MATLAB C Math Library

The Language of Technical Computing

Computation

Visualization

Programming

The MATH WORKS Inc.

Reference

Version 2.1

How to Contact The MathWorks:

508-647-7000 Phone

508-647-7001 Fax

The MathWorks, Inc. Mail
3 Apple Hill Drive
Natick, MA 01760-2098

`http://www.mathworks.com` Web
`ftp.mathworks.com` Anonymous FTP server
`comp.soft-sys.matlab` Newsgroup

`support@mathworks.com` Technical support
`suggest@mathworks.com` Product enhancement suggestions
`bugs@mathworks.com` Bug reports
`doc@mathworks.com` Documentation error reports
`subscribe@mathworks.com` Subscribing user registration
`service@mathworks.com` Order status, license renewals, passcodes
`info@mathworks.com` Sales, pricing, and general information

MATLAB C Math Library Reference

Printing History: January 1998 Version 1.2
January 1999 Revised for Version 2.0 (Release 11) Online only
September 2000 Revised for Version 2.1 (Release 12) Online only

Using the C Math Library Function Reference

This reference gives you quick access to the prototypes and call syntax for the MATLAB C Math Library functions. The functions fall into two groups: the mathematical functions and the utility functions. This section discusses the organization of the reference pages.

Refer to the online *Application Program Interface Reference* for documentation of the `mx` routines that let you create, access, and delete arrays.

Reference Page Format

Use the reference pages to look up the prototype and syntax for a MATLAB C Math Library function. At the bottom of each page, you'll find a link to the documentation for the MATLAB version of the function. Use the MATLAB function page to look up the description of the arguments and the behavior of the function.

Structure

A reference page for a MATLAB C Math Library function includes these sections:

- Purpose
- C Prototype
- C Syntax
- MATLAB Syntax
- See Also links to the MATLAB version of the function and to the calling conventions

One C prototype represents the MATLAB syntax.

To make the reference pages easier to read:

- The variable names that appear in the "MATLAB Syntax" section are used as parameter names in the prototype for a function.
- The first call to a function listed under "C Syntax" corresponds to the first call listed under "MATLAB Syntax." The second C call corresponds to the second MATLAB call, and so forth.

The “C Syntax” section shows only the calls supported by the library. When you link to the MATLAB version of the function, you may notice MATLAB syntax that supports objects. Because this version of the MATLAB C Math Library does not support objects, that documentation does not apply to the C version of the function.

Typographic Conventions

- String arrays, including those that represent a function name, are italicized to indicate that you must assign a value to the variable.
- In general, a lowercase variable name/argument indicates a vector.
- In general, an uppercase variable name/argument indicates a matrix.
- Assignments to input arguments
- Calls to `mlfEnterNewContext()` and `mlfRestorePreviousContext()`
- Deletion of allocated arrays (Calls to `mxDestroyArray()` are not shown.)

Notes on the Format

- Assignments to input arguments are not shown
- Calls to `mlfEnterNewContext()` and `mlfRestorePreviousContext()` are not shown.
- Deletion of allocated arrays, using `mxDestroyArray()`, are not shown.
- Occasionally, you’ll find a C prototype where the parameter names do not match those used in the “MATLAB Syntax” section. The correspondence between the arguments used to call the function and the C prototype is too varied to represent in one prototype. `O1`, `O2`, etc., and `I1`, `I2`, etc., substitute for the output argument and input argument names in the prototype.
- Occasionally, a call to `mxCreateString()` returns a string array that is used as an input argument; a call to `mlfScalar()` returns an integer array; a call to `mlfHorzcat()` returns a vector.

C Math Library Calling Conventions

This section demonstrates the calling conventions that apply to the MATLAB C Math Library functions, including what data type to use for C input and output arguments, how to handle optional arguments, and how to handle MATLAB's multiple output values in C.

Refer to the "How to Call MATLAB Functions" section of Chapter 6 in the *MATLAB C Math Library User's Guide* for further discussion of the calling conventions and for a list of exceptions to the calling conventions.

Constructing a C Prototype

One C prototype supports all the possible ways to call a particular MATLAB C Math Library function. You can reconstruct the C prototype by examining the MATLAB syntax for a function.

For example, the MATLAB function `svd()` has the following syntax:

```
s = svd(X)
[U,S,V] = svd(X)
[U,S,V] = svd(X,0)
```

To construct the C prototype for `mlfSvd()`, follow this procedure.

1. Find the MATLAB syntax that includes the largest number of return values and input arguments.

   ```
   [U,S,V] = svd(X,0)
   ```

2. Use the first return value, `U`, as the return value from the C version of the function. (C routines can only have one return value.) The data type for the return value is `mxArray *`.

3. Add the remaining return values, `S` and `V`, as output arguments to the C function. Output arguments appear before any input arguments in the argument list. The second return value becomes the first output argument, as so on. The data type for the C output arguments is `mxArray **`.

4. Specify the maximum number of input arguments in the C routine prototype. The data type for input arguments is `mxArray *`.

Here is the complete C prototype for the svd function. Compare it to the original MATLAB syntax.

```
mxArray *mlfSvd(mxArray **S, mxArray **V, mxArray *X,
                mxArray *Zero);
```

Note Contrast the data type for an output argument with the data type for an input argument. The type for an output argument is the address of a pointer to an mxArray. The type for an input argument is a pointer to an mxArray.

Translating MATLAB Syntax into C Syntax

This procedure demonstrates how to translate the MATLAB svd() calls into MATLAB C Math Library calls to mlfSvd(). The procedure applies to library functions in general.

In this procedure, mlfAssign(), rather than the assignment operator (=), assigns the return value from mlfSvd() to an array variable. This usage indicates that the automated memory management provided by the library is in effect.

Note that within a call to a MATLAB C Math Library function, an output argument is preceded by &; an input argument is not.

MATLAB Syntax

```
s = svd(X)
[U,S,V] = svd(X)
[U,S,V] = svd(X,0)
```

The MATLAB arguments to svd() fall into these categories:

U (or s) is a required output argument.

S and V are optional output arguments.

X is a required input argument.

0 is an optional input argument.

1. Declare input, output, and return variables as `mxArray *` variables. Assign values to the input variables. Initialize output and return variables to `NULL`.

2. Make the first output argument the return value from the function.

   ```
   mlfAssign(&s,
   mlfAssign(&U,
   mlfAssign(&U,
   ```

3. Pass any additional required or optional output arguments as the first arguments to the function. Pass a `NULL` argument wherever an optional output argument does not apply to the particular call.

   ```
   mlfAssign(&s, mlfSvd(NULL,NULL,
   mlfAssign(&U, mlfSvd(&S,&V,
   mlfAssign(&U, mlfSvd(&S,&V,
   ```

   ```
   s = mlfSvd(NULL,NULL,
   U = mlfSvd(&S,&V,
   U = mlfSvd(&S,&V,
   ```

4. Pass any required or optional input arguments that apply to the C function, following the output arguments. Pass a `NULL` argument wherever an optional input argument does not apply to the particular call.

   ```
   mlfAssign(&s, mlfSvd(NULL,NULL,X,NULL));
   mlfAssign(&U, mlfSvd(&S,&V,X,NULL));
   mlfAssign(&U, mlfSvd(&S,&V,X,Zero));
   ```

Note `NULL` arguments always follow significant arguments; a `NULL` argument cannot appear between two output arguments or between two input arguments. Move the significant output or input argument before the `NULL` argument.

Function Reference

This section contains an alphabetical listing of the routines in the MATLAB C Math Library.

Note For information about the MATLAB C Math Library utility routines, see “Utility Routine Reference” on page -382. These routines appear in a separate alphabetical listing.

Purpose

Matrix and array arithmetic

C Prototype

```
/* Matrix Arithmetic */
mxArray *mlfPlus(mxArray *A, mxArray *B);
mxArray *mlfMinus(mxArray *A, mxArray *B);
mxArray *mlfUnaryminus(mxArray *A);
mxArray *mlfUminus(mxArray *A);
mxArray *mlfMtimes(mxArray *A, mxArray *B);
mxArray *mlfMrdivide(mxArray *A, mxArray *B);
mxArray *mlfMldivide(mxArray *A, mxArray *B);
mxArray *mlfMpower(mxArray *A, mxArray *B);
mxArray *mlfCtranspose(mxArray *A);

/* Array Arithmetic */
mxArray *mlfTimes(mxArray *A, mxArray *B);
mxArray *mlfRdivide(mxArray *A, mxArray *B);
mxArray *mlfLdivide(mxArray *A, mxArray *B);
mxArray *mlfPower(mxArray *A, mxArray *B);
mxArray *mlfTranspose(mxArray *A);
```

Arithmetic Operators

C Syntax

```
#include "matlab.h"

mxArray *A, *B;             /* Input arguments */
mxArray *C = NULL;          /* Return value */

/* Matrix Arithmetic */
mlfAssign(&C, mlfPlus(A,B));
mlfAssign(&C, mlfMinus(A,B));
mlfAssign(&C, mlfUnaryminus(A));
mlfAssign(&C, mlfUminus(A));
mlfAssign(&C, mlfMtimes(A,B));
mlfAssign(&C, mlfMrdivide(A,B));
mlfAssign(&C, mlfMldivide(A,B));
mlfAssign(&C, mlfMpower(A,B));
mlfAssign(&C, mlfCtranspose(A));

/* Array Arithmetic */
mlfAssign(&C, mlfTimes(A,B));
mlfAssign(&C, mlfRdivide(A,B));
mlfAssign(&C, mlfLdivide(A,B));
mlfAssign(&C, mlfPower(A,B));
mlfAssign(&C, mlfTranspose(A));
```

MATLAB Syntax

```
A+B
A-B
A*B     A.*B
A/B     A./B
A\B     A.\B
A^B     A.^B
A'      A.'
```

See Also

MATLAB Arithmetic Operators Calling Conventions

Purpose

Relational operations

C Prototype

```
mxArray *mlfLt(mxArray *A, mxArray *B);
mxArray *mlfGt(mxArray *A, mxArray *B);
mxArray *mlfLe(mxArray *A, mxArray *B);
mxArray *mlfGe(mxArray *A, mxArray *B);
mxArray *mlfEq(mxArray *A, mxArray *B);
mxArray *mlfNe(mxArray *A, mxArray *B);
mxArray *mlfNeq(mxArray *A, mxArray *B);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *B;             /* Input arguments */
mxArray *C = NULL;          /* Return value */

mlfAssign(&C, mlfLt(A,B));
mlfAssign(&C, mlfGt(A,B));
mlfAssign(&C, mlfLe(A,B));
mlfAssign(&C, mlfGe(A,B));
mlfAssign(&C, mlfEq(A,B));
mlfAssign(&C, mlfNe(A,B));
mlfAssign(&C, mlfNeq(A,B));
```

MATLAB Syntax

```
A < B
A > B
A <= B
A >= B
A == B
A ~= B
```

See Also

MATLAB Relational Operators Calling Conventions

Logical Operators

Purpose

Logical operations

C Prototype

```
mxArray *mlfAnd(mxArray *A, mxArray *B);
mxArray *mlfOr(mxArray *A, mxArray *B);
mxArray *mlfNot(mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *B;             /* Input arguments */
mxArray *C = NULL;          /* Return value */

mlfAssign(&C, mlfAnd(A,B));
mlfAssign(&C, mlfOr(A,B));
mlfAssign(&C, mlfNot(A));
```

MATLAB Syntax

```
A & B
A | B
~A
```

See Also

MATLAB Logical Operators Calling Conventions

mlfAbs

Purpose

Absolute value and complex magnitude

C Prototype

```
mxArray *mlfAbs(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfAbs(X));
```

MATLAB Syntax

```
Y = abs(X)
```

See Also

MATLAB `abs` Calling Conventions

Purpose

Inverse cosine and inverse hyperbolic cosine

C Prototype

```
mxArray *mlfAcos(mxArray *X);
mxArray *mlfAcosh(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfAcos(X));
mlfAssign(&Y, mlfAcosh(X));
```

MATLAB Syntax

```
Y = acos(X)
Y = acosh(X)
```

See Also

MATLAB acos, acosh Calling Conventions

mlfAcot, mlfAcoth

Purpose

Inverse cotangent and inverse hyperbolic cotangent

C Prototype

```
mxArray *mlfAcot(mxArray *X);
mxArray *mlfAcoth(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfAcot(X));
mlfAssign(&Y, mlfAcoth(X));
```

MATLAB Syntax

```
Y = acot(X)
Y = acoth(X)
```

See Also

MATLAB acot, acoth Calling Conventions

Purpose

Inverse cosecant and inverse hyperbolic cosecant

C Prototype

```
mxArray *mlfAcsc(mxArray *X);
mxArray *mlfAcsch(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfAcsc(X));
mlfAssign(&Y, mlfAcsch(X));
```

MATLAB Syntax

```
Y = acsc(X)
Y = acsch(X)
```

See Also

MATLAB acsc, acsch Calling Conventions

mlfAll

Purpose

Test to determine if all elements are nonzero

C Prototype

```
mxArray *mlfAll(mxArray *A, mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *dim;           /* Optional input argument(s) */
mxArray *B = NULL;      /* Return value */

mlfAssign(&B, mlfAll(A,NULL));
mlfAssign(&B, mlfAll(A,dim));
```

MATLAB Syntax

```
B = all(A)
B = all(A,dim)
```

See Also

MATLAB `all` Calling Conventions

Purpose

Phase angle

C Prototype

```
mxArray *mlfAngle(mxArray *Z);
```

C Syntax

```
#include "matlab.h"

mxArray *Z;             /* Required input argument(s) */
mxArray *P = NULL;      /* Return value */

mlfAssign(&P, mlfAngle(Z));
```

MATLAB Syntax

```
P = angle(Z)
```

See Also

MATLAB `angle` Calling Conventions

mlfAny

Purpose

Test for any nonzeros

C Prototype

```
mxArray *mlfAny(mxArray *A, mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *dim;           /* Optional input argument(s) */
mxArray *B = NULL;      /* Return value */

mlfAssign(&B, mlfAny(A,NULL));
mlfAssign(&B, mlfAny(A,dim));
```

MATLAB Syntax

```
B = any(A)
B = any(A,dim)
```

See Also

MATLAB any Calling Conventions

Purpose

Inverse secant and inverse hyperbolic secant

C Prototype

```
mxArray *mlfAsec(mxArray *X);
mxArray *mlfAsech(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfAsec(X));
mlfAssign(&Y, mlfAsech(X));
```

MATLAB Syntax

```
Y = asec(X)
Y = asech(X)
```

See Also

MATLAB asec, asech Calling Conventions

mlfAsin, mlfAsinh

Purpose

Inverse sine and inverse hyperbolic sine

C Prototype

```
mxArray *mlfAsin(mxArray *X);
mxArray *mlfAsinh(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfAsin(X));
mlfAssign(&Y, mlfAsinh(X));
```

MATLAB Syntax

```
Y = asin(X)
Y = asinh(X)
```

See Also

MATLAB `asin`, `asinh` Calling Conventions

Purpose

Inverse tangent and inverse hyperbolic tangent

C Prototype

```
mxArray *mlfAtan(mxArray *X);
mxArray *mlfAtanh(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfAtan(X));
mlfAssign(&Y, mlfAtanh(X));
```

MATLAB Syntax

```
Y = atan(X)
Y = atanh(X)
```

See Also

MATLAB `atan`, `atanh` Calling Conventions

mlfAtan2

Purpose

Four-quadrant inverse tangent

C Prototype

```
mxArray *mlfAtan2(mxArray *Y, mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *Y, *X;         /* Required input argument(s) */
mxArray *P = NULL;      /* Return value */

mlfAssign(&P, mlfAtan2(Y,X));
```

MATLAB Syntax

```
P = atan2(Y,X)
```

See Also

MATLAB `atan` Calling Conventions

mlfBalance

Purpose

Improve accuracy of computed eigenvalues

C Prototype

```
mxArray *mlfBalance(mxArray **B, mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *B = NULL;      /* Optional output argument or return */
mxArray *D = NULL;      /* Return value */

mlfAssign(&D, mlfBalance(&B,A));
mlfAssign(&B, mlfBalance(NULL,A));
```

MATLAB Syntax

```
[D,B] = balance(A)
B = balance(A)
```

See Also

MATLAB `balance` Calling Conventions

Purpose Base to decimal number conversion

C Prototype

```
mxArray *mlfBase2dec(mxArray *str, mxArray *base);
```

C Syntax

```
#include "matlab.h"

mxArray *str;               /* String array(s) */
mxArray *base;              /* Required input argument(s) */
mxArray *d = NULL;          /* Return value */

mlfAssign(&d, mlfBase2dec(str,base));
```

MATLAB Syntax

```
d = base2dec('strn',base)
```

See Also MATLAB `base2dec` Calling Conventions

mlfBeta, mlfBetainc, mlfBetaln

Purpose

Beta functions

C Prototype

```
mxArray *mlfBeta(mxArray *Z, mxArray *W);
mxArray *mlfBetainc(mxArray *X, mxArray *Z, mxArray *W);
mxArray *mlfBetaln(mxArray *Z, mxArray *W);
```

C Syntax

```
#include "matlab.h"

mxArray *Z, *W, *X;             /* Required input argument(s) */
mxArray *B = NULL, *I = NULL, *L = NULL;/* Return value */

mlfAssign(&B, mlfBeta(Z,W));
mlfAssign(&I, mlfBetainc(X,Z,W));
mlfAssign(&L, mlfBetaln(Z,W));
```

MATLAB Syntax

```
B = beta(Z,W)
I = betainc(X,Z,W)
L = betaln(Z,W)
```

See Also

MATLAB `beta`, `betainc`, `betaln` Calling Conventions

Description

`B = beta(Z,W)` computes the beta function for corresponding elements of the complex arrays `Z` and `W`. The arrays must be the same size (or either can be scalar).

`I = betainc(X,Z,W)` computes the incomplete beta function. The elements of `X` must be in the closed interval [0,1].

`L = betaln(Z,W)` computes the natural logarithm of the beta function, `log(beta(Z,W))`, without computing `beta(Z,W)`. Since the `beta` function can range over very large or very small values, its logarithm is sometimes more useful.

Purpose

BiConjugate Gradients method

Minimum number of arguments: eleven, maximum: user-defined. Terminate the argument list with a `NULL`.

C Prototype

```
mxArray *mlfBicg(mxArray **flag,
                 mxArray **relres,
                 mxArray **iter,
                 mxArray **resvec,
                 mxArray *A,
                 mxArray *b,
                 mxArray *tol,
                 mxArray *maxit,
                 mxArray *M1,
                 mxArray *M2,
                 mxArray *x0,
                 ...);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *b;                 /* Required input argument(s) */
mxArray *tol, *maxit;           /* Optional input argument(s) */
mxArray *M, *M1, *M2, *x0;      /* Optional input argument(s) */
mxArray *flag=NULL,*relres=NULL;/* Optional output argument(s) */
mxArray *iter=NULL,*resvec=NULL;/* Optional output argument(s) */
mxArray *x = NULL;              /* Return value */

mlfAssign(&x, mlfBicg(NULL,NULL,NULL,NULL,
                      A,b,NULL,NULL,NULL,NULL,NULL,NULL));
mlfAssign(&x, mlfBicg(NULL,NULL,NULL,NULL,
                      A,b,tol,NULL,NULL,NULL,NULL,NULL));
mlfAssign(&x, mlfBicg(NULL,NULL,NULL,NULL,
                      A,b,tol,maxit,NULL,NULL,NULL,NULL));
mlfAssign(&x, mlfBicg(NULL,NULL,NULL,NULL,
                      A,b,tol,maxit,M,NULL,NULL,NULL));
mlfAssign(&x, mlfBicg(NULL,NULL,NULL,NULL,
                      A,b,tol,maxit,M1,M2,NULL,NULL));
mlfAssign(&x, mlfBicg(NULL,NULL,NULL,NULL,
                      A,b,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfBicg(&flag,NULL,NULL,NULL,
                      A,b,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfBicg(&flag,&relres,NULL,NULL,
                      A,b,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfBicg(&flag,&relres,&iter,NULL,
                      A,b,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfBicg(&flag,&relres,&iter,&resvec,
                      A,b,tol,maxit,M1,M2,x0,NULL));
```

MATLAB Syntax

```
x = bicg(A,b)
bicg(A,b,tol)
bicg(A,b,tol,maxit)
bicg(A,b,tol,maxit,M)
bicg(A,b,tol,maxit,M1,M2)
bicg(A,b,tol,maxit,M1,M2,x0)
[x,flag] = bicg(A,b,tol,maxit,M1,M2,x0)
[x,flag,relres] = bicg(A,b,tol,maxit,M1,M2,x0)
[x,flag,relres,iter] = bicg(A,b,tol,maxit,M1,M2,x0)
[x,flag,relres,iter,resvec] = bicg(A,b,tol,maxit,M1,M2,x0)
```

See Also MATLAB `bicg` Calling Conventions

mlfBicgstab

Purpose

BiConjugate Gradients Stabilized method

Minimum number of arguments: eleven, maximum: user-defined. Terminate the argument list with a NULL.

C Prototype

```
mxArray *mlfBicgstab(mxArray **flag,
                     mxArray **relres,
                     mxArray **iter,
                     mxArray **resvec,
                     mxArray *A,
                     mxArray *b,
                     mxArray *tol,
                     mxArray *maxit,
                     mxArray *M1,
                     mxArray *M2,
                     mxArray *x0,
                     ...);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *b;                 /* Required input argument(s) */
mxArray *tol, *maxit;           /* Optional input argument(s) */
mxArray *M, *M1, *M2, *x0;      /* Optional input argument(s) */
mxArray *flag=NULL,*relres=NULL;/* Optional output argument(s) */
mxArray *iter=NULL,*resvec=NULL;/* Optional output argument(s) */
mxArray *x = NULL;              /* Return value */

mlfAssign(&x, mlfBicgstab(NULL,NULL,NULL,NULL,
                          A,b,NULL,NULL,NULL,NULL,NULL,NULL));
mlfAssign(&x, mlfBicgstab(NULL,NULL,NULL,NULL,
                          A,b,tol,NULL,NULL,NULL,NULL,NULL));
mlfAssign(&x, mlfBicgstab(NULL,NULL,NULL,NULL,
                          A,b,tol,maxit,NULL,NULL,NULL,NULL));
mlfAssign(&x, mlfBicgstab(NULL,NULL,NULL,NULL,
                          A,b,tol,maxit,M,NULL,NULL,NULL));
mlfAssign(&x, mlfBicgstab(NULL,NULL,NULL,NULL,
                          A,b,tol,maxit,M1,M2,NULL,NULL));
mlfAssign(&x, mlfBicgstab(NULL,NULL,NULL,NULL,
                          A,b,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfBicgstab(NULL,NULL,NULL,NULL,
                           A,b,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfBicgstab(&flag,NULL,NULL,NULL,
                           A,b,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfBicgstab(&flag,&relres,NULL,NULL,
                           A,b,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfBicgstab(&flag,&relres,&iter,NULL,
                           A,b,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfBicgstab(&flag,&relres,&iter,&resvec,
                           A,b,tol,maxit,M1,M2,x0,NULL));
```

mlfBicgstab

MATLAB Syntax

```
x = bicgstab(A,b)
bicgstab(A,b,tol)
bicgstab(A,b,tol,maxit)
bicgstab(A,b,tol,maxit,M)
bicgstab(A,b,tol,maxit,M1,M2)
bicgstab(A,b,tol,maxit,M1,M2,x0)
x = bicgstab(A,b,tol,maxit,M1,M2,x0)
[x,flag] = bicgstab(A,b,tol,maxit,M1,M2,x0)
[x,flag,relres] = bicgstab(A,b,tol,maxit,M1,M2,x0)
[x,flag,relres,iter] = bicgstab(A,b,tol,maxit,M1,M2,x0)
[x,flag,relres,iter,resvec] = bicgstab(A,b,tol,maxit,M1,M2,x0)
```

See Also MATLAB `bicgstab` Calling Conventions

Purpose

Binary to decimal number conversion

C Prototype

```
mxArray *mlfBin2dec(mxArray *binarystr);
```

C Syntax

```
#include "matlab.h"

mxArray *binarystr;         /* Required input argument(s) */
mxArray *decnumber = NULL;  /* Return value */

mlfAssign(&decnumber, mlfBin2dec(binarystr));
```

MATLAB Syntax

`bin2dec(`*`binarystr`*`)`

See Also

MATLAB `bin2dec` Calling Conventions

mlfBitand

Purpose

Bit-wise AND

C Prototype

```
mxArray *mlfBitand(mxArray *A, mxArray *B);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *B;             /* Required input argument(s) */
mxArray *C = NULL;          /* Return value */

mlfAssign(&C, mlfBitand(A,B));
```

MATLAB Syntax

```
C = bitand(A,B)
```

See Also

MATLAB `bitand` Calling Conventions

Purpose

Complement bits

C Prototype

```
mxArray *mlfBitcmp(mxArray *A, mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *n;             /* Required input argument(s) */
mxArray *C = NULL;          /* Return value */

mlfAssign(&C, mlfBitcmp(A,n));
```

MATLAB Syntax

```
C = bitcmp(A,n)
```

See Also

MATLAB `bitcmp` Calling Conventions

mlfBitget

Purpose

Get bit

C Prototype

```
mxArray *mlfBitget(mxArray *A, mxArray *bit);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *bit;           /* Required input argument(s) */
mxArray *C = NULL;          /* Return value */

mlfAssign(&C, mlfBitget(A,bit));
```

MATLAB Syntax

```
C = bitget(A,bit)
```

See Also

MATLAB `bitget` Calling Conventions

Purpose Maximum floating-point integer

C Prototype

```
mxArray *mlfBitmax(void);
```

C Syntax

```
#include "matlab.h"

mxArray *C = NULL;          /* Return value */

mlfAssign(&C, mlfBitmax());
```

MATLAB Syntax

```
bitmax
```

See Also MATLAB `bitmax` Calling Conventions

mlfBitor

Purpose

Bit-wise OR

C Prototype

```
mxArray *mlfBitor(mxArray *A, mxArray *B);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *B;             /* Required input argument(s) */
mxArray *C = NULL;          /* Return value */

mlfAssign(&C, mlfBitor(A,B));
```

MATLAB Syntax

```
C = bitor(A,B)
```

See Also

MATLAB `bitor` Calling Conventions

Purpose

Set bit

C Prototype

```
mxArray *mlfBitset(mxArray *A, mxArray *bit, mxArray *v)
```

C Syntax

```
#include "matlab.h"

mxArray *A, *bit;       /* Required input argument(s) */
mxArray *v;             /* Optional input argument(s) */
mxArray *C = NULL;      /* Return value */

mlfAssign(&C, mlfBitset(A,bit,NULL));
mlfAssign(&C, mlfBitset(A,bit,v));
```

MATLAB Syntax

```
C = bitset(A,bit)
C = bitset(A,bit,v)
```

See Also

MATLAB `bitset` Calling Conventions

mlfBitshift

Purpose

Bit-wise shift

C Prototype

```
mxArray *mlfBitshift(mxArray *A, mxArray *k, mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *k;             /* Required input argument(s) */
mxArray *n;                 /* Optional input argument(s) */
mxArray *C = NULL;          /* Return value */

mlfAssign(&C, mlfBitshift(A,n));
mlfAssign(&C, mlfBitshift(A,k,n));
```

MATLAB Syntax

```
C = bitshift(A,n)
C = bitshift(A,k,n)
```

See Also

MATLAB `bitshift` Calling Conventions

Purpose

Bit-wise XOR

C Prototype

```
mxArray *mlfBitxor(mxArray *A, mxArray *B);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *B;             /* Required input argument(s) */
mxArray *C = NULL;          /* Return value */

mlfAssign(&C, mlfBitxor(A,B));
```

MATLAB Syntax

```
C = bitxor(A,B)
```

See Also

MATLAB `bitxor` Calling Conventions

mlfBlanks

Purpose

A string of blanks

C Prototype

```
mxArray *mlfBlanks(mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *n;             /* Required input argument(s) */
mxArray *r = NULL;      /* Return value */

mlfAssign(&r, mlfBlanks(n));
```

MATLAB Syntax

```
blanks(n)
```

See Also

MATLAB `blanks` Calling Conventions

mlfCalendar, mlfVCalendar

Purpose

Calendar

C Prototype

```
mxArray *mlfCalendar(mxArray *y, mxArray *m);
void mlfVCalendar(mxArray *y, mxArray *m);
```

C Syntax

```
#include "matlab.h"

mxArray *d, *y, *m;     /* Input argument(s) */
mxArray *c = NULL;      /* Return value */

mlfAssign(&c, mlfCalendar(NULL,NULL));
mlfAssign(&c, mlfCalendar(d,NULL));
mlfAssign(&c, mlfCalendar(y,m));

mlfVCalendar(NULL,NULL);
mlfVCalendar(d,NULL);
mlfVCalendar(y,m);
```

MATLAB Syntax

```
c = calendar
c = calendar(d)
c = calendar(y,m)
calendar(...)
```

See Also

MATLAB `calendar` Calling Conventions

Purpose

Transform Cartesian coordinates to polar or cylindrical

C Prototype

```
mxArray *mlfCart2pol(mxArray **RHO, mxArray **Z_out, mxArray *X,
                     mxArray *Y, mxArray *Z_in);
```

C Syntax

```
#include "matlab.h"

mxArray *X, *Y;         /* Required input argument(s) */
mxArray *Z_in;          /* Optional input argument(s) */
mxArray *RHO = NULL;    /* Required output argument(s) */
mxArray *Z_out = NULL;  /* Optional output argument(s) */
mxArray *THETA = NULL;  /* Return value */

mlfAssign(&THETA, mlfCart2pol(&RHO,&Z_out,X,Y,Z_in));
mlfAssign(&THETA, mlfCart2pol(&RHO,NULL,X,Y,NULL));
```

MATLAB Syntax

```
[THETA,RHO,Z] = cart2pol(X,Y,Z)
[THETA,RHO] = cart2pol(X,Y)
```

See Also

MATLAB `cart2pol` Calling Conventions

mlfCart2sph

Purpose

Transform Cartesian coordinates to spherical

C Prototype

```
mxArray *mlfCart2sph(mxArray **PHI, mxArray **R, mxArray *X,
                     mxArray *Y, mxArray *Z);
```

C Syntax

```
#include "matlab.h"

mxArray *X, *Y, *Z;             /* Required input argument(s) */
mxArray *PHI = NULL,*R = NULL;  /* Required output argument(s) */
mxArray *THETA = NULL;          /* Return value */

mlfAssign(&THETA, mlfCart2sph(&PHI,&R,X,Y,Z));
```

MATLAB Syntax

```
[THETA,PHI,R] = cart2sph(X,Y,Z)
```

See Also

MATLAB `cart2sph` Calling Conventions

Purpose

Concatenate arrays

Minimum number of arguments: two, maximum: user-defined. Terminate the list of arguments with a `NULL`.

C Prototype

```
mxArray *mlfCat(mxArray *dim, mxArray *A1, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *dim, *A1;          /* Required input argument(s) */
mxArray *A2, *A3, *A4;      /* Optional input argument(s) */
mxArray *C = NULL;          /* Return value */

mlfAssign(&C, mlfCat(dim,A1,A2,NULL));
mlfAssign(&C, mlfCat(dim,A1,A2,A3,A4,...,NULL));
```

MATLAB Syntax

```
C = cat(dim,A,B)
C = cat(dim,A1,A2,A3,A4...)
```

See Also

MATLAB `cat` Calling Conventions

mlfCdf2rdf

Purpose

Convert complex diagonal form to real block diagonal form

C Prototype

```
mxArray *mlfCdf2rdf(mxArray **D_out, mxArray *V_in, mxArray *D_in);
```

C Syntax

```
#include "matlab.h"

mxArray *V_in, *D_in;       /* Required input argument(s) */
mxArray *D_out = NULL;      /* Required output argument(s) */
mxArray *V_out = NULL;      /* Return value */

mlfAssign(&V_out, mlfCdf2rdf(&D_out,V_in,D_in));
```

MATLAB Syntax

```
[V,D] = cdf2rdf(V,D)
```

See Also

MATLAB `cdf2rdf` Calling Conventions

Purpose

Round toward infinity

C Prototype

```
mxArray *mlfCeil(mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *B = NULL;      /* Return value */

mlfAssign(&B, mlfCeil(A));
```

MATLAB Syntax

```
B = ceil(A)
```

See Also

MATLAB `ceil` Calling Conventions

mlfCell

Purpose

Create empty cell array

Minimum number of arguments: one, maximum: user-defined. Terminate the list of arguments with a `NULL`.

C Prototype

```
mxArray *mlfCell(mxArray *in1, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *n;             /* Required input argument(s) */
mxArray *m;             /* Optional input argument(s) */
mxArray *p;             /* Optional input argument(s) */
mxArray *A;             /* Optional input argument(s) */
mxArray *c = NULL;      /* Return value */

mlfAssign(&c, mlfCell(n,NULL));
mlfAssign(&c, mlfCell(m,n,NULL));
mlfAssign(&c, mlfCell(mlfHorzcat(m,n,NULL),NULL));
mlfAssign(&c, mlfCell(m,n,p,...,NULL));
mlfAssign(&c, mlfCell(mlfHorzcat(m,n,p,...,NULL),NULL));
mlfAssign(&c, mlfCell(mlfSize(NULL,A,NULL),NULL));
```

MATLAB Syntax

```
c = cell(n)
c = cell(m,n)
c = cell([m n])
c = cell(m,n,p,...)
c = cell([m n p ...])
c = cell(size(A))
```

See Also

MATLAB `cell` Calling Conventions

Purpose Display cell array contents

C Prototype

```
void mlfCelldisp(mxArray *c, mxArray *s);
```

C Syntax

```
#include "matlab.h"

mxArray *c;                 /* Required input argument(s) */
mxArray *s;                 /* Optional input argument(s) */

mlfCelldisp(c,s);
```

MATLAB Syntax

```
celldisp(c,s)
```

See Also MATLAB `celldisp` Calling Conventions

mlfCell2struct

Purpose

Cell array to structure array conversion

C Prototype

```
mxArray *mlfCell2struct(mxArray *c, mxArray *fields, mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *c, *fields, *dim;  /* Required input argument(s) */
mxArray *s = NULL;          /* Return value */

mlfAssign(&s, mlfCell2struct(c,fields,dim));
```

MATLAB Syntax

```
s = cell2struct(c,fields,dim)
```

See Also

MATLAB `cell2struct` Calling Conventions

Purpose

Apply a function to each element in a cell array

C Prototype

```
mxArray *mlfCellfun(mxArray *fname, mxArray *C, mxArray *k);
```

C Syntax

```
#include "matlab.h"

mxArray *C;                 /* Required input argument(s) */
mxArray *k;                 /* Optional input argument(s) */
mxArray *D = NULL;          /* Return value */

mlfAssign(&D, mlfCellfun(mxCreateString("fname"),C,NULL));
mlfAssign(&D, mlfCellfun(mxCreateString("size"),C,k));
mlfAssign(&D, mlfCellfun('isclass',C,classname)
```

MATLAB Syntax

```
D = cellfun('fname',C)
D = cellfun('size',C,k)
D = cellfun('isclass',C,classname)
```

See Also

MATLAB `cellfun` Calling Conventions

mlfCellhcat

Purpose

Horizontally concatenate cell arrays. Emulates the MATLAB cell array concatenation operator ({ }).

Minimum number of arguments: two, maximum: user-defined. Terminate the list of arguments with a `NULL`.

