Reference Guide

MATFOR 3.0

in C++

Contents

CHAPTER 1

Introduction

MATFOR has two main documentations namely MATFOR in C++ user's Guide, and MATFOR in C++ Reference Guide.

MATFOR in C++ user's Guide provides an overview of the MATFOR concepts such as an introduction to mfArray with focus on its constructions and syntax, a introduction to using linear algebra and a quick overview of using the Graphics procedures.

MATFOR in C++ Reference Guide provides a detailed description of the procedures available in MATFOR. The descriptions include information such as modules to use, procedure syntax, and input and output data type. Examples are included for most procedures. The Reference Guide is frequently updated. Please download the latest copy from MATFOR web support page.

This reference guide was written for users who have some background knowledge in programming with Fortran. For more information about using Fortran, please refer to your compiler's documentation.

Typographical Conventions

Before you start using this guide, it is important to understand the terms and typographical conventions used in the documentation.

The following kinds of formatting in the text identify special information.

Formatting Convention	Type of Information
Special Bold	Used to emphasize the importance of a point or a title.
`Emphasis` and `codes`	Represent variable expressions such as parameters, procedures and example codes.

Function Descriptions

Convention

The descriptions of MATFOR functions follow a fixed format. The general format is listed and described below.

Function Name

<Summary of function>

Module

<This section describes the libraries to be included in order to use the function>

Prototype

<This section describes most of the commonly used format of the function. Generally, MATFOR functions accept mfArray as input and output argument. Wherever it is convenient, other data types are supported as input argument and are presented as such in the prototype section. For example,

`void mfAxis(char* mode)` supports a string data type containing as input. You can also use an mfArray containing a string as input.>

Descriptions

<This section provides more detailed descriptions of the argument type, and application of the functions.>

Example

<This section usually presents a program code that uses the current function and the result of the program.>

See Also

<This section lists a suggestion of related functions.>

MATFOR Function Naming Conventions

MATFOR functions are all characterized by prefixing with an "*mf*". By default, MATFOR functions use the mfArray as input and output arguments. In special cases, functions may also accept C++ data types as input and output arguments.

Functions with "mf" as prefix

MATFOR functions generally adopt the following function prototypes.

```
mfArray mfFunction(mfArray x, ...);

void mfFunction(mfOutArray OutArray, mfArray
x, ...);
```

The first function prototype returns an mfArray as function output. For example, *y = mfSin(x).*

The second function prototype introduces does not return a typical function output, but introduces a new MATFOR class—the mfOutArray class. A variable declared as type mfOutArray is recognized by MATFOR internally as a function returns argument. This was designed to handle multi-output functions conveniently. Function *mfOut* is typically used to convert a list of mfArrays to the mfOutArray class. For example, the LU decomposition function mfLu returns two variables *l* and *u*, containing the lower and upper triangular matrix of the input mfArray respectively. This

is implemented as,

```
mfLu(mfOut(l, u), a);
```

The statements below list some typical MATFOR function calls.

```
mfViewPause();
mfSurf(x, y, z);
mfSubplot(2, 2, 1);
mfCos(mfOut(y), x);
mfLU(mfOut(l, u), a);
mfMeshgrid(mfOut(a, b), m, n);
```

MATFOR Parameters

The table below lists the MATFOR parameters provided for your convenience. Parameter mf() and MF_COL are mfArrays while the rest of the parameters are double precision data.

Parameter	Data Type	Descriptions
mf()	mfArray	Empty mfArray
MF_COL	mfArray	Colon ":"' operator
MF_I	complex	(0.0, 1.0)
MF_PI	double	π
MF_EPS	double	The smallest positive number
MF_INF	double	Positive infinity number.
MF_NAN	double	Not a number.
MF_E	double	Natural logarithm number.
MF_REALMAX	double	Largest representable number.
MF_REALMIN	double	Smallest representable number.

- In MATFOR, we add "mf" before functions. In such functions, every first letter of a meaningful word is in capital, for example, mfNumOfDays.
- But if there are variables, the beginning letter of the first meaningful word is small form, and the first letter of the rest meaningful words is also in capital, like "numOfDays."
- Additionally, users may read "[ ]" in brackets( ) , that means variables inside the square brackets are optional. For example, mfGet([a]). Users may write as mfGet( ), or mfGet(a).

CHAPTER 2

Essential Set of MATFOR Routines

It contains the essential set of MATFOR routines to be included in your programs. The procedures included in ESS are listed below for your reference.

mfArray constructor

`mfArray :: mfArray`	mfArray object constructor

Information

`mfArray :: IsUndefined`	Return true if mfArray is undefined.
`mfArray :: IsEmpty`	Return true if mfArray is empty.
`mfArray :: IsScalar`	Return true if mfArray is sclar.
`mfArray :: IsLogical`	Return true if mfArray is logical.
`mfArray :: IsString`	Return true if mfArray is string.
`mfArray :: IsReal`	Return true if mfArray is real.
`mfArray :: IsComplex`	Return true if mfArray is complex.
`mfArray :: Size`	Return the total number of elements in an array, or the extent of an array along a specified dimension.
`mfArray :: GetM`	Return the number of elements of first dimension in an array.
`mfArray :: GetN`	Return the number of elements of second dimension in an array.
`mfArray :: GetZ`	Return the number of elements of third dimension in an array.

