

Utilisation des bibliothèques d'algèbre linéaire Blas et Lapack

Jean Hare, Jacques Lefrère (révision 2019)

14 février 2019

Table des matières

1	Introduction	2
1.1	BLAS http://www.netlib.org/blas	2
1.2	LAPACK http://www.netlib.org/lapack/	2
2	Organisation du code	3
2.1	Les types de données	3
2.2	Les classes de matrices	3
2.3	Les niveaux de procédures	3
2.4	Les règles de nommage	3
2.5	La documentation	4
3	Utilisation	4
3.1	Accès aux différentes versions installées	4
3.2	En Fortran 95	5
3.3	En C, standard C99	7
4	Installation des bibliothèques	9
4.1	Installation automatique	9
4.2	Installation manuelle	10
5	Quelques détails techniques et historiques	11
5.1	BLAS http://www.netlib.org/blas	11
5.2	LAPACK http://www.netlib.org/lapack/	11
A	Un exemple de chaîne d'appels en fortran 95	12
A.1	Code utilisateur	12
A.2	Interface générique	12
A.3	Passage au code fortran 77	12
A.4	Nouvelle genericité via le module F77_LAPACK	13
A.5	Le code lui-même en fortran 77	13
A.6	Résumé du chaînage des appels	13
B	Stockage partiel ou compact des matrices creuses	14
B.1	Stockage «complet» (Full storage)	14
B.2	Stockage «compact» (Packed storage)	14
B.3	Stockage «bande» (Band storage)	14
C	Référence de Blas	15
D	Référence de Lapack	17

1 Introduction

L'algèbre linéaire est très souvent utilisée pour le calcul scientifique. Plutôt que de développer des codes fondés sur des algorithmes classiques (qui sont néanmoins présentés en cours), avec une efficacité (très) modérée, nous avons choisi de nous appuyer sur des bibliothèques standard, dont l'usage est extrêmement répandu. Il s'agit ici de deux bibliothèques respectivement dénommées BLAS (pour *Basic Linear Algebra Subprograms*) et LAPACK (pour *Linear Algebra PACKage*). Comme ces noms le suggèrent, la bibliothèque BLAS fournit des sous-programmes de manipulations des vecteurs et matrices qui sont autant des «briques de base» pour les fonctions de plus haut niveau, que l'on trouve par exemple dans LAPACK.

Ces deux bibliothèques sont très souvent intégrées dans des logiciels tiers, comme PYTHON (NUMPY), MATLAB, SCILAB, MATHEMATICA, MAPLE, OCTAVE, O-MATRIX, FREEMAT, RLAB, MACSYMA, ou encore les bibliothèques IMSL, GSL *etc.* Des versions adaptées et/ou optimisées sont aussi proposées dans les bibliothèques vendues avec les compilateurs Intel, Sun, NAG, Lahey *etc.*

Sans se substituer aux innombrables références sur le sujet, ce document est une introduction minimale permettant de faire les premiers pas dans leur utilisation, et comporte quelques indications sur leur installation.

Nous présentons ici l'usage d'une implémentation open-source permettant néanmoins une très bonne optimisation pour les ordinateurs individuels les plus communs. Il s'agit d'une version en C de BLAS dénommée «OpenBlas» (<http://www.openblas.net/>), couplée à l'utilisation du LAPACK de référence (<http://www.netlib.org/lapack/>) et à un ensemble d'interfaces, `lapacke` pour l'usage en C, et `blas95/lapack95` pour l'usage en Fortran 95 ou 2003.

1.1 Blas <http://www.netlib.org/blas>

La bibliothèque BLAS fournit des opérations de bas niveau pour la manipulation des vecteurs et des matrices. Elle est subdivisée en trois parties, baptisées «level 1», «level 2», «level 3», qui prennent en charge respectivement les opérations vecteur-vecteur, vecteur-matrice, et matrice-matrice.

C'est cette couche de bas niveau qui doit être optimisée en fonction de l'architecture de la machine utilisée et des jeux d'extensions étendus supportés par son CPU (pour plus de précisions, voir § 5). Comme LAPACK fait un usage intensif des fonctions de BLAS, *l'efficacité* de l'ensemble repose alors presque exclusivement sur l'optimisation de BLAS. C'est pourquoi il existe des versions optimisées vendues par les constructeurs, dont on trouvera la liste à l'URL <http://www.netlib.org/blas/faq.html#5>, et par les éditeurs de compilateurs.

Nous nous intéressons ici à l'implémentation OpenBlas¹, maintenue par Xianyi ZHANG (Chinese Academy of Sciences). Comme cette version de BLAS est écrite en C, alors que LAPACK est en Fortran 90, et que les prototypes des fonctions de BLAS sont fixés par l'antique version en Fortran 77, il est très utile de recourir à des modules d'interface pour s'abstraire des problèmes d'interopérabilité.

1.2 Lapack <http://www.netlib.org/lapack/>

La bibliothèque LAPACK est aujourd'hui en Fortran 90. Fondée en grande partie sur les procédures de BLAS, cette bibliothèque réalise de nombreuses opérations matricielles, comme la résolution d'équations linéaires, l'ajustement par les moindres carrés linéaires, la diagonalisation et autres opérations spectrales, et la décomposition en valeurs singulières, ainsi que de nombreuses opérations de factorisation de matrices qui sont souvent des intermédiaires pour les opérations précitées.

1. OpenBlas est une évolution de GotoBlas2, créé par Kazushige GOTO.

2 Organisation du code

2.1 Les types de données

Les manipulations de BLAS/LAPACK peuvent opérer sur 4 types de données, à savoir les réels en simple et double précision (32 et 64 bits), et les complexes en simple et double précision. Ces 4 types sont gérés par des procédures portant le même nom, à l'exception de la première lettre qui désigne le type de données : **S** pour simple, **D** pour double, **C** pour complexe, **Z** pour double-complexe.

2.2 Les classes de matrices

L'efficacité des opérations matricielles (voire leur simple possibilité) et l'algorithme utilisé dépendent fortement des propriétés que présentent certaines matrices, qu'elles soient triangulaires, symétriques, positives, hermitiennes, unitaires ou orthogonales, *etc.* ou encore rien de tout cela (auquel cas elles sont dites «générales»). Aussi y a-t-il dans LAPACK des procédures adaptées à chaque cas particulier, qu'il incombe à l'utilisateur de choisir en fonction de ce qu'il sait *a priori* des matrices qu'il manipule.

2.3 Les niveaux de procédures

Il y a essentiellement deux niveaux de procédures :

- les procédures de calcul («computationnelles») qui effectuent des opérations élémentaires, comme la décomposition *LU* ou l'inversion d'une matrice triangulaire, et que l'utilisateur aura rarement besoin d'appeler ;
- les procédures pilotes («drivers») de plus haut niveau qui effectuent des opérations complexes en appelant les procédures de calcul et/ou les fonctions de BLAS.

Notons toutefois que même les procédures pilotes sont spécifiques en type de données et en classe de matrice : contrairement à ce que font MATHEMATICA ou MAPLE, aucune procédure ne fera un test sur le caractère symétrique de telle ou telle matrice.

2.4 Les règles de nommage

Compte tenu de la diversité décrite ci-dessus, il est essentiel de fixer et de connaître une règle de nommage des procédures (il y en a plus de 950!).

Le nom de chaque procédure de LAPACK ou de BLAS est construit sur la base de 6 caractères comme *XYZZZ*², le sixième caractère étant parfois manquant :

- Le premier caractère **X** est, comme indiqué plus haut, **S**, **D**, **C** ou **Z**, et indique le type de données.
- Les deux caractères suivants **YY** indiquent le type de matrices concernées comme suit³ :

BD	Bidiagonale	PO	Symétrique/Hermitique, défin. pos.
DI	Diagonale	PP	Symétrique/Hermitique, défin. pos., compacté
GB	Générale bande	PT	Symétrique/Hermitique, défin. pos., tridiag.
GE	Générale	SB	Symétrique bande (S ou D)
GG	Matrices générales, pb généralisé	SP	Symétrique, compacté
GT	Générale tridiagonale	ST	Symétrique tridiagonale (S ou D)
HB	Hermitique (C ou Z)	SY	Symétrique
HE	Hermitique bande (C ou Z)	TB	Triangulaire bande
HG	Hessenberg triang. sup., pb généralisé	TG	Triangulaires, pb généralisé
HP	Hermitique, compacté (C ou Z)	TP	Triangulaire, compacté
HS	Hessenberg triangulaire sup.	TR	Triangulaire, ou quasi
OP	Orthogonale compacté (S ou D)	TZ	Trapézoïdale
OR	Orthogonale (S ou D)	UN	Unitaire (C ou Z)
PB	Symétrique/Hermitique, défin. pos., bande	UP	Unitaire, compacté (C ou Z)

2. C'était le maximum en Fortran 77.

3. Une matrice de type Hessenberg supérieure est une matrice carrée «quasi-triangulaire», c'est à dire qui ne contient que des zéros au dessous de la première sous-diagonale http://en.wikipedia.org/wiki/Hessenberg_matrix

- Enfin les trois derniers caractères **ZZZ** désignent le type d'opération réalisée. Les cas sont trop nombreux pour être tous exposés, et une documentation devra être consultée. Donnons toutefois le nom des procédures pilotes en fonction de leur usage (noter les doublons) :

SV(X)	Équations linéaires
LS, LSY , LSS, LSD	Moindres carrés
LSE, GLM	Moindres carrés généralisés
EV(X), EVD, EVR	Éléments propres, cas symétrique
ES(X), EV(X)	Éléments propres, cas non-symétrique (YY=GE)
SVD, SDD	Décomposition en valeurs singulières (YY=GE)
GV(X), GVD	Éléments propres, cas symétrique défini, généralisé
ES(X), EV(X)	Éléments propres, cas non-symétrique, généralisé (YY=GG)
GV(X), GVD, ES(X), EV(X), SVD	Décomposition en valeurs singulières, généralisé

NB1 : dans le premier tableau, «compacté» (“packed”) signifie que la symétrie de la matrice «creuse» a été mise à profit pour stocker ses éléments dans un vecteur de dimension inférieure à $N \times M$ (voir l'annexe A p. 14 pour la définition de ce format).

