## CONTENTS

1 Introduction .......................... 1
   1.1 Attributions ........................ 1

2 Framework Functions ................. 2
   2.1 culaInitialize ...................... 2
   2.2 culaShutdown ....................... 2
   2.3 culaGetLastStatus .................. 2
   2.4 culaGetStatusString ............... 2
   2.5 culaGetErrorInfo ................... 3
   2.6 culaFreeBuffers .................... 3
   2.7 culaGetDeviceCount ................. 3
   2.8 culaSelectDevice ................... 3
   2.9 culaGetExecutingDevice .......... 4
   2.10 culaGetDeviceInfo ................. 4
   2.11 culaDeviceMalloc ................. 4
   2.12 culaDeviceFree .................... 4

3 Linear Algebra Routines ............. 6
   3.1 Data Types .......................... 6
   3.2 Host Interface Compared To Device Interface ........... 6
   3.3 Note on Leading Dimensions ............ 6
   3.4 BDSQR ................................ 7
   3.5 GEBRD ............................... 9
   3.6 GEEV ................................ 12
   3.7 GEHRD ............................... 15
   3.8 GELQF ............................... 17
   3.9 GELS ................................ 18
   3.10 GEQRF ............................... 20
   3.11 GERQF ............................... 22
   3.12GESV ................................. 23
   3.13GESVD ............................... 25
   3.14GETRF ............................... 27
   3.15GETRI ............................... 29
   3.16GETRS ............................... 30
   3.17GGLSE ............................... 32
   3.18GGRQF ............................... 34
   3.19 ORGBR/UNGBR ......................... 37
   3.20 ORGHR/UNGHR ......................... 38
   3.21 ORGLQ/UNGLQ ......................... 40
4 Differences Between CULA and LAPACK
4.1 No Workspace Parameters

5 Common Errors
5.1 Pivot Arrays
5.2 Padding With Zeros
INTRODUCTION

This guide documents CULA’s Programming Interface. CULA™ is an implementation of the Linear Algebra PACK-age (LAPACK) interface for CUDA™-enabled NVIDIA® graphics processing units (GPUs). It is a companion document to the CULA Programmer’s Guide.

This guide is split into the following sections:

• **Framework Functions** - These section documents functions are used in initializing CULA, shutting it down, and querying information about errors.

• **Linear Algebra Routines** - This section documents the Linear Algebra functions that CULA provides

• **Differences Between CULA and LAPACK** - This section lists some of the ways in which CULA differs from LAPACK.

• **Common Errors** - This section lists some of the common errors that apply to usage of several functions.

1.1 **Attributions**

This work has been made possible by the NASA Small Business Innovation Research (SBIR) program. We recognize NVIDIA for their support.

CULA is built on NVIDIA CUDA 2.3 and NVIDIA CUBLAS.

CULA uses the Intel® Math Kernel Library (MKL) internally. For more information, please see the MKL product page at http://www.intel.com/software/products/mkl.

The original version of LAPACK from which CULA implements a similar interface can be obtained at http://www.netlib.org/lapack. Much of this Reference Manual is based upon the documentation released with netlib.
FRAMEWORK FUNCTIONS

This section documents the functions that are used in initializing CULA, shutting it down, and querying information about errors.

2.1 culaInitialize

Description
Initializes CULA Must be called before using any other function. Some functions have an exception to this rule: culaGetDeviceCount, culaSelectDevice

Returns
culaNoError on a successful initialization or the culaStatus enum that specifies an error

2.2 culaShutdown

Description
Shuts down CULA Must be called to deallocate CULA internal data

2.3 culaGetLastError

Description
Returns the last status code returned from a CULA function

Returns
The last CULA status code

2.4 culaGetStatusString

Description
Associates a culaStatus enum with a readable error string

Parameters
• e - A culaStatus error code

Returns
A string that corresponds with the specified culaStatus enum

2.5 culaGetErrorInfo

Description
This function is used to provide extended functionality that LAPACK’s info parameter typically provides

Returns
Extended information about the last error or zero if it is unavailable

2.6 culaFreeBuffers

Description
Releases any memory buffers stored internally by CULA

2.7 culaGetDeviceCount

Description
Reports the number of GPU devices Can be called before culaInitialize

Parameters
• num - Pointer to receive the number of devices

Returns
culaNoError on sucess, culaArgumentError on invalid pointer

2.8 culaSelectDevice

Description
Selects a device with which CULA will operate To bind without error, this function must be called before culaInitialize

Parameters
• dev - Specifies the device id of the GPU device

Returns
culaNoError on sucess, culaArgumentError on an invalid device id, culaRuntimeError if the running thread has already been bound to a GPU device
2.9 culaGetExecutingDevice

Description
Reports the id of the GPU device executing CULA

Parameters
- dev - Pointer to receive the GPU device number

Returns
culaNoError on success, culaArgumentError on invalid pointer

2.10 culaGetDeviceInfo

Description
Prints information to a buffer about a specified device

Parameters
- dev - CUDA device id to print information about
- buf - Pointer to a buffer into which information will be printed
- bufsize - The size of buf, printed information will not exceed bufsize

Returns
culaNoError on success, culaArgumentError on invalid buf pointer, invalid device id, or invalid bufsize

2.11 culaDeviceMalloc

Description
Allocates memory on the device in a pitch that is optimal for CULA

Parameters
- rows - The number of rows of the matrix
- cols - The number of columns of the matrix
- elesize - The size in bytes of the desired element
- pitch - The pitch of the allocation in bytes (where *pitch/elesize >= rows)

Returns
culaNoError on successful allocation, culaInsufficientMemory on failure

2.12 culaDeviceFree

Description
Frees memory that has been allocated with culaDeviceMalloc

Parameters
• mem - Pointer to a buffer that is to be freed

Returns

culaNoError on successful free, culaArgumentError on failure
This section documents the Linear Algebra functions that CULA provides. For each function, a high-level description of that function is given, followed by a listing of each of the function’s parameters. Where applicable, differences from LAPACK will also be listed.

### 3.1 Data Types

CULA provides 4 data types with which you can perform computations, with one function for each data type. Rather than document each routine separately, this guide includes only one reference for each of the functions, and instead documents the differences between each of these functions (if any) in the generic function description.

Most functions only take pointers to one data type that applies to the S, D, C, or Z variant of the function in question. For the majority of functions, these parameters will be denoted as S/D/C/Z. For those functions that have parameters that differ from their variant, the difference will be noted. For example, for a ‘C/Z’ function that has real (non-complex) parameters, these parameters will be denoted as S/D (although others may be denoted as S/D/C/Z). For an ‘S’ function that has complex parameters, these parameters will be denoted as (C/Z).

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Host Interface Type</th>
<th>Device Interface Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>culaFloat</td>
<td>culaDeviceFloat</td>
</tr>
<tr>
<td>D</td>
<td>culaDouble</td>
<td>culaDeviceDouble</td>
</tr>
<tr>
<td>C</td>
<td>culaFloatComplex</td>
<td>culaDeviceFloatComplex</td>
</tr>
<tr>
<td>Z</td>
<td>culaDoubleComplex</td>
<td>culaDeviceDoubleComplex</td>
</tr>
</tbody>
</table>

### 3.2 Host Interface Compared To Device Interface

For Host interface functions, all matrices/vectors are submitted as pointers to host data. For the Device interface, all matrices/vectors are submitted as pointers to GPU data, as allocated by either the CUDA toolkit or via `cudaDeviceMalloc` (available in CULA premium). All Device interface routines have the word `Device` as part of the function name; all other functions are Host interface.

### 3.3 Note on Leading Dimensions

All LAPACK matrices are specified as a pointer and a “leading dimension” parameter. The leading dimension describes the allocated size of the matrix, which may be equal to or larger than the actual matrix height. Thus if a matrix input is described as size “(LDA,N)” it simply means that the storage for the matrix is at least \( LDA \times N \) in size. The section of that array that contains valid data will be described by other parameters, often \( M \) and \( N \). There will typically be a note differentiating between these.
3.4 BDSQR

CULA Routines

The BDSQR functionality is implemented by the following CULA routines:

- **Host Memory**
  - `culaSbdsqr`
  - `culaDbdsqr`
  - `culaCbdsqr`
  - `culaZbdsqr`
  - `culaBdsqr` (C++ style, type overloaded)

- **Device Memory**
  - `culaDeviceSbdsqr`
  - `culaDeviceDbdsqr`
  - `culaDeviceCbdsqr`
  - `culaDeviceZbdsqr`
  - `culaDeviceBdsqr` (C++ style, type overloaded)

Description

BDSQR computes the singular values and, optionally, the right and/or left singular vectors from the singular value decomposition (SVD) of a real N-by-N (upper or lower) bidiagonal matrix B using the implicit zero-shift QR algorithm.

The SVD of B has the form

\[ B = Q \cdot S \cdot P^T \]

where S is the diagonal matrix of singular values, Q is an orthogonal matrix of left singular vectors, and P is an orthogonal matrix of right singular vectors. If left singular vectors are requested, this subroutine actually returns U*Q instead of Q, and, if right singular vectors are requested, this subroutine returns P^T * VT instead of P^T, for given real input matrices U and VT. When U and VT are the orthogonal/unitary matrices that reduce a general matrix A to bidiagonal form: A = U * B * VT, as computed by `GEBRD`, then

\[ A = (U * Q) * S * (P^T * VT) \]

is the SVD of A. Optionally, the subroutine may also compute Q^T * C for a given real input matrix C.


Parameters

- **uplo**
  - Type: char
  - Direction: Input
  = ‘U’: B is upper bidiagonal; = ‘L’: B is lower bidiagonal.

- **n**
  - Type: int
The order of the matrix B. N >= 0.

- **ncvt**
  - Type: int
  - Direction: Input
  The number of columns of the matrix VT. NCVT >= 0.

- **nru**
  - Type: int
  - Direction: Input
  The number of rows of the matrix U. NRU >= 0.

- **ncc**
  - Type: int
  - Direction: Input
  The number of columns of the matrix C. NCC >= 0.

- **d**
  - Type: S/D Pointer
  - Direction: Input/Output
  - Dimension: (N)
  On entry, the n diagonal elements of the bidiagonal matrix B.
  On exit, if INFO=0, the singular values of B in decreasing order.

- **e**
  - Type: S/D Pointer
  - Direction: Input/Output
  - Dimension: (N-1)
  On entry, the N-1 offdiagonal elements of the bidiagonal matrix B.
  On exit, if INFO = 0, E is destroyed; if INFO > 0, D and E will contain the diagonal and superdiagonal elements of a bidiagonal matrix orthogonally equivalent to the one given as input.

- **vt**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: (LDVT, NCVT)
  On entry, an N-by-NCVT matrix VT.
  On exit, VT is overwritten by P^T * VT. Not referenced if NCVT = 0.

