CULA Reference Manual
www.culatools.com

Release R14 (CUDA 4.1)

EM Photonics, Inc.
www.emphotonics.com

January 31, 2012

CONTENTS

CHAPTER

ONE

INTRODUCTION

This guide documents CULA's Programming Interface. CULA™ is an implementation of the Linear Algebra PACKage (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:

- *Conventions* - This section documents the conventions that CULA uses.
- *Framework Functions* - These section documents functions are used in initializing CULA, shutting it down, and querying information about errors.
- *LAPACK Routines* - This section documents the LAPACK functions that CULA provides.
- *BLAS Routines* - This section documents the BLAS 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 4.0 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.

CHAPTER

TWO

CONVENTIONS

2.1 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).

Symbol	Host Interface Type	Device Interface Type
S	culaFloat	culaDeviceFloat
D	culaDouble	culaDeviceDouble
C	culaFloatComplex	culaDeviceFloatComplex
Z	culaDoubleComplex	culaDeviceDoubleComplex

2.2 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 `culaDeviceMalloc` (available in CULA premium). All Device interface routines have the word *Device* as part of the function name; all other functions are Host interface.

There are some pointer arguments for which this not the case; these will often be output scalar arguments rather than matrices or vectors. These are denoted as "(type) Pointer, always host" etc.

2.3 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 x 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.

CHAPTER

THREE

FRAMEWORK FUNCTIONS

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

3.1 culaInitialize

Description

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

Returns

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

3.2 culaShutdown

Description

Shuts down CULA

3.3 culaGetLastStatus

Description

Returns the last status code returned from a CULA function

Returns

The last CULA status code

3.4 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

3.5 culaGetStatusAsString

Description

Returns the culaStatus name as a string

Parameters

- **e** - A culaStatus error code

Returns

A string that corresponds with the specified culaStatus enum

3.6 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

3.7 culaGetErrorInfoString

Description

Associates a culaStatus and culaInfo with a readable error string

Parameters

- **e** - A culaStatus error code
- **i** - An culaInfo error code
- **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 a successful error report or culaArgumentError on an invalid argument to this function

3.8 culaFreeBuffers

Description

Releases any memory buffers stored internally by CULA

3.9 culaGetVersion

Description

Reports the version number of CULA

Returns

An integer in the format XXXYY where XXX is the major version number and YY is the minor version number of CULA. On error, a 0 is returned.

3.10 culaGetCudaMinimumVersion

Description

Reports the CUDA_VERSION that the running version of CULA was compiled against, which indicates the minimum version of CUDA that is required to use this library

Returns

An integer in the format XXXYY where XXX is the major version number and YY is the minor version number of CUDA that this version of CULA was compiled against. On error, a 0 is returned.

3.11 culaGetCudaRuntimeVersion

Description

Reports the version of the CUDA runtime that the operating system linked against when the program was loaded

Returns

An integer in the format XXXYY where XXX is the major version number and YY is the minor version number of the CUDA runtime. On error, a 0 is returned.

3.12 culaGetCudaDriverVersion

Description

Reports the version of the CUDA driver installed on the system

Returns

An integer in the format XXXYY where XXX is the major version number and YY is the minor version number of the CUDA driver currently installed on the system. If no driver is installed, a 0 is returned.

3.13 culaGetCublasMinimumVersion

Description

Reports the CUBLAS_VERSION that the running version of CULA was compiled against, which indicates the minimum version of CUBLAS that is required to use this library

Returns

An integer in the format XXXYY where XXX is the major version number and YY is the minor version number of CUBLAS that this version of CULA was compiled against. On error, a 0 is returned.

3.14 culaGetCublasRuntimeVersion

Description

Reports the version of the CUBLAS runtime that operating system linked against when the program was loaded

Returns

An integer in the format XXXYY where XXX is the major version number and YY is the minor version number of the CUBLAS runtime. On error, a 0 is returned.

3.15 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

3.16 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

3.17 culaGetExecutingDevice

Description

Reports the id of the GPU device executing CULA

Parameters

- **dev** - Pointer to receive the GPU device number

Returns

culaNoError on sucess, culaArgumentError on invalid pointer

3.18 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 sucess, culaArgumentError on invalid buf pointer, invalid device id, or invalid bufsize

3.19 culaGetOptimalPitch

Description

Calculates a pitch that is optimal for CULA when using the device interface

Parameters

- **pitch** - The optimal pitch for the specified matrix in elements (where *pitch >= rows)
- **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

Returns

culaNoError on successful allocation, culaInsufficientMemory on failure

3.20 culaDeviceMalloc

Description

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

Parameters

- **mem** - Pointer to which a newly allocated buffer will be assigned
- **pitch** - The pitch of the allocation in elements (where *pitch >= rows)
- **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

Returns

culaNoError on successful allocation, culaInsufficientMemory on failure

3.21 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

CHAPTER

FOUR

LAPACK ROUTINES

This section documents the LAPACK 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.

4.1 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

$$B = Q * S * P^T$$

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.

See "Computing Small Singular Values of Bidiagonal Matrices With Guaranteed High Relative Accuracy," by J. Demmel and W. Kahan, LAPACK Working Note #3 (or SIAM J. Sci. Statist. Comput. vol. 11, no. 5, pp. 873-912, Sept 1990) and "Accurate singular values and differential qd algorithms," by B. Parlett and V. Fernando, Technical Report CPAM-554, Mathematics Department, University of California at Berkeley, July 1992 for a detailed description of the algorithm.

Parameters

- **uplo**
  - Type: `char`
  - Direction: `Input`

  = 'U': B is upper bidiagonal;

  = 'L': B is lower bidiagonal.
- **n**
  - Type: `int`
  - Direction: `Input`

  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. NCC > 0 is currently not supported and will return culaFeatureNotImplemented.
- **d**
  - Type: `S/D Pointer`
  - Direction: `Input/Output`
  - Dimension: `(N)`

  On entry, the n diagonal elements of the bidiagonal matrix B.

  On exit, if culaNoError is returned, 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 culaNoError is returned, E is destroyed; if culaDataError is returned, 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`
  - Direction: `Input`

  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 Q^T * 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.

Errors

On a culaDataError, where `culaGetErrorInfo()` = i:

if NCVT = NRU = NCC = 0, and

i = 1, a split was marked by a positive value in E

i = 2, current block of Z not diagonalized after 30*N iterations (in inner while loop)

i = 3, termination criterion of outer while loop not met program created more than N unreduced blocks)

otherwise the algorithm did not converge; D and E contain the elements of a bidiagonal matrix which is orthogonally similar to the input matrix B; i elements of E have not converged to zero.

For more information on data errors see the *Data Errors* section.

Differences from LAPACK

See *No Workspace Parameters* section.

4.2 GBTRF

CULA Routines

The GBTRF functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSgbtrf`
  - `culaDgbtrf`
  - `culaCgbtrf`
  - `culaZgbtrf`
  - `culaGbtrf` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSgbtrf`
  - `culaDeviceDgbtrf`
  - `culaDeviceCgbtrf`
  - `culaDeviceZgbtrf`
  - `culaDeviceGbtrf` (C++ style, type overloaded)

Description

GBTRF computes an LU factorization of an m-by-n band matrix A using partial pivoting with row interchanges.

This is the blocked version of the algorithm, calling Level 3 BLAS.

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.

- **kl**

  - Type: `int`
  - Direction: `Input`

  The number of subdiagonals within the band of A. KL >= 0.

- **ku**

  - Type: `int`
  - Direction: `Input`

  The number of superdiagonals within the band of A. KU >= 0.

- **ab**

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

  On entry, the matrix A in band storage, in rows KL+1 to 2*KL+KU+1; rows 1 to KL of the array need not be set. The j-th column of A is stored in the j-th column of the array AB as follows: AB(kl+ku+1+i-j,j) = A(i,j) for max(1,j-ku)<=i<=min(m,j+kl)

  On exit, details of the factorization: U is stored as an upper triangular band matrix with KL+KU superdiagonals in rows 1 to KL+KU+1, and the multipliers used during the factorization are stored in rows KL+KU+2 to 2*KL+KU+1. See below for further details.

- **ldab**

  - Type: `int`
  - Direction: `Input`

  The leading dimension of the array AB. LDAB >= 2*KL+KU+1.

- **ipiv**

  - Type: `int Pointer`
  - 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).

Performance Details

The performance of GBTRF scales with the bandwidth of the matrix, not the size.

Errors

On a culaDataError, where `culaGetErrorInfo()` = +i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

For more information on data errors see the *Data Errors* section.

Differences from LAPACK

See *No Workspace Parameters* section.

4.3 GEBRD

CULA Routines

The GEBRD functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSgebrd`
  - `culaDgebrd`
  - `culaCgebrd`
  - `culaZgebrd`
  - `culaGebrd` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSgebrd`
  - `culaDeviceDgebrd`
  - `culaDeviceCgebrd`
  - `culaDeviceZgebrd`
  - `culaDeviceGebrd` (C++ style, type overloaded)

Description

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 >= 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 >= 0.
- **n**
  - Type: `int`
  - Direction: `Input`

  The number of columns in the matrix A. N >= 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 >= 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 >= max(1,M).

- **d**
  - 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 >= 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.

- **tauq**
  - 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.

- **taup**
  - 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 >= n,

Q = H(1) H(2) . . . H(n) and P = G(1) G(2) . . . G(n-1)

Each H(i) and G(i) has the form:

H(i) = I - tauq * v * v' and G(i) = I - taup * u * u'

where tauq and taup 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+2:n) is stored on exit in A(i,i+2:n); tauq is stored in TAUQ(i) and taup in TAUP(i).

If m < n,

Q = H(1) H(2) . . . H(m-1) and P = G(1) G(2) . . . G(m)

Each H(i) and G(i) has the form:

H(i) = I - tauq * v * v' and G(i) = I - taup * u * u'

where tauq and taup 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); tauq is stored in TAUQ(i) and taup in TAUP(i).

The contents of A on exit are illustrated by the following examples:

```
m = 6 and n = 5 (m > n):          m = 5 and n = 6 (m < n):

  (  d   e   u1  u1  u1 )           (  d   u1  u1  u1  u1  u1 )
  (  v1  d   e   u2  u2 )           (  e   d   u2  u2  u2  u2 )
  (  v1  v2  d   e   u3 )           (  v1  e   d   u3  u3  u3 )
  (  v1  v2  v3  d   e  )           (  v1  v2  e   d   u4  u4 )
  (  v1  v2  v3  v4  d  )           (  v1  v2  v3  e   d   u5 )
  (  v1  v2  v3  v4  v5 )
```

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.

4.4 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 * v(j) = lambda(j) * v(j)

where lambda(j) is its eigenvalue.

The left eigenvector u(j) of A satisfies

$u(j)^H * A = lambda(j) * u(j)^H$

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 >= 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 >= 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)
  - Type: `C/Z Pointer`
  - Direction: `Output`
  - Dimension: `(N)`

  W contains the computed eigenvalues.

  Note: the **wi** and **wr** 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.

Errors

On a culaDataError, where `culaGetErrorInfo()` = i, the QR algorithm failed to compute all the eigenvalues, and no eigenvectors have been computed; elements i+1:N of WR and WI contain eigenvalues which have converged.

For more information on data errors see the *Data Errors* section.

Differences from LAPACK

See *No Workspace Parameters* section.

4.5 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 real/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`
  - Dimension: `(LDA,N)`

  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) . . . H(ihi-1).

Each H(i) has the form

H(i) = I - tau * v * v'

where tau is a real/complex scalar, and v is a real/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:

```
on entry,                       on exit,

( a   a   a   a   a   a   a )    (  a   a   h   h   h   h   a )
(     a   a   a   a   a   a )    (      a   h   h   h   h   a )
(     a   a   a   a   a   a )    (      h   h   h   h   h   h )
(     a   a   a   a   a   a )    (      v2  h   h   h   h   h )
(     a   a   a   a   a   a )    (      v2  v3  h   h   h   h )
(     a   a   a   a   a   a )    (      v2  v3  v4  h   h   h )
(                     a   )    (                      a   )
```

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

Differences from LAPACK

See *No Workspace Parameters* section.

4.6 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 * 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, 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 <= 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**

  - 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(k) . . . H(2) H(1), 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:n) is stored on exit in A(i,i+1:n), and tau in TAU(i).

Differences from LAPACK

See *No Workspace Parameters* section.

4.7 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 * X = B.
3. If TRANS = 'T' and m >= n: find the minimum norm solution of an undetermined system A^T * 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^T * 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^T.
- **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.

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

The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >=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, A is overwritten by details of its QR factorization as returned by *GEQRF*;

if M < N, A is overwritten by details of its LQ factorization as returned by *GELQF*.

- **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,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 culaNoError is returned, 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).

Errors

On a culaDataError, where `culaGetErrorInfo()` = i, the i-th diagonal element of the triangular factor of A is zero, so that A does not have full rank; the least squares solution could not be computed.

For more information on data errors see the *Data Errors* section.

Differences from LAPACK

See *No Workspace Parameters* section.

4.8 GEQLF

CULA Routines

The GEQLF functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSgeqlf`
  - `culaDgeqlf`
  - `culaCgeqlf`
  - `culaZgeqlf`
  - `culaGeqlf` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSgeqlf`
  - `culaDeviceDgeqlf`
  - `culaDeviceCgeqlf`
  - `culaDeviceZgeqlf`
  - `culaDeviceGeqlf` (C++ style, type overloaded)

Description

GEQLF computes a QL factorization of a real/complex M-by-N matrix A: A = Q * L.

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 lower triangle of the subarray A(m-n+1:m,1:n) contains the N-by-N lower triangular matrix L; if m <= n, the elements on and below the (n-m)-th superdiagonal contain the M-by-N lower trapezoidal matrix L; the remaining elements, with the array TAU, 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).

- **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) . . . H(2) H(1), where k = min(m,n).

Each H(i) has the form

H(i) = I - tau * v * v'

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

Differences from LAPACK

See *No Workspace Parameters* section.