C Prototype

```
mxArray *mlfCellhcat(mxArray *pa, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *B, *C;         /* Optional input argument(s) */
mxArray *D = NULL;      /* Return value */

mlfAssign(&D, mlfCellhcat(A,NULL));
mlfAssign(&D, mlfCellhcat(A,B,NULL));
mlfAssign(&D, mlfCellhcat(A,B,C,...,NULL));
```

MATLAB Syntax

```
D = { A B };
D = { A B C };
```

See Also

MATLAB Special Characters Calling Conventions

Purpose

Create cell array of strings from character array

C Prototype

```
mxArray *mlfCellstr(mxArray *S)
```

C Syntax

```
#include "matlab.h"

mxArray *S;                 /* Required input argument(s) */
mxArray *c = NULL;          /* Return value */

mlfAssign(&c, mlfCellstr(S));
```

MATLAB Syntax

```
c = cellstr(S)
```

See Also

MATLAB `cellstr` Calling Conventions

mlfCgs

Purpose

Conjugate Gradients Squared method

Minimum number of arguments: eleven, maximum: user-defined. Terminate the argument list with a `NULL`.

C Prototype

```
mxArray *mlfCgs(mxArray **flag,
                mxArray **relres,
                mxArray **iter,
                mxArray **resvec,
                mxArray *A,
                mxArray *b,
                mxArray *tol,
                mxArray *maxit,
                mxArray *M1,
                mxArray *M2,
                mxArray *x0,
                ...);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *b;                 /* Required input argument(s) */
mxArray *tol, *maxit;           /* Optional input argument(s) */
mxArray *M, *M1, *M2, *x0;      /* Optional input argument(s) */
mxArray *flag=NULL,*relres=NULL;/* Optional output argument(s) */
mxArray *iter=NULL,*resvec=NULL;/* Optional output argument(s) */
mxArray *x = NULL;              /* Return value */

mlfAssign(&x, mlfCgs(NULL,NULL,NULL,NULL,
                     A,b,NULL,NULL,NULL,NULL,NULL,NULL));
mlfAssign(&x, mlfCgs(NULL,NULL,NULL,NULL,
                     A,b,tol,NULL,NULL,NULL,NULL,NULL));
mlfAssign(&x, mlfCgs(NULL,NULL,NULL,NULL,
                     A,b,tol,maxit,NULL,NULL,NULL,NULL));
mlfAssign(&x, mlfCgs(NULL,NULL,NULL,NULL,
                     A,b,tol,maxit,M,NULL,NULL,NULL));
mlfAssign(&x, mlfCgs(NULL,NULL,NULL,NULL,
                     A,b,tol,maxit,M1,M2,NULL,NULL));
mlfAssign(&x, mlfCgs(NULL,NULL,NULL,NULL,
                     A,b,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfCgs(&flag,NULL,NULL,NULL,
                     A,b,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfCgs(&flag,&relres,NULL,NULL,
                     A,b,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfCgs(&flag,&relres,&iter,NULL,
                     A,b,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfCgs(&flag,&relres,&iter,&resvec,
                     A,b,tol,maxit,M1,M2,x0,NULL));
```

mlfCgs

MATLAB Syntax

```
x = cgs(A,b)
cgs(A,b,tol)
cgs(A,b,tol,maxit)
cgs(A,b,tol,maxit,M)
cgs(A,b,tol,maxit,M1,M2)
cgs(A,b,tol,maxit,M1,M2,x0)
x = cgs(A,b,tol,maxit,M1,M2,x0)
[x,flag] = cgs(A,b,tol,maxit,M1,M2,x0)
[x,flag,relres] = cgs(A,b,tol,maxit,M1,M2,x0)
[x,flag,relres,iter] = cgs(A,b,tol,maxit,M1,M2,x0)
[x,flag,relres,iter,resvec] = cgs(A,b,tol,maxit,M1,M2,x0)
```

See Also MATLAB cgs Calling Conventions

Purpose

Create character array (string)

Minimum input arguments: one, maximum: user-defined. Terminate the list of arguments with a NULL.

C Prototype

```
mxArray *mlfChar(mxArray *in1, ... );
```

C Syntax

```
#include "matlab.h"

mxArray *X, *C, *t1;    /* Required input argument(s) */
mxArray *t2, *t3;       /* Optional input argument(s) */
mxArray *S = NULL;      /* Return value */

mlfAssign(&S, mlfChar(X,NULL));
mlfAssign(&S, mlfChar(C,NULL));
mlfAssign(&S, mlfChar(t1,t2,t3,...,NULL));
```

MATLAB Syntax

```
S = char(X)
S = char(C)
S = char(t1,t2,t3...)
```

See Also

MATLAB char Calling Conventions

mlfChol

Purpose

Cholesky factorization

C Prototype

```
mxArray *mlfChol(mxArray **p, mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *p;             /* Optional output argument(s) */
mxArray *R = NULL;      /* Return value */

mlfAssign(&R, mlfChol(NULL,X));
mlfAssign(&R, mlfChol(&p,X));
```

MATLAB Syntax

```
R = chol(X)
[R,p] = chol(X)
```

See Also

MATLAB `chol` Calling Conventions

Purpose

Rank 1 update to Cholesky factorization

C Prototype

```
mxArray *mlfCholupdate(mxArray **p, mxArray *R, mxArray *x,
                       mxArray *flag);
```

C Syntax

```
#include "matlab.h"

mxArray *R, *x;         /* Required input argument(s) */
mxArray *flag;          /* Optional string input argument */
mxArray *p = NULL;      /* Optional output argument(s) */
mxArray *R1 = NULL;     /* Return value */

mlfAssign(&R1, mlfCholupdate(NULL,R,x,NULL));
mlfAssign(&R1, mlfCholupdate(NULL,R,x,flag));
mlfAssign(&R1, mlfCholupdate(&p,R,x,flag));
```

MATLAB Syntax

```
R1 = cholupdate(R,x)
R1 = cholupdate(R,x,'+')
R1 = cholupdate(R,x,'-')
[R1,p] = cholupdate(R,x,'-')
```

See Also

MATLAB `cholupdate` Calling Conventions

mlfCholinc

Purpose

Incomplete Cholesky factorizations

C Prototype

```
mxArray *mlfCholinc(mxArray **p, mxArray *X, mxArray *droptol);
```

C Syntax

```
#include "matlab.h"

mxArray *X, *droptol, *options; /* Required input argument(s) */
mxArray *p = NULL;              /* Optional output argument(s) */
mxArray *R = NULL;              /* Return value */

mlfAssign(&R, mlfCholinc(NULL,X,droptol));
mlfAssign(&R, mlfCholinc(NULL,X,options));
mlfAssign(&R, mlfCholinc(NULL,X,mxCreateString("0")));
mlfAssign(&R, mlfCholinc(&p,X,mxCreateString("0")));
mlfAssign(&R, mlfCholinc(NULL,X,mxCreateString("inf")));
```

MATLAB Syntax

```
R = cholinc(X,droptol)
R = cholinc(X,options)
R = cholinc(X,'0')
[R,p] = cholinc(X,'0')
R = cholinc(X,'inf')
```

See Also

MATLAB `cholinc` Calling Conventions

Purpose

Create object or return class of object

C Prototype

```
mxArray *mlfClassName(mxArray *obj);
```

C Syntax

```
#include "matlab.h"

mxArray *obj;               /* Required input argument(s) */
mxArray *str = NULL;        /* Return value */

mlfAssign(&str, mlfClassName(obj));
```

MATLAB Syntax

```
str = class(object)
obj = class(s,'class_name')
obj = class(s,'class_name',parent1,parent2...)
```

See Also

MATLAB `class` Calling Conventions

mlfClock

Purpose

Current time as a date vector

C Prototype

```
mxArray *mlfClock();
```

C Syntax

```
#include "matlab.h"

mxArray *c = NULL;      /* Return value */

mlfAssign(&c, mlfClock());
```

MATLAB Syntax

```
c = clock
```

See Also

MATLAB `clock` Calling Conventions

Purpose

Sparse column minimum degree permutation

C Prototype

```
mxArray *mlfColmmd(mxArray *S);
```

C Syntax

```
#include "matlab.h"

mxArray *S;                 /* Required input argument(s) */
mxArray *p = NULL;          /* Return value */

mlfAssign(&p, mlfColmmd(S));
```

MATLAB Syntax

```
p = colmmd(S)
```

See Also

MATLAB `colmmd` Calling Conventions

mlfColperm

Purpose

Sparse column permutation based on nonzero count

C Prototype

```
mxArray *mlfColperm(mxArray *S)
```

C Syntax

```
#include "matlab.h"

mxArray *S;                 /* Required input argument(s) */
mxArray *j = NULL;          /* Return value */

mlfAssign(&j, mlfColperm(S));
```

MATLAB Syntax

```
j = colperm(S)
```

See Also

MATLAB `colperm` Calling Conventions

Purpose

Companion matrix

C Prototype

```
mxArray *mlfCompan(mxArray *u);
```

C Syntax

```
#include "matlab.h"

mxArray *u;             /* Required input argument(s) */
mxArray *A = NULL;      /* Return value */

mlfAssign(&A, mlfCompan(u));
```

MATLAB Syntax

```
A = compan(u)
```

See Also

MATLAB compan Calling Conventions

mlfComputer

Purpose

Identify the computer on which MATLAB is running

C Prototype

```
mxArray *mlfComputer(mxArray **maxsize);
```

C Syntax

```
#include "matlab.h"

mxArray *maxsize;           /* Optional input argument(s) */
mxArray *str = NULL;        /* Return value */

mlfAssign(&str, mlfComputer(NULL));
mlfAssign(&str, mlfComputer(&maxsize));
```

MATLAB Syntax

```
str = computer
[str,maxsize] = computer
```

See Also

MATLAB `computer` Calling Conventions

Purpose

Condition number with respect to inversion

C Prototype

```
mxArray *mlfCond(mxArray *X, *p);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *p = NULL;      /* Optional input argument(s) */
mxArray *c = NULL;      /* Return value */

mlfAssign(&c, mlfCond(X,NULL));
mlfAssign(&c, mlfCond(X,p));
```

MATLAB Syntax

```
c = cond(X)
c = cond(X,p)
```

See Also

MATLAB cond Calling Conventions

mlfCondeig

Purpose

Condition number with respect to eigenvalues

C Prototype

```
mxArray *mlfCondeig(mxArray **D, mxArray **s, mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A;                      /* Required input argument(s) */
mxArray *D = NULL, *s = NULL;    /* Optional output argument(s) */
mxArray *c= NULL, *V = NULL;     /* Return value */

mlfAssign(&c, mlfCondeig(NULL,NULL,A));
mlfAssign(&V, mlfCondeig(&D,&s,A));
```

MATLAB Syntax

```
c = condeig(A)
[V,D,s] = condeig(A)
```

See Also

MATLAB `condeig` Calling Conventions

Purpose

1-norm matrix condition number estimate

C Prototype

```
mxArray *mlfCondest(mxArray **v, mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A;                 /* Required input argument(s) */
mxArray *v = NULL;          /* Optional output argument(s) */
mxArray *c = NULL;          /* Return value */

mlfAssign(&c, mlfCondest(NULL,A));
mlfAssign(&c, mlfCondest(&v,A));
```

MATLAB Syntax

```
c = condest(A)
[c,v] = condest(A)
```

See Also

MATLAB `condest` Calling Conventions

mlfConj

Purpose

Complex conjugate

C Prototype

```
mxArray *mlfConj(mxArray *Z);
```

C Syntax

```
#include "matlab.h"

mxArray *Z;             /* Required input argument(s) */
mxArray *ZC = NULL;     /* Return value */

mlfAssign(&ZC, mlfConj(Z));
```

MATLAB Syntax

```
ZC = conj(Z)
```

See Also

MATLAB `conj` Calling Conventions

Purpose

Convolution and polynomial multiplication

C Prototype

```
mxArray *mlfConv(mxArray *u, mxArray *v);
```

C Syntax

```
#include "matlab.h"

mxArray *u, *v;         /* Required input argument(s) */
mxArray *w = NULL;      /* Return value */

mlfAssign(&w, mlfConv(u,v));
```

MATLAB Syntax

```
w = conv(u,v)
```

See Also

MATLAB conv

Calling Conventions

mlfConv2

Purpose

Two-dimensional convolution

C Prototype

```
mxArray *mlfConv2(mxArray *I1, mxArray *I2, mxArray *I3,
                  mxArray *I4);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *B, *hcol, *hrow;   /* Input argument(s) */
mxArray *C = NULL;              /* Return value */

mlfAssign(&C, mlfConv2(A,B,NULL,NULL));
mlfAssign(&C, mlfConv2(A,B,mxCreateString("shape"),NULL));
mlfAssign(&C, mlfConv2(hcol,hrow,A,NULL));
mlfAssign(&C, mlfConv2(hcol,hrow,A,mxCreateString("shape")));
```

MATLAB Syntax

```
C = conv2(A,B)
C = conv2(hcol,hrow,A)
C = conv2(...,'shape')
```

See Also

MATLAB conv2 Calling Conventions

Purpose

Correlation coefficients

C Prototype

```
mxArray *mlfCorrcoef(mxArray *X, mxArray *y);
```

C Syntax

```
#include "matlab.h"

mxArray *X, *x, *y;     /* Input argument(s) */
mxArray *S = NULL;      /* Return value */

mlfAssign(&S, mlfCorrcoef(X,NULL));
mlfAssign(&S, mlfCorrcoef(x,y));
```

MATLAB Syntax

```
S = corrcoef(X)
S = corrcoef(x,y)
```

See Also

MATLAB `corrcoef` Calling Conventions

mlfCos, mlfCosh

Purpose

Cosine and hyperbolic cosine

C Prototype

```
mxArray *mlfCos(mxArray *X);
mxArray *mlfCosh(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfCos(X));
mlfAssign(&Y, mlfCosh(X));
```

MATLAB Syntax

```
Y = cos(X)
Y = cosh(X)
```

See Also

MATLAB cos, cosh Calling Conventions

Purpose

Cotangent and hyperbolic cotangent

C Prototype

```
mxArray *mlfCot(mxArray *X);
mxArray *mlfCoth(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfCot(X));
mlfAssign(&Y, mlfCoth(X));
```

MATLAB Syntax

```
Y = cot(X)
Y = coth(X)
```

See Also

MATLAB `cot`, `coth` Calling Conventions

mlfCov

Purpose

Covariance matrix

Minimum input arguments: one, maximum: user-defined. Terminate the list of arguments with a `NULL`.

C Prototype

```
mxArray *mlfCov(mxArray *x, ...)
```

C Syntax

```
#include "matlab.h"

mxArray *x;             /* Required input argument(s) */
mxArray *Y, *Z;         /* Optional input argument(s) */
mxArray *C = NULL;      /* Return value */

mlfAssign(&C, mlfCov(x,NULL));
mlfAssign(&C, mlfCov(x,Y,NULL));
mlfAssign(&C, mlfCov(x,Y,Z,NULL));
```

MATLAB Syntax

```
C = cov(X)
C = cov(x,y)
```

See Also

MATLAB `cov` Calling Conventions

Purpose

Sort complex numbers into complex conjugate pairs

C Prototype

```
mxArray *mlfCplxpair(mxArray *A, mxArray *tol, mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *A;                     /* Required input argument(s) */
mxArray *tol, *dim;             /* Optional input argument(s) */
mxArray *null_matrix = NULL;    /* Optional input argument(s) */
mxArray *B = NULL;              /* Return value */

mlfAssign(&B, mlfCplxpair(A,NULL,NULL));
mlfAssign(&B, mlfCplxpair(A,tol,NULL));

mlfAssign(&null_matrix, mlfZeros(mlfScalar(0),mlfScalar(0),NULL));
mlfAssign(&B, mlfCplxpair(A,null_matrix,dim));

mlfAssign(&B, mlfCplxpair(A,tol,dim));
```

MATLAB Syntax

```
B = cplxpair(A)
B = cplxpair(A,tol)
B = cplxpair(A,[],dim)
B = cplxpair(A,tol,dim)
```

See Also

MATLAB `cplxpair` Calling Conventions

mlfCross

Purpose

Vector cross product

C Prototype

```
mxArray *mlfCross(mxArray *U, mxArray *V, mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *U, *V;         /* Required input argument(s) */
mxArray *dim;           /* Optional input argument(s) */
mxArray *W = NULL;      /* Return value */

mlfAssign(&W, mlfCross(U,V,NULL));
mlfAssign(&W, mlfCross(U,V,dim));
```

MATLAB Syntax

```
W = cross(U,V)
W = cross(U,V,dim)
```

See Also

MATLAB `cross` Calling Conventions

Purpose

Cosecant and hyperbolic cosecant

C Prototype

```
mxArray *mlfCsc(mxArray *x);
mxArray *mlfCsch(mxArray *x);
```

C Syntax

```
#include "matlab.h"

mxArray *x;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfCsc(x));
mlfAssign(&Y, mlfCsch(x));
```

MATLAB Syntax

```
Y = csc(x)
Y = csch(x)
```

See Also

MATLAB csc, csch Calling Conventions

mlfCumprod

Purpose

Cumulative product

C Prototype

```
mxArray *mlfCumprod(mxArray *A, *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *dim;           /* Optional input argument(s) */
mxArray *B = NULL;      /* Return value */

mlfAssign(&B, mlfCumprod(A,NULL));
mlfAssign(&B, mlfCumprod(A,dim));
```

MATLAB Syntax

```
B = cumprod(A)
B = cumprod(A,dim)
```

See Also

MATLAB `cumprod` Calling Conventions

Purpose

Cumulative sum

C Prototype

```
mxArray *mlfCumsum(mxArray *A, *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *dim;           /* Optional input argument(s) */
mxArray *B = NULL;      /* Return value */

mlfAssign(&B, mlfCumsum(A,NULL));
mlfAssign(&B, mlfCumsum(A,dim));
```

MATLAB Syntax

```
B = cumsum(A)
B = cumsum(A,dim)
```

See Also

MATLAB `cumsum` Calling Conventions

mlfCumtrapz

Purpose

Cumulative trapezoidal numerical integration

C Prototype

```
mxArray *mlfCumtrapz(mxArray *Y, mxArray *X, mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *Y;                 /* Required input argument(s) */
mxArray *X, *dim;           /* Optional input argument(s) */
mxArray *Z = NULL;          /* Return value */

mlfAssign(&Z, mlfCumtrapz(Y,NULL,NULL));
mlfAssign(&Z, mlfCumtrapz(X,Y,NULL));
mlfAssign(&Z, mlfCumtrapz(Y,dim,NULL));
mlfAssign(&Z, mlfCumtrapz(X,Y,dim));
```

MATLAB Syntax

```
Z = cumtrapz(Y)
Z = cumtrapz(X,Y)
Z = cumtrapz(... dim)
```

See Also

MATLAB `cumtrapz` Calling Conventions

mlfDate

Purpose

Current date string

C Prototype

```
mxArray *mlfDate();
```

C Syntax

```
#include "matlab.h"

mxArray *str = NULL;    /* Return value */

mlfAssign(&str, mlfDate());
```

MATLAB Syntax

```
str = date
```

See Also

MATLAB `date` Calling Conventions

Purpose

Serial date number

C Prototype

```
mxArray *mlfDatenum(mxArray *Y, mxArray *M, mxArray *D, mxArray *H,
                    mxArray *MI, mxArray *S);
```

C Syntax

```
#include "matlab.h"

mxArray *str;           /* String array(s) */
mxArray *P, *Y, *M, *D; /* Input argument(s) */
mxArray *H, *MI, *S;    /* Input argument(s) */

mlfAssign(&N, mlfDatenum(str,NULL,NULL,NULL,NULL,NULL));
mlfAssign(&N, mlfDatenum(str,P,NULL,NULL,NULL,NULL));
mlfAssign(&N, mlfDatenum(Y,M,D,NULL,NULL,NULL));
mlfAssign(&N, mlfDatenum(Y,M,D,H,MI,S));
```

MATLAB Syntax

```
N = datenum(str)
N = datenum(str,P)
N = datenum(Y,M,D)
N = datenum(Y,M,D,H,MI,S)
```

See Also

MATLAB `datenum` Calling Conventions

mlfDatestr

Purpose

Date string format

C Prototype

```
mxArray *mlfDatestr(mxArray *D, mxArray *dateform, mxArray *P);
```

C Syntax

```
#include "matlab.h"

mxArray *dateform;          /* Numeric or string array */
mxArray *D;                 /* Required input argument(s) */
mxArray *P;                 /* Optional input argument(s) */
mxArray *str = NULL;        /* Return value */

mlfAssign(&str, mlfDatestr(D,dateform,NULL));
mlfAssign(&str, mlfDatestr(D,dateform,P));
```

MATLAB Syntax

```
str = datestr(D,dateform)
str = datestr(D,dateform,P)
```

See Also

MATLAB `datestr` Calling Conventions

Purpose

Date components

C Prototype

```
mxArray *mlfDatevec(mxArray **M, mxArray **D, mxArray **H,
                    mxArray **MI, mxArray **S, mxArray *A,
                    mxArray *P);
```

C Syntax

```
#include "matlab.h"

mxArray *A;                     /* Required input argument(s) */
mxArray *P;                     /* Optional input argument(s) */
mxArray *M = NULL, *D = NULL;   /* Optional output argument(s) */
mxArray *H = NULL, *MI = NULL;  /* Optional output argument(s) */
mxArray *S = NULL;              /* Optional output argument(s) */
mxArray *C = NULL, *Y = NULL;   /* Return value */

mlfAssign(&C, mlfDatevec(NULL,NULL,NULL,NULL,NULL,A,NULL));
mlfAssign(&C, mlfDatevec(NULL,NULL,NULL,NULL,NULL,A,P));
mlfAssign(&Y, mlfDatevec(&M,&D,&H,&MI,&S,A,NULL));
mlfAssign(&Y, mlfDatevec(&M,&D,&H,&MI,&S,A,P));
```

MATLAB Syntax

```
C = datevec(A)
C = datevec(A,P)
[Y,M,D,H,MI,S] = datevec(A)
```

See Also

MATLAB datevec Calling Conventions

mlfDblquad

Purpose

Numerical double integration

C Prototype

```
mxArray *mlfDblquad(mxArray *func, mxArray *inmin, mxArray *inmax,
                    mxArray *outmin, mxArray *outmax,
                    mxArray *tol, mxArray *method);
```

C Syntax

```
#include "matlab.h"

mxArray *func;              /* String array(s) */
mxArray *inmin, *inmax;     /* Required input argument(s) */
mxArray *outmin, *outmax;   /* Required input argument(s) */
mxArray *tol, *method;      /* Optional input argument(s) */
mxArray *result = NULL;     /* Return value */

mlfAssign(&result, mlfDblquad(func,inmin,inmax,outmin,outmax,
                    NULL,NULL));
mlfAssign(&result, mlfDblquad(func,inmin,inmax,outmin,outmax,
                    tol,NULL));
mlfAssign(&result, mlfDblquad(func,inmin,inmax,outmin,outmax,
                    tol,method));
```

MATLAB Syntax

```
result = dblquad('fun',inmin,inmax,outmin,outmax)
result = dblquad('fun',inmin,inmax,outmin,outmax,tol)
result = dblquad('fun',inmin,inmax,outmin,outmax,tol,method)
```

See Also

MATLAB `dblquad` Calling Conventions

Purpose

Deal inputs to outputs

Minimum number of arguments: one, maximum: user-defined. Terminate the list of arguments with a `NULL`.

C Prototype

```
mxArray *mlfDeal(mlfVarargoutList *varargout, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *X, *X1;            /* Required input argument(s) */
mxArray *X2,*X3;            /* Optional input argument(s) */
mxArray *Y2=NULL,*Y3=NULL;  /* Optional output argument(s) */
mxArray *Y1=NULL;           /* Return value */

mlfDeal(mlfVarargout(&Y1,&Y2,&Y3,...,NULL),X,NULL);
mlfDeal(mlfVarargout(&Y1,&Y2,&Y3,...,NULL),X1,X2,X3,...,NULL);
```

Note For routines where all the output arguments are passed to `mlfVarargout()`, you do not need to use the first output argument as the return value for the function. The first output argument is, in effect, the routine return value.

MATLAB Syntax

```
[Y1,Y2,Y3,...] = deal(X)
[Y1,Y2,Y3,...] = deal(X1,X2,X3,...)
```

See Also

MATLAB `deal` Calling Conventions

mlfDeblank

Purpose

Strip trailing blanks from the end of a string

C Prototype

```
mxArray *mlfDeblank(mxArray *str_in);
```

C Syntax

```
#include "matlab.h"

mxArray *str;           /* String array(s) */
mxArray *c;             /* Cell array */

mlfAssign(&str, mlfDeblank(str));
mlfAssign(&c, mlfDeblank(c));
```

MATLAB Syntax

```
str = deblank(str)
c = deblank(c)
```

See Also

MATLAB `deblank` Calling Conventions

Purpose

Decimal number to base conversion

C Prototype

```
mxArray *mlfDec2base(mxArray *d, mxArray *base, mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *d, *base;          /* Required input argument(s) */
mxArray *n;                 /* Optional input argument(s) */
mxArray *str = NULL;        /* Return value */

mlfAssign(&str, mlfDec2base(d,base,NULL));
mlfAssign(&str, mlfDec2base(d,base,n));
```

MATLAB Syntax

```
str = dec2base(d,base)
str = dec2base(d,base,n)
```

See Also

MATLAB `dec2base` Calling Conventions

mlfDec2bin

Purpose

Decimal to binary number conversion

C Prototype

```
mxArray *mlfDec2bin(mxArray *d, mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *d;                 /* Required input argument(s) */
mxArray *n;                 /* Optional input argument(s) */
mxArray *str = NULL;        /* Return value */

mlfAssign(&str, mlfDec2bin(d,NULL));
mlfAssign(&str, mlfDec2bin(d,n));
```

MATLAB Syntax

```
str = dec2bin(d)
str = dec2bin(d,n)
```

See Also

MATLAB `dec2bin` Calling Conventions

Purpose

Decimal to hexadecimal number conversion

C Prototype

```
mxArray *mlfDec2hex(mxArray *d, mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *d;             /* Required input argument(s) */
mxArray *n;             /* Optional input argument(s) */
mxArray *str = NULL;    /* Return value */

mlfAssign(&str, mlfDec2hex(d,NULL));
mlfAssign(&str, mlfDec2hex(d,n));
```

MATLAB Syntax

```
str = dec2hex(d)
str = dec2hex(d,n)
```

See Also

MATLAB `dec2hex` Calling Conventions

mlfDeconv

Purpose

Deconvolution and polynomial division

C Prototype

```
mxArray *mlfDeconv(mxArray **r, mxArray *v, mxArray *u);
```

C Syntax

```
#include "matlab.h"

mxArray *v, *u;         /* Required input argument(s) */
mxArray *r = NULL;      /* Required output argument(s) */
mxArray *q = NULL;      /* Return value */

mlfAssign(&q, mlfDeconv(&r,v,u));
```

MATLAB Syntax

```
[q,r] = deconv(v,u)
```

See Also

MATLAB deconv Calling Conventions

Purpose

Discrete Laplacian

Minimum number of arguments: one; maximum number: user-defined. Terminate the list of arguments with a `NULL`.

C Prototype

```
mxArray *mlfDel2(mxArray *U, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *U;             /* Required input argument(s) */
mxArray *hx, *hy, *hz;  /* Optional input argument(s) */
mxArray *L = NULL;      /* Return value */

mlfAssign(&L, mlfDel2(U,NULL));
mlfAssign(&L, mlfDel2(U,hx,NULL));
mlfAssign(&L, mlfDel2(U,hx,hy,NULL));
mlfAssign(&L, mlfDel2(U,hx,hy,hz,...,NULL));
```

MATLAB Syntax

```
L = del2(U)
L = del2(U,h)
L = del2(U,hx,hy)
L = del2(U,hx,hy,hz,...)
```

See Also

MATLAB `del2` Calling Conventions

mlfDet

Purpose

Matrix determinant

C Prototype

```
mxArray *mlfDet(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *d = NULL;      /* Return value */

mlfAssign(&d, mlfDet(X));
```

MATLAB Syntax

```
d = det(X)
```

See Also

MATLAB `det` Calling Conventions

Purpose

Diagonal matrices and diagonals of a matrix. The variable v can be a vector or a matrix.

C Prototype

```
mxArray *mlfDiag(mxArray *v, mxArray *k);
```

C Syntax

```
#include "matlab.h"

mxArray *v, *k, *X;

mlfAssign(&X, mlfDiag(v,k));
mlfAssign(&X, mlfDiag(v,NULL));
mlfAssign(&v, mlfDiag(X,k));
mlfAssign(&v, mlfDiag(X,NULL));
```

MATLAB Syntax

```
X = diag(v,k)
X = diag(v)
v = diag(X,k)
v = diag(X)
```

See Also

MATLAB diag Calling Conventions

mlfDiff

Purpose

Differences and approximate derivatives

C Prototype

```
mxArray *mlfDiff(mxArray *X, mxArray *n, mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *n, *dim;       /* Optional input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfDiff(X,NULL,NULL));
mlfAssign(&Y, mlfDiff(X,n,NULL));
mlfAssign(&Y, mlfDiff(X,n,dim));
```

MATLAB Syntax

```
Y = diff(X)
Y = diff(X,n)
Y = diff(X,n,dim)
```

See Also

MATLAB `diff` Calling Conventions

Purpose

Display text or array

C Prototype

```
void mlfDisp(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */

mlfDisp(X);
```

MATLAB Syntax

```
disp(X)
```

See Also

MATLAB `disp` Calling Conventions

mlfDmperm

Purpose

Dulmage-Mendelsohn decomposition

C Prototype

```
mxArray *mlfDmperm(mxArray **q, mxArray **r, mxArray **s,
                   mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A;                     /* Required input argument(s) */
mxArray *q = NULL, *r = NULL;   /* Optional output argument(s) */
mxArray *s = NULL;              /* Optional output argument(s) */
mxArray *p = NULL;              /* Return value */

mlfAssign(&p, mlfDmperm(NULL,NULL,NULL,A));
mlfAssign(&p, mlfDmperm(&q,&r,NULL,A));
mlfAssign(&p, mlfDmperm(&q,&r,&s,A));
```

MATLAB Syntax

```
p = dmperm(A)
[p,q,r] = dmperm(A)
[p,q,r,s] = dmperm(A)
```

See Also

MATLAB `dmperm` Calling Conventions

Purpose

Convert to double precision

C Prototype

```
mxArray *mlfDouble(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;                 /* Required input argument(s) */
mxArray *R = NULL;          /* Return value */

mlfAssign(&R, mlfDouble(X));
```

MATLAB Syntax

```
double(X)
```

See Also

MATLAB `double` Calling Conventions

mlfEig

Purpose

Eigenvalues and eigenvectors

C Prototype

```
mxArray *mlfEig(mxArray **D, mxArray *A, mxArray *B);
```

C Syntax

```
#include "matlab.h"

mxArray *A;                     /* Required input argument(s) */
mxArray *B;                     /* Optional input argument(s) */
mxArray *D = NULL;              /* Optional output argument(s) */
mxArray *d = NULL, *V = NULL;   /* Return value */

mlfAssign(&d, mlfEig(NULL,A,NULL));
mlfAssign(&V, mlfEig(&D,A,NULL));
mlfAssign(&V, mlfEig(&D,A,mxCreateString("nobalance")));
mlfAssign(&d, mlfEig(NULL,A,B));
mlfAssign(&V, mlfEig(&D,A,B));
```

MATLAB Syntax

```
d = eig(A)
[V,D] = eig(A)
[V,D] = eig(A,'nobalance')
d = eig(A,B)
[V,D] = eig(A,B)
```

See Also

MATLAB `eig` Calling Conventions

Purpose

Find a few eigenvalues and eigenvectors

Minimum number of arguments: two, maximum: user-defined. Terminate the argument list with a NULL.