- **ldvt**
  - Type: int
The leading dimension of the array VT. LDVT >= max(1,N) if NCVT > 0; LDVT >= 1 if NCVT = 0.

- **u**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: (LDU, N)
  On entry, an NRU-by-N matrix U.
  On exit, U is overwritten by U * Q.
  Not referenced if NRU = 0.

- **ldu**
  - Type: int
  - Direction: Input
  The leading dimension of the array U. LDU >= max(1,NRU).

- **c**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: (LDC, NCC)
  On entry, an N-by-NCC matrix C.
  On exit, C is overwritten by Qᵀ * C.
  Not referenced if NCC = 0.

- **ldc**
  - Type: int
  - Direction: Input
  The leading dimension of the array C. LDC >= max(1,N) if NCC > 0; LDC >=1 if NCC = 0.

**Differences from LAPACK**

See No Workspace Parameters section.

### 3.5 GEBRD

**CULA Routines**

The GEBRD functionality is implemented by the following CULA routines:

- **Host Memory**
  - culaSgebrd
  - culaDgebrd
  - culaCgebrd
  - culaZgebrd
GEBRD reduces a general real M-by-N matrix A to upper or lower bidiagonal form B by an orthogonal/unitary transformation: $Q^T A P = B$.

If $m \geq n$, B is upper bidiagonal; if $m < n$, B is lower bidiagonal.

**Parameters**

- **m**
  - Type: int
  - Direction: Input
  
  The number of rows in the matrix A. $M \geq 0$.

- **n**
  - Type: int
  - Direction: Input
  
  The number of columns in the matrix A. $N \geq 0$.

- **a**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: $(LDA, N)$

  On entry, the M-by-N general matrix to be reduced.
  On exit, if $m \geq n$, the diagonal and the first superdiagonal are overwritten with the upper bidiagonal matrix B; the elements below the diagonal, with the array TAUQ, represent the orthogonal/unitary matrix Q as a product of elementary reflectors, and the elements above the first superdiagonal, with the array TAUP, represent the orthogonal/unitary matrix P as a product of elementary reflectors;
  On exit, if $m < n$, the diagonal and the first subdiagonal are overwritten with the lower bidiagonal matrix B; the elements below the first subdiagonal, with the array TAUQ, represent the orthogonal/unitary matrix Q as a product of elementary reflectors, and the elements above the diagonal, with the array TAUP, represent the orthogonal/unitary matrix P as a product of elementary reflectors. See Further Details.

- **lda**
  - Type: int
  - Direction: Input

  The leading dimension of the array A. $LDA \geq \max(1, M)$.
- Type: S/D Pointer
- Direction: Output
- Dimension: \((\min(M, N))\)

The diagonal elements of the bidiagonal matrix \(B\): \(D(i) = A(i,i)\).

**\(e\)**
- Type: S/D Pointer
- Direction: Output
- Dimension: \((\min(M, N) - 1)\)

The off-diagonal elements of the bidiagonal matrix \(B\): if \(m \geq n\), \(E(i) = A(i,i+1)\) for \(i = 1, 2, ..., n-1\); if \(m < n\), \(E(i) = A(i+1,i)\) for \(i = 1, 2, ..., m-1\).

**\(\tau_{aue}\)**
- Type: S/D/C/Z Pointer
- Direction: Output
- Dimension: \((\min(M, N))\)

The scalar factors of the elementary reflectors which represent the orthogonal/unitary matrix \(Q\). See Further Details.

**\(\tau_{aup}\)**
- Type: S/D/C/Z Pointer
- Direction: Output
- Dimension: \((\min(M, N))\)

The scalar factors of the elementary reflectors which represent the orthogonal/unitary matrix \(P\). See Further Details.

**Further Details**

The matrices \(Q\) and \(P\) are represented as products of elementary reflectors:

If \(m \geq n\),
\[
Q = H(1) H(2) \ldots H(n) \quad \text{and} \quad P = G(1) G(2) \ldots G(n-1)
\]

Each \(H(i)\) and \(G(i)\) has the form:
\[
H(i) = I - \tau_{aue} v v' \quad \text{and} \quad G(i) = I - \tau_{aup} u u'
\]
where \(\tau_{aue}\) and \(\tau_{aup}\) are real scalars, and \(v\) and \(u\) are real vectors; \(v(1:i-1) = 0, v(i) = 1, \) and \(v(i+1:m)\) is stored on exit in \(A(i+1:m,i)\); \(u(1:i) = 0, u(i+1) = 1, \) and \(u(i+1:n)\) is stored on exit in \(A(i,i+1:n)\); \(\tau_{aue}\) is stored in \(\text{TAUQ}(i)\) and \(\tau_{aup}\) in \(\text{TAUP}(i)\).

If \(m < n\),
\[
Q = H(1) H(2) \ldots H(m-1) \quad \text{and} \quad P = G(1) G(2) \ldots G(m)
\]

Each \(H(i)\) and \(G(i)\) has the form:
\[
H(i) = I - \tau_{aue} v v' \quad \text{and} \quad G(i) = I - \tau_{aup} u u'
\]
where \(\tau_{aue}\) and \(\tau_{aup}\) are real scalars, and \(v\) and \(u\) are real vectors; \(v(1:i) = 0, v(i+1) = 1, \) and \(v(i+2:m)\) is stored on exit in \(A(i+2:m,i)\); \(u(1:i-1) = 0, u(i) = 1, \) and \(u(i+1:n)\) is stored on exit in \(A(i,i+1:n)\); \(\tau_{aue}\) is stored in \(\text{TAUQ}(i)\) and \(\tau_{aup}\) in \(\text{TAUP}(i)\).
The contents of A on exit are illustrated by the following examples:

\[
\begin{align*}
\text{m = 6 and n = 5 (m > n):} & & \text{m = 5 and n = 6 (m < n):} \\
( d & e & u1 & u1 & u1 ) & & ( d & u1 & u1 & u1 & u1 ) \\
( v1 & v2 & d & e & u2 & u2 ) & & ( e & d & u2 & u2 & u2 ) \\
( v1 & v2 & v3 & d & e & e ) & & ( v1 & e & d & u3 & u3 & u3 ) \\
( v1 & v2 & v3 & v4 & d & ) & & ( v1 & v2 & e & d & u4 & u4 ) \\
( v1 & v2 & v3 & v4 & v5 ) & & ( v1 & v2 & v3 & e & d & u5 ) \\
\end{align*}
\]

where d and e denote diagonal and off-diagonal elements of B, vi denotes an element of the vector defining H(i), and ui an element of the vector defining G(i).

**Differences from LAPACK**

See *No Workspace Parameters* section.

### 3.6 GEEV

**CULA Routines**

The GEEV functionality is implemented by the following CULA routines:

- **Host Memory**
  - `culaSgeev`
  - `culaDgeev`
  - `culaCgeev`
  - `culaZgeev`
  - `culaGeev` (C++ style, type overloaded)

- **Device Memory**
  - `culaDeviceSgeev`
  - `culaDeviceDgeev`
  - `culaDeviceCgeev`
  - `culaDeviceZgeev`
  - `culaDeviceGeev` (C++ style, type overloaded)

**Description**

GEEV computes for an N-by-N real nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors.

The right eigenvector \( v(j) \) of A satisfies

\[
A \ast v(j) = \lambda_j \ast v(j)
\]

where \( \lambda_j \) is its eigenvalue.

The left eigenvector \( u(j) \) of A satisfies

\[
u(j)^H \ast A = \lambda_j \ast u(j)^H
\]
where \( u(j)^H \) denotes the conjugate transpose of \( u(j) \).

The computed eigenvectors are normalized to have Euclidean norm equal to 1 and largest component real.

**Parameters**

- **jobvl**
  - Type: char
  - Direction: Input
  - = ‘N’: left eigenvectors of \( A \) are not computed; = ‘V’: left eigenvectors of \( A \) are computed.

- **jobvr**
  - Type: char
  - Direction: Input
  - = ‘N’: right eigenvectors of \( A \) are not computed; = ‘V’: right eigenvectors of \( A \) are computed.

- **n**
  - Type: int
  - Direction: Input
  - The order of the matrix \( A \). \( N \geq 0 \).

- **a**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: (LDA, N)
  - On entry, the \( N \)-by-\( N \) matrix \( A \).
  - On exit, \( A \) has been overwritten.

- **lda**
  - Type: int
  - Direction: Input
  - The leading dimension of the array \( A \). \( LDA \geq \text{max}(1,N) \).

- **wr** (Real variants (S/D) only)
  - Type: S/D Pointer
  - Direction: Output
  - Dimension: (N)

- **wi** (Real variants (S/D) only)
  - Type: S/D Pointer
  - Direction: Output
  - Dimension: (N)

  \( WR \) and \( WI \) contain the real and imaginary parts, respectively, of the computed eigenvalues. Complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part first.

- **w** (Complex variants (C/Z) only)
- **W** contains the computed eigenvalues.

Note: the \( w_i \) and \( w_r \) parameters only appear in the real (non-complex) variants of geev; in the complex variants these are bundled into one \( w \).

- **vl**
  - **Type:** S/D/C/Z Pointer
  - **Direction:** Output
  - **Dimension:** \((LDVL, N)\)
  
  If \( JOBVL = 'V' \), the left eigenvectors \( u(j) \) are stored one after another in the columns of \( VL \), in the same order as their eigenvalues.
  
  If \( JOBVL = 'N' \), \( VL \) is not referenced.
  
  For real variants (S/D): If the \( j \)-th eigenvalue is real, then \( u(j) = VL(:,j) \), the \( j \)-th column of \( VL \). If the \( j \)-th and \((j+1)\)-st eigenvalues form a complex conjugate pair, then \( u(j) = VL(:,j) + i*VL(:,j+1) \) and \( u(j+1) = VL(:,j) - i*VL(:,j+1) \).
  
  For complex variants (C/Z): \( u(j) = VL(:,j) \), the \( j \)-th column of \( VL \).

- **ldvl**
  - **Type:** int
  - **Direction:** Input
  
  The leading dimension of the array \( VL \). \( LDVL >= 1 \); if \( JOBVL = 'V' \), \( LDVL >= N \).

- **vr**
  - **Type:** S/D/C/Z Pointer
  - **Direction:** Output
  - **Dimension:** \((LDVR, N)\)
  
  If \( JOBVR = 'V' \), the right eigenvectors \( v(j) \) are stored one after another in the columns of \( VR \), in the same order as their eigenvalues.
  
  If \( JOBVR = 'N' \), \( VR \) is not referenced.
  
  If the \( j \)-th eigenvalue is real, then \( v(j) = VR(:,j) \), the \( j \)-th column of \( VR \). If the \( j \)-th and \((j+1)\)-st eigenvalues form a complex conjugate pair, then \( v(j) = VR(:,j) + i*VR(:,j+1) \) and \( v(j+1) = VR(:,j) - i*VR(:,j+1) \).
  