4.9 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/complex 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/complex 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.

4.10 GEQRFP

CULA Routines

The GEQRFP functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSgeqrfp`
  - `culaDgeqrfp`
  - `culaCgeqrfp`
  - `culaZgeqrfp`
  - `culaGeqrfp` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSgeqrfp`
  - `culaDeviceDgeqrfp`
  - `culaDeviceCgeqrfp`
  - `culaDeviceZgeqrfp`
  - `culaDeviceGeqrfp` (C++ style, type overloaded)

Description

GEQRFP computes a QR factorization of a real/complex M-by-N matrix A: A = Q * R.

This function is similar to *GEQRF* with the addition that it takes special care to avoid errors arising from denormalized numbers.

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 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/complex scalar, and v is a real/complex 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.

4.11 GEQRS

CULA Routines

The GEQRS functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSgeqrs`
  - `culaDgeqrs`
  - `culaCgeqrs`
  - `culaZgeqrs`
  - `culaGeqrs` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSgeqrs`
  - `culaDeviceDgeqrs`

  - `culaDeviceCgeqrs`
  - `culaDeviceZgeqrs`
  - `culaDeviceGeqrs` (C++ style, type overloaded)

Description

Solve the least squares problem

    min || A*X - B ||

using the QR factorization

    A = Q*R

computed by *GEQRF*.

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. M >= N >= 0.
- **nrhs**
  - Type: `int`
  - Direction: `Input`

  The number of columns of B. NRHS >= 0.
- **a**
  - Type: `S/D/C/Z Pointer`,
  - Direction: `Input`
  - Dimension: `(LDA,N)`

  Details of the QR factorization of the original matrix A as returned by *GEQRF*.
- **lda**
  - Type: `int`
  - Direction: `Input`

  The leading dimension of the array A. LDA >= M.
- **tau**
  - Type: `S/D/C/Z Pointer`,
  - Direction: `Input`
  - Dimension: `(N)`

  Details of the orthogonal matrix Q.

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

  On entry, the M-by-NRHS right hand side matrix B.

  On exit, the N-by-NRHS solution matrix X.
- **ldb**
  - Type: `int`
  - Direction: `Input`

  The leading dimension of the array B. LDB >= M.

Differences from LAPACK

See *No Workspace Parameters* section.

4.12 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 >= 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(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.

4.13 GESV

The GESV functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSgesv`
  - `culaDgesv`

  - `culaCgesv`
  - `culaZgesv`
  - `culaGesv` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSgesv`
  - `culaDeviceDgesv`
  - `culaDeviceCgesv`
  - `culaDeviceZgesv`
  - `culaDeviceGesv` (C++ style, type overloaded)

Description

GESV computes the solution to a real system of linear equations

A * 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 * L * 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 * X = B.

Parameters

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

  The number of linear equations, i.e., 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/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 >= max(1,N).

- **ipiv**
  - Type: `int Pointer`
  - 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 culaNoError is returned, the N-by-NRHS solution matrix X.

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

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

Errors

On a culaDataError, where `culaGetErrorInfo()` = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

For more information on data errors see the *Data Errors* section.

Further Details

See *Pivot Arrays* section.

4.14 GESV (Iteratively Refined)

The GESV (Iteratively Refined) functionality is implemented by the following CULA routines:

- Host Memory
  - `culaDsgesv`
  - `culaZcgesv`
  - `culaGesv` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceDsgesv`
  - `culaDeviceZcgesv`
  - `culaDeviceGesv` (C++ style, type overloaded)

Description

GESV computes the solution to a real/complex system of linear equations

A * X = B,

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

GESV first attempts to factorize the matrix in single-precision and use this factorization within an iterative refinement procedure to produce a solution with double-precision normwise backward error quality (see below). If the approach fails the method switches to a double-precision factorization and solve.

The iterative refinement process is stopped if

ITER > ITERMAX

or for all the RHS we have:

RNRM < SQRT(N)*XNRM*ANRM*EPS*BWDMAX

where

- ITER is the number of the current iteration in the iterative refinement process
- RNRM is the infinity-norm of the residual
- XNRM is the infinity-norm of the solution
- ANRM is the infinity-operator-norm of the matrix A
- EPS is the machine epsilon for double precision

The value ITERMAX and BWDMAX are fixed to 30 and 1.0D+00 respectively.

Parameters

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

  The number of linear equations, i.e., 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: `D/Z Pointer`
  - Direction: `Input` or `Input/Output`
  - Dimension: `(LDA,N)`

  On entry, the N-by-N coefficient matrix A.

  On exit, if iterative refinement has been successfully used (culaNoError is returned and iter > 0, see description below), then A is unchanged, if double precision factorization has been used (culaNoError is returned and iter < 0, see description below), then the array A contains 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 >= max(1,N).
- **ipiv**

  - Type: `int Pointer`
  - Direction: `Output`
  - Dimension: `(N)`

  The pivot indices that define the permutation matrix P; row i of the matrix was interchanged with row IPIV(i). Corresponds either to the single precision factorization (if culaNoError is returned and iter > 0) or the double precision factorization (if culaNoError is returned and iter < 0).

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

  The N-by-NRHS right hand side matrix B.

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

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

- **x**
  - Type: `D/Z Pointer`
  - Direction: `Input/Output`
  - Dimension: `(LDX,NRHS)`

  If culaNoError is returned, the N-by-NRHS solution matrix X.

- **ldx**
  - Type: `int`
  - Direction: `Input`

  The leading dimension of the array X. LDX >= max(1,N).

- **iter**
  - Type: `int Pointer`
  - Direction: `Output`
  - Dimension: `(1)`

  > 0: iterative refinement has been sucessfully used. Returns the number of iterations

  < 0: iterative refinement has failed, double-precision factorization has been performed

Value	Operation
-1	the routine fell back to full precision for implementation- or machine-specific reasons
-2	narrowing the precision induced an overflow, the routine fell back to full precision
-3	failure of single precision GETRF
-31	stop the iterative refinement after the 30th iterations

Errors

On a culaDataError, where `culaGetErrorInfo() = i`, U(i,i) computed in double precision is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

For more information on data errors see the *Data Errors* section.

Differences from LAPACK

See *No Workspace Parameters* section. See *Pivot Arrays* section.

4.15 GESDD

CULA Routines

The GESDD functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSgesdd`
  - `culaDgesdd`
  - `culaCgesdd`
  - `culaZgesdd`
  - `culaGesdd` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSgesdd`
  - `culaDeviceDgesdd`
  - `culaDeviceCgesdd`
  - `culaDeviceZgesdd`
  - `culaDeviceGesdd` (C++ style, type overloaded)

Description

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

A = U * SIGMA * conjugate-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 unitary matrix, and V is an N-by-N 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 GESDD 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.

The divide and conquer algorithm makes very mild assumptions about floating point arithmetic. It will work on machines with a guard digit in add/subtract, or on those binary machines without guard digits which subtract like the Cray X-MP, Cray Y-MP, Cray C-90, or Cray-2. It could conceivably fail on hexadecimal or decimal machines without guard digits, but we know of none.

Parameters

- **JOBZ**
  - Type: `char`
  - Direction: `Input`

  Specifies options for computing all or part of the matrix U:

  = 'A': all M columns of U and all N rows of V**H are returned in the arrays U and VT;

  = 'S': the first min(M,N) columns of U and the first min(M,N) rows of V**H are returned in the arrays U and VT;

  = 'O': If M >= N, the first N columns of U are overwritten in the array A and all rows of V**H are returned in the array VT; otherwise, all columns of U are returned in the array U and the first M rows of V**H are overwritten in the array A;

  = 'N': no columns of U or rows of V**H are computed.

- **M**
  - Type: `int`
  - Direction: `Input`

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

- **N**
  - Type: `int`
  - Direction: `Input`

  The number of columns of the input 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 JOBZ = 'O', A is overwritten with the first N columns of U (the left singular vectors, stored columnwise)

  if JOBZ = 'O' and M >= N, A is overwritten with the first M rows of V**H (the right singular vectors, stored rowwise) otherwise.

  On exit, if JOBZ != 'O', the contents of A are destroyed.

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

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

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

  The singular values of A, sorted so that S(i) >= S(i+1).

- **U**
  - Type: `S/D/C/Z Pointer`,
  - Direction: `Output`
  - Dimension: `(LDU,UCOL)`

  UCOL = M if JOBZ = 'A' or JOBZ = 'O' and M < N; UCOL = min(M,N) if JOBZ = 'S'.

  If JOBZ = 'A' or JOBZ = 'O' and M < N, U contains the M-by-M unitary matrix U;

  if JOBZ = 'S', U contains the first min(M,N) columns of U (the left singular vectors, stored column-wise);

  if JOBZ = 'O' and M >= N, or JOBZ = 'N', U is not referenced.
- **LDU**
  - Type: `int`
  - Direction: `Input`

  The leading dimension of the array U. LDU >= 1; if JOBZ = 'S' or 'A' or JOBZ = 'O' and M < N, LDU >= M.
- **VT**
  - Type: `S/D/C/Z Pointer`,
  - Direction: `Output`
  - Dimension: `(LDVT,N)`

  If JOBZ = 'A' or JOBZ = 'O' and M >= N, VT contains the N-by-N unitary matrix V**H;

  if JOBZ = 'S', VT contains the first min(M,N) rows of V**H (the right singular vectors, stored rowwise);

  if JOBZ = 'O' and M < N, or JOBZ = 'N', VT is not referenced.
- **LDVT**
  - Type: `int`
  - Direction: `Input`

  The leading dimension of the array VT. LDVT >= 1; if JOBZ = 'A' or JOBZ = 'O' and M >= N, LDVT >= N; if JOBZ = 'S', LDVT >= min(M,N).

Differences from LAPACK

Whereas the traditional implementation of this function utilizes a divide-and-conquer method, this implementation simply forwards to the *GESVD* function.

See *No Workspace Parameters* section.

4.16 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 * SIGMA * 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 >= 0.

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

  The number of columns of the input 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 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 != 'O' and JOBVT != 'O', the contents of A are destroyed.

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

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

- **s**
  - Type: `S/D Pointer`
  - Direction: `Output`
  - Dimension: `(min(M,N))`

  The singular values of A, sorted so that S(i) >= 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 >= 1; if JOBU = 'S' or 'A', LDU >= 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 >= 1; if JOBVT = 'A', LDVT >= N; if JOBVT = 'S', LDVT >= min(M,N).

Errors

On a culaDataError, where `culaGetErrorInfo()` = i, if *GEBRD* did not converge, i specifies how many superdiagonals of an intermediate bidiagonal form B did not converge to zero.

For more information on data errors see the *Data Errors* section.

Differences from LAPACK

See *No Workspace Parameters* section.

4.17 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`
  - Direction: `Input`

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

- **ipiv**
  - Type: `int Pointer`
  - 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).

Errors

On a culaDataError, where `culaGetErrorInfo()` = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

For more information on data errors see the *Data Errors* section.

Further Details

See *Pivot Arrays* section.

4.18 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 culaNoError is returned, 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: `int Pointer`
  - Direction: `Input`
  - Dimension: `(N)`

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

Errors

On a culaDataError, where `culaGetErrorInfo()` = i, U(i,i) is exactly zero; the matrix is singular and its inverse could not be computed.

For more information on data errors see the *Data Errors* section.

Further Details

See *Pivot Arrays* section.

Differences from LAPACK

See *No Workspace Parameters* section.

4.19 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 * X = B or A' * X = B

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

Parameters

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

  Specifies the form of the system of equations: = 'N': A * X = B (No transpose)

  = 'T': A'* X = B (Transpose)

  = 'C': A'* X = B (Conjugate transpose = Transpose)
- **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: `int Pointer`
  - 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.

4.20 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:

minimize || c - A*x ||_2 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

rank(B) = P and 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**
  - 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 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`
  - Dimension: `(M)`

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.

Errors

On a culaDataError, where `culaGetErrorInfo()` = 1, the upper triangular factor R associated with B in the generalized RQ factorization of the pair (B, A) is singular; the least squares solution could not be computed.

On a culaDataError, where `culaGetErrorInfo()` = 2, the (N-P) by (N-P) part of the upper trapezoidal factor T associated with A in the generalized RQ factorization of the pair (B, A) is singular; the least squares solution could not be computed.

For more information on data errors see the *Data Errors* section.

Differences from LAPACK

See *No Workspace Parameters* section.

4.21 GGRQF

CULA Routines

The GGRQF functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSggrqf`
  - `culaDggrqf`
  - `culaCggrqf`
  - `culaZggrqf`
  - `culaGgrqf` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSggrqf`
  - `culaDeviceDggrqf`
  - `culaDeviceCggrqf`
  - `culaDeviceZggrqf`
  - `culaDeviceGgrqf` (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:

$$A = R * Q$$
$$B = Z * T * Q$$

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)

$$A * inv(B) = (R * inv(T)) * Z'$$

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`
  - Direction: `Output`
  - Dimension: `(min(M,N))`

  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 >= 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`

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

- **taub**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Output`
  - Dimension: `(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) . . . H(k), where k = min(m,n).

Each H(i) has the form

H(i) = I - taua * v * v'

where 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 taua in 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) . . . H(k), where k = min(p,n).

Each H(i) has the form

H(i) = I - taub * v * v'

where 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 taub in 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

See *No Workspace Parameters* section.

4.22 LACPY

CULA Routines

The LACPY functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSlacpy`
  - `culaDlacpy`
  - `culaClacpy`
  - `culaZlacpy`
  - `culaLacpy` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSlacpy`
  - `culaDeviceDlacpy`
  - `culaDeviceClacpy`
  - `culaDeviceZlacpy`
  - `culaDeviceLacpy` (C++ style, type overloaded)

Description

LACPY copies all or part of a two-dimensional matrix A to another matrix B.

Parameters

- **UPLO**
  - Type: `char`
  - Direction: `Input`

  Specifies the part of the matrix A to be copied to B.

