THE LAPACKE C INTERFACE TO LAPACK

Contents

1. Introduction

This document describes a two-level C interface to LAPACK, consisting of a high-level interface and a middle-level interface. The high-level interface handles all workspace memory allocation internally, while the middle-level interface requires the user to provide workspace arrays as in the original FORTRAN interface. Both interfaces provide support for both column-major and row-major matrices. The prototypes for both interfaces, associated macros and type definitions are contained in the header file lapacke.h.

1.1. **Naming Schemes.** The naming scheme for the high-level interface is to take the FORTRAN LAPACK routine name, make it lower case, and add the prefix LAPACKE_. For example, the LAPACK subroutine DGETRF becomes LAPACKE_dgetrf.

Date: September 30, 2011.

The naming scheme for the middle-level interface is to take the FORTRAN LAPACK routine name, make it lower case, then add the prefix LAPACKE_ and the suffix _work. For example, the LAPACK subroutine DGETRF becomes LAPACKE_dgetrf_work.

1.2. **Complex Types.** Complex data types are defined by the macros lapack_complex_float and lapack_complex_double, which represent single precision and double precision complex data types respectively. It is assumed throughout that the real and imaginary components are stored contiguously in memory, with the real component first. The lapack_complex_float and lapack_complex_double macros can be either C99 _Complex types, a C struct defined type, C++ STL complex types, or a custom complex type. See lapacke.h for more details.

1.3. **Array Arguments.** Arrays are passed as pointers, not as a pointer to pointers. All the LAPACKE routines that take one or more 2D arrays as a pointer receive a single extra parameter of type int. This argument must be equal to either LAPACK_ROW_MAJOR or LAPACK_COL_MAJOR which are defined in lapacke.h, specifying whether the arrays are stored in row-major or column-major order. If a routine has multiple array inputs, they must all use the same ordering.

Note that using row-major ordering may require more memory and time than column-major ordering, because the routine must transpose the row-major order to the column-major order required by the underlying LAPACK routine.

Each 2D array argument in a FORTRAN LAPACK routine has an additional argument that specifies its leading dimension. For row-major 2D arrays, elements within a row are assumed to be contiguous and elements from one row to the next are assumed to be a leading dimension apart. For column-major 2D arrays, elements within a column are assumed to be contiguous and elements from one column to the next are assumed to be a leading dimension apart.

1.4. **Aliasing of Arguments.** Unless specified otherwise, only input arguments (that is, scalars passed by values and arrays specified with the const qualifier) may be legally aliased on a call to C interface to LAPACK.

1.5. **INFO Parameters.** The LAPACKE interface functions set their lapack_int return value to the value of the INFO parameter, which contains information such as error and exit conditions. This differs from LAPACK routines, which return this information as a FORTRAN integer parameter.

In LAPACKE, INFO is used exactly as it is in LAPACK. If INFO returns the row or column number of a matrix using 1-based indexing in FORTRAN, the value is not adjusted for zero-based indexing.

1.6. **NaN Checking.** The high-level interface includes an optional, on by default, NaN check on all matrix inputs before calling any LAPACK routine. This option affects all routines. If the inputs contain any NaNs, the input parameter corresponding matrix will be flagged with an INFO parameter error. For example, if the fifth parameter is found to contain a NaN, the function will return with the value -5.

The NaN check, as well as other parameters, can be disabled by defining LAPACK_DISABLE_NAN_CHECK macro in lapacke.h. The middle-level interface does not contain the NaN check.

1.7. **Integers.** Variables with the FORTRAN type integer are converted to lapack_int in LAPACKE. This conforms with modifiable integer type size, especially given ILP64 programming model: re-defining lapack_int as long int (8 bytes) will be enough to support this model, as lapack_int is defined as int (4 bytes) by default, supporting LP64 programming model.

1.8. **Logicals.** FORTRAN logicals are converted to lapack_logical, which is defined as lapack_int.

1.9. **Memory Management.** All memory management is handled by the functions LAPACKE_malloc and LAPACKE_free. This allows users to easily use their own memory manager instead of the default by modifying their definitions in lapacke.h.

This interface should be thread-safe to the extent that these memory management routines and the underlying LAPACK routines are thread-safe.

1.10. **New Error Codes.** Since the high level interface does not use work arrays, error notification is needed in the event of a user running out of memory.

If a work array cannot be allocated, LAPACK_WORK_MEMORY_ERROR is returned by the function; if there was insufficient memory to complete a transposition, LAPACK_TRANSPOSE_MEMORY_ERROR is returned.