  For complex variants (C/Z): \( v(j) = VR(:,j) \), the \( j \)-th column of \( VR \).

- **ldvr**
  - **Type:** int
  - **Direction:** Input
  
  The leading dimension of the array \( VR \). \( LDVR >= 1 \); if \( JOBVR = 'V' \), \( LDVR >= N \).

**Differences from LAPACK**

See No Workspace Parameters section.
3.7 GEHRD

CULA Routines

The GEHRD functionality is implemented by the following CULA routines:

- **Host Memory**
  - culaSgehrd
  - culaDgehrd
  - culaCgehrd
  - culaZgehrd
  - culaGehrd (C++ style, type overloaded)

- **Device Memory**
  - culaDeviceSgehrd
  - culaDeviceDgehrd
  - culaDeviceCgehrd
  - culaDeviceZgehrd
  - culaDeviceGehrd (C++ style, type overloaded)

Description

GEHRD reduces a complex general matrix A to upper Hessenberg form H by an unitary similarity transformation: Q’ * A * Q = H.

Parameters

- **N**
  - Type: int
  - Direction: Input
  
The order of the matrix A. N >= 0.

- **ILO**
  - Type: int
  - Direction: Input

- **IHI**
  - Type: int
  - Direction: Input

It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to GEBAL; otherwise they should be set to 1 and N respectively. See Further Details. 1 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, if N=0.

- **A**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
On entry, the N-by-N general matrix to be reduced.

On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the elements below the first subdiagonal, with the array TAU, represent the unitary matrix Q as a product of elementary reflectors. See Further Details.

- **LDA**
  - Type: int
  - Direction: Input
  The leading dimension of the array A. LDA >= max(1,N).

- **TAU**
  - Type: S/D/C/Z Pointer
  - Direction: Output
  - Dimension: (N-1)
  The scalar factors of the elementary reflectors (see Further Details). Elements 1:ILO-1 and IHI:N-1 of TAU are set to zero.

Further Details
The matrix Q is represented as a product of (ihi-ilo) elementary reflectors

\[ Q = H(ilo) H(ilo+1) \ldots H(ihi-1). \]

Each H(i) has the form

\[ H(i) = I - \tau \cdot v \cdot v' \]

where \( \tau \) is a complex scalar, and \( v \) is a complex vector with \( v(1:i) = 0, v(i+1) = 1 \) and \( v(ihi+1:n) = 0; v(i+2:ihi) \) is stored on exit in \( A(i+2:ihi,i) \), and \( \tau \) in TAU(i).

The contents of A are illustrated by the following example, with \( n = 7, \) ilo = 2 and ihi = 6:

<table>
<thead>
<tr>
<th>on entry,</th>
<th>on exit,</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (a \ a \ a \ a \ a \ a \ a) )</td>
<td>( (a \ a \ h \ h \ h \ h \ a) )</td>
</tr>
<tr>
<td>( (a \ a \ a \ a \ a \ a) )</td>
<td>( (a \ h \ h \ h \ h \ a) )</td>
</tr>
<tr>
<td>( (a \ a \ a \ a \ a \ a) )</td>
<td>( (h \ h \ h \ h \ h \ h) )</td>
</tr>
<tr>
<td>( (a \ a \ a \ a \ a \ a) )</td>
<td>( (v2 \ h \ h \ h \ h \ h) )</td>
</tr>
<tr>
<td>( (a \ a \ a \ a \ a \ a) )</td>
<td>( (v2 \ v3 \ h \ h \ h \ h) )</td>
</tr>
<tr>
<td>( (a \ a \ a \ a \ a \ a) )</td>
<td>( (v2 \ v3 \ v4 \ h \ h \ h) )</td>
</tr>
<tr>
<td>( (a) )</td>
<td>( (a) )</td>
</tr>
</tbody>
</table>

where \( a \) denotes an element of the original matrix A, \( h \) denotes a modified element of the upper Hessenberg matrix H, and \( v_i \) denotes an element of the vector defining H(i).

This file is a slight modification of LAPACK-3.0’s GEHRD subroutine incorporating improvements proposed by Quintana-Orti and Van de Geijn (2006). (See DLAHR2.)

Differences from LAPACK
See No Workspace Parameters section.
3.8 GELQF

CULA Routines

The GELQF functionality is implemented by the following CULA routines:

- Host Memory
  - culaSgelqf
  - culaDgelqf
  - culaCgelqf
  - culaZgelqf
  - culaGelqf (C++ style, type overloaded)

- Device Memory
  - culaDeviceSgelqf
  - culaDeviceDgelqf
  - culaDeviceCgelqf
  - culaDeviceZgelqf
  - culaDeviceGelqf (C++ style, type overloaded)

Description

GELQF computes an LQ factorization of a real M-by-N matrix A: \( A = L \times Q \).

Parameters

- **m**
  - Type: int
  - Direction: Input
  - The number of rows of the matrix A. \( M \geq 0 \).

- **n**
  - Type: int
  - Direction: Input
  - The number of columns of the matrix A. \( N \geq 0 \).

- **a**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: (LDA, N)
  - On entry, the M-by-N matrix A.
  - On exit, the elements on and below the diagonal of the array contain the \( m \)-by-min(\( m \),n) lower trapezoidal matrix L (L is lower triangular if \( m \leq n \)); the elements above the diagonal, with the array TAU, represent the orthogonal/unitary matrix Q as a product of elementary reflectors (see Further Details).

- **lda**
• **tau**
  
  - **Type:** S/D/C/Z Pointer
  - **Direction:** Output
  - **Dimension:** (min(M, N))

  The scalar factors of the elementary reflectors (see Further Details).

**Further Details**

The matrix Q is represented as a product of elementary reflectors

\[ Q = H(k) \ldots H(2) H(1), \text{ where } k = \min(m,n) \]

Each H(i) has the form

\[ H(i) = I - \tau \cdot v \cdot v' \]

where tau is a real scalar, and v is a real vector with v(1:i-1) = 0 and v(i) = 1; v(i+1:n) is stored on exit in A(i,i+1:n), and tau in TAU(i).

**Differences from LAPACK**

See *No Workspace Parameters* section.

### 3.9 GELS

**CULA Routines**

The GELS functionality is implemented by the following CULA routines:

- **Host Memory**
  - `culaSgels`
  - `culaDgels`
  - `culaCgels`
  - `culaZgels`
  - `culaGels` (C++ style, type overloaded)

- **Device Memory**
  - `culaDeviceSgels`
  - `culaDeviceDgels`
  - `culaDeviceCgels`
  - `culaDeviceZgels`
  - `culaDeviceGels` (C++ style, type overloaded)

**Description**

GELS solves overdetermined or underdetermined real linear systems involving an M-by-N matrix A, or its transpose, using a QR or LQ factorization of A. It is assumed that A has full rank.
The following options are provided:

1. If TRANS = ‘N’ and m >= n: find the least squares solution of an overdetermined system, i.e., solve the least squares problem minimize \( \| B - A^*X \| \).
2. If TRANS = ‘N’ and m < n: find the minimum norm solution of an underdetermined system \( A \ast X = B \).
3. If TRANS = ‘T’ and m >= n: find the minimum norm solution of an underdetermined system \( A^{\text{T}} \ast X = B \).
4. If TRANS = ‘T’ and m < n: find the least squares solution of an overdetermined system, i.e., solve the least squares problem minimize \( \| B - A^{\text{T}} \ast X \| \).

Several right hand side vectors \( b \) and solution vectors \( x \) can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix \( B \) and the N-by-NRHS solution matrix \( X \).

Parameters

• trans
  - Type: char
  - Direction: Input
    = ‘N’: the linear system involves \( A \); = ‘T’: the linear system involves \( A^{\text{T}} \).

• m
  - Type: int
  - Direction: Input
    The number of rows of the matrix \( A \). \( M \geq 0 \).

• n
  - Type: int
  - Direction: Input
    The number of columns of the matrix \( A \). \( N \geq 0 \).

• nrhs
  - Type: int
  - Direction: Input
    The number of right hand sides, i.e., the number of columns of the matrices \( B \) and \( X \). NRHS \( \geq 0 \).

• a
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: (LDA, N)
    On entry, the M-by-N matrix \( A \).
    On exit, if \( M \geq N \), \( A \) is overwritten by details of its QR factorization as returned by \( \text{GEQRF} \);
    if \( M < N \), \( A \) is overwritten by details of its LQ factorization as returned by \( \text{GELQF} \).

• lda
  - Type: int
  - Direction: Input
    The leading dimension of the array \( A \). LDA \( \geq \max(1,M) \).
• **b**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: (LDB,NRHS)

  On entry, the matrix B of right hand side vectors, stored columnwise; B is M-by-NRHS if TRANS = ‘N’, or N-by-NRHS if TRANS = ‘T’.

  On exit, if INFO = 0, B is overwritten by the solution vectors, stored columnwise:
  - if TRANS = ‘N’ and m >= n, rows 1 to n of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of elements N+1 to M in that column;
  - if TRANS = ‘N’ and m < n, rows 1 to N of B contain the minimum norm solution vectors;
  - if TRANS = ‘T’ and m >= n, rows 1 to M of B contain the minimum norm solution vectors;
  - if TRANS = ‘T’ and m < n, rows 1 to M of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of elements M+1 to N in that column.

• **ldb**
  - Type: int
  - Direction: Input

  The leading dimension of the array B. LDB >= MAX(1,M,N).

**Differences from LAPACK**

See *No Workspace Parameters* section.

### 3.10 GEQRF

**CULA Routines**

The GEQRF functionality is implemented by the following CULA routines:

- **Host Memory**
  - culaSgeqrf
  - culaDgeqrf
  - culaCgeqrf
  - culaZgeqrf
  - culaGeqrf (C++ style, type overloaded)

- **Device Memory**
  - culaDeviceSgeqrf
  - culaDeviceDgeqrf
  - culaDeviceCgeqrf
  - culaDeviceZgeqrf
  - culaDeviceGeqrf (C++ style, type overloaded)
Description
GEQRF computes a QR factorization of a real M-by-N matrix A: A = Q * R.

Parameters
• m
  - Type: int
  - Direction: Input
  The number of rows of the matrix A. M >= 0.
• n
  - Type: int
  - Direction: Input
  The number of columns of the matrix A. N >= 0.
• a
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: (LDA,N)
  On entry, the M-by-N matrix A.
  On exit, the elements on and above the diagonal of the array contain the min(M,N)-by-N upper trapezoidal matrix R (R is upper triangular if m >= n); the elements below the diagonal, with the array TAU, represent the orthogonal/unitary matrix Q as a product of min(m,n) elementary reflectors (see Further Details).
• lda
  - Type: int
  - Direction: Input
  The leading dimension of the array A. LDA >= max(1,M).
• tau
  - Type: S/D/C/Z Pointer
  - Direction: Output
  - Dimension: (min(M,N))
  The scalar factors of the elementary reflectors (see Further Details).