C Prototype

```
mxArray *mlfEigs(mxArray **d, mxArray **flag, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *A;                     /* Required input argument(s) */
mxArray *n, *B, *k;             /* Optional input argument(s) */
mxArray *sigma, *options;       /* Optional input argument(s) */
mxArray *D = NULL, *flag = NULL;/* Optional output argument(s) */
mxArray *d = NULL, *V = NULL;   /* Return value */

mlfAssign(&d, mlfEigs(NULL,NULL,A,NULL));
mlfAssign(&d, mlfEigs(NULL,NULL,mxCreateString("Afun"),n,NULL));
mlfAssign(&d, mlfEigs(NULL,NULL,A,B,k,sigma,options,NULL));
mlfAssign(&d, mlfEigs(NULL,NULL,mxCreateString("Afun"),n,
                      B,k,sigma,options,NULL));

mlfAssign(&V, mlfEigs(&D,NULL,A,NULL));
mlfAssign(&V, mlfEigs(&D,NULL,mxCreateString("Afun"),n,NULL));
mlfAssign(&V, mlfEigs(&D,NULL,A,B,k,sigma,options,NULL));
mlfAssign(&V, mlfEigs(&D,NULL,mxCreateString("Afun"),n,
                      B,k,sigma,options,NULL));

mlfAssign(&V, mlfEigs(&D,&flag,A,NULL));
mlfAssign(&V, mlfEigs(&D,&flag,mxCreateString("Afun"),n,NULL));
mlfAssign(&V, mlfEigs(&D,&flag,A,B,k,sigma,options,NULL));
mlfAssign(&V, mlfEigs(&D,&flag,mxCreateString("Afun"),n,
                      B,k,sigma,options,NULL));
```

mlfEigs

MATLAB Syntax

```
d = eigs(A)
d = eigs('Afun',n)
d = eigs(A,B,k,sigma,options)
d = eigs('Afun',n,B,k,sigma,options)
[V,D] = eigs(A,...)
[V,D] = eigs('Afun',n,...)
[V,D,flag] = eigs(A,...)
[V,D,flag] = eigs('Afun',n,...)
```

See Also MATLAB `eigs` Calling Conventions

Purpose

Jacobi elliptic functions

C Prototype

```
mxArray *mlfEllipj(mxArray **CN, mxArray **DN, mxArray *U,
                   mxArray *M, mxArray *tol);
```

C Syntax

```
#include "matlab.h"

mxArray *U, *M;                 /* Required input argument(s) */
mxArray *tol;                   /* Optional input argument(s) */
mxArray *CN = NULL, *DN = NULL; /* Required output argument(s) */
mxArray *SN = NULL;             /* Return value */

mlfAssign(&SN, mlfEllipj(&CN,&DN,U,M,NULL));
mlfAssign(&SN, mlfEllipj(&CN,&DN,U,M,tol));
```

MATLAB Syntax

```
[SN,CN,DN]  =  ellipj(U,M)
[SN,CN,DN]  =  ellipj(U,M,tol)
```

See Also

MATLAB `ellipj` Calling Conventions

mlfEllipke

Purpose

Complete elliptic integrals of the first and second kind

C Prototype

```
mxArray *mlfEllipke(mxArray **E, mxArray *M, mxArray *tol);
```

C Syntax

```
#include "matlab.h"

mxArray *M;             /* Required input argument(s) */
mxArray *tol;           /* Optional input argument(s) */
mxArray *E = NULL;      /* Optional output argument(s) */
mxArray *K = NULL;      /* Return value */

mlfAssign(&K, mlfEllipke(NULL,M,NULL));
mlfAssign(&K, mlfEllipke(&E,M,NULL));
mlfAssign(&K, mlfEllipke(&E,M,tol));
```

MATLAB Syntax

```
K = ellipke(M)
[K,E] = ellipke(M)
[K,E] = ellipke(M,tol)
```

See Also

MATLAB `ellipke` Calling Conventions

Purpose

End of month

C Prototype

```
mxArray *mlfEomday(mxArray *Y, mxArray *M);
```

C Syntax

```
#include "matlab.h"

mxArray *Y, *M;             /* Required input argument(s) */
mxArray *E = NULL;          /* Return value */

mlfAssign(&E, mlfEomday(Y,M));
```

MATLAB Syntax

```
E = eomday(Y,M)
```

See Also

MATLAB eomday Calling Conventions

mlfEps

Purpose

Floating-point relative accuracy

C Prototype

```
mxArray *mlfEps();
```

C Syntax

```
#include "matlab.h"

mxArray *R = NULL;      /* Return value */

mlfAssign(&R, mlfEps());
```

MATLAB Syntax

```
eps
```

See Also

MATLAB eps Calling Conventions

Purpose

Error functions

C Prototype

```
mxArray *mlfErf(mxArray *X);
mxArray *mlfErfc(mxArray *X);
mxArray *mlfErfcx(mxArray *X);
mxArray *mlfErfinv(mxArray *Y);
```

C Syntax

```
#include "matlab.h"

mxArray *X = NULL;
mxArray *Y = NULL;

mlfAssign(&Y, mlfErf(X));   /* Error function */
mlfAssign(&Y, mlfErfc(X));  /* Complementary error function */
mlfAssign(&Y, mlfErfcx(X)); /* Scaled complementary error
                             * function*/
mlfAssign(&X, mlfErfinv(Y));/* Inverse of the error function */
```

MATLAB Syntax

```
Y = erf(X)              /* Error function */
Y = erfc(X)             /* Complementary error function */
Y = erfcx(X)            /* Scaled complementary error function */
X = erfinv(Y)           /* Inverse of the error function */
```

See Also

MATLAB `erf`, `erfc`, `erfx`, `erfinv` Calling Conventions

mlfError

Purpose

Display error messages

C Prototype

```
void mlfError(mxArray *mssg);
```

C Syntax

```
#include "matlab.h"

mxArray *mssg;          /* String array(s) */

mlfError(mssg);
```

MATLAB Syntax

```
error('error_message')
```

See Also

MATLAB `error` Calling Conventions

Purpose

Elapsed time

C Prototype

```
mxArray *mlfEtime(mxArray *t2, mxArray *t1);
```

C Syntax

```
#include "matlab.h"

mxArray *t2, *t1;       /* Required input argument(s) */
mxArray *e = NULL;      /* Return value */

mlfAssign(&e, mlfEtime(t2,t1));
```

MATLAB Syntax

```
e = etime(t2,t1)
```

See Also

MATLAB `etime` Calling Conventions

mlfExp

Purpose

Exponential

C Prototype

```
mxArray *mlfExp(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfExp(X));
```

MATLAB Syntax

```
Y = exp(X)
```

See Also

MATLAB `exp` Calling Conventions

Purpose

Exponential integral

C Prototype

```
mxArray *mlfExpint(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfExpint(X));
```

MATLAB Syntax

```
Y = expint(X)
```

See Also

MATLAB `expint` Calling Conventions

mlfExpm

Purpose

Matrix exponential

C Prototype

```
mxArray *mlfExpm(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfExpm(X));
```

MATLAB Syntax

```
Y = expm(X)
```

See Also

MATLAB `expm` Calling Conventions

Purpose

Matrix exponential via Pade approximation

C Prototype

```
mxArray *mlfExpm1(mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *E = NULL;      /* Return value */

mlfAssign(&E, mlfExpm1(A));
```

MATLAB Syntax

```
E = expm1(A)
```

See Also

MATLAB `expm` Calling Conventions

mlfExpm2

Purpose

Matrix exponential via Taylor series

C Prototype

```
mxArray *mlfExpm2(mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *E = NULL;      /* Return value */

mlfAssign(&E, mlfExpm2(A));
```

MATLAB Syntax

```
E = expm2(A)
```

See Also

MATLAB `expm` Calling Conventions

Purpose

Matrix exponential via eigenvalues and eigenvectors

C Prototype

```
mxArray *mlfExpm3(mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *E = NULL;      /* Return value */

mlfAssign(&E, mlfExpm3(A));
```

MATLAB Syntax

```
E = expm3(A)
```

See Also

MATLAB `expm` Calling Conventions

mlfEye

Purpose

Identity matrix

C Prototype

```
mxArray *mlfEye(mxArray *m, mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *n, *m, *A;         /* Input argument(s) */
mxArray *Y = NULL;          /* Return value */

mlfAssign(&Y, mlfEye(n,NULL));
mlfAssign(&Y, mlfEye(m,n));
mlfAssign(&Y, mlfEye(mlfSize(NULL,A,NULL)));
```

MATLAB Syntax

```
Y  =  eye(n)
Y  =  eye(m,n)
Y  =  eye(size(A))
```

See Also

MATLAB eye Calling Conventions

mlfFactor

Purpose

Prime factors

C Prototype

```
mxArray *mlfFactor(mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *n;                 /* Required input argument(s) */
mxArray *f = NULL;          /* Return value */

mlfAssign(&f, mlfFactor(n));
```

MATLAB Syntax

```
f = factor(n)
f = factor(symb)
```

See Also

MATLAB `factor` Calling Conventions

Purpose

Close one or more open files

C Prototype

```
mxArray *mlfFclose(mxArray *fid);
```

C Syntax

```
#include "matlab.h"

mxArray *all_str;       /* String array(s) */
mxArray *fid;           /* Required input argument(s) */
mxArray *status = NULL; /* Return value */

mlfAssign(&status, mlfFclose(fid));
mlfAssign(&status, mlfFclose(mxCreateString("all")));
```

MATLAB Syntax

```
status = fclose(fid)
status = fclose('all')
```

See Also

MATLAB `fclose` Calling Conventions

mlfFeof

Purpose

Test for end-of-file

C Prototype

```
mxArray *mlfFeof(mxArray *fid);
```

C Syntax

```
#include "matlab.h"

mxArray *fid;           /* Required input argument(s) */
mxArray *eofstat = NULL;/* Return value */

mlfAssign(&eofstat, mlfFeof(fid));
```

MATLAB Syntax

```
eofstat = feof(fid)
```

See Also

MATLAB `feof` Calling Conventions

Purpose

Query MATLAB about errors in file input or output

C Prototype

```
mxArray *mlfFerror(mxArray **errnum, mxArray *fid,
                   mxArray *clear);
```

C Syntax

```
#include "matlab.h"

mxArray *fid;           /* Required input argument(s) */
mxArray *clear;         /* Optional input argument(s) */
mxArray *errnum = NULL; /* Optional output argument(s) */
mxArray *message = NULL;/* Return value */

mlfAssign(&message, mlfFerror(NULL,fid,NULL));
mlfAssign(&message, mlfFerror(NULL,fid,
                              mxCreateString("clear"));
mlfAssign(&message, mlfFerror(&errnum,fid,NULL));
mlfAssign(&message, mlfFerror(&errnum,fid,
                              mxCreateString("clear"));
```

MATLAB Syntax

```
message = ferror(fid)
message = ferror(fid,'clear')
[message,errnum] = ferror(...)
```

See Also

MATLAB `ferror` Calling Conventions

mlfFeval

Purpose

Function evaluation.

Minimum number of arguments: two, maximum: user-defined. Terminate the argument list with a `NULL`.

C Prototype

```
mxArray *mlfFeval(mlfVarargoutList *varargout,
                  void (*mxfn)(int nlhs,
                               mxArray **plhs,
                               int nrhs,
                               mxArray **prhs),
                  ...);
```

C Syntax

```
#include "matlab.h"

mxArray *x1, *x2;       /* Optional input argument(s) */
mxArray *y1, *y2;       /* Optional output argument(s) */

mlfFeval(mlfVarargout(y1,y2,...,NULL),
         mlfFevalLookup(mxCreateString("function")),
         x1, x2,...,NULL);
```

Note With pure varargout functions, do not use the first output argument as the return value for the function. Pass all output arguments to `mlfVarargout()`. The first output argument is, in effect, the routine return value.

MATLAB Syntax

```
[y1,y2, ...] = feval(function,x1,...,xn)
```

See Also

MATLAB `feval` Calling Conventions

Purpose

One-dimensional fast Fourier transform

C Prototype

```
mxArray *mlfFft(mxArray *X, mxArray *n, mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *X;                     /* Required input argument(s) */
mxArray *n, *dim;               /* Optional input argument(s) */
mxArray *null_matrix = NULL;    /* Optional input argument(s) */
mxArray *Y = NULL;              /* Return value */

mlfAssign(&Y, mlfFft(X,NULL,NULL));
mlfAssign(&Y, mlfFft(X,n,NULL));

mlfAssign(&null_matrix, mlfZeros(mlfScalar(0),mlfScalar(0),NULL));
mlfAssign(&Y, mlfFft(X,null_matrix,dim));

mlfAssign(&Y, mlfFft(X,n,dim));
```

MATLAB Syntax

```
Y = fft(X)
Y = fft(X,n)
Y = fft(X,[],dim)
Y = fft(X,n,dim)
```

See Also

MATLAB `fft` Calling Conventions

mlfFft2

Purpose

Two-dimensional fast Fourier transform

C Prototype

```
mxArray *mlfFft2(mxArray *X, mxArray *m, mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *m, *n;         /* Optional input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfFft2(X,NULL,NULL));
mlfAssign(&Y, mlfFft2(X,m,n));
```

MATLAB Syntax

```
Y = fft2(X)
Y = fft2(X,m,n)
```

See Also

MATLAB `fft2` Calling Conventions

Purpose

Multidimensional fast Fourier transform

C Prototype

```
mxArray *mlfFftn(mxArray *X, mxArray * siz);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *siz;           /* Optional input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign (&Y, mlfFftn(X,NULL));
mlfAssign (&Y, mlfFftn(X,siz));
```

MATLAB Syntax

```
Y = fftn(X)
Y = fftn(X,siz)
```

See Also

MATLAB `fftn` Calling Conventions

mlfFftshift

Purpose

Shift DC component of fast Fourier transform to center of spectrum

C Prototype

```
mxArray *mlfFftshift(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfFftshift(X));
```

MATLAB Syntax

```
Y = fftshift(X)
```

See Also

MATLAB `fftshift` Calling Conventions

Purpose

Read line from file, discard newline character

C Prototype

```
mxArray *mlfFgetl(mxArray *fid);
```

C Syntax

```
#include "matlab.h"

mxArray *fid;           /* Required input argument(s) */
mxArray *line = NULL;   /* Return value */

mlfAssign(&line, mlfFgetl(fid));
```

MATLAB Syntax

```
line = fgetl(fid)
```

See Also

MATLAB `fgetl` Calling Conventions

mlfFgets

Purpose

Return the next line of a file as a string with line terminator(s)

C Prototype

```
mxArray *mlfFgets(mxArray **EOL, mxArray *fid, mxArray *nchar);
```

C Syntax

```
#include "matlab.h"

mxArray *fid;           /* Required input argument(s) */
mxArray *nchar;         /* Optional input argument(s) */
mxArray *EOL = NULL;    /* Optional output argument(s) */
mxArray *line = NULL;   /* Return value */

mlfAssign(&line, mlfFgets(NULL,fid,NULL));
mlfAssign(&line, mlfFgets(NULL,fid,nchar));
mlfAssign(&line, mlfFgets(&EOL,fid,NULL));
mlfAssign(&line, mlfFgets(&EOL,fid,nchar));
```

MATLAB Syntax

```
line = fgets(fid)
line = fgets(fid,nchar)
```

See Also

MATLAB `fgets` Calling Conventions

Purpose

Field names of a structure

C Prototype

```
mxArray *mlfFieldnames(mxArray *s);
```

C Syntax

```
#include "matlab.h"

mxArray *s;                 /* Required input argument(s) */
mxArray *names = NULL;      /* Return value */

mlfAssign(&names, mlfFieldnames(s));
```

MATLAB Syntax

```
names = fieldnames(s)
```

See Also

MATLAB `fieldnames` Calling Conventions

mlfFilter

Purpose

Filter data with an infinite impulse response (IIR) or finite impulse response (FIR) filter

C Prototype

```
mxArray *mlfFilter(mxArray **zf, mxArray *b, mxArray *a,
                   mxArray *X, mxArray *zi, mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *b, *a, *X;         /* Required input argument(s) */
mxArray *null_array = NULL; /* Optional input argument(s) */
mxArray *zi, *dim;          /* Optional input argument(s) */
mxArray *zf = NULL;         /* Optional output argument(s) */
mxArray *y = NULL;          /* Return value */

mlfAssign(&y, mlfFilter(NULL,b,a,X,NULL,NULL));
mlfAssign(&y, mlfFilter(&zf,b,a,X,NULL,NULL));
mlfAssign(&y, mlfFilter(&zf,b,a,X,zi,NULL));

mlfAssign(&null_array, mlfZeros(mlfScalar(0),mlfScalar(0),NULL));
mlfAssign(&y, mlfFilter(NULL,b,a,X,zi,dim));
mlfAssign(&y, mlfFilter(&zf,b,a,X,null_array,dim));
```

MATLAB Syntax

```
y = filter(b,a,X)
[y,zf] = filter(b,a,X)
[y,zf] = filter(b,a,X,zi)
y = filter(b,a,X,zi,dim)
[...] = filter(b,a,X,[],dim)
```

See Also

MATLAB `filter` Calling Conventions

Purpose Two-dimensional digital filtering

C Prototype

```
mxArray *mlfFilter2(mxArray *h, mxArray *X, mxArray *shape);
```

C Syntax

```
#include "matlab.h"

mxArray *h, *X;         /* Required input argument(s) */
mxArray *shape;         /* Optional input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfFilter2(h,X,NULL));
mlfAssign(&Y, mlfFilter2(h,X,shape));
```

MATLAB Syntax

```
Y = filter2(h,X)
Y = filter2(h,X,shape)
```

See Also MATLAB `filter2` Calling Conventions

mlfFind

Purpose

Find indices and values of nonzero elements

C Prototype

```
mxArray *mlfFind(mxArray **j, mxArray **v, mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;                     /* Required input argument(s) */
mxArray *j = NULL, *v = NULL;   /* Optional output argument(s) */
mxArray *k = NULL, *i = NULL;   /* Return value */

mlfAssign(&k, mlfFind(NULL,NULL,X));
mlfAssign(&i, mlfFind(&j,NULL,X));
mlfAssign(&i, mlfFind(&j,&v,X));
```

MATLAB Syntax

```
k = find(X)
[i,j] = find(X)
[i,j,v] = find(X)
```

See Also

MATLAB `find` Calling Conventions

Purpose

Find one string within another

C Prototype

```
mxArray *mlfFindstr(mxArray *str1, mxArray *str2);
```

C Syntax

```
#include "matlab.h"

mxArray *str1, *str2;   /* String array(s) */
mxArray *k = NULL;      /* Return value */

mlfAssign(&k, mlfFindstr(str1,str2));
```

MATLAB Syntax

```
k = findstr(str1,str2)
```

See Also

MATLAB `findstr` Calling Conventions

mlfFix

Purpose

Round towards zero

C Prototype

```
mxArray *mlfFix(mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *B = NULL;      /* Return value */

mlfAssign(&B, mlfFix(A));
```

MATLAB Syntax

```
B = fix(A)
```

See Also

MATLAB `fix` Calling Conventions

Purpose

Flip matrices from left to right

C Prototype

```
mxArray *mlfFliplr(mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *B = NULL;      /* Return value */

mlfAssign(&B, mlfFliplr(A));
```

MATLAB Syntax

```
B = fliplr(A)
```

See Also

MATLAB `fliplr` Calling Conventions

mlfFlipud

Purpose

Flip matrices from up to down

C Prototype

```
mxArray *mlfFlipud(mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *B = NULL;      /* Return value */

mlfAssign(&B, mlfFlipud(A));
```

MATLAB Syntax

```
B = flipud(A)
```

See Also

MATLAB `flipud` Calling Conventions

Purpose

Round towards minus infinity

C Prototype

```
mxArray *mlfFloor (mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *B = NULL;      /* Return value */

mlfAssign(&B, mlfFloor(A));
```

MATLAB Syntax

```
B = floor(A)
```

See Also

MATLAB `floor` Calling Conventions

mlfFlops

Purpose Count floating-point operations

C Prototype

```
mxArray *mlfFlops(mxArray *m);
```

C Syntax

```
#include "matlab.h"

mxArray *f = NULL;          /* Return value */

mlfAssign(&f, mlfFlops(NULL));
mlfAssign(&f, mlfFlops(mlfScalar(0)));
```

Note The math library function `mlfFlops()` always returns the flops count, even if a count is passed as an input value.

MATLAB Syntax

```
f = flops
flops(0)
```

See Also MATLAB `flops` Calling Conventions

Purpose

Minimize a function of one variable

Note The `mlfFmin` routine was replaced by `mlfFminbnd` in Release 11 (MATLAB 5.3). In Release 12 (MATLAB 6.0), `mlfFmin` displays a warning and calls `mlfFminbnd`.

Minimum number of arguments: five, maximum: user-defined. Terminate the argument list with a `NULL`.

C Prototype

```
mxArray *mlfFmin(mxArray **options_out, mxArray *func,
                 mxArray *x1, mxArray *x2,
                 mxArray *options_in, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *x1, *x2;               /* Required input argument(s) */
mxArray *options_in, *P1, *P2;  /* Optional input argument(s) */
mxArray *options_out = NULL;    /* Optional output argument(s) */
mxArray *x = NULL;              /* Return value */

mlfAssign(&x, mlfFmin(NULL,mxCreateString("func"),
                      x1,x2,NULL,NULL));
mlfAssign(&x, mlfFmin(NULL,mxCreateString("func"),
                      x1,x2,options_in,NULL));
mlfAssign(&x, mlfFmin(NULL,mxCreateString("func"),
                      x1,x2,options_in,P1,P2,...,NULL));
mlfAssign(&x, mlfFmin(&options_out,mxCreateString("func"),
                      x1,x2,NULL,NULL));
mlfAssign(&x, mlfFmin(&options_out,mxCreateString("func"),
                      x1,x2,options_in,NULL));
mlfAssign(&x, mlfFmin(&options_out,mxCreateString("func"),
                      x1,x2,options_in,P1,P2,...,NULL));
```

mlfFmin

MATLAB Syntax

```
x = fmin('func',x1,x2)
x = fmin('func',x1,x2,options)
x = fmin('func',x1,x2,options,P1,P2,...)
[x,options] = fmin(...)
```

See Also MATLAB `fmin` Calling Conventions

Purpose

Minimize a function of one variable on a fixed interval

Minimum number of arguments: seven, maximum: user-defined. Terminate the argument list with a `NULL`.

C Prototype

```
mxArray *mlfFminbnd(mxArray **fval, mxArray **exitflag,
                    mxArray **output, mxArray *funfcn,
                    mxArray *ax, mxArray *bx,
                    mxArray *options, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *x1, *x2;               /* Required input argument(s) */
mxArray *options, *P1, *P2;     /* Optional input argument(s) */
mxArray *fval = NULL;           /* Optional output argument(s) */
mxArray *exitflag = NULL;       /* Optional output argument(s) */
mxArray *output = NULL;         /* Optional output argument(s) */
mxArray *x = NULL;              /* Return value */

/* MATLAB syntax: x = fminbnd(func,x1,x2) */
mlfAssign(&x, mlfFminbnd(NULL,NULL,NULL,mxCreateString("func"),
                         x1,x2,NULL,NULL));

/* MATLAB syntax: x = fminbnd(func,x1,x2,options) */
mlfAssign(&x, mlfFminbnd(NULL,NULL,NULL,mxCreateString("func"),
                         x1,x2,options,NULL));

/* MATLAB syntax: x = fminbnd(func,x1,x2,P1,P2,...) */
mlfAssign(&x, mlfFminbnd(NULL,NULL,NULL,mxCreateString("func"),
                         x1,x2,options,P1,P2,...,NULL));

/* MATLAB syntax: [x,fval] = fminbnd(...) */
mlfAssign(&x, mlfFminbnd(&fval,NULL,NULL,mxCreateString("func"),
                         x1,x2,NULL,NULL));
mlfAssign(&x, mlfFminbnd(&fval,NULL,NULL,mxCreateString("func"),
                         x1,x2,options,NULL));
mlfAssign(&x, mlfFminbnd(&fval,NULL,NULL,mxCreateString("func"),
                         x1,x2,options,P1,P2,...,NULL));

/* MATLAB syntax: [x,fval,exitflag ] = fminbnd(...) */
mlfAssign(&x,
  mlfFminbnd(&fval,&exitflag,NULL,mxCreateString("func"),
                     x1,x2,NULL,NULL));
mlfAssign(&x,
  mlfFminbnd(&fval,&exitflag,NULL,mxCreateString("func"),
                     x1,x2,options,NULL));
mlfAssign(&x,
```

```
  mlfFminbnd(&fval,&exitflag,NULL,mxCreateString("func"),
                    x1,x2,options,P1,P2,...,NULL));

/* MATLAB syntax: [x,fval,exitflag,output] = fminbnd(...) */
mlfAssign(&x,
  mlfFminbnd(&fval,&exitflag,&output,mxCreateString("func"),
                    x1,x2,NULL,NULL));
mlfAssign(&x,
  mlfFminbnd(&fval,&exitflag,&output,mxCreateString("func"),
                    x1,x2,options,NULL));
mlfAssign(&x,
  mlfFminbnd(&fval,&exitflag,&output,mxCreateString("func"),
                    x1,x2,options,P1,P2,...,NULL));
```

Syntax

```
x = fminbnd(func,x1,x2)
x = fminbnd(func,x1,x2,options)
x = fminbnd(func,x1,x2,options,P1,P2,...)
[x,fval] = fminbnd(...)
[x,fval,exitflag] = fminbnd(...)
[x,fval,exitflag,output] = fminbnd(...)
```

See Also

MATLAB `fminbnd` Calling Conventions

mlfFmins

Purpose

Minimize a function of several variables

Note The `mlfFmins` routine was replaced by `mlfFminsearch` in Release 11 (MATLAB 5.3). In Release 12 (MATLAB 6.0), `mlfFmins` displays a warning and calls `mlfFminsearch`.

Minimum number of arguments: five, maximum: user-defined. Terminate argument list with a `NULL`.

C Prototype

```
mxArray *mlfFmins(mxArray **options_out, mxArray *func, mxArray *x0,
                          mxArray *options_in, mxArray *grad, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *x0;                    /* Required input argument(s) */
mxArray *options_in, *P1, *P2;  /* Optional input argument(s) */
mxArray *null_matrix = NULL;    /* Optional input argument(s) */
mxArray *options_out = NULL;    /* Optional output argument(s) */
mxArray *x = NULL;              /* Return value */

mlfAssign(&null_matrix, mlfZeros(mlfScalar(0),mlfScalar(0),NULL));

mlfAssign(&x, mlfFmins(NULL,mxCreateString("func"),
                        x0,NULL,NULL,NULL));
mlfAssign(&x, mlfFmins(NULL,mxCreateString("func"),
                        x0,options_in,NULL,NULL));
mlfAssign(&x, mlfFmins(NULL,mxCreateString("func"),
                        x0,options_in,null_matrix,P1,P2,...,NULL));
mlfAssign(&x, mlfFmins(&options_out,mxCreateString("func"),
                        x0,NULL,NULL,NULL));
mlfAssign(&x, mlfFmins(&options_out,mxCreateString("func"),
                        x0,options_in,NULL,NULL));
mlfAssign(&x, mlfFmins(&options_out,mxCreateString("func"),
                        x0,options_in,null_matrix,P1,P2,...,NULL));
```

MATLAB Syntax

```
x = fmins('func',x0)
x = fmins('func',x0,options)
x = fmins('func',x0,options,[],P1,P2, ...)
[x,options] = fmins(...)
```

See Also MATLAB `fmins` Calling Conventions

mlfFminsearch

Purpose

Minimize a function of several variables

Minimum number of arguments: six, maximum: user-defined. Terminate argument list with a `NULL`.

C Prototype

```
mxArray *mlfFminsearch(mxArray **fval, mxArray **exitflag,
                       mxArray **output, mxArray *funfcn,
                       mxArray *x0, mxArray *options, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *x0;                    /* Required input argument(s) */
mxArray *options, *P1, *P2;     /* Optional input argument(s) */
mxArray *fval = NULL;           /* Optional output argument(s) */
mxArray *exitflag = NULL;       /* Optional output argument(s) */
mxArray *output = NULL;         /* Optional output argument(s) */
mxArray *x = NULL;              /* Return value */

/* MATLAB syntax: x = fminsearch(fun,x0) */
mlfAssign(&x, mlfFminsearch(NULL,NULL,NULL,mxCreateString("func"),
                        x0,NULL,NULL));

/* MATLAB syntax: x = fminsearch(fun,x0,options) */
mlfAssign(&x, mlfFminsearch(NULL,NULL,NULL,mxCreateString("func"),
                        x0,options,NULL));

/* MATLAB syntax: x = fminsearch(fun,x0,options,P1,P2,...) */
mlfAssign(&x, mlfFminbnd(NULL,NULL,NULL,mxCreateString("func"),
                        x0,options,P1,P2,...,NULL));

/* MATLAB syntax: [x,fval] = fminsearch(...) */
mlfAssign(&x,
  mlfFminsearch(&fval,NULL,NULL,mxCreateString("func"),
                  x0,NULL,NULL));
mlfAssign(&x,
  mlfFminsearch(&fval,NULL,NULL,mxCreateString("func"),
                  x0,options,NULL));
mlfAssign(&x,
  mlfFminsearch(&fval,NULL,NULL,mxCreateString("func"),
                  x0,options,P1,P2,...,NULL));

/* MATLAB syntax: [x,fval,exitflag] = fminsearch(...)*/
mlfAssign(&x,
  mlfFminsearch(&fval,&exitflag,NULL,mxCreateString("func"),
                  x0,NULL,NULL));
mlfAssign(&x,
```

```
  mlfFminsearch(&fval,&exitflag,NULL,mxCreateString("func"),
                x0,options,NULL));
mlfAssign(&x,
  mlfFminsearch(&fval,&exitflag,NULL,mxCreateString("func"),
               x0,options,P1,P2,...,NULL));

/* MATLAB syntax: [x,fval,exitflag,output] = fminsearch(...) */
mlfAssign(&x,
  mlfFminsearch(&fval,&exitflag,&output,mxCreateString("func"),
                    x0,NULL,NULL));
mlfAssign(&x,
  mlfFminsearch(&fval,&exitflag,&output,mxCreateString("func"),
                    x0,options,NULL));
mlfAssign(&x,
  mlfFminsearch(&fval,&exitflag,&output,mxCreateString("func"),
                    x0,options,P1,P2,...,NULL));
```

Syntax

```
x = fminsearch(fun,x0)
x = fminsearch(fun,x0,options)
x = fminsearch(fun,x0,options,P1,P2,...)
[x,fval] = fminsearch(...)
[x,fval,exitflag] = fminsearch(...)
[x,fval,exitflag,output] = fminsearch(...)
```

See Also

MATLAB `fminsearch` Calling Conventions

Purpose

Open a file or obtain information about open files

C Prototype

```
mxArray *mlfFopen(mxArray **O1, mxArray **O2, mxArray *I1,
                  mxArray *I2, mxArray *I3);
```

C Syntax

```
#include "matlab.h"

mxArray *format;                    /* String array(s) */
mxArray *permission;                /* String array(s) */
mxArray *message, *filename;        /* String array(s) */
mxArray *fid = NULL, *fids = NULL;  /* Return value */

mlfAssign(&fid, mlfFopen(NULL,NULL,filename,permission,NULL));
mlfAssign(&fid,
          mlfFopen(&message,NULL,filename,permission,format));
mlfAssign(&fids,
          mlfFopen(NULL,NULL,mxCreateString("all"),NULL,NULL));
mlfAssign(&filename, mlfFopen(&permission,&format,fid,NULL,NULL));
```

MATLAB Syntax

```
fid = fopen(filename,permission)
[fid,message] = fopen(filename,permission,format)
fids = fopen('all')
[filename,permission,format] = fopen(fid)
```

See Also

MATLAB `fopen` Calling Conventions

mlfFormat

Purpose

Control the output display format

C Prototype

```
void mlfFormat(mxArray *format, mxArray *precision);
```

C Syntax

```
#include "matlab.h"

mxArray *format, *precision;/* String array(s) */

mlfFormat(NULL,NULL);
mlfFormat(format,NULL);
mlfFormat(format,precision);
```

MATLAB Syntax

```
format
format(format)
format(format,precision)
```

See Also

MATLAB `format` Calling Conventions

Purpose

Write formatted data to file

Minimum number of arguments: two; maximum number of input arguments: user-defined. Terminate the argument list with a NULL.

C Prototype

```
mxArray *mlfFprintf(mxArray *in1, mxArray *in2, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *format;                /* String array(s) */
mxArray *A;                     /* Required input argument(s) */
mxArray *fid;                   /* Optional input argument(s) */
mxArray *count = NULL, *R = NULL;/* Return value */

mlfAssign(&count, mlfFprintf(fid,format,A,..,NULL));
mlfAssign(&R, mlfFprintf(format,A,...,NULL));
```

MATLAB Syntax

```
count = fprintf(fid,format,A,...)
fprintf(format,A,...)
```

See Also

MATLAB `fprintf` Calling Conventions

mlfFread

Purpose

Read binary data from file

C Prototype

```
mxArray *mlfFread(mxArray **count, mxArray *fid, mxArray *size,
                  mxArray *precision, mxArray *skip);
```

C Syntax

```
#include "matlab.h"

mxArray *precision;         /* String array(s) */
mxArray *fid, *size;        /* Required input argument(s) */
mxArray *skip;              /* Optional input argument(s) */
mxArray *count = NULL;      /* Required output argument(s) */
mxArray *A = NULL;          /* Return value */

mlfAssign(&A, mlfFread(&count,fid,size,precision,NULL));
mlfAssign(&A, mlfFread(&count,fid,size,precision,skip));
```

MATLAB Syntax

```
[A,count] = fread(fid,size,precision)
[A,count] = fread(fid,size,precision,skip)
```

See Also

MATLAB `fread` Calling Conventions

Purpose

Determine frequency spacing for frequency response

C Prototype

```
mxArray *mlfFreqspace(mxArray **f2, mxArray *n, mxArray *whole_str);
```

C Syntax

```
#include "matlab.h"

mxArray *x;                 /* Dimension vector */
mxArray *n, *N;             /* Input argument(s) */
mxArray *f2 = NULL;         /* Optional output argument(s) */
mxArray *y1 = NULL;         /* Optional output argument(s) */
mxArray *f1 = NULL, *x1 = NULL, *f = NULL;/* Return value */

mlfAssign(&f1, mlfFreqspace(&f2,n,NULL));
mlfAssign(&f1, mlfFreqspace(&f2,mlfHorzcat(m,n,NULL),NULL));
mlfAssign(&x1, mlfFreqspace(&y1,n,mxCreateString("meshgrid")));
mlfAssign(&x1, mlfFreqspace(&y1,
          mlfHorzcat(m,n,NULL),mxCreateString("meshgrid")));
mlfAssign(&f, mlfFreqspace(NULL,N,NULL));
mlfAssign(&f, mlfFreqspace(NULL,N,mxCreateString("whole")));
```

MATLAB Syntax

```
[f1,f2] = freqspace(n)
[f1,f2] = freqspace([m n])
[x1,y1] = freqspace(...,'meshgrid')
f = freqspace(N)
f = freqspace(N,'whole')
```

See Also

MATLAB `freqspace` Calling Conventions

mlfFrewind

Purpose

Rewind an open file

C Prototype

```
mxArray *mlfFrewind(mxArray *fid);
```

C Syntax

```
#include "matlab.h"

mxArray *fid;           /* Required input argument(s) */
mxArray *R = NULL;      /* Return value */

mlfAssign(&R, mlfFrewind(fid));
```

MATLAB Syntax

```
frewind(fid)
```

See Also

MATLAB `frewind` Calling Conventions

Purpose

Read formatted data from file

C Prototype

```
mxArray *mlfFscanf(mxArray **count, mxArray *fid, mxArray *format,
                   mxArray *size);
```

C Syntax

```
#include "matlab.h"

mxArray *format;        /* String array(s) */
mxArray *fid;           /* Required input argument(s) */
mxArray *size;          /* Optional input argument(s) */
mxArray *count = NULL;  /* Optional output argument(s) */
mxArray *A = NULL;      /* Return value */

mlfAssign(&A, mlfFscanf(NULL,fid,format,NULL));
mlfAssign(&A, mlfFscanf(&count,fid,format,size));
```

MATLAB Syntax

```
A = fscanf(fid,format)
[A,count] = fscanf(fid,format,size)
```

See Also

MATLAB `fscanf` Calling Conventions

mlfFseek

Purpose

Set file position indicator

C Prototype

```
mxArray *mlfFseek(mxArray *fid, mxArray *offset, mxArray *origin);
```

C Syntax

```
#include "matlab.h"

mxArray *origin;            /* String array(s) */
mxArray *fid, *offset;      /* Required input argument(s) */
mxArray *status = NULL;     /* Return value */

mlfAssign(&status, mlfFseek(fid,offset,origin));
```

MATLAB Syntax

```
status = fseek(fid,offset,origin)
```

See Also

MATLAB `fseek` Calling Conventions

Purpose

Get file position indicator

C Prototype

```
mxArray *mlfFtell (mxArray *fid);
```

C Syntax

```
#include "matlab.h"

mxArray *fid;               /* Required input argument(s) */
mxArray *position = NULL;   /* Return value */

mlfAssign(&position, mlfFtell(fid));
```

MATLAB Syntax

```
position = ftell(fid)
```

See Also

MATLAB `ftell` Calling Conventions

mlfFull

Purpose

Convert sparse matrix to full matrix

C Prototype

```
mxArray *mlfFull(mxArray *S);
```

C Syntax

```
#include "matlab.h"

mxArray *S;                 /* Required input argument(s) */
mxArray *A = NULL;          /* Return value */

mlfAssign(&A, mlfFull(S));
```

MATLAB Syntax

```
A = full(S)
```

See Also

MATLAB `full` Calling Conventions

Purpose

Evaluate functions of a matrix

C Prototype

```
mxArray *mlfFunm(mxArray **esterr, mxArray *X, mxArray *function);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *esterr = NULL; /* Optional output argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfFunm(NULL,X,mxCreateString("function")));
mlfAssign(&Y, mlfFunm(&esterr,X,mxCreateString("function")));
```

MATLAB Syntax

```
Y = funm(X,'function')
[Y,esterr] = funm(X,'function')
```

See Also

MATLAB `funm` Calling Conventions

mlfFwrite

Purpose

Write binary data to a file

C Prototype

```
mxArray *mlfFwrite(mxArray *fid, mxArray *A, mxArray *precision,
                   mxArray *skip);
```

C Syntax

```
#include "matlab.h"

mxArray *precision;         /* String array(s) */
mxArray *fid, *A;           /* Required input argument(s) */
mxArray *skip;              /* Optional input argument(s) */
mxArray *count = NULL;      /* Return value */

mlfAssign(&count, mlfFwrite(fid,A,precision,NULL));
mlfAssign(&count, mlfFwrite(fid,A,precision,skip));
```

MATLAB Syntax

```
count = fwrite(fid,A,precision)
count = fwrite(fid,A,precision,skip)
```

See Also

MATLAB `fwrite` Calling Conventions

Purpose

Zero of a function of one variable

Minimum number of arguments: five; maximum number of arguments: user-defined. Terminate the argument list with a NULL.