  = 'U': Upper triangular part

  = 'L': Lower triangular part

  Otherwise: All of the matrix A
- **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`
  - Dimension: `(LDA,N)`

  The m by n matrix A. If UPLO = 'U', only the upper trapezium is accessed; if UPLO = 'L', only the lower trapezium is accessed.

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

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

- **B**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Output`
  - Dimension: `(LDB,N)`

  On exit, B = A in the locations specified by UPLO.

- **LDB**
  - Type: `int`
  - Direction: `Input`

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

4.23 LAG2

CULA Routines

The LAG2 functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSlag2d`
  - `culaDlag2s`
  - `culaClag2z`
  - `culaZlag2c`
  - `culaLag2` (C++ style, type overloaded)

- Device Memory
  - `culaDeviceSlag2d`
  - `culaDeviceDlag2s`
  - `culaDeviceClag2z`
  - `culaDeviceZlag2c`
  - `culaDeviceLag2` (C++ style, type overloaded)

Description

LAG2 converts a matrix to the opposite precision; S->D, D->S, C->Z, or Z->C.

Note that while it is possible to overflow while converting from double to single, it is not possible to overflow when converting from single to double.

Parameters

- **M**
  - Type: `int`
  - Direction: `Input`

  The number of lines of the matrix A. M >= 0.
- **N**
  - Type: `int`
  - Direction: `Input`

  The number of columns of the matrix A. N >= 0.
- **SA**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(LDSA,N)`

  On entry, the M-by-N coefficient matrix SA.
- **LDSA**
  - Type: `int`
  - Direction: `Input`

  The leading dimension of the array SA. LDSA >= max(1,M).
- **A**
  - Type: `S/D/C/Z Pointer` (of opposite precision to SA)
  - Direction: `Output`
  - Dimension: `(LDA,N)`

  On exit, the M-by-N coefficient matrix A.
- **LDA**
  - Direction: `Input`

  - Type: `int`

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

4.24 LAR2V

CULA Routines

The LAR2V functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSlar2v`
  - `culaDlar2v`
  - `culaClar2v`
  - `culaZlar2v`
  - `culaLar2v` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSlar2v`
  - `culaDeviceDlar2v`
  - `culaDeviceClar2v`
  - `culaDeviceZlar2v`
  - `culaDeviceLar2v` (C++ style, type overloaded)

Description

LAR2V applies a vector of plane rotations with real cosines from both sides to a sequence of 2-by-2 symmetric/hermitian matrices, defined by the elements of the vectors x, y and z.

Parameters

- **N**
  - Type: `int`
  - Direction: `Input`

  The number of plane rotations to be applied.
- **X**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input/Output`
  - Dimension: `(1+(N-1)*INCX1)`

  The vector x; the elements of x are assumed to be real.
- **Y**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input/Output`
  - Dimension: `(1+(N-1)*INCX1)`

  The vector y; the elements of y are assumed to be real.

- **Z**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input/Output`
  - Dimension: `(1+(N-1)*INCX1)`

  The vector z.
- **INCX**
  - Type: `int`
  - Direction: `Input`

  The increment between elements of X, Y and Z. INCX > 0.
- **C**
  - Type: `S/D Pointer`
  - Direction: `Input`
  - Dimension: `(1+(N-1)*INCC)`

  The cosines of the plane rotations.
- **S**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(1+(N-1)*INCC)`

  The sines of the plane rotations.
- **INCC**
  - Type: `int`
  - Direction: `Input`

  The increment between elements of C and S. INCC > 0.

4.25 LARFB

CULA Routines

The LARFB functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSlarfb`
  - `culaDlarfb`
  - `culaClarfb`
  - `culaZlarfb`
  - `culaLarfb` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSlarfb`

- `culaDeviceDlarfb`
- `culaDeviceClarfb`
- `culaDeviceZlarfb`
- `culaDeviceLarfb` (C++ style, type overloaded)

Description

LARFB applies a block reflector H or its transpose H' to an M-by-N matrix C, from either the left or the right.

Parameters

- **SIDE**
  - Type: `char`
  - Direction: `Input`

  = 'L': apply H or H^T from the Left

  = 'R': apply H or H^T from the Right
- **TRANS**
  - Type: `char`
  - Direction: `Input`

  = 'N': apply H (No transpose)

  = 'C': apply H^T (Conjugate transpose)
- **DIRECT**
  - Type: `char`
  - Direction: `Input`

  Indicates how H is formed from a product of elementary reflectors

  = 'F': H = H(1) H(2) . . . H(k) (Forward)

  = 'B': H = H(k) . . . H(2) H(1) (Backward)
- **STOREV**
  - Type: `char`
  - Direction: `Input`

  Indicates how the vectors which define the elementary reflectors are stored:

  = 'C': Columnwise

  = 'R': Rowwise
- **M**
  - Type: `int`
  - Direction: `Input`

  The number of rows of the matrix C.
- **N**
  - Type: `int`

  - Direction: `Input`

  The number of columns of the matrix C.

- **K**

  - Type: `int`
  - Direction: `Input`

  The order of the matrix T (= the number of elementary reflectors whose product defines the block reflector).

- **V**

  - Type: `S/D/C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(LDV,K)` if STOREV = 'C'
  - Dimension: `(LDV,M)` if STOREV = 'R' and SIDE = 'L'
  - Dimension: `(LDV,N)` if STOREV = 'R' and SIDE = 'R'

- **LDV**

  - Type: `int`
  - Direction: `Input`

  The leading dimension of the array V.

  If STOREV = 'C' and SIDE = 'L', LDV >= max(1,M);

  if STOREV = 'C' and SIDE = 'R', LDV >= max(1,N);

  if STOREV = 'R', LDV >= K.

- **T**

  - Type: `S/D/C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(LDT,K)`

  The triangular K-by-K matrix T in the representation of the block reflector.

- **LDT**

  - Type: `int`
  - Direction: `Input`

  The leading dimension of the array T. LDT >= K.

- **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 H*C or H^T * C or C * H or C * H^T.

- **LDC**

- Type: `int`
- Direction: `Input`

The leading dimension of the array C. LDC >= max(1,M).

Differences from LAPACK

See *No Workspace Parameters* section.

4.26 LARFG

CULA Routines

The LARFG functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSlarfg`
  - `culaDlarfg`
  - `culaClarfg`
  - `culaZlarfg`
  - `culaLarfg` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSlarfg`
  - `culaDeviceDlarfg`
  - `culaDeviceClarfg`
  - `culaDeviceZlarfg`
  - `culaDeviceLarfg` (C++ style, type overloaded)

Description

LARFG generates a elementary reflector H of order n, such that

$$H' * \begin{pmatrix} alpha \\ x \end{pmatrix} = \begin{pmatrix} beta \\ 0 \end{pmatrix}$$
$$H' * H = I$$

where alpha and beta are scalars, with beta real, and x is an (n-1)-element vector. H is represented in the form

$$H = I - tau * \begin{pmatrix} 1 \\ v \end{pmatrix} * \begin{pmatrix} 1 & v' \end{pmatrix}$$

where tau is a scalar and v is a (n-1)-element vector. Note that H is not symmetric/hermitian.

If the elements of x are all zero and alpha is real, then tau = 0 and H is taken to be the unit matrix.

Otherwise $1 <= real(tau) <= 2$ and $abs(tau - 1) <= 1$.

Parameters

- **N**
  - Type: `int`

  - Direction: `Input`

  The order of the elementary reflector.

- **ALPHA**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input/Output`

  On entry, the value alpha.

  On exit, it is overwritten with the value beta.

- **X**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input/Output`
  - Dimension: `(1+(N-2)*abs(INCX))`

  On entry, the vector x.

  On exit, it is overwritten with the vector v.

- **INCX**
  - Type: `int`
  - Direction: `Input`

  The increment between elements of X. INCX > 0.

- **TAU**
  - Type: `S/D/C/Z`
  - Direction: `Output`

    The value tau.

4.27 LARGV

CULA Routines

The LARGV functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSlargv`
  - `culaDlargv`
  - `culaClargv`
  - `culaZlargv`
  - `culaLargv` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSlargv`
  - `culaDeviceDlargv`
  - `culaDeviceClargv`

- `culaDeviceZlargv`
- `culaDeviceLargv` (C++ style, type overloaded)

Description

LARGV generates a vector of plane rotations with real cosines, determined by elements of the vectors x and y. For i = 1,2,...,n

$$K(k) = \begin{pmatrix} c(i) & s(i) \\ -conjg(s(i)) & c(i) \end{pmatrix} \begin{pmatrix} x(i) & y(i) \end{pmatrix} = \begin{pmatrix} r(i) & 0 \end{pmatrix}$$

where

$$c(i)^2 + abs(s(i))^2 = 1$$

The following conventions are used (these are the same as in LARTG, but differ from the BLAS1 routine ROTG):

If y(i)=0, then c(i)=1 and s(i)=0.

If x(i)=0, then c(i)=0 and s(i) is chosen so that r(i) is real.

Parameters

- **N**
  - Type: `int`
  - Direction: `Input`

  The number of plane rotations to be generated.
- **X**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input/Output`
  - Dimension: `(1+(N-1)*INCX1)`

  On entry, the vector x.

  On exit, x(i) is overwritten by r(i), for i = 1,...,n.
- **INCX**
  - Type: `int`
  - Direction: `Input`

  The increment between elements of X. INCX > 0.
- **Y**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input/Output`
  - Dimension: `(1+(N-1)*INCY)`

  On entry, the vector y.

  On exit, the sines of the plane rotations.
- **INCY**
  - Type: `int`

  - Direction: `Input`

  The increment between elements of Y. INCY > 0.

- **C**
  - Type: `S/D Pointer`
  - Direction: `Output`
  - Dimension: `(1+(N-1)*INCC)`

  The cosines of the plane rotations.

- **INCC**
  - Type: `int`
  - Direction: `Input`

  The increment between elements of C. INCC > 0.

4.28 LARTV

CULA Routines

The LARTV functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSlartv`
  - `culaDlartv`
  - `culaClartv`
  - `culaZlartv`
  - `culaLartv` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSlartv`
  - `culaDeviceDlartv`
  - `culaDeviceClartv`
  - `culaDeviceZlartv`
  - `culaDeviceLartv` (C++ style, type overloaded)

Description

LARTV applies a vector of plane rotations with real cosines to elements of the vectors x and y. For i = 1,2,...,n

$$\begin{pmatrix} x(i) \\ y(i) \end{pmatrix} := \begin{pmatrix} c(i) & s(i) \\ -conjg(s(i)) & c(i) \end{pmatrix} \begin{pmatrix} x(i) \\ y(i) \end{pmatrix}$$

Parameters

- **N**
  - Type: `int`
  - Direction: `Input`

  The number of plane rotations to be applied.

- **X**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input/Output`
  - Dimension: `(1+(N-1)*INCX1)`

  The vector x.
- **INCX**
  - Type: `int`
  - Direction: `Input`

  The increment between elements of X. INCX > 0.
- **Y**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input/Output`
  - Dimension: `(1+(N-1)*INCY)`

  The vector y.
- **INCY**
  - Type: `int`
  - Direction: `Input`

  The increment between elements of Y. INCY > 0.
- **C**
  - Type: `S/D Pointer`
  - Direction: `Input`
  - Dimension: `(1+(N-1)*INCC)`

  The cosines of the plane rotations.
- **S**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(1+(N-1)*INCC)`

  The sines of the plane rotations.
- **INCC**
  - Type: `int`
  - Direction: `Input`

  The increment between elements of C and S. INCC > 0.

4.29 LASCL

CULA Routines

The LASCL functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSlascl`
  - `culaDlascl`
  - `culaClascl`
  - `culaZlascl`
  - `culaLascl` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSlascl`
  - `culaDeviceDlascl`
  - `culaDeviceClascl`
  - `culaDeviceZlascl`
  - `culaDeviceLascl` (C++ style, type overloaded)

Description

LASCL multiplies the M by N matrix A by the real scalar CTO/CFROM. This is done without over/underflow as long as the final result CTO*A(I,J)/CFROM does not over/underflow. TYPE specifies that A may be full, upper triangular, lower triangular, upper Hessenberg, or banded.

Parameters

- **TYPE**
  - Type: `char`
  - Direction: `Input`

  TYPE indices the storage type of the input matrix.

  = 'G': A is a full matrix.

  = 'L': A is a lower triangular matrix.

  = 'U': A is an upper triangular matrix.

  = 'H': A is an upper Hessenberg matrix.

  = 'B': A is a symmetric band matrix with lower bandwidth KL and upper bandwidth KU and with the only the lower half stored.

  = 'Q': A is a symmetric band matrix with lower bandwidth KL and upper bandwidth KU and with the only the upper half stored.

  = 'Z': A is a band matrix with lower bandwidth KL and upper bandwidth KU. See CGBTRF for storage details.
- **KL**
  - Type: `int`

  - Direction: `Input`

  The lower bandwidth of A. Referenced only if TYPE = 'B', 'Q' or 'Z'.

- **KU**

  - Type: `int`
  - Direction: `Input`

  The upper bandwidth of A. Referenced only if TYPE = 'B', 'Q' or 'Z'.

- **CFROM**

  - Type: `S/D/C/Z`
  - Direction: `Input` REAL

- **CTO**

  - Type: `S/D/C/Z`
  - Direction: `Input`

  The matrix A is multiplied by CTO/CFROM. A(I,J) is computed without over/underflow if the final result CTO*A(I,J)/CFROM can be represented without over/underflow. CFROM must be nonzero.

- **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)`

  The matrix to be multiplied by CTO/CFROM. See TYPE for the storage type.

- **LDA**

  - Type: `int`
  - Direction: `Input`

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

4.30 LASET

CULA Routines

The LASET functionality is implemented by the following CULA routines:

- Host Memory

  - `culaSlaset`
  - `culaDlaset`
  - `culaClaset`
  - `culaZlaset`
  - `culaLaset` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSlaset`
  - `culaDeviceDlaset`
  - `culaDeviceClaset`
  - `culaDeviceZlaset`
  - `culaDeviceLaset` (C++ style, type overloaded)

Description

LASET initializes a 2-D array A to BETA on the diagonal and ALPHA on the offdiagonals.