Further Details
The matrix Q is represented as a product of elementary reflectors
Q = H(1) H(2) . . . H(k), where k = min(m,n).
Each H(i) has the form
H(i) = I - tau * v * v'
where tau is a real scalar, and v is a real vector with v(1:i-1) = 0 and v(i) = 1; v(i+1:m) is stored on exit in A(i+1:m,i), and tau in TAU(i).

Differences from LAPACK
See No Workspace Parameters section.

3.11 GERQF

CULA Routines
The GERQF functionality is implemented by the following CULA routines:

- Host Memory
  - culaSgerqf
  - culaDgerqf
  - culaCgerqf
  - culaZgerqf
  - culaGerqf (C++ style, type overloaded)

- Device Memory
  - culaDeviceSgerqf
  - culaDeviceDgerqf
  - culaDeviceCgerqf
  - culaDeviceZgerqf
  - culaDeviceGerqf (C++ style, type overloaded)

Description
GERQF computes an RQ factorization of a real M-by-N matrix A: A = R * Q.

Parameters

- m
  - Type: int
  - Direction: Input
  The number of rows of the matrix A. M >= 0.

- n
  - Type: int
  - Direction: Input
  The number of columns of the matrix A. N >= 0.

- a
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: (LDA, N)
  On entry, the M-by-N matrix A.
  On exit, if m <= n, the upper triangle of the subarray A(1:m,n-m+1:n) contains the M-by-M upper triangular matrix R; if m >= n, the elements on and above the (m-n)-th subdiagonal contain
the M-by-N upper trapezoidal matrix \( R \); the remaining elements, with the array \( TAU \), represent the orthogonal/unitary matrix \( Q \) as a product of \( \min(m,n) \) elementary reflectors (see Further Details).

- **lda**
  
  - Type: int
  
  - Direction: Input
  
  The leading dimension of the array \( A \). \( LDA \geq \max(1,M) \).

- **tau**
  
  - Type: S/D/C/Z Pointer
  
  - Direction: Output
  
  - Dimension: \( (\min(M,N)) \)

  The scalar factors of the elementary reflectors (see Further Details).

**Further Details**

The matrix \( Q \) is represented as a product of elementary reflectors

\[
Q = H(1) H(2) \ldots H(k), \text{ where } k = \min(m,n).
\]

Each \( H(i) \) has the form

\[
H(i) = I - \tau \cdot v \cdot v' \]

where \( \tau \) is a real scalar, and \( v \) is a real vector with \( v(n-k+i+1:n) = 0 \) and \( v(n-k+i) = 1 \); \( v(1:n-k+i-1) \) is stored on exit in \( A(m-k+i,1:n-k+i-1) \), and \( \tau \) in \( TAU(i) \).

**Differences from LAPACK**

See No Workspace Parameters section.

### 3.12 GESV

The GESV functionality is implemented by the following CULA routines:

- **Host Memory**
  
  - cudaSgesv
  
  - cudaDgesv
  
  - cudaCgesv
  
  - cudaZgesv
  
  - cudaGesv (C++ style, type overloaded)

- **Device Memory**
  
  - cudaDeviceSgesv
  
  - cudaDeviceDgesv
  
  - cudaDeviceCgesv
  
  - cudaDeviceZgesv
  
  - cudaDeviceGesv (C++ style, type overloaded)
Description

GESV computes the solution to a real system of linear equations

\[ A \times X = B, \]

where \( A \) is an \( N \)-by-\( N \) matrix and \( X \) and \( B \) are \( N \)-by-\( NRHS \) matrices.

The LU decomposition with partial pivoting and row interchanges is used to factor \( A \) as

\[ A = P \times L \times U, \]

where \( P \) is a permutation matrix, \( L \) is unit lower triangular, and \( U \) is upper triangular. The factored form of \( A \) is then used to solve the system of equations \( A \times X = B \).

Parameters

- **n**
  - Type: int
  - Direction: Input
  
  The number of linear equations, i.e., the order of the matrix \( A \). \( N \geq 0 \).

- **nrhs**
  - Type: int
  - Direction: Input
  
  The number of right hand sides, i.e., the number of columns of the matrix \( B \). \( NRHS \geq 0 \).

- **a**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: \((LDA, N)\)

  On entry, the \( N \)-by-\( N \) coefficient matrix \( A \).
  
  On exit, the factors \( L \) and \( U \) from the factorization \( A = P*L*U \); the unit diagonal elements of \( L \) are not stored.

- **lda**
  - Type: int
  - Direction: Input
  
  The leading dimension of the array \( A \). \( LDA \geq \max(1,N) \).

- **ipiv**
  - Type: INTEGER array
  - Direction: Output
  - Dimension: \((N)\)

  The pivot indices that define the permutation matrix \( P \); row \( i \) of the matrix was interchanged with row \( IPIV(i) \).

- **b**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
– Dimension: (LDB, NRHS)
On entry, the N-by-NRHS matrix of right hand side matrix B.
On exit, if INFO = 0, the N-by-NRHS solution matrix X.

• **ldb**
  – Type: int
  – Direction: Input
  The leading dimension of the array B. LDB >= max(1,N).

Further Details
See *Pivot Arrays* section.

### 3.13 GESVD

#### CULA Routines

The GESVD functionality is implemented by the following CULA routines:

- **Host Memory**
  - `culaSgesvd`
  - `culaDgesvd`
  - `culaCgesvd`
  - `culaZgesvd`
  - `culaGesvd` (C++ style, type overloaded)

- **Device Memory**
  - `culaDeviceSgesvd`
  - `culaDeviceDgesvd`
  - `culaDeviceCgesvd`
  - `culaDeviceZgesvd`
  - `culaDeviceGesvd` (C++ style, type overloaded)

#### Description

GESVD computes the singular value decomposition (SVD) of a real M-by-N matrix A, optionally computing the left and/or right singular vectors. The SVD is written

\[ A = U \cdot \text{SIGMA} \cdot \text{transpose}(V) \]

where SIGMA is an M-by-N matrix which is zero except for its min(m,n) diagonal elements, U is an M-by-M orthogonal/unitary matrix, and V is an N-by-N orthogonal/unitary matrix. The diagonal elements of SIGMA are the singular values of A; they are real and non-negative, and are returned in descending order. The first min(m,n) columns of U and V are the left and right singular vectors of A.

Note that the routine returns V\(^T\), not V. This \(^T\) output of GESVD is notable because other implementations, such as Matlab, return the non-transposed version via syntax like `[U S V] = svd(A)`. The V matrix then needs to be transposed to reconstruct the original matrix, such as `U*S*V'`. LAPACK avoids this by pre-transposing this output, but for those working with both LAPACK and Matlab code, this is a common pitfall.

#### Parameters
• **jobu**
  - Type: `char`
  - Direction: `Input`
  Specifies options for computing all or part of the matrix U:
  = `A`: all M columns of U are returned in array U;
  = `S`: the first min(m,n) columns of U (the left singular vectors) are returned in the array U;
  = `O`: the first min(m,n) columns of U (the left singular vectors) are overwritten on the array A;
  = `N`: no columns of U (no left singular vectors) are computed.

• **jobvt**
  - Type: `char`
  - Direction: `Input`
  Specifies options for computing all or part of the matrix $V^T$:
  = `A`: all N rows of $V^T$ are returned in the array VT;
  = `S`: the first min(m,n) rows of $V^T$ (the right singular vectors) are returned in the array VT;
  = `O`: the first min(m,n) rows of $V^T$ (the right singular vectors) are overwritten on the array A;
  = `N`: no rows of $V^T$ (no right singular vectors) are computed.
  JOBVT and JOBU cannot both be `O`.

• **m**
  - Type: `int`
  - Direction: `Input`
  The number of rows of the input matrix A. $M \geq 0$.

• **n**
  - Type: `int`
  - Direction: `Input`
  The number of columns of the input matrix A. $N \geq 0$.

• **a**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input/Output`
  - Dimension: `(LDA, N)`
  On entry, the M-by-N matrix A.
  On exit,
  if JOBU = `O`, A is overwritten with the first min(m,n) columns of U (the left singular vectors, stored columnwise);
  if JOBVT = `O`, A is overwritten with the first min(m,n) rows of $V^T$ (the right singular vectors, stored rowwise);
  if JOBU .ne. `O` and JOBVT .ne. `O`, the contents of A are destroyed.

• **lda**
• s
  - Type: S/D Pointer
  - Direction: Output
  - Dimension: \((\min(M,N))\)
  The singular values of A, sorted so that \(S(i) \geq S(i+1)\).

• u
  - Type: S/D/C/Z Pointer
  - Direction: Output
  - Dimension: \((LDU, UCOL)\)
    \((LDU,M)\) if \(JOBU = 'A'\) or \((LDU,\min(M,N))\) if \(JOBU = 'S'\). If \(JOBU = 'A'\), \(U\) contains the \(M\)-by-\(M\) orthogonal/unitary matrix \(U\); if \(JOBU = 'S'\), \(U\) contains the first \(\min(m,n)\) columns of \(U\) (the left singular vectors, stored columnwise); if \(JOBU = 'N'\) or \('O'\), \(U\) is not referenced.

• ldu
  - Type: int
  - Direction: Input
  The leading dimension of the array \(U\). \(LDU \geq 1\); if \(JOBU = 'S'\) or \('A'\), \(LDU \geq M\).

• vt
  - Type: S/D/C/Z Pointer
  - Direction: Output
  - Dimension: \((LDVT,N)\)
    If \(JOBVT = 'A'\), \(VT\) contains the \(N\)-by-\(N\) orthogonal/unitary matrix \(V^T\); if \(JOBVT = 'S'\), \(VT\) contains the first \(\min(m,n)\) rows of \(V^T\) (the right singular vectors, stored rowwise); if \(JOBVT = 'N'\) or \('O'\), \(VT\) is not referenced.

• ldvt
  - Type: int
  - Direction: Input
  The leading dimension of the array \(VT\). \(LDVT \geq 1\); if \(JOBVT = 'A'\), \(LDVT \geq N\); if \(JOBVT = 'S'\), \(LDVT \geq \min(M,N)\).

Differences from LAPACK
See No Workspace Parameters section.

3.14 GETRF

CULA Routines
The GETRF functionality is implemented by the following CULA routines:
• Host Memory
  - culaSgetrf
  - culaDgetrf
  - culaCgetrf
  - culaZgetrf
  - culaGetrf (C++ style, type overloaded)

• Device Memory
  - culaDeviceSgetrf
  - culaDeviceDgetrf
  - culaDeviceCgetrf
  - culaDeviceZgetrf
  - culaDeviceGetrf (C++ style, type overloaded)

Description
GETRF computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.
The factorization has the form
\[ A = P * L * U \]
where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U is upper triangular (upper trapezoidal if m < n).