`mfArray :: Shape`	Return shape of the specified mfArray.
`mfArray :: GetDims`	Return the number of dimensions or rank of a mfArray.
`mfArray :: Display`	Display mfArray data on a console window.

Operator

`mfArray :: operator ()`	Display mfArray data on a console window.
`mfArray :: operator -()`	mfArray negative
`mfArray :: operator *`	mfArray multiplication
`mfArray :: operator`	mfArray right division
`mfArray :: operator +`	mfArray addition or unary plus
`mfArray :: operator -`	mfArray subtraction or unary minus
`mfArray :: operator >`	mfArray greater than comparison
`mfArray :: operator <`	mfArray less than comparison
`mfArray :: operator >=`	mfArray less than or equal to
`mfArray :: operator <=`	mfArray less than or equal to
`mfArray :: operator ==`	mfArray equality comparison
`mfArray :: operator !=`	mfArray inequlity comparison
`mfArray :: operator =`	mfArray assign

Member Function

`mfArray :: Get`	Get elements of a mfArray.
`mfArray :: Set`	Write data to elements of a mfArray.
`mfArray :: Pow`	mfArray power
`mfArray :: T`	mfArray transpose
`mfArray :: H`	mfArray complex transpose
`mfArray :: Mul`	mfArray multiplication
`mfArray :: Ldiv`	mfArray right division

`mfArray :: Rdiv`	mfArray lift division
`mfArray :: HC`	Horizontal concatenation
`mfArray :: VC`	Vertical concatenation

Special Function

`mfArray mfAll`	Return logical true if all elements of mfArray are non-zeros
`mfArray mfAny`	Return logical true if any element of mfArray is nonzero
`mfArray mfColon`	Pick out selected rows,columns and elements of vector, matrices and mfArrays
`mfArray mfCreateComplex`	Create a complex matrix
`mfArray mfCreateFromCArray`	Convert C++ data to mfArray
`mfArray mfCreateFromFArray`	Convert Fortran data to mfArray
`mfArray mfCreateRealMatrix`	Convert Fortran data to mfArray
`mfArray mfFormat`	Specify display format of mfArray
`mfArray mfLDiv`	Matrix left divide operators
`mfArray mfRDiv`	Matrix right divide operators
`mfArray mfMul`	Return matrix product of mfArrays x and y
`mfArray mfOut`	Specify a list of mfArrays as output of a procedure
`mfArray mfTranspos`	`Return a mfArray transpose`
`mfArray mfHcat`	Horizontal concatenation
`mfArray mfVcat`	Vertical concatenation

File Io

`mfSave`	Save mfArray as binary file.
`mfLoad`	Load binary file into mfArray.

`mfSaveAscii`	Save mfArray as ASCII file.
`mfLoadAscii`	Load ASCII file into mfArray.

Parameter	
EMPTY_ARRAY	Create an empty mfArray
COL	mfColon(start,end)
C_COLON	mfColon(start,end)
MF_PI	Ratio of a circle's circumference to its diameter
MF_INF	Positive infinity
MF_NAN	Not a number
MF_EPS	Relative accuracy for floating point
MF_NINF	Negative infinity
MF_E	2.71828182845905
MF_REALMAX	Largest number of floating point
MF_REALMIN	Smallest number of floating point

Constructor

mfArray::mfArray

mfArray constructor.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray();
mfArray(const mfArray& rhs);

mfArray(const char* str);
mfArray(int value);
mfArray(double value);
mfArray(dcomplex value);
mfArray(bool value);

mfArray (int row, int col);
mfArray (int row, int col, int depth);
mfArray (int row, int col, int depth, int idx4);
mfArray (int row, int col, int depth, int idx4, int idx5);
mfArray (int row, int col, int depth, int idx4, int idx5, int idx6);
mfArray (int row, int col, int depth, int idx4, int idx5, int idx6, int
idx7);

mfArray (const double* p, int row, int col);
mfArray (const double* p, int row, int col, int depth);
mfArray (const double* p, int row, int col, int depth, int idx4);
mfArray (const double* p, int row, int col, int depth, int idx4, int idx5);
mfArray (const double* p, int row, int col, int depth, int idx4, int idx5,
int idx6);
mfArray (const double* p, int row, int col, int depth, int idx4, int idx5,
int idx6, int idx7);

mfArray (const float* p, int row, int col);
mfArray (const float* p, int row, int col, int depth);
mfArray (const float* p, int row, int col, int depth, int idx4);
mfArray (const float* p, int row, int col, int depth, int idx4, int idx5);
mfArray (const float* p, int row, int col, int depth, int idx4, int idx5,
int idx6);
mfArray (const float* p, int row, int col, int depth, int idx4, int idx5,
int idx6, int idx7);

mfArray (const dcomplex* p, int row, int col);
mfArray (const dcomplex* p, int row, int col, int depth);
mfArray (const dcomplex* p, int row, int col, int depth, int idx4);
mfArray (const dcomplex* p, int row, int col, int depth, int idx4, int
idx5);
```

