Utilisation des bibliothèques d'algèbre linéaire Blas et Lapack

Jean Hare

9 mars 2012

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 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://xianyi.github.com/OpenBLAS/`), 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 OpenBlas [1], 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 problème 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. 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 double et simple 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 («computationelles») 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 plus de 950 !).

Le nom de chaque procédure de LAPACK ou de BLAS est construit sur la base de 6 caractères comme `XYYZZZ` [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 suit [3] :

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'est 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 pour la définition de ce format).
NB2 : Les problèmes «généralisés» sont des problèmes à deux matrices, où la seconde matrice soit ajoute 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 indispensable 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.

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.
URL `http://www.netlib.org/lapack/explore-html/index.html`.
– Sur le site de E.R. Jessup, un assistant permet de sélectionner aisément la bonne fonction
URL `http://www.cs.colorado.edu/~jessup/lapack/drivers.html`.
– Sur le site d'Intel, la documentation de la MKL d'Intel, comporte une aide sur Blas et Lapack
URL `http://tinyurl.com/intel-mkl-blas95-level3`
URL `http://tinyurl.com/intel-mkl-lapack-lin`.
– 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`, et `Fortan95` ;
URL `http://docs.oracle.com/cd/E19059-01/stud.10/819-0497/`.
– Enfin, on trouve en annexe de ce document les fiches "*Quick reference*" de Blas et de Lapack.

3 Utilisation

3.1 En Fortran 95

L'utilisation de Blas et de Lapack en Fortran 95 ne pose pas de problèmes particuliers. 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 Lapack95 :
– 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 généricité 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 est alors précisé lors de la déclaration du tableau via le module `la_precision`, comme il est illustré ci-dessous.
– le paramètre de `status` est géré via une variable entière optionnelle `info`.

La documentation spécifique de Lapack 95 se trouve sur le site de `netlib.org`, avec une liste des procédures pilotes à l'URL `http://www.netlib.org/lapack95/L90index`.

3.1.1 Compilation

La seule subtilité vient de la compilation des bibliothèques, qui crée des bibliothèques `.a` et des interfaces `.mod` qui sont dépendantes à la fois de la machine et du compilateur utilisé. C'est pourquoi, dans le répertoire d'installation `/home/lefrere/lapack/`, sur `sappli1.ccre.upmc.fr`, on trouvera les répertoires `include/sappli/` et `lib/sappli/` contenant les fichiers adaptés au travail en `ssh` sur `sappli`, et `include/mdv2010/` et `lib/mdv2010/` pour travailler à l'UTÈS sur une machine virtuelle `mandriva 2010`.
Le `makefile` doit donc contenir les macros suivantes, `LA_SYSTEM` étant choisie selon le contexte :

```
LA_INCPATH=/home/lefrere/lapack/include/$(LA_SYSTEM)
LA_LIBPATH=/home/lefrere/lapack/lib/$(LA_SYSTEM)
```

Les règles de compilation et d'édition de liens utilisent les options :

`-I$(LA_INCPATH)` et `-L$(LA_LIBPATH)`.

La compilation se fera avec `gfortran`, de préférence en utilisant l'option `-std=f2003`.
L'édition de liens appellera *dans cet ordre* les bibliothèques suivantes :

`-llapack95 -lblas95 -lopenblas -lpthread`.

3.1.2 Exemple

Le module ci-dessous définit deux opérations matricielles, la multiplication avec Blas et l'inversion avec Lapack.