Parameters
• m
  - Type: int
  - Direction: Input
  The number of rows of the matrix A. M >= 0.

• n
  - Type: int
  - Direction: Input
  The number of columns of the matrix A. N >= 0.

• a
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: (LDA, N)
  On entry, the M-by-N matrix to be factored.
  On exit, the factors L and U from the factorization \( A = P*L*U \); the unit diagonal elements of L are not stored.

• lda
  - Type: int
The leading dimension of the array A. LDA >= max(1,M).

• ipiv
  – Type: INTEGER array
  – Direction: Output
  – Dimension: (min(M, N))

The pivot indices; for 1 <= i <= min(M, N), row i of the matrix was interchanged with row IPIV(i).

Further Details
See Pivot Arrays section.

### 3.15 GETRI

CULA Routines
The GETRI functionality is implemented by the following CULA routines:

• Host Memory
  – culaSgetri
  – culaDgetri
  – culaCgetri
  – culaZgetri
  – culaGetri (C++ style, type overloaded)

• Device Memory
  – culaDeviceSgetri
  – culaDeviceDgetri
  – culaDeviceCgetri
  – culaDeviceZgetri
  – culaDeviceGetri (C++ style, type overloaded)

Description
GETRI computes the inverse of a matrix using the LU factorization computed by GETRF.

This method inverts U and then computes inv(A) by solving the system inv(A)*L = inv(U) for inv(A).

If solution to a system of linear equations, please favor the routines GESV, GETRF/GETRS, or GELS instead as they will provide more accurate answers in this case.

Parameters

• n
  – Type: int
  – Direction: Input

  The order of the matrix A. N >= 0.

• a
- Type: S/D/C/Z Pointer
- Direction: Input/Output
- Dimension: (LDA, N)

On entry, the factors L and U from the factorization A = P*L*U as computed by GETRF.

On exit, if INFO = 0, the inverse of the original matrix A.

• lda
  - Type: int
  - Direction: Input
  
The leading dimension of the array A. LDA >= max(1,N).

• ipiv
  - Type: INTEGER array
  - Direction: Input
  - Dimension: (N)
  
The pivot indices from GETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

Further Details
See Pivot Arrays section.

Differences from LAPACK
See No Workspace Parameters section.

3.16 GETRS

CULA Routines

The GETRS functionality is implemented by the following CULA routines:

• Host Memory
  - culaSgetrs
  - culaDgetrs
  - culaCgetrs
  - culaZgetrs
  - culaGetrs (C++ style, type overloaded)

• Device Memory
  - culaDeviceSgetrs
  - culaDeviceDgetrs
  - culaDeviceCgetrs
  - culaDeviceZgetrs
  - culaDeviceGetrs (C++ style, type overloaded)
Description
GETRS solves a system of linear equations

\[ A \cdot X = B \text{ or } A' \cdot X = B \]

with a general N-by-N matrix A using the LU factorization computed by GETRF.

Parameters
- **trans**
  - Type: char
  - Direction: Input

- **n**
  - Type: int
  - Direction: Input
  The order of the matrix A. N >= 0.

- **nrhs**
  - Type: int
  - Direction: Input
  The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

- **a**
  - Type: S/D/C/Z Pointer
  - Direction: Input
  - Dimension: (LDA, N)
  The factors L and U from the factorization A = P*L*U as computed by GETRF.

- **lda**
  - Type: int
  - Direction: Input
  The leading dimension of the array A. LDA >= max(1,N).

- **ipiv**
  - Type: INTEGER array
  - Direction: Input
  - Dimension: (N)
  The pivot indices from GETRF; for 1<=i<=N, row i of the matrix was interchanged with row IPIV(i).

- **b**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
– Dimension: (LDB, NRHS)

On entry, the right hand side matrix B.
On exit, the solution matrix X.

• ldb
  – Type: int
  – Direction: Input

  The leading dimension of the array B. LDB >= max(1,N).

Further Details

See Pivot Arrays section.

3.17 GGLSE

CULA Routines

The GGLSE functionality is implemented by the following CULA routines:

• Host Memory
  – culaSgglse
  – culaDgglse
  – culaCgglse
  – culaZgglse
  – culaGglse (C++ style, type overloaded)

• Device Memory
  – culaDeviceSgglse
  – culaDeviceDgglse
  – culaDeviceCgglse
  – culaDeviceZgglse
  – culaDeviceGglse (C++ style, type overloaded)

Description

GGLSE solves the linear equality-constrained least squares (LSE) problem:

\[
\text{minimize } \| c - A^*x \|_2 \text{ subject to } B^*x = d
\]

where A is an M-by-N matrix, B is a P-by-N matrix, c is a given M-vector, and d is a given P-vector. It is assumed that P <= N <= M+P, and

\[
\text{rank}(B) = P \text{ and } \text{rank}( (A) ) = N. \ ( (B) )
\]

These conditions ensure that the LSE problem has a unique solution, which is obtained using a generalized RQ factorization of the matrices (B, A) given by

\[
B = (0 R)^*Q, \ A = Z^*T^*Q.
\]

Parameters

• m
The number of rows of the matrix A. M >= 0.

- **n**
  - Type: int
  - Direction: Input
  - The number of columns of the matrices A and B. N >= 0.

- **p**
  - Type: int
  - Direction: Input
  - The number of rows of the matrix B. 0 <= P <= N <= M+P.

- **a**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: (LDA, N)
  - On entry, the M-by-N matrix A.
  - On exit, the elements on and above the diagonal of the array contain the min(M,N)-by-N upper trapezoidal matrix T.

- **lda**
  - Type: int
  - Direction: Input
  - The leading dimension of the array A. LDA >= max(1,M).

- **b**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: (LDB, N)
  - On entry, the P-by-N matrix B.
  - On exit, the upper triangle of the subarray B(1:P,N-P+1:N) contains the P-by-P upper triangular matrix R.

- **ldb**
  - Type: int
  - Direction: Input
  - The leading dimension of the array B. LDB >= max(1,P).

- **c**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
On entry, \( C \) contains the right hand side vector for the least squares part of the LSE problem. On exit, the residual sum of squares for the solution is given by the sum of squares of elements \( N-P+1 \) to \( M \) of vector \( C \).

- \( d \)
  - Type: \( S/D/C/Z \) Pointer
  - Direction: Input/Output
  - Dimension: \( (P) \)
  On entry, \( D \) contains the right hand side vector for the constrained equation. On exit, \( D \) is destroyed.

- \( x \)
  - Type: \( S/D/C/Z \) Pointer
  - Direction: Output
  - Dimension: \( (N) \)
  On exit, \( X \) is the solution of the LSE problem.

**Differences from LAPACK**
See *No Workspace Parameters* section.

### 3.18 GGRQF

**CULA Routines**
The GGRQF functionality is implemented by the following CULA routines:

- **Host Memory**
  - `cudaSggrqf`
  - `cudaDggrqf`
  - `cudaCggrqf`
  - `cudaZggrqf`
  - `cudaGgrqf` (C++ style, type overloaded)

- **Device Memory**
  - `cudaDeviceSggrqf`
  - `cudaDeviceDggrqf`
  - `cudaDeviceCggrqf`
  - `cudaDeviceZggrqf`
  - `cudaDeviceGgrqf` (C++ style, type overloaded)

**Description**
GGRQF computes a generalized RQ factorization of an \( M \)-by-\( N \) matrix \( A \) and a \( P \)-by-\( N \) matrix \( B \):

\[
A = R^*Q, \quad B = Z^*T^*Q.
\]
where Q is an N-by-N orthogonal/unitary matrix, Z is a P-by-P orthogonal/unitary matrix, and R and T assume one of the forms:

if M <= N, R = ( 0 R12 ) M, or if M > N, R = ( R11 ) M-N, N-M M ( R21 ) N N

where R12 or R21 is upper triangular, and

if P >= N, T = ( T11 ) N, or if P < N, T = ( T11 T12 ) P, ( 0 ) P-N P N-P N

where T11 is upper triangular.

In particular, if B is square and nonsingular, the GRQ factorization of A and B implicitly gives the RQ factorization of A*inv(B):

\[ A*\text{inv}(B) = (R*\text{inv}(T))*Z' \]

where inv(B) denotes the inverse of the matrix B, and Z' denotes the transpose of the matrix Z.

Parameters

- **m**
  - Type: int
  - Direction: Input
  - The number of rows of the matrix A. M >= 0.

- **p**
  - Type: int
  - Direction: Input
  - The number of rows of the matrix B. P >= 0.

- **n**
  - Type: int
  - Direction: Input
  - The number of columns of the matrices A and B. N >= 0.

- **a**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: (LDA, N)
  - On entry, the M-by-N matrix A.
  - On exit, if M <= N, the upper triangle of the subarray A(1:M,N-M+1:N) contains the M-by-M upper triangular matrix R; if M > N, the elements on and above the (M-N)-th subdiagonal contain the M-by-N upper trapezoidal matrix R; the remaining elements, with the array TAUA, represent the orthogonal/unitary matrix Q as a product of elementary reflectors (see Further Details).

- **lda**
  - Type: int
  - Direction: Input
  - The leading dimension of the array A. LDA >= max(1,M).

- **taua**
  - Type: S/D/C/Z Pointer
The scalar factors of the elementary reflectors which represent the orthogonal/unitary matrix $Q$ (see Further Details).

- **b**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: $(LDB, N)$

On entry, the $P$-by-$N$ matrix $B$.

On exit, the elements on and above the diagonal of the array contain the min$(P,N)$-by-$N$ upper trapezoidal matrix $T$ ($T$ is upper triangular if $P \geq N$); the elements below the diagonal, with the array $TAUB$, represent the orthogonal/unitary matrix $Z$ as a product of elementary reflectors (see Further Details).

- **ldb**
  - Type: int
  - Direction: Input
  - Dimension: $\text{min}(P,N)$

The leading dimension of the array $B$. $LDB \geq \text{max}(1,P)$.

- **taub**
  - Type: S/D/C/Z Pointer
  - Direction: Output
  - Dimension: $(\text{min}(P,N))$

The scalar factors of the elementary reflectors which represent the orthogonal/unitary matrix $Z$ (see Further Details).

Further Details

The matrix $Q$ is represented as a product of elementary reflectors

$$Q = H(1) H(2) \ldots H(k), \text{ where } k = \text{min}(m,n).$$

Each $H(i)$ has the form

$$H(i) = I - \text{taua} \cdot v \cdot v'$$

where $\text{taua}$ is a real scalar, and $v$ is a real vector with $v(n-k+i+1:n) = 0$ and $v(n-k+i) = 1$; $v(1:n-k+i-1)$ is stored on exit in $A(m-k+i,1:n-k+i-1)$, and $\text{taua}$ in $\text{TAUA}(i)$. To form $Q$ explicitly, use LAPACK subroutine ORGRQ. To use $Q$ to update another matrix, use LAPACK subroutine $ORMRQ/UNMRQ$.