NB2 : Les problèmes «généralisés» sont des problèmes à deux matrices, où la seconde matrice soit ajoutée des contraintes, soit remplace l'identité dans l'équation à résoudre.

NB3 : Lorsque un **X** apparaît entre parenthèses, c'est qu'il y a deux procédures pilotes, la version «simple», avec un nombre réduit de paramètres, et la version «experte» qui donne un meilleur contrôle au prix d'un nombre accru de paramètres.

2.5 La documentation

Dans cette foison de procédures, il est nécessaire de disposer d'une documentation technique précise, qui permette (1) de sélectionner la bonne fonction, (2) de connaître son prototype et la nature précise de ses arguments. Dans le cas du C, une consultation des fichiers d'entête sur la machine de travail est indispensable pour connaître avec exactitude son prototype.

On pourra recourir aux ressources suivantes :

- Sur le site Netlib, la documentation détaillée officielle hypertexte de LAPACK, avec des liens vers le code, et une description précise des arguments et le graphe des appels.
<http://www.netlib.org/lapack/explore-html/index.html>
- Sur le site d'Intel, la documentation de la MKL d'Intel, comporte des aides sur BLAS et LAPACK
<https://software.intel.com/en-us/articles/intel-math-kernel-library-documentation/>
- Sur le site de Oracle, une liste complète des fonctions de la *Sun performance library* contenant les fonctions de BLAS et de LAPACK, donne explicitement les prototypes C (sans les préfixes `cblas_` ou `LAPACKE_` associé aux interfaces) et Fortan95 ;
<http://docs.oracle.com/cd/E19422-01/819-3691/> (v11, figée)
- La documentation spécifique de `lapack95` se trouve sur le site de `netlib.org`, avec une liste des procédures pilotes à l'URL <http://www.netlib.org/lapack95/L90index>.
- Enfin, on trouve en annexe de ce document les fiches «*Quick reference*» de BLAS (p. 15) et de LAPACK (p. 17) créées par Mark Gates (<http://web.eecs.utk.edu/~mgates3/>). On pourra aussi consulter sa liste des procédures des deux bibliothèques à partir de <http://www.icl.utk.edu/~mgates3/docs/lapack.html>.

3 Utilisation

3.1 Accès aux différentes versions installées

L'installation des bibliothèques produit des fichiers dépendant à la fois de la machine et du compilateur utilisés. Ils sont répartis dans deux répertoires selon la phase, compilation (1) ou édition de liens (2), dans

laquelle ils seront invoqués :

1. un répertoire `include/` contenant les modules d'interfaces (`.mod`) pour le compilateur `gfortran` et les prototypes (`.h`) pour le compilateur `gcc` ;
2. un répertoire `lib/` contenant les bibliothèques statiques (archives `.a`) et dynamiques (`.so`) pour l'éditeur de liens.

Des versions des bibliothèques ont donc été installées pour chaque machine dans des répertoires différents :

- `/home/lefrere/M1/sappli1-64/` sur le serveur `sappli1.datacenter.dsi.upmc.fr` pour travailler en `ssh` sur `sappli1` ;
- `/home/lefrere/M1/OpenSuse/` pour travailler à l'UTÈS sur les machines virtuelles `OpenSuse`.

Afin de simplifier l'accès aux bibliothèques, une variable d'environnement `REP_LAPACK` est créée lors de l'exécution de `MNI.bashrc`. Selon la machine utilisée, elle prend la valeur `/home/lefrere/M1/sappli1-64/` ou `/home/lefrere/M1/OpenSuse/`, ce qui permet d'utiliser le même fichier `makefile` sur les deux machines.

Comme les bibliothèques ne sont pas installées dans les répertoires standard, il faut compléter les chemins de recherche pour le compilateur (option `-I`) et pour l'éditeur de liens (option `-L`). Le fichier `makefile` doit donc contenir les options suivantes, la variable d'environnement `REP_LAPACK` étant choisie selon la machine :

1. `-I${REP_LAPACK}/include` pour la compilation ;
2. `-L${REP_LAPACK}/lib` pour l'édition de liens.

3.2 En Fortran 95

L'utilisation de BLAS et de LAPACK en Fortran 95 ne pose pas de problème particulier. Un avantage très important de l'interface de LAPACK 95 est le recours à diverses caractéristiques du Fortran 90/95/2003 : les tableaux de taille connue, les paramètres optionnels et les fonctions génériques.

Il en résulte des caractéristiques spécifiques de LAPACK 95 :

- Les tableaux sont représentés par des descripteurs qui incluent le profil du tableau, ce qui évite de passer les nombreux paramètres de taille.
- Les tableaux utilisés comme espace temporaire de travail sont dimensionnés, alloués, et desalloués automatiquement, sans que l'utilisateur ait à s'en soucier.
- Le recours aux paramètres optionnels pour toutes les options avancées ou non standard,
- La généralité des fonctions : la même procédure est appelée pour tous les types. Les variantes en `S`, `D`, `C`, `Z` cèdent la place à une unique version, dont le nom est le même qu'en Fortran 77, hormis la lettre de type qui disparaît au profit du préfixe `LA_`. Le type (réel ou complexe) est alors précisé lors de la déclaration du tableau dans l'appelant et la variante (simple ou double) via le module `la_precision`, comme il est illustré ci-dessous.
- le paramètre de `status` est géré via une variable entière optionnelle `info`.

3.2.1 Compilation et édition de liens

1. Les interfaces des procédures génériques de LAPACK et BLAS seront rendues visibles par des instructions du type⁴ `USE f95_lapack`, `ONLY:la_gesv` ou `USE blas95`, `ONLY:gemm`. Noter que le module `la_precision` définit les constantes symboliques `sp` et `dp` comme des variantes de type réel (`kind`) pour la simple et la double précision : il peut donc être invoqué dans `mncs_precision` afin de choisir la précision de travail `wp` comme `sp` ou `dp`.

La compilation se fera avec `gfortran` (de préférence en utilisant l'option `-std=f2003`).

2. L'édition de liens se fera aussi avec `gfortran` et elle appellera *dans cet ordre* les bibliothèques suivantes : `-llapack95 -lblas95 -lopenblas -lpthread`

4. On notera l'emploi de la clause `ONLY` pour éviter une avalanche d'avertissements concernant les centaines d'autres procédures non utilisées.

3.2.2 Exemple

Le module `matop` ci-dessous définit deux opérations matricielles, la multiplication avec BLAS et l'inversion avec LAPACK.

```

                                matop.f90
1  ! exemples d'appel a Blas et Lapack via les interfaces f95
2  module matop
3      use la_precision, only: wp=>sp
4      ! definit wp (working precision) a sp (simple precision)
5      use blas95, only: gemm
6      use f95_lapack, only: la_gesv
7      implicit none
8      contains
9          ! Multiplication de deux matrices  A(M,K) . B(K,N) = C(M,N)      !
10         ! A, B, C sont des tableaux 2-D usuels du Fortran                !
11         ! multiplication par la subroutine generique GEMM de BLAS95       !
12         ! voir http://docs.oracle.com/cd/E19422-01/819-3691/sgemm.htm (Sun) !
13         ! ou https://software.intel.com/en-us/node/468480 (MKL 2017) !
14         subroutine matmulblas(A,B,C)
15             real(kind=wp), dimension(:,:), intent(in)  :: A, B
16             real(kind=wp), dimension(:,:), intent(out) :: C
17             integer:: m, k, n, kp
18             m = size(A, 1)
19             k = size(A, 2)
20             kp = size(B, 1)
21             n = size(B, 2)
22             if ((k/=kp) .or. (m/=size(C,1)) .or. (n/=size(C,2))) then
23                 write(*,*) "Erreur dans matmulblas, tailles incompatibles"
24                 stop
25             end if
26             call GEMM(A, B, C) ! generique
27         end subroutine matmulblas
28         ! Inversion de la matrice carree A(M,M)                          !
29         ! A est un tableau 2-D usuel du Fortran                          !
30         ! inversion par la subroutine generique LA_GESV de LAPACK95       !
31         ! voir http://www.netlib.org/lapack95/lug95/node72.html !
32         ! ou https://software.intel.com/en-us/node/468876 (MKL 2017) !
33         subroutine invmatlapack(A, invA)
34             ! A est utilise comme espace de travail
35             ! et contient la decomposition LU en sortie => intent(in) impossible
36             real(kind=wp), dimension(:,:), intent(inout) :: A
37             real(kind=wp), dimension(:,:), intent(out)  :: invA
38             integer:: m, i, info
39             m = size(A, 1)
40             if ( (m/=size(A,2)) .or. (m/=size(invA,1)) .or. (m/=size(invA,2))) then
41                 stop "Erreur dans invmatlapack, tailles incompatibles"
42             end if
43             invA(:, :) = 0._wp
44             do i=1, m
45                 invA(i,i) = 1._wp
46             end do
47             call LA_GESV(A, invA, info=info) ! generique
48             if (info/=0) then
49                 write(*,*) "Erreur dans invmatlapack, code", info
50                 stop
51             end if
52         end subroutine invmatlapack
53     end module matop