src/matop.f90

```
! exemples d'appel a Blas et Lapack via les interfaces f95
module matop
   use la_precision, only:wp=>sp
   ! definit  wp (working precision )  a sp (simple precision)
   use blas95, only: gemm
   use f95_lapack, only: la_gesv
   implicit none
   contains
   ! Multiplication  de deux matrices   A(M,K) . B(K,N) = C(M,N)        !
   ! A, B, C sont des tableaux  2-D   usuels du Fortran                 !
   ! multiplication par la subroutine generique GEMM de BLAS95         !
   ! voir http://docs.sun.com/source/817-6700/dgemm.html               !
   ! ou  http://tinyurl.com/intel-mkl-blas95-level3                    !
     subroutine matmulblas(A,B,C)
        real, dimension(:,:), intent(in)  :: A,B
        real, dimension(:,:), intent(out) :: C
        integer:: m,k,n, kp
        m=size(A,1)
        k=size(A,2)
        kp=size(B,1)
        n=size(B,2)
        if ((k/=kp) .or. (m/=size(C,1)) .or. (n/=size(C,2))) then
           write(*,*) "Erreur dans matmulblas, tailles incompatibles"
           stop
        end if
        call GEMM(A,B,C)
     end subroutine matmulblas
```

```
   ! Inversion de la matrice carree A(M,M)                              !
   ! A est un tableau 2-D  usuel du Fortran                            !
   ! inversion par la subroutine generique LA_GESV de LAPACK95         !
   ! voir http://www.netlib.org/lapack95/lug95/node72.html             !
   ! ou  http://tinyurl.com/intel-mkl-lapack-lin                       !
      subroutine invmatlapack(A,invA)
         ! A est utilise comme esoace de travail
         ! et contient la décomposition LU en sortie
         real, dimension(:,:), intent(inout) :: A
         real, dimension(:,:), intent(out) :: invA
         integer:: m, i, info
         m=size(A,1)
         if ( (m/=size(A,2)) .or. (m/=size(invA,1)) .or. (m/=size(invA,2))) then
            write(*,*) "Erreur dans invmatlapack, tailles incompatibles"
            stop
         endif
         invA(:,:)=0.0
         do i=1, m
            invA(i,i)=1.0
         end do
         call LA_GESV(A,invA,info=info)
         if (info/=0) then
            write(*,*) "Erreur dans invmatlapack, code", info
            stop
         end if
      end subroutine invmatlapack
end module matop
```

3.2 En C

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 `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 `lapacke_` (en minuscules 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 utilisé [4].

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* [5]. La solution de ce problème fait l'objet des deux paragraphes qui suivent.

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

5. 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.

3.2.1 Stockage des matrices en tableau 1D

La solution **recommandée** consiste à coder systématiquement les matrices sous la forme linéaire, ou «aplatie», 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)+nlig*(j-1)`, variant entre `0` et `nlig*ncol-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[icm(nlig,ncol,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.

3.2.2 Stockage des matrices en tableaux 2D

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`.
- 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.

3.2.3 Compilation

Comme en Fortran, les bibliothèques et les fichiers d'entête dépendent à la fois de la machine et de la version du compilateur utilisé. C'est pourquoi, dans le répertoire d'installation `/home/lefrere/lapack/`, sur `sappli1.ccre.upmc.fr`, on trouvera les répertoires `include/sappli/` et `lib/sappli/` contenant les fichiers adaptés au travail en `ssh` sur `sappli`, et `include/mdv2010/` et `lib/mdv2010/` pour travailler à l'UTÈS sur une machine virtuelle `mandriva 2010`. Le `makefile` doit contenir les macros suivantes, `LA_SYSTEM` étant défini selon le contexte :

```
LA_INCPATH=/home/lefrere/lapack/include/$(LA_SYSTEM)
LA_LIBPATH=/home/lefrere/lapack/lib/$(LA_SYSTEM)
```

et les règles de compilation et d'édition de liens doivent utiliser les options :

`-I$(LA_INCPATH)` et `-L$(LA_LIBPATH)`.

La compilation avec `gcc` se fera en utilisant l'option `-std=c99`.

L'édition de liens appelera *dans cet ordre*[6] les bibliothèques suivantes :

```
-llapacke  -lopenblas -lpthread  -lm
```

NB : Sous Windows/MinGW, on n'appellera pas la bibliothèque `pthread`.

6. Lors de l'édition de liens, 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/`.

3.2.4 Exemple

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

src/matop.h

```
void matmulblas(const int m,const int k, const int n, real *A, real *B, real *C) ;
void invmatlapack ( const int n , real *A , real * invA ) ;
```