C Prototype

```
mxArray *mlfFzero(mxArray **fval, mxArray **exitflag,
                  mxArray **output, mxArray *Func,
                  mxArray *x, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *func, *x0;             /* Required input argument(s) */
mxArray *options, *P1, *P2;     /* Optional input argument(s) */
mxArray *fval, *exitflag, *output;/* Optional output argument(s) */
mxArray *x = NULL;              /* Return value */

mlfAssign(&x, mlfFzero(NULL,NULL,NULL,func,x0,NULL));
mlfAssign(&x, mlfFzero(NULL,NULL,NULL,func,x0,options,NULL));
mlfAssign(&x, mlfFzero(NULL,NULL,NULL,
                       func,x,options,P1,P2,...,NULL));

mlfAssign(&x, mlfFzero(&fval,NULL,NULL,func,x0,NULL));
mlfAssign(&x, mlfFzero(&fval,NULL,NULL,func,x0,options,NULL));
mlfAssign(&x, mlfFzero(&fval,NULL,NULL,
                        func,x,options,P1,P2,...,NULL));

mlfAssign(&x, mlfFzero(&fval,&exitflag,NULL,func,x0,NULL));
mlfAssign(&x, mlfFzero(&fval,&exitflag,NULL,
                        func,x0,options,NULL));
mlfAssign(&x, mlfFzero(&fval,&exitflag,NULL,
                        func,x,options,P1,P2,...,NULL));

mlfAssign(&x, mlfFzero(&fval,&exitflag,&output,func,x0,NULL));
mlfAssign(&x, mlfFzero(&fval,&exitflag,&output,
                        func,x0,options,NULL));
mlfAssign(&x, mlfFzero(&fval,&exitflag,&output,
                        func,x,options,P1,P2,...,NULL));
```

mlfFzero

MATLAB Syntax

```
x = fzero(fun,x0)
x = fzero(fun,x0,options)
x = fzero(fun,x0,options,P1,P2,...)
[x,fval] = fzero(...)
[x,fval,exitflag] = fzero(...)
[x,fval,exitflag,output] = fzero(...)
```

See Also MATLAB `fzero` Calling Conventions

mlfGamma, mlfGammainc, mlfGammaln

Purpose

Gamma functions

C Prototype

```
mxArray *mlfGamma(mxArray *A, mxArray *DoNotUse);
mxArray *mlfGammainc(mxArray *X, mxArray *A);
mxArray *mlfGammaln(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *A, X;              /* Input argument(s) */
mxArray *Y = NULL;          /* Return value */

mlfAssign(&Y, mlfGamma(A,NULL));/* Gamma function accepts only */
                                /* one argument */
mlfAssign(&Y, mlfGammainc(X,A));/* Incomplete gamma function */
mlfAssign(&Y, mlfGammaln(A));   /* Logarithm of gamma function */
```

MATLAB Syntax

```
Y = gamma(A)                /* Gamma function */
Y = gammainc(X,A)           /* Incomplete gamma function */
Y = gammaln(A)              /* Logarithm of gamma function */
```

See Also

MATLAB `gamma`, `gammainc`, `gammaln`Calling Conventions

Purpose

Greatest common divisor

C Prototype

```
mxArray *mlfGcd(mxArray **C, mxArray **D, mxArray *A, mxArray *B);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *B;                 /* Required input argument(s) */
mxArray *C = NULL, *D = NULL;   /* Optional output argument(s) */
mxArray *G = NULL;              /* Return value */

mlfAssign(&G, mlfGcd(NULL,NULL,A,B));
mlfAssign(&G, mlfGcd(&C,&D,A,B));
```

MATLAB Syntax

```
G = gcd(A,B)
[G,C,D] = gcd(A,B)
```

See Also

MATLAB `gcd` Calling Conventions

mlfGetfield

Purpose

Get field of structure array

Minimum number of arguments: two, maximum: user-defined. Terminate the argument list with a `NULL`.

C Prototype

```
mxArray *mlfGetfield(mxArray *in1, mxArray *in2, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *s;                 /* Required input argument(s) */
mxArray *i, *j, *k;         /* Optional input argument(s) */
mxArray *f = NULL;          /* Return value */

mlfAssign(&f, mlfGetfield(s,mxCreateString("field"),NULL));
mlfAssign(&f, mlfGetfield(s,
                          mlfCellhcat(i,j,NULL),
                          mxCreateString("field"),
                          mlfCellhcat(k,NULL),
                          NULL));
```

MATLAB Syntax

```
f = getfield(s,'field')
f = getfield(s,{i,j},'field',{k})
```

See Also

MATLAB `getfield` Calling Conventions

Purpose

Generalized Minimum Residual method (with restarts)

Minimum number of arguments: twelve, maximum: user-defined. Terminate the argument list with a `NULL`.

C Prototype

```
mxArray *mlfGmres(mxArray **flag, mxArray **relres, mxArray **iter,
                  mxArray **resvec, mxArray *A, mxArray *b,
                  mxArray *restart, mxArray *tol, mxArray *maxit,
                  mxArray *M1, mxArray *M2, mxArray *x0,...);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *b, *restart;       /* Required input argument(s) */
mxArray *tol, *maxit;           /* Optional input argument(s) */
mxArray *M, *M1, M2, x0;        /* Optional input argument(s) */
mxArray *flag=NULL,*relres=NULL;/* Optional output argument(s) */
mxArray *iter=NULL,*resvec=NULL;/* Optional output argument(s) */
mxArray *x = NULL;              /* Return value */

mlfAssign(&x, mlfGmres(NULL,NULL,NULL,NULL,
                       A,b,restart,NULL,NULL,NULL,NULL,NULL,NULL));
mlfAssign(&x, mlfGmres(NULL,NULL,NULL,NULL,
                       A,b,restart,tol,NULL,NULL,NULL,NULL,NULL));
mlfAssign(&x, mlfGmres(NULL,NULL,NULL,NULL,
                       A,b,restart,tol,maxit,NULL,NULL,NULL,NULL));
mlfAssign(&x, mlfGmres(NULL,NULL,NULL,NULL,
                       A,b,restart,tol,maxit,M,NULL,NULL,NULL));
mlfAssign(&x, mlfGmres(NULL,NULL,NULL,NULL,
                       A,b,restart,tol,maxit,M1,M2,NULL,NULL));
mlfAssign(&x, mlfGmres(NULL,NULL,NULL,NULL,
                       A,b,restart,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfGmres(&flag,NULL,NULL,NULL,
                       A,b,restart,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfGmres(&flag,&relres,NULL,NULL,
                       A,b,restart,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfGmres(&flag,&relres,&iter,NULL,
                       A,b,restart,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfGmres(&flag,&relres,&iter,&resvec,
                       A,b,tol,restart,maxit,M1,M2,x0,NULL));
```

MATLAB Syntax

```
x = gmres(A,b,restart)
gmres(A,b,restart,tol)
gmres(A,b,restart,tol,maxit)
gmres(A,b,restart,tol,maxit,M)
gmres(A,b,restart,tol,maxit,M1,M2)
gmres(A,b,restart,tol,maxit,M1,M2,x0)
[x,flag] = gmres(A,b,restart,tol,maxit,M1,M2,x0)
[x,flag,relres] = gmres(A,b,restart,tol,maxit,M1,M2,x0)
[x,flag,relres,iter] = gmres(A,b,restart,tol,maxit,M1,M2,x0)
[x,flag,relres,iter,resvec] = gmres(A,b,restart,tol,maxit,M1,M2,x0)
```

See Also MATLAB `gmres` Calling Conventions

mlfGradient

Purpose

Numerical gradient

Minimum number of arguments: four, maximum: user-defined. Terminate the argument list with a `NULL`.

C Prototype

```
mxArray *mlfGradient(mlfVarargoutList *varargout, mxArray *f, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *F;             /* Required input argument(s) */
mxArray *h, *h1;        /* Optional input argument(s) */
mxArray *FY = NULL;     /* Optional output argument(s) */
mxArray *FX = NULL;     /* Return value */

mlfAssign(&FX, mlfGradient(NULL,F,NULL,NULL));
mlfAssign(&FX, mlfGradient(NULL,F,h,NULL));
mlfAssign(&FX, mlfGradient(&FY,F,NULL,NULL));
mlfAssign(&FX, mlfGradient(&FY,F,h,NULL));
mlfAssign(&FX, mlfGradient(&FY,F,h1,h2));

mlfGradient(mlfVarargout(&FX,NULL),F,NULL,NULL));
mlfGradient(mlfVarargout(&FX,&FY,NULL),F,h,NULL));
mlfGradient(mlfVarargout(&Fx,&Fy,&Fz,...,NULL),F,NULL,NULL));
mlfAssign(&FX, mlfGradient(mlfVarargout(&I,&J,NULL)&FY,F,h,NULL));
mlfAssign(&FX, mlfGradient(mlfVarargout(&I,&J,NULL)&FY,F,h1,h2));

mlfInd2sub(mlfVarargout(&I,&J,NULL),siz,IND);
mlfInd2sub(mlfVarargout(&I1,I2,I3,...,NULL),siz,IND);
```

Note With pure varargout functions, do not use the first output argument as the return value for the function. Pass all output arguments to `mlfVarargout()`. You do not need to use `mlfAssign()` to assign a return value.

MATLAB Syntax

```
FX = gradient(F)
[FX,FY] = gradient(F)
[Fx,Fy,Fz,...] = gradient(F)
[...] = gradient(F,h)
[...] = gradient(F,h1,h2,...)
```

See Also MATLAB `gradient` Calling Conventions

mlfGriddata

Purpose

Data gridding

C Prototype

```
mxArray *mlfGriddata(mxArray **YI, mxArray **ZI, mxArray *x,
                     mxArray *y, mxArray *z, mxArray *xi,
                     mxArray *yi);
```

C Syntax

```
#include "matlab.h"

mxArray *x, *y, *z;         /* Required input argument(s) */
mxArray *xi, *yi;           /* Optional input argument(s) */
mxArray *XI, *YI, *ZI;

mlfAssign(&ZI, mlfGriddata(NULL,NULL,x,y,z,XI,YI));
mlfAssign(&XI, mlfGriddata(&YI,&ZI,x,y,z,xi,yi));
```

MATLAB Syntax

```
ZI = griddata(x,y,z,XI,YI)
[XI,YI,ZI] = griddata(x,y,z,xi,yi)
[...] = griddata(...,method)
```

See Also

MATLAB `griddata` Calling Conventions

Purpose

Hadamard matrix

C Prototype

```
mxArray *mlfHadamard(mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *n;             /* Required input argument(s) */
mxArray *H = NULL;      /* Return value */

mlfAssign(&H, mlfHadamard(n));
```

MATLAB Syntax

```
H = hadamard(n)
```

See Also

MATLAB `hadamard` Calling Conventions

mlfHankel

Purpose

Hankel matrix

C Prototype

```
mxArray *mlfHankel(mxArray *c, mxArray *r);
```

C Syntax

```
#include "matlab.h"

mxArray *c;             /* Required input argument(s) */
mxArray *r;             /* Optional input argument(s) */
mxArray *H = NULL;      /* Return value */

mlfAssign(&H, mlfHankel(c,NULL));
mlfAssign(&H, mlfHankel(c,r));
```

MATLAB Syntax

```
H = hankel(c)
H = hankel(c,r)
```

See Also

MATLAB `hankel` Calling Conventions

Description

`H = hankel(c)` returns the square Hankel matrix whose first column is `c` and whose elements are zero below the first anti-diagonal.

`H = hankel(c,r)` returns a Hankel matrix whose first column is `c` and whose last row is `r`. If the last element of `c` differs from the first element of `r`, the last element of `c` prevails.

Purpose

Hessenberg form of a matrix

C Prototype

```
mxArray *mlfHess(mxArray **H, mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A;                      /* Required input argument(s) */
mxArray *H = NULL, *P = NULL;    /* Optional output argument
                                        and return value*/

mlfAssign(&P, mlfHess(&H,A));
mlfAssign(&H, mlfHess(NULL,A));
```

MATLAB Syntax

```
[P,H] = hess(A)
H = hess(A)
```

See Also

MATLAB hess Calling Conventions

mlfHex2dec

Purpose

IEEE hexadecimal to decimal number conversion

C Prototype

```
mxArray *mlfHex2dec(mxArray *hex_value);
```

C Syntax

```
#include "matlab.h"

mxArray *hex_value;     /* Hexadecimal integer or string array */
mxArray *d = NULL;      /* Return value */

mlfAssign(&d, mlfHex2dec(hex_value));
```

MATLAB Syntax

```
d = hex2dec('hex_value')
```

See Also

MATLAB `hex2dec` Calling Conventions

Description

`d = hex2dec('hex_value')` converts *hex_value* to its floating-point integer representation. The argument *hex_value* is a hexadecimal integer stored in a MATLAB string. If *hex_value* is a character array, each row is interpreted as a hexadecimal string.

Purpose

Hexadecimal to double precision number conversion

C Prototype

```
mxArray *mlfHex2num(mxArray *hex_value);
```

C Syntax

```
#include "matlab.h"

mxArray *hex_value;     /* String array(s) */
mxArray *f = NULL;      /* Return value */

mlfAssign(&f, mlfHex2num(hex_value));
```

MATLAB Syntax

```
f = hex2num('hex_value')
```

See Also

MATLAB `hex2num` Calling Conventions

mlfHilb

Purpose

Hilbert matrix

C Prototype

```
mxArray *mlfHilb(mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *n;             /* Required input argument(s) */
mxArray *H = NULL;      /* Return value */

mlfAssign(&H, mlfHilb(n));
```

MATLAB Syntax

```
H = hilb(n)
```

See Also

MATLAB `hilb` Calling Conventions

Description

`H = hilb(n)` returns the Hilbert matrix of order `n`.

Purpose

Horizontal concatenation.

Minimum number of arguments: one, maximum: user-defined. Terminate the argument list with a `NULL`.

C Prototype

```
mxArray *mlfHorzcat(mxArray *A, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *A,                 /* Required input argument(s) */
mxArray *B, *C;             /* Optional input argument(s) */
mxArray *R = NULL;          /* Return value */

mlfAssign(&R, mlfHorzcat(A,B,C,...,NULL));
```

MATLAB Syntax

```
[A,B,C...]
horzcat(A,B,C...)
```

See Also

MATLAB `cat` Calling Conventions

mlfI

Purpose

Imaginary unit

C Prototype

```
mxArray *mlfI(void);
```

C Syntax

```
#include "matlab.h"

mxArray *R = NULL;          /* Return value */

mlfAssign(&R, mlfI());
```

MATLAB Syntax

```
i
```

See Also

MATLAB `i` Calling Conventions

Purpose

One-dimensional cubic Interpolation

This MATLAB 4 function has been subsumed into `mlfInterp1`.

See Also

MATLAB `interp1` Calling Conventions

mlfIfft

Purpose

Inverse one-dimensional fast Fourier transform

C Prototype

```
mxArray *mlfIfft(mxArray *X, mxArray *n, mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *X;                     /* Required input argument(s) */
mxArray *n, *dim;               /* Optional input argument(s) */
mxArray *null_matrix = NULL;    /* Optional input argument(s) */
mxArray *y = NULL;              /* Return value */

mlfAssign(&y, mlfIfft(X,NULL,NULL));
mlfAssign(&y, mlfIfft(X,n,NULL));

mlfAssign(&null_matrix, mlfZeros(mlfScalar(0),mlfScalar(0),NULL));
mlfAssign(&y, mlfIfft(X,null_matrix,dim));

mlfAssign(&y, mlfIfft(X,n,dim));
```

MATLAB Syntax

```
y = ifft(X)
y = ifft(X,n)
y = ifft(X,[],dim)
y = ifft(X,n,dim)
```

See Also

MATLAB `ifft` Calling Conventions

Purpose

Inverse two-dimensional fast Fourier transform

C Prototype

```
mxArray *mlfIfft2(mxArray *X, mxArray *m, mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *m, *n;         /* Optional input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfIfft2(X,NULL,NULL));
mlfAssign(&Y, mlfIfft2(X,m,n));
```

MATLAB Syntax

```
Y = ifft2(X)
Y = ifft2(X,m,n)
```

See Also

MATLAB `ifft2` Calling Conventions

Description

`Y = ifft2(X)` returns the two-dimensional inverse fast Fourier transform of matrix `X`.

`Y = ifft2(X,m,n)` returns the `m`-by-`n` inverse fast Fourier transform of matrix `X`.

mlfIfftn

Purpose

Inverse multidimensional fast Fourier transform

C Prototype

```
mxArray *mlfIfftn(mxArray *X, mxArray * siz);
```

C Syntax

```
#include “matlab.h”

mxArray *X;             /* Required input argument(s) */
mxArray *siz;           /* Optional input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfIfftn(X,NULL));
mlfAssign(&Y, mlfIfftn(X,siz));
```

MATLAB Syntax

```
Y = ifftn(X)
Y = ifftn(X,siz)
```

See Also

MATLAB `ifftn` Calling Conventions

Purpose

Imaginary part of a complex number

C Prototype

```
mxArray *mlfImag(mxArray *Z);
```

C Syntax

```
#include "matlab.h"

mxArray *Z;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfImag(Z));
```

MATLAB Syntax

```
Y = imag(Z)
```

See Also

MATLAB `imag` Calling Conventions

mlfInd2sub

Purpose

Subscripts from linear index

C Prototype

```
mxArray *mlfInd2sub(mlfVarargoutList *varargout,
                    mxArray *siz,
                    mxArray *IND);
```

C Syntax

```
#include "matlab.h"

mxArray *siz,*IND;          /* Required input argument(s) */
mxArray *I,*J;
mxArray *I1,*I2,*I3;

mlfInd2sub(mlfVarargout(&I,&J,NULL),siz,IND);
mlfInd2sub(mlfVarargout(&I1,I2,I3,...,NULL),siz,IND);
```

Note With pure `varargout` functions, do not use the first output argument as the return value for the function. Pass all output arguments to `mlfVarargout()`. You do not need to use `mlfAssign()` to assign a return value.

MATLAB Syntax

```
[I,J] = ind2sub(siz,IND)
[I1,I2,I3,...,In] = ind2sub(siz,IND)
```

See Also

MATLAB `ind2sub` Calling Conventions

Purpose Infinity

C Prototype

```
mxArray *mlfInf();
```

C Syntax

```
#include "matlab.h"

mxArray *R = NULL;      /* Return value */

mlfAssign(&R, mlfInf());
```

MATLAB Syntax

```
Inf
```

See Also MATLAB `inf` Calling Conventions

mlfInpolygon

Purpose

Detect points inside a polygonal region

C Prototype

```
mxArray *mlfInpolygon(mxArray *X, mxArray *Y, mxArray *xv,
                      mxArray *yv);
```

C Syntax

```
#include "matlab.h"

mxArray *X, *Y, *xv, *yv;   /* Required input argument(s) */
mxArray *IN = NULL;         /* Return value */

mlfAssign(&IN, mlfInpolygon(X,Y,xv,yv));
```

MATLAB Syntax

```
IN = inpolygon(X,Y,xv,yv)
```

See Also

MATLAB `inpolygon` Calling Conventions

Purpose

Integer to string conversion

C Prototype

```
mxArray *mlfInt2str(mxArray *N);
```

C Syntax

```
#include "matlab.h"

mxArray *N;             /* Required input argument(s) */
mxArray *str = NULL;    /* Return value */

mlfAssign(&str, mlfInt2str(N));
```

MATLAB Syntax

```
str = int2str(N)
```

See Also

MATLAB `int2str` Calling Conventions

mlfInterp1

Purpose

One-dimensional data interpolation (table lookup)

C Prototype

```
mxArray *mlfInterp1(mxArray *x, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *x, *Y, *xi;    /* Required input argument(s) */
mxArray *method;        /* String array(s) */
mxArray *yi = NULL;     /* Return value */

mlfAssign(&yi, mlfInterp1(x,Y,xi,NULL));
mlfAssign(&yi, mlfInterp1(x,Y,xi,method,NULL));
```

MATLAB Syntax

```
yi = interp1(x,Y,xi)
yi = interp1(x,Y,xi,method)
```

See Also

MATLAB `interp1` Calling Conventions

Purpose

Quick one-dimensional linear interpolation

C Prototype

```
mxArray *mlfInterp1q(mxArray *x, mxArray *Y, mxArray *xi);
```

C Syntax

```
#include "matlab.h"

mxArray *x, *Y, *xi;    /* Required input argument(s) */
mxArray *F = NULL;      /* Return value */

mlfAssign(&F, mlfInterp1q(x,Y,xi));
```

MATLAB Syntax

```
F = interp1q(x,Y,xi)
```

See Also

MATLAB `interp1` Calling Conventions

mlfInterp2

Purpose

Two-dimensional data interpolation (table lookup)

C Prototype

```
mxArray *mlfInterp2(mxArray *X, mxArray *Y, mxArray *Z,
                    mxArray *XI, mxArray *YI, mxArray *method);
```

C Syntax

```
#include "matlab.h"

mxArray *method;            /* String array(s) */
mxArray *Z, *XI, *YI;       /* Input argument(s) */
mxArray *X, *Y, *ntimes;    /* Input argument(s) */
mxArray *ZI = NULL;         /* Return value */

mlfAssign(&ZI, mlfInterp2(X,Y,Z,XI,YI,NULL));
mlfAssign(&ZI, mlfInterp2(Z,XI,YI,NULL,NULL,NULL));
mlfAssign(&ZI, mlfInterp2(Z,ntimes,NULL,NULL,NULL,NULL));
mlfAssign(&ZI, mlfInterp2(X,Y,Z,XI,YI,method));
```

MATLAB Syntax

```
ZI = interp2(X,Y,Z,XI,YI)
ZI = interp2(Z,XI,YI)
ZI = interp2(Z,ntimes)
ZI = interp2(X,Y,Z,XI,YI,method)
```

See Also

MATLAB `interp2` Calling Conventions

Purpose

Two-dimensional bilinear data interpolation

This MATLAB 4 function has been subsumed by `mlfInterp2` in MATLAB 5.

See Also

MATLAB `interp2` Calling Conventions

mlfInterp5

Purpose Two-dimensional bicubic data interpolation

This MATLAB 4 function has been subsumed by `mlfInterp2` in MATLAB 5.

See Also MATLAB `interp2` Calling Conventions

Purpose

Two-dimensional nearest neighbor interpolation

This MATLAB 4 function has been subsumed by `mlfInterp2` in MATLAB 5.

See Also

MATLAB `interp2` Calling Conventions

mlfInterpft

Purpose

One-dimensional interpolation using the fast Fourier transform method

C Prototype

```
mxArray *mlfInterpft(mxArray *x, mxArray *n, mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *x, *n;         /* Required input argument(s) */
mxArray *dim;           /* Optional input argument(s) */
mxArray *y = NULL;      /* Return value */

mlfAssign(&y, mlfInterpft(x,n,NULL));
mlfAssign(&y, mlfInterpft(x,n,dim));
```

MATLAB Syntax

```
y = interpft(x,n)
y = interpft(x,n,dim)
```

See Also

MATLAB `interpft` Calling Conventions

Purpose

Set intersection of two vectors

C Prototype

```
mxArray *mlfIntersect(mxArray **ia, mxArray **ib, mxArray *a,
                      mxArray *b, mxArray *flag);
```

C Syntax

```
mlfAssign(&c, mlfIntersect(NULL,NULL,a,b,NULL));
mlfAssign(&c, mlfIntersect(NULL,NULL,A,B,mxCreateString("rows")));

mlfAssign(&c, mlfIntersect(&ia,&ib,a,b,NULL));
mlfAssign(&c, mlfIntersect(&ia,&ib,A,B,mxCreateString("rows")));
```

MATLAB Syntax

```
c = intersect(a,b)
c = intersect(A,B,'rows')
[c,ia,ib] = intersect(...)
```

See Also

MATLAB `intersect` Calling Conventions

mlfInv

Purpose

Matrix inverse

C Prototype

```
mxArray *mlfInv(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfInv(X));
```

MATLAB Syntax

```
Y = inv(X)
```

See Also

MATLAB `inv` Calling Conventions

Purpose

Inverse of the Hilbert matrix

C Prototype

```
mxArray *mlfInvhilb(mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *n;             /* Required input argument(s) */
mxArray *H = NULL;      /* Return value */

mlfAssign(&H, mlfInvhilb(n));
```

MATLAB Syntax

```
H = invhilb(n)
```

See Also

MATLAB `invhilb` Calling Conventions

mlfIpermute

Purpose

Inverse permute the dimensions of a multidimensional array

C Prototype

```
mxArray *mlfIpermute(mxArray *B, mxArray *order);
```

C Syntax

```
#include "matlab.h"

mxArray *B, *order;         /* Required input argument(s) */
mxArray *A = NULL;          /* Return value */

mlfAssign(&A, mlfIpermute(B,order));
```

MATLAB Syntax

```
A = ipermute(B,order)
```

See Also

MATLAB `ipermute` Calling Conventions

Purpose

Detect state

Minimum number of arguments for `mlfIsequal()`: one, maximum: user-defined. Terminate argument list with a `NULL`.

C Prototype

```
mxArray *mlfIscell(mxArray *C);
mxArray *mlfIscellstr(mxArray *s);
mxArray *mlfIschar(mxArray *A);
mxArray *mlfIsempty(mxArray *S);
mxArray *mlfIsequal(mxArray *A, mxArray *B, ...);
mxArray *mlfIsfield(mxArray *s, mxArray *f);
mxArray *mlfIsfinite(mxArray *A);
mxArray *mlfIsieee(void);
mxArray *mlfIsinf(mxArray *A);
mxArray *mlfIsletter(mxArray *str);
mxArray *mlfIslogical(mxArray *A);
mxArray *mlfIsnan(mxArray *A);
mxArray *mlfIsnumeric(mxArray *A);
mxArray *mlfIsprime(mxArray *A);
mxArray *mlfIsreal(mxArray *A);
mxArray *mlfIsspace(mxArray *str);
mxArray *mlfIssparse(mxArray *S);
mxArray *mlfIsstruct(mxArray *S);
mxArray *mlfIsstudent(void);
mxArray *mlfIsunix(void);
mxArray *mlfIsvms(void);
```

C Syntax

```
#include "matlab.h"

mxArray *str;                   /* String array(s) */
mxArray *A, *B, *C, *S;         /* Required input argument(s) */
mxArray *k = NULL, *TF = NULL;  /* Return value */

mlfAssign(&k, mlfIscell(C));
mlfAssign(&k, mlfIscellstr(S));
mlfAssign(&k, mlfIschar(S));
mlfAssign(&k, mlfIsieee());
mlfAssign(&k, mlfIsempty(A));
mlfAssign(&k, mlfIsequal(A,B,NULL)); // Terminate arg list with NULL
mlfAssign(&k, mlfIsfield(S,mxCreateString("field")));
mlfAssign(&TF, mlfIsfinite(A));
mlfAssign(&TF, mlfIsinf(A));
mlfAssign(&TF, mlfIsletter(mxCreateString("str")));
mlfAssign(&k, mlfIslogical(A));
mlfAssign(&TF, mlfIsnan(A));
mlfAssign(&k, mlfIsnumeric(A));
mlfAssign(&TF, mlfIsprime(A));
mlfAssign(&k, mlfIsreal(A));
mlfAssign(&TF, mlfIsspace(mxCreateString("str")));
mlfAssign(&k, mlfIssparse(S));
mlfAssign(&k, mlfIsstruct(S));
mlfAssign(&k, mlfIsstudent());
mlfAssign(&k, mlfIsunix());
mlfAssign(&k, mlfIsvms());
```

MATLAB Syntax

```
k = iscell(C)
k = isobject(A)
TF = ishandle(H)
k = ishold

k = iscellstr(S)
k = ischar(S)
k = isempty(A)
k = isequal(A,B,...)
k = isfield(S,'field')
k = isieee
TF = isfinite(A)
TF = isinf(A)
TF = isletter('str')
k = islogical(A)
k = isppc
k = issparse(S)
k = isglobal(NAME)

k = isnumeric(A)
TF = isnan(A)
TF = isprime(A)
k = isreal(A)
TF = isspace('str')
k = issparse(S)
k = isstruct(S)
k = isstudent
k = isunix
k = isvms
```

See Also

MATLAB is

Calling Conventions

mlfIsa

Purpose

Detect an object of a given class

C Prototype

```
mxArray *mlfIsa(mxArray *obj, mxArray *class_name);
```

C Syntax

```
#include "matlab.h"

mxArray class_name;         /* String array(s) */
mxArray obj;                /* Required input argument(s) */

mlfAssign(&K, mlfIsa(obj,class_name));
```

MATLAB Syntax

```
K = isa(obj,'class_name')
```

See Also

MATLAB `isa` Calling Conventions

Purpose

Detect members of a set

C Prototype

```
mxArray *mlfIsmember(mxArray *a, mxArray *S, mxArray *rows);
```

C Syntax

```
#include "matlab.h"

mxArray *a, *A, *S;         /* Input argument(s) */
mxArray *k = NULL;          /* Return value */

mlfAssign(&k, mlfIsmember(a,S,NULL));
mlfAssign(&k, mlfIsmember(A,S,mxCreateString("rows")));
```

MATLAB Syntax

```
k = ismember(a,S)
k = ismember(A,S,'rows')
```

See Also

MATLAB `ismember` Calling Conventions

mlfIsstr

Purpose

Detect strings

This MATLAB 4 function has been renamed `mlfIsChar` (`mlfIs*`) in MATLAB 5.

See Also

MATLAB `ischar` Calling Conventions

mlfJ

Purpose

Imaginary unit

C Prototype

```
mxArray *mlfJ(void);
```

C Syntax

```
#include "matlab.h"

mxArray *R = NULL;      /* Return value */

mlfAssign(&R, mlfJ());
```

MATLAB Syntax

```
j
```

See Also

MATLAB `j` Calling Conventions

Purpose

Kronecker tensor product

C Prototype

```
mxArray *mlfKron(mxArray *X, mxArray *Y);
```

C Syntax

```
#include "matlab.h"

mxArray *X, *Y;         /* Required input argument(s) */
mxArray *K = NULL;      /* Return value */

mlfAssign(&K, mlfKron(X,Y));
```

MATLAB Syntax

```
K = kron(X,Y)
```

See Also

MATLAB `kron` Calling Conventions

lasterr

Purpose

Last error message

C Prototype

```
mxArray *mlfLasterr(mxArray *msg);
```

C Syntax

```
include "matlab.h"

mxArray *msg;           /* Optional input argument(s) */
mxArray *str = NULL;    /* Return value */

mlfAssign(&str, mlfLasterr());
str = mlfLasterr(mxCreateString(""));
```

MATLAB Syntax

```
str = lasterr
lasterr('')
```

See Also

MATLAB `lasterr` Calling Conventions

Purpose

Least common multiple

C Prototype

```
mxArray *mlfLcm(mxArray *A, mxArray *B);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *B;         /* Required input argument(s) */
mxArray *L = NULL;      /* Return value */

mlfAssign(&L, mlfLcm(A,B));
```

MATLAB Syntax

```
L = lcm(A,B)
```

See Also

MATLAB `lcm` Calling Conventions

mlfLegendre

Purpose

Associated Legendre functions

C Prototype

```
mxArray *mlfLegendre(mxArray *n, mxArray *X, mxArray *sch_str);
```

C Syntax

```
#include "matlab.h"

mxArray *n, *X;                 /* Required input argument(s) */
mxArray *P = NULL, *S = NULL;   /* Return value */

mlfAssign(&P, mlfLegendre(n,X,NULL));
mlfAssign(&S, mlfLegendre(n,X,mxCreateString("sch")));
```

MATLAB Syntax

```
P = legendre(n,X)
S = legendre(n,X,'sch')
```

See Also

MATLAB `legendre` Calling Conventions

Purpose

Length of vector

C Prototype

```
mxArray *mlfLength(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *n = NULL;      /* Return value */

mlfAssign(&n, mlfLength(X));
```

MATLAB Syntax

```
n = length(X)
```

See Also

MATLAB `length` Calling Conventions

mlfLin2mu

Purpose

Linear to mu-law conversion

C Prototype

```
mxArray *mlfLin2mu(mxArray *y);
```

C Syntax

```
#include "matlab.h"

mxArray *y;                 /* Required input argument(s) */
mxArray *mu = NULL;         /* Return value */

mlfAssign(&mu, mlfLin2mu(y));
```

MATLAB Syntax

```
mu = lin2mu(y)
```

See Also

MATLAB `lin2mu` Calling Conventions

Purpose

Generate linearly spaced vectors

C Prototype

```
mxArray *mlfLinspace(mxArray *a, mxArray *b, mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *a, *b;         /* Required input argument(s) */
mxArray *n;             /* Optional input argument(s) */
mxArray *y = NULL;      /* Return value */

mlfAssign(&y, mlfLinspace(a,b,NULL));
mlfAssign(&y, mlfLinspace(a,b,n));
```

MATLAB Syntax

```
y = linspace(a,b)
y = linspace(a,b,n)
```

See Also

MATLAB `linspace` Calling Conventions

mlfLoad

Purpose

Load variables from disk.

Minimum number of arguments: one; maximum: user-defined. Terminate the argument list with NULL.

C Prototype

```
void mlfLoad(mxArray *fname, ... );
```

C Syntax

```
#include "matlab.h"

mxArray *fname;         /* Required input argument(s) */
mxArray *x, *y, *z;     /* Output arguments */

mlfLoad(fname,"X",&x,NULL));
mlfLoad(fname,"X",&x, /* Name/variable pairs */
              "Y",&y,
              "Z",&z,
              ...,
              NULL)); /* Terminate with a NULL */
```

Note mlfLoad() uses a nonstandard calling convention. You specify the arguments in pairs: the name of the variable whose value you want to load from disk and the address of a variable in which you want this value to be stored. Specify the variable to load is specified as a standard C char pointer. You can specify as many of these pairs as you want; terminate the argument list with a NULL.