```

3.3 En C, standard C99

L'utilisation en C de BLAS/LAPACK est plus délicate. Certes, la version de BLAS que nous utilisons est écrite en C, mais les difficultés viennent essentiellement de deux caractères spécifiques du C par rapport au Fortran : la dissymétrie des dimensions des tableaux à 2 dimensions, et le passage des arguments par valeur, alors que le Fortran les passe par adresse ou descripteur. En outre, dans le cas de l'allocation dynamique, les données d'un tableau Fortran sont toujours dans un bloc unique, alors qu'elles peuvent être séparées dans un tableau 2D en C. C'est pourquoi on recommande de travailler en C99 avec des tableaux automatiques, qui éliminent ce problème.

En ce qui concerne les types, on peut généralement utiliser les types définis en C99, y compris les complexes, mais certains types entiers sont redéfinis pour représenter les valeurs de retour des fonctions, ou les booléens du Fortran. Pour pouvoir changer aisément de précision, on aura intérêt à utiliser un fichier d'entête `mncs_precision.h` contenant la définition `typedef float real;` pour la simple précision ou `typedef double real;` pour la double précision. La question du passage des arguments est résolue par les interfaces.

Concrètement, avec `openblas` et l'interface `lapacke` (<http://www.netlib.org/lapack/lapacke.html>), le nommage des fonctions reprend celui des fonctions des BLAS et LAPACK originaux en Fortran 77, en y adjoignant le préfixe `cblas_` pour les fonctions de BLAS, et `LAPACK_` (en majuscules et avec le `e` final) pour les fonctions de LAPACK. En revanche, il n'y a pas de généricité à proprement parler, et on devra donc choisir la fonction adaptée au type utilisé⁵.

La principale difficulté qui subsiste réside dans l'ordre de stockage des éléments dans un tableau 2D. En C, les éléments sont rangés selon l'ordre `RowMajor` c'est à dire que les éléments contigus sont ceux qui appartiennent à la même *ligne*, alors que le stockage en fortran est en `ColMajor`, c'est à dire que les éléments contigus sont ceux qui appartiennent à la même *colonne*⁶. La solution de ce problème fait l'objet des deux paragraphes qui suivent.

3.3.1 Stockage des matrices en tableau 1D

La solution **recommandée** consiste à coder systématiquement les matrices sous la forme linéaire, ou «aplatissement», c'est à dire d'un tableau unidimensionnel (automatique ou non) construit par concaténation **des colonnes** selon le schéma suivant :

si M est une matrice de taille $n_{\text{lig}} \times n_{\text{col}}$ dont les éléments s'écrivent M_{ij} où $i \in [1, n_{\text{lig}}]$ et $j \in [1, n_{\text{col}}]$, au lieu de définir un tableau `real m[nlig][ncol]` pour la stocker, on définit un «vecteur» `real m[nlig*ncol]` tel que $M_{ij} = m[k]$ où $k = (i-1) + n_{\text{lig}} * (j-1)$, variant entre 0 et $n_{\text{lig}} * n_{\text{col}} - 1$. Cette solution permet de stocker les éléments dans l'ordre `ColMajor` attendu par BLAS et LAPACK.

NB : On pourra avoir intérêt à définir une fonction auxiliaire `int icm(nlig,ncol,i,j)` qui calcule l'indice k , en sorte que $M_{ij} = m[\text{icm}(n_{\text{lig}}, n_{\text{col}}, i, j)]$.

Si l'on opte pour cette solution, les fonctions de BLAS/LAPACK seront appelées avec comme premier argument la constante `CblasColMajor`, et les paramètres de profil de matrice seront conformes à ceux utilisés en Fortran, et tels que décrits dans la documentation.

3.3.2 Stockage des matrices en tableaux 2D (tableaux de tableaux)

Une autre approche possible consiste à coder la matrice M_{ij} sous la forme d'un tableau **automatique** `real m[nlig][ncol]`. Les éléments seront alors rangés sous la forme `RowMajor`. Cela se traduit par les modifications suivantes :

- L'appel des fonctions de BLAS/LAPACK comportera en premier argument la constante `CblasRowMajor`.

5. On peut éventuellement faire un test sur `sizeof(real)` comme dans l'exemple présenté plus bas.

6. En informatique, la notion de lignes et de colonnes n'est pas vraiment définie, mais nous adoptons ici la convention mathématique selon laquelle le premier indice est celui des lignes et le second celui des colonnes.

- Lors de l'appel des fonctions, les tableaux devront être convertis en tableau 1D par un «cast» (`real *`) `m`.
- Les paramètres de nombre de colonnes ou de lignes, ou de «leading dimension» (LDx dans la documentation) seront *généralement* échangés, et donc non-conformes à l'essentiel de la documentation.
- En interne, les procédures commenceront toujours par transposer la matrice avant d'appeler les fonctions, ce qui peut être très pénalisant en termes de ressources et/ou d'efficacité.

3.3.3 Compilation et édition de liens

1. Les directives `#include "cblas.h"` et `#include "lapacke.h"` permettent de déclarer les prototypes des fonctions invoquées. La compilation avec `gcc` se fera en utilisant l'option `-std=c99`.
2. L'édition de liens se fera de préférence avec `gfortran`⁷ en appelant *dans cet ordre* les bibliothèques suivantes : `-lopenblas -lpthread -lm` (rappelons que `openblas` intègre aujourd'hui LAPACK).

3.3.4 Exemple

Le module ci-dessous définit deux opérations matricielles, la multiplication à l'aide de BLAS et l'inversion à l'aide de LAPACK.

```

----- matop.h -----
1 void matmulblas(const int m, const int k, const int n,
2                 real *A, real *B, real *C) ;
3 void invmatlapack ( const int n , real *A , real * invA ) ;
-----

----- matop.c -----
1 /*===== Module matop.c =====*/
2 /* exemple d'operations matricielles avec Blas/Lapack      */
3 /* version recommandee en ColMajor avec matrices aplaties */
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include "cblas.h"
7 #include "lapacke.h"
8 #include "mncs_precision.h"
9 // mncs_precision.h contient typedef float real; ou typedef double real;
10 #include "matop.h"
11
12 /* Multiplication de deux matrices  A[m*k] . B[k*n] = C[m*n]      */
13 /* multiplication par BLAS xGEMM,                                  */
14 /* cf http://docs.oracle.com/cd/E19422-01/819-3691/sgemm.htm (Sun) */
15 /* cf https://software.intel.com/en-us/node/520775 (MKL 2017)     */
16
17 void matmulblas(const int m, const int k, const int n,
18                 real *A, real *B, real *C) {
19     if (sizeof(real)==sizeof(float)) { //selection du type utilisé : float
20         cblas_sgemm(CblasColMajor, CblasNoTrans, CblasNoTrans,
21                    m, n, k, 1.0f, (float*) A, m, (float*)B, k, 0.0f, (float*)C,m) ;
22     }
23     else { //selection du type utilisé : double
24         cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans,
25                    m, n, k, 1.0, (double*)A, m, (double*)B, k, 0.0, (double*)C,m) ;
26     }
27 }
28