Parameters

- **UPLO**
  - Type: `char`
  - Direction: `Input`

  Specifies the part of the matrix A to be set.

  = 'U': Upper triangular part is set. The lower triangle is unchanged.

  = 'L': Lower triangular part is set. The upper triangle is unchanged.

  Otherwise: All of the matrix A is set.
- **M**
  - Type: `int`
  - Direction: `Input`

  On entry, M specifies the number of rows of A.
- **N**
  - Type: `int`
  - Direction: `Input`

  On entry, N specifies the number of columns of A.
- **ALPHA**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input`

  All the offdiagonal array elements are set to ALPHA.
- **BETA**
  - Type: `S/D/C/Z Pointer`

  - Direction: `Input`

  All the diagonal array elements are set to BETA.

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

  On entry, the m by n matrix A.

  On exit, A(i,j) = ALPHA, 1 <= i <= m, 1 <= j <= n, i!=j; A(i,i) = BETA , 1 <= i <= min(m,n)

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

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

4.31 LASR

CULA Routines

The LASR functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSlasr`
  - `culaDlasr`
  - `culaClasr`
  - `culaZlasr`
  - `culaLasr` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSlasr`
  - `culaDeviceDlasr`
  - `culaDeviceClasr`
  - `culaDeviceZlasr`
  - `culaDeviceLasr` (C++ style, type overloaded)

Description

LASR applies a sequence of real plane rotations to a matrix A, from either the left or the right.

When SIDE = 'L', the transformation takes the form

$$A := P * A$$

and when SIDE = 'R', the transformation takes the form

$$A := A * P^T$$

where P is an orthogonal matrix consisting of a sequence of z plane rotations, with z = M when SIDE = 'L' and z = N when SIDE = 'R', and P**T is the transpose of P.

When DIRECT = 'F' (Forward sequence), then

$$P = P(z-1) * ... * P(2) * P(1)$$

and when DIRECT = 'B' (Backward sequence), then

$$P = P(1) * P(2) * ... * P(z-1)$$

where P(k) is a plane rotation matrix defined by the 2-by-2 rotation

$$R(k) = \begin{pmatrix} c(k) & s(k) \\ -s(k) & c(k) \end{pmatrix}$$

When PIVOT = 'V' (Variable pivot), the rotation is performed for the plane (k,k+1), i.e., P(k) has the form

$$P(k) = \begin{pmatrix} 1 & & & & & & & \\ & ... & & & & & & \\ & & 1 & & & & & \\ & & & c(k) & s(k) & & & \\ & & & -s(k) & c(k) & & & \\ & & & & & 1 & & \\ & & & & & & ... & \\ & & & & & & & 1 \end{pmatrix}$$

where R(k) appears as a rank-2 modification to the identity matrix in rows and columns k and k+1.

When PIVOT = 'T' (Top pivot), the rotation is performed for the plane (1,k+1), so P(k) has the form

$$P(k) = \begin{pmatrix} c(k) & & & & s(k) & & & \\ & 1 & & & & & & \\ & & ... & & & & & \\ & & & 1 & & & & \\ -s(k) & & & & c(k) & & & \\ & & & & & 1 & & \\ & & & & & & ... & \\ & & & & & & & 1 \end{pmatrix}$$

where R(k) appears in rows and columns 1 and k+1.

Similarly, when PIVOT = 'B' (Bottom pivot), the rotation is performed for the plane (k,z), giving P(k) the form

$$P(k) = \begin{pmatrix} 1 & & & & & & & \\ & ... & & & & & & \\ & & 1 & & & & & \\ & & & c(k) & & & & s(k) \\ & & & & 1 & & & \\ & & & & & ... & & \\ & & & & & & 1 & \\ & & & -s(k) & & & & c(k) \end{pmatrix}$$

where R(k) appears in rows and columns k and z. The rotations are performed without ever forming P(k) explicitly.

Parameters

- **SIDE**
  - Type: `char`

  - Direction: `Input`

Specifies whether the plane rotation matrix P is applied to A on the left or the right.

= 'L': Left, compute A := P*A

= 'R': Right, compute A:= A*P**T

- **PIVOT**
  - Type: `char`
  - Direction: `Input`

Specifies the plane for which P(k) is a plane rotation matrix.

= 'V': Variable pivot, the plane (k,k+1)

= 'T': Top pivot, the plane (1,k+1)

= 'B': Bottom pivot, the plane (k,z)

- **DIRECT**
  - Type: `char`
  - Direction: `Input`

Specifies whether P is a forward or backward sequence of plane rotations.

= 'F': Forward, P = P(z-1)*...*P(2)*P(1)

= 'B': Backward, P = P(1)*P(2)*...*P(z-1)

- **M**
  - Type: `int`
  - Direction: `Input`

The number of rows of the matrix A. If m <= 1, an immediate return is effected.

- **N**
  - Type: `int`
  - Direction: `Input`

The number of columns of the matrix A. If n <= 1, an immediate return is effected.

- **C**
  - Type: `S/D Pointer`
  - Direction: `Input`
  - Dimension: `(M-1)` if SIDE = 'L'
  - Dimension: `(N-1)` if SIDE = 'R'

The cosines c(k) of the plane rotations.

- **S**
  - Type: `S/D Pointer`
  - Direction: `Input`
  - Dimension: `(M-1)` if SIDE = 'L'

  - Dimension: `(N-1)` if SIDE = 'R'

  The sines s(k) of the plane rotations. The 2-by-2 plane rotation part of the matrix P(k), R(k), has the form

$$R(k) = \begin{pmatrix} c(k) & s(k) \\ -s(k) & c(k) \end{pmatrix}$$

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

  The M-by-N matrix A. On exit, A is overwritten by P*A if SIDE = 'R' or by A*P**T if SIDE = 'L'.

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

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

4.32 LAT2

CULA Routines

The LAT2 functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSlat2d`
  - `culaDlat2s`
  - `culaClat2z`
  - `culaZlat2c`
  - `culaLat2` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSlat2d`
  - `culaDeviceDlat2s`
  - `culaDeviceClat2z`
  - `culaDeviceZlat2c`
  - `culaDeviceLat2` (C++ style, type overloaded)

Description

LAT2 converts a triangular matrix to the opposite precision; S->D, D->S, C->Z, or Z->C.

Parameters

- **UPLO**
  - Type: `char`

  - Direction: `Input`

= 'U': A is upper triangular;

= 'L': A is lower triangular.

- **N**
  - Type: `int`
  - Direction: `Input`

  The number of rows and columns of the matrix A. N >= 0.

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

  On entry, the N-by-N triangular coefficient matrix A.

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

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

- **SA**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Output`
  - Dimension: `(LDSA,N)`

  Only the UPLO part of SA is referenced. On exit, if a culaDataError is returned, the content of the UPLO part of SA is unspecified.

- **LDSA**
  - Type: `int`
  - Direction: `Input`

  The leading dimension of the array SA. LDSA >= max(1,M).

4.33 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`
  - Direction: `Input`

  The number of rows of the matrix Q or P^T to be returned. M >= 0.
- **n**
  - Type: `int`
  - Direction: `Input`

  The number of columns of the matrix Q or P^T to be returned. N >= 0. If VECT = 'Q', M >= N >= min(M,K); if VECT = 'P', N >= M >= 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 >= 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 >= max(1,M).

- **tau**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(min(M,K))` if VECT = 'Q'
  - Dimension: `(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.

4.34 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) . . . H(ihi-1).

Parameters

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

  The order of the matrix Q. N >= 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 <= ILO <= IHI <= N, if N > 0; ILO=1 and IHI=0, 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.

4.35 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) . . . H(2) H(1)

as returned by *GELQF*.

Parameters

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

  The number of rows of the matrix Q. M >= 0.
- **n**
  - Type: `int`
  - Direction: `Input`

  The number of columns of the matrix Q. N >= M.
- **k**
  - Type: `int`

  - Direction: `Input`

  The number of elementary reflectors whose product defines the matrix Q. M >= K >= 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,...,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 >= 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.

4.36 ORGQL/UNGQL

CULA Routines

The ORGQL/UNGQL functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSorgql`
  - `culaDorgql`
  - `culaCungql`
  - `culaZungql`
  - `culaOrgql` (C++ style, type overloaded)
  - `culaUngql` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSorgql`
  - `culaDeviceDorgql`
  - `culaDeviceCungql`
  - `culaDeviceZungql`

- `culaDeviceOrgql` (C++ style, type overloaded)
- `culaDeviceUngql` (C++ style, type overloaded)

Description

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

Q = H(k) . . . H(2) H(1)

as returned by *GELQF*.

Parameters

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

  The number of rows of the matrix Q. M >= 0.

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

  The number of columns of the matrix Q. M >= N >= 0.

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

  The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0.

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

  On entry, the (n-k+i)-th column must contain the vector which defines the elementary reflector H(i), for i = 1,2,...,k, as returned by *GELQF* in the last k columns 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 >= 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.

4.37 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) . . . H(k)

as returned by *GEQRF*.

Parameters

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

  The number of rows of the matrix Q. M >= 0.
- **n**
  - Type: `int`
  - Direction: `Input`

  The number of columns of the matrix Q. M >= N >= 0.
- **k**
  - Type: `int`

  - Direction: `Input`

  The number of elementary reflectors whose product defines the matrix Q. N >= K >= 0.

- **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,...,k, as returned by *GEQRF* in the first k columns 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 >= 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.

4.38 ORGRQ/UNGRQ

CULA Routines

The ORGRQ/UNGRQ functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSorgrq`
  - `culaDorgrq`
  - `culaCungrq`
  - `culaZungrq`
  - `culaOrgrq` (C++ style, type overloaded)
  - `culaUngrq` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSorgrq`
  - `culaDeviceDorgrq`
  - `culaDeviceCungrq`
  - `culaDeviceZungrq`

- `culaDeviceOrgrq` (C++ style, type overloaded)
- `culaDeviceUngrq` (C++ style, type overloaded)

Description

ORGRQ/UNGRQ generates an M-by-N complex matrix Q with orthonormal rows, which is defined as the last M rows of a product of K elementary reflectors of order N

Q = H(1)' H(2)' . . . H(k)'

as returned by *GERQF*.

Parameters

- **M**
  - Direction: `Input`
  - Type: `int`

    The number of rows of the matrix Q. M >= 0.
- **N**
  - Direction: `Input`
  - Type: `int`

    The number of columns of the matrix Q. N >= M.
- **K**
  - Direction: `Input`
  - Type: `int`

    The number of elementary reflectors whose product defines the matrix Q. M >= K >= 0.
- **A**
  - Direction: `Input/Output`
  - Type: `S/D/C/Z Pointer`,
  - Dimension: `(LDA,N)`

    On entry, the (m-k+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. On exit, the M-by-N matrix Q.
- **LDA**
  - Direction: `Input`
  - Type: `int`

    The first dimension of the array A. LDA >= max(1,M).
- **TAU**
  - Direction: `Input`
  - Type: `S/D/C/Z Pointer`,
  - Dimension: `(K)`

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

Differences from LAPACK

See *No Workspace Parameters* section.

4.39 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

	SIDE = 'L'	**SIDE = 'R'**
TRANS = 'N'	Q * C	C * Q
TRANS = 'T'	Q^T * C	C * Q^T

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

$$Q = H(k)...H(2)H(1)$$

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 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`
  - Direction: `Input`
  - Dimension: `(LDA,M)` if SIDE = 'L'
  - Dimension: `(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 >= max(1,M).

Differences from LAPACK

See *No Workspace Parameters* section.

4.40 ORMQL/UNMQL

CULA Routines

The ORMQL/UNMQL functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSormql`
  - `culaDormql`
  - `culaCunmql`
  - `culaZunmql`
  - `culaOrmql` (C++ style, type overloaded)
  - `culaUnmql` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSormql`
  - `culaDeviceDormql`
  - `culaDeviceCunmql`
  - `culaDeviceZunmql`
  - `culaDeviceOrmql` (C++ style, type overloaded)
  - `culaDeviceUnmql` (C++ style, type overloaded)

Description

ORMQL/UNMQL overwrites the general real/complex M-by-N matrix C with

	SIDE = 'L'	**SIDE = 'R'**
TRANS = 'N':	Q * C	C * Q
TRANS = 'C':	Q**H * C	C * Q**H

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

Q = H(k) . . . 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 Q**H from the Left;

  = 'R': apply Q or Q**H from the Right.
- **trans**
  - Type: `char`
  - Direction: `Input`

  = 'N': No transpose, apply Q;

  = 'C': Transpose, apply Q**H.
- **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`,
  - Direction: `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 *GELQF* in the last k columns 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.

  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)`

  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**H*C or C*Q**H 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.

4.41 ORMQR/UNMQR

CULA Routines

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

- Host Memory
  - `culaSormqr`
  - `culaDormqr`
  - `culaCormqr`
  - `culaZormqr`
  - `culaOrmqr` (C++ style, type overloaded)
  - `culaUnmqr` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSormqr`
  - `culaDeviceDormqr`

- culaDeviceCormqr
- culaDeviceZormqr
- culaDeviceOrmqr (C++ style, type overloaded)
- culaDeviceUnmqr (C++ style, type overloaded)

Description

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

	SIDE = 'L'	**SIDE = 'R'**
TRANS = 'N'	Q * C	C * Q
TRANS = 'T'	Q^T * C	C * Q^T

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 *GEQRF*. 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`
  - Direction: `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 *GEQRF* in the first k columns 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. 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)`

  TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by *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 * 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 >= max(1,M).

Differences from LAPACK

See *No Workspace Parameters* section.

4.42 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)

Description

ORMRQ/UNMRQ overwrites the general real M-by-N matrix C with

	SIDE = 'L'	**SIDE = 'R'**
TRANS = 'N'	Q * C	C * Q
TRANS = 'T'	Q^T * C	C * Q^T

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 *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`
  - Direction: `Input`
  - Dimension: `(LDA,M)` if SIDE = 'L'
  - Dimension: `(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.