MATLAB Syntax

```
load fname X
load fname X,Y,Z
load fname X,Y,Z...
```

See Also

MATLAB load Calling Conventions

Purpose

Natural logarithm

C Prototype

```
mxArray *mlfLog(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfLog(X));
```

MATLAB Syntax

```
Y = log(X)
```

See Also

MATLAB `log` Calling Conventions

mlfLog2

Purpose

Base 2 logarithm and dissect floating-point numbers into exponent and mantissa

C Prototype

```
mxArray *mlfLog2(mxArray **E, mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *E = NULL;      /* Optional output argument(s) */
mxArray *Y = NULL, *F = NULL;/* Return value */

mlfAssign(&Y, mlfLog2(NULL,X));
mlfAssign(&F, mlfLog2(&E,X));
```

MATLAB Syntax

```
Y = log2(X)
[F,E] = log2(X)
```

See Also

MATLAB `log2` Calling Conventions

Purpose

Common (base 10) logarithm

C Prototype

```
mxArray *mlfLog10(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfLog10(X));
```

MATLAB Syntax

```
Y = log10(X)
```

See Also

MATLAB `log10` Calling Conventions

mlfLogical

Purpose

Convert numeric values to logical

C Prototype

```
mxArray *mlfLogical(mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *K = NULL;      /* Return value */

mlfAssign(&K, mlfLogical(A));
```

MATLAB Syntax

```
K = logical(A)
```

See Also

MATLAB `logical` Calling Conventions

Purpose

Matrix logarithm

C Prototype

```
mxArray *mlfLogm(mxArray **esterr, mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *esterr = NULL; /* Optional output argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfLogm(NULL,X));
mlfAssign(&Y, mlfLogm(&esterr,X));
```

MATLAB Syntax

```
Y = logm(X)
[Y,esterr] = logm(X)
```

See Also

MATLAB `logm` Calling Conventions

mlfLogspace

Purpose

Generate logarithmically spaced vectors

C Prototype

```
mxArray *mlfLogspace(mxArray *a, mxArray *b, mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *a, *b, *n, *pi;/* Input argument(s) */
mxArray *y = NULL;      /* Return value */

mlfAssign(&y, mlfLogspace(a,b,NULL));
mlfAssign(&y, mlfLogspace(a,b,n));
mlfAssign(&y, mlfLogspace(a,pi,NULL));
```

MATLAB Syntax

```
y = logspace(a,b)
y = logspace(a,b,n)
y = logspace(a,pi)
```

See Also

MATLAB `logspace` Calling Conventions

Purpose

Convert string to lower case

C Prototype

```
mxArray *mlfLower(mxArray *str);
```

C Syntax

```
#include "matlab.h"

mxArray *str;           /* String array(s) */
mxArray *t = NULL;      /* Return value */

mlfAssign(&t, mlfLower(str));
```

MATLAB Syntax

```
t = lower('str')
```

See Also

MATLAB `lower` Calling Conventions

mlfLscov

Purpose

Least squares solution in the presence of known covariance

C Prototype

```
mxArray *mlfLscov(mxArray **dx, mxArray *A, mxArray *b, mxArray *V);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *b, *V;     /* Required input argument(s) */
mxArray *dx = NULL;     /* Optional output argument(s) */
mxArray *x = NULL;      /* Return value */

mlfAssign(&x, mlfLscov(NULL,A,b,V));
mlfAssign(&x, mlfLscov(&dx,A,b,V));
```

MATLAB Syntax

```
x = lscov(A,b,V)
[x,dx] = lscov(A,b,V)
```

See Also

MATLAB `lscov` Calling Conventions

Purpose

Linear least squares with nonnegativity constraints

C Prototype

```
mxArray *mlfLsqnonneg(mxArray **resnorm, mxArray **resid,
                      mxArray **exitflag, mxArray **output,
                      mxArray **lambda, mxArray *C,
                      mxArray *d, mxArray *x0,
                      mxArray *options);
```

mlfLsqnonneg

C Syntax

```
#include "matlab.h"

mxArray *C, *d;         /* Required input argument(s) */
mxArray *x0, *options;  /* Optional input argument(s) */
mxArray *resnorm = NULL;/* Optional output argument(s) */
mxArray *residual = NULL;/* Optional output argument(s) */
mxArray *exitflag = NULL;/* Optional output argument(s) */
mxArray *output = NULL; /* Optional output argument(s) */
mxArray *lambda = NULL; /* Optional output argument(s) */
mxArray *x = NULL;      /* Return value */

/* MATLAB syntax: x = lsqnonneg(C,d) */
mlfAssign(&x,
  mlfLsqnonneg(NULL,NULL,NULL,NULL,NULL,C,d,NULL,NULL));

/* MATLAB syntax: x = lsqnonneg(C,d,x0) */
mlfAssign(&x,
  mlfLsqnonneg(NULL,NULL,NULL,NULL,NULL,C,d,x0,NULL));

/* MATLAB syntax: x = lsqnonneg(C,d,x0,options) */
mlfAssign(&x,
  mlfLsqnonneg(NULL,NULL,NULL,NULL,NULL,C,d,x0,options));

/* MATLAB syntax: [x, resnorm ] = lsqnonneg(...) */
mlfAssign(&x,
  mlfLsqnonneg(resnorm,NULL,NULL,NULL,NULL,C,d,NULL,NULL));
mlfAssign(&x,
  mlfLsqnonneg(resnorm,NULL,NULL,NULL,NULL,C,d,x0,NULL));
mlfAssign(&x,
  mlfLsqnonneg(resnorm,NULL,NULL,NULL,NULL,C,d,x0,options));

/* MATLAB syntax: [x, resnorm, residual ] = lsqnonneg(...) */
mlfAssign(&x,
  mlfLsqnonneg(resnorm,residual,NULL,NULL,NULL,C,d,NULL,NULL));
mlfAssign(&x,
  mlfLsqnonneg(resnorm,residual,NULL,NULL,NULL,C,d,x0,NULL));
mlfAssign(&x,
```

```
   mlfLsqnonneg(resnorm,residual,NULL,NULL,NULL,C,d,x0,options));

/* MATLAB syntax: [x,resnorm,residual,exitflag] = lsqnonneg(...) */
mlfAssign(&x,
  mlfLsqnonneg(resnorm,residual,exitflag,NULL,NULL,C,d,NULL,
               NULL));
mlfAssign(&x,
  mlfLsqnonneg(resnorm,residual,exitflag,NULL,NULL,C,d,x0,NULL));
mlfAssign(&x,
  mlfLsqnonneg(resnorm,residual,exitflag,NULL,NULL,C,d,x0,
               options));

/* MATLAB: [x,resnorm,residual,exitflag,output] = lsqnonneg(...) *
mlfAssign(&x,
   mlfLsqnonneg(resnorm,residual,exitflag,output,NULL,C,d,NULL,
                NULL));
mlfAssign(&x,
   mlfLsqnonneg(resnorm,residual,exitflag,output,NULL,C,d,x0,
                NULL));
mlfAssign(&x,
   mlfLsqnonneg(resnorm,residual,exitflag,output,NULL,C,d,x0,
                options));

/* [x,resnorm,residual,exitflag,output,lambda] = lsqnonneg(...) */
mlfAssign(&x,
   mlfLsqnonneg(resnorm,residual,exitflag,output,lambda,C,d,NULL,
                NULL));
mlfAssign(&x,
   mlfLsqnonneg(resnorm,residual,exitflag,output,lambda,C,d,x0,
                NULL));
mlfAssign(&x,
   mlfLsqnonneg(resnorm,residual,exitflag,output,lambda,C,d,x0,
                options));
```

mlfLsqnonneg

MATLAB Syntax

```
x = lsqnonneg(C,d)
x = lsqnonneg(C,d,x0)
x = lsqnonneg(C,d,x0,options)
[x,resnorm] = lsqnonneg(...)
[x,resnorm,residual] = lsqnonneg(...)
[x,resnorm,residual,exitflag] = lsqnonneg(...)
[x,resnorm,residual,exitflag,output] = lsqnonneg(...)
[x,resnorm,residual,exitflag,output,lambda] = lsqnonneg(...)
```

See Also MATLAB `lsqnonneg` Calling Conventions

Purpose

LU matrix factorization

C Prototype

```
extern mxArray *mlfLu(mxArray **U, mxArray **P,
                      mxArray *X, mxArray *thresh);
```

C Syntax

```
#include "matlab.h"

mxArray *X;                     /* Required input argument(s) */
mxArray *thresh;                /* Optional input argument(s) */
mxArray *U = NULL, *P = NULL;   /* Optional output argument(s) */
mxArray *L = NULL;              /* Return value */

mlfAssign(&L, mlfLu(&U,NULL,X,NULL));
mlfAssign(&L, mlfLu(&U,&P,X,NULL));
mlfAssign(&L, mlfLu(NULL,NULL,X,NULL));
mlfAssign(&L, mlfLu(NULL,NULL,X,thresh));
```

MATLAB Syntax

```
[L,U] = lu(X)
[L,U,P] = lu(X)
lu(X)
lu(X,thresh)
```

See Also

MATLAB `lu` Calling Conventions

mlfLuinc

Purpose

Incomplete LU matrix factorizations

C Prototype

```
mxArray *mlfLuinc(mxArray **U,
                  mxArray **P,
                  mxArray *X,
                  mxArray *droptol);
```

C Syntax

```
#include "matlab.h"

mxArray *X;                 /* Required input argument(s) */
mxArray *droptol, *options; /* Optional input argument(s) */
mxArray *U = NULL, P = NULL;/* Optional output argument(s) */
mxArray *L = NULL;          /* Return value */

mlfAssign(&L, mlfLuinc(NULL,NULL,X,mxCreateString("0")));
mlfAssign(&L, mlfLuinc(&U,NULL,X,mxCreateString("0")));
mlfAssign(&L, mlfLuinc(&U,&P,X,mxCreateString("0")));
mlfAssign(&L, mlfLuinc(NULL,NULL,X,droptol));
mlfAssign(&L, mlfLuinc(NULL,NULL,X,options));
mlfAssign(&L, mlfLuinc(&U,NULL,X,options));
mlfAssign(&L, mlfLuinc(&U,NULL,X,droptol));
mlfAssign(&L, mlfLuinc(&U,&P,X,options));
mlfAssign(&L, mlfLuinc(&U,&P,X,droptol));
```

MATLAB Syntax

```
luinc(X,'0')
[L,U] = luinc(X,'0')
[L,U,P] = luinc(X,'0')
luinc(X,droptol)
luinc(X,options)
[L,U] = luinc(X,options)
[L,U] = luinc(X,droptol)
[L,U,P] = luinc(X,options)
[L,U,P] = luinc(X,droptol)
```

See Also

MATLAB `luinc` Calling Conventions

Purpose

Magic square

C Prototype

```
mxArray *mlfMagic(mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *n;             /* Required input argument(s) */
mxArray *M = NULL;      /* Return value */

mlfAssign(&M, mlfMagic(n));
```

MATLAB Syntax

```
M = magic(n)
```

See Also

MATLAB `magic` Calling Conventions

mlfMat2str

Purpose

Convert a matrix into a string

C Prototype

```
mxArray *mlfMat2str(mxArray *A, mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *n;             /* Optional input argument(s) */
mxArray *str = NULL;    /* Return value */

mlfAssign(&str, mlfMat2str(A,NULL));
mlfAssign(&str, mlfMat2str(A,n));
```

MATLAB Syntax

```
str = mat2str(A)
str = mat2str(A,n)
```

See Also

MATLAB `mat2str` Calling Conventions

Purpose

Maximum elements of an array

C Prototype

```
mxArray *mlfMax(mxArray **I, mxArray *A, mxArray *B, mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *B, *dim;       /* Optional input argument(s) */
mxArray *I = NULL;      /* Optional output argument(s) */
mxArray *C = NULL;      /* Return value */

mlfAssign(&C, mlfMax(NULL,A,NULL,NULL));
mlfAssign(&C, mlfMax(NULL,A,B,NULL));
mlfAssign(&C, mlfMax(NULL,
                     A,
                     mlfZeros(mlfScalar(0),NULL),
                     dim));

mlfAssign(&C, mlfMax(&I,A,NULL,NULL));
mlfAssign(&C, mlfMax(&I,A,B,NULL));
mlfAssign(&C, mlfMax(&I,
                     A,
                     mlfZeros(mlfScalar(0),NULL),
                     dim));
```

MATLAB Syntax

```
C = max(A)
C = max(A,B)
C = max(A,[],dim)
[C,I] = max(...)
```

See Also

MATLAB `max` Calling Conventions

mlfMean

Purpose

Average or mean value of arrays

C Prototype

```
mxArray *mlfMean(mxArray *A, mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *dim;           /* Optional input argument(s) */
mxArray *M = NULL;      /* Return value */

mlfAssign(&M, mlfMean(A,NULL));
mlfAssign(&M, mlfMean(A,dim));
```

MATLAB Syntax

```
M = mean(A)
M = mean(A,dim)
```

See Also

MATLAB `mean` Calling Conventions

Purpose

Median value of arrays

C Prototype

```
mxArray *mlfMedian(mxArray *A, mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *dim;           /* Optional input argument(s) */
mxArray *M = NULL;      /* Return value */

mlfAssign(&M, mlfMedian(A,NULL));
mlfAssign(&M, mlfMedian(A,dim));
```

MATLAB Syntax

```
M = median(A)
M = median(A,dim)
```

See Also

MATLAB `median` Calling Conventions

mlfMeshgrid

Purpose

Generate X and Y matrices for three-dimensional plots

C Prototype

```
mxArray *mlfMeshgrid(mxArray **Y, mxArray **Z, mxArray *x,
                     mxArray *y, mxArray *z);
```

C Syntax

```
#include "matlab.h"

mxArray *x;             /* Required input argument(s) */
mxArray *y, *z;         /* Optional input argument(s) */
mxArray *Y;             /* Required output argument(s) */
mxArray *Z;             /* Optional output argument(s) */
mxArray *X = NULL;      /* Return value */

mlfAssign(&X, mlfMeshgrid(&Y,NULL,x,y,NULL));
mlfAssign(&X, mlfMeshgrid(&Y,NULL,x,NULL,NULL));
mlfAssign(&X, mlfMeshgrid(&Y,&Z,x,y,z));
```

MATLAB Syntax

```
[X,Y] = meshgrid(x,y)
[X,Y] = meshgrid(x)
[X,Y,Z] = meshgrid(x,y,z)
```

See Also

MATLAB meshgrid Calling Conventions

Purpose

The name of the currently running M-file

C Prototype

```
mxArray *mlfMfilename();
```

C Syntax

```
#include "matlab.h"

mxArray *R = NULL;      /* Return value */

mlfAssign(&R, mlfMfilename());
```

MATLAB Syntax

```
mfilename
```

See Also

MATLAB `mfilename` Calling Conventions

mlfMin

Purpose

Minimum elements of an array

C Prototype

```
mxArray *mlfMin(mxArray **I, mxArray *A, mxArray *B, mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *B, *dim;       /* Optional input argument(s) */
mxArray *I;             /* Optional output argument(s) */
mxArray *C = NULL;      /* Return value */

mlfAssign(&C, mlfMin(NULL,A,NULL,NULL));
mlfAssign(&C, mlfMin(NULL,A,B,NULL));
mlfAssign(&C, mlfMin(NULL,
                     A,
                     mlfZeros(mlfScalar(0),NULL),
                     dim));

mlfAssign(&C, mlfMin(&I,A,NULL,NULL));
mlfAssign(&C, mlfMin(&I,A,B,NULL));
mlfAssign(&C, mlfMin(&I,
                     A,
                     mlfZeros(mlfScalar(0),NULL),
                     dim));
```

MATLAB Syntax

```
C = min(A)
C = min(A,B)
C = min(A,[],dim)
[C,I] = min(...)
```

See Also

MATLAB `min` Calling Conventions

Purpose

Modulus (signed remainder after division)

C Prototype

```
mxArray *mlfMod(mxArray *X, mxArray *Y);
```

C Syntax

```
#include "matlab.h"

mxArray *X, *Y;         /* Required input argument(s) */
mxArray *M = NULL;      /* Return value */

mlfAssign(&M, mlfMod(X,Y));
```

MATLAB Syntax

```
M = mod(X,Y)
```

See Also

MATLAB `mod` Calling Conventions

mlfMu2lin

Purpose

Mu-law to linear conversion

C Prototype

```
mxArray *mlfMu2lin(mxArray *mu);
```

C Syntax

```
#include "matlab.h"

mxArray *mu;                /* Required input argument(s) */
mxArray *y = NULL;          /* Return value */

mlfAssign(&y, mlfMu2lin(mu));
```

MATLAB Syntax

```
y = mu2lin(mu)
```

See Also

MATLAB `mu2lin` Calling Conventions

Purpose Not-a-Number

C Prototype

```
mxArray *mlfNan();
```

C Syntax

```
#include "matlab.h"

mxArray *R = NULL;      /* Return value */

mlfAssign(&R, mlfNan());
```

MATLAB Syntax

```
NaN
```

See Also MATLAB `NaN` Calling Conventions

mlfNargchk

Purpose

Check number of input arguments

C Prototype

```
mxArray *mlfNargchk(mxArray *low, mxArray *high, mxArray *number);
```

C Syntax

```
#include "matlab.h"

mxArray *low, *high, *number;   /* Required input argument(s) */
mxArray *msg = NULL;            /* Return value */

mlfAssign(&msg, mlfNargchk(low,high,number));
```

MATLAB Syntax

```
msg = nargchk(low,high,number)
```

See Also

MATLAB nargchk Calling Conventions

Purpose

All combinations of the *n* elements in v taken *k* at a time

C Prototype

```
mxArray *mlfNchoosek(mxArray *v, mxArray *k);
```

C Syntax

```
#include "matlab.h"

mxArray *v;             /* Required input vector of length n */
mxArray *k;             /* Required input scalar, group size */
mxArray *C = NULL;      /* Output array of combinations */

mlfAssign(&C, mlfNchoosek(v,k));
```

MATLAB Syntax

```
C = nchoosek(v,k)
```

See Also

MATLAB nchoosek Calling Conventions

mlfNdims

Purpose

Number of array dimensions

C Prototype

```
mxArray *mlfNdims(mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *n = NULL;      /* Return value */

mlfAssign(&n, mlfNdims(A));
```

MATLAB Syntax

```
n = ndims(A)
```

See Also

MATLAB `ndims` Calling Conventions

Purpose

Next power of two

C Prototype

```
mxArray *mlfNextpow2(mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *p = NULL;      /* Return value */

mlfAssign(&p, mlfNextpow2(A));
```

MATLAB Syntax

```
p = nextpow2(A)
```

See Also

MATLAB `nextpow2` Calling Conventions

mlfNnls

Purpose

Nonnegative least squares

Note The `mlfNnls` routine was replaced by `mlfLsqnonneg` in Release 11 (MATLAB 5.3). In Release 12 (MATLAB 6.0), `mlfNnls` displays a warning and calls `mlfLsnonneg`.

C Prototype

```
mxArray *mlfNnls(mxArray **w, mxArray *A, mxArray *b, mxArray *tol);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *b;         /* Required input argument(s) */
mxArray *tol;           /* Optional input argument(s) */
mxArray *w = NULL;      /* Optional output argument(s) */
mxArray *x = NULL;      /* Return value */

mlfAssign(&x, mlfNnls(NULL,A,b,NULL));
mlfAssign(&x, mlfNnls(NULL,A,b,tol));
mlfAssign(&x, mlfNnls(&w,A,b,NULL));
mlfAssign(&x, mlfNnls(&w,A,b,tol));
```

MATLAB Syntax

```
x = nnls(A,b)
x = nnls(A,b,tol)
[x,w] = nnls(A,b)
[x,w] = nnls(A,b,tol)
```

See Also

MATLAB `nnls` Calling Conventions

Purpose

Number of nonzero matrix elements

C Prototype

```
mxArray *mlfNnz(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;                 /* Required input argument(s) */
mxArray *n = NULL;          /* Return value */

mlfAssign(&n, mlfNnz(X));
```

MATLAB Syntax

```
n = nnz(X)
```

See Also

MATLAB `nnz` Calling Conventions

mlfNonzeros

Purpose

Nonzero matrix elements

C Prototype

```
mxArray *mlfNonzeros(mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A;                 /* Required input argument(s) */
mxArray *s = NULL;          /* Return value */

mlfAssign(&s, mlfNonzeros(A));
```

MATLAB Syntax

```
s = nonzeros(A)
```

See Also

MATLAB nonzeros Calling Conventions

Purpose

Vector and matrix norms

C Prototype

```
mxArray *mlfNorm(mxArray *A, mxArray *p);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *p;             /* Optional input argument(s) */
mxArray *n = NULL;      /* Return value */

mlfAssign(&n, mlfNorm(A,NULL));
mlfAssign(&n, mlfNorm(A,p));
```

MATLAB Syntax

```
n = norm(A)
n = norm(A,p)
```

See Also

MATLAB `norm` Calling Conventions

mlfNormest

Purpose

Two-norm estimate

C Prototype

```
mxArray *mlfNormest(mxArray **count, mxArray *S, mxArray *tol);
```

C Syntax

```
#include "matlab.h"

mxArray *S;             /* Required input argument(s) */
mxArray *tol;           /* Optional input argument(s) */
mxArray *count = NULL;  /* Optional output argument(s) */
mxArray *nrm = NULL;    /* Return value */

mlfAssign(&nrm, mlfNormest(NULL,S,NULL));
mlfAssign(&nrm, mlfNormest(NULL,S,tol));
mlfAssign(&nrm, mlfNormest(&count,S,NULL));
mlfAssign(&nrm, mlfNormest(&count,S,tol));
```

MATLAB Syntax

```
nrm = normest(S)
nrm = normest(S,tol)
[nrm,count] = normest(...)
```

See Also

MATLAB `normest` Calling Conventions

Purpose

Current date and time

C Prototype

```
mxArray *mlfNow();
```

C Syntax

```
#include "matlab.h"

mxArray *t = NULL;      /* Return value */

mlfAssign(&t, now());
```

MATLAB Syntax

```
t = now
```

See Also

MATLAB now

Calling Conventions

mlfNull

Purpose

Null space of a matrix

C Prototype

```
mxArray *mlfNull(mxArray *A, mxArray *how);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *how;           /* Optional input argument(s) */
mxArray *B = NULL;      /* Return value */

mlfAssign(&B, mlfNull(A,NULL));
mlfAssign(&B, mlfNull(A,how));
```

MATLAB Syntax

```
B = null(A)
```

See Also

MATLAB `null` Calling Conventions

Purpose

Convert a numeric array into a cell array

C Prototype

```
mxArray *mlfNum2cell(mxArray *A, mxArray *dims);
```

C Syntax

```
#include "matlab.h"

mxArray *A;                 /* Required input argument(s) */
mxArray *dims;              /* Optional input argument(s) */
mxArray *c = NULL;          /* Return value */

mlfAssign(&c, mlfNum2cell(A,NULL));
mlfAssign(&c, mlfNum2cell(A,dims));
```

MATLAB Syntax

```
c = num2cell(A)
c = num2cell(A,dims)
```

See Also

MATLAB `num2cell` Calling Conventions

mlfNum2str

Purpose

Number to string conversion

C Prototype

```
mxArray *mlfNum2str(mxArray *A, mxArray *format);
```

C Syntax

```
#include "matlab.h"

mxArray *format;            /* String array(s) */
mxArray *A;                 /* Required input argument(s) */
mxArray *precision;         /* Optional input argument(s) */
mxArray *str;               /* Return value */

mlfAssign(&str, mlfNum2str(A,NULL));
mlfAssign(&str, mlfNum2str(A,precision));
mlfAssign(&str, mlfNum2str(A,format));
```

MATLAB Syntax

```
str = num2str(A)
str = num2str(A,precision)
str = num2str(A,format)
```

See Also

MATLAB `num2str` Calling Conventions

Purpose

Amount of storage allocated for nonzero matrix elements

C Prototype

```
mxArray *mlfNzmax(mxArray *S);
```

C Syntax

```
#include "matlab.h"

mxArray *S;                 /* Required input argument(s) */
mxArray *n = NULL;          /* Return value */

mlfAssign(&n, mlfNzmax(S));
```

MATLAB Syntax

```
n = nzmax(S)
```

See Also

MATLAB `nzmax` Calling Conventions

mlfOde45, mlfOde23, mlfOde113, mlfOde15s, mlfOde23s

Purpose

Solve differential equations

Minimum number of arguments: six; maximum number: user-defined. Terminate the argument list with a `NULL`.

C Prototype

Substitute `mlfOde45`, `mlfOde23`, etc., for *solver*.

```
mxArray *solver(mxArray **yout, mlfVarargoutList *varargout,
                  mxArray *odefile, mxArray *tspan, mxArray *y0,
                  mxArray *options, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *func;                  /* String array(s) */
mxArray *tspan, *y0, *options;  /* Input argument(s) */
mxArray *p1, *p2;               /* Optional input argument(s) */
mxArray *Y=NULL;                /* Output arguments */
mxArray *TE=NULL, *YE=NULL, *IE=NULL;   /* Output arguments */
mxArray *T = Null;              /* Return value */

mlfAssign(&T, solver(&Y,mlfVarargout(NULL),
                         func,tspan,y0,NULL,NULL));
mlfAssign(&T, solver(&Y,mlfVarargout(NULL),
                         func,tspan,y0,options,NULL));
mlfAssign(&T, solver(&Y,mlfVarargout(NULL),
                         func,tspan,y0,options,p1,p2,...,NULL));
mlfAssign(&T, solver(&Y,mlfVarargout(&TE,&YE,&IE,NULL),
                         func,tspan,y0,options,NULL));
```

MATLAB Syntax

```
[T,Y] = solver('F',tspan,y0)
[T,Y] = solver('F',tspan,y0,options)
[T,Y] = solver('F',tspan,y0,options,p1,p2...)
[T,Y,TE,YE,IE] = solver('F',tspan,y0,options)
```

See Also

MATLAB `ode45`, `ode23`, `ode113`, `ode15s`, `ode23s`Calling Conventions

Purpose

Extract properties from options structure created with odeset

C Prototype

```
mxArray *mlfOdeget(mxArray *options, mxArray *name_str,
                   mxArray *default);
```

C Syntax

```
#include "matlab.h"

mxArray *name_str;          /* String array(s) */
mxArray *options;           /* Required input argument(s) */
mxArray *default;           /* Optional input argument(s) */
mxArray *o = NULL;          /* Return value */

mlfAssign(&o, mlfOdeget(options,name_str,NULL));
mlfAssign(&o, mlfOdeget(options,name_str,default));
```

MATLAB Syntax

```
o = odeget(options,'name')
o = odeget(options,'name',default)
```

See Also

MATLAB odeget Calling Conventions

mlfOdeset

Purpose

Create or alter options structure for input to ODE solvers

Minimum number of arguments: one; maximum number: user-defined. Terminate the argument list with a NULL.

C Prototype

```
mxArray *mlfOdeset(mxArray *name1, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *name1, *name2;         /* String array(s) */
mxArray *value1, *value2;       /* Input values */
mxArray *oldopts, *newopts;     /* Input argument(s) */
mxArray *options = NULL;        /* Return value */

mlfAssign(&options, mlfOdeset(name1,value1,
                              name2,value2,...,NULL));
mlfAssign(&options, mlfOdeset(oldopts,name1,value1,...,NULL));
mlfAssign(&options, mlfOdeset(oldopts,newopts,NULL));
mlfAssign(&options, mlfOdeset(NULL));
```

MATLAB Syntax

```
options = odeset('name1',value1,'name2',value2,...)
options = odeset(oldopts,'name1',value1,...)
options = odeset(oldopts,newopts)
odeset
```

See Also

MATLAB odeset Calling Conventions

Purpose

Create an array of all ones

Minimum number of arguments: one; maximum number: user-defined. Terminate the argument list with a `NULL`.

C Prototype

```
mxArray *mlfOnes(mxArray *n, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *m, *n;             /* Input argument(s) */
mxArray *d1, *d2, *d3;      /* Input argument(s) */
mxArray *A;                 /* Input argument(s) */
mxArray *Y = NULL;          /* Return value */

mlfAssign(&Y, mlfOnes(n,NULL));
mlfAssign(&Y, mlfOnes(m,n,NULL));
mlfAssign(&Y, mlfOnes(mlfHorzcat(m,n,NULL),NULL));
mlfAssign(&Y, mlfOnes(d1,d2,d3,...,NULL));
mlfAssign(&Y, mlfOnes(mlfHorzcat(d1,d2,d3,...,NULL),NULL));
mlfAssign(&Y, mlfOnes(mlfSize(NULL,A,NULL),NULL));
```

MATLAB Syntax

```
Y = ones(n)
Y = ones(m,n)
Y = ones([m n])
Y = ones(d1,d2,d3...)
Y = ones([d1 d2 d3...])
Y = ones(size(A))
```

See Also

MATLAB ones Calling Conventions

mlfOptimget

Purpose

Get optimization options structure parameter values

C Prototype

```
mxArray *mlfOptimget(mxArray *options, mxArray *name,
                     mxArray *default);
```

C Syntax

```
#include "matlab.h"

mxArray *options;           /* Input argument(s) */
mxArray *default;           /* Input argument(s) */
mxArray *val = NULL;        /* Return value */

mlfAssign(&val, mlfOptimget(options,NULL,NULL));
mlfAssign(&val, mlfOptimget(options,
                            mxCreateString("param"),NULL));
mlfAssign(&val, mlfOptimget(options,
                            mxCreateString("param"),default));
```

MATLAB Syntax

```
val = optimget(options,'param')
val = optimget(options,'param',default)
```

See Also

fminbnd, MATLAB optimset, MATLAB fminsearch, MATLAB fzero, MATLAB lsqnonneg

Purpose

Create or edit optimization options parameter structure

Minimum number of arguments: one; maximum number: user-defined. Terminate the argument list with a NULL.

C Prototype

```
mxArray *mlfOptimset(mxArray *param1, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *param1, *param2;   /* String array(s) */
mxArray *value1, *value2;   /* Input argument(s) */
mxArray *optimfun;          /* Input argument(s) */
mxArray *oldopts, *newopts; /* Input argument(s) */
mxArray *options = NULL;    /* Return value */

mlfAssign(&options, mlfOptimset(param1,value1,
                                param2,value2,...,NULL));
mlfOptimset(NULL);
mlfAssign(&options, mlfOptimset(NULL));
mlfAssign(&options, mlfOptimset(optimfun,NULL));
mlfAssign(&options, mlfOptimset(oldopts,param1,value1,...,NULL));
mlfAssign(&options, mlfOptimset(oldopts,newopts,NULL));
```

MATLAB Syntax

```
options = optimset('param1',value1,'param2',value2,...)
optimset
options = optimset
options = optimset(optimfun)
options = optimset(oldopts,'param1',value1,...)
options = optimset(oldopts,newopts)
```

See Also

fminbnd, MATLAB optimget, MATLAB fminsearch, MATLAB fzero, MATLAB lsqnonneg

mlfOrth

Purpose

Range space of a matrix

C Prototype

```
mxArray *mlfOrth(mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *B = NULL;      /* Return value */

mlfAssign(&B, mlfOrth(A));
```

MATLAB Syntax

```
B = orth(A)
```

See Also

MATLAB `orth` Calling Conventions

mlfPascal

Purpose

Pascal matrix

C Prototype

```
mxArray *mlfPascal(mxArray *n, mxArray *k);
```

C Syntax

```
#include "matlab.h"

mxArray *n;             /* Required input argument(s) */
mxArray *A = NULL;      /* Return value */

mlfAssign(&A, mlfPascal(n,NULL));
mlfAssign(&A, mlfPascal(n,mlfScalar(1)));
mlfAssign(&A, mlfPascal(n,mlfScalar(2)));
```

MATLAB Syntax

```
A = pascal(n)
A = pascal(n,1)
A = pascal(n,2)
```

See Also

MATLAB `pascal` Calling Conventions

Purpose

Preconditioned Conjugate Gradients method

Minimum number of arguments: eleven; minimum number: user-defined. Terminate the argument list with a NULL.

C Prototype

```
mxArray *mlfPcg(mxArray **flag,
                mxArray **relres,
                mxArray **iter,
                mxArray **resvec,
                mxArray *A,
                mxArray *b,
                mxArray *tol,
                mxArray *maxit,
                mxArray *M1,
                mxArray *M2,
                mxArray *x0,
                ...);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *b;                 /* Required input argument(s) */
mxArray *tol, *maxit;           /* Optional input argument(s) */
mxArray *M, *M1, *M2, *x0;      /* Optional input argument(s) */
mxArray *flag=NULL,*relres=NULL;/* Optional output argument(s) */
mxArray *iter=NULL,*resvec=NULL;/* Optional output argument(s) */
mxArray *x = NULL;              /* Return value */

mlfAssign(&x, mlfPcg(NULL,NULL,NULL,NULL,
                     A,b,NULL,NULL,NULL,NULL,NULL,NULL));
mlfAssign(&x, mlfPcg(NULL,NULL,NULL,NULL,
                     A,b,tol,NULL,NULL,NULL,NULL,NULL));
mlfAssign(&x, mlfPcg(NULL,NULL,NULL,NULL,
                     A,b,tol,maxit,NULL,NULL,NULL,NULL));
mlfAssign(&x, mlfPcg(NULL,NULL,NULL,NULL,
                     A,b,tol,maxit,M,NULL,NULL,NULL));
mlfAssign(&x, mlfPcg(NULL,NULL,NULL,NULL,
                     A,b,tol,maxit,M1,M2,NULL,NULL));
mlfAssign(&x, mlfPcg(NULL,NULL,NULL,NULL,
                     A,b,tol,maxit,M1,M2,x0,NULL);
mlfAssign(&x, mlfPcg(NULL,NULL,NULL,NULL,
                     A,b,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfPcg(&flag,NULL,NULL,NULL,
                     A,b,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfPcg(&flag,&relres,NULL,NULL,
                     A,b,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfPcg(&flag,&relres,&iter,NULL,
                     A,b,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfPcg(&flag,&relres,&iter,&resvec,
                     A,b,tol,maxit,M1,M2,x0,NULL));
```

MATLAB Syntax

```
x = pcg(A,b)
pcg(A,b,tol)
pcg(A,b,tol,maxit)
pcg(A,b,tol,maxit,M)
pcg(A,b,tol,maxit,M1,M2)
pcg(A,b,tol,maxit,M1,M2,x0)
x = pcg(A,b,tol,maxit,M1,M2,x0)
[x,flag] = pcg(A,b,tol,maxit,M1,M2,x0)
[x,flag,relres] = pcg(A,b,tol,maxit,M1,M2,x0)
[x,flag,relres,iter] = pcg(A,b,tol,maxit,M1,M2,x0)
[x,flag,relres,iter,resvec] = pcg(A,b,tol,maxit,M1,M2,x0)
```

See Also MATLAB pcg, Calling Conventions

mlfPchip

Purpose

Piecewise Cubic Hermite Interpolating Polynomial

C Prototype

```
mxArray *mlfPchip(mxArray *X, mxArray *Y, mxArray *XI );
```

C Syntax

```
#include "matlab.h"

mxArray *X, *Y, *XI;        /* Required input argument(s) */
mxArray *YI = NULL;         /* Return value */
mxArray *PP = NULL;         /* Return value */

mlfAssign(&YI, mlfPchip(X,Y,XI));
mlfAssign(&PP, mlfPchip(X,Y,NULL));
```

MATLAB Syntax

```
YI = pchip(X,Y,XI)
PP = pchip(X,Y)
```

See Also

MATLAB `pchip` Calling Conventions

Purpose

All possible permutations

C Prototype

```
mxArray *mlfPerms(mxArray * );
```

C Syntax

```
#include "matlab.h"

mxArray *v;                 /* Required input argument(s) */
mxArray *P = NULL;          /* Return value */

mlfAssign(&P, mlfPerms(v));
```

MATLAB Syntax

```
P = perms(v)
```

See Also

MATLAB `perms` Calling Conventions

mlfPermute

Purpose

Rearrange the dimensions of a multidimensional array

C Prototype

```
mxArray *mlfPermute(mxArray *A, mxArray *order);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *order;     /* Required input argument(s) */
mxArray *B = NULL;      /* Return value */

mlfAssign(&B, mlfPermute(A,order);
```

MATLAB Syntax

```
B = permute(A,order)
```

See Also

MATLAB `permute` Calling Conventions

Purpose

Ratio of a circle's circumference to its diameter π

C Prototype

```
mxArray *mlfPi();
```

C Syntax

```
#include "matlab.h"

mxArray *R = NULL;      /* Return value */

mlfAssign(&R, mlfPi());
```

MATLAB Syntax

```
pi
```

See Also

MATLAB `pi` Calling Conventions

mlfPinv

Purpose

Moore-Penrose pseudoinverse of a matrix

C Prototype

```
mxArray *mlfPinv(mxArray *A, mxArray *tol);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *tol;           /* Optional input argument(s) */
mxArray *B = NULL;      /* Return value */

mlfAssign(&B, mlfPinv(A,NULL));
mlfAssign(&B, mlfPinv(A,tol));
```

MATLAB Syntax

```
B = pinv(A)
B = pinv(A,tol)
```

See Also

MATLAB `pinv` Calling Conventions

Purpose

Given's plane rotation.