```

7. Si l'on confie l'édition de liens à `gcc`, il pourra être nécessaire d'ajouter `-lgfortran`, et pour que le compilateur puisse trouver cette bibliothèque, une directive `-L$(pathtolibgfortran)` où la macro `pathtolibgfortran` donnera le chemin approprié; en général, c'est un sous répertoire de `/usr/lib/gcc/`. Mais le plus simple est de confier l'édition de liens à `gfortran`, qui connaît cette bibliothèque.

```

29 /* Inversion d'une matrice carrée A[n*n] par LAPACKE_xgesv */
30 /* cf http://docs.oracle.com/cd/E19059-01/stud.10/819-0497/sgesv.html */
31 /* cf https://software.intel.com/en-us/node/520973 (MKL 2017) */
32 void invmatlapack ( const int n , real *A , real * invA ) {
33     int i , k , info ;
34     int * ipiv ; // en retour, indique des lignes qui ont occasionné un pivotement
35     ipiv = calloc ( n , sizeof (int) ) ;
36     for ( k =0; k < n * n ; k ++ ) { //construction de l'identité aplatie
37         invA [ k ] = (real)(0.0) ;
38     }
39     for ( i =0; i < n ; i ++ ) { // diagonale de l'identité
40         invA [ i*(n+1) ] = (real)(1.0);
41     }
42     /* -----
43        extrait du fichier d'entete lapacke.h avec la commande unix suivante :
44        grep -A 2 'LAPACKE_[sd]gesv(' ${REP_LAPACK}/include/lapacke.h
45     lapack_int LAPACKE_sgesv( int matrix_layout, lapack_int n, lapack_int nrhs,
46                             float* a, lapack_int lda, lapack_int* ipiv, float* b,
47                             lapack_int ldb );
48     lapack_int LAPACKE_dgesv( int matrix_layout, lapack_int n, lapack_int nrhs,
49                             double* a, lapack_int lda, lapack_int* ipiv,
50                             double* b, lapack_int ldb );
51     ----- */
52     // en retour de [sd]gesv, B=invA contient l'inverse de A,
53     // et A est remplacé par sa factorisation LU
54     if (sizeof(real)==sizeof(float)) { // selection du type utilisé : float
55         info = LAPACKE_sgesv (CblasColMajor, n , n , (float*) A , n ,
56                             ipiv , (float*) invA , n);
57     }
58     else { // selection du type utilisé : double
59         info = LAPACKE_dgesv (CblasColMajor, n , n , (double*) A , n ,
60                             ipiv , (double*) invA , n);
61     }
62     if ( info != 0 ) {
63         fprintf ( stderr , " Erreur dans invmatlapack code =% d \n " , info ) ;
64         exit ( EXIT_FAILURE ) ;
65     }
66     free(ipiv); ipiv = NULL; // ipiv est utile si on exploite la decomposition LU
67 }

```

4 Installation des bibliothèques

4.1 Installation automatique

L'installation pourra être réalisée quasi-automatiquement sous Linux32/64 grâce à quelques outils à télécharger, en particulier le script `InstallOpenBlas.sh`. Elle ne nécessite pas les droits d'administrateur sur la machine : elle se fera donc sous votre répertoire d'accueil ou tout répertoire accessible en écriture. En contre-partie, il faudra informer le compilateur et l'éditeur de liens des chemins particuliers permettant d'utiliser cette bibliothèque. Elle comporte une phase de compilation à partir des sources, puis une phase d'installation proprement dite qui consiste à déposer au bon endroit les fichiers créés. On distinguera donc :

- le répertoire de construction, où seront compilées les bibliothèques,
- et le répertoire d'installation, où seront rendus disponibles les bibliothèques et les interfaces.

Le répertoire de compilation pourra être supprimé une fois l'installation terminée. Il est donc recommandé de ne pas placer le répertoire d'installation dans celui de construction.

1. Téléchargement préliminaire

Créer un répertoire de construction (`build-lapack` par exemple) et en faire votre répertoire de travail. Y télécharger l'archive `OpenBlasInstall.tgz` depuis la page web de l'UE

```
wget "http://wwwens.aero.jussieu.fr/lefrere/master/mni/mnacs/OpenBlasInstall.tgz"
```

La dézipper et extraire les fichiers dans le répertoire de construction (`build-lapack`)

```
tar -xvzf OpenBlasInstall.tgz
```

2. Configuration

Choisir le répertoire (`lapack` par exemple) où vous souhaitez finalement installer la bibliothèque. Éditer le fichier `OpenBlasInstall/installpath.inc` pour indiquer le chemin absolu de ce répertoire (la procédure pourra le créer s'il n'existe pas encore).

C'est ce chemin que devra comporter la variable d'environnement `REP_LAPACK` pour pouvoir utiliser directement les fichiers `makefile` proposés. Pour ce faire, il suffira d'insérer dans le fichier personnel d'initialisation `~/.bashrc` la ligne suivante :

```
REP_LAPACK=${HOME}/lapack ; export REP_LAPACK
```

si le répertoire choisi est `~/lapack/`

On peut aussi définir dans ce fichier des options spécifiques pour certaines architectures via la variable `LOCAL_BUILD_FLAGS`. Par exemple, `LOCAL_BUILD_FLAGS="NO_AVX2=1"` permet de ne pas activer les possibilités `AVX2` détectées avec le processeur, mais qui sont encore mal gérées par le compilateur.

3. Compilation et installation

Copier le script `InstallOpenBlas.sh` de `OpenBlasInstall` dans le répertoire de construction (`build-lapack`)

```
/bin/cp -f ./OpenBlasInstall/InstallOpenBlas.sh ./
```

Lancer ce script en dupliquant les messages à l'écran et dans le fichier `install.log` :

```
./InstallOpenBlas.sh 2>&1 | tee install.log
```

Le script va successivement :

- (a) Créer le répertoire d'installation si nécessaire ;
- (b) Télécharger les archives de `openblas`⁸, `lapack95` et `blas95`. Les fichiers d'archives seront stockés dans le sous-répertoire `network-files`.
- (c) Extraire les fichiers de ces archives ;
- (d) Copier les fichiers complémentaires fournis via `OpenBlasInstall.tgz` ;
- (e) Lancer la compilation de `openblas`, `lapack` et `lapacke`. C'est la phase la plus longue de l'installation, qui peut durer plusieurs minutes, voire dizaines de minutes. Elle comporte aussi les tests.
- (f) Installer la bibliothèque `openblas` (qui inclut `lapack` et `lapacke`) ;
- (g) Lancer la compilation de `lapack95` puis l'installer ;
- (h) Lancer la compilation de `blas95` puis l'installer.

4.2 Installation manuelle

En cas de difficulté en cours de compilation par exemple, il sera possible de reprendre à la main le cheminement proposé par la procédure `InstallOpenBlas.sh`, sans avoir à refaire les téléchargements.

Entrer dans le répertoire `openblas` et exécuter successivement les cibles du fichier `myMakefile` :

```
cd openblas
make -f myMakefile build           # compilation openblas, lapack, lapacke (cf. 3e)
make -f myMakefile install        # installation openblas, lapack, lapacke (cf. 3f)
make -f myMakefile lapack95       # compilation lapack95 (cf. 3g)
make -f myMakefile lapack95install # installation lapack95 (cf. 3g)
make -f myMakefile blas95         # compilation blas95 (cf. 3h)
make -f myMakefile blas95install  # installation blas95 (cf. 3h)
```

⁸. Le script `InstallOpenBlas.sh` appelle pour cela `download-tgz.sh` qui, dans l'édition 2019, télécharge la version 0.3.5 de `openblas`, datant du 31/12/2018, qui utilise `lapack v3.8` (qui date du 12/11/2017).

5 Quelques détails techniques et historiques

Développées initialement dans les années 1970-80, à l'*University of Tennessee* sous l'égide de la *National Science Foundation*, les bibliothèques Blas et Lapack relèvent du domaine public. De ce fait elles sont très souvent intégrées dans des logiciels tiers, comme MATLAB, SCILAB, MATHEMATICA, MAPLE, OCTAVE, O-MATRIX, FREEMAT, RLAB, MACSYMA, ou encore les bibliothèques IMSL, GSL *etc.* Des versions adaptées et/ou optimisées sont aussi intégrées dans les bibliothèques fournies avec les compilateurs Intel, Sun, NAG, Lahey *etc.* Le développement de Lapack a permis d'adapter les algorithmes déjà présents dans LINPACK et EISPACK à des processeurs vectoriels et parallèles en tirant parti des performances de BLAS dans ce domaine.

Ces bibliothèques ont initialement été développées en FORTRAN 77, qui n'est plus guère utilisé. Il existe donc de nombreuses variantes pour permettre leur utilisation dans des langages plus modernes, dont notamment le FORTRAN 95/2003 et le C89/99 qui nous intéressent ici⁹, mais la matrice initiale continue d'imposer une certaine structure aux appels de fonctions et à leurs arguments, rendant nécessaire (voire indispensable) le recours à des interfaces.

5.1 Blas <http://www.netlib.org/blas>

La bibliothèque Blas, initialement écrite en FORTRAN77, fournit des opérations de bas niveau pour la manipulation des vecteurs et des matrices. Elle est subdivisée en trois parties, baptisées «level 1», «level 2», «level 3», qui prennent en charge respectivement les opérations vecteur-vecteur, vecteur-matrice, et matrice-matrice.