- **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 *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 Q^T * C or C * Q^T 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.

4.43 PBTRF

CULA Routines

The PBTRF functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSpbtrf`
  - `culaDpbtrf`
  - `culaCpbtrf`
  - `culaZpbtrf`
  - `culaPbtrf` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSpbtrf`
  - `culaDeviceDpbtrf`
  - `culaDeviceCpbtrf`
  - `culaDeviceZpbtrf`
  - `culaDevicePbtrf` (C++ style, type overloaded)

Description

PBTRF computes the Cholesky factorization of a real symmetric/hermitian positive definite band matrix A.

The factorization has the form

$$A = U' * U, if UPLO =' U',$$

or

$$A = L * L', if UPLO =' L',$$

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

This is the blocked version of the algorithm, calling Level 3 BLAS.

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 columns of the matrix A. N >= 0.
- **kd**
  - Type: `int`

  - Direction: `Input`

  The number of superdiagonals of the matrix A if UPLO = 'U', or the number of subdiagonals if UPLO = 'L'. KD >= 0.

- **ab**

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

  On entry, the upper or lower triangle of the symmetric band matrix A, stored in the first KD+1 rows of the array. The j-th column of A is stored in the j-th column of the array AB as follows:

  if UPLO = 'U', AB(kd+1+i-j,j) = A(i,j) for max(1,j-kd)<=i<=j;

  if UPLO = 'L', AB(1+i-j,j) = A(i,j) for j<=i<=min(n,j+kd).

  On exit, the triangular factor U or L from the Cholesky factorization A = U^T * U or A = L * L^T of the band matrix A, in the same storage format as A.

- **ldab**

  - Type: `int`
  - Direction: `Input`

  The leading dimension of the array AB. LDAB >= KD+1.

Performance Details

The performance of PBTRF scales with the bandwidth of the matrix, not the size.

Errors

On a culaDataError, where `culaGetErrorInfo()` = i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

For more information on data errors see the *Data Errors* section.

Differences from LAPACK

See *No Workspace Parameters* section.

4.44 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`
- `culaDevicePosv` (C++ style, type overloaded)

Description

POSV computes the solution to a real system of linear equations

A * 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

$$A = U' * U,$$

if UPLO = 'U', or

$$A = L * L',$$

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 * 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 >= 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/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 culaNoError is returned, the factor U or L from the Cholesky factorization A = U^T * U or A = L * L^T .

- **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 N-by-NRHS right hand side matrix B.

  On exit, if culaNoError is returned, the N-by-NRHS solution matrix X.

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

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

4.45 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

$$A = U' * U$$

if UPLO = 'U', or

$$A = L * L'$$

if UPLO = '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**
  - 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 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 culaNoError is returned, the factor U or L from the Cholesky factorization A = U^T * U or A = L * L^T.
- **lda**
  - Type: `int`
  - Direction: `Input`

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

Errors

On a culaDataError, where `culaGetErrorInfo()` = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

For more information on data errors see the *Data Errors* section.

4.46 POTRI

CULA Routines

The POTRI functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSpotri`
  - `culaDpotri`
  - `culaCpotri`
  - `culaZpotri`
  - `culaPotri` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSpotri`
  - `culaDeviceDpotri`
  - `culaDeviceCpotri`
  - `culaDeviceZpotri`
  - `culaDevicePotri` (C++ style, type overloaded)

Description

POTRI computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization $A = U^H * U$ or $A = L * L^H$ 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.
- **a**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input/Output`
  - Dimension: `(LDA,N)`

  On entry, the triangular factor U or L from the Cholesky factorization $A = U^H * U$ or $A = L * L^H$, as computed by *POTRF*.

  On exit, the upper or lower triangle of the (Hermitian) inverse of A, overwriting the input factor U or L.

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

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

Errors

On a culaDataError, where `culaGetErrorInfo()` = i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

4.47 POTRS

CULA Routines

The POTRS functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSpotrs`
  - `culaDpotrs`
  - `culaCpotrs`
  - `culaZpotrs`
  - `culaPotrs` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSpotrs`
  - `culaDeviceDpotrs`
  - `culaDeviceCpotrs`
  - `culaDeviceZpotrs`
  - `culaDevicePotrs` (C++ style, type overloaded)

Description

POTRS solves a system of linear equations A*X = B with a symmetric positive definite matrix A using the Cholesky factorization $A = U^T * U$ or $A = L * L^T$ 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).

4.48 STEBZ

CULA Routines

The STEBZ functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSstebz`
  - `culaDstebz`
  - `culaStebz` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSstebz`
  - `culaDeviceDstebz`
  - `culaDeviceStebz` (C++ style, type overloaded)

Description

STEBZ computes the eigenvalues of a symmetric tridiagonal matrix T. The user may ask for all eigenvalues, all eigenvalues in the half-open interval (VL, VU], or the IL-th through IU-th eigenvalues.

To avoid overflow, the matrix must be scaled so that its largest element is no greater than overflow**(1/2) * underflow**(1/4) in absolute value, and for greatest accuracy, it should not be much smaller than that.

See W. Kahan "Accurate Eigenvalues of a Symmetric Tridiagonal Matrix", Report CS41, Computer Science Dept., Stanford University, July 21, 1966.

Please note that complex versions of this function do not exist.

Parameters

- **range**
  - Type: `char`
  - Direction: `Input`

  = 'A': ("All") all eigenvalues will be found.

  = 'V': ("Value") all eigenvalues in the half-open interval (VL, VU] will be found.

  = 'I': ("Index") the IL-th through IU-th eigenvalues (of the entire matrix) will be found.
- **order**
  - Type: `char`
  - Direction: `Input`

  = 'B': ("By Block") the eigenvalues will be grouped by split-off block (see IBLOCK, ISPLIT) and ordered from smallest to largest within the block. This code behaves similarly to "E" and will report only one block.

  = 'E': ("Entire matrix") the eigenvalues for the entire matrix will be ordered from smallest to largest.
- **n**
  - Type: `int`
  - Direction: `Input`

  The order of the tridiagonal matrix T. N >= 0.
- **vl**
  - Type: `S/D Value`
  - Direction: `Input`
- **vu**
  - Type: `S/D Value`
  - Direction: `Input`

  If RANGE='V', the lower and upper bounds of the interval to be searched for eigenvalues. Eigenvalues less than or equal to VL, or greater than VU, will not be returned. VL < VU. Not referenced if RANGE = 'A' or 'I'.
- **il**
  - Type: `int`
  - Direction: `Input`

- **iu**
  - Type: `int`
  - Direction: `Input`

  If RANGE='I', the indices (in ascending order) of the smallest and largest eigenvalues to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

- **abstol**
  - Type: `S/D`
  - Direction: `Input`

  The absolute tolerance for the eigenvalues. An eigenvalue (or cluster) is considered to be located if it has been determined to lie in an interval whose width is ABSTOL or less. If ABSTOL is less than or equal to zero, then ULP*| T | will be used, where | T | means the 1-norm of T.

  Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold, not zero.

- **d**
  - Type: `S/D Pointer`
  - Direction: `Input`
  - Dimension: `(N)`

  The N diagonal elements of the tridiagonal matrix T.

- **e**
  - Type: `S/D Pointer`
  - Direction: `Input`
  - Dimension: `(N-1)`

  The (N-1) off-diagonal elements of the tridiagonal matrix T. Not referenced if N <= 1.

- **m**
  - Type: `int Pointer, always host`
  - Direction: `Output`

  The actual number of eigenvalues found. 0 <= M <= N.

- **nsplit**
  - Type: `int Pointer, always host`
  - Direction: `Output`

  The number of diagonal blocks in the matrix T. 1 <= NSPLIT <= N.

- **w**
  - Type: `S/D Pointer`
  - Direction: `Output`
  - Dimension: `(N)`

  On exit, the first M elements of W will contain the eigenvalues. (STEBZ may use the remaining N-M elements as workspace.)

- **iblock**

  - Type: `int Pointer`
  - Direction: `Output`
  - Dimension: `(N)`

  At each row/column j where E(j) is zero or small, the matrix T is considered to split into a block diagonal matrix. On exit, if culaNoError is returned, IBLOCK(i) specifies to which block (from 1 to the number of blocks) the eigenvalue W(i) belongs. (STEBZ may use the remaining N-M elements as workspace.)

- **isplit**
  - Type: `int Pointer`
  - Direction: `Output`
  - Dimension: `(N)`

  The splitting points, at which T breaks up into submatrices. The first submatrix consists of rows/columns 1 to ISPLIT(1), the second of rows/columns ISPLIT(1)+1 through ISPLIT(2), etc., and the NSPLIT-th consists of rows/columns ISPLIT(NSPLIT-1)+1 through ISPLIT(NSPLIT)=N.

  (Only the first NSPLIT elements will actually be used, but since the user cannot know a priori what value NSPLIT will have, N words must be reserved for ISPLIT.)

Errors

On a culaDataError, where `culaGetErrorInfo()` = 1 or 3: Bisection failed to converge for some eigenvalues; these eigenvalues are flagged by a negative block number. The effect is that the eigenvalues may not be as accurate as the absolute and relative tolerances. This is generally caused by unexpectedly inaccurate arithmetic.

On a culaDataError, where `culaGetErrorInfo()` = 2 or 3: RANGE='I' only: Not all of the eigenvalues IL:IU were found.

For more information on data errors see the *Data Errors* section.

Differences from LAPACK

See *No Workspace Parameters* section.

4.49 STEQR

CULA Routines

The STEQR functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSsteqr`
  - `culaDsteqr`
  - `culaCsteqr`
  - `culaZsteqr`
  - `culaSteqr` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSsteqr`
  - `culaDeviceDsteqr`

- culaDeviceCsteqr
- culaDeviceZsteqr
- culaDeviceSteqr (C++ style, type overloaded)

Description

STEQR computes all eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the implicit QL or QR method. The eigenvectors of a full or band real/complex symmetric/Hermitian matrix can also be found if *SYTRD/HETRD* or HPTRD or HBTRD has been used to reduce this matrix to tridiagonal form.

Parameters

- **compz**
  - Type: char
  - Direction: Input

  = 'N': Compute eigenvalues only.

  = 'V': Compute eigenvalues and eigenvectors of the original symmetric/Hermitian matrix. On entry, Z must contain the unitary matrix used to reduce the original matrix to tridiagonal form.

  = 'I': Compute eigenvalues and eigenvectors of the tridiagonal matrix. Z is initialized to the identity matrix.

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

  The order of the matrix. N >= 0.

- **d**
  - Type: S/D Pointer,
  - Direction: Input/Output
  - Dimension: (N)

  On entry, the diagonal elements of the tridiagonal matrix.

  On exit, if culaNoError is returned, the eigenvalues in ascending order.

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

  On entry, the (n-1) subdiagonal elements of the tridiagonal matrix.

  On exit, E has been destroyed.

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

  On entry, if COMPZ = 'V', then Z contains the unitary matrix used in the reduction to tridiagonal form.

On exit, if culaNoError is returned, then if COMPZ = 'V', Z contains the orthonormal eigenvectors of the original symmetric/Hermitian matrix, and if COMPZ = 'I', Z contains the orthonormal eigenvectors of the symmetric tridiagonal matrix. If COMPZ = 'N', then Z is not referenced.

- **ldz**
  - Type: `int`
  - Direction: `Input`

  The leading dimension of the array Z. LDZ >= 1, and if eigenvectors are desired, then LDZ >= max(1,N).

Errors

On a culaDataError, where `culaGetErrorInfo()` > 0, the algorithm has failed to find all the eigenvalues in a total of 30*N iterations;

where `culaGetErrorInfo()` = i, then i elements of E have not converged to zero; on exit, D and E contain the elements of a symmetric tridiagonal matrix which is orthogonally similar to the original matrix.

For more information on data errors see the *Data Errors* section.

Differences from LAPACK

See *No Workspace Parameters* section.

4.50 SYEV/HEEV

CULA Routines

The SYEV/HEEV functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSsyev`
  - `culaDsyev`
  - `culaCheev`
  - `culaZheev`
  - `culaSyev` (C++ style, type overloaded)
  - `culaHeev` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSsyev`
  - `culaDeviceDsyev`
  - `culaDeviceCheev`
  - `culaDeviceZheev`
  - `culaDeviceSyev` (C++ style, type overloaded)
  - `culaDeviceHeev` (C++ style, type overloaded)

Description

SYEV/HEEV computes all eigenvalues and, optionally, eigenvectors of a real/complex symmetric/Hermitian matrix A.

Parameters

- **jobz**
  - Type: `char`
  - Direction: `Input`

  = 'N': Compute eigenvalues only;

  = 'V': Compute eigenvalues and eigenvectors.
- **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.
- **a**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input/Output`
  - Dimension: `(LDA, N)`

  On entry, the symmetric/Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A.

  On exit, if JOBZ = 'V', then if culaNoError is returned, A contains the orthonormal eigenvectors of the matrix A. If JOBZ = 'N', then on exit the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.
- **lda**
  - Type: `int`
  - Direction: `Input`

  The leading dimension of the array A. LDA >= max(1,N).
- **w**
  - Type: `S/D Pointer`
  - Direction: `Output`
  - Dimension: `(N)`

  If culaNoError is returned, the eigenvalues in ascending order.

Errors

On a culaDataError, where `culaGetErrorInfo()` = i, the algorithm failed to converge; i off-diagonal elements of an intermediate tridiagonal form did not converge to zero.

For more information on data errors see the *Data Errors* section.

Differences from LAPACK

See *No Workspace Parameters* section.

4.51 SYEVX/HEEVX

CULA Routines

The SYEVX/HEEVX functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSsyevx`
  - `culaDsyevx`
  - `culaCheevx`
  - `culaZheevx`
  - `culaSyevx` (C++ style, type overloaded)
  - `culaHeevx` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSsyevx`
  - `culaDeviceDsyevx`
  - `culaDeviceCheevx`
  - `culaDeviceZheevx`
  - `culaDeviceSyevx` (C++ style, type overloaded)
  - `culaDeviceHeevx` (C++ style, type overloaded)

Description

SYEVX/HEEVX computes selected eigenvalues and, optionally, eigenvectors of a real/complex symmetric/Hermitian matrix A. Eigenvalues and eigenvectors can be selected by specifying either a range of values or a range of indices for the desired eigenvalues.

Parameters

- **jobz**
  - Type: `char`
  - Direction: `Input`

  = 'N': Compute eigenvalues only;

  = 'V': Compute eigenvalues and eigenvectors.
- **range**
  - Type: `char`
  - Direction: `Input`

  = 'A': all eigenvalues will be found.

  = 'V': all eigenvalues in the half-open interval (VL,VU] will be found.

  = 'I': the IL-th through IU-th eigenvalues will be found.

- **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.

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

  On entry, the symmetric/Hermitian matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A.

  On exit, the lower triangle (if UPLO='L') or the upper triangle (if UPLO='U') of A, including the diagonal, is destroyed.

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

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

- **vl**
  - Type: `S/D/C/Z`
  - Direction: `Input`

- **vu**
  - Type: `S/D/C/Z`
  - Direction: `Input`

  If RANGE='V', the lower and upper bounds of the interval to be searched for eigenvalues. VL < VU. Not referenced if RANGE = 'A' or 'I'.

- **il**
  - Type: `int`
  - Direction: `Input`

- **iu**
  - Type: `int`
  - Direction: `Input`

  If RANGE='I', the indices (in ascending order) of the smallest and largest eigenvalues to be returned. 1 <= IL <= IU <= N, if N > 0; IL = 1 and IU = 0 if N = 0. Not referenced if RANGE = 'A' or 'V'.

- **abstol**
  - Type: `S/D/C/Z`
  - Direction: `Input`

  The absolute error tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval [a,b] of width less than or equal to

  ABSTOL + EPS * max(| a | , | b |) ,

  where EPS is the machine precision. If ABSTOL is less than or equal to zero, then EPS*| T | will be used in its place, where | T | is the 1-norm of the tridiagonal matrix obtained by reducing A to tridiagonal form.

  Eigenvalues will be computed most accurately when ABSTOL is set to twice the underflow threshold, not zero. If this routine returns with culaDataError, indicating that some eigenvectors did not converge, try setting ABSTOL to the recommended value.

  See "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," by Demmel and Kahan, LAPACK Working Note #3.

- **m**
  - Type: `int Pointer, always host`
  - Direction: `Output`

  The total number of eigenvalues found. 0 <= M <= N. If RANGE = 'A', M = N, and if RANGE = 'I', M = IU-IL+1.

- **w**
  - Type: `S/D Pointer`,
  - Direction: `Output`
  - Dimension: `(N)`

  On normal exit, the first M elements contain the selected eigenvalues in ascending order.

- **z**
  - Type: `S/D/C/Z Pointer`,
  - Direction: `Output`
  - Dimension: `(LDZ, max(1,M))`

  If JOBZ = 'V', then if culaNoError is returned, the first M columns of Z contain the orthonormal eigenvectors of the matrix A corresponding to the selected eigenvalues, with the i-th column of Z holding the eigenvector associated with W(i).

  If an eigenvector fails to converge, then that column of Z contains the latest approximation to the eigenvector, and the index of the eigenvector is returned in IFAIL.

  If JOBZ = 'N', then Z is not referenced.

  Note: the user must ensure that at least max(1,M) columns are supplied in the array Z; if RANGE = 'V', the exact value of M is not known in advance and an upper bound must be used.

- **ldz**
  - Type: `int`
  - Direction: `Input`

  The leading dimension of the array Z. LDZ >= 1, and if JOBZ = 'V', LDZ >= max(1,N).

- **ifail**
  - Type: `int Pointer`
  - Direction: `Output`
  - Dimension: `(N)`

  If JOBZ = 'V', then if culaNoError is returned, the first M elements of IFAIL are zero. If culaDataError is returned, then IFAIL contains the indices of the eigenvectors that failed to converge.

  If JOBZ = 'N', then IFAIL is not referenced.

Errors

On a culaDataError, where `culaGetErrorInfo()` = i, then i eigenvectors failed to converge. Their indices are stored in array IFAIL.

For more information on data errors see the *Data Errors* section.

Differences from LAPACK

See *No Workspace Parameters* section.

4.52 SYRDB/HERDB

CULA Routines

The SYRDB/HERDB functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSsyrdb`
  - `culaDsyrdb`
  - `culaCherdb`
  - `culaZherdb`
  - `culaSyrdb` (C++ style, type overloaded)
  - `culaHerdb` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSsyrdb`
  - `culaDeviceDsyrdb`
  - `culaDeviceCherdb`
  - `culaDeviceZherdb`
  - `culaDeviceSyrdb` (C++ style, type overloaded)
  - `culaDeviceHerdb` (C++ style, type overloaded)

Description

Given a symmetric/Hermitian N-by-N matrix A, SYRDB/HERDB reduces A to a symmetric tridiagonal form (T) using a series of orthogonal transformations (Q), such that

$$A = Q * T * Q'.$$

Note: This function should be used in place of *SYTRD/HETRD* when an orthogonal Q is not needed as the performance of these functions does not scale to large matrix sizes.

See "A framework for symmetric band reduction," by Bischof, Lang, and Sun, ACM Transactions on Mathematical Software (TOMS) archive Volume 26, Issue 4.

Parameters

- **jobz**
  - Type: `char`
  - Direction: `Input`

  Specifies which matrices outputs are contained within the outputs A and Z.

  = 'N' : Forms the symmetric tridiagonal matrix T. Overwrites A with the banded matrix, B, and the information needed to construct Q_B.

  = 'V' : Forms the symmetric tridiagonal matrix T. Overwrites A with Q.

  = 'U' : Forms the symmetric tridiagonal matrix T. Overwrites A with the banded matrix, B, and the information needed to construct Q_B. Also, overwrites Z with Z*Q.

- **uplo**
  - Type: `char`
  - Direction: `Input`

  Specifies the triangular region where the symmetric input matrix is defined.

  = 'U' : defined within the upper triangular part of A

  = 'L' : defined within the lower triangular part of A

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

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

- **kd**
  - Type: `int`
  - Direction: `Input`

  The bandwidth of the banded output matrix, B. KD >= 1.

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

  On entry, A is a real/complex symmetric/Hermitian matrix described by UPLO.

  On exit, given the job code of 'V', A is overwritten by Q. Given a job code of 'N' or 'U', A is overwritten by B and Q_B as defined by UPLO and KD.

- **lda**
  - Type: `int`

  - Direction: `Input`

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

- **d**
  - Type: `S/D/C/Z Pointer`,
  - Direction: `Output`
  - Dimension: `max(1,N)`

  Holds the diagonal elements of the symmetric tridiagonal matrix T.

- **e**
  - Type: `S/D/C/Z Pointer`,
  - Direction: `Output`
  - Dimension: `max(N-1)`

  Holds the off-diagonal elements of the symmetric tridiagonal matrix T.

- **tau**
  - Type: `S/D/C/Z Pointer`,
  - Direction: `Output`
  - Dimension: `max(1,N-KD-1)`

  The scalar factors of the elementary reflectors that form Q_B.

- **z**
  - Type: `S/D Pointer`,
  - Direction: `Input/Output`
  - Dimension: `(N-1)`

  An optional matrix that can be multiplied by Q given a 'U' job code.

  = 'U' : contains the product of itself and Q (Z*Q).

  = 'N', 'V' : not referenced

- **ldz**
  - Type: `int`
  - Direction: `Input`

  The leading dimension of the array Z. LDZ >= max(1,N).

Differences from LAPACK

See *No Workspace Parameters* section.

4.53 SYTRD/HETRD

This function is not implemented in CULA. If a tridiagonal reduction is required, please see *SYRDB/HERDB*

4.54 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**
  - Type: `int`
  - Direction: `Input`

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

Errors

On a culaDataError, where `culaGetErrorInfo()` = i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

For more information on data errors see the *Data Errors* section.

4.55 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

$$A * X = B or A' * X = B,$$

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 * X = B (No transpose)

  = 'T': A^T * X = B (Transpose)

  = 'C': A^H * 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 culaNoError is returned, the solution matrix X.

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

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

Errors

On a culaDataError, where `culaGetErrorInfo()` = i, the i-th diagonal element of A is zero, indicating that the matrix is singular and the solutions X have not been computed.

For more information on data errors see the *Data Errors* section.

CHAPTER

FIVE

BLAS ROUTINES

This section documents the BLAS functions that CULA provides (see the *cula_blas.h* and *cula_blas_device.h* headers). 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 BLAS will also be listed.

5.1 GEMM

CULA Routines

The GEMM functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSgemm`
  - `culaDgemm`
  - `culaCgemm`
  - `culaZgemm`
  - `culaGemm` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSgemm`
  - `culaDeviceDgemm`
  - `culaDeviceCgemm`
  - `culaDeviceZgemm`
  - `culaDeviceGemm` (C++ style, type overloaded)

Description

GEMM performs one of the matrix-matrix operations

$$C := alpha * op(A) * op(B) + beta * C,$$

where op(X) is one of

$$op(X) = X$$

or

$$op(X) = X'$$

or

$$op(X) = conjg(X'),$$

alpha and beta are scalars, and A, B and C are matrices, with op(A) an m by k matrix, op(B) a k by n matrix and C an m by n matrix.

Parameters

- **transa**
  - Type: `char`.
  - Direction: `Input`

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

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

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

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

- **transb**
  - Type: `char`.
  - Direction: `Input`

    On entry, TRANSB specifies the form of op(B) to be used in the matrix multiplication as follows:

    TRANSB = 'N' or 'n', op(B) = B.

    TRANSB = 'T' or 't', op(B) = B^T.

    TRANSB = 'C' or 'c', op(B) = conjg(B^T).

- **m**
  - Type: `int`.
  - Direction: `Input`

    On entry, M specifies the number of rows of the matrix op(A) and of the matrix C. M must be at least zero.

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

    On entry, N specifies the number of columns of the matrix op(B) and the number of columns of the matrix C. N must be at least zero.

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

    On entry, K specifies the number of columns of the matrix op(A) and the number of rows of the matrix op(B). K must be at least zero.

- **alpha**
  - Type: `S/D/C/Z`

  - Direction: `Input`

    On entry, ALPHA specifies the scalar alpha.

- **a**

  - Type: `S/D/C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(M,K)` if TRANSA = 'N'
  - Dimension: `(K,M)` if TRANSA != 'N'

    Before entry with TRANSA = 'N' or 'n', the leading m by k part of the array A must contain the matrix A, otherwise the leading k by m part of the array A must contain the matrix A.

- **lda**

  - Type: `int`.
  - Direction: `Input`

    On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANSA = 'N' or 'n' then LDA must be at least max(1, m), otherwise LDA must be at least max(1, k).

- **b**

  - Type: `S/D/C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(K,N)` if TRANSA = 'N'
  - Dimension: `(N,K)` if TRANSA != 'N'

    Before entry with TRANSB = 'N' or 'n', the leading k by n part of the array B must contain the matrix B, otherwise the leading n by k part of the array B must contain the matrix B.

- **ldb**

  - Type: `int`.
  - Direction: `Input`

    On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANSB = 'N' or 'n' then LDB must be at least max(1, k), otherwise LDB must be at least max(1, n).

- **beta**

  - Type: `S/D/C/Z`
  - Direction: `Input`

    On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.

- **c**

  - Type: `S/D/C/Z Pointer`
  - Direction: `Input/Output`
  - Dimension: `(M,N)`

Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry.

On exit, the array C is overwritten by the m by n matrix (alpha*op(A)*op(B) + beta*C).

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

    On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, m).

5.2 GEMV

CULA Routines

The GEMV functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSgemv`
  - `culaDgemv`
  - `culaCgemv`
  - `culaZgemv`
  - `culaGemv` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSgemv`
  - `culaDeviceDgemv`
  - `culaDeviceCgemv`
  - `culaDeviceZgemv`
  - `culaDeviceGemv` (C++ style, type overloaded)

Description

GEMV performs one of the matrix-vector operations

$$y := alpha * A * x + beta * y,$$

or

$$y := alpha * A' * x + beta * y,$$

or

$$y := alpha * conjg(A') * x + beta * y,$$

where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.

Parameters

- **trans**
  - Type: `char`.

  - Direction: `Input`

  On entry, TRANS specifies the operation to be performed as follows:

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

  TRANS = 'T' or 't' $y := alpha * A^T * x + beta * y$.

  TRANS = 'C' or 'c' $y := alpha * conjg(A^T) * x + beta * y$.

- **m**
  - Type: `int`.
  - Direction: `Input`

  On entry, M specifies the number of rows of the matrix A. M must be at least zero.

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

  On entry, N specifies the number of columns of the matrix A. N must be at least zero.

- **alpha**
  - Type: `S/D/C/Z`
  - Direction: `Input`

  On entry, ALPHA specifies the scalar alpha.

- **a**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(M,N)`

  Before entry, the leading m by n part of the array A must contain the matrix of coefficients.

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

  On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, m).

- **x**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(N*INCX)` if TRANS = 'N'
  - Dimension: `(M*INCX)` if TRANS != 'N'

  Before entry, the incremented array X must contain the vector x.

- **incx**
  - Type: `int`.

  - Direction: `Input`

  On entry, INCX specifies the increment for the elements of X. INCX must not be zero.

- **beta**

  - Type: `S/D/C/Z`
  - Direction: `Input`

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

- **y**

  - Type: `S/D/C/Z Pointer`
  - Direction: `Input/Output`
  - Dimension: `(M,N)`
  - Dimension: `(M*INCY)` if TRANS = 'N'
  - Dimension: `(N*INCY)` if TRANS != 'N'

  Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.

- **incy**

  - Type: `int`.
  - Direction: `Input`

  On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.

5.3 HEMM

CULA Routines

The HEMM functionality is implemented by the following CULA routines:

- Host Memory
  - `culaChemm`
  - `culaZhemm`
  - `culaHemm` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceChemm`
  - `culaDeviceZhemm`
  - `culaDeviceHemm` (C++ style, type overloaded)

Description

HEMM performs one of the matrix-matrix operations

C := alpha*A*B + beta*C,

or

C := alpha*B*A + beta*C,

where alpha and beta are scalars, A is an hermitian matrix and B and C are m by n matrices.

Parameters

- **side**
  - Type: `char`.
  - Direction: `Input`

  On entry, SIDE specifies whether the hermitian matrix A appears on the left or right in the operation as follows:

  SIDE = 'L' or 'l' C := alpha*A*B + beta*C,

  SIDE = 'R' or 'r' C := alpha*B*A + beta*C,

- **uplo**
  - Type: `char`.
  - Direction: `Input`

  On entry, UPLO specifies whether the upper or lower triangular part of the hermitian matrix A is to be referenced as follows:

  UPLO = 'U' or 'u' Only the upper triangular part of the hermitian matrix is to be referenced.

  UPLO = 'L' or 'l' Only the lower triangular part of the hermitian matrix is to be referenced.

- **m**
  - Type: `int`.
  - Direction: `Input`

  On entry, M specifies the number of rows of the matrix C. M must be at least zero.

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

  On entry, N specifies the number of columns of the matrix C. N must be at least zero.

- **alpha**
  - Type: `C/Z`
  - Direction: `Input`

  On entry, ALPHA specifies the scalar alpha.

- **a**
  - Type: `C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(M,M)` if SIDE = 'L'
  - Dimension: `(N,N)` if SIDE = 'R'

  Before entry with SIDE = 'L' or 'l', the m by m part of the array A must contain the hermitian matrix, such that when UPLO = 'U' or 'u', the leading m by m upper triangular part of the array A must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading m by m lower triangular part of the

array A must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of A is not referenced.

Before entry with SIDE = 'R' or 'r', the n by n part of the array A must contain the hermitian matrix, such that when UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of A is not referenced.

Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero.

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

  On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least max(1, m), otherwise LDA must be at least max(1, n).

- **b**
  - Type: `C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(M,N)`

  Before entry, the leading m by n part of the array B must contain the matrix B.

- **ldb**
  - Type: `int`.
  - Direction: `Input`

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

- **beta**
  - Type: `C/Z`
  - Direction: `Input`

  On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.

- **c**
  - Type: `C/Z Pointer`
  - Direction: `Input/Output`
  - Dimension: `(M,N)`

  Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry.

  On exit, the array C is overwritten by the m by n updated matrix.

- **ldc**
  - Type: `int`.

- Direction: `Input`

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, m).

5.4 HER2K

CULA Routines

The HER2K functionality is implemented by the following CULA routines:

- Host Memory
  - `culaCher2k`
  - `culaZher2k`
  - `culaHer2k` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceCher2k`
  - `culaDeviceZher2k`
  - `culaDeviceHer2k` (C++ style, type overloaded)

Description

HER2K performs one of the hermitian rank 2k operations

C := alpha*A*conjg(B^T) + conjg(alpha)*B*conjg(A^T) + beta*C,

or

C := alpha*conjg(A^T)*B + conjg(alpha)*conjg(B^T)*A + beta*C,

where alpha and beta are scalars with beta real, C is an n by n hermitian matrix and A and B are n by k matrices in the first case and k by n matrices in the second case.

Parameters

- **uplo**
  - Type: `char`.
  - Direction: `Input`

  On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

  UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.

  UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.
- **trans**
  - Type: `char`.
  - Direction: `Input`

  On entry, TRANS specifies the operation to be performed as follows:

  TRANS = 'N' or 'n' C := alpha*A*conjg(B^T) + conjg(alpha)*B*conjg(A^T) + beta*C.

  TRANS = 'C' or 'c' C := alpha*conjg(A^T)*B + conjg(alpha)*conjg(B^T)*A + beta*C.

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

  On entry, N specifies the order of the matrix C. N must be at least zero.

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

  On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrices A and B, and on entry with TRANS = 'C' or 'c', K specifies the number of rows of the matrices A and B. K must be at least zero.

- **alpha**
  - Type: `C/Z`
  - Direction: `Input`

  On entry, ALPHA specifies the scalar alpha.

- **a**
  - Type: `C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(N,K)` if TRANS = 'N'
  - Dimension: `(K,N)` if TRANS != 'N'

  Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.

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

  On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k).

- **b**
  - Type: `C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(N,K)` if TRANS = 'N'
  - Dimension: `(K,N)` if TRANS != 'N'

  Before entry with TRANS = 'N' or 'n', the leading n by k part of the array B must contain the matrix B, otherwise the leading k by n part of the array B must contain the matrix B.

- **ldb**
  - Type: `int`.

  - Direction: `Input`

  On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDB must be at least max(1, n), otherwise LDB must be at least max(1, k).

- **beta**

  - Type: `S/D`
  - Direction: `Input`

  On entry, BETA specifies the scalar beta.

- **c**

  - Type: `C/Z Pointer`
  - Direction: `Input/Output`
  - Dimension: `(N,N)`

  Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix.

  Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.

  Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

- **ldc**

  - Type: `int`.
  - Direction: `Input`

  On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, n).

5.5 HERK

CULA Routines

The HERK functionality is implemented by the following CULA routines:

- Host Memory
  - `culaCherk`
  - `culaZherk`
  - `culaHerk` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceCherk`
  - `culaDeviceZherk`
  - `culaDeviceHerk` (C++ style, type overloaded)

Description

HERK performs one of the hermitian rank k operations

C := alpha*A*conjg(A^T) + beta*C,

or

C := alpha*conjg(A^T)*A + beta*C,

where alpha and beta are real scalars, C is an n by n hermitian matrix and A is an n by k matrix in the first case and a k by n matrix in the second case.

Parameters

- **uplo**
  - Type: `char`.
  - Direction: `Input`

  On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

  UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.

  UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

- **trans**
  - Type: `char`.
  - Direction: `Input`

  On entry, TRANS specifies the operation to be performed as follows:

  TRANS = 'N' or 'n' C := alpha*A*conjg(A^T) + beta*C.

  TRANS = 'C' or 'c' C := alpha*conjg(A^T)*A + beta*C.

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

  On entry, N specifies the order of the matrix C. N must be at least zero.

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

  On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrix A, and on entry with TRANS = 'C' or 'c', K specifies the number of rows of the matrix A. K must be at least zero.

- **alpha**
  - Type: `S/D`
  - Direction: `Input`

  On entry, ALPHA specifies the scalar alpha.

- **a**
  - Type: `C/Z Pointer`
  - Direction: `Input`

  - Dimension: `(N,K)` if TRANS = 'N'
  - Dimension: `(K,N)` if TRANS != 'N'

  Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.

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

  On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k).

- **beta**
  - Type: `S/D`
  - Direction: `Input`

  On entry, BETA specifies the scalar beta.

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

  Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the hermitian matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix.

  Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the hermitian matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.

  Note that the imaginary parts of the diagonal elements need not be set, they are assumed to be zero, and on exit they are set to zero.

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

  On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, n).

5.6 SYMM

CULA Routines

The SYMM functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSsymm`

  - `culaDsymm`
  - `culaCsymm`
  - `culaZsymm`
  - `culaSymm` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSsymm`
  - `culaDeviceDsymm`
  - `culaDeviceCsymm`
  - `culaDeviceZsymm`
  - `culaDeviceSymm` (C++ style, type overloaded)

Description

SYMM performs one of the matrix-matrix operations

C := alpha*A*B + beta*C,

or

C := alpha*B*A + beta*C,

where alpha and beta are scalars, A is a symmetric matrix and B and C are m by n matrices.

Parameters

- **side**
  - Type: `char`.
  - Direction: `Input`

  On entry, SIDE specifies whether the symmetric matrix A appears on the left or right in the operation as follows:

  SIDE = 'L' or 'l' C := alpha*A*B + beta*C,

  SIDE = 'R' or 'r' C := alpha*B*A + beta*C,
- **uplo**
  - Type: `char`.
  - Direction: `Input`

  On entry, UPLO specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows:

  UPLO = 'U' or 'u' Only the upper triangular part of the symmetric matrix is to be referenced.

  UPLO = 'L' or 'l' Only the lower triangular part of the symmetric matrix is to be referenced.
- **m**
  - Type: `int`.
  - Direction: `Input`

  On entry, M specifies the number of rows of the matrix C. M must be at least zero.
- **n**
  - Type: `int`.

  - Direction: `Input`

  On entry, N specifies the number of columns of the matrix C. N must be at least zero.

- **alpha**

  - Type: `S/D/C/Z`
  - Direction: `Input`

  On entry, ALPHA specifies the scalar alpha.

- **a**

  - Type: `S/D/C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(M,M)` if SIDE = 'L'
  - Dimension: `(N,N)` if SIDE = 'R'

  Before entry with SIDE = 'L' or 'l', the m by m part of the array A must contain the symmetric matrix, such that when UPLO = 'U' or 'u', the leading m by m upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading m by m lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced.

  Before entry with SIDE = 'R' or 'r', the n by n part of the array A must contain the symmetric matrix, such that when UPLO = 'U' or 'u', the leading n by n upper triangular part of the array A must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of A is not referenced, and when UPLO = 'L' or 'l', the leading n by n lower triangular part of the array A must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of A is not referenced.

- **lda**

  - Type: `int`.
  - Direction: `Input`

  On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When SIDE = 'L' or 'l' then LDA must be at least max(1, m), otherwise LDA must be at least max(1, n).

- **b**

  - Type: `S/D/C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(M,N)`

  Before entry, the leading m by n part of the array B must contain the matrix B.

- **ldb**

  - Type: `int`.
  - Direction: `Input`

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

- **beta**

  - Type: `S/D/C/Z`

  - Direction: `Input`

  On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.

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

  Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry.

  On exit, the array C is overwritten by the m by n updated matrix.

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

  On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, m).

5.7 SYR2K

CULA Routines

The SYR2K functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSsyr2k`
  - `culaDsyr2k`
  - `culaCsyr2k`
  - `culaZsyr2k`
  - `culaSyr2k` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSsyr2k`
  - `culaDeviceDsyr2k`
  - `culaDeviceCsyr2k`
  - `culaDeviceZsyr2k`
  - `culaDeviceSyr2k` (C++ style, type overloaded)

Description

SYR2K performs one of the symmetric rank 2k operations

C := alpha*A*B^T + alpha*B*A^T + beta*C,

or

C := alpha*A^T * B + alpha*B^T * A + beta*C,

where alpha and beta are scalars, C is an n by n symmetric matrix and A and B are n by k matrices in the first case and k by n matrices in the second case.

Parameters

- **uplo**
  - Type: `char`.
  - Direction: `Input`

  On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

  UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.

  UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.

- **trans**
  - Type: `char`.
  - Direction: `Input`

  On entry, TRANS specifies the operation to be performed as follows:

  TRANS = 'N' or 'n' C := alpha*A*B^T + alpha*B*A^T + beta*C.

  TRANS = 'T' or 't' C := alpha*A^T * B + alpha*B^T * A + beta*C.

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

  On entry, N specifies the order of the matrix C. N must be at least zero.

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

  On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrices A and B, and on entry with TRANS = 'T' or 't', K specifies the number of rows of the matrices A and B. K must be at least zero.

- **alpha**
  - Type: `S/D/C/Z`
  - Direction: `Input`

  On entry, ALPHA specifies the scalar alpha.

- **a**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(N,K)` if TRANS = 'N'
  - Dimension: `(K,N)` if TRANS != 'N'

  Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.

- **lda**

  - Type: `int`.
  - Direction: `Input`

  On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k).

- **b**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(N,K)` if TRANS = 'N'
  - Dimension: `(K,N)` if TRANS != 'N'

  Before entry with TRANS = 'N' or 'n', the leading n by k part of the array B must contain the matrix B, otherwise the leading k by n part of the array B must contain the matrix B.

- **ldb**
  - Type: `int`.
  - Direction: `Input`

  On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDB must be at least max(1, n), otherwise LDB must be at least max(1, k).

- **beta**
  - Type: `S/D/C/Z`
  - Direction: `Input`

  On entry, BETA specifies the scalar beta.

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

  Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix.

  Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.

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

  On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, n).

5.8 SYRK

CULA Routines

The SYRK functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSsyrk`
  - `culaDsyrk`
  - `culaCsyrk`
  - `culaZsyrk`
  - `culaSyrk` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSsyrk`
  - `culaDeviceDsyrk`
  - `culaDeviceCsyrk`
  - `culaDeviceZsyrk`
  - `culaDeviceSyrk` (C++ style, type overloaded)

Description

SYRK performs one of the symmetric rank k operations

C := alpha*A*A^T + beta*C,

or

C := alpha*A^T * A + beta*C,

where alpha and beta are scalars, C is an n by n symmetric matrix and A is an n by k matrix in the first case and a k by n matrix in the second case.

Parameters

- **uplo**
  - Type: `char`.
  - Direction: `Input`

  On entry, UPLO specifies whether the upper or lower triangular part of the array C is to be referenced as follows:

  UPLO = 'U' or 'u' Only the upper triangular part of C is to be referenced.

  UPLO = 'L' or 'l' Only the lower triangular part of C is to be referenced.
- **trans**
  - Type: `char`.
  - Direction: `Input`

  On entry, TRANS specifies the operation to be performed as follows:

  TRANS = 'N' or 'n' C := alpha*A*A^T + beta*C.

  TRANS = 'T' or 't' C := alpha*A^T * A + beta*C.

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

  On entry, N specifies the order of the matrix C. N must be at least zero.

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

  On entry with TRANS = 'N' or 'n', K specifies the number of columns of the matrix A, and on entry with TRANS = 'T' or 't', K specifies the number of rows of the matrix A. K must be at least zero.

- **alpha**
  - Type: `S/D/C/Z`
  - Direction: `Input`

  On entry, ALPHA specifies the scalar alpha.