C Prototype

```
mxArray *mlfPlanerot(mxArray **y, mxArray *x);
```

C Syntax

```
#include "matlab.h"

mxArray *x;                 /* Required input argument(s) */
mxArray *y = NULL;          /* Required output argument(s) */
mxArray *g = NULL;          /* Return value */

mlfAssign(&g, mlfPlanerot(&y,x));
```

MATLAB Syntax

```
[g,y] = planerot(x)
```

See Also

Calling Conventions

mlfPol2cart

Purpose

Transform polar or cylindrical coordinates to Cartesian

C Prototype

```
mxArray *mlfPol2cart(mxArray **Y, mxArray **Z_out, mxArray *THETA,
                     mxArray *RHO, mxArray *Z_in);
```

C Syntax

```
#include "matlab.h"

mxArray *THETA, *RHO;   /* Required input argument(s) */
mxArray *Z_in;          /* Optional input argument(s) */
mxArray *Y = NULL;      /* Required output argument(s) */
mxArray *Z_out = NULl;  /* Optional output argument(s) */
mxArray *X = NULL;      /* Return value */

mlfAssign(&X, mlfPol2cart(&Y,NULL,THETA,RHO,NULL));
mlfAssign(&X, mlfPol2cart(&Y,&Z_out,THETA,RHO,Z_in));
```

MATLAB Syntax

```
[X,Y] = pol2cart(THETA,RHO)
[X,Y,Z] = pol2cart(THETA,RHO,Z)
```

See Also

MATLAB `pol2cart` Calling Conventions

Purpose

Polynomial with specified roots

C Prototype

```
mxArray *mlfPoly(mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *r;         /* Input argument(s) */
mxArray *p = NULL;      /* Return value */

mlfAssign(&p, mlfPoly(A));
mlfAssign(&p, mlfPoly(r));
```

MATLAB Syntax

```
p = poly(A)
p = poly(r)
```

See Also

MATLAB `poly` Calling Conventions

mlfPolyarea

Purpose

Area of polygon

C Prototype

```
mxArray *mlfPolyarea(mxArray *X, mxArray *Y, mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *X, *Y;             /* Required input argument(s) */
mxArray *dim;               /* Optional input argument(s) */
mxArray *A = NULL;          /* Return value */

mlfAssign(&A, mlfPolyarea(X,Y,NULL));
mlfAssign(&A, mlfPolyarea(X,Y,dim));
```

MATLAB Syntax

```
A = polyarea(X,Y)
A = polyarea(X,Y,dim)
```

See Also

MATLAB `polyarea` Calling Conventions

Purpose

Polynomial derivative

C Prototype

```
mxArray *mlfPolyder(mxArray **d, mxArray *a, mxArray *b);
```

C Syntax

```
#include "matlab.h"

mxArray *a, *b, *p;             /* Input argument(s) */
mxArray *d = NuLL;              /* Optional output argument(s) */
mxArray *k = NULL, *q = NULL;   /* Return value */

mlfAssign(&k, mlfPolyder(NULL,p,NULL));
mlfAssign(&k, mlfPolyder(NULL,a,b));
mlfAssign(&q, mlfPolyder(&d,b,a));
```

MATLAB Syntax

```
k = polyder(p)
k = polyder(a,b)
[q,d] = polyder(b,a)
```

See Also

MATLAB `polyder` Calling Conventions

mlfPolyeig

Purpose

Polynomial eigenvalue problem

Minimum number of arguments: one; maximum number: user-defined. Terminate the argument list with a `NULL`.

C Prototype

```
mxArray *mlfPolyeig(mxArray **E, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *A0, *A1;       /* Required input argument(s) */
mxArray *e;             /* Required output argument(s) */
mxArray *X = NULL;      /* Return value */

mlfAssign(&X, mlfPolyeig(&e,A0,A1,...,NULL));
```

MATLAB Syntax

```
[X,e] = polyeig(A0,A1,...Ap)
```

See Also

MATLAB `polyeig` Calling Conventions

Purpose

Polynomial curve fitting

C Prototype

```
mxArray *mlfPolyfit(mxArray **s, mxArray *x, mxArray *y,
                    mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *x, *y, *n;         /* Required input argument(s) */
mxArray *s = NULL;          /* Optional output argument(s) */
mxArray *p = NULL;          /* Return value */

mlfAssign(&p, mlfPolyfit(NULL,x,y,n));
mlfAssign(&p, mlfPolyfit(&s,x,y,n));
```

MATLAB Syntax

```
p = polyfit(x,y,n)
[p,s] = polyfit(x,y,n)
```

See Also

MATLAB `polyfit` Calling Conventions

mlfPolyval

Purpose

Polynomial evaluation

C Prototype

```
mxArray *mlfPolyval(mxArray **delta, mxArray *p, mxArray *x,
                    mxArray *S);
```

C Syntax

```
#include "matlab.h"

mxArray *p, *x;         /* Required input argument(s) */
mxArray *S;             /* Optional input argument(s) */
mxArray *delta = NULL;  /* Optional output argument(s) */
mxArray *y = NULL;      /* Return value */

mlfAssign(&y, mlfPolyval(NULL,p,x,NULL));
mlfAssign(&y, mlfPolyval(&delta,p,x,S));
```

MATLAB Syntax

```
y = polyval(p,x)
[y,delta] = polyval(p,x,S)
```

See Also

MATLAB `polyval` Calling Conventions

Purpose

Matrix polynomial evaluation

C Prototype

```
mxArray *mlfPolyvalm(mxArray *p, mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *p, *X;         /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfPolyvalm(p,X));
```

MATLAB Syntax

```
Y = polyvalm(p,X)
```

See Also

MATLAB `polyvalm` Calling Conventions

mlfPow2

Purpose

Base 2 power and scale floating-point numbers

C Prototype

```
mxArray *mlfPow2(mxArray *F, mxArray *E);
```

C Syntax

```
#include "matlab.h"

mxArray *Y, *F, *E;         /* Input argument(s) */
mxArray *X = NULL;          /* Return value */

mlfAssign(&X, mlfPow2(Y,NULL));
mlfAssign(&X, mlfPow2(F,E));
```

MATLAB Syntax

```
X = pow2(Y)
X = pow2(F,E)
```

See Also

MATLAB `pow2` Calling Conventions

Purpose

Generate list of prime numbers

C Prototype

```
mxArray *mlfPrimes(mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *n;                 /* Required input argument(s) */
mxArray *p = NULL;          /* Return value */

mlfAssign(&p, mlfPrimes(n));
```

MATLAB Syntax

```
p = primes(n)
```

See Also

MATLAB `primes` Calling Conventions

mlfProd

Purpose

Product of array elements

C Prototype

```
mxArray *mlfProd(mxArray *A, *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *A;                 /* Required input argument(s) */
mxArray *dim;               /* Optional input argument(s) */
mxArray *B = NULL;          /* Return value */

mlfAssign(&B, mlfProd(A,NULL));
mlfAssign(&B, mlfProd(A,dim));
```

MATLAB Syntax

```
B = prod(A)
B = prod(A,dim)
```

See Also

MATLAB `prod` Calling Conventions

Purpose

Quasi-Minimal Residual method

Minimum number of arguments: eleven, maximum: user-defined. Terminate the argument list with a NULL.

C Prototype

```
mxArray *mlfQmr(mxArray **flag,
                mxArray **relres,
                mxArray **iter,
                mxArray **resvec,
                mxArray *A,
                mxArray *b,
                mxArray *tol,
                mxArray *maxit,
                mxArray *M1,
                mxArray *M2,
                mxArray *x0,
                ...);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *b;                 /* Required input argument(s) */
mxArray *tol, *maxit;           /* Optional input argument(s) */
mxArray *M, *M1, *M2, *x0;      /* Optional input argument(s) */
mxArray *flag=NULL,*relres=NULL;/* Optional output argument(s) */
mxArray *iter=NULL,*resvec=NULL;/* Optional output argument(s) */
mxArray *x = NULL;              /* Return value */

mlfAssign(&x, mlfQmr(NULL,NULL,NULL,NULL,
                     A,b,NULL,NULL,NULL,NULL,NULL,NULL));
mlfAssign(&x, mlfQmr(NULL,NULL,NULL,NULL,
                     A,b,tol,NULL,NULL,NULL,NULL,NULL);
mlfAssign(&x, mlfQmr(NULL,NULL,NULL,NULL,
                     A,b,tol,maxit,NULL,NULL,NULL,NULL);
mlfAssign(&x, mlfQmr(NULL,NULL,NULL,NULL,
                     A,b,tol,maxit,M,NULL,NULL,NULL);
mlfAssign(&x, mlfQmr(NULL,NULL,NULL,NULL,
                     A,b,tol,maxit,M1,M2,NULL,NULL);
mlfAssign(&x, mlfQmr(NULL,NULL,NULL,NULL,
                     A,b,tol,maxit,M1,M2,x0,NULL);
mlfAssign(&x, mlfQmr(NULL,NULL,NULL,NULL,
                     A,b,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfQmr(&flag,NULL,NULL,NULL,
                     A,b,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfQmr(&flag,&relres,NULL,NULL,
                     A,b,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfQmr(&flag,&relres,&iter,NULL,
                      A,b,tol,maxit,M1,M2,x0,NULL));
mlfAssign(&x, mlfQmr(&flag,&relres,&iter,&resvec,
                     A,b,tol,maxit,M1,M2,x0,NULL));
```

MATLAB Syntax

```
x = qmr(A,b)
qmr(A,b,tol)
qmr(A,b,tol,maxit)
qmr(A,b,tol,maxit,M1)
qmr(A,b,tol,maxit,M1,M2)
qmr(A,b,tol,maxit,M1,M2,x0)
x = qmr(A,b,tol,maxit,M1,M2,x0)
[x,flag] = qmr(A,b,tol,maxit,M1,M2,x0)
[x,flag,relres] = qmr(A,b,tol,maxit,M1,M2,x0)
[x,flag,relres,iter] = qmr(A,b,tol,maxit,M1,M2,x0)
[x,flag,relres,iter,resvec] = qmr(A,b,tol,maxit,M1,M2,x0)
```

See Also MATLAB `qmr` Calling Conventions

mlfQr

Purpose

Orthogonal-triangular decomposition

C Prototype

```
mxArray *mlfQr(mxArray **R, mxArray **E,
               mxArray *in1, mxArray *in2, mxArray *in3);
```

C Syntax

```
#include "matlab.h"

mxArray *X;                     /* Required input argument(s) */
mxArray *R = NULL, *E = NULL;   /* Optional output argument(s) */
mxArray *Q = NULL, *A = NULL;   /* Return value */

mlfAssign(&Q, mlfQr(&R,NULL,X,NULL,NULL));
mlfAssign(&Q, mlfQr(&R,&E,X,NULL,NULL));
mlfAssign(&Q, mlfQr(&R,NULL,X,mlfScalar(0),NULL));
mlfAssign(&Q, mlfQr(&R,&E,X,mlfScalar(0),NULL));
mlfAssign(&A, mlfQr(NULL,NULL,X,NULL,NULL));
```

MATLAB Syntax

```
[Q,R] = qr(X)
[Q,R,E] = qr(X)
[Q,R] = qr(X,0)
[Q,R,E] = qr(X,0)
A = qr(X)
```

See Also

MATLAB `qr` Calling Conventions

Purpose

Delete column from QR factorization

C Prototype

```
mxArray *mlfQrdelete(mxArray **R_out, mxArray *Q_in, mxArray *R_in,
                     mxArray *j);
```

C Syntax

```
#include "matlab.h"

mxArray *Q_in, *R_in, *j;   /* Required input argument(s) */
mxArray *R_out = NULL;      /* Required output argument(s) */
mxArray *Q = NULL;          /* Return value */

mlfAssign(&Q, mlfQrdelete(&R_out,Q_in,R_in,j));
```

MATLAB Syntax

```
[Q,R] = qrdelete(Q,R,j);
```

See Also

MATLAB `qrdelete` Calling Conventions

mlfQrinsert

Purpose

Insert column in QR factorization

C Prototype

```
mxArray *mlfQrinsert(mxArray **R_out, mxArray *Q_in, mxArray *R_in,
                     mxArray *j, mxArray *x);
```

C Syntax

```
#include "matlab.h"

mxArray *Q_in, *R_in, *j, *x;   /* Required input argument(s) */
mxArray *R_out = NULL;          /* Required output argument(s) */
mxArray *Q = NULL;              /* Return value */

mlfAssign(&Q, mlfQrinsert(&R_out,Q_in,R_in,j,x));
```

MATLAB Syntax

```
[Q,R] = qrinsert(Q,R,j,x)
```

See Also

MATLAB `qrinsert` Calling Conventions

Purpose

Numerical evaluation of integrals

Minimum number of arguments: six, maximum: user-defined. Terminate the argument list with a NULL.

C Prototype

```
mxArray *mlfQuad(mxArray **cnt, mxArray *funfcn, mxArray *a,
                 mxArray *b, mxArray *tol, mxArray *trace, ...);
mxArray *mlfQuad8(mxArray **cnt, mxArray *funfcn, mxArray *a,
                 mxArray *b, mxArray *tol, mxArray *trace, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *func;              /* String array(s) */
mxArray *a, *b, *tol;       /* Required input argument(s) */
mxArray *trace, *P1, *P2;   /* Optional input argument(s) */
mxArray *q = NULL;          /* Return value */

mlfAssign(&q, mlfQuad(NULL,func,a,b,NULL,NULL,NULL));
mlfAssign(&q, mlfQuad(NULL,func,a,b,tol,NULL,NULL));
mlfAssign(&q, mlfQuad(NULL,func,a,b,tol,trace,NULL));
mlfAssign(&q, mlfQuad(NULL,func,a,b,tol,trace,P1,P2,...,NULL));

mlfAssign(&q, mlfQuad8(NULL,func,a,b,NULL,NULL,NULL));
mlfAssign(&q, mlfQuad8(NULL,func,a,b,tol,NULL,NULL));
mlfAssign(&q, mlfQuad8(NULL,func,a,b,tol,trace,NULL));
mlfAssign(&q, mlfQuad8(NULL,func,a,b,tol,trace,P1,P2,...,NULL));
```

MATLAB Syntax

```
q = quad('fun',a,b)
q = quad('fun',a,b,tol)
q = quad('fun',a,b,tol,trace)
q = quad('fun',a,b,tol,trace,P1,P2,...)
q = quad8(...)
```

See Also

MATLAB quad, quad8 Calling Conventions

mlfQz

Purpose

QZ factorization for generalized eigenvalues

C Prototype

```
mxArray *mlfQz(mxArray **BB, mxArray **Q, mxArray **Z, mxArray **V,
               mxArray *A, mxArray *B);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *B;                 /* Required input argument(s) */
mxArray *BB = NULL, *Q = NULL;  /* Required output argument(s) */
mxArray *Z = NULL, *V=NULL;     /* Required output argument(s) */
mxArray *AA = NULL;             /* Return value */

mlfAssign(&AA, mlfQz(&BB,&Q,&Z,&V,A,B));
```

MATLAB Syntax

```
[AA,BB,Q,Z,V] = qz(A,B)
```

See Also

MATLAB qz Calling Conventions

mlfRand

Purpose

Uniformly distributed random numbers and arrays

C Prototype

```
mxArray *mlfRand(mxArray *n, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *m, *n, *p, *A;         /* Input argument(s) */
mxArray *Y = NULL, *s = NULL;   /* Return value */

mlfAssign(&Y, mlfRand(n,NULL));
mlfAssign(&Y, mlfRand(m,n,NULL));
mlfAssign(&Y, mlfRand(mlfHorzcat(m,n,NULL),NULL));
mlfAssign(&Y, mlfRand(m,n,p,...,NULL));
mlfAssign(&Y, mlfRand(mlfHorzcat(m,n,p,...,NULL),NULL));
mlfAssign(&Y, mlfRand(mlfSize(NULL,A,NULL),NULL));
mlfAssign(&Y, mlfRand(NULL));
mlfAssign(&s, mlfRand(mxCreateString("state"),NULL));
```

MATLAB Syntax

```
Y = rand(n)
Y = rand(m,n)
Y = rand([m n])
Y = rand(m,n,p,...)
Y = rand([m n p...])
Y = rand(size(A))
rand
s = rand('state')
```

See Also

MATLAB `rand` Calling Conventions

Purpose

Normally distributed random numbers and arrays

C Prototype

```
mxArray *mlfRandn(mxArray *n, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *m, *n, *p, *A;         /* Input argument(s) */
mxArray *Y = NULL, *s = NULL;   /* Return value */

mlfAssign(&Y, mlfRandn(n,NULL));
mlfAssign(&Y, mlfRandn(m,n,NULL));
mlfAssign(&Y, mlfRandn(mlfHorzcat(m,n,NULL),NULL));
mlfAssign(&Y, mlfRandn(m,n,p,...,NULL));
mlfAssign(&Y, mlfRandn(mlfHorzcat(m,n,p,...,NULL),NULL));
mlfAssign(&Y, mlfRandn(mlfSize(NULL,A,NULL),NULL));
mlfAssign(&Y, mlfRandn(NULL));
mlfAssign(&s, mlfRandn(mxCreateString("state"),NULL));
```

MATLAB Syntax

```
Y = randn(n)
Y = randn(m,n)
Y = randn([m n])
Y = randn(m,n,p,...)
Y = randn([m n p...])
Y = randn(size(A))
randn
s = randn('state')
```

See Also

MATLAB `randn` Calling Conventions

mlfRandperm

Purpose

Random permutation

C Prototype

```
mxArray *mlfRandperm(mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *n;                 /* Required input argument(s) */
mxArray *p = NULL;          /* Return value */

mlfAssign(&p, mlfRandperm(n));
```

MATLAB Syntax

```
p = randperm(n)
```

See Also

MATLAB `randperm` Calling Conventions

Purpose

Rank of a matrix

C Prototype

```
mxArray *mlfRank(mxArray *A, mxArray *tol);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *tol;           /* Optional input argument(s) */
mxArray *k = NULL;      /* Return value */

mlfAssign(&k, mlfRank(A,NULL));
mlfAssign(&k, mlfRank(A,tol));
```

MATLAB Syntax

```
k = rank(A)
k = rank(A,tol)
```

See Also

MATLAB `rank` Calling Conventions

mlfRat, mlfRats

Purpose

Rational fraction approximation

C Prototype

```
mxArray *mlfRat(mxArray **D, mxArray *X, mxArray *tol);
mxArray *mlfRats(mxArray *X, mxArray *strlength);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *tol;           /* Optional input for mlfRat */
mxArray *strlength;     /* Optional input for mlfRats */
mxArray *D =NULL;       /* Required output argument for mlfRat */
mxArray *N = NULL, *str = NULL;/* Return values for mlfRat */
mxArray *S = NULL;      /* Return value for mlfRats */

mlfAssign(&N, mlfRat(&D,X,NULL));
mlfAssign(&N, mlfRat(&D,X,tol));
mlfAssign(&str, mlfRat(NULL,X,NULL));
mlfAssign(&str, mlfRat(NULL,X,tol));

mlfAssign(&S, mlfRats(X,strlength));
mlfAssign(&S, mlRats(X,NULL));
```

MATLAB Syntax

```
[N,D] = rat(X)
[N,D] = rat(X,tol)
rat(...)
S = rats(X,strlength)
S = rats(X)
```

See Also

MATLAB `rat`, `rats` Calling Conventions

Purpose

Matrix reciprocal condition number estimate

C Prototype

```
mxArray *mlfRcond(mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *c = NULL;      /* Return value */

mlfAssign(&c, mlfRcond(A));
```

MATLAB Syntax

```
c = rcond(A)
```

See Also

MATLAB `rcond` Calling Conventions

mlfReal

Purpose

Real part of complex number

C Prototype

```
mxArray *mlfReal(mxArray *Z);
```

C Syntax

```
#include "matlab.h"

mxArray *Z;             /* Required input argument(s) */
mxArray *X = NULL;      /* Return value */

mlfAssign(&X, mlfReal(Z));
```

MATLAB Syntax

```
X = real(Z)
```

See Also

MATLAB `real` Calling Conventions

Purpose Largest positive floating-point number

C Prototype

```
mxArray *mlfRealmax();
```

C Syntax

```
#include "matlab.h"

mxArray *n = NULL;      /* Return value */

mlfAssign(&n, mlfRealmax());
```

MATLAB Syntax

```
n = realmax
```

See Also MATLAB `realmax` Calling Conventions

mlfRealmin

Purpose

Smallest positive floating-point number

C Prototype

```
mxArray *mlfRealmin();
```

C Syntax

```
#include "matlab.h"

mxArray *n = NULL;      /* Return value */

mlfAssign(&n, mlfRealmin());
```

MATLAB Syntax

```
n = realmin
```

See Also

MATLAB `realmin` Calling Conventions

Purpose

Rectangle intersection area

C Prototype

```
mxArray *mlfRectint(mxArray *a, mxArray *b);
```

C Syntax

```
#include "matlab.h"

mxArray *a, *b;         /* Required input argument(s) */
mxArray *R = NULL;      /* Return value */

mlfAssign(&R, mlfRectint(a,b));
```

MATLAB Syntax

```
rectint(a,b)
```

See Also

MATLAB `rectint` Calling Conventions

mlfRem

Purpose

Remainder after division

C Prototype

```
mxArray *mlfRem(mxArray *X, mxArray *Y);
```

C Syntax

```
#include "matlab.h"

mxArray *X, *Y;         /* Required input argument(s) */
mxArray *R = NULL;      /* Return value */

mlfAssign(&R, mlfRem(X,Y));
```

MATLAB Syntax

```
R = rem(X,Y)
```

See Also

MATLAB `rem` Calling Conventions

Purpose

Replicate and tile an array

C Prototype

```
mxArray *mlfRepmat(mxArray *A, mxArray *m, mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *x;                 /* Dimension vector */
mxArray *A, *m, *n, *p;     /* Input argument(s) */
mxArray *B = NULL;          /* Return value */

mlfAssign(&B, mlfRepmat(A,m,n));
mlfAssign(&B, mlfRepmat(A,mlfHorzcat(m,n,NULL),NULL));
mlfAssign(&B, mlfRepmat(A,mlfHorzcat(m,n,p,...,NULL),NULL));
```

MATLAB Syntax

```
B = repmat(A,m,n)
B = repmat(A,[m n])
B = repmat(A,[m n p...])
```

See Also

MATLAB `repmat` Calling Conventions

mlfReshape

Purpose

Reshape array.

Minimum number of arguments: two; maximum number of arguments: user-defined. Terminate the list of arguments with a `NULL`.

C Prototype

```
mxArray *mlfReshape(mxArray *A, mxArray *m, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *A;                 /* Required input argument(s) */
mxArray *m, *n, *p, *siz;   /* Optional input argument(s) */
mxArray *B = NULL;          /* Return value */

mlfAssign(&B, mlfReshape(A,m,n,NULL));
mlfAssign(&B, mlfReshape(A,m,n,p,...,NULL));
mlfAssign(&B, mlfReshape(A,mlfHorzcat(m,n,p,...,NULL),NULL));
mlfAssign(&B, mlfReshape(A,siz,NULL));
```

MATLAB Syntax

```
B = reshape(A,m,n)
B = reshape(A,m,n,p,...)
B = reshape(A,[m n p...])
B = reshape(A,siz)
```

See Also

MATLAB `reshape` Calling Conventions

Purpose

Residue of a repeated pole

C Prototype

```
mxArray *mlfResi2(mxArray *u, mxArray *v, mxArray *pole, mxArray *n,
                  mxArray *k);
```

C Syntax

```
#include "matlab.h"

mxArray *u, *v, *pole;      /* Required input argument(s) */
mxArray *n, *k;             /* Required input argument(s) */
mxArray *R = NULL;          /* Return value */

mlfAssign(&R, mlfResi2(u,v,pole,n,k));
```

MATLAB Syntax

```
resi2(u,v,pole,n,k)
```

See Also

Calling Conventions

mlfResidue

Purpose

Convert between partial fraction expansion and polynomial coefficients

C Prototype

```
mxArray *mlfResidue(mxArray **O1, mxArray **O2, mxArray *I1,
                    mxArray *I2, mxArray *I3);
```

C Syntax

```
#include "matlab.h"

mxArray *r = NULL, *p, *k;
mxArray *b = NULL, *a;

mlfAssign(&r, mlfResidue(&p,&k,b,a,NULL));
mlfAssign(&b, mlfResidue(&a,NULL,r,p,k));
```

MATLAB Syntax

```
[r,p,k] = residue(b,a)
[b,a] = residue(r,p,k)
```

See Also

MATLAB `residue` Calling Conventions

Purpose

Remove structure fields

C Prototype

```
mxArray *mlfRmfield(mxArray *s, mxArray *field);
```

C Syntax

```
#include "matlab.h"

mxArray *s;             /* Required input argument and
                           return value */
mxArray *FIELDS;        /* Optional input argument(s) */

mlfAssign(&s, mlfRmfield(s,mxCreateString("field")));
mlfAssign(&s, mlfRmfield(s,FIELDS));
```

MATLAB Syntax

```
s = rmfield(s,'field')
s = rmfield(s,FIELDS)
```

See Also

MATLAB `rmfield` Calling Conventions

mlfRoots

Purpose

Polynomial roots

C Prototype

```
mxArray *mlfRoots(mxArray *c);
```

C Syntax

```
#include "matlab.h"

mxArray *c;             /* Required input argument(s) */
mxArray *r = NULL;      /* Return value */

mlfAssign(&r, mlfRoots(c));
```

MATLAB Syntax

```
r = roots(c)
```

See Also

MATLAB `roots` Calling Conventions

Purpose

Classic symmetric eigenvalue test matrix (Rosser matrix)

C Prototype

```
mxArray *mlfRosser();
```

C Syntax

```
#include "matlab.h"

mxArray A = NULL;           /* Return value */

mlfAssign(&A, mlfRosser());
```

MATLAB Syntax

```
A = rosser
```

See Also

MATLAB `gallery` Calling Conventions

mlfRot90

Purpose

Rotate matrix 90 degrees

C Prototype

```
mxArray *mlfRot90(mxArray *A, mxArray *k);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *k;             /* Optional input argument(s) */
mxArray *B = NULL;      /* Return value */

mlfAssign(&B, mlfRot90(A,NULL));
mlfAssign(&B, mlfRot90(A,k));
```

MATLAB Syntax

```
B = rot90(A)
B = rot90(A,k)
```

See Also

MATLAB `rot90` Calling Conventions

Purpose

Round to nearest integer

C Prototype

```
mxArray *mlfRound(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfRound(X));
```

MATLAB Syntax

```
Y = round(X)
```

See Also

MATLAB `round` Calling Conventions

mlfRref

Purpose

Reduced row echelon form

C Prototype

```
mxArray *mlfRref(mxArray **jb, mxArray *A, mxArray *tol);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *tol;           /* Optional input argument(s) */
mxArray *jb = NULL;     /* Optional output argument(s) */
mxArray *R = NULL;      /* Return value */

mlfAssign(&R, mlfRref(NULL,A,NULL));
mlfAssign(&R, mlfRref(&jb,A,NULL));
mlfAssign(&R, mlfRref(&jb,A,tol));
```

MATLAB Syntax

```
R = rref(A)
[R,jb] = rref(A)
[R,jb] = rref(A,tol)
```

See Also

MATLAB `rref` Calling Conventions

Purpose

Convert real Schur form to complex Schur form

C Prototype

```
mxArray *mlfRsf2csf(mxArray **T_out, mxArray *U_in, mxArray *T_in);
```

C Syntax

```
#include "matlab.h"

mxArray *U_in, *T_in;       /* Required input argument(s) */
mxArray *T_out = NULL;      /* Required output argument(s) */
mxArray *U_out = NULL;      /* Return value */

mlfAssign(&U_out, mlfRsf2csf(&T_out,U_in,T_in));
```

MATLAB Syntax

```
[U,T] = rsf2csf(U,T)
```

See Also

MATLAB `rsf2csf` Calling Conventions

mlfSave

Purpose

Save variables to disk

Minimum number of arguments: four, maximum: user-defined. Terminate the argument list to `mlfSave()` with a `NULL`.

C Prototype

```
void mlfSave(mxArray *file, const char *mode, ... );
```

C Syntax

```
#include "matlab.h"

mxArray *file, *x, *y, *z;

mlfSave(mxCreateString("fname"),
        "w","X",x,NULL);               /* overwrite data */
mlfSave(mxCreateString("fname"),
        "u","X",x,"Y",y,"Z",z,NULL);   /* append to data */
```

MATLAB Syntax

```
save fname X
save fname X,Y,Z
```

See Also

MATLAB save Calling Conventions

Purpose

Schur decomposition

C Prototype

```
mxArray *mlfSchur(mxArray **T, mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A;                 /* Required input argument(s) */
mxArray *T = NULL, *U = NULL;/* Return value */

mlfAssign(&U, mlfSchur(&T,A));
mlfAssign(&T, mlfSchur(NULL,A));
```

MATLAB Syntax

```
[U,T] = schur(A)
T = schur(A)
```

See Also

MATLAB schur Calling Conventions

mlfSec, mlfSech

Purpose

Secant and hyperbolic secant

C Prototype

```
mxArray *mlfSec(mxArray *X);
mxArray *mlfSech(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfSec(X));
mlfAssign(&Y, mlfSech(X));
```

MATLAB Syntax

```
Y = sec(X)
Y = sech(X)
```

See Also

MATLAB sec, sech Calling Conventions

Purpose

Return the set difference of two vectors

C Prototype

```
mxArray *mlfSetdiff(mxArray **i, mxArray *A, mxArray *B,
                    mxArray *rows_str);
```

C Syntax

```
#include "matlab.h"

mxArray *a, *b, *A, *B;     /* Input argument(s) */
mxArray *i;                 /* Optional output argument(s) */
mxArray *c = NULL;          /* Return value */

mlfAssign(&c, mlfSetdiff(NULL,a,b,NULL));
mlfAssign(&c, mlfSetdiff(NULL,A,B,mxCreateString("rows")));
mlfAssign(&c, mlfSetdiff(&i,a,b,NULL));
mlfAssign(&c, mlfSetdiff(&i,A,B,rows_str));
```

MATLAB Syntax

```
c = setdiff(a,b)
c = setdiff(A,B,'rows')
[c,i] = setdiff(...)
```

See Also

MATLAB `setdiff` Calling Conventions

mlfSetfield

Purpose

Set field of structure array

C Prototype

```
mxArray *mlfSetfield(mxArray *in1, mxArray *in2, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *s, *v;             /* Required input argument(s) */
mxArray *i, *j, *k;         /* Optional input argument(s) */
mxArray *s;                 /* Return value */

mlfAssign (&s, mlfSetfield(s,mxCreateString("field"),v,NULL));
mlfAssign (&s,
   mlfSetfield(s,mlfCellhcat(i,j,NULL),mxCreateString("field"),
               mlfCellhcat(k,NULL),v,NULL));
```

MATLAB Syntax

```
s = setfield(s,'field',v)
s = setfield(s,{i,j},'field',{k},v)
```

See Also

MATLAB `setfield` Calling Conventions

Purpose

Set string flag

This function has been renamed to `mlfChar`.

mlfSetxor

Purpose

Set exclusive-or of two vectors

C Prototype

```
mxArray *mlfSetxor(mxArray **ia, mxArray **ib, mxArray *A,
                   mxArray *B, mxArray *rows_str);
```

C Syntax

```
#include "matlab.h"

mxArray *rows_str;          /* String array(s) */
mxArray *a, *b, *A, *B;     /* Input argument(s) */
mxArray *ia, *ib;           /* Optional output argument(s) */
mxArray *c = NULL;          /* Return value */

mlfAssign(&c, mlfSetxor(NULL,NULL,a,b,NULL));
mlfAssign(&c, mlfSetxor(NULL,NULL,A,B,mlfChar("rows")));
mlfAssign(&c, mlfSetxor(&ia,&ib,a,b,NULL));
mlfAssign(&c, mlfSetxor(&ia,&ib,A,B,rows_str));
```

MATLAB Syntax

```
c = setxor(a,b)
c = setxor(A,B,'rows')
[c,ia,ib] = setxor(...)
```

See Also

MATLAB `setxor` Calling Conventions

Purpose

Shift dimensions

C Prototype

```
mxArray *mlfShiftdim(mxArray **nshifts, mxArray *X, mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *X;                 /* Required input argument(s) */
mxArray *n;                 /* Optional input argument(s) */
mxArray *nshifts;           /* Optional output argument(s) */
mxArray *B = NULL;          /* Return value */

mlfAssign(&B, mlfShiftdim(NULL,X,n));
mlfAssign(&B, mlfShiftdim(&nshifts,X,NULL));
```

MATLAB Syntax

```
B = shiftdim(X,n)
[B,nshifts] = shiftdim(X)
```

See Also

MATLAB `shiftdim` Calling Conventions

mlfSign

Purpose

Signum function

C Prototype

```
mxArray *mlfSign(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfSign(X));
```

MATLAB Syntax

```
Y = sign(X)
```

See Also

MATLAB `sign` Calling Conventions

Purpose

Sine and hyperbolic sine

C Prototype

```
mxArray *mlfSin(mxArray *X);
mxArray *mlfSinh(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfSin(X));
mlfAssign(&Y, mlfSinh(X));
```

MATLAB Syntax

```
Y = sin(X)
Y = sinh(X)
```

See Also

MATLAB `sin`, `sinh` Calling Conventions

mlfSize

Purpose

Array dimensions

C Prototype

```
mxArray *mlfSize(mlfVarargoutList *varargout, mxArray *X,
                 mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *X;                     /* Required input argument(s) */
mxArray *dim;                   /* Optional input argument(s) */
mxArray *n;                     /* Optional output argument(s) */
mxArray *d2=NULL, *d3=NULL;     /* Optional output argument(s) */
mxArray *d = NULL, *m = NULL;   /* Return value */
mxArray *d1 = NULL;             /* Return value */

mlfAssign(&d, mlfSize(NULL,X,NULL));
mlfSize(mlfVarargout(&m,&n,NULL),X,NULL);
mlfAssign(&m, mlfSize(NULL,X,dim));

/* If X is a four-dimensional array */
mlfSize(mlfVarargout(&d1,&d2,&d3,...,NULL),X,NULL);
```

MATLAB Syntax

```
d = size(X)
[m,n] = size(X)
m = size(X,dim)
[d1,d2,d3,...,dn] = size(X)
```

See Also

MATLAB `size` Calling Conventions

Purpose

Sort elements in ascending order

C Prototype

```
mxArray *mlfSort(mxArray **INDEX, mxArray *A, mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *dim;           /* Optional input argument(s) */
mxArray *INDEX;         /* Optional output argument(s) */
mxArray *B = NULL;      /* Return value */

mlfAssign(&B, mlfSort(NULL,A,NULL));
mlfAssign(&B, mlfSort(&INDEX,A,NULL));
mlfAssign(&B, mlfSort(NULL,A,dim));
```