src/matop.c

```
/*===================  Module matop.c ========================*/
/* exemple d'operations matricielles avec Blas/Lapack         */
/* version recommandee en ColMajor avec matrices aplaties     */
#include <stdio.h>
#include <stdlib.h>
#include "cblas.h"
#include "lapacke.h"
#include "precision.h" // precision.h contient typedef float real; ou typedef double
    real;
#include "matop.h"

/* Multiplication  de deux matrices   A[m*k] . B[k*n] = C[m*n]        */
/* multiplication par BLAS xGEMM,                                    */
/* cf http://docs.oracle.com/cd/E19059-01/stud.10/819-0497/sgemm.html */

void matmulblas(const int m,const int k, const int n, real *A, real *B, real *C) {
   if (sizeof(real)==sizeof(float)) { //selection du type utilisé : float
      cblas_sgemm(CblasColMajor, CblasNoTrans,CblasNoTrans,
                  m, n,k, 1.0,(float*) A, m,(float*)B, k, 0.0, (float*)C,m) ;
   }
   else {   //selection du type utilisé : double
      cblas_dgemm(CblasColMajor, CblasNoTrans,CblasNoTrans,
                  m, n,k, 1.0, (double*)A, m,(double*)B, k, 0.0, (double*)C,m) ;
   }
}

/* Inversion d'une matrice carrée  A[n*n]    par lapack xGESV         */
/* cf http://docs.oracle.com/cd/E19059-01/stud.10/819-0497/sgesv.html */
void invmatlapack ( const int n , real *A , real * invA ) {
   int i , k , info ;

   int * ipiv ;  // en retour,  indique des lignes qui ont occasionné un pivotement
   ipiv = calloc (n , sizeof ( int ) ) ;
   for ( k =0; k < n * n ; k ++) { //construction de l'identité
      invA [ k ] = (real)(0.0) ;
   }
   for ( i =0; i < n ; i ++) { // diagonale de l'identité
      invA [ i*(n+1) ]= (real)(1.0);
   }
   // en retour invA contient  l'inverse, et A est remplacé par sa factorisation LU
   if (sizeof(real)==sizeof(float)) { //selection du type utilisé : float
      info=lapacke_sgesv (CblasColMajor,n , n , (float*) A , n ,
                          ipiv , (float*) invA , n);
   }
   else { //selection du type utilisé : double
      info=lapacke_dgesv (CblasColMajor, n , n , (double*) A , n ,
                          ipiv , (double*) invA , n);
   }
   if ( info !=0) {
      fprintf ( stderr , " Erreur dans invmatlapack code =% d \n " , info ) ;
      exit ( EXIT_FAILURE ) ;
   }
}
```

4 Installation

L'installation est relativement aisée et pourra être réalisée sous Linux32/64, ou sous Windows32/64 avec MinGW/Msys, en se conformant aux instructions suivantes :

1. **Téléchargement**
   Télécharger OpenBlas depuis http ://xianyi.github.com/OpenBLAS/
   `>> wget http://github.com/xianyi/OpenBLAS/tarball/v0.1alpha2.5/index.html -O openblas.tgz`
   ou, alternativement depuis http ://www.phys.ens.fr/∼hare
   `>> wget http://www.phys.ens.fr/~hare/MP025/openblas.tgz`
   Dézipper l'archive
   `>> tar -xvzf openblas.tgz`
   Renommer le répertoire obtenu(par défaut `xianyi-OpenBLAS-9dcea60`) sous le nom `openblas`
   `>> mv xianyi-OpenBLAS-* openblas`
   Télécharger l'archive OpenBlasInstall depuis http ://www.phys.ens.fr/∼hare
   `>> wget http://www.phys.ens.fr/~hare/MP025/OpenBlasInstall.tgz`
   La dézipper où vous voulez (par exemple à côté de `openblas`)
   `>> tar -xvzf OpenBlasInstall.tgz`
   Copier le contenu de `OpenBlasInstall` dans `openblas`
   `>>  /bin/cp -f ./OpenBlasInstall/* ./openblas/`
2. **Préparation**
   Si nécessaire, créer le répertoire `lapack` sur votre compte ou autre répertoire où vous avez des droits en écriture, et modifier le fichier `openblas/installpath.inc` en conséquence.
3. **Compilation**
   Entrer dans le répertoire
   `>> cd openblas`
   et exécuter successivemnt les cibles du fichier `mymakefile` :
   (la cible `build` comprend le téléchargement, la compilation et le test de Lapack 3.4.0)

```
>> make -f mymakefile build
>> make -f mymakefile install
>> make -f mymakefile lapacke
>> make -f mymakefile lapackeinstall
>> make -f mymakefile lapack95
>> make -f mymakefile lapack95install
>> make -f mymakefile blas95
>> make -f mymakefile blas95install
```

