Matrix Toolbox Manual

Chapter 3

Library Function Descriptions

Chapter 3 - Library Function Descriptions

A. Introduction

This chapter contains the man3 pages for the function calls in the kmatrix library that is contained in the Matrix toolbox. The man3 pages are alphabetically ordered by function name.

ALL programs that utilize these function calls MUST include the statement:

```
#include <matrix.h>
```

A.1. The kmatrix Library

The kmatrix library (libkmatrix.a) contains matrix operators that are based on the LAPACK library. Function operations include extracting rows, columns, and diagonals, computing eigenvectors and eigenvalues, computing the inverse matrix, computing the least-squares solution to a set of linear equations, LU decomposition, computing the matrix product, and singular value decomposition. All kmatrix functions are written to operate on width-height planes of the polymorphic data model. If the depth, time, or elements dimensions of data object are greater than one, the operation is repeated for each width-height plane. These functions operate only on the value segment, and do not check for any other data segments. The kmatrix operators are implemented using the polymorphic data services, which is a part of the Khoros 2.0 kappserv library.

A.2. LAPACK Libraries & Fortran Dependency

The kmatrix library depends on the LAPACK libraries lzlapack, ldlapack, and lblas, which are included in the Matrix toolbox.

This section is incomplete and will be completed as we learn more about Fortran dependency and portability issues.

A.3. Side Effects

The function calls in kmatrix should have no side effects on the source objects passed into them, and should only affect those segments and attributes of the destination object that the operation *needs* to modify. For example, if a destination object is passed in that already has value, mask, and geometry data, and a multiplication operation is performed on the source value data, the destination mask and geometry data will not be modified, and will remain intact.

Each function call frees all memory that has been allocated within the call, and closes all temporary objects that were opened within the call. The function call will *not* free memory or close objects that were passed into it.

A.4. DISCLAIMER — WARNING

The kmatrix library is in its first prototype stage, therefore, the behavior and API of these functions may change in the future.

A.5. On-Line Access to man3 Pages

Any library function man3 page can be accessed by typing the following command.

```
% kman <function_name>
```

The keywords (-k) option in kman allows the user to provide a keyword string for searching. When invoked with the *-k* option, kman will perform the search and provide a list of the names and short descriptions of functions found by the search.

```
% kman -k "keyword string"
```

B. kmatrix Library Functions

B.1. kroutine Function Calls in the kmatrix Library

B.1.1. lmcovar() — *compute the covariance matrix for a given set of data vectors*

Synopsis

```
int lmcovar(
kobject in_obj,
kobject out_obj)
```

Input Arguments

`in_obj`
input data object

Output Arguments

`out_obj`
output covariance matrix object

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

lmcovar is used to compute the covariance matrix for the set of data vectors supplied in the input data object. The covariance matrix is computed for all w*h*d*t data vectors, each of length e. The output will be of type DOUBLE or DCOMPLEX depending on the input data object.

Restrictions

Restrictions on data or input as applicable

B.1.2. lmeigen() — *compute eigenvectors and eigenvalues*

Synopsis

```
int lmeigen(
kobject in_obj,
kobject evec_obj,
kobject eval_obj)
```

Output Arguments

`evec_obj`
matrix of eigenvectors kobject

`eval_obj`
diagonal matrix of eigenvalues

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

lmeigen computes the eigenvectors and eigenvalues of a square matrix. The matrix is NOT required to be symmetric.

All output is in double complex form for generality.

If the eigenvalues are complex, then they will occur in complex conjugate pairs.

The eignenvectors are normalized to have a unit Euclidian norm and a purely real largest component.

lmeigen uses the ZGEEV routine from LAPACK to actually obtain the decomposition.

Restrictions

All output is of type double complex.