MATLAB Syntax

```
B = sort(A)
[B,INDEX] = sort(A)
B = sort(A,dim)
```

See Also

MATLAB `sort` Calling Conventions

mlfSortrows

Purpose

Sort rows in ascending order

C Prototype

```
mxArray *mlfSortrows(mxArray **index, mxArray *A, mxArray *column);
```

C Syntax

```
#include "matlab.h"

mxArray *A;                 /* Required input argument(s) */
mxArray *column;            /* Optional input argument(s) */
mxArray *index;             /* Optional output argument(s) */
mxArray *B = NULL;          /* Return value */

mlfAssign(&B, mlfSortrows(NULL,A,NULL));
mlfAssign(&B, mlfSortrows(NULL,A,column));
mlfAssign(&B, mlfSortrows(&index,A,column));
```

MATLAB Syntax

```
B = sortrows(A)
B = sortrows(A,column)
[B,index] = sortrows(A)
```

See Also

MATLAB `sortrows` Calling Conventions

Purpose

Allocate space for sparse matrix

C Prototype

```
mxArray *mlfSpalloc(mxArray *m, mxArray *n, mxArray *nzmax);
```

C Syntax

```
#include "matlab.h"

mxArray *m, *n, *nzmax;     /* Required input argument(s) */
mxArray *S = NULL;          /* Return value */

mlfAssign(&S, mlfSpalloc(m,n,nzmax));
```

MATLAB Syntax

```
S = spalloc(m,n,nzmax)
```

See Also

MATLAB `spalloc` Calling Conventions

mlfSparse

Purpose

Create sparse matrix

C Prototype

```
mxArray *mlfSparse(mxArray *i, mxArray *j, mxArray *s,
                   mxArray *m, mxArray *n, mxArray *nzmax);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *i;                 /* Required input argument(s) */
mxArray *j, *s, *m, *n, *nzmax; /* Optional input argument(s) */
mxArray *S = NULL;              /* Return value */

mlfAssign(&S, mlfSparse(A,NULL,NULL,NULL,NULL,NULL));
mlfAssign(&S, mlfSparse(i,j,s,m,n,nzmax));
mlfAssign(&S, mlfSparse(i,j,s,m,n,NULL));
mlfAssign(&S, mlfSparse(i,j,s,NULL,NULL,NULL));
mlfAssign(&S, mlfSparse(m,n,NULL,NULL,NULL,NULL));
```

MATLAB Syntax

```
S = sparse(A)
S = sparse(i,j,s,m,n,nzmax)
S = sparse(i,j,s,m,n)
S = sparse(i,j,s)
S = sparse(m,n)
```

See Also

MATLAB sparse, Calling Conventions

Purpose

Create a MATLAB sparse matrix from an external sparse matrix format

C Prototype

```
mxArray *mlfSpconvert(mxArray *D);
```

C Syntax

```
#include "matlab.h"

mxArray *D;                 /* Required input argument(s) */
mxArray *S = NULL;          /* Return value */

mlfAssign(&S, mlfSpconvert(D));
```

MATLAB Syntax

```
S = spconvert(D)
```

See Also

MATLAB `spconvert` Calling Conventions

mlfSpdiags

Purpose

Extract and create sparse band and diagonal matrices

C Prototype

```
mxArray *mlfSpdiags(mxArray **res2,
                    mxArray *arg1,
                    mxArray *arg2,
                    mxArray *arg3,
                    mxArray *arg4);
```

C Syntax

```
#include "matlab.h"

mxArray *A;
mxArray *d, *m, *n;
mxArray *B = NULL, *A = NULL;

mlfAssign(&B, mlfSpdiags(&d,A,NULL,NULL,NULL));
mlfAssign(&B, mlfSpdiags(NULL,A,d,NULL,NULL));
mlfAssign(&A, mlfSpdiags(NULL,B,d,A,NULL));
mlfAssign(&A, mlfSpdiags(NULL,B,d,m,n));
```

MATLAB Syntax

```
[B,d] = spdiags(A)
B = spdiags(A,d)
A = spdiags(B,d,A)
A = spdiags(B,d,m,n)
```

See Also

MATLAB `spdiags` Calling Conventions

Purpose

Sparse identity matrix

C Prototype

```
mxArray *mlfSpeye(mxArray *m, mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *m;                 /* Required input argument(s) */
mxArray *n;                 /* Optional input argument(s) */
mxArray *S = NULL;          /* Return value */

mlfAssign(&S, mlfSpeye(m,n));
mlfAssign(&S, mlfSpeye(n,NULL));
```

MATLAB Syntax

```
S = speye(m,n)
S = speye(n)
```

See Also

MATLAB speye, Calling Conventions

mlfSpfun

Purpose

Apply function to nonzero sparse matrix elements

C Prototype

```
mxArray *mlfSpfun(mxArray *function, mxArray *s);
```

C Syntax

```
#include "matlab.h"

mxArray *S;                 /* Required input argument(s) */
mxArray *f = NULL;          /* Return value */

mlfAssign(&f, mlfSpfun(mxCreateString("function"),S));
```

MATLAB Syntax

```
f = spfun('function',S)
```

See Also

MATLAB `spfun` Calling Conventions

Purpose

Transform spherical coordinates to Cartesian

C Prototype

```
mxArray *mlfSph2cart(mxArray **y, mxArray **z, mxArray *THETA,
                     mxArray *PHI, mxArray *R);
```

C Syntax

```
#include "matlab.h"

mxArray *THETA, *PHI, *R;       /* Required input argument(s) */
mxArray *y = NULL, *z = NULL;   /* Required output argument(s) */
mxArray *x = NULL;              /* Return value */

mlfAssign(&x, mlfSph2cart(&y,&z,THETA,PHI,R));
```

MATLAB Syntax

```
[x,y,z] = sph2cart(THETA,PHI,R)
```

See Also

MATLAB `sph2cart` Calling Conventions

mlfSpline

Purpose

Cubic spline interpolation

C Prototype

```
mxArray *mlfSpline(mxArray *x, mxArray *y, mxArray *xi);
```

C Syntax

```
#include "matlab.h"

mxArray *x, *y;             /* Required input argument(s) */
mxArray *xi;                /* Optional input argument(s) */
mxArray *yi = NULL, *pp = NULL;/* Return value */

mlfAssign(&yi, mlfSpline(x,y,xi));
mlfAssign(&pp, mlfSpline(x,y,NULL));
```

MATLAB Syntax

```
yi = spline(x,y,xi)
pp = spline(x,y)
```

See Also

MATLAB `spline` Calling Conventions

Purpose

Replace nonzero sparse matrix elements with ones

C Prototype

```
mxArray *mlfSpones(mxArray *S);
```

C Syntax

```
#include "matlab.h"

mxArray *S;             /* Required input argument(s) */
mxArray *R = NULL;      /* Return value */

mlfAssign(&R, mlfSpones(S));
```

MATLAB Syntax

```
R = spones(S)
```

See Also

MATLAB spones Calling Conventions

mlfSpparms, mlfVSpparms

Purpose

Set parameters for sparse matrix routines

C Prototype

```
mxArray *mlfSpparms(mxArray **values,
                    mxArray *key,
                    mxArray *value);
void mlfVSpparms(mxArray *key, mxArray *value);
```

C Syntax

```
#include "matlab.h"

mxArray *value, *values_in;     /* Optional input argument(s) */
mxArray *values_out=NULL;       /* Return value */
mxArray *keys=NULL;             /* Return value */

mlfVSpparms(mxCreateString("key"),value));
mlfVSpparms(NULL,NULL);
mlfAssign(&values_out, mlfSpparms(NULL,NULL,NULL));
mlfAssign(&keys, mlfSpparms(&values_out,NULL,NULL));
mlfVSpparms(values_in,NULL);
mlfAssign(&value, mlfSpparms(NULL,mxCreateString("key"),NULL));
mlfVSpparms(mxCreateString("default"),NULL);
mlfVSpparms(mxCreateString("tight"),NULL);
```

MATLAB Syntax

```
spparms('key',value)
spparms
values = spparms
[keys,values] = spparms
spparms(values)
value = spparms('key')
spparms('default')
spparms('tight')
```

See Also

MATLAB spparms Calling Conventions

Purpose

Sparse uniformly distributed random matrix

C Prototype

```
mxArray *mlfSprand(mxArray *m,
                   mxArray *n,
                   mxArray *density,
                   mxArray *rc);
```

C Syntax

```
#include "matlab.h"

mxArray *S, *m;             /* Required input argument(s) */
mxArray *n, *density, *rc;  /* Optional input argument(s) */
mxArray *R = NULL;          /* Return value */

mlfAssign(&R, mlfSprand(S,NULL,NULL,NULL));
mlfAssign(&R, mlfSprand(m,n,density,NULL));
mlfAssign(&R, mlfSprand(m,n,density,rc));
```

MATLAB Syntax

```
R = sprand(S)
R = sprand(m,n,density)
R = sprand(m,n,density,rc)
```

See Also

MATLAB sprand Calling Conventions

mlfSprandn

Purpose

Sparse normally distributed random matrix

C Prototype

```
mxArray *mlfSprandn(mxArray *arg1,
                    mxArray *n,
                    mxArray *density,
                    mxArray *rc);
```

C Syntax

```
#include "matlab.h"

mxArray *S, *m;             /* Required input argument(s) */
mxArray *n, *density, *rc;  /* Optional input argument(s) */
mxArray *R = NULL;          /* Return value */

mlfAssign(&R, mlfSprandn(S,NULL,NULL,NULL));
mlfAssign(&R, mlfSprandn(m,n,density,NULL));
mlfAssign(&R, mlfSprandn(m,n,density,rc));
```

MATLAB Syntax

```
R = sprandn(S)
R = sprandn(m,n,density)
R = sprandn(m,n,density,rc)
```

See Also

MATLAB `sprandn` Calling Conventions

Purpose

Sparse symmetric random matrix

C Prototype

```
mxArray *mlfSprandsym(mxArray *arg1,
                      mxArray *density,
                      mxArray *rc,
                      mxArray *kind);
```

C Syntax

```
#include "matlab.h"

mxArray *S, *n;             /* Required input argument(s) */
mxArray *density, *rc, kind;/* Optional input argument(s) */
mxArray *R = NULL;          /* Return value */

mlfAssign(&R, sprandsym(S,NULL,NULL,NULL));
mlfAssign(&R, sprandsym(n,density,NULL,NULL));
mlfAssign(&R, sprandsym(n,density,rc,NULL));
mlfAssign(&R, sprandsym(n,density,rc,kind));
```

MATLAB Syntax

```
R = sprandsym(S)
R = sprandsym(n,density)
R = sprandsym(n,density,rc)
R = sprandsym(n,density,rc,kind)
```

See Also

MATLAB `sprandsym` Calling Conventions

mlfSprintf

Purpose

Write formatted data to a string

Minimum number of arguments: two, maximum: user-defined. Terminate the list of arguments with a `NULL`.

C Prototype

```
mxArray *mlfSprintf(mxArray **errmsg,
                    mxArray *format,
                    mxArray *A, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *format;            /* String array(s) */
mxArray *A;                 /* Input argument(s) */
mxArray *errmsg;            /* Optional output argument(s) */
mxArray *s = NULL;          /* Return value */

mlfAssign(&s, mlfSprintf(NULL,format,A,...,NULL));
mlfAssign(&s, mlfSprintf(&errmsg,format,A,...,NULL));
```

MATLAB Syntax

```
s = sprintf(format,A,...)
[s,errmsg] = sprintf(format,A,...)
```

See Also

MATLAB `sprintf` Calling Conventions

Purpose

Square root

C Prototype

```
mxArray *mlfSqrt(mxArray *A);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *B = NULL;      /* Return value */

mlfAssign(&B, mlfSqrt(A));
```

MATLAB Syntax

```
B = sqrt(A)
```

See Also

MATLAB `sqrt` Calling Conventions

mlfSqrtm

Purpose

Matrix square root

C Prototype

```
mxArray *mlfSqrtm(mxArray **esterr, mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *esterr;        /* Optional output argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfSqrtm(NULL,X));
mlfAssign(&Y, mlfSqrtm(&esterr,X));
```

MATLAB Syntax

```
Y = sqrtm(X)
[Y,esterr] = sqrtm(X)
```

See Also

MATLAB `sqrtm` Calling Conventions

Purpose

Read string under format control

C Prototype

```
mxArray *mlfSscanf(mxArray **count, mxArray **errmsg,
                   mxArray **nextindex, mxArray *s,
                   mxArray *format, mxArray *size);
```

C Syntax

```
#include "matlab.h"

mxArray *format;               /* String array(s) */
mxArray *s;                    /* Required input argument(s) */
mxArray *size;                 /* Optional input argument(s) */
mxArray *count = NULL;         /* Optional output argument(s) */
mxArray *errmsg = NULL;        /* Optional output argument(s) */
mxArray *nextindex = NULL;     /* Optional output argument(s) */
mxArray *A = NULL;             /* Return value */

mlfAssign(&A, mlfSscanf(NULL,NULL,NULL,s,format,NULL));
mlfAssign(&A, mlfSscanf(NULL,NULL,NULL,s,format,size));
mlfAssign(&A, mlfSscanf(&count,&errmsg,&nextindex,s,format,NULL));
mlfAssign(&A, mlfSscanf(&count,&errmsg,&nextindex,s,format,size));
```

MATLAB Syntax

```
A = sscanf(s,format)
A = sscanf(s,format,size)
[A,count,errmsg,nextindex] = sscanf(...)
```

See Also

MATLAB sscanf Calling Conventions

mlfStd

Purpose

Standard deviation

C Prototype

```
mxArray *mlfStd(mxArray *x, mxArray *flag, mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *x;                 /* Required input argument(s) */
mxArray *flag, *dim;        /* Optional input argument(s) */
mxArray *s = NULL;          /* Return value */

mlfAssign(&s, mlfStd(x,NULL,NULL));
mlfAssign(&s, mlfStd(x,flag,NULL));
mlfAssign(&s, mlfStd(x,flag,dim));
```

MATLAB Syntax

```
s = std(X)
s = std(X,flag)
s = std(X,flag,dim)
```

See Also

MATLAB `std` Calling Conventions

Purpose

Convert string to double-precision value

C Prototype

```
mxArray *mlfStr2double(mxArray *s);
```

C Syntax

```
#include "matlab.h"

mxArray *C;                 /* Required input argument(s) */
mxArray *x=NULL, *X=NULL;   /* Return value */

mlfAssign(&x, mlfStr2double(mxCreateString("str")));
mlfAssign(&X, mlfStr2double(C));
```

MATLAB Syntax

```
x = str2double('str')
X = str2double(C)
```

See Also

MATLAB `str2double` Calling Conventions

mlfStr2mat

Purpose

Form blank padded character matrix from strings.

C Prototype

```
mxArray *mlfStr2mat(mxArray *str1, ... );
```

C Syntax

```
#include "matlab.h"

mxArray *S = NULL;          /* Return value */

mlfAssign(&S, mlfStr2mat(mxCreateString("str1"),NULL));
mlfAssign(&S, mlfStr2mat(mxCreateString("str1"),
                         mxCreateString("str2"),...,NULL));
```

MATLAB Syntax

```
S = str2mat(t1,t2,t3,...)
```

See Also

MATLAB `str2mat` Calling Conventions

Purpose

String to number conversion

C Prototype

```
mxArray *mlfStr2num(mxArray *str);
```

C Syntax

```
#include "matlab.h"

mxArray *x = NULL;      /* Return value */

mlfAssign(&x, mlfStr2num(mxCreateString("str")));
```

MATLAB Syntax

```
x = str2num('str')
```

See Also

MATLAB `str2num` Calling Conventions

mlfStrcat

Purpose

String concatenation

Minimum number of arguments: two. Maximum: user-defined. Terminate all arguments lists with a `NULL`.

C Prototype

```
mxArray *mlfStrcat(mxArray *s1, ... );
```

C Syntax

```
#include "matlab.h"

mxArray *s1, *s2;           /* Required input argument(s) */
mxArray *s3;                /* Optional input argument(s) */
mxArray *t = NULL;          /* Return value */

mlfAssign(&t, mlfStrcat(s1,s2,s3,...,NULL));
```

MATLAB Syntax

```
t = strcat(s1,s2,s3,...)
```

See Also

MATLAB `strcat` Calling Conventions

Purpose

Compare strings

C Prototype

```
mxArray *mlfStrcmp(mxArray *str1, mxArray *str2);
```

C Syntax

```
#include "matlab.h"

mxArray *S, *T;                 /* Input argument(s) */
mxArray *k = NULL, *TF = NULL;  /* Return value */

mlfAssign(&k, mlfStrcmp(mxCreateString("str1"),
                        mxCreateString("str2")));
mlfAssign(&TF, mlfStrcmp(S,T));
```

MATLAB Syntax

```
k = strcmp('str1','str2')
TF = strcmp(S,T)
```

See Also

MATLAB `strcmp` Calling Conventions

mlfStrcmpi

Purpose

Compare strings ignoring case

C Prototype

```
mxArray *mlfStrcmpi(mxArray *str1, mxArray *str2);
```

C Syntax

```
#include "matlab.h"

mxArray *S, *T;                 /* Input argument(s) */
mxArray *k = NULL, *TF = NULL;  /* Return value */

mlfAssign(&k, mlfStrcmpi(mxCreateString("str1"),
                         mxCreateString("str2")));
mlfAssign(&TF, mlfStrcmpi(S,T));
```

MATLAB Syntax

```
strcmpi('str1','str2')
strcmpi(S,T)
```

See Also

MATLAB `strcmpi` Calling Conventions

Purpose

Justify a character array

C Prototype

```
mxArray *mlfStrjust(mxArray *S, mxArray *justify);
```

C Syntax

```
#include "matlab.h"

mxArray *S;                 /* Required input argument(s) */
mxArray *T = NULL;          /* Return value */

mlfAssign(&T, mlfStrjust(S, NULL));
mlfAssign(&T, mlfStrjust(S, mxCreateString("right")));
mlfAssign(&T, mlfStrjust(S, mxCreateString("left")));
mlfAssign(&T, mlfStrjust(S, mxCreateString("center")));
```

MATLAB Syntax

```
T = strjust(S)
T = strjust(S,'right')
T = strjust(S,'left')
T = strjust(S,'center')
```

See Also

MATLAB `strjust` Calling Conventions

mlfStrmatch

Purpose

Find possible matches for a string

C Prototype

```
mxArray *mlfStrmatch(mxArray *str, mxArray *STRS, mxArray *flag);
```

C Syntax

```
#include "matlab.h"

mxArray *STRS;              /* Required input argument(s) */
mxArray *i=NULL;            /* Return value */

mlfAssign(&i, mlfStrmatch(mxCreateString("str"),STRS,NULL));
mlfAssign(&i, mlfStrmatch(mxCreateString("str"),STRS,
                          mxCreateString("exact")));
```

MATLAB Syntax

```
i = strmatch('str',STRS)
i = strmatch('str',STRS,'exact')
```

See Also

MATLAB `strmatch` Calling Conventions

Purpose

Compare the first n characters of two strings

C Prototype

```
mxArray *mlfStrncmp(mxArray *str1, mxArray *str2, mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *S, *T, *n;             /* Input argument(s) */
mxArray *k = NULL, *TF = NULL;  /* Return value */

mlfAssign(&k, mlfStrncmp(mxCreateString("str1"),
                         mxCreateString("str2"),n));
mlfAssign(&TF, mlfStrncmp(S,T,n));
```

MATLAB Syntax

```
k = strncmp('str1','str2',n)
TF = strncmp(S,T,n)
```

See Also

MATlAB `strncmp` Calling Conventions

mlfStrncmpi

Purpose

Compare the first n characters of two strings, ignoring case

C Prototype

```
mxArray *mlfStrncmpi(mxArray *str1, mxArray *str2);
```

C Syntax

```
#include "matlab.h"

mxArray *S, *T;                 /* Input argument(s) */
mxArray *k = NULL, *TF = NULL;  /* Return value */

mlfAssign(&k, mlfStrncmpi(mxCreateString("str1"),
                          mxCreateString("str2")));
mlfAssign(&TF, mlfStrncmpi(S,T));
```

MATLAB Syntax

```
strncmpi('str1','str2',n)
TF = strncmpi(S,T,n)
```

See Also

MATlAB `strncmpi` Calling Conventions

Purpose

String search and replace

C Prototype

```
mxArray *mlfStrrep(mxArray *str1, mxArray *str2, mxArray *str3);
```

C Syntax

```
#include "matlab.h"

mxArray *str = NULL;            /* Return value */

mlfAssign(&str, mlfStrrep(mxCreateString("str1"),
                          mxCreateString("str2"),
                          mxCreateString("str3")));
```

MATLAB Syntax

```
str = strrep('str1','str2','str3')
```

See Also

MATLAB `strrep` Calling Conventions

mlfStrtok

Purpose

First token in string

C Prototype

```
mxArray *mlfStrtok(mxArray **rem, mxArray *str, mxArray *delimiter);
```

C Syntax

```
#include "matlab.h"

mxArray *delimiter;     /* Optional input argument(s) */
mxArray *rem = NULL;    /* Optional output argument(s) */
mxArray *token = NULL;  /* Return value */

mlfAssign(&token, mlfStrtok(NULL,mxCreateString("str"),
                            delimiter));
mlfAssign(&token, mlfStrtok(NULL,mxCreateString("str"),NULL));
mlfAssign(&token, mlfStrtok(&rem,mxCreateString("str"),NULL));
mlfAssign(&token, mlfStrtok(&rem,mxCreateString("str"),
                            delimiter));
```

MATLAB Syntax

```
token = strtok('str',delimiter)
token = strtok('str')
[token,rem] = strtok(...)
```

See Also

MATLAB `strtok` Calling Conventions

Purpose

Create structure array

C Prototype

```
mxArray *mlfStruct(mxArray *field1, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *values1, *values2; /* Optional input argument(s) */
mxArray *s = NULL;          /* Return value */

mlfAssign(&s, mlfStruct(mxCreateString("field1"),values1,
                        mxCreateString("field2"),values2,
                        ...,
                        NULL));
```

MATLAB Syntax

```
s = struct('field1',values1,'field2',values2,...)
```

See Also

MATLAB `struct` Calling Conventions

mlfStruct2cell

Purpose

Structure to cell array conversion

C Prototype

```
mxArray *mlfStruct2cell(mxArray *s);
```

C Syntax

```
#include "matlab.h"

mxArray *s;                 /* Required input argument(s) */
mxArray *c = NULL;          /* Return value */

mlfAssign(&c, mlfStruct2cell(s));
```

MATLAB Syntax

```
c = struct2cell(s)
```

See Also

MATLAB `struct2cell` Calling Conventions

Purpose

Vertical concatenation of strings

Minimum number of arguments: two, maximum: user-defined. Terminate the list of arguments with a `NULL`.

C Prototype

```
mxArray *mlfStrvcat(mxArray *t1, ... );
```

C Syntax

```
#include "matlab.h"

mxArray *t1;            /* Required input argument(s) */
mxArray *t2, *t3;       /* Optional input argument(s) */
mxArray *S = NULL;      /* Return value */

mlfAssign(&S, mlfStrvcat(t1,t2,NULL));
mlfAssign(&S, mlfStrvcat(t1,t2,t3,...,NULL));
```

MATLAB Syntax

```
S = strvcat(t1,t2,t3,...)
```

See Also

MATLAB `strvcat` Calling Conventions

mlfSub2ind

Purpose

Single index from subscript

Minimum number of arguments: two, maximum: user-defined. Terminate the list of arguments with a `NULL`.

C Prototype

```
mxArray *mlfSub2ind(mxArray *siz, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *siz, *I, *J;
mxArray *I1, *I2;
mxArray *IND = NULL;    /* Return value */

mlfAssign(&IND, mlfSub2ind(siz,I,J,NULL));
mlfAssign(&IND, mlfSub2ind(siz,I1,I2,...,NULL));
```

MATLAB Syntax

```
IND = sub2ind(siz,I,J)
IND = sub2ind(siz,I1,I2,...,In)
```

Purpose

Angle between two subspaces

C Prototype

```
mxArray *mlfSubspace(mxArray *A, mxArray *B);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *B;         /* Required input argument(s) */
mxArray *theta = NULL;  /* Return value */

mlfAssign(&theta, mlfSubspace(A,B));
```

MATLAB Syntax

```
theta = subspace(A,B)
```

See Also

MATLAB subspace Calling Conventions

mlfSum

Purpose

Sum of array elements

C Prototype

```
mxArray *mlfSum(mxArray *A, mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *sum;           /* Optional output argument(s) */
mxArray *B = NULL;      /* Return value */

mlfAssign(&B, sum(A,NULL));
mlfAssign(&B, sum(A,dim));
```

MATLAB Syntax

```
B = sum(A)
B = sum(A,dim)
```

See Also

MATLAB `sum` Calling Conventions

Purpose

Singular value decomposition

C Prototype

```
mxArray *mlfSvd(mxArray **S, mxArray **V, mxArray *X,
                mxArray *Zero);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *S, *V;         /* Optional output argument(s) */
mxArray *U = NULL, *s = NULL;/* Return value */

mlfAssign(&s, mlfSvd(NULL,NULL,X,NULL));
mlfAssign(&U, mlfSvd(&S,&V,X,NULL));
mlfAssign(&U, mlfSvd(&S,&V,X,mlfScalar(0)));
```

MATLAB Syntax

```
s = svd(X)
[U,S,V] = svd(X)
[U,S,V] = svd(X,0)
```

See Also

MATLAB svd Calling Conventions

mlfSvds

Purpose

A few singular values

C Prototype

```
mxArray *mlfSvds(mxArray **S, mxArray **V, mxArray **flag, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *A;                     /* Required input argument(s) */
mxArray *k;                     /* Optional input argument(s) */
mxArray *S = NULL, *V = NULL;   /* Optional output argument(s) */
mxArray *s = NULL, *U = NULL;   /* Return value */

mlfAssign(&s, mlfSvds(NULL,NULL,A,NULL));
mlfAssign(&s, mlfSvds(NULL,NULL,A,k,NULL));
mlfAssign(&s, mlfSvds(NULL,NULL,A,k,mlfScalar(0)));

mlfAssign(&U, mlfSvds(&S,&V,A,NULL,NULL));
mlfAssign(&U, mlfSvds(&S,&V,A,k,NULL));
mlfAssign(&U, mlfSvds(&S,&V,A,k,mlfScalar(0)));
```

MATLAB Syntax

```
s = svds(A)
s = svds(A,k)
s = svds(A,k,0)
[U,S,V] = svds(A,...)
```

See Also

MATLAB svds Calling Conventions

Purpose

Sparse symmetric minimum degree ordering

C Prototype

```
mxArray *mlfSymmmd(mxArray *S);
```

C Syntax

```
#include "matlab.h"

mxArray *S;                 /* Required input argument(s) */
mxArray *p = NULL;          /* Return value */

mlfAssign(&p, mlfSymmmd(S));
```

MATLAB Syntax

```
p = symmmd(S)
```

See Also

MATLAB `symmmd` Calling Conventions

mlfSymrcm

Purpose

Sparse reverse Cuthill-McKee ordering

C Prototype

```
mxArray *mlfSymrcm(mxArray *S);
```

C Syntax

```
#include "matlab.h"

mxArray *S;                 /* Required input argument(s) */
mxArray *r = NULL;          /* Return value */

mlfAssign(&r, mlfSymrcm(S));
```

MATLAB Syntax

```
r = symrcm(S)
```

See Also

MATLAB `symrcm` Calling Conventions

mlfTan, mlfTanh

Purpose

Tangent and hyperbolic tangent

C Prototype

```
mxArray *mlfTan(mxArray *X);
mxArray *mlfTanh(mxArray *X);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *Y = NULL;      /* Return value */

mlfAssign(&Y, mlfTan(X));
mlfAssign(&Y, mlfTanh(X));
```

MATLAB Syntax

```
Y = tan(X)
Y = tanh(X)
```

See Also

MATLAB `tan`, `tanh` Calling Conventions

Purpose

Stopwatch timer

C Prototype

```
void mxArray *mlfTic(void);
mxArray *mlfToc(void);
void mlfVToc(void);
```

C Syntax

```
#include "matlab.h"

mxArray *t = NULL;      /* Return value */

mlfTic();
    any statements
mlfVToc();
mlfAssign(&t, mlfToc());
```

MATLAB Syntax

```
tic
    any statements
toc
t = toc
```

See Also

MATLAB `tic`, `toc` Calling Conventions

mlfTobool

Purpose

Convert an array to a Boolean value by reducing the rank of the array to a scalar

C Prototype

```
bool mlfTobool(mxArray *t);
```

C Syntax

```
#include "matlab.h

mxArray *A;             /* Input argument(s) */

/* equivalent to: if(A != 0) */

if(mlfTobool(mlfNe(A,mlfScalar(0))))
    {
        /* test succeeded, do something */
    }
```

See Also

Calling Conventions

Purpose

Toeplitz matrix

C Prototype

```
mxArray *mlfToeplitz(mxArray *c, mxArray *r);
```

C Syntax

```
#include "matlab.h"

mxArray *r;             /* Required input argument(s) */
mxArray *c;             /* Optional input argument(s) */
mxArray *T = NULL;      /* Return value */

mlfAssign(&T, mlfToeplitz(c,r));
mlfAssign(&T, mlfToeplitz(r,NULL));
```

MATLAB Syntax

```
T = toeplitz(c,r)
T = toeplitz(r)
```

See Also

MATLAB `toeplitz` Calling Conventions

mlfTrace

Purpose

Sum of diagonal elements

C Prototype

```
mxArray *mlfTrace(mxArray *a);
```

C Syntax

```
#include "matlab.h"

mxArray *A;             /* Required input argument(s) */
mxArray *b = NULL;      /* Return value */

mlfAssign(&b, mlfTrace(A));
```

MATLAB Syntax

```
b = trace(A)
```

See Also

MATLAB `trace` Calling Conventions

Purpose

Trapezoidal numerical integration

C Prototype

```
mxArray *mlfTrapz(mxArray *Y, mxArray *X, mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *Y;             /* Required input argument(s) */
mxArray *X;             /* Optional input argument(s) */
mxArray *Z = NULL;      /* Return value */

mlfAssign(&Z, mlfTrapz(Y,NULL,NULL));
mlfAssign(&Z, mlfTrapz(X,Y,NULL));
mlfAssign(&Z, mlfTrapz(Y,dim,NULL));
mlfAssign(&Z, mlfTrapz(X,Y,dim));
```

MATLAB Syntax

```
Z = trapz(Y)
Z = trapz(X,Y)
Z = trapz(...,dim)
```

See Also

MATLAB `trapz` Calling Conventions

mlfTril

Purpose

Lower triangular part of a matrix

C Prototype

```
mxArray *mlfTril(mxArray *X, mxArray *k);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *k;             /* Optional input argument(s) */
mxArray *L = NULL;      /* Return value */

mlfAssign(&L, mlfTril(X,NULL));
mlfAssign(&L, mlfTril(X,k));
```

MATLAB Syntax

```
L = tril(X)
L = tril(X,k)
```

See Also

MATLAB `tril` Calling Conventions

Purpose

Upper triangular part of a matrix

C Prototype

```
mxArray *mlfTriu(mxArray *X, mxArray *k);
```

C Syntax

```
#include "matlab.h"

mxArray *X;             /* Required input argument(s) */
mxArray *k;             /* Optional input argument(s) */
mxArray *U = NULL;      /* Return value */

mlfAssign(&U, mlfTriu(X,NULL));
mlfAssign(&U, mlfTriu(X,k));
```

MATLAB Syntax

```
U = triu(X)
U = triu(X,k)
```

See Also

MATLAB `triu` Calling Conventions

mlfUnion

Purpose

Set union of two vectors

C Prototype

```
mxArray *mlfUnion(mxArray **ia, mxArray **ib, mxArray *a,
                  mxArray *b, mxArray *rows_str);
```

C Syntax

```
#include "matlab.h"

mxArray *a, *b, *A, *B;     /* Input argument(s) */
mxArray *ia, *ib;           /* Optional output argument(s) */
mxArray *c = NULL;          /* Return value */

mlfAssign(&c, mlfUnion(NULL,NULL,a,b,NULL));
mlfAssign(&c, mlfUnion(NULL,NULL,A,B,mxCreateString("rows")));
mlfAssign(&c, mlfUnion(&ia,&ib,a,b,NULL));
mlfAssign(&c, mlfUnion(&ia,&ib,A,B,mxCreateString("rows")));
```

MATLAB Syntax

```
c = union(a,b)
c = union(A,B,'rows')
[c,ia,ib] = union(...)
```

See Also

MATLAB `union` Calling Conventions

Purpose

Unique elements of a vector

C Prototype

```
mxArray *mlfUnique(mxArray **i, mxArray **j, mxArray *a,
                   mxArray *rows_str);
```

C Syntax

```
#include "matlab.h"

mxArray *a, *A;                 /* Input argument(s) */
mxArray *i = NULl, *j = NULL;   /* Optional output argument(s) */
mxArray *b = NULL;              /* Return value */

mlfAssign(&b, mlfUnique(NULL,NULL,a,NULL));
mlfAssign(&b, mlfUnique(NULL,NULL,A,mxCreateString("rows")));
mlfAssign(&b, mlfUnique(&i,&j,a,NULL));
mlfAssign(&b, mlfUnique(&i,&j,A,mxCreateString("rows")));
```

MATLAB Syntax

```
b = unique(a)
b = unique(A,'rows')
[b,i,j] = unique(...)
```

See Also

MATLAB unique Calling Conventions

mlfUnwrap

Purpose

Correct phase angles

C Prototype

```
mxArray *mlfUnwrap(mxArray *P, mxArray *tol, mxArray *dim);
```

C Syntax

```
#include "matlab.h"

mxArray *P;                     /* Required input argument(s) */
mxArray *null_matrix = NULL;    /* Optional input argument(s) */
mxArray *tol, *dim;             /* Optional input argument(s) */
mxArray *Q = NULL;              /* Return value */

mlfAssign(&Q, mlfUnwrap(P,NULL,NULL));
mlfAssign(&Q, mlfUnwrap(P,tol,NULL));

null_matrix = mlfZeros(mlfScalar(0),mlfScalar(0),NULL);
mlfAssign(&Q, mlfUnwrap(P,null_matrix,dim));

mlfAssign(&Q, mlfUnwrap(P,tol,dim));
```

MATLAB Syntax

```
Q = unwrap(P)
Q = unwrap(P,tol)
Q = unwrap(P,[],dim)
Q = unwrap(P,tol,dim)
```

See Also

MATLAB `unwrap` Calling Conventions

Purpose

Convert string to upper case

C Prototype

```
mxArray *mlfUpper(mxArray *str);
```

C Syntax

```
#include "matlab.h"

mxArray *str;           /* String array(s) */
mxArray *t = NULL;      /* Return value */

mlfAssign(&t, mlfUpper(str));
```

MATLAB Syntax

```
t  = upper('str')
```

See Also

MATLAB upper

Calling Conventions

mlfVander

Purpose

Test matrix (Vandermonde matrix)

C Prototype

```
mxArray *mlfVander(mxArray *c);
```

C Syntax

```
#include "matlab.h"

mxArray c;                  /* Required input argument(s) */
mxArray A = NULL;           /* Return value */

mlfAssign(&A, mlfVander(c));
```

MATLAB Syntax

```
A = vander(c);
```

See Also

MATLAB `gallery` Calling Conventions

Purpose

Vertical concatenation

Minimum number of arguments: two, maximum: user-defined. Terminate the list of arguments with a `NULL`.