C'est cette couche de bas niveau qui doit être optimisée en fonction de l'architecture de la machine utilisée et des jeux d'extensions étendus supportés par son CPU. Cette optimisation délicate peut être obtenue de différentes façons, qui ne sont pas toutes documentées. Comme LAPACK (ou ses diverses extensions) fait un usage intensif des fonctions de BLAS (et/ou de ses extensions), *l'efficacité* de l'ensemble repose alors presque exclusivement sur l'optimisation de BLAS. Aussi existe-t-il des versions optimisées vendues par les constructeurs, dont on trouvera la liste à l'URL <http://www.netlib.org/blas/faq.html#5>, et par les éditeurs de compilateurs. Mais nous nous concentrerons ici sur une solution *open-source*, et relativement portable.

L'optimisation en fonction de l'architecture de la machine repose toujours sur la gestion de la mémoire, qui est le principal facteur limitant lorsqu'on manipule des structures de données importantes.

- Une approche fructueuse consiste à optimiser l'utilisation rationnelle de cache mémoire, en laissant par exemple l'une des deux matrices (ou à défaut un sous-bloc de celle-ci) dans le cache de niveau 1 (L1). Cette approche est celle suivie par le projet ATLAS (pour «Automatically Tuned Linear Algebra Software» <http://sourceforge.net/projects/math-atlas/>), qui repose sur une version des `cblas` (c'est à dire une ré-implémentation de `blas` en langage C) qui est (laborieusement) compilée avec toutes les options possibles pour obtenir la version la plus efficace sur la machine sur laquelle on construit la bibliothèque.
- Une autre approche, tout en tenant compte bien sûr de la taille limitée des caches L1 et L2, se concentre principalement sur la gestion des portions plus éloignées de la mémoire : dans les processeurs modernes la mémoire est virtualisée, et la conversion entre adresses virtuelles et adresses physiques est réalisée sur la base d'un cache (TLB=Translation lookaside buffer) pour la traduction rapide ; en cas d'accès à des emplacements qui ne sont pas dans ce cache, on est conduit à une recherche beaucoup plus lente, qui consomme du temps CPU. C'est donc l'utilisation du TLB qui est optimisée. Cette approche est celle suivie par Kazushige Goto (Texas Advanced Computing Center) dans le développement d'une version de `cblas` écrite en assembleur, sous le nom de GotoBLAS (<https://www.tacc.utexas.edu/research-development/tacc-software/gotoblas2>). La dernière version de GotoBLAS est maintenant maintenue par Xianyi Zhang (Institute of Software, Chinese Academy of Sciences) sous le nom de OpenBLAS (<http://www.openblas.net/>).

C'est cette dernière version que nous utiliserons. Elle comporte maintenant `lapack` et `lapacke`, donc tous les outils pour le C, dans une seule bibliothèque.

5.2 Lapack <http://www.netlib.org/lapack/>

Développée initialement en FORTRAN 77, la bibliothèque LAPACK est aujourd'hui en FORTRAN 90 (elle ne se compile donc plus avec `g77`). Fondée en grande partie sur les procédures de BLAS, cette bibliothèque réalise de nombreuses opérations matricielles, comme la résolution d'équations linéaires, l'ajustement par les moindres carrés linéaires, la diagonalisation et autres opérations spectrales, et la décomposition en valeurs singulières, ainsi que de nombreuses opérations de factorisation de matrices qui sont souvent des intermédiaires pour les opérations précitées.

9. Il existe des versions C++, C#, Java, Python, *etc*

A Un exemple de chaîne d'appels en fortran 95

Pour illustrer l'usage de la généricité et des arguments optionnels du fortran 95, on étudie le début de la séquence d'appels déclenchés par l'appel à `la_syev` pour la recherche des éléments propres d'une matrice réelle symétrique.

A.1 Code utilisateur

Le code utilisateur appelle le sous-programme générique `syev`. Il comporte les instructions suivantes

```
use la_precision, only: wp=>sp ! choix de la simple précision
use f95_lapack, only : la_syev
...
call la_syev(...)
```

avec au minimum les deux arguments obligatoires définissant la matrice et le second membre. Même si le code de retour est optionnel, on prendra soin de toujours le récupérer et de le tester.

A.2 Interface générique

La bibliothèque `lapack95` comporte, dans `openblas/LAPACK95/SRC/`, un fichier source déclarant dans le module `F95_LAPACK` les interfaces génériques des procédures spécifiques. Dans le cas où les quatre variantes de types ont été compilées, il s'agit du fichier `f95_lapack_single_double_complex_dcomplex.f90` (réel simple et double précision et de même pour les complexes). Mais pour la procédure `LA_SYEV`, seuls les deux sous-programmes `SSYEV_F95` et `DSYEV_F95` correspondant aux deux variantes de réels sont déclarés :

```
MODULE F95_LAPACK
...
INTERFACE LA_SYEV
SUBROUTINE SSYEV_F95( A, W, JOBZ, UPLO, INFO )
  USE LA_PRECISION, ONLY: WP => SP
  CHARACTER(LEN=1), INTENT(IN), OPTIONAL :: JOBZ, UPLO
  INTEGER, INTENT(OUT), OPTIONAL :: INFO
  REAL(WP), INTENT(INOUT) :: A(:, :)
  REAL(WP), INTENT(OUT) :: W(:)
END SUBROUTINE SSYEV_F95
SUBROUTINE DSYEV_F95( A, W, JOBZ, UPLO, INFO )
  USE LA_PRECISION, ONLY: WP => DP
  CHARACTER(LEN=1), INTENT(IN), OPTIONAL :: JOBZ, UPLO
  INTEGER, INTENT(OUT), OPTIONAL :: INFO
  REAL(WP), INTENT(INOUT) :: A(:, :)
  REAL(WP), INTENT(OUT) :: W(:)
END SUBROUTINE DSYEV_F95
END INTERFACE
...
END MODULE F95_LAPACK
```

A.3 Passage au code fortran 77

Dans `openblas/LAPACK95/SRC/la_ssyev.f90` est défini le sous-programme `SSYEV_F95`, qui n'est qu'un «wrapper» vers `SYEV_F77`, plus proche du standard fortran 77. Au préalable, `SSYEV_F95` calcule les tailles des tableaux, gère les arguments optionnels et l'allocation dynamique d'espace de travail.

Noter enfin que `SYEV_F77` n'est qu'un renommage local de `LA_SYEV` du module `F77_LAPACK`, destiné à éviter une collision avec le nom générique.

```
SUBROUTINE SSYEV_F95( A, W, JOBZ, UPLO, INFO )
  USE LA_PRECISION, ONLY: WP => SP
  USE LA_AUXMOD, ONLY: ERINFO, LSAME
  USE F77_LAPACK, ONLY: SYEV_F77 => LA_SYEV, ILAENV_F77 => ILAEN
```

```
...
CALL SYEV_F77( LJOBZ, LUPLO, N, A, LD, W, WORK, LWORK, LINFO )
```

A.4 Nouvelle généricité via le module F77_LAPACK

Dans le fichier `f77_lapack_single_double_complex_dcomplex.f90`, au sein du module `F77_LAPACK`, est déclarée l'interface générique `LA_SYEV` des sous-programmes `SSYEV` et `DSYEV` spécifiques de chaque sous-type.

```
MODULE F77_LAPACK
...
INTERFACE LA_SYEV
  SUBROUTINE SSYEV( JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, INFO )
    USE LA_PRECISION, ONLY: WP => SP
    CHARACTER(LEN=1), INTENT(IN) :: JOBZ, UPLO
    INTEGER, INTENT(IN) :: LDA, LWORK, N
    INTEGER, INTENT(OUT) :: INFO
    REAL(WP), INTENT(INOUT) :: A(LDA,*)
    REAL(WP), INTENT(OUT) :: W(*)
    REAL(WP), INTENT(OUT) :: WORK(*)
  END SUBROUTINE SSYEV
  SUBROUTINE DSYEV( JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, INFO )
    USE LA_PRECISION, ONLY: WP => DP
    CHARACTER(LEN=1), INTENT(IN) :: JOBZ, UPLO
    INTEGER, INTENT(IN) :: LDA, LWORK, N
    INTEGER, INTENT(OUT) :: INFO
    REAL(WP), INTENT(INOUT) :: A(LDA,*)
    REAL(WP), INTENT(OUT) :: W(*)
    REAL(WP), INTENT(OUT) :: WORK(*)
  END SUBROUTINE DSYEV
END INTERFACE
...
END MODULE F77_LAPACK
```

Dans notre cas, cela signifie que l'on appellera en fait `SSYEV`

A.5 Le code lui-même en fortran 77

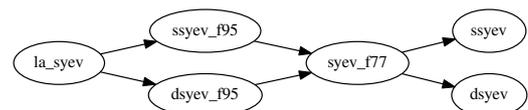
Le sous programme `SSYEV` qui effectue les calculs (via d'autres appels) n'est plus une procédure de module, mais une procédure externe, à la fortran 77, dont le source est situé dans `openblas/lapack-netlib/SRC/ssyev.f`. Noter que le code objet de ce sous-programme est placé dans la bibliothèque `openblas`.

```
SUBROUTINE SSYEV( JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, INFO )
```

A.6 Résumé du chaînage des appels

Tous les sous-programmes de cette chaîne ne sont que des interfaces pour arriver au code fortran 77 qui se charge du calcul : à cet effet, il appelle d'autres procédures LAPACK non représentées ici, qui elles-mêmes appellent des procédures BLAS de bas niveau.