2. Function List

This section list the currently available LAPACK subroutines that are available in the LAPACKE C interface. The LAPACK base names are given below; the corresponding LAPACKE function name is LAPACKE_x*base* or LAPACKE_x*base*_work where x is the type: s or d for single or double precision real, c or z for single or double precision complex, with *base* representing the base name. Function prototypes are given in the file lapacke.h. See the LAPACK documentation for detailed information about the routines and their parameters.

2.1. **Real Functions.** The following LAPACK subroutine base names are supported for single precision (s) and double precision (d), in both the high-level and middle-level interfaces:

bdsdc bdsqr disna gbbrd gbcon gbequ gbequb gbrfs gbrfsx gbsv gbsvx gbsvxx gbtrf gbtrs gebak gebal gebrd gecon geequ geequb gees geesx geev geevx gehrd gejsv gelqf gels gelsd gelss gelsy geqlf geqp3 geqpf geqrf geqrfp gerfs gerfsx gerqf gesdd gesv gesvd gesvj gesvx gesvxx getrf getri getrs ggbak ggbal gges ggesx ggev ggevx ggglm gghrd gglse ggqrf ggrqf ggsvd ggsvp gtcon gtrfs gtsv gtsvx gttrf gttrs hgeqz hsein hseqr opgtr opmtr orgbr orghr orglq orgql orgqr orgrq orgtr ormbr ormhr ormlq ormql ormqr ormrq ormrz ormtr pbcon pbequ pbrfs pbstf pbsv pbsvx pbtrf pbtrs pftrf pftri pftrs pocon poequ poequb porfs porfsx posv posvx posvxx potrf potri potrs ppcon ppequ pprfs ppsv ppsvx pptrf pptri pptrs pstrf ptcon pteqr ptrfs ptsv ptsvx pttrf pttrs sbev sbevd sbevx sbgst sbgv sbgvd sbgvx sbtrd sfrk spcon spev spevd spevx spgst spgv spgvd spgvx sprfs spsv spsvx sptrd sptrf sptri sptrs stebz stedc stegr stein stemr steqr sterf stev stevd stevr stevx sycon syequb syev syevd syevr syevx sygst sygv sygvd sygvx syrfs syrfsx sysv sysvx sysvxx sytrd sytrf sytri sytrs tbcon tbrfs tbtrs tfsm tftri tfttp tfttr tgevc tgexc tgsen tgsja tgsna tgsyl tpcon tprfs tptri tptrs tpttf tpttr trcon trevc trexc trrfs trsen trsna trsyl trtri trtrs trttf trttp tzrzf

2.2. **Complex Functions.** The following LAPACK subroutine base names are supported for complex single precision (c) and complex double precision (z), in both the high-level and middle-level interfaces:

bdsqr gbbrd gbcon gbequ gbequb gbrfs gbrfsx gbsv gbsvx gbsvxx gbtrf gbtrs gebak gebal gebrd gecon geequ geequb gees geesx geev geevx gehrd gelqf gels gelsd gelss gelsy geqlf geqp3 geqpf geqrf geqrfp gerfs gerfsx gerqf gesdd gesv gesvd gesvx gesvxx getrf getri getrs ggbak ggbal gges ggesx ggev ggevx ggglm gghrd gglse ggqrf ggrqf ggsvd ggsvp gtcon gtrfs gtsv gtsvx gttrf gttrs hbev hbevd hbevx hbgst hbgv hbgvd hbgvx hbtrd hecon heequb heev heevd heevr heevx hegst hegv hegvd hegvx herfs herfsx hesv hesvx hesvxx hetrd hetrf hetri hetrs hfrk hgeqz hpcon hpev hpevd hpevx hpgst hpgv hpgvd hpgvx hprfs hpsv hpsvx hptrd hptrf hptri hptrs hsein hseqr pbcon pbequ pbrfs pbstf pbsv pbsvx pbtrf pbtrs pftrf pftri pftrs pocon poequ poequb porfs porfsx posv posvx posvxx potrf potri potrs ppcon ppequ pprfs ppsv ppsvx pptrf pptri pptrs pstrf ptcon pteqr ptrfs ptsv ptsvx pttrf pttrs spcon sprfs spsv spsvx sptrf sptri sptrs stedc stegr stein stemr steqr sycon syequb syrfs syrfsx sysv sysvx sysvxx sytrf sytri sytrs tbcon tbrfs tbtrs tfsm tftri tfttp tfttr tgevc tgexc tgsen tgsja tgsna tgsyl tpcon tprfs tptri tptrs tpttf tpttr trcon trevc trexc trrfs trsen trsna trsyl trtri trtrs trttf trttp tzrzf ungbr unghr unglq ungql ungqr ungrq ungtr unmbr unmhr unmlq unmql unmqr unmrq unmrz unmtr upgtr upmtr

2.3. **Mixed Precision Functions.** The following LAPACK subroutine base names are supported only for double precision (d) and complex double precision (z):

sgesv sposv

3. Examples

This section contains examples of calling LAPACKE functions from a C program.