The matrix $Z$ is represented as a product of elementary reflectors

$$Z = H(1) H(2) \ldots H(k), \text{ where } k = \text{min}(p,n).$$

Each $H(i)$ has the form

$$H(i) = I - \text{taub} \cdot v \cdot v'$$

where $\text{taub}$ is a real scalar, and $v$ is a real vector with $v(1:i-1) = 0$ and $v(i) = 1$; $v(i+1:p)$ is stored on exit in $B(i+1:p,i)$, and $\text{taub}$ in $\text{TAUB}(i)$. To form $Z$ explicitly, use LAPACK subroutine $ORGQR/UNGQR$. To use $Z$ to update another matrix, use LAPACK subroutine $ORMQR/UNMQR$.

Differences from LAPACK
3.19 ORGBR/UNGBR

CULA Routines

The ORGBR/UNGBR functionality is implemented by the following CULA routines:

- Host Memory
  - culaSorgbr
  - culaDorgbr
  - culaCungbr
  - culaZungbr
  - culaOrgbr (C++ style, type overloaded)
  - culaUngbr (C++ style, type overloaded)

- Device Memory
  - culaDeviceSorgbr
  - culaDeviceDorgbr
  - culaDeviceCungbr
  - culaDeviceZungbr
  - culaDeviceOrgbr (C++ style, type overloaded)
  - culaDeviceUngbr (C++ style, type overloaded)

Description

ORGBR/UNGBR generates one of the real/complex orthogonal/unitary matrices Q or P^T determined by GEBRD when reducing a real matrix A to bidiagonal form: A = Q * B * P^T. Q and P^T are defined as products of elementary reflectors H(i) or G(i) respectively.

If VECT = ‘Q’, A is assumed to have been an M-by-K matrix, and Q is of order M: if m >= k, Q = H(1) H(2) . . . H(k) and ORGBR returns the first n columns of Q, where m >= n >= k; if m < k, Q = H(1) H(2) . . . H(m-1) and ORGBR returns Q as an M-by-M matrix.

If VECT = ‘P’, A is assumed to have been a K-by-N matrix, and P^T is of order N: if k < n, P^T = G(k) . . . G(2) G(1) and ORGBR returns the first m rows of P^T, where n >= m >= k; if k >= n, P^T = G(n-1) . . . G(2) G(1) and ORGBR returns P^T as an N-by-N matrix.

Parameters

- vect
  - Type: char
  - Direction: Input

  Specifies whether the matrix Q or the matrix P^T is required, as defined in the transformation applied by GEBRD: = ‘Q’: generate Q; = ‘P’: generate P^T.

- m
  - Type: int
The number of rows of the matrix Q or $P^T$ to be returned. $M \geq 0$.

- **n**
  - **Type:** int
  - **Direction:** Input

  The number of columns of the matrix Q or $P^T$ to be returned. $N \geq 0$. If VECT = ‘Q’, $M \geq N \geq \min(M,K)$; if VECT = ‘P’, $N \geq M \geq \min(N,K)$.

- **k**
  - **Type:** int
  - **Direction:** Input

  If VECT = ‘Q’, the number of columns in the original M-by-K matrix reduced by $GEBRD$. If VECT = ‘P’, the number of rows in the original K-by-N matrix reduced by $GEBRD$. $K \geq 0$.

- **a**
  - **Type:** S/D/C/Z Pointer
  - **Direction:** Input/Output
  - **Dimension:** $(LDA,N)$

  On entry, the vectors which define the elementary reflectors, as returned by $GEBRD$.

  On exit, the M-by-N matrix Q or $P^T$.

- **lda**
  - **Type:** int
  - **Direction:** Input

  The leading dimension of the array A. $LDA \geq \max(1,M)$.

- **tau**
  - **Type:** S/D/C/Z Pointer, dimension $(\min(M,K))$ if VECT = ‘Q’ $(\min(N,K))$ if VECT = ‘P’

  $TAU(i)$ must contain the scalar factor of the elementary reflector $H(i)$ or $G(i)$, which determines Q or $P^T$, as returned by $GEBRD$ in its array argument TAUQ or TAUP.

**Differences from LAPACK**

See *No Workspace Parameters* section.

### 3.20 ORGHR/UNGHR

**CULA Routines**

The ORGHR/UNGHR functionality is implemented by the following CULA routines:

- **Host Memory**
  - `culaSorghr`
- culaDorghr
- culaCunghr
- culaZunghr
- culaOrghr (C++ style, type overloaded)
- culaUnghr (C++ style, type overloaded)

**Device Memory**
- culaDeviceSorghr
- culaDeviceDorghr
- culaDeviceCunghr
- culaDeviceZunghr
- culaDeviceOrghr (C++ style, type overloaded)
- culaDeviceUnghr (C++ style, type overloaded)

**Description**

ORGHR/UNGHR generate a real/complex orthogonal/unitary matrix Q which is defined as the product of 
IHI-ILO elementary reflectors of order N, as returned by GEHRD:

\[ Q = H(ilo) H(ilo+1) \ldots H(ihi-1). \]

**Parameters**

- **N**
  - Type: int
  - Direction: Input
  
  The order of the matrix Q. \( N \geq 0. \)

- **ILO**
  - Type: int
  - Direction: Input

- **IHI**
  - Type: int
  - Direction: Input

ILO and IHI must have the same values as in the previous call of GEHRD. Q is equal to the unit 
matrix except in the submatrix \( Q(ilo+1:ihi,ilo+1:ihi) \). \( 1 \leq ILO \leq IHI \leq N, \text{ if } N > 0; ILO=1 \text{ and } IHI=0, \text{ if } N=0. \)

- **A**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: \((LDA,N)\)

On entry, the vectors which define the elementary reflectors, as returned by GEHRD.

On exit, the N-by-N unitary matrix Q.

- **LDA**
- Type: int
- Direction: Input

The leading dimension of the array A. LDA >= max(1,N).

• TAU
  - Type: S/D/C/Z Pointer
  - Direction: Input
  - Dimension: (N-1)

  TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by GEHRD.

Differences from LAPACK

See No Workspace Parameters section.

3.21 ORGLQ/UNGLQ

CULA Routines

The ORGLQ/UNGLQ functionality is implemented by the following CULA routines:

- Host Memory
  - culaSorglq
  - culaDorglq
  - culaCunglq
  - culaZunglq
  - culaOrglq (C++ style, type overloaded)
  - culaUnglq (C++ style, type overloaded)

- Device Memory
  - culaDeviceSorglq
  - culaDeviceDorglq
  - culaDeviceCunglq
  - culaDeviceZunglq
  - culaDeviceOrglq (C++ style, type overloaded)
  - culaDeviceUnglq (C++ style, type overloaded)

Description

ORGLQ/UNGLQ generates an M-by-N real matrix Q with orthonormal rows, which is defined as the first M rows of a product of K elementary reflectors of order N

\[ Q = H(k) \ldots H(2) H(1) \]

as returned by GELQF.

Parameters

• m
  - Type: int
- **Direction**: Input

The number of rows of the matrix $Q$. $M \geq 0$.

- **$n$**
  - **Type**: int
  - **Direction**: Input

The number of columns of the matrix $Q$. $N \geq M$.

- **$k$**
  - **Type**: int
  - **Direction**: Input

The number of elementary reflectors whose product defines the matrix $Q$. $M \geq K \geq 0$.

- **$a$**
  - **Type**: S/D/C/Z Pointer
  - **Direction**: Input/Output
  - **Dimension**: $(LDA, N)$

On entry, the $i$-th row must contain the vector which defines the elementary reflector $H(i)$, for $i = 1, 2, \ldots, k$, as returned by $GELQF$ in the first $k$ rows of its array argument $A$.

On exit, the $M$-by-$N$ matrix $Q$.

- **$lda$**
  - **Type**: int
  - **Direction**: Input

The first dimension of the array $A$. $LDA \geq \max(1, M)$.

- **$tau$**
  - **Type**: S/D/C/Z Pointer
  - **Direction**: Input
  - **Dimension**: $(K)$

$TAU(i)$ must contain the scalar factor of the elementary reflector $H(i)$, as returned by $GELQF$.

**Differences from LAPACK**

See *No Workspace Parameters* section.

### 3.22 ORGQR/UNGQR

**CULA Routines**

The ORGQR/UNGQR functionality is implemented by the following CULA routines:

- **Host Memory**
  - `culaSorgqr`
  - `culaDorgqr`
  - `culaCungqr`
- culaZungqr
- culaOrgqr (C++ style, type overloaded)
- culaUngqr (C++ style, type overloaded)

• Device Memory
  - culaDeviceSorgqr
  - culaDeviceDorgqr
  - culaDeviceCungqr
  - culaDeviceZungqr
  - culaDeviceOrgqr (C++ style, type overloaded)
  - culaDeviceUngqr (C++ style, type overloaded)

Description

ORGQR/UNGQR generates an M-by-N real/complex matrix Q with orthonormal columns, which is defined as the first N columns of a product of K elementary reflectors of order M

\[ Q = H(1) H(2) \ldots H(k) \]

as returned by \textit{GEQRF}.

Parameters

• \textit{m}
  - Type: int
  - Direction: Input
  The number of rows of the matrix Q. \( M \geq 0 \).

• \textit{n}
  - Type: int
  - Direction: Input
  The number of columns of the matrix Q. \( M \geq N \geq 0 \).

• \textit{k}
  - Type: int
  - Direction: Input
  The number of elementary reflectors whose product defines the matrix Q. \( N \geq K \geq 0 \).

• \textit{a}
   - Type: S/D/C/Z Pointer
   - Direction: Input/Output
   - Dimension: (LDA, N)
  On entry, the i-th column must contain the vector which defines the elementary reflector H(i), for \( i = 1,2,\ldots,k \), as returned by \textit{GEQRF} in the first k columns of its array argument A.
  On exit, the M-by-N matrix Q.

• \textit{lda}
   - Type: int
The first dimension of the array A. LDA >= max(1,M).

- **tau**
  - Type: S/D/C/Z Pointer
  - Direction: Input
  - Dimension: (K)

TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by `GEQRF`.

**Differences from LAPACK**

See *No Workspace Parameters* section.