- **a**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(N,K)` if TRANS = 'N'
  - Dimension: `(K,N)` if TRANS != 'N'

  Before entry with TRANS = 'N' or 'n', the leading n by k part of the array A must contain the matrix A, otherwise the leading k by n part of the array A must contain the matrix A.

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

  On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANS = 'N' or 'n' then LDA must be at least max(1, n), otherwise LDA must be at least max(1, k).

- **beta**
  - Type: `S/D/C/Z`
  - Direction: `Input`

  On entry, BETA specifies the scalar beta.

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

  Before entry with UPLO = 'U' or 'u', the leading n by n upper triangular part of the array C must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of the array C is overwritten by the upper triangular part of the updated matrix.

Before entry with UPLO = 'L' or 'l', the leading n by n lower triangular part of the array C must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of the array C is overwritten by the lower triangular part of the updated matrix.

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

  On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, n).

5.9 TRMM

CULA Routines

The TRMM functionality is implemented by the following CULA routines:

- Host Memory
  - `culaStrmm`
  - `culaDtrmm`
  - `culaCtrmm`
  - `culaZtrmm`
  - `culaTrmm` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceStrmm`
  - `culaDeviceDtrmm`
  - `culaDeviceCtrmm`
  - `culaDeviceZtrmm`
  - `culaDeviceTrmm` (C++ style, type overloaded)

Description

TRMM performs one of the matrix-matrix operations

B := alpha*op(A)*B, or B := alpha*B*op(A)

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

op(A) = A or op(A) = A^T or op(A) = conjg(A^T).

Parameters

- **side**
  - Type: `char`.
  - Direction: `Input`

  On entry, SIDE specifies whether op(A) multiplies B from the left or right as follows:

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

SIDE = ‘R’ or ‘r’ B := alpha*B*op(A).

- **uplo**
  - Type: `char`.
  - Direction: `Input`

  On entry, UPLO specifies whether the matrix A is an upper or lower triangular matrix as follows:

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

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

- **transa**
  - Type: `char`.
  - Direction: `Input`

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

  TRANSA = ‘N’ or ‘n’ op(A) = A.

  TRANSA = ‘T’ or ‘t’ op(A) = A^T.

  TRANSA = ‘C’ or ‘c’ op(A) = conjg(A^T).

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

  On entry, DIAG specifies whether or not A is unit triangular as follows:

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

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

- **m**
  - Type: `int`.
  - Direction: `Input`

  On entry, M specifies the number of rows of B. M must be at least zero.

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

  On entry, N specifies the number of columns of B. N must be at least zero.

- **alpha**
  - Type: `S/D/C/Z`
  - Direction: `Input`

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

- **a**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input`

  - Dimension: `(K,K)`

  Before entry with UPLO = 'U' or 'u', the leading k by k upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced.

  Before entry with UPLO = 'L' or 'l', the leading k by k lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced.

  Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.

- **lda**

  - Type: `int`.
  - Direction: `Input`

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

- **b**

  - Type: `S/D/C/Z Pointer`
  - Direction: `Input/Output`
  - Dimension: `(M,N)`

  Before entry, the leading m by n part of the array B must contain the matrix B, and on exit is overwritten by the transformed matrix.

- **ldb**

  - Type: `int`.
  - Direction: `Input`

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

5.10 TRSM

CULA Routines

The TRSM functionality is implemented by the following CULA routines:

- Host Memory
  - `culaStrsm`
  - `culaDtrsm`
  - `culaCtrsm`
  - `culaZtrsm`
  - `culaTrsm` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceStrsm`
  - `culaDeviceDtrsm`
  - `culaDeviceCtrsm`

- `culaDeviceZtrsm`
- `culaDeviceTrsm` (C++ style, type overloaded)

Description

TRSM solves one of the matrix equations

$$op(A) * X = alpha * B,$$

or

$$X * op(A) = alpha * B,$$

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

$$op(A) = A$$

or

$$op(A) = A'$$

or

$$op(A) = conjg(A').$$

The matrix X is overwritten on B.

Parameters

- **side**
  - Type: `char`.
  - Direction: `Input`

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

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

  SIDE = 'R' or 'r' X*op(A) = alpha*B.
- **uplo**
  - Type: `char`.
  - Direction: `Input`

  On entry, UPLO specifies whether the matrix A is an upper or lower triangular matrix as follows:

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

  UPLO = 'L' or 'l' A is a lower triangular matrix.
- **transa**
  - Type: `char`.
  - Direction: `Input`

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

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

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

  TRANSA = 'C' or 'c' op(A) = conjg(A^T).

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

  On entry, DIAG specifies whether or not A is unit triangular as follows:

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

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

- **m**
  - Type: `int`.
  - Direction: `Input`

  On entry, M specifies the number of rows of B. M must be at least zero.

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

  On entry, N specifies the number of columns of B. N must be at least zero.

- **alpha**
  - Type: `S/D/C/Z`
  - Direction: `Input`

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

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

  Before entry with UPLO = 'U' or 'u', the leading k by k upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced.

  Before entry with UPLO = 'L' or 'l', the leading k by k lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced.

  Note that when DIAG = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity.

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

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

- **b**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input/Output`

- Dimension: `(M,N)`

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

- **ldb**
  - Type: `int`.
  - Direction: `Input`

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

CHAPTER

SIX

AUXILIARY ROUTINES

This section documents the Auxiliary routines that CULA provides. These functions are not part of a standardized package, but they augment the functionality that each of the standard routines provide.

6.1 GeConjugate

CULA Routines

The GeConjugate functionality is implemented by the following CULA routines:

- Host Memory
  - `culaCgeConjugate`
  - `culaZgeConjugate`
  - `culaConjugate` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceCgeConjugate`
  - `culaDeviceZgeConjugate`
  - `culaDeviceConjugate` (C++ style, type overloaded)

Description

GeConjugate conjugates a target matrix A such that on exit each element of A has had the operation imag(A) = -imag(A) performed.

Parameters

- **m**
  - Type: `int`.
  - Direction: `Input`

  M specifies the number of rows of A. M must be at least zero.
- **n**
  - Type: `int`.
  - Direction: `Input`

  N specifies the number of columns of A. M must be at least zero.
- **a**

  - Type: `C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(M,N)`

  The m by n matrix A for which each element is conjugated.

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

  The leading dimension of the array A.

6.2 TrConjugate

CULA Routines

The TrConjugate functionality is implemented by the following CULA routines:

- Host Memory
  - `culaCtrConjugate`
  - `culaZtrConjugate`
  - `culaConjugate` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceCtrConjugate`
  - `culaDeviceZtrConjugate`
  - `culaDeviceConjugate` (C++ style, type overloaded)

Description

TrConjugate conjugates a target matrix A such that on exit the upper or lower triangle of A, as specified by uplo, has had the operation imag(A) = -imag(A) performed. The diag flag may be set to specify whether the diagonal is conjugated or not.

Parameters

- **uplo**
  - Type: `int`.
  - Direction: `Input`

  = 'U': A is upper triangular

  = 'L': A is lower triangular part

- **diag**
  - Type: `int`.
  - Direction: `Input`

  Specifies whether elements along the diagonal are conjugated.

  = 'D': A(i,i) are conjugated

  = 'N': A(i,i) are untouched

- **m**
  - Type: `int`.
  - Direction: `Input`

  M specifies the number of rows of A. M must be at least zero.
- **n**
  - Type: `int`.
  - Direction: `Input`

  N specifies the number of columns of A. M must be at least zero.
- **a**
  - Type: `C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(M,N)`

  The m by n matrix A for which each element is conjugated.
- **lda**
  - Type: `int`.
  - Direction: `Input`

  The leading dimension of the array A.

6.3 GeNancheck

CULA Routines

The GeNancheck functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSgeNancheck`
  - `culaDgeNancheck`
  - `culaCgeNancheck`
  - `culaZgeNancheck`
  - `culaGeNancheck` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSgeNancheck`
  - `culaDeviceDgeNancheck`
  - `culaDeviceCgeNancheck`
  - `culaDeviceZgeNancheck`
  - `culaDeviceGeNancheck` (C++ style, type overloaded)

Description

GeNancheck checks a given matrix for invalid values. If one is discovered, culaDataError is returned; otherwise, culaNoError is returned.

Parameters

- **m**
  - Type: `int`.
  - Direction: `Input`

  M specifies the number of rows of A. M must be at least zero.
- **n**
  - Type: `int`.
  - Direction: `Input`

  N specifies the number of columns of A. M must be at least zero.
- **a**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(M,N)`

  The m by n matrix A that is checked for invalid values.
- **lda**
  - Type: `int`.
  - Direction: `Input`

  The leading dimension of the array A.

6.4 GeTranspose

CULA Routines

The GeTranspose functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSgeTranspose`
  - `culaDgeTranspose`
  - `culaCgeTranspose`
  - `culaZgeTranspose`
  - `culaGeTranspose` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSgeTranspose`
  - `culaDeviceDgeTranspose`
  - `culaDeviceCgeTranspose`
  - `culaDeviceZgeTranspose`

- `culaDeviceGeTranspose` (C++ style, type overloaded)

Description

GeTranspose performs an out-of-place transpose of a matrix A into a matrix B so that B = A^T.

GeTranspose does not support an in-place transpose; A and B must be located in different memory. For an in-place transpose on square matrices, see *GeTransposeInplace*.

Parameters

- **m**
  - Type: `int`.
  - Direction: `Input`

  M specifies the number of rows of A. M must be at least zero.
- **n**
  - Type: `int`.
  - Direction: `Input`

  N specifies the number of columns of A. M must be at least zero.
- **a**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(M,N)`

  The m by n matrix A that is to be transposed.
- **lda**
  - Type: `int`.
  - Direction: `Input`

  The leading dimension of the array A.
- **b**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(N,M)`

  The destination matrix B; upon completion, B = A^T.
- **ldb**
  - Type: `int`.
  - Direction: `Input`

  The leading dimension of the array B.

6.5 GeTransposeConjugate

CULA Routines

The GeTransposeConjugate functionality is implemented by the following CULA routines:

- Host Memory
  - `culaCgeTransposeConjugate`
  - `culaZgeTransposeConjugate`
  - `culaGeTransposeConjugate` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceCgeTransposeConjugate`
  - `culaDeviceZgeTransposeConjugate`
  - `culaDeviceGeTransposeConjugate` (C++ style, type overloaded)

Description

GeTransposeConjugate performs a tranpose as *GeTranspose* does, with the addition that each element of B is conjugated so that for all (i,j) B(j,i) = conjg(A(i,j)) .

GeTransposeConjugate does not support an in-place transpose; A and B must be located in different memory. For an in-place transpose on square matrices, see *GeTransposeConjugateInplace*.

Parameters

- **m**
  - Type: `int`.
  - Direction: `Input`

  M specifies the number of rows of A. M must be at least zero.
- **n**
  - Type: `int`.
  - Direction: `Input`

  N specifies the number of columns of A. M must be at least zero.
- **a**
  - Type: `C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(M,N)`

  The m by n matrix A that is to be transposed.
- **lda**
  - Type: `int`.
  - Direction: `Input`

  The leading dimension of the array A.
- **b**
  - Type: `C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(N,M)`

  The destination matrix B; upon completion, $B = A^T$.
- **ldb**

  - Type: `int`.
  - Direction: `Input`

  The leading dimension of the array B.

6.6 GeTransposeInplace

CULA Routines

The GeTransposeInplace functionality is implemented by the following CULA routines:

- Host Memory
  - `culaSgeTransposeInplace`
  - `culaDgeTransposeInplace`
  - `culaCgeTransposeInplace`
  - `culaZgeTransposeInplace`
  - `culaGeTransposeInplace` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceSgeTransposeInplace`
  - `culaDeviceDgeTransposeInplace`
  - `culaDeviceCgeTransposeInplace`
  - `culaDeviceZgeTransposeInplace`
  - `culaDeviceGeTransposeInplace` (C++ style, type overloaded)

Description

GeTransposeInplace performs an in-place transpose of a square matrix A onto itself so that A = A^T.

Parameters

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

  N specifies the size of the square matrix A. N must be at least zero.
- **a**
  - Type: `S/D/C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(N,N)`

  The N by N square matrix A that is to be transposed.
- **lda**
  - Type: `int`.
  - Direction: `Input`

  The leading dimension of the array A.

6.7 GeTransposeConjugateInplace

CULA Routines

The GeTransposeConjugateInplace functionality is implemented by the following CULA routines:

- Host Memory
  - `culaCgeTransposeConjugateInplace`
  - `culaZgeTransposeConjugateInplace`
  - `culaGeTransposeConjugateInplace` (C++ style, type overloaded)
- Device Memory
  - `culaDeviceCgeTransposeConjugateInplace`
  - `culaDeviceZgeTransposeConjugateInplace`
  - `culaDeviceGeTransposeConjugateInplace` (C++ style, type overloaded)

Description

GeTransposeConjugateInplace performs a tranpose as *GeTransposeInplace* does, with the addition that each element of A is conjugated so that for all (i,j) A(j,i) = conjg(A(i,j)) .

Parameters

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

  N specifies the size of the square matrix A. N must be at least zero.
- **a**
  - Type: `C/Z Pointer`
  - Direction: `Input`
  - Dimension: `(N,N)`

  The N by N square matrix A that is to be transposed.
- **lda**
  - Type: `int`.
  - Direction: `Input`

  The leading dimension of the array A.

CHAPTER

SEVEN

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.

7.1 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

EIGHT

COMMON ERRORS

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

8.1 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."

8.2 Data Errors

Several functions report data errors by returning `culaDataError`. These type of errors occur when ill-conditioned or singular matrices are discovered. In most cases, when a data error is discovered, the computation is aborted and the error is reported with a `culaDataError` return code. When this value is returned, further information about this error can be requested by calling *culaGetErrorInfo*, which returns an integer equivalent to that which would be returned by the analogous LAPACK function. Refer to the particular function for a description of what this interger value means.

8.3 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.