3.1. **Calling DGEQRF.** Suppose you wish to call the function DGEQRF, which computes the QR factorization of a double precision real rectangular matrix in LAPACK, and you wish to have the LAPACKE interface handle the necessary work space memory allocation for you.

The base name for this function is geqrf, which is included in the list of real functions above. The LAPACKE function name is then constructed by prepending LAPACK_ followed by d to the base name: LAPACKE_dgeqrf.

We will assume that our matrix is stored in column-major order in the $m \times n$ array a, which has a leading dimension of lda. The variable declarations should be as follows:

```
lapack_int m, n, lda, info;
double *a, *tau;
```

The LAPACKE function call is then:

```
info = LAPACKE_dgeqrf( LAPACK_COL_MAJOR, m, n, a, lda, tau );
```

3.2. **Calling CUNGQR.** Suppose you wish to call the function CUNGQR, which generates Q from the results of a QR factorization of a single precision complex rectangular matrix, and you wish to provide the required workspace.

The base name for this function is ungqr, which is included in the list of complex functions above. The LAPACKE function name is then constructed by prepending LAPACK_ followed by c to the base name, with the suffix _work to indicate that the user will supply the work space: LAPACKE_cungqr_work.

We will assume again that our matrix is stored in column-major order in the $m \times n$ array a, which has a leading dimension of lda. From the LAPACK documentation, the work space array work must have a length of at least n; the length of work is given in lwork. The variable declarations should be as follows:

```
lapack_int m, n, k, lda, lwork, info;
lapack_complex_float *a, *tau, *work;
```

The LAPACKE function call is then:

info = LAPACKE_cungqr_work(LAPACK_COL_MAJOR, m, n, k, a, lda, tau, work, lwork);

3.3. **Calling DGELS.** In this example, we wish solve the least squares problem

$$\min_x ||B - Ax||$$

for two right-hand sides using the LAPACK routine DGELS. For input we will use the 5×3 matrix

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 3 & 4 \\ 3 & 5 & 2 \\ 4 & 2 & 5 \\ 5 & 4 & 3 \end{pmatrix}$$

and the 5×2 matrix

$$B = \begin{pmatrix} -10 & -3 \\ 12 & 14 \\ 14 & 12 \\ 16 & 16 \\ 18 & 16 \end{pmatrix}.$$

We will first store the input matrix as a static C two-dimensional array, which is stored in row-major order, and let LAPACKE handle the work space array allocation. The LAPACK base name for this function is gels, and we will use double precision (d), so the LAPACKE function name is LAPACKE_dgels.

Note that the leading dimensions of the arrays are in this case the number of *columns*, thus lda= 3 and ldb= 2. The output for each right hand side is stored in b as consecutive vectors of length 3. The correct answer for this problem is the 3×2 matrix

$$\begin{pmatrix} 2 & 1 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}.$$

A complete C program for this example is given in Figure 1. Note that when the arrays are passed to the LAPACK routine, they must be dereferenced, since LAPACK is expecting arrays of type double *, *not* double **.

Alternatively, we can use column-major ordering for the matrices in this example, as shown in Figure 2. Here, the matrices are stored as static one-dimensional C arrays. These arrays have a leading dimension that is equal to the number of rows.

```
#include <stdio.h>
#include <lapacke.h>

int main (int argc, const char * argv[])
{
   double a[5][3] = {1,1,1,2,3,4,3,5,2,4,2,5,5,4,3};
   double b[5][2] = {-10,-3,12,14,14,12,16,16,18,16};
   lapack_int info,m,n,lda,ldb,nrhs;
   int i,j;

   m = 5;
   n = 3;
   nrhs = 2;
   lda = 3;
   ldb = 2;

   info = LAPACKE_dgels(LAPACK_ROW_MAJOR,'N',m,n,nrhs,*a,lda,*b,ldb);

   for(i=0;i<n;i++)
   {
      for(j=0;j<nrhs;j++)
      {
         printf("%lf ",b[i][j]);
      }
      printf("\n");
   }
   return(info);
}
```

FIGURE 1. Calling DGELS using row-major order.

3.4. **Calling CGEQRF and the CBLAS.** In this example, we will call the LAPACK routine CGEQRF to compute the QR factorization of a matrix. We then call CUNGQR to construct the Q matrix and then use the CBLAS routine CGEMM to compute $Q^H Q - I$ to check that Q is Hermitian. The error $||Q^H Q - I||$ is printed at the end of the program.