Notes :
Le premier `wget` ne marche généralement pas : utiliser votre navigateur
Linux
- La première cible `build` déclenche normalement les tests automatiques ; sinon, il faudra l'appeler une seconde fois (ces tests peuvent durer très longtemps, entre 1 et 20 min).
- Utiliser `-lpthread` parmi les bibliothèques nécessaires.
- La présence de `libopenblas.so` dans `lib` peut perturber la compilation, auquel cas le renommer provisoirement.

Windows
- Ne pas mettre `-lpthread` parmi les bibliothèques nécessaires.
- La bibliothèque `openblas` obtenue est un `.lib` à renommer `.a` à l'étape d'installation.

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 librairies 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 deBLAS 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 ici[7], 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 utilisés 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 LAPACK (et/ou de ses extensions), *l'efficacité* de l'ensemble repose alors presque exclusivement sur l'optimisation deBLAS. 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 lorqu'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 niveaux 1 (L1). Cette approche est celle suivie par le projet ATLAS (pour «Automatically Tuned Linear Algebra Software» `http://www.netlib.org/atlas` et `http://sourceforge.net/projects/math-atlas/`), qui repose sur une version des `cblas` (c'est à dire une ré-impléméntation 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 éloignée 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 a 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 (`http://www.tacc.utexas.edu/tacc-projects/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://xianyi.github.com/OpenBLAS/`).

C'est cette dernière version que nous utiliserons en 2011-2012, contrairement aux années précédentes.

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.

Références rapides

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

A 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).

A.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 $i \leqslant j$ sont stockés dans la matrice `M(i,j)` et les autres éléments sont ignorés ;

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

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

A.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 $i \leqslant j$ avec $k = i + j(j-1)/2$ pour $0 < i \leqslant j$

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

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`.

A.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 $kl = 2$:

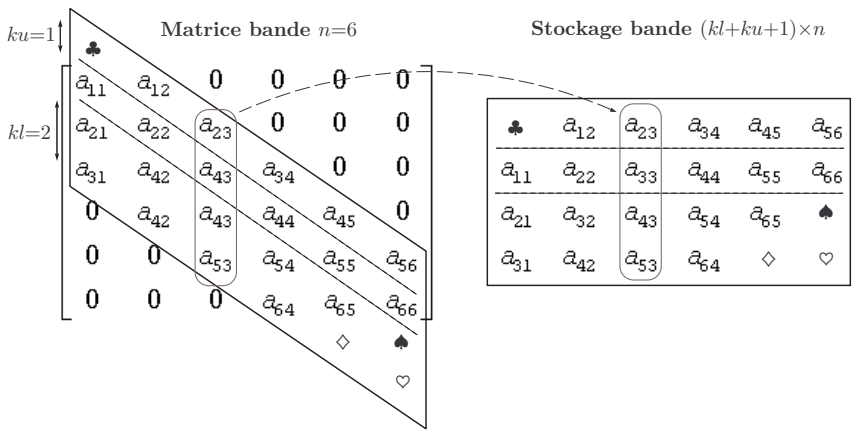

FIGURE 1 – *Stockage bande*

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

BLAS Fortran 77 prototypes

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

name	(**dim**, **scalar**, **vector**, **vector**, **scalars**, **param**)	**description**	**equation**	**prefixes**
_rotg	(a, b, c, s)	generate plane rotation		s, d
_rotmg	(d1, d2, a, b, param)	generate modified plane rotation		s, d
_rot	(n, x, incx, y, incy, c, s)	apply plane rotation		s, d
_rotm	(n, x, incx, y, incy, param)	apply modified plane rotation		s, d
_swap	(n, x, incx, y, incy)	swap vectors	$x \leftrightarrow y$	s, d, c, z
_scal	(n, alpha, x, incx)	scale vector	$y = \alpha y$	s, d, c, z, cs, zd
_copy	(n, x, incx, y, incy)	copy vector	$y = x$	s, d, c, z
_axpy	(n, alpha, x, incx, y, incy)	update vector	$y = y + \alpha x$	s, d, c, z
_dot	(n, x, incx, y, incy)	dot product	$= x^t y$	s, d, ds
_dotu	(n, x, incx, y, incy)	(complex)	$= x^t y$	c, z
_dotc	(n, x, incx, y, incy)	(complex conj)	$= x^h y$	c, z
__dot	(n, x, incx, y, incy)	(?)	$= \alpha + x^t y$	sds
_nrm2	(n, x, incx)	2-norm	$= \|x\|_2$	s, d, sc, dz
_asum	(n, x, incx)	1-norm	$= \|\mathrm{Re}(x)\|_1 + \|\mathrm{Im}(x)\|_1$	s, d, sc, dz
i_amax	(n, x, incx)	∞-norm	$= i$ such that $\|\mathrm{Re}(x_i)\| + \|\mathrm{Im}(x_i)\|$ is max	s, d, c, z