B.1.3. lmexchg() — *exchange rows and/or columns of a matrix*

Synopsis

```
int lmexchg( kobject in_obj, int r1, int r2, int c1, int c2, kobject out_obj)
```

Input Arguments

`in_obj`
  input matrix object

`r1`
  index of first row to be exchanged

`r2`
  index of second row to be exchanged

`c1`
  index of first column to be exchanged

`c2`
  index of second column to be exchanged

Output Arguments

`out_obj`
  output matrix object

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

lmexchg can exchange one pair of rows and one pair of columns of a matrix. If only a column exchange is desired, then the indices of the rows to be exchanged be set to the same value (i.e. exchange a row with itself). Likewise if only a row exchange is desired, set the column indices to the same number.

Row exchanges are done before column exchanges.

The exchange operation will take place across all planes of (DxTxE) data. The actual data is copied as (for a row exchange) a WxDxTxE hyperplane. It is implicitly assumed that this hyperplane will easily fit in memory. Columns are copied as HxDxTxE hyperplanes under the same assumption.

Restrictions

Restrictions on data or input as applicable

B.1.4. lmextract_diag() — *extract row, column, or diagonal from a matrix*

Synopsis

```
int lmextract_diag(
kobject in_obj,
int row_flag,
int col_flag,
int diag_flag,
int num,
kobject out_obj)
```

Input Arguments

`in_obj`
input matrix object int

`row_flag`
if non-zero, extract specified row int

`col_flag`
if non-zero, extract specified column int

`diag_flag`
if non-zero, extract diagonal int

`num`
number of row or column to extract

Output Arguments

`out_obj`
output object

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

lmextract_diag is used to extract a row, column or the diagonal from a matrix.

B.1.5. lmgdiag() — *generate a diagonal matrix*

Synopsis

```
int
lmgdiag(
kobject vec_obj,
int r,
int c,
```

```
double cr,
double ci,
int type,
kobject out_obj)
```

Input Arguments

`vec_obj`
(kobject) vector of diagonal values

`r`
number of rows in diagonal matrix

`c`
number of columns in diagonal matrix

`cr`
real part of constant diagonal value

`ci`
imaginary part of constant diagonal value

`type`
data type for matrix (FLOAT,DOUBLE,COMPLEX,DCOMPLEX)

Output Arguments

`out_obj`
output matrix object

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

Generate a diagonal matrix (possible non-square) of a specified data type and with either a constant diagonal value (vec_obj == NULL) or non-constant values (vec_obj != NULL). Matrix size is specified by r and c. A non-NULL vec_obj overrides any values for cr and ci.

B.1.6. lminvert() — *compute inverse of matrix*

Synopsis

```
int lminvert(
kobject in_obj,
int diag,
kobject out_obj)
```

Input Arguments

`in_obj`
input matrix object int

`diag`
request inversion of diagonal only if non-zero

Output Arguments

`out_obj`

output matrix object for inverse

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

lminvert computes the inverse of matrix. If diagonal only is requested, compute 1/A(i,i); else matrix must be square and LU decomposition with backsolve is used to obtain the matrix inverse.

B.1.7. lmlse() — *compute least-squares solution to a set of linear equations*

Synopsis

```
int lmlse (
kobject A_obj,
kobject B_obj,
kobject X_obj)
```

Input Arguments

`A_obj`

A matrix kobject

`B_obj`

set of b vectors (col vectors) for RHS

Output Arguments

`X_obj`

set of x vectors (col vectors for solution

Returns

TRUE (1) on success, FALSE (0) on failure

Description

Library Routine for mlse - this routine employs the LAPACK DGELS or ZGELS driver to obtain the least-squares solution to a set of linear equations AX=B where B is made up of column vectors, each of which is assumed to be an independent right-hand-side, and X is made up of column vectors, each of which is an independent solution to the corresponding X column vector.

For an MxN matrix A, if M>=N then the solution is obtained using the QR factorization; if M<N, then the solution is obtained using the LQ factorization.