In the first version, given below in Figure 3, we let LAPACKE handle the memory allocation for the workspace internally. The second version, shown in Figure 4, uses the workspace query facility for both CGEQRF and CUNGQR to obtain the optimal size for the parameter lwork, which we use to allocate our own workspace in the array work.

```
#include <stdio.h>
#include <lapacke.h>

int main (int argc, const char * argv[])
{
   double a[5*3] = {1,2,3,4,5,1,3,5,2,4,1,4,2,5,3};
   double b[5*2] = {-10,12,14,16,18,-3,14,12,16,16};
   lapack_int info,m,n,lda,ldb,nrhs;
   int i,j;

   m = 5;
   n = 3;
   nrhs = 2;
   lda = 5;
   ldb = 5;

   info = LAPACKE_dgels(LAPACK_COL_MAJOR,'N',m,n,nrhs,a,lda,b,ldb);

   for(i=0;i<n;i++)
   {
      for(j=0;j<nrhs;j++)
      {
         printf("%lf ",b[i+ldb*j]);
      }
      printf("\n");
   }
   return(info);
}
```

FIGURE 2. Calling DGELS using column-major order.

```
#include <stdio.h>
#include <stdlib.h>
#include <lapacke.h>
#include <cblas.h>
int main (int argc, const char * argv[])
{
   lapack_complex_float *a,*tau,*r,one,zero;
   lapack_int info,m,n,lda;
   int i,j;
   float err=0.0;
   m = 10;   n = 5;   lda = m;
   one = lapack_make_complex_float(1.0,0.0);
   zero= lapack_make_complex_float(0.0,0.0);
   a = calloc(m*n,sizeof(lapack_complex_float));
   r = calloc(n*n,sizeof(lapack_complex_float));
   tau = calloc(m,sizeof(lapack_complex_float));
   for(j=0;j<n;j++)
      for(i=0;i<m;i++)
         a[i+j*m] = lapack_make_complex_float(i+1,j+1);
   info = LAPACKE_cgeqrf(LAPACK_COL_MAJOR,m,n,a,lda,tau);
   info = LAPACKE_cungqr(LAPACK_COL_MAJOR,m,n,n,a,lda,tau);
   for(j=0;j<n;j++)
      for(i=0;i<n;i++)
         r[i+j*n]=(i==j)?-one:zero;
   cblas_cgemm(CblasColMajor,CblasConjTrans,CblasNoTrans,
               n,n,m,&one,a,lda,a,lda,&one,r,n );
   for(i=0;i<n;i++)
      for(j=0;j<n;j++)
         err=MAX(err,cabs(r[i+j*n]));
   printf("error=%e\n",err);
   free(tau);
   free(r);
   free(a);
   return(info);
}
```

FIGURE 3. Calling CGEQRF and CUNGQR to compute Q.

```
#include <stdio.h>
#include <stdlib.h>
#include <lapacke.h>
#include <cblas.h>

int main (int argc, const char * argv[])
{
   lapack_complex_float *a,*tau,*r,*work,one,zero,query;
   lapack_int info,m,n,lda,lwork;
   int i,j;
   float err;
   m = 10;   n = 5;   lda = m;
   one = lapack_make_complex_float(1.0,0.0);
   zero= lapack_make_complex_float(0.0,0.0);
   a = calloc(m*n,sizeof(lapack_complex_float));
   r = calloc(n*n,sizeof(lapack_complex_float));
   tau = calloc(m,sizeof(lapack_complex_float));
   for(j=0;j<n;j++)
      for(i=0;i<m;i++)
         a[i+j*m] = lapack_make_complex_float(i+1,j+1);
   info = LAPACKE_cgeqrf_work(LAPACK_COL_MAJOR,m,n,a,lda,tau,&query,-1);
   lwork = (lapack_int)query;
   info = LAPACKE_cungqr_work(LAPACK_COL_MAJOR,m,n,n,a,lda,tau,&query,-1);
   lwork = MAX(lwork,(lapack_int)query);
   work = calloc(lwork,sizeof(lapack_complex_float));
   info = LAPACKE_cgeqrf_work(LAPACK_COL_MAJOR,m,n,a,lda,tau,work,lwork);
   info = LAPACKE_cungqr_work(LAPACK_COL_MAJOR,m,n,n,a,lda,tau,work,lwork);
   for(j=0;j<n;j++)
      for(i=0;i<n;i++)
         r[i+j*n]=(i==j)?-one:zero;
   cblas_cgemm(CblasColMajor,CblasConjTrans,CblasNoTrans,
               n,n,m,&one,a,lda,a,lda,&one,r,n);
   err=0.0;
   for(i=0;i<n;i++)
      for(j=0;j<n;j++)
         err=MAX(err,cabs(r[i+j*n]));
   printf("error=%e\n",err);
   free(work);
   free(tau);
   free(r);
   free(a);
   return(info);
}
```

FIGURE 4. Calling CGEQRF and CUNGQR with workspace querying.