entry, BETA specifies the scalar beta. When
          BETA is supplied as zero then Y need not be set on
          input.  Unchanged on exit.

Y (input/output)
          ( 1 + ( m - 1 )*abs( INCY ) ) when TRANSA = 'N' or
          'n'  and  at  least  ( 1 + ( n - 1 )*abs( INCY ) )
          otherwise.  Before entry with BETA non-zero, the
          incremented  array Y must contain the vector y. On
          exit, Y is overwritten by the updated vector y.

INCY (input)
          On entry, INCY specifies the increment for the
          elements of Y. INCY <> 0.  Unchanged on exit.
```

Compilation-édition de liens A l'étape édition de liens, il faut préciser au compilateur l'endroit où rechercher l'objet *sgemv.o* :

```
blas2 : $(obj_blas2)
        g77 -o blas2 $(obj_blas2) -lblas
```

Mesurer le temps écoulé. Qu'observe-t-on ?

6. Implémenter maintenant l'algorithme :

PROD_MV2(a, x, b) :

```
1  for j ← 1 to n
2  ▷ Boucle sur les colonnes
3       do
4          for i ← 1 to m
5          ▷ Boucle sur les lignes
6               do
7                  y(i) ← a(i, j) x(j) + y(i)
```

Quelle est la différence entre cet algorithme et celui proposé à la question 1 ? Mesurer le temps CPU écoulé. Essayer d'expliquer le résultat.

Exercice 3 FACTORISATION DE CHOLESKY (tiré de [1] p.142)
Une factorisation de Cholesky est une factorisation triangulaire adaptée aux matrices symétriques définies positives. Soit $A \in \mathbb{R}^{n\times n}$ une matrice symétrique définie positive, *i.e.* telle que

$$\forall x \in \mathbb{R}^n, \quad x^t\, A\, x > 0.$$

Alors, il existe une unique matrice $L \in \mathbb{R}^{n\times n}$ triangulaire inférieure et à termes diagonaux positifs telle que :

$$A = L\, L^T.$$

L'algorithme suivant permet de calculer L (L remplace A au fur et à mesure de sa construction)

CHOL(a)

1 **for** $j \leftarrow 1$ **to** n
2 ▷ Boucle sur les colonnes
3 **do**
4 **if** $j > 1$
5 **then**
6 $A(j:n, j) \leftarrow A(j:n, j) - A(j:n, 1:j-1)\, A(j, 1:j-1)^T$
7 $A(j:n, j) \leftarrow A(j:n, j)/\sqrt{A(j, j)}$

Programmation :

1. Recopier le fichier `mat_sym.dat`. Ce fichier contient une matrice réelle carrée (de taille $n \times n$) symétrique définie positive sous le format défini par (1). Préparer un programme principal qui relit cette matrice depuis le disque et l'affiche à l'aide des routines développées à l'exercice 1. Écrire un `makefile`.
2. Implémenter dans une routine `chol.f` l'algorithme, en utilisant des appels BLAS dès que possible (voir le BLAS Quick Reference Guide). Valider la routine : vérifier que le facteur de Cholesky inférieur de A est égal à :

$$\begin{pmatrix} 5 & 0 & 0 & 0 \\ 6 & 2 & 0 & 0 \\ 3 & 5 & 8 & 0 \\ 4 & 3 & 1 & 6 \end{pmatrix}$$

3. Initialiser un vecteur $x_{exact} = (1, \ldots, 1)$. Calculer $b = A\, x_{exact}$ en appelant une routine BLAS. Laquelle est la plus appropriée ?
4. On veut à présent résoudre le système linéaire

$$A\, x = b,$$

en utilisant la factorisation de Cholesky de A.
Écrire une routine `solve.f` qui :
– prend en entre le facteur de Cholesky inférieur calculé par `chol` (L) et le second membre du système linéaire (b),
– retourne la solution.
On utilisera la routine BLAS de niveau 3 `strsm` dont voilà les spécifications (retour de la commande `man strsm`) :

```
PURPOSE
       STRSM  solves one of the matrix equations

       where  alpha  is a scalar, X and B are m by n matrices, A is a unit, or
       non-unit,  upper or lower triangular matrix  and  op( A )  is one  of

          op( A ) = A   or   op( A ) = A'.

       The matrix X is overwritten on B.


PARAMETERS
       SIDE   - CHARACTER*1.
```

On entry, SIDE specifies whether op(A) appears on the left or right of X as follows:

SIDE = 'L' or 'l' op(A)*X = alpha*B.

SIDE = 'R' or 'r' X*op(A) = alpha*B.

Unchanged on exit.

UPLO - CHARACTER*1.
On entry, UPLO specifies whether the matrix A is an upper or lower triangular matrix as follows:

UPLO = 'U' or 'u' A is an upper triangular matrix.

UPLO = 'L' or 'l' A is a lower triangular matrix.

Unchanged on exit.

TRANSA - CHARACTER*1. On entry, TRANSA specifies the form of op(A) to be used in the matrix multiplication as follows:

TRANSA = 'N' or 'n' op(A) = A.

TRANSA = 'T' or 't' op(A) = A'.

TRANSA = 'C' or 'c' op(A) = A'.

Unchanged on exit.

DIAG - CHARACTER*1.
On entry, DIAG specifies whether or not A is unit triangular as follows:

DIAG = 'U' or 'u' A is assumed to be unit triangular.

DIAG = 'N' or 'n' A is not assumed to be unit triangular.

Unchanged on exit.

M - INTEGER.
On entry, M specifies the number of rows of B. M must be at least zero. Unchanged on exit.

N - INTEGER.
On entry, N specifies the number of columns of B. N must be at least zero. Unchanged on exit.

ALPHA - REAL .
On entry, ALPHA specifies the scalar alpha. When alpha is zero then A is not referenced and B need not be set before entry. Unchanged on exit.

A - REAL array of DIMENSION (LDA, k), where k is m

```
          when  SIDE = 'L' or 'l'  and is  n  when  SIDE  =  'R'  or  'r'.
          Before  entry   with   UPLO  = 'U' or 'u',  the  leading  k by k
          upper triangular part of the array  A  must  contain  the  upper
          triangular  matrix   and the strictly lower triangular part of A
          is not referenced.  Before entry  with  UPLO = 'L' or 'l',   the
          leading   k by k lower triangular part of the array  A must con-
          tain the lower triangular matrix  and the strictly upper  trian-
          gular  part  of A is not referenced.  Note that when  DIAG = 'U'
          or 'u',  the diagonal elements of A  are not referenced  either,
          but are assumed to be  unity.  Unchanged on exit.

   LDA    - INTEGER.
          On  entry, LDA specifies the first dimension of A as declared in
          the calling (sub) program.  When  SIDE = 'L' or  'l'   then  LDA
          must be at least  max( 1, m ),  when  SIDE = 'R' or 'r' then LDA
          must be at least max( 1, n ).  Unchanged on exit.

   B      - REAL             array of DIMENSION ( LDB, n ).
          Before entry,  the leading  m by n part of  the  array   B  must
          contain   the   right-hand   side   matrix  B,  and  on exit  is
          overwritten by the solution matrix  X.

   LDB    - INTEGER.
          On entry, LDB specifies the first dimension of B as declared  in
          the  calling  (sub)  program.  LDB  must  be  at  least max( 1,
          m ).  Unchanged on exit.
```

5. On va maintenant évaluer le temps CPU nécessaire à la résolution du système linéaire. Modifier le programme principal de façon à :
   - remplacer la lecture d'une matrice depuis le disque par l'initialisation d'une matrice a de taille $n = 3000$, diagonale telle que
   $$a(i,i) = i, \quad i \in [1:n].$$
   - Initialiser x_{exact}, tel que
   $$x_{exact}(i) = 1,\ i \in [1:n],$$
   et $b = A\,x_{exact}$.
   - Supprimer les affichages de matrices et de vecteurs.
   - Calculer et afficher $\dfrac{||x - x_{exact}||_2}{||x_{exact}||_2}$.
   - Introduire un timer (appel à la fonction `etime`) avant la factorisation et après la résolution du système. Afficher le temps CPU écoulé.

   Tester !
6. Chercher dans la librairie LAPACK :
   - une routine permettant de remplacer `chol.f`,
   - une routine permettant de remplacer `solve.f`.

   Remplacer dans le programme principal et mesurer le temps CPU.
   Pour l'édition de liens, il faut indiquer la librairie lapack avant la librairie blas, ce qui correspond dans le makefile à une règle du type :

```
chol : $(obj_chol)
       g77 -o chol $(obj_chol) -llapack -lblas
```

7. On peut reformuler l'algorithme de factorisation de Cholesky, de façon à privilégier les opérations matrices-matrices (au lieu d'opérations matrices-vecteurs). En effet, les opérations matrices-matrices sont les opérations qui permettent d'utiliser les machines à mémoire hiérarchiques de la façon la plus efficace. Une version par blocs (de taille nb) de l'algorithme de Cholesky s'écrit :

CHOL_BLOC(a)

1 **for** $j \leftarrow 1$ **to** n, par pas de nb
2 **do**
3 ▷ jb taille réelle du bloc colonne
4 $jb = min(nb, n - j + 1)$
5 ▷ Mise a jour du bloc colonne courant
6 $A(j:n, j:j+jb-1) \leftarrow A(j:n, j:j+jb-1) - A(j:n, 1:j-1)\, A(j:j+jb-1, 1:j-1)^T$
7 ▷ Factorisation Cholesky du bloc diagonal
8 $L = \leftarrow$ `chol`(A)
9 **if** $(j + jb \leq n)$
10 **then**
11 ▷ Calcul du bloc colonne
12 $A(j+jb:n, j:j+jb-1) \leftarrow A(j+jb:n, j:j+jb-1)\, L^{-T}$

Implémenter l'algorithme dans une routine `chol_bloc` et le tester (par exemple avec $nb = 64$) .

Annexe : les librairies LAPACK et BLAS

BLAS

BLAS Basic Linear Algebra Subprograms, est une librairie de routines qui effectuent des opérations de base impliquant des matrices et des vecteurs. Une implémentation gratuite de cette librairie est disponible sur le site netlib à l'adresse

`http ://www.netlib.org/blas`

Cette bibliothèque sert de support à d'autres librairies scientifiques comme LAPACK. Il en existe de multiples implémentations, qui ont toutes la même interface générique (liste des paramètres d'appel et action de chaque routine) :

- implémentation gratuite standard déjà citée
- implémentation constructeur : le constructeur écrit une implémentation en langage machine, la plus optimisée possible.
- implémentation gratuite qui s'optimise automatiquement ATLAS (Automatically Tuned Linear Algebra Subroutines) disponible sur le site netlib à l'adresse

`http ://www.netlib.org/atlas`

Il existe trois niveaux dans BLAS :

- Le niveau 1, package initial, effectue des opérations de bas niveau comme le produit scalaire et la combinaison linéaire de vecteurs, soit des couts en $O(n)$ opérations flottantes, pour n la dimension des vecteurs impliqués.

- Le niveau 2 comprend les opérations courantes sur les matrice/vecteur qui apparaissent dans les algorithmes d'algèbre linéaire, soit des couts en $O(n^2)$ opérations flottantes.
- Le niveau 3 correspond aux opérations matrice/matrice, soit des couts en $O(n^3)$ opérations flottantes avec un cout en $O(n^2)$ pour les mouvements de données. Ces routines permettent une réutilisation efficace des données résidant dans les caches.

L'utilisation de BLAS dans un programme écrit en C est possible. Les détails de l'interface sont donnés dans la manpage de BLAS.

LAPACK

LAPACK, Linear Algebra PACKage, est une bibliothèque de routines Fortran 77 pour résoudre des problèmes d'algèbre linéaire : résolution de systèmes linéaires, problèmes de moindres carrés, problèmes aux valeurs propres, factorisations de matrice, élimination de Gauss ....
Ces routines font appel pour les opérations algébriques élémentaires aux routines BLAS. Comme pour BLAS, il existe diverses implémentations de LAPACK, gratuites ou non (bon nombre de constructeurs d'ordinateurs proposent des versions optimisées de cette librairie), toutes respectant le même standard.
Une implémentation gratuite de LAPACK est disponible à l'adresse

http ://www.netlib.org/lapack

Cette bibliothèque existe depuis de nombreuses années, et a été écrite en FORTRAN pour des raisons historiques et d'efficacité. Il est néanmoins possible d'appeler les routines de cette bibliothèque depuis des programmes C, C++.

Références

[1] G. H. Golub and C. F. V. Loan. *Matrix computations.* John Hopkins University Press, third edition, 1996.