```
mfArray (const dcomplex* p, int row, int col, int depth, int idx4, int
idx5, int idx6);
mfArray (const dcomplex* p, int row, int col, int depth, int idx4, int
idx5, int idx6, int idx7);

mfArray (const bool* p, int row, int col);
mfArray (const bool* p, int row, int col, int depth);
mfArray (const bool* p, int row, int col, int depth, int idx4);
mfArray (const bool* p, int row, int col, int depth, int idx4, int idx5);
mfArray (const bool* p, int row, int col, int depth, int idx4, int idx5,
int idx6);
mfArray (const bool* p, int row, int col, int depth, int idx4, int idx5,
int idx6, int idx7);
```

Descriptions

Each of these constructors initializes a new mfArray object with the specified data.

Parameters

`rhs`
An existing mfArray object to be copied into this mfArray object.

`row`
A count of the number of elements in each mfArray obiect column.

`col`
A count of the number of elements in each mfArray obiect row.

`depth`
The size of mfArray object third dimension.

`idx4`
The size of mfArray object 4th dimension.

`idx5`
The size of mfArray object 5th dimension.

`idx6`
The size of mfArray object 6th dimension.

`idx7`
The size of mfArray object 7th dimension.

`str`
A pointer to a character.

`value`
A scalar could be int, double and complex data type.

*p

A pointer to an array could be float, double or complex data type.

Example

Code

```
// ************************************************
// AnCAD example file
// file: %mfArray.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"


void main() {

    // Construct an uninitialized mfArray
    // Initialize it using the assignment operator to a scalar.
    mfArray a;
    a = 2.0;

    // Construct and initialize an mfArray using an existing double array
    double input[6]={1.0,3.0,4.0,5.0,6.0,2.0};
    mfArray b(input, 2,3);

    // Construct and initialize an mfArray using a complex variable
    dcomplex c(1.0, 2.0);
    mfArray d(c);
    mfDisplay(a, "a", b, "b", d, "d");

    // Construct an mfArray using a string
    mfArray mfg("It's a wonderful day!");
    mfDisplay(mfg, "mfg");

}
```

Result

```
a =

  2

 b =

  1  4  6
  3  5  2

 d =


 1.0000 +2.0000i


 mfg =
 It's a wonderful day!
```

See Also

Arithmetic operators

mfArray::mfComplexArray

mfArray complex constructor.

Module

`cml.h/cml.lib`

C++ Prototype

```
static mfArray mfComplexArray (int row, int col);
static mfArray mfComplexArray (int row, int col, int depth);
static mfArray mfComplexArray (int row, int col, int depth, int idx4);
static mfArray mfComplexArray (int row, int col, int depth, int idx4, int
idx5);
static mfArray mfComplexArray (int row, int col, int depth, int idx4, int
idx5, int idx6);
static mfArray mfComplexArray (int row, int col, int depth, int idx4, int
idx5, int idx6, int idx7);
```

Descriptions

Each of these constructors initializes a new complex mfArray object with given dimensions.

Parameters

`rhs`
An existing mfArray object to be copied into this mfArray object.

`row`
A count of the number of elements in each mfArray obiect column.

`col`
A count of the number of elements in each mfArray obiect row.

`depth`
The size of mfArray object third dimension.

`idx4`
The size of mfArray object 4th dimension.

`idx5`
The size of mfArray object 5th dimension.

`idx6`
The size of mfArray object 6th dimension.

`idx7`
The size of mfArray object 7th dimension.

Example

Code

```
// ************************************************
// AnCAD example file
// file: %mfComplexArray.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"


void main() {

    mfArray a = mfArray::mfComplexArray(2,2);
    dcomplex c(1.0, 2.0);

    a(mfI(1,2),mfI(1,2)) = c;
    mfDisplay(a, "a");
}
```

Result

```
a =


 1.0000 +2.0000i   1.0000 +2.0000i
 1.0000 +2.0000i   1.0000 +2.0000i
```

See Also

Arithmetic operators

Information

GetM, GetN, GetZ

Get 1st dimension, 2st dimension and 3st dimension length.

Module

`cml.h/cml.lib`

C++ Prototype

```
int mfArray ::  GetM() const;
int mfArray ::  GetN() const;
int mfArray ::  GetZ() const;
```

Descriptions

The function `GetM` gets 1st dimension length.
The function `GetN` gets 2nd dimension length.
The function `GetZ` gets 3rd dimension length.

Example

Code

```
// ************************************************
// AnCAD example file
// file: %GetM.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray a, S1;
    int S;

    //Construct a 2-by-3-by-4 mfArray variable containing ones.
    a = mfOnes(2,3,4);

   // Display the number of elements in each dimension of the mfArray.