Début du chaînage des appels successifs



Dans le cas de la simple précision, le cheminement devient : `la_syev` du module `f95_lapack` → (interface générique) → `SSYEV_F95` spécifique → `call SYEV_F77` `SYEV_F77` en fait déclaré `la_syev`, mais dans le module `f77_lapack` → (interface générique) → `SSYEV` spécifique définition de `SSYEV` dans `lapack` version `f77` : fichier `ssyev.f`

B Stockage partiel ou compact des matrices creuses

Les matrices sont dites «creuses» lorsqu’elles contiennent une fraction significative de zéros. Pour économiser des ressources (mémoire ou temps de calcul), il est parfois utiles de ne pas stocker tous ces zéros, mais de recourir à un stockage plus dense. On définit ainsi les trois schémas de stockage suivants (en plus du stockage normal).

B.1 Stockage «complet» (Full storage)

Ce schéma est utilisé pour des matrices carrées réelles symétriques, complexes hermitiennes, ou des matrices triangulaires, et ne stocke que la moitié de la matrice (plus exactement $n(n + 1)/2$ éléments si n est la taille de la matrice) tout en conservant un conteneur sous forme de tableau «carré». Une variable `uplo` prenant la valeur 'U' ou 'L' est alors utilisée pour indiquer si c’est la partie supérieure (up) ou inférieure (low) de la matrice qui est stockée, de la façon suivante :

- U : les éléments $M_{i,j}$ pour $i \leq j$ sont stockés dans la matrice $M(i, j)$ et les autres éléments ignorés ;
- L : les éléments $M_{i,j}$ pour $i \geq j$ sont stockés dans la matrice $M(i, j)$ et les autres éléments ignorés ;

Compte tenu des problèmes d’arrangement des éléments, on évitera d’utiliser cette méthode en langage C.

B.2 Stockage «compact» (Packed storage)

Ce schéma est lui aussi utilisé pour le même type de matrices et utilise aussi la variable `uplo`. Il utilise par contre un conteneur 1D de taille $n(n + 1)/2$ structuré de la façon suivante (en `ColMajor!`) :

- U : `m(k)` contient $M_{i,j}$ pour $i \leq j$ avec $k = i + j(j - 1)/2$ pour $0 < i \leq j$
- L : `m(k)` contient $M_{i,j}$ pour $i \leq j$ avec $k = i + (2n - j)(j - 1)/2$ pour $0 < j \leq i$.

NB : Expressions en indices «mathématiques», coïncidant avec ceux du Fortran. Ceux du C sont décalés. Les procédures ou fonctions utilisant de telles matrices ont un nom commençant par `p`.

B.3 Stockage «bande» (Band storage)

Ce schéma s’applique bien sûr aux matrices «bandes», c’est à dire ne comportant que des zéros au delà d’une certaine distance de la diagonale. Plus spécifiquement, si les éléments non-nuls de M de profil $n \times n$ sont dans la diagonale, dans $kl < n$ sous-diagonales et dans $ku < n$ super-diagonales, on utilise un conteneur rectangulaire comportant $kl + ku + 1$ lignes et n colonnes, chaque sous-ou-super-diagonale étant stockée sur une ligne de la matrice, en conservant l’alignement *par colonne* des éléments ($m_{i,j}$ est au dessus de $m_{i+1,j}$ et ainsi de suite).

On peut représenter cette forme de stockage par le dessin suivant, pour $n = 6$, $ku = 1$ et $kl = 2$:

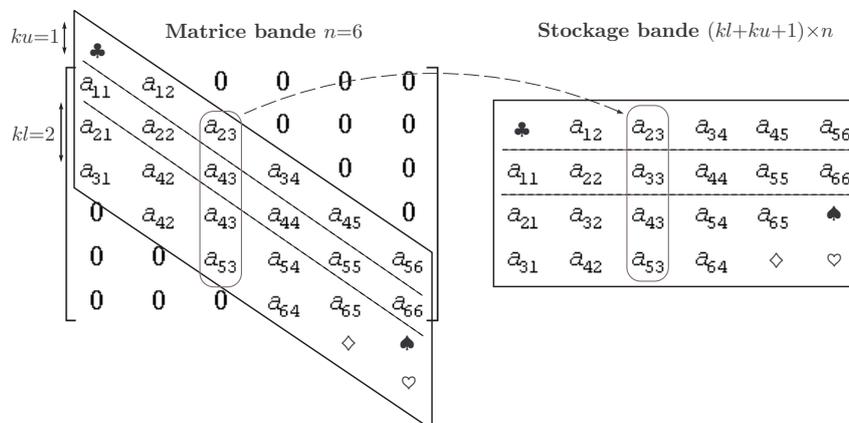


FIGURE 1 – Stockage bande : les symboles représentent des éléments non utilisés

Les procédures ou fonctions utilisant ce schéma ont un nom commençant par `b`.

BLAS Fortran 77 prototypes

Level 1 BLAS: vector, $O(n)$ operations

precisions	name	(size arguments)	description	equation	flops : data
s, d, c, z	axpy	(n, alpha, x, incx, y, incy)	update vector	$y = y + \alpha x$	$2n : 2n$
s, d, c, z, cs, zd	scal	(n, alpha, x, incx)	scale vector	$y = \alpha y$	$n : n$
s, d, c, z	copy	(n, x, incx, y, incy)	copy vector	$y = x$	$0 : 2n$
s, d, c, z	swap	(n, x, incx, y, incy)	swap vectors	$x \leftrightarrow y$	$0 : 2n$
s, d	dot	(n, x, incx, y, incy)	dot product	$= x^T y$	$2n : 2n$
c, z	dotu	(n, x, incx, y, incy)	(complex)	$= x^T y$	$2n : 2n$
c, z	dotc	(n, x, incx, y, incy)	(complex conj)	$= x^H y$	$2n : 2n$
sds, ds	dot	(n, x, incx, y, incy)	(internally double precision)	$= x^T y$	$2n : 2n$
s, d, sc, dz	nrm2	(n, x, incx)	2-norm	$= \ x\ _2$	$2n : n$
s, d, sc, dz	asum	(n, x, incx)	1-norm	$= \ \text{Re}(x)\ _1 + \ \text{Im}(x)\ _1$	$n : n$
s, d, c, z	i_amax	(n, x, incx)	∞ -norm	$= \text{argmax}_i (\text{Re}(x_i) + \text{Im}(x_i))$	$n : n$
s, d, c, z	rotg	(a, b, c, s)	generate plane (Given's) rotation (c real, s complex)		$O(1) : O(1)$
s, d, c, z †	rot	(n, x, incx, y, incy, c, s)	apply plane rotation (c real, s complex)		$6n : 2n$
cs, zd	rot	(n, x, incx, y, incy, c, s)	apply plane rotation (c & s real)		$6n : 2n$
s, d	rotmg	(d1, d2, a, b, param)	generate modified plane rotation		$O(1) : O(1)$
s, d	rotm	(n, x, incx, y, incy, param)	apply modified plane rotation		$6n : 2n$

Level 2 BLAS: matrix-vector, $O(n^2)$ operations

precisions	name	(options size arguments)	description	equation	flops : data
s, d, c, z	gemv	(trans, m, n, alpha, A, ldA, x, incx, beta, y, incy)	general matrix-vector multiply	$y = \alpha A^* x + \beta y$	$2mn : mn$
c, z	hemv	(uplo, n, alpha, A, ldA, x, incx, beta, y, incy)	Hermitian matrix-vector mul.	$y = \alpha Ax + \beta y$	$2n^2 : n^2/2$
s, d †	symv	(uplo, n, alpha, A, ldA, x, incx, beta, y, incy)	symmetric matrix-vector mul.	$y = \alpha Ax + \beta y$	$2n^2 : n^2/2$
s, d, c, z	trmv	(uplo, trans, diag, n, A, ldA, x, incx)	triangular matrix-vector mul.	$x = A^* x$	$n^2 : n^2/2$
s, d, c, z	trsv	(uplo, trans, diag, n, A, ldA, x, incx)	triangular solve	$x = A^{-*} x$	$n^2 : n^2/2$
s, d	ger	(m, n, alpha, x, incx, y, incy, A, ldA)	general rank-1 update	$A = A + \alpha xy^T$	$2mn : mn$
c, z	geru	(m, n, alpha, x, incx, y, incy, A, ldA)	general rank-1 update (complex)	$A = A + \alpha xy^T$	$2mn : mn$
c, z	gerc	(m, n, alpha, x, incx, y, incy, A, ldA)	general rank-1 update (complex conj)	$A = A + \alpha xy^H$	$2mn : mn$
s, d †	syr	(uplo, n, alpha, x, incx, A, ldA)	symmetric rank-1 update	$A = A + \alpha xx^T$	$n^2 : n^2/2$
c, z	her	(uplo, n, alpha, x, incx, A, ldA)	Hermitian rank-1 update	$A = A + \alpha xx^H$	$n^2 : n^2/2$
s, d	syr2	(uplo, n, alpha, x, incx, y, incy, A, ldA)	symmetric rank-2 update	$A = A + \alpha xy^T + \alpha yx^T$	$2n^2 : n^2/2$
c, z	her2	(uplo, n, alpha, x, incx, y, incy, A, ldA)	Hermitian rank-2 update	$A = A + \alpha xy^H + y(\alpha x)^H$	$2n^2 : n^2/2$