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

name	(**options**, **dim**, **bandwidth**, **scalar**, **matrix**, **vector**, **scalar**, **vector**)	**description**	**equation**	**prefixes**
_gemv	(trans, m, n, alpha, A, ldA, x, incx, beta, y, incy)	general matrix-vector multiply	$y = \alpha A^* x + \beta y$	s, d, c, z
_gbmv	(trans, m, n, kl, ku, alpha, A, ldA, x, incx, beta, y, incy)	(banded)	$y = \alpha A^* x + \beta y$	s, d, c, z
_hemv	(uplo, n, alpha, A, ldA, x, incx, beta, y, incy)	hermetian mat-vec	$y = \alpha A x + \beta y$	c, z
_hbmv	(uplo, n, k, alpha, A, ldA, x, incx, beta, y, incy)	(banded)	$y = \alpha A x + \beta y$	c, z
_hpmv	(uplo, n, alpha, Ap, x, incx, beta, y, incy)	(packed)	$y = \alpha A x + \beta y$	c, z
_symv	(uplo, n, alpha, A, ldA, x, incx, beta, y, incy)	symmetric mat-vec	$y = \alpha A x + \beta y$	s, d, (c, z)†
_sbmv	(uplo, n, k, alpha, A, ldA, x, incx, beta, y, incy)	(banded)	$y = \alpha A x + \beta y$	s, d
_spmv	(uplo, n, alpha, Ap, x, incx, beta, y, incy)	(packed)	$y = \alpha A x + \beta y$	s, d, (c, z)†
_trmv	(uplo, trans, diag, n, A, ldA, x, incx)	triangular mat-vec	$x = A^* x$	s, d, c, z
_tbmv	(uplo, trans, diag, n, k, A, ldA, x, incx)	(banded)	$x = A^* x$	s, d, c, z
_tpmv	(uplo, trans, diag, n, Ap, x, incx)	(packed)	$x = A^* x$	s, d, c, z
_trsv	(uplo, trans, diag, n, A, ldA, x, incx)	triangular solve	$x = A^{-*} x$	s, d, c, z
_tbsv	(uplo, trans, diag, n, k, A, ldA, x, incx)	(banded)	$x = A^{-*} x$	s, d, c, z
_tpsv	(uplo, trans, diag, n, Ap, x, incx)	(packed)	$x = A^{-*} x$	s, d, c, z

A^* denotes A, A^T, or A^H;
A^{-*} denotes A^{-1}, A^{-T}, or A^{-H}, depending on options and data type.
A is $m \times n$ or $n \times n$.