### 3.23 ORMLQ/UNMLQ

**CULA Routines**

The ORMLQ/UNMLQ functionality is implemented by the following CULA routines:

- **Host Memory**
  - `culaSormlq`
  - `culaDormlq`
  - `culaCunmlq`
  - `culaZunmlq`
  - `culaOrmlq` (C++ style, type overloaded)
  - `culaUnmlq` (C++ style, type overloaded)

- **Device Memory**
  - `culaDeviceSormlq`
  - `culaDeviceDormlq`
  - `culaDeviceCunmlq`
  - `culaDeviceZunmlq`
  - `culaDeviceOrmlq` (C++ style, type overloaded)
  - `culaDeviceUnmlq` (C++ style, type overloaded)

**Description**

ORMLQ/UNMLQ overwrite the general real/complex M-by-N matrix C with:

<table>
<thead>
<tr>
<th>TRANS</th>
<th>SIDE = ‘L’</th>
<th>SIDE = ‘R’</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘N’</td>
<td>Q * C</td>
<td>C * Q</td>
</tr>
<tr>
<td>‘T’</td>
<td>Q^T * C</td>
<td>C * Q^T</td>
</tr>
</tbody>
</table>

where Q is a real/complex orthogonal/unitary matrix defined as the product of k elementary reflectors

\[ Q = H(k) \ldots H(2) H(1) \]

as returned by `GELQF`. Q is of order M if SIDE = ‘L’ and of order N if SIDE = ‘R’.

**Parameters**
• side
  - Type: char
  - Direction: Input
  = ’L’: apply Q or QT from the Left; = ’R’: apply Q or QT from the Right.

• trans
  - Type: char
  - Direction: Input
  = ’N’: No transpose, apply Q; = ’T’: Transpose, apply QT.

• m
  - Type: int
  - Direction: Input
  The number of rows of the matrix C. M >= 0.

• n
  - Type: int
  - Direction: Input
  The number of columns of the matrix C. N >= 0.

• k
  - Type: int
  - Direction: Input
  The number of elementary reflectors whose product defines the matrix Q. If SIDE = ’L’, M >= K >= 0; if SIDE = ’R’, N >= K >= 0.

• a
  - Type: S/D/C/Z Pointer, dimension
  - Direction: Input
  (LDA,M) if SIDE = ’L’, (LDA,N) if SIDE = ’R’
  The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by GELQF in the first k rows of its array argument A. A is modified by the routine but restored on exit.

• lda
  - Type: int
  - Direction: Input
  The leading dimension of the array A. LDA >= max(1,K).

• tau
  - Type: S/D/C/Z Pointer
  - Direction: Input
  - Dimension: (K)
  TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by GELQF.
• **c**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: (LDC, N)
  On entry, the M-by-N matrix C.
  On exit, C is overwritten by Q * C or Q^T * C or C * Q^T or C * Q.

• **ldc**
  - Type: int
  - Direction: Input
  The leading dimension of the array C. LDC \(\geq\) max(1, M).

**Differences from LAPACK**

See *No Workspace Parameters* section.

### 3.24 ORMQR/UNMQR

**CULA Routines**

The ORMQR/UNMQR functionality is implemented by the following CULA routines:

- **Host Memory**
  - `cudaSormqr`
  - `cudaDormqr`
  - `cudaCormqr`
  - `cudaZormqr`
  - `cudaOrmqr` (C++ style, type overloaded)
  - `cudaUnmqr` (C++ style, type overloaded)

- **Device Memory**
  - `cudaDeviceSormqr`
  - `cudaDeviceDormqr`
  - `cudaDeviceCormqr`
  - `cudaDeviceZormqr`
  - `cudaDeviceOrmqr` (C++ style, type overloaded)
  - `cudaDeviceUnmqr` (C++ style, type overloaded)

**Description**

ORMQR/UNMQR overwrites the general real M-by-N matrix C with

<table>
<thead>
<tr>
<th></th>
<th>SIDE = ‘L’</th>
<th>SIDE = ‘R’</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANS = ‘N’</td>
<td>Q * C</td>
<td>C * Q</td>
</tr>
<tr>
<td>TRANS = ‘T’</td>
<td>Q^T * C</td>
<td>C * Q^T</td>
</tr>
</tbody>
</table>

where Q is a real/complex orthogonal/unitary matrix defined as the product of k elementary reflectors.
Q = H(1) H(2) . . . H(k)
as returned by \textit{GEQRF}. Q is of order M if SIDE = ‘L’ and of order N if SIDE = ‘R’.

Parameters

- **side**
  - Type: \textit{char}
  - Direction: \textit{Input}
  - ‘L’: apply Q or Q^T from the Left; ‘R’: apply Q or Q^T from the Right.

- **trans**
  - Type: \textit{char}
  - Direction: \textit{Input}
  - ‘N’: No transpose, apply Q; ‘T’: Transpose, apply Q^T.

- **m**
  - Type: \textit{int}
  - Direction: \textit{Input}
  - The number of rows of the matrix C. M >= 0.

- **n**
  - Type: \textit{int}
  - Direction: \textit{Input}
  - The number of columns of the matrix C. N >= 0.

- **k**
  - Type: \textit{int}
  - Direction: \textit{Input}
  - The number of elementary reflectors whose product defines the matrix Q. If SIDE = ‘L’, M >= K >= 0; if SIDE = ‘R’, N >= K >= 0.

- **a**
  - Type: \textit{S/D/C/Z Pointer}
  - Direction: \textit{Input}
  - Dimension: (LDA, K)
  - The i-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by \textit{GEQRF} in the first k columns of its array argument \textit{A}. \textit{A} is modified by the routine but restored on exit.

- **lda**
  - Type: \textit{int}
  - Direction: \textit{Input}
  - The leading dimension of the array \textit{A}. If SIDE = ‘L’, LDA >= max(1,M); if SIDE = ‘R’, LDA >= max(1,N).

- **tau**
- Type: S/D/C/Z Pointer
- Direction: Input
- Dimension: (K)

$\text{TAU}(i)$ must contain the scalar factor of the elementary reflector $H(i)$, as returned by $\text{GEQRF}$. 

- **c**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: (LDC, N)

On entry, the M-by-N matrix C.

On exit, C is overwritten by $Q \times C$ or $Q^T \times C$ or $C \times Q^T$ or $C \times Q$.

- **ldc**
  - Type: int
  - Direction: Input

The leading dimension of the array C. $\text{LDC} \geq \text{max}(1,M)$.

**Differences from LAPACK**

See No Workspace Parameters section.

### 3.25 ORMRQ/UNMRQ

**CULA Routines**

The ORMRQ/UNMRQ functionality is implemented by the following CULA routines:

- **Host Memory**
  - culaSormrq
  - culaDormrq
  - culaCormrq
  - culaZormrq
  - culaOrmrq (C++ style, type overloaded)
  - culaUnmrq (C++ style, type overloaded)

- **Device Memory**
  - culaDeviceSormrq
  - culaDeviceDormrq
  - culaDeviceCormrq
  - culaDeviceZormrq
  - culaDeviceOrmrq (C++ style, type overloaded)
  - culaDeviceUnmrq (C++ style, type overloaded)
 ORMQR/UNMRQ overwrites the general real M-by-N matrix C with

<table>
<thead>
<tr>
<th>TRANS = ‘N’</th>
<th>SIDE = ‘L’</th>
<th>SIDE = ‘R’</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANS = ‘T’</td>
<td>Q * C</td>
<td>C * Q</td>
</tr>
<tr>
<td></td>
<td>Q^T * C</td>
<td>C * Q^T</td>
</tr>
</tbody>
</table>

where Q is a real/complex orthogonal/unitary matrix defined as the product of k elementary reflectors

Q = H(1) H(2) \ldots H(k)

as returned by GERQF. Q is of order M if SIDE = ‘L’ and of order N if SIDE = ‘R’.

Parameters

• **side**
  - Type: char
  - Direction: Input
  = ‘L’: apply Q or Q^T from the Left; = ‘R’: apply Q or Q^T from the Right.

• **trans**
  - Type: char
  - Direction: Input
  = ‘N’: No transpose, apply Q; = ‘T’: Transpose, apply Q^T.

• **m**
  - Type: int
  - Direction: Input
  The number of rows of the matrix C. M >= 0.

• **n**
  - Type: int
  - Direction: Input
  The number of columns of the matrix C. N >= 0.

• **k**
  - Type: int
  - Direction: Input
  The number of elementary reflectors whose product defines the matrix Q. If SIDE = ‘L’, M >= K >= 0; if SIDE = ‘R’, N >= K >= 0.

• **a**
  - Type: S/D/C/Z Pointer, dimension
  - Direction: Input
  (LDA,M) if SIDE = ‘L’, (LDA,N) if SIDE = ‘R’

  The i-th row must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by GERQF in the last k rows of its array argument A. A is modified by the routine but restored on exit.
– Type: int
– Direction: Input
The leading dimension of the array A. LDA >= max(1,K).

• tau
  – Type: S/D/C/Z Pointer
  – Direction: Input
  – Dimension: (K)
  TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by GERQF.

• c
  – Type: S/D/C/Z Pointer
  – Direction: Input/Output
  – Dimension: (LDC,N)
  On entry, the M-by-N matrix C.
  On exit, C is overwritten by Q * C or Qᵀ * C or C * Qᵀ or C * Q.

• ldc
  – Type: int
  – Direction: Input
  The leading dimension of the array C. LDC >= max(1,M).

Differences from LAPACK
See No Workspace Parameters section.

3.26 POSV

CULA Routines
The POSV functionality is implemented by the following CULA routines:
  • Host Memory
    – culaSposv
    – culaDposv
    – culaCposv
    – culaZposv
    – culaPosv (C++ style, type overloaded)
  • Device Memory
    – culaDeviceSposv
    – culaDeviceDposv
    – culaDeviceCposv
    – culaDeviceZposv
POSV computes the solution to a real system of linear equations

\[ A \cdot X = B, \]

where \( A \) is an \( N \)-by-\( N \) symmetric positive definite matrix and \( X \) and \( B \) are \( N \)-by-\( NRHS \) matrices.

The Cholesky decomposition is used to factor \( A \) as

\[ A = U^T \cdot U, \text{ if UPLO} = 'U', \text{ or } A = L \cdot L^T, \text{ if UPLO} = 'L', \]

where \( U \) is an upper triangular matrix and \( L \) is a lower triangular matrix. The factored form of \( A \) is then used to solve the system of equations \( A \cdot X = B \).

**Parameters**

- **uplo**
  - Type: char
  - Direction: Input
  
  = ‘U’: Upper triangle of \( A \) is stored; = ‘L’: Lower triangle of \( A \) is stored.

- **n**
  - Type: int
  - Direction: Input

  The number of linear equations, i.e., the order of the matrix \( A \). \( N \geq 0 \).

- **nrhs**
  - Type: int
  - Direction: Input

  The number of right hand sides, i.e., the number of columns of the matrix \( B \). \( NRHS \geq 0 \).

- **a**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: \((LDA, N)\)

  On entry, the symmetric matrix \( A \). If UPLO = ‘U’, the leading \( N \)-by-\( N \) upper triangular part of \( A \) contains the upper triangular part of the matrix \( A \), and the strictly lower triangular part of \( A \) is not referenced. If UPLO = ‘L’, the leading \( N \)-by-\( N \) lower triangular part of \( A \) contains the lower triangular part of the matrix \( A \), and the strictly upper triangular part of \( A \) is not referenced.