Level 2 BLAS, band storage

precisions	name	(options size bandwidth arguments)	description	equation
s, d, c, z	gbmv	(trans, m, n, kl, ku, alpha, A, ldA, x, incx, beta, y, incy)	band general matrix-vector multiply	$y = \alpha A^* x + \beta y$
c, z	hbm	(uplo, n, k, alpha, A, ldA, x, incx, beta, y, incy)	band Hermitian matrix-vector mul.	$y = \alpha Ax + \beta y$
s, d	sbmv	(uplo, n, k, alpha, A, ldA, x, incx, beta, y, incy)	band symmetric matrix-vector mul.	$y = \alpha Ax + \beta y$
s, d, c, z	tbm	(uplo, trans, diag, n, k, A, ldA, x, incx)	band triangular matrix-vector mul.	$x = A^* x$
s, d, c, z	tbsv	(uplo, trans, diag, n, k, A, ldA, x, incx)	band triangular solve	$x = A^{-*} x$

Level 2 BLAS, packed storage

precisions	name (options	size arguments)	description	equation	flops : data	
c, z	hpmv (uplo,	n,	alpha, Ap,	x, incx, beta, y, incy)	packed Hermitian matrix-vector mul.	$y = \alpha Ax + \beta y$ $2n^2 : n^2/2$	
s, d †	spmv (uplo,	n,	alpha, Ap,	x, incx, beta, y, incy)	packed symmetric matrix-vector mul.	$y = \alpha Ax + \beta y$ $2n^2 : n^2/2$	
s, d, c, z	tpmv (uplo, trans, diag, n,	Ap,	x, incx)	packed triangular matrix-vector mul.	$x = A^*x$ $n^2 : n^2/2$	
s, d, c, z	tpsv (uplo, trans, diag, n,	Ap,	x, incx)	packed triangular solve	$x = A^{-*}x$ $n^2 : n^2/2$	
s, d †	spr (uplo,	n,	alpha, x, incx,	Ap)	packed symmetric rank-1 update	$A = A + \alpha xx^T$ $n^2 : n^2/2$
c, z	hpr (uplo,	n,	alpha, x, incx,	Ap)	packed Hermitian rank-1 update	$A = A + \alpha xx^H$ $n^2 : n^2/2$
s, d	spr2 (uplo,	n,	alpha, x, incx, y, incy, Ap)	packed symmetric rank-2 update	$A = A + \alpha xy^T + \alpha yx^T$ $2n^2 : n^2/2$	
c, z	hpr2 (uplo,	n,	alpha, x, incx, y, incy, Ap)	packed Hermitian rank-2 update	$A = A + \alpha xy^H + y(\alpha x)^H$ $2n^2 : n^2/2$	

Level 3 BLAS: matrix-matrix, $O(n^3)$ operations

precisions	name (options	size arguments)	description	equation	flops : data
s, d, c, z	gemm (transa, transb, m, n, k, alpha, A, ldA, B, ldb, beta, C, ldc)	m, n, k,	alpha, A, ldA, B, ldb, beta, C, ldc)	general matrix-matrix multiply	$C = \alpha A^*B^* + \beta C$	$2mnk : mk + nk + mn$
s, d, c, z	symm (side, uplo,	m, n,	alpha, A, ldA, B, ldb, beta, C, ldc)	symmetric matrix-matrix mul.	$C = \alpha AB + \beta C$	$l : 2m^2n : m^2 + mn$
c, z	hemm (side, uplo,	m, n,	alpha, A, ldA, B, ldb, beta, C, ldc)	Hermitian matrix-matrix mul.	$C = \alpha AB + \beta C$	$l : 2m^2n : m^2 + mn$
s, d, c, z	trmm (side, uplo, transa, diag,	m, n,	alpha, A, ldA, B, ldb)	triangular matrix-matrix mul.	$B = \alpha A^*B$ or $B = \alpha BA^*$	$l : m^2n : m^2 + mn$
s, d, c, z	trsm (side, uplo, transa, diag,	m, n,	alpha, A, ldA, B, ldb)	triangular solve matrix	$B = \alpha A^{-*}B$ or $B = \alpha BA^{-*}$	$l : m^2n : m^2 + mn$
s, d, c, z	syrk (uplo, trans,	n, k,	alpha, A, ldA,	beta, C, ldc)	symmetric rank- k update	$C = \alpha AA^T + \beta C$ $kn^2 : n^2/2$
c, z	herk (uplo, trans,	n, k,	alpha, A, ldA,	beta, C, ldc)	Hermitian rank- k update	$C = \alpha AA^H + \beta C$ $kn^2 : n^2/2$
s, d, c, z	syr2k (uplo, trans,	n, k,	alpha, A, ldA, B, ldb, beta, C, ldc)	symmetric rank- $2k$ update	$C = \alpha AB^T + \bar{\alpha} BA^T + \beta C$ $2kn^2 : n^2/2$	
c, z	her2k (uplo, trans,	n, k,	alpha, A, ldA, B, ldb, beta, C, ldc)	Hermitian rank- $2k$ update	$C = \alpha AB^H + \bar{\alpha} BA^H + \beta C$ $2kn^2 : n^2/2$	

A^* denotes A , A^T , or A^H ;

A^{-*} denotes A^{-1} , A^{-T} , or A^{-H} , depending on options and data type.

The destination matrix is $m \times n$ or $n \times n$. For matrix-matrix, the common dimension of A^* and B^* is k .

Flops and data are most significant term only. In complex, each mul becomes 6 flops and each add becomes 2 flops.

Prefixes

s – real (float) d – double
 c – complex z – complex*16
 ge – general gb – general banded
 sy – symmetric sb – symmetric banded sp – symmetric packed
 he – Hermitian hb – Hermitian banded hp – Hermitian packed
 tr – triangular tb – triangular banded tp – triangular packed

† LAPACK adds complex routines [cz]rot,

and complex-symmetric routines for symv, spmv, syr, spr,

but only with Fortran calling conventions, not in CBLAS.

Options

trans = ‘N’o transpose: A , ‘T’ranspose: A^T , ‘C’onjugate transpose: A^H
 uplo = ‘U’pper triangular, ‘L’ower triangular
 diag = ‘N’on-unit triangular, ‘U’nit triangular
 side = ‘L’eft: AB , ‘R’ight: BA
 ldA is major stride—number of rows of parent matrix A . Useful for submatrices.

For real matrices, trans = ‘T’ and ‘C’ are the same.

For Hermitian matrices, trans = ‘T’ is not allowed.

For complex symmetric matrices, trans = ‘C’ is not allowed.



BLAS and LAPACK guides are available from <http://web.eecs.utk.edu/~mgates3/docs/>

Copyright ©2007–2015 by Mark Gates. Updated June 22, 2015.

You may freely copy and modify this document under the [Creative Commons Attribution license](https://creativecommons.org/licenses/by/4.0/).

When distributing, please include the above link and copyright for this original document.

LAPACK summary

Prefixes

Each routine has a prefix, denoted by a hyphen - in this guide, made of 3 letters xyy , where x is the data type, and yy is the matrix type.

Fortran data type	C data type	Abbreviation
real	float	s
double precision	double	d
complex	float complex	c
complex*16	double complex	z

Matrix type	full	packed	RFP	banded	tridiag	generalized problem
general	ge			gb	gt	gg
symmetric	sy	sp	sf	sb	st	
Hermitian	he	hp	hf	hb		
SPD / HPD	po	pp	pf	pb	pt	
triangular	tr	tp	tf	tb		tg
upper Hessenberg	hs					hg
trapezoidal	tz					
orthogonal	or	op				
unitary	un	up				
diagonal					di	
bidagonal					bd	

Storage

Full matrices are stored column-wise (so-called “Fortran” order). For symmetric, Hermitian, and triangular matrices, elements below/above the diagonal (for upper/lower respectively) are not accessed. Similarly for upper Hessenberg, elements below the subdiagonal are not accessed.