Level 2, continued

name (options, dim, scalar, vector, vector, matrix)	description	equation	prefixes
_ger (m, n, alpha, x, incx, y, incy, A, ldA)	general rank-1 update	$A = A + \alpha xy^T$	s, d
_geru (m, n, alpha, x, incx, y, incy, A, ldA)	(complex)	$A = A + \alpha xy^T$	c, z
_gerc (m, n, alpha, x, incx, y, incy, A, ldA)	(complex conj)	$A = A + \alpha xy^H$	c, z
_her (uplo, n, alpha, x, incx, A, ldA)	hermetian rank-1 update	$A = A + \alpha xx^H$	c, z
_hpr (uplo, n, alpha, x, incx, Ap)	(packed)	$A = A + \alpha xx^H$	c, z
_her2 (uplo, n, alpha, x, incx, y, incy, A, ldA)	hermetian rank-2 update	$A = A + \alpha xy^H + y(\alpha x)^H$	c, z
_hpr2 (uplo, n, alpha, x, incx, y, incy, Ap)	(packed)	$A = A + \alpha xy^H + y(\alpha x)^H$	c, z
_syr (uplo, n, alpha, x, incx, A, ldA)	symmetric rank-1 update	$A = A + \alpha xx^T$	s, d, (c, z)†
_spr (uplo, n, alpha, x, incx, Ap)	(packed)	$A = A + \alpha xx^T$	s, d, (c, z)†
_syr2 (uplo, n, alpha, x, incx, y, incy, A, ldA)	symmetric rank-2 update	$A = A + \alpha xy^T + \alpha yx^T$	s, d
_spr2 (uplo, n, alpha, x, incx, y, incy, Ap)	(packed)	$A = A + \alpha xy^T + \alpha yx^T$	s, d

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

name (options, dim, scalar, matrix, matrix, scalar, matrix)	description	equation	prefixes
_gemm (transa, transb, m, n, k, alpha, A, ldA, B, ldB, beta, C, ldC)	general matrix-matrix multiply	$C = \alpha A^* B^* + \beta C$	s, d, c, z
_symm (side, uplo, m, n, alpha, A, ldA, B, ldB, beta, C, ldC)	symmetric mat-mat	$C = \alpha AB + \beta C$	s, d, c, z
_hemm (side, uplo, m, n, alpha, A, ldA, B, ldB, beta, C, ldC)	hermetian mat-mat	$C = \alpha AB + \beta C$	c, z
_syrk (uplo, trans, n, k, alpha, A, ldA, beta, C, ldC)	symmetric rank-k update	$C = \alpha AA^T + \beta C$	s, d, c, z
_herk (uplo, trans, n, k, alpha, A, ldA, beta, C, ldC)	hermetian rank-k update	$C = \alpha AA^H + \beta C$	c, z
_syr2k (uplo, trans, n, k, alpha, A, ldA, B, ldB, beta, C, ldC)	symmetric rank-2k update	$C = \alpha AB^T + \bar{\alpha} BA^T + \beta C$	s, d, c, z
_her2k (uplo, trans, n, k, alpha, A, ldA, B, ldB, beta, C, ldC)	hermetian rank-2k update	$C = \alpha AB^H + \bar{\alpha} BA^H + \beta C$	c, z
_trmm (side, uplo, transa, diag, m, n, alpha, A, ldA, B, ldB)	triangular mat-mat	$B = \alpha A^* B$ or $B = \alpha BA^*$	s, d, c, z
_trsm (side, uplo, transa, diag, m, n, alpha, A, ldA, B, ldB)	triangular solve mat	$B = \alpha A^{-*} B$ or $B = \alpha BA^{-*}$	s, d, c, z

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 mat-mat, the common dimension of A and B is k.

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 – hermetian	hb – hermetian banded	hp – hermetian packed
tr – triangular	tb – triangular banded	tp – triangular packed

† LAPACK adds these complex symmetric routines.

Options

tranx = '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, transx = 'T' and 'C' are the same.
For Hermitian matrices, transx = 'T' is not allowed.
For complex symmetric matrices, transx = 'C' is not allowed.

Updated June 13, 2008. BLAS and LAPACK guides available from http://www.ews.uiuc.edu/~mrgates2/.
Reference: *BLAS Quick Reference Guide* from http://www.netlib.org/blas/faq.html

LAPACK quick reference guide

Prefixes

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

Data type			
s	single	d	double
c	complex single	z	complex double

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

For symmetric, Hermition, 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 is by columns. For example, a 3×3 upper triangular is stored as

$$[\underbrace{a_{11}}\ \underbrace{a_{12}\ a_{22}}\ \underbrace{a_{13}\ a_{23}\ a_{33}}]$$

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

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

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

Updated March 19, 2008.
BLAS and LAPACK guides available from http://www.ews.uiuc.edu/~mrgates2/.
Reference: *LAPACK Users Guide* from http://www.netlib.org/lapack/faq.html

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]

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]

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]

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, pb, pt; Symmetric sy, sp, he, hp, Triangular tr, 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`
`-mql` — 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]