  On exit, if INFO = 0, the factor \( U \) or \( L \) from the Cholesky factorization \( A = U^T \cdot U \) or \( A = L \cdot L^T \).

- **lda**
  - Type: int
  - Direction: Input

  The leading dimension of the array \( A \). \( LDA \geq \max(1,N) \).

- **b**
  - Type: S/D/C/Z Pointer
On entry, the N-by-NRHS right hand side matrix B.
On exit, if INFO = 0, the N-by-NRHS solution matrix X.

- **Idb**
  - Type: int
  - Direction: Input
  The leading dimension of the array B. LDB >= max(1,N).

## 3.27 POTRF

### CULA Routines

The POTRF functionality is implemented by the following CULA routines:

- **Host Memory**
  - culaSpotrf
  - culaDpotrf
  - culaCpotrf
  - culaZpotrf
  - culaPotrf (C++ style, type overloaded)

- **Device Memory**
  - culaDeviceSpotrf
  - culaDeviceDpotrf
  - culaDeviceCpotrf
  - culaDeviceZpotrf
  - culaDevicePotrf (C++ style, type overloaded)

### Description

POTRF computes the Cholesky factorization of a real symmetric positive definite matrix A.

The factorization has the form

\[ A = U^T \circ U, \text{ if } \text{UPLO} = \text{‘U’}, \text{ or } A = L \circ L^T, \text{ if } \text{UPLO} = \text{‘L’}, \]

where U is an upper triangular matrix and L is lower triangular.

### Parameters

- **uplo**
  - Type: char
  - Direction: Input
  
  = ‘U’: Upper triangle of A is stored; = ‘L’: Lower triangle of A is stored.
- **n**
The order of the matrix $A$. $N \geq 0$.

- **a**
  - Type: $S/D/C/Z$ Pointer
  - Direction: Input/Output
  - Dimension: $(LDA, N)$

  On entry, the symmetric matrix $A$. If $UPLO = 'U'$, the leading $N$-by-$N$ upper triangular part of $A$ contains the upper triangular part of the matrix $A$, and the strictly lower triangular part of $A$ is not referenced. If $UPLO = 'L'$, the leading $N$-by-$N$ lower triangular part of $A$ contains the lower triangular part of the matrix $A$, and the strictly upper triangular part of $A$ is not referenced.

  On exit, if $INFO = 0$, the factor $U$ or $L$ from the Cholesky factorization $A = U^T \ast U$ or $A = L \ast L^T$.

- **lda**
  - Type: int
  - Direction: Input

  The leading dimension of the array $A$. $LDA \geq \max(1, N)$.

### 3.28 POTRS

**CULA Routines**

The POTRS functionality is implemented by the following CULA routines:

- Host Memory
  - `cudaSpotrs`
  - `cudaDpotrs`
  - `cudaCpotrs`
  - `cudaZpotrs`
  - `cudaPotrs` (C++ style, type overloaded)

- Device Memory
  - `cudaDeviceSpotrs`
  - `cudaDeviceDpotrs`
  - `cudaDeviceCpotrs`
  - `cudaDeviceZpotrs`
  - `cudaDevicePotrs` (C++ style, type overloaded)

**Description**

POTRS solves a system of linear equations $A \times X = B$ with a symmetric positive definite matrix $A$ using the Cholesky factorization $A = U^T \ast U$ or $A = L \ast L^T$ computed by POTRF.

**Parameters**

- **uplo**
– Type: char
– Direction: Input
   = ‘U’: Upper triangle of A is stored; = ‘L’: Lower triangle of A is stored.

• n
– Type: int
– Direction: Input
The order of the matrix A. N >= 0.

• nrhs
– Type: int
– Direction: Input
The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

• a
– Type: S/D/C/Z Pointer
– Direction: Input
– Dimension: (LDA, N)
The triangular factor U or L from the Cholesky factorization A = U^T * U or A = L * L^T, as computed by POTRF.

• lda
– Type: int
– Direction: Input
The leading dimension of the array A. LDA >= max(1,N).

• b
– Type: S/D/C/Z Pointer
– Direction: Input/Output
– Dimension: (LDB, NRHS)
On entry, the right hand side matrix B.
On exit, the solution matrix X.

• ldb
– Type: int
– Direction: Input
The leading dimension of the array B. LDB >= max(1,N).

### 3.29 TRTRI

#### CULA Routines
The TRTRI functionality is implemented by the following CULA routines:

• Host Memory
- `culaStrtri`
- `culaDtrtri`
- `culaCtrtri`
- `culaZtrtri`
- `culaTrtri` (C++ style, type overloaded)

**Device Memory**
- `culaDeviceStrtri`
- `culaDeviceDtrtri`
- `culaDeviceCtrtri`
- `culaDeviceZtrtri`
- `culaDeviceTrtri` (C++ style, type overloaded)

**Description**

`TRTRI` computes the inverse of a real upper or lower triangular matrix `A`.

**Parameters**

- **uplo**
  - Type: char
  - Direction: Input
  - `U`: `A` is upper triangular; `L`: `A` is lower triangular.

- **diag**
  - Type: char
  - Direction: Input
  - `N`: `A` is non-unit triangular; `U`: `A` is unit triangular.

- **n**
  - Type: int
  - Direction: Input
  - The order of the matrix `A`. `N` >= 0.

- **a**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: `(LDA, N)`
  - On entry, the triangular matrix `A`. If `UPLO` = ‘U’, the leading `N`-by-`N` upper triangular part of the array `A` contains the upper triangular matrix, and the strictly lower triangular part of `A` is not referenced. If `UPLO` = ‘L’, the leading `N`-by-`N` lower triangular part of the array `A` contains the lower triangular matrix, and the strictly upper triangular part of `A` is not referenced. If `DIAG` = ‘U’, the diagonal elements of `A` are also not referenced and are assumed to be 1.
  - On exit, the (triangular) inverse of the original matrix, in the same storage format.

- **lda**
The leading dimension of the array A. LDA >= max(1,N).

3.30 TRTRS

CULA Routines
The TRTRS functionality is implemented by the following CULA routines:

- Host Memory
  - culaStrtrs
  - culaDtrtrs
  - culaCtrtrs
  - culaZtrtrs
  - culaTrtrs (C++ style, type overloaded)

- Device Memory
  - culaDeviceStrtrs
  - culaDeviceDtrtrs
  - culaDeviceCtrtrs
  - culaDeviceZtrtrs
  - culaDeviceTrtrs (C++ style, type overloaded)

Description
TRTRS solves a triangular system of the form

\[ A \times X = B \] or \[ A^T \times X = B, \]

where A is a triangular matrix of order N, and B is an N-by-NRHS matrix. A check is made to verify that A is nonsingular.

Parameters

- **uplo**
  - Type: char
  - Direction: Input
    - ‘U’: A is upper triangular; ‘L’: A is lower triangular.

- **trans**
  - Type: char
  - Direction: Input
    Specifies the form of the system of equations: ‘N’: \( A \times X = B \) (No transpose) = ‘T’: \( A^T \times X = B \) (Transpose) = ‘C’: \( A^H \times X = B \) (Conjugate transpose = Transpose)

- **diag**
  - Type: char
- Direction: Input

= ‘N’: A is non-unit triangular; = ‘U’: A is unit triangular.

- **n**
  - Type: int
  - Direction: Input
  The order of the matrix A. N >= 0.

- **nrhs**
  - Type: int
  - Direction: Input
  The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

- **a**
  - Type: S/D/C/Z Pointer
  - Direction: Input
  - Dimension: (LDA,N)
  The triangular matrix A. If UPLO = ‘U’, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = ‘L’, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = ‘U’, the diagonal elements of A are also not referenced and are assumed to be 1.

- **lda**
  - Type: int
  - Direction: Input
  The leading dimension of the array A. LDA >= max(1,N).

- **b**
  - Type: S/D/C/Z Pointer
  - Direction: Input/Output
  - Dimension: (LDB,NRHS)
  On entry, the right hand side matrix B.
  On exit, if INFO = 0, the solution matrix X.

- **ldb**
  - Type: int
  - Direction: Input
  The leading dimension of the array B. LDB >= max(1,N).
DIFFERENCES BETWEEN CULA AND LAPACK

The usage of some CULA functions differ slightly from their LAPACK equivalents, though they perform the same operations. This section details some of the API-wide ways that CULA and LAPACK differ.

4.1 No Workspace Parameters

Many LAPACK functions require a workspace for internal operation. For those LAPACK functions that utilize a workspace, workspace sizes are queried by providing a -1 argument to what is typically an LWORK parameter. Upon inspecting this parameter, the LAPACK function will determine the workspace required for this particular problem size and will return the value in the WORK parameter. LAPACK (and other similar packages) then require the programmer to provide a pointer to memory of sufficient size, which often requires that the programmer allocate new memory.

CULA uses both main and GPU workspace memories, and as such, LAPACK’s workspace query is not appropriate, as the LAPACK interface allows for the specification of only one workspace. Instead of providing a more complicated interface that adds parameters for both main and GPU workspace memories, CULA requires neither. Instead, any workspaces that are required are allocated and tracked internally. This organization yields no significant performance loss, and furthermore reduces the number of function calls by removing the need for a workspace query.

**Note:** Any workspaces that have been allocated internally may be cleared by calling `culaFreeBuffers()`.
CHAPTER FIVE

COMMON ERRORS

This section lists some of the common errors users make when using CULA and similar LAPACK packages.

5.1 Pivot Arrays

This section applies to functions in the LU Family (getrf, gesv, getrs, etc.).

CULA pivot arrays follow LAPACK conventions. These arrays are created for serial evaluation and describe a series of row interchanges. The array [2 3 3] states “swap the first row with the second, then the second row with the third, then the third row is unchanged.” For those working with Matlab, note that Matlab follows a different convention, in which the pivot array describes a set of parallel row interchanges. The Matlab array [2 3 1] is equivalent to the first example, and states “for the first row, obtain the second row; for the second row obtain the third; and for the third row obtain the first.”

5.2 Padding With Zeros

A common GPU usage pattern is to pad matrices to multiples of 8/16/32 elements in order to achieve performance. Routines such as GEMM (matrix-matrix multiply, as found in CUBLAS) are data-insensitive to these extra elements if they are zero. CULA routines function differently and in many cases will react poorly if the zeros are included in the computation space; for instance, solving a system in which the coefficient matrix has a full row or column of zeros will result in a culaDataError as the matrix is indeed singular. To avoid this problem, please be sure to set the LDx parameters to the padded size, but to set the remainder of the inputs that describe sizes (M, N, etc) to match the size of the valid data for the computation (that is: the size before padding.) This will avoid many data errors.