C Prototype

```
mxArray *mlfVertcat(mxArray *A, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *B;             /* Required input argument(s) */
mxArray *C;                 /* Optional output argument(s) */
mxArray *R = NULL;          /* Return value */

mlfAssign(&R, mlfVertcat(A,B,C,...,NULL));
```

MATLAB Syntax

```
[A;B;C...]
vertcat(A,B,C...)
```

See Also

MATLAB `cat` Calling Conventions

mlfWarning

Purpose

Display warning message

C Prototype

```
mxArray *mlfWarning(mxArray **f, mxArray *message);
```

C Syntax

```
#include "matlab.h"

mxArray *f;             /* Optional output argument(s) */
mxArray *s = NULL;      /* Return value */

mlfAssign(&s, mlfWarning(NULL,mxCreateString("I'm sorry Dave")));
mlfAssign(&s, mlfWarning(NULL,mxCreateString("on")));
mlfAssign(&s, mlfWarning(NULL,mxCreateString("off")));
mlfAssign(&s, mlfWarning(NULL,mxCreateString("backtrace")));
mlfAssign(&s, mlfWarning(NULL,mxCreateString("debug")));
mlfAssign(&s, mlfWarning(NULL,mxCreateString("once")));
mlfAssign(&s, mlfWarning(NULL,mxCreateString("always")));
mlfAssign(&s, mlfWarning(&f,NULL));
```

MATLAB Syntax

```
warning('message')
warning on
warning off
warning backtrace
warning debug
warning once
warning always
[s,f] = warning
```

See Also

MATLAB `warning` Calling Conventions

Purpose

Day of the week

C Prototype

```
mxArray *mlfWeekday(mxArray **S, mxArray *D);
```

C Syntax

```
#include "matlab.h"

mxArray *D;                 /* Required input argument(s) */
mxArray *S;                 /* Required output argument(s) */
mxArray *N = NULL;          /* Return value */

mlfAssign(&N, mlfWeekday(&S, D));
```

MATLAB Syntax

```
[N,S] = weekday(D)
```

See Also

MATLAB `weekday` Calling Conventions

mlfWilkinson

Purpose

Wilkinson's eigenvalue test matrix

C Prototype

```
mxArray *mlfWilkinson(mxArray *n);
```

C Syntax

```
#include "matlab.h"

mxArray *n;             /* Required input argument(s) */
mxArray *W = NULL;      /* Return value */

mlfAssign(&W, mlfWilkinson(n));
```

MATLAB Syntax

```
W = wilkinson(n)
```

See Also

MATLAB `wilkinson` Calling Conventions

Purpose

Exclusive OR

C Prototype

```
mxArray *mlfXor(mxArray *A, mxArray *B);
```

C Syntax

```
#include "matlab.h"

mxArray *A, *B;         /* Required input argument(s) */
mxArray *C = NULL;      /* Return value */

mlfAssign(&C, mlfXor(A,B));
```

MATLAB Syntax

```
C = xor(A,B)
```

See Also

MATLAB `xor` Calling Conventions

mlfZeros

Purpose

Create an array of all zeros

C Prototype

```
mxArray *mlfZeros(mxArray *in1, ...);
```

C Syntax

```
#include "matlab.h"

mxArray *m, *n;             /* Input argument(s) */
mxArray *A;                 /* Input argument(s) */
mxArray *d1, *d2, *d3;      /* Input argument(s) */
mxArray *B = NULL;          /* Return value */

mlfAssign(&B, mlfZeros(n,NULL));
mlfAssign(&B, mlfZeros(m,n,NULL));
mlfAssign(&B, mlfZeros(mlfHorzcat(m,n,NULL),NULL));
mlfAssign(&B, mlfZeros(d1,d2,d3,...,NULL));
mlfAssign(&B, mlfZeros(mlfHorzcat(d1,d2,d3,...,NULL),NULL));
mlfAssign(&B, mlfZeros(mlfSize(NULL,A,NULL),NULL));
```

MATLAB Syntax

```
B = zeros(n)
B = zeros(m,n)
B = zeros([m n])
B = zeros(d1,d2,d3...)
B = zeros([d1 d2 d3...])
B = zeros(size(A))
```

See Also

MATLAB `zeros` Calling Conventions

Utility Routine Reference

This section contains all the MATLAB C Math Library utility routines. These routines provide array creation, array indexing, and other capabilities.

Purpose

Handles assignments that include one and two-dimensional indexing.

This routine is superseded by the `mlfIndexAssign()` routine, which supports multidimensional, cell array, and structure indexing.

C Prototype

```
void mlfArrayAssign(mxArray *destination, mxArray *source, ... );
```

Arguments

`mxArray *destination`
Specifies the destination array that will be modified.

`mxArray *source`
Specifies the source array that contains the new values for the destination array.

`optional mxArray* arguments`
Specify one or two indices that form the subscript for the *destination* array.

Terminate the argument list by passing `NULL` as the last argument.

Return

This function returns `void`. The result of the assignment is stored in the argument `destination`.

Description

Use the function `mlfArrayAssign()` to make assignments that involve indexing. The arguments to `mlfArrayAssign()` consist of a destination array, a source array, and one or two index arrays that represent the subscript. The subscript specifies the elements that are to be modified in the destination array; the source array specifies the new values for those elements. The subscript is only applied to the destination array.

The functions are defined to accept a variable number of indices. Supply one index `mxArray` argument to perform one-dimensional indexing. Supply two index `mxArray` arguments to perform two-dimensional indexing.

Example

```
mxArray *fortyfive = mlfScalar(45);
mxArray *three = mlfScalar(3);
mxArray *one = mlfScalar(1);
mlfArrayAssign(A, fortyfive, three, one, NULL);
```

writes the value 45 into the element at row three, column one of array `A`. If you assign a value to a location that does not exist in the array, the array grows to include that element.

See Also `mlfArrayDelete`, `mlfArrayRef`, `mlfColon`, `mlfCreateColonIndex`, `mlfEnd`

Purpose

Delete elements from a one or two-dimensional array.

This routine is superseded by the `mlfIndexDelete()` routine, which supports multidimensional, cell array, and structure indexing.

C Prototype

```
void mlfArrayDelete(mxArray *destination, mxArray *index1, ... );
```

Arguments

`mxArray *destination`
Specifies the array that you want to delete elements from.

`mxArray *index1`
Specifies an index that is used to form the subscript.

`optional mxArray* arguments`
Additional index arguments that are used to form the subscript.

Terminate the argument list by passing `NULL` as the last argument.

Return

This function returns `void`. The result of the deletion is stored in the argument `destination`.

Description

Use the function `mlfArrayDelete()` to delete elements from an array. This function is equivalent to the MATLAB statement, `A(B) = []`. Instead of specifying a subscript for the elements you want to replace with other values, specify a subscript for the elements you want removed from the array. The MATLAB C Math Library removes those elements and shrinks the array.

When you delete a single element from a matrix, the matrix is converted into a row vector that contains one fewer element than the original matrix. You can also delete more than one element from a matrix, shrinking the matrix by that number of elements. To retain the rectangularity of the matrix, however, you must delete one or more entire rows or columns.

Example

```
mlfArrayDelete(A, three, one, NULL);
```

This function removes the element at row three, column one from array `A`. Note that removing an element from a matrix reshapes the matrix into a vector.

See Also

`mlfArrayAssign`, `mlfArrayRef`, `mlfColon`, `mlfCreateColonIndex`, `mlfEnd`

mlfArrayRef

Purpose

Handles one and two-dimensional indexed array references.

This routine is superseded by the `mlfIndexRef()` routine, which supports multidimensional, cell array, and structure indexing.

C Prototype

```
mxArray *mlfArrayRef(mxArray *array, ... );
```

Arguments

`mxArray *array`
Specifies the target array.

`optional mxArray* arguments`
Specify the indices that form the subscript. Pass one index for one-dimensional indexing. Pass two indices for two-dimensional indexing.

Terminate the argument list by passing `NULL` as the last argument.

Return

This function returns a pointer to a newly allocated `mxArray` that contains the result of the indexing operation.

Description

`mlfArrayRef()` extracts the elements specified by the subscript from the target array and returns the result in a new `mxArray`. `mlfArrayRef()` is the only indexing function to return a value.

Example

```
mxArray *two = mlfScalar(2), B;
B = mlfArrayRef(A, two, two, NULL);
```

This statement selects the element at row 2, column 2 in array `A` and returns it in `B`.

See Also

`mlfArrayAssign`, `mlfArrayDelete`, `mlfColon`, `mlfCreateColonIndex`, `mlfEnd`

Purpose

Assign an array value to a variable

C Prototype

```
mxArray *mlfAssign(mxArray *volatile *dest, mxArray *src);
```

Arguments

`mxArray **dest`
The address of a pointer to the target array (the left-hand side of an assignment statement). You must initialize `*dest` to `NULL` or to a valid array.

`mxArray *src`
A pointer to the value you want to assign (the right-hand side of an assignment statement)

Return

Returns `*dest`, the pointer to the target array.

Description

By default, all the arrays returned by the MATLAB C Math Library routines are *temporary* arrays. This allows you to nest calls to library routines as arguments to other routines. You do not need to deallocate the arrays returned by the nested calls; the library routines delete them automatically. (Arrays that are returned by routines that you write, using `mlfReturnValue()`, are also temporary.)

To make an array persist, you must *bind* the array to a variable using the `mlfAssign()` routine. This routine replaces the standard C assignment operator (=). You must explicitly free arrays that are bound to variables.

`mlfAssign()` assigns `src` to `*dest`. `src` points to the source array. `*dest` is equivalent to the left-hand side of an assignment statement. `src` is equivalent to the right-and side of an assignment statement.

If `*dest` already points to a valid array, `mlfAssign()` destroys that array before assigning the source array to it. However, if `*dest` points to an input argument to the current function, different rules apply. If `*dest` points to a temporary array, it is destroyed; if it points to a bound array, the assignment does not take place.

Functions that take output arguments (`mxArray**` arguments), including the indexed assignment functions, follow the same rules as `mlfAssign()` when deleting existing valid arrays.

If `src`, the right-hand side of the assignment, points to a bound array, `*dest`, receives a copy of the array. The copy is a *shared-data* copy. The actual data associated with the array is not copied until a function modifies the data in the

array, for example, a call to `mlfIndexAssign()` modifies two rows of an array. At that point the data itself is copied to the array before the modifications are made, and the array is no longer points to shared data.

For example,

```
mlfIndexAssign(&B, "(?,?)",
                   mlfScalar(2), mlfScalar(2),
                   mlfScalar(0));
```

modifies the value at position (2,2) in array `B`. At that point, the library copies the data to itself before the modifications are made, and the array no longer points to shared data.

Shared data itself is not freed until all arrays that use the data have been destroyed. The functions `mxGetPr()` and `mxGetPi()` that access data stored in an array directly handle shared data correctly; `mxSetPr()` and `mxSetPi()` modify the data correctly.

Example

This code assigns the matrix product of array `Q` and array `R` to `*Z`

```
mlfAssign(&Z, mlfMtimes(Q, R));
```

where `Q`, `R`, and `Z` are `mxArray*` variables. `Z` is initialized to `NULL` and `Q` and `R` point to existing arrays.

If you decide not to use the automated memory management features of the library, this code performs the same matrix multiplication.

```
Z = mlfMtimes(Q, R);
```

See Also

`mlfIndexAssign`

Purpose

Create vectors and use in array subscripting

C Prototype

```
mxArray *mlfColon(mxArray *start, mxArray *step, mxArray *end);
```

Arguments

`mxArray *start`
Initial value.

`mxArray *step`
Increment value, or final value if only start and end values are specified.

`mxArray *end`
Final value, `NULL` if only start and end values are passed.

Description

This function lets you specify a vector index.

Example

This example specifies the vector `[ 1 2 3 4 5 6 7 8 9 10 ]`.

```
mxArray *vector_index = NULL;

mlfAssign(&vector_index,
          mlfColon(mlfScalar(1), mlfScalar(10), NULL));
```

This example is equivalent to a call to `mlfCreateColonIndex()`.

```
mxArray *colon = NULL;

mlfAssign(&colon, mlfColon(NULL,NULL,NULL));
```

See Also

`mlfIndexAssign`, `mlfIndexDelete`, `mlfIndexRef`, `mlfCreateColonIndex`, `mlfEnd`

mlfComplexScalar

Purpose

Create and initialize a complex 1-by-1 array

C Prototype

```
mxArray *mlfComplexScalar(double v, double i);
```

Arguments

`double v`
Initial content of the real part of the array.

`double i`
Initial content of the imaginary part of the array.

Description

This function creates a complex 1-by-1 array whose contents are initialized to the real part, `v`, and the imaginary part, `i`.

See Also

`mlfScalar`

Purpose

Create an array that acts like the colon operator when passed as an index to an indexing function

C Prototype

```
mxArray *mlfCreateColonIndex(void);
```

Description

The `mlfCreateColonIndex()` index, which loosely interpreted means "all," selects, for example, all the columns in a row or all the rows in a column.

Example

The call to `mlfIndexRef()` selects all the elements in the first row of array `A` and assigns them to array `B`.

```
mlfAssign(&B, mlfIndexRef(A,
                          "(?,?)",         /* Format string */
                          mlfScalar(1),    /* Index value   */
                          mlfCreateColonIndex()); /* Colon  */
```

See Also

`mlfIndexAssign`, `mlfIndexDelete`, `mlfIndexRef`, `mlfColon`, `mlfEnd`

mlfDoubleMatrix

Purpose

Create a matrix of double precision values

C Prototype

```
mxArray *mlfDoubleMatrix(int m, int n, const double *pr,
                                       const double *pi);
```

Arguments

`int m`
Number of rows.

`int n`
Number of columns.

`const double *pr`
Pointer to values to initialize the `mxArray` array vector of real values.

`const double *pi`
Pointer to values to initialize the `mxArray` array vector of imaginary values. Specify `NULL` if there is no imaginary part.

Description

This routine creates a complex, two-dimensional array whose contents are initialized to the real part, `pr`, and the imaginary part, `pi`.

Example

This example creates a 3-by-2 matrix of double precision, complex numbers.

```
static double real_data[] = { 1, 2, 3, 4, 5, 6 };
static double cplx_data[] = { 7, 8, 9, 10, 11, 12 };

mxArray *mat1 = NULL;

mlfAssign(&mat1, mlfDoubleMatrix(3, 2, real_data, cplx_data));
```

Purpose

Generate the last index for an array dimension

C Prototype

```
mxArray *mlfEnd(mxArray *array, mxArray *dim, mxArray *numindices);
```

Arguments

`mxArray *array`
Specifies the target array.

`mxArray *dim`
Dimension in the target array for which the last index is determined.

`mxArray *numindices`
Total number of dimensions; that is, the total number of indices in the indexing subscript.

Description

The `mlfEnd()` function, which corresponds to the MATLAB `end()` function, provides another way of specifying a vector index. Given an array, a dimension (1 = row, 2 = column, 3 = page, etc.), and the number of indices in the subscript, `mlfEnd()` returns the index of the last element in the specified dimension. You can then use that scalar array to generate a vector index to be used in one or two-dimensional indexing.

Given the row dimension, `mlfEnd()` returns the number of columns. Given the column dimension, it returns the number of rows. For a matrix and a one-dimensional index, `mlfEnd()` treats the matrix like a vector and returns the number of elements in the matrix. The number of indices in the subscript corresponds to the number of index arguments you pass to `mlfArrayRef()`.

Example

This example extracts the elements in row five of page four in this three-dimensional array. The example first uses `mlfColon()` to create a vector of all the indices along the second dimension. The first argument to `mlfColon()` indicates the vector should start with 1. The second argument is a nested call to `mlfEnd()`, which defines the end value of the vector. The arguments to

mlfEnd() indicate that the target array is C, the dimension being measured is the second dimension, and the total number of subscripts in the index is 3.

```
/* In MATLAB: A(5,21:end,4) */
mlfAssign(&index, mlfColon(mlfScalar(1),
                           mlfEnd(C, mlfScalar(2),mlfScalar(3)),
                           NULL));

mlfAssign(&D, mlfIndexRef(C, "(?,?,?)",  /* Three dimension index */
                          mlfScalar(5),   /* Row    */
                          index,          /* Column */
                          mlfScalar(4))); /* Page   */
```

See Also

mlfIndexAssign, mlfIndexDelete, mlfIndexRef, mlfColon, mlfCreateColonIndex

Purpose

Establish a new memory context for the arrays passed to a function as input and output arguments

C Prototype

```
void mlfEnterNewContext(int nout, int nin, ...);
```

Arguments

`int nout`
Specifies the number of array (`mxArray **`) output arguments passed to the current function. Specify 0 if there are no output arguments or no array output arguments.

`int nin`
Specifies the number of array (`mxArray *`) input arguments. Specify 0 if there are no input arguments or no array input arguments.

`optional mxArray** arguments`
Pass each of the `mxArray**` output arguments that were passed to the current function.

`optional mxArray* arguments`
Pass each of the `mxArray*` input arguments that were passed to the current function.

You only need to list the `mxArray**` and `mxArray*` arguments. For example, if a function takes an argument of type `char*` or `int`, you do not need to include it in the count of output and input arguments or in the list of the arguments themselves.

You do *not* need to terminate the list with `NULL`; the function detects the end of the argument list from the values of `nout` and `nin`.

For more information on array input and output arguments, see the "Calling Conventions" section of the *MATLAB C Math Library User's Guide.*

Description

`mlfEnterNewContext()`, along with `mlfRestorePreviousContext()`, `mlfReturnValue()`, and `mlfAssign()`, implement automated memory management in the MATLAB C Math Library. Each function in the library includes calls to these functions. The functions that you write can also use automated memory management by calling these functions.

A call to `mlfEnterNewContext()` signals that MATLAB C Math Library automated memory management is in effect for the current function. It deletes

any existing contents of the output arguments passed to it and sets the state of any temporary arrays, passed as input arguments, to bound.

`mlfEnterNewContext()` is paired with the function `mlfRestorePreviousContext()`. A call to `mlfEnterNewContext()` is typically the first line of code in a function, following the declaration of local variables. It must precede any calls to functions that take the array input and output parameters as arguments. A matching call to `mlfRestorePreviousContext()` is typically the last line of code immediately preceding the `return` statement.

`mlfEnterNewContext()` also recognizes when the current function was called from a function that does not use automated memory management. In that environment, it ensures that the input arguments, which are all temporary arrays, are handled correctly and not deleted by the automated memory management. Output arguments that do not point to `NULL` or to a valid array are also handled correctly.

Example

For a function defined as follows,

```
mxArray *ArrayMemory(mxArray **z_out, mxArray *x_in,
                     mxArray *y_in)
```

use the following call to `mlfEnterNewContext()` at the beginning of your function.

```
mlfEnterNewContext(1, 2, z_out, x_in, y_in);
```

Use the following call in your `main()` routine.

```
mlfEnterNewContext(0, 0);
```

See Also

`mlfRestorePreviousContext`, `mlfReturnValue`, `mlfAssign`

Purpose

Returns a pointer to the routine to be executed by `mlfFeval()`.

C Prototype

```
mlxFunctionPtr mlfFevalLookup(mxArray *fcn);
```

Arguments

`mxArray *fcn`

Character array specifying name of the routine to be executed.

Description

To specify the routine executed by the `mlfFeval()` routine, you must pass a pointer to the routine as an argument. The `mlfFevalLookup()` routine returns a pointer to the routine named as its only argument.

Example

This example shows how you nest a call to `mlfFevalLookup()` as an argument to `mlfFeval()`.

```
mlfFeval(mlfVarargout(y1,y2,...,NULL),
         mlfFevalLookup(mxCreateString("foo")),
         x1, x2,...,NULL);
```

mlfFevalTableSetup

Purpose

Registers a thunk function table with the MATLAB C Math Library

C Prototype

```
void mlfFevalTableSetup ( mlfFuncTab *mlfUfuncTable );
```

Arguments

`mlfFuncTab *mlfUfuncTable`
Pointer to a local `feval` table. Each entry is composed of a string representing a function name, a pointer to that function, and a pointer to a thunk function that knows how to execute the function.

Description

A call to `mlfFevalTableSetup()` adds the entries in a local table to the MATLAB C Math Library built-in `feval` function table. `mlfFeval()` accesses the library's built-in function table to locate the function pointers that are associated with a given function name.

Purpose

Assign a value to an element (or elements) in the target array

C Prototype

```
mxArray *mlfIndexAssign(mxArray *volatile *pa,
                        const char* index_string, ...);
```

Arguments

`mxArray **pa`
Specifies the address of the target array that will be modified.

`const char* index_string`
A string that specifies the dimensionality of the subscript and the style of indexing (array indexing or cell array indexing) that is applied to the target array. Use a ? in place of each index value, for example, `"(?,?)"`. `mlfIndexRef()` does *not* use index values specified in the subscript string.

`Additional arguments`
[Optional] Arrays that specify the values of the indices followed by the source array. Provide one index for one-dimensional indexing, two for two-dimensional indexing, *n* indices for *n*-dimensional indexing.

You do not need to terminate the list of arguments with `NULL`; the indexing functions can detect the number of expected arguments.

Return

Returns a pointer to the modified array.

Description

Use the function `mlfIndexAssign()` to make array assignments that involve indexing. The arguments to `mlfIndexAssign()` consist of the destination array, an index string that specifies the elements that are to be modified in the destination array, one or more index arrays that specify the value for the subscript, and the source array.

The subscript specifies the elements that are to be modified in the destination array; the source array specifies the new values for those elements. The subscript is only applied to the destination array.

Example

This code writes the value `45` into the element at row three, column one of array `A`. If you assign a value to a location that does not exist in the array, the array grows to include that element.

```
/* In MATLAB: A(3,1) = 45 */
mxArray *A = NULL;
```

```
mlfIndexAssign(&A,             /* Destination array   */
               "(?,?)",        /* Index format string */
               mlfScalar(3),   /* Subscript value     */
               mlfScalar(1),   /* Subscript value     */
                mlfScalar(45)); /* Source array        */
```

See Also `mlfIndexRef`, `mlfIndexDelete`

Purpose

Deletes from the target array the element (or elements) specified by the subscript

C Prototype

```
mxArray *mlfIndexDelete(mxArray *volatile *pa,
                        const char* index_string, ...);
```

Arguments

`mxArray **pa`
Specifies the address of the array you want to delete elements from.

`const char* index_string`
A string that specifies the dimensionality of the subscript and the style of indexing (array indexing or cell array indexing) that is applied to the target array. Use a ? in place of each index value, for example, `"(?,?)"`. `mlfIndexRef()` does *not* use index values specified in the subscript string.

`mxArray* arguments`
[Optional] Arrays that specify the index values.

You do not need to terminate the list of arguments with `NULL`; the indexing functions can detect the number of expected arguments.

Return

Returns a pointer to the modified array.

Description

Use the function `mlfIndexDelete()` to delete elements from an array. This function is equivalent to the MATLAB statement, `A(B) = []`. The MATLAB C Math Library removes the elements and shrinks the array.

When you delete a single element from a matrix, the matrix is converted into a row vector that contains one fewer element than the original array. You can also delete more than one element from an array, shrinking the array by that number of elements. To retain the rectangularity of a matrix, however, you must delete one or more entire rows or columns.

Example

This code removes the element at row three, column one from array `A`. Note that removing an element from a matrix reshapes the matrix into a vector.

```
/* In MATLAB: A(3,1) = [] */
mxArray *A = NULL;
mlfIndexDelete(&A, "(3,1)", mlfScalar(3), mlfScalar(1));
```

See Also

`mlfIndexRef`, `mlfIndexAssign`

mlfIndexRef

Purpose

Extract elements specified by the subscript from the target array and return the result in a new `mxArray`

C Prototype

```
mxArray *mlfIndexRef(mxArray *pa, const char* index_string, ...);
```

Arguments

`mxArray *pa`
Specifies the array that you want to extract elements from.

`const char* index_string`
A string that specifies the dimensionality of the subscript and the style of indexing (array indexing or cell array indexing) that is applied to the target array. Use a ? in the place of each index value, for example, `"(?,?)"`. `mlfIndexRef()` does *not* use index values specified in the subscript string.

`mxArray* arguments`
[Optional] Arrays that specify the values of the indices. Provide one index for one-dimensional indexing, two for two-dimensional indexing, *n* indices for *n*-dimensional indexing.

You do not need to terminate the list of arguments with `NULL`; the indexing functions can detect the number of expected arguments.

Return

Returns a pointer to a new `mxArray` that contains the extracted data.

Description

`mlfIndexRef()`, along with `mlfIndexAssign()` and `mlfIndexDelete()`, provides access to array elements in the MATLAB C Math Library. These routines emulate the MATLAB indexing operator `()`.

`mlfIndexRef()` copies the value of an array element into another array; it does not modify the element in the target array. To assign a value to an array element, use `mlfIndexAssign()`. To delete the value of an array element, use `mlfIndexDelete()`.

Example

This code selects the element at row 2, column 2 in array `A` and returns it in `B`.

```
/* In MATLAB: B = A(2,2) */
mxArray *B = NULL;

mlfAssign(&B, mlfIndexRef(A,            /* Target array       */
                          "(?,?)",      /* Index format string */
```

```
                                        mlfScalar(2), /* Subscript value    */
                                        mlfScalar(2))); /* Subscript value   */
```

See Also `mlfIndexAssign, mlfIndexDelete`

mlfIndexVarargout

Purpose

Build a list of output arguments, some of which are indexed expressions

C Prototype

```
mlfVarargoutList *mlfIndexVarargout(mxArray **ppa, ...);
```

Arguments

`mxArray **ppa`
A pointer to a pointer to an array.

Return

A cell array containing the output arguments.

Description

Constructs a varargout list with index expressions for one or more outputs. Note that this function MUST be called as an input argument for the automatic memory management to work properly. Its result should not be saved in a variable. A null or empty string means there's no index for the output argument. Increments nargout by the number of outputs in the current index expression.When the variable `varargout` appears as the last output argument in the definition of a MATLAB function, that function can return any number of outputs, starting at that position in the argument list.

If you are indexing into any of the arrays that you pass as `varargout` output arguments, you must use `mlfIndexVarargout()` to form the `varargout` list. For indexed arguments, you specify the a pointer to the source array pointer, the index format string, and the index values, just as you would with the `mlfIndexRef()` routine. For nonindexed arguments, you specify the array argument paired with a `NULL` argument.

Example

In this example, output arguments `z` and `n` are indexed expressions. Note the similarity to `mlfIndexRef()` syntax. Because argument `m` is nonindexed, you follow it with a `NULL` argument to indicate that there is no associated indexing syntax with this argument.

```
mxArray *x = NULL, *y = NULL, *z = NULL, *m = NULL, *n = NULL;

mlfAssign(&x, mlfVarargout_Function(&y,
                  mlfIndexVarargout(&z, "(?)", mlfScalar(1),
                                    &m, NULL,
                                &n, "{?}", mlfCreateColonIndex(),
                                    NULL),
                  a, b)));
```

See Also

`mlfVarargout`, `mlfIndexRef`, `mlfIndexAssign`

mlfPrintf

Purpose Format output similar to `printf`

C Prototype

```
int mlfPrintf(const char *fmt, ...);
```

Arguments

`const char *fmt`
String to print. String may include `printf`-style format characters that specify the format for subsequent strings.

Description Uses the installed print handler to display the output.

See Also `mlfPrintMatrix`, `mlfSetPrintHandler`

Purpose Print the contents of an array

C Prototype

```
void mlfPrintMatrix(mxArray *m);
```

Arguments

`mxArray *m`
Array to print

Description `mlfPrintMatrix()` calls the installed print handler.

To print the contents of a cell array, use `mlfCelldisp()`.

See Also `mlfPrintf`, `mlfSetPrintHandler`

mlfRestorePreviousContext

Purpose

Restore the input variables to the memory context at the time of the function call

C Prototype

```
void mlfRestorePreviousContext(int nout, int nin, ...);
```

Arguments

`int nout`
Specifies the number of array (`mxArray **`) output arguments passed to the current function. Specify 0 if there are no output arguments or no array output arguments.

`int nin`
Specifies the number of array (`mxArray *`) input arguments. Specify 0 if there are no input arguments or no array input arguments.

`optional mxArray** arguments`
Pass each of the `mxArray**` output arguments that were passed to the current function.

`optional mxArray* arguments`
Pass each of the `mxArray*` input arguments that were passed to the current function.

You do *not* need to terminate the list with `NULL`; the function detects the end of the argument list from the values of `nout` and `nin`.

For more information on array input and output arguments, see the "Calling Conventions" section of the *MATLAB C Math Library User's Guide*.

Description

`mlfRestorePreviousContext()`, along with `mlfEnterNewContext()`, `mlfReturnValue()`, and `mlfAssign()`, implement automated memory management in the MATLAB C Math Library. Each function in the library includes calls to these functions. The functions that you write can also use automated memory management by calling these functions.

`mlfRestorePreviousContext()` restores the state of the array input arguments to their state at the time of the function call.

`mlfRestorePreviousContext()` then performs an important deletion: it deletes any temporary variables that were passed to the current function. This behavior allows you to nest function calls as arguments to functions that use automated memory management. You do not need to worry about deleting the array returned from nested function.

`mlfRestorePreviousContext()` is paired with the function `mlfEnterNewContext()`. A call to `mlfRestorePreviousContext()` is typically the last line of code immediately preceding the return from a function. A matching call to `mlfEnterNewContext()` begins the function.

`mlfRestorePreviousContext()` also recognizes when the current function was called from a function that does not use automated memory management. In that environment, it does not delete any array input argument that is passed to it.

Example

For a function defined as follows,

```
mxArray *ArrayMemory(mxArray **z_out, mxArray *x_in,
                     mxArray *y_in)
```

use the following call to `mlfRestorePreviousContext()` at the end of your function prior to the return statement.

```
mlfRestorerPreviousContext(1, 2, z_out, x_in, y_in);
```

Use the following call in your `main()` routine.

```
mlfRestorePreviousContext(0, 0);
```

See Also

`mlfEnterNewContext`, `mlfReturnValue`, `mlfAssign`

mlfReturnValue

Purpose

Mark an array as a return value from a function that uses automated memory management

C Prototype

```
mxArray *mlfReturnValue(mxArray *a);
```

Arguments

`mxArray *a`
Pointer to the array that will be the return value from the current function. The value is typically the result of an assignment made within the function or the value of an output argument set by a function call. These arrays are bound arrays at the time of the call.

If you want to return an array input argument, assign it to a variable first and then pass this bound variable to `mlfReturnValue()`. Do not pass an array that is an input argument to `mlfReturnValue()`.

Return

Returns the argument passed to `mlfReturnValue()`, which allows you to nest a call to `mlfReturnValue()` within the `return` statement.

Description

`mlfReturnValue()`, along with `mlfEnterNewContext()`, `mlfRestorePreviousContext()`, and `mlfAssign()`, implement automated memory management in the MATLAB C Math Library. Each function in the library includes calls to these functions. The functions that you write can also use automated memory management by calling these functions.

`mlfReturnValue()` is used to return a temporary `mxArray` from a function.

The arrays that are returned from MATLAB C Math Library functions are always temporary. `mlfReturnValue()` sets the state of the array passed to it to temporary but does *not* delete the array. By calling it at the end of a function, you can then nest calls to your function and not worry about assigning the `mxArray*` return from your function to a variable.

You do not need to call `mlfReturnValue()` if you are writing a function that does not return a pointer to an array.

Example

For a function defined as follows

```
mxArray *ArrayMemory(mxArray **z_out, mxArray *x_in,
                     mxArray *y_in)
```

that contains the following assignment to a local variable,

```
mlfAssign(&result_local,
          mlfSqrt(mlfPlus(mlfSin(x_in), mlfCos(x_in))));
```

use the following call to `mlfReturnValue()` to return that local variable from the function as a temporary array.

```
return mlfReturnValue(result_local);
```

See Also

`mlfEnterNewContext`, `mlfRestorePreviousContext`, `mlfAssign`

mlfScalar

Purpose Create and initialize a 1-by-1 array

C Prototype

```
mxArray *mlfScalar(double v);
```

Arguments

`double v`
Initial contents of the array

Description This function creates a 1-by-1 array whose contents are initialized to the value of v.

Example

```
mxArray *one = NULL;
mlfAssign(&one, mlfScalar(1));
```

See Also `mlfComplexScalar`

Purpose Register an error handler function with the MATLAB C Math Library

C Prototype

```
void mlfSetErrorHandler(void(* EH)(const char*, bool));
```

Arguments

`void(* EH)(const char*, bool)`
A pointer to a function that takes a `char *` argument and a Boolean argument that indicates whether the first argument is an error message or a warning. The MATLAB C Math Library calls this function rather than its default error handler when an error or warning must be displayed.

Description This function lets you to control how errors are displayed and handled.

mlfSetLibraryAllocFcns

Purpose

Set the MATLAB C Math Library's memory management functions

C Prototype

```
void mlfSetLibraryAllocFcns(calloc_proc calloc_fcn,
   free_proc free_fcn, realloc_proc realloc_fcn, malloc_proc
   malloc_fcn);
```

Arguments

`calloc_proc calloc_fcn`
The function that `mxCalloc` uses to perform memory allocation operations.

`free_proc free_fcn`
The function that `mxFree` uses to perform memory deallocation (freeing) operations.

`realloc_proc realloc_fcn`
The function that `mxRealloc` uses to perform memory reallocation operations.

`malloc_proc malloc_fcn`
The function to be called in place of `malloc` to perform memory allocation operations.

Definition

This function lets you register your own allocation and deallocation routines with the MATLAB C Math Library. It gives you complete control over memory management.

Purpose

Register a print handler with the MATLAB C Math Library

C Prototype

```
void mlfSetPrintHandler(void(* PH)(const char *));
```

Arguments

`void(* PH)(const char *)`
Pointer to a function that takes a single argument, a `const char *` (the message to be displayed), and returns `void`. This function displays the character string.

Description

Instead of calling `printf` directly, the MATLAB C Math Library calls a print handler when it needs to display an error message or warning. The default print handler used by the library takes a single argument, a `const char *` (the message to be displayed), and returns `void`.

To register your function and change which print handler the library uses, you must call the routine `mlfSetPrintHandler`. If you use an alternate print handler, you must call `mlfSetPrintHandler` before calling other library routines.

See Also

`mlfSetErrorHandler`

mlfVarargout

Purpose

Build a list of varargout output arguments from array variables.

Minimum number of input arguments: one, maximum: user-defined. Terminate the argument list with a NULL.

C Prototype

```
mlfVarargoutList *mlfVarargout(mxArray **ppa, ...);
```

Arguments

mxArray **ppa
A pointer to a pointer to an array.

Return

A cell array containing the output arguments.

Description

When the variable varargout appears as the last output argument in the definition of a MATLAB function, that function can return any number of outputs, starting at that position in the argument list.

In the MATLAB C Math Library, you use mlfVarargout() to construct the output argument list for varargout routines. The routine puts the arguments in a cell array. You must terminate the list with a NULL.

Constructs a varargout list from non-indexed arrays.

Initialize function inputs and outputs to NULL.

Varargout functions can return just 1 output value, returned as the result from the varargout function.

Example

This example uses the mlfDeal() routine to illustrate constructing a varargout list. The example creates a vector with the values [1 2 3 4 5 6 7 8 9 10] and copies this vector to each of the output arrays, A and B.

```
mxArray *A = NULL;
mxArray *B = NULL;

mlfDeal(mlfVarargout(&A,&B,NULL),mlfColon(mlfScalar(1),
                                          mlfScalar(10),NULL));
```

See Also

mlfIndexVarargout