B.1.8. lmlud() — *compute LU decomposition*

Synopsis

```
int lmlud (
     kobject in_obj,
     kobject lt_obj,
     kobject ut_obj,
     kobject pi_obj)
```

Input Arguments

`in_obj`
: matrix to be factored

Output Arguments

`lt_obj`
: object to receive the lower triangular factor kobject

`ut_obj`
: object to receive the upper triangular factor kobject

`pi_obj`
: object to receive the pivot vector

Returns

TRUE (1) on success, FALSE (0) on failure

Description

lmlud computes LU decomposition of a general matrix using partial pivoting and row interchanges.

The L matrix is lower triangular with unit diagonal elements. The U matrix is upper triangular. The pivot vector indicates that the i'th row was interchanged with the row given in the i'th value in the vector.

lmlud used the DGETRF or ZGETRF routines from LAPACK to perform the decomposition.

B.1.9. lmmul() — *form matrix product C=AB*

Synopsis

```
int lmmul( kobject a_obj, kobject b_obj, kobject c_obj)
```

Input Arguments

`a_obj`
: A matrix object kobject

`b_obj`
B matrix object

Output Arguments
`c_obj`
C matrix object

Returns
TRUE (1) on success, FALSE (0) otherwise

Description
lmmul forms the matrix product C=AB where A is MxN, B is NxP, and C is MxP.

B.1.10. lmpow() — *compute positive integral power of a matrix*

Synopsis
```
int lmpow(kobject in_obj, int pow, kobject out_obj)
```

Input Arguments
`in_obj`
input matrix object, must be square
`pow`
non-negative integer power

Output Arguments
`out_obj`
output matrix object

Returns
TRUE (1) on success, FALSE (0) otherwise

Description
lmpow raises a square input matrix to a non-negative integral power by multiplying the matrix with itself. A power of zero yields the unit matrix.

Restrictions
Restrictions on data or input as applicable

B.1.11. lmrcsum() — *compute row and column sums for a matrix*

Synopsis

```
int lmrcsum(
kobject in_obj,
int scale_type,
kobject rowsum_obj,
kobject colsum_obj)
```

Input Arguments

`in_obj`
: input matrix object integer

`scale_type`
: 1 for scaling by unity, 2 for scaling by 1/N, 3 for scaling by 1/(N-1), any other value is treated as scaling by unity.

Output Arguments

`rowsum_obj`
: output object to receive the row sum vectors (NULL if not wanted). kobject

`colsum_obj`
: output object to receive the column sum vectors (NULL if not wanted).

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

lmrcsum conputes the row and column sums for a matrix object. If the rowsum object pointer is NULL, then the row sums are not computed, and likewise for the colsum object pointer.

Restrictions

Restrictions on data or input as applicable

B.1.12. lmreplicate() — *replicate a given matrix as submatrices of a larger matrix*

Synopsis

```
int lmreplicate(
kobject in_obj,
int r,
int c,
kobject out_obj)
```

Input Arguments

`in_obj`
: input object

`r`
: replication count along row direction

`c`
: replication count along column direction

Output Arguments

`out_obj`
: output matrix

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

lmreplicate is used to replicate an input matrix in a rectagular array, forming a larger output matrix. The input matrix thus appears as submatrices of the output matrix.

B.1.13. lmsvd() — *compute singular value decomposition (SVD)*

Synopsis

```
int lmsvd(
kobject in_obj,
kobject u_obj,
kobject s_obj,
kobject v_obj)
```

Input Arguments

`in_obj`
: input matrix object

Output Arguments

`u_obj`
: matrix object to receive the U matrix kobject

`s_obj`
: matrix object to receive the S matrix kobject

`v_obj`
: matrix object to receive the V matrix

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

lmsvd computes the singular value decomposition of a general matrix, A=USVt, where U and V are orthogonal/unitary and S is diagonal, real, and non-negative. Here, Vt means transpose(V).

For MxN matrix A, U is MxM, S is MxN, and V is NxN where the columns of U and V contain the left and right singular vectors respectively and the diagonal elements of S are the singular values in descending sorted order.

lmsvd uses the DGESVD and ZGESVD routines from LAPACK to obtain the decomposition.

Table of Contents

This page left intentionally blank