Packed storage for triangular or symmetric matrices is by columns. For example, a 3×3 upper triangular matrix is stored as

$$U = \begin{bmatrix} 1 & 2 & 4 \\ \cdot & 3 & 5 \\ \cdot & \cdot & 6 \end{bmatrix} \Rightarrow \left[\underbrace{1}_{\text{column 1}} \quad \underbrace{2 \ 3}_{\text{column 2}} \quad \underbrace{4 \ 5 \ 6}_{\text{column 3}} \right]$$

and a 3×3 lower triangular matrix is stored as

$$L = \begin{bmatrix} 1 & \cdot & \cdot \\ 2 & 4 & \cdot \\ 3 & 5 & 6 \end{bmatrix} \Rightarrow \left[\underbrace{1 \ 2 \ 3}_{\text{column 1}} \quad \underbrace{4 \ 5}_{\text{column 2}} \quad \underbrace{6}_{\text{column 3}} \right]$$

Rectangular full packed (RFP) for triangular or symmetric matrices reuses fast routines for full matrices, but with half the storage and avoiding the inefficient indexing of packed storage. It divides the matrix into 4 quadrants, transposes one of the triangles and packs it next to the other triangle; see [LAPACK Working Note 199](#). For example,

$$L = \begin{bmatrix} L_{11} & \cdot \\ L_{21} & L_{22} \end{bmatrix} \Rightarrow \begin{bmatrix} L_{11} & \text{and } L_{22}^T \\ & L_{21} \end{bmatrix}$$

$$L = \begin{bmatrix} a_{11} & \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{12} & a_{22} & \cdot & \cdot & \cdot & \cdot \\ a_{13} & a_{23} & a_{33} & \cdot & \cdot & \cdot \\ a_{14} & a_{24} & a_{34} & \mathbf{a_{44}} & \cdot & \cdot \\ a_{15} & a_{25} & a_{35} & \mathbf{a_{45}} & \mathbf{a_{55}} & \cdot \\ a_{16} & a_{26} & a_{36} & \mathbf{a_{46}} & \mathbf{a_{56}} & \mathbf{a_{66}} \end{bmatrix} \Rightarrow \begin{bmatrix} \mathbf{a_{44}} & \mathbf{a_{45}} & \mathbf{a_{46}} \\ a_{11} & \mathbf{a_{55}} & \mathbf{a_{56}} \\ a_{12} & a_{22} & \mathbf{a_{66}} \\ a_{13} & a_{23} & a_{33} \\ a_{14} & a_{24} & a_{34} \\ a_{15} & a_{25} & a_{35} \\ a_{16} & a_{26} & a_{36} \end{bmatrix}$$

There are 4 versions depending on whether n is even or odd and whether the upper or lower triangle is stored. Conversion routines are provided.

Banded storage puts columns of the matrix in corresponding columns of the array, and diagonals in rows of the array, for example:

$$\begin{bmatrix} a_{11} & a_{12} & \cdot & \cdot & \cdot \\ a_{21} & a_{22} & a_{23} & \cdot & \cdot \\ a_{31} & a_{32} & a_{33} & a_{34} & \cdot \\ \cdot & a_{42} & a_{43} & a_{44} & a_{45} \\ \cdot & \cdot & a_{53} & a_{54} & a_{55} \end{bmatrix} \Rightarrow \begin{bmatrix} * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{bmatrix} \begin{array}{l} \text{1st diagonal} \\ \text{2nd (main) diagonal} \\ \text{3rd diagonal} \\ \text{4th diagonal} \end{array}$$

Bi- and tri-diagonal matrices are stored as 2 or 3 vectors of length n and $n - 1$.



BLAS and LAPACK guides are available from <http://web.eecs.utk.edu/~mgates3/>

Copyright ©2007–2011 by Mark Gates. Updated September 6, 2011.

You may freely copy and modify this document under the [Creative Commons Attribution license](#). When distributing, please include the above link and copyright for this original document.

Drivers

Drivers are higher level routines that solve an entire problem.

Linear system, solve $Ax = b$.

-sv — solve
 -svx — expert; also $A^T x = b$ or $A^H x = b$, condition number, error bounds, scaling
 Matrix types [General ge, gb, gt; SPD po, pp, pb, pt; Symmetric sy, sp, he, hp]

Linear least squares, minimize $\|b - Ax\|_2$.

-ls — full rank, $\text{rank}(A) = \min(m, n)$, uses QR .
 -lsy — rank deficient, uses complete orthogonal factorization.
 -lsd — rank deficient, uses SVD.
 Matrix types [General ge]

Generalized linear least squares.

Minimize $\|c - Ax\|_2$ subject to $Bx = d$.

-lse — B full row rank, matrix $\begin{bmatrix} A \\ B \end{bmatrix}$ full col rank.

Minimize $\|y\|_2$ subject to $d = Ax + By$.

-glm — A full col rank, matrix $\begin{bmatrix} A & B \end{bmatrix}$ full row rank.
 Matrix types [General gg]

SVD singular value decomposition, $A = U\Sigma V^H$

-svd — singular values, [left, right vectors]
 -sdd — divide-and-conquer; faster but more memory
 Matrix types [General ge]

Generalized SVD, $A = U\Sigma_1 Q^T$ and $B = V\Sigma_2 Q^T$

-svd — singular values, [left, right vectors]
 Matrix types [General gg]

Conversion routines

-tfttp — RFP	to	Packed
-tfttr — RFP	to	Triangular full
-trttp — Triangular full	to	Packed
-trttf — Triangular full	to	RFP
-tpttr — Packed	to	Triangular full
-tpttf — Packed	to	RFP

Eigenvalues, solve $Ax = \lambda x$.

Symmetric

-ev — all eigenvalues, [eigenvectors]
 -evx — expert; also subset
 -evd — divide-and-conquer; faster but more memory
 -evr — relative robust; fastest and least memory
 Matrix types [Symmetric sy, sp, sb, st, he, hp, hb]

Nonsymmetric

-ev — eigenvalues, [left, right eigenvectors]
 -evx — expert; also balance matrix, condition numbers
 -es — Schur factorization
 -esx — expert; also condition numbers
 Matrix types [General ge]

Generalized eigenvalue, solve $Ax = \lambda Bx$

Symmetric, B SPD

-gv — all eigenvalues, [eigenvectors]
 -gvx — expert; also subset
 -gvd — divide-and-conquer, faster but more memory
 Matrix types [Symmetric sy, sp, sb, he, hp, hb]

Nonsymmetric

-ev — eigenvalues, [left, right eigenvectors]
 -evx — expert; also balance matrix, condition numbers
 -es — Schur factorization
 -esx — expert; also condition numbers
 Matrix types [General gg]

Computational routines

Computational routines perform one step of solving the problem. Drivers call a sequence of computational routines.

Triangular factorization

-trf — factorize: General LU , Cholesky LL^T , tridiag LDL^T , sym. indefinite LDL^T
 -trs — solve using factorization
 -con — condition number estimate
 -rfs — error bounds, iterative refinement
 -tri — inverse (not for band)
 -equ — equilibrate A (not for tridiag, symmetric indefinite, triangular)
 Matrix types [General ge, gb, gt; SPD po, pp, pf, pb, pt; Symmetric sy, sp, he, hp,
 Triangular tr, tp, tf, tb]

Orthogonal factorization

-qp3 — QR factorization, with pivoting
 -qrf — QR factorization
 -rqf — RQ factorization
 -qlf — QL factorization
 -lqf — LQ factorization
 -tzrqf — RQ factorization
 -tzrzf — RZ trapezoidal factorization
 Matrix types [General ge, some Trapezoidal tz]

-gqr — generate Q after -qrf
 -grq — generate Q after -rqf
 -gql — generate Q after -qlf
 -glq — generate Q after -lqf
 -mqr — multiply by Q after -qrf
 -mrq — multiply by Q after -rqf
 -mq1 — multiply by Q after -qlf
 -mlq — multiply by Q after -lqf
 -mrz — multiply by Q after -tzrzf
 Matrix types [Orthogonal or, un]

Generalized orthogonal factorization

-qrf — QR of A , then RQ of $Q^T B$
 -rqf — RQ of A , then QR of BQ^T
 Matrix types [General gg]

Eigenvalue

-trd — tridiagonal reduction
 Matrix types [Symmetric sy, he, sp, hp]

-gtr — generate matrix after -trd
 -mtr — multiply matrix after -trd
 Matrix types [Orthogonal or, op, un, up]

Symmetric tridiagonal eigensolvers
 -eqr — using implicitly shifted QR
 -pteqr — using Cholesky and bidiagonal QR
 -erf — using square-root free QR
 -edc — using divide-and-conquer
 -egr — using relatively robust representation
 -ebz — eigenvalues using bisection
 -ein — eigenvectors using inverse iteration
 Matrix types [Symmetric tridiag st, one SPD pt]

Nonsymmetric
 -hrd — Hessenberg reduction
 -bal — balance
 -bak — back transforming
 Matrix types [General ge]

-ghr — generate matrix after -hrd
 -mhr — multiply matrix after -hrd
 Matrix types [Orthogonal or, un]

-eqr — Schur factorization
 -ein — eigenvectors using inverse iteration
 Matrix types [upper Hessenberg hs]

-evc — eigenvectors
 -exc — reorder Schur factorization
 -syl — Sylvester equation
 -sna — condition numbers
 -sen — condition numbers of eigenvalue cluster/subspace
 Matrix types [Triangular tr]