    S = a.GetM();
    mfDisplay(S, "a.GetM()");
    S = a.GetN();
    mfDisplay(S, "a.GetN()");
    S = a.GetZ();
    mfDisplay(S, "a.GetZ()");

}
```

Result

```
a.GetM() =

  2

 a.GetN() =

  3
```

```
a.GetZ() =
 4
```

GetDims

Return the dimension of an mfArray. (Scalar, vectors and matrix returns ndims =2)

Module

`cml.h/cml.lib`

C++ Prototype

```
void mfArray :: GetDims(int dims[7]) const;
```

Descriptions

The function `GetDims` returns the dimension of an mfArray.

Example

Code

```
// ************************************************
// AnCAD example file
// file: %GetDims.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include <iostream.h>

void main() {

    mfArray a, b;
    int i, dims[7];


    // Construct a 2-by-3-by-4 mfArray variable containing ones.
    a = mfOnes(2,3,4);

    // Display dimensions of the mfArray.
    a.GetDims(dims);

    cout << "dims = " << endl;
    for(i=0;i<=6;++i)
    {
        cout << dims[i] << " ";
    }
    cout << endl;

}
```

Result

```
dims =
2 3 4 1 1 1 1
```

Operator

Arithmetic Operators

Arithmetic operators contain mfArray object operators and member functions.

Include

```
cml.h/cml.lib
```

Prototype

mfArray :: operator ()

```
mfArray operator()(mfIndexArray r) const;
mfArray operator()(mfIndexArray r, mfIndexArray c) const;
                   mfArray operator()(mfIndexArray r,
                   mfIndexArray c, mfIndexArray z) const;
mfArray operator()(mfIndexArray r, mfIndexArray c, mfIndexArray
                   z, mfIndexArray idx4) const;
mfArray operator()(mfIndexArray r, mfIndexArray c, mfIndexArray
                   z, mfIndexArray idx4, mfIndexArray idx5)
                   const;
mfArray operator()(mfIndexArray r, mfIndexArray c, mfIndexArray
                   z, mfIndexArray idx4, mfIndexArray idx5,
                   mfIndexArray idx6) const;
mfArray operator()(mfIndexArray r, mfIndexArray c, mfIndexArray
                   z, mfIndexArray idx4, mfIndexArray idx5,
                   mfIndexArray idx6, mfIndexArray idx7) const;
```

mfArray :: operator =

```
mfArray& operator =(const mfArray& rhs);
```

mfArray :: operator +

```
mfArray operator +(const mfArray& rhs) const;
```

mfArray :: operator -

```
mfArray operator -(const mfArray& rhs) const;
```

mfArray :: operator *

```
mfArray operator *(const mfArray& rhs) const;
```

mfArray :: operator /

```
mfArray operator /(const mfArray& rhs) const;
```

mfArray :: - ()

```
mfArray operator - () const;
```

mfArray :: +=

```
mfArray& operator +=(const mfArray& rhs);
```

mfArray :: -=

```
mfArray& operator -=(const mfArray& rhs);
```

mfArray :: *=

```
mfArray& operator *=(const mfArray& rhs);
```

mfArray :: /=

```
mfArray& operator /=(const mfArray& rhs);
```

mfArray :: !

```
mfArray operator ! () const;
```

mfArray :: |

```
mfArray operator | (const mfArray& rhs) const;
```

mfArray :: &

```
mfArray operator & (const mfArray& rhs) const;
```

mfArray :: ^

```
mfArray operator ^ (const mfArray& rhs) const;
```

mfArray :: Pow

```
mfArray Pow(const mfArray& rhs) const;
```

mfArray :: T

```
mfArray T() const;
```

mfArray :: H

```
mfArray H() const;
```

mfArray :: Mul

```
mfArray Mul(const mfArray& rhs) const;
```

mfArray :: LDiv

```
mfArray LDiv(const mfArray& rhs) const;
```

mfArray :: RDiv

```
mfArray RDiv(const mfArray& rhs) const;
```

mfArray :: HC

```
mfArray HC(const mfArray& rhs) const;
```

mfArray :: VC

```
mfArray VC(const mfArray& rhs) const;
```

Descriptions

mfArray object has the following operator and member functions:

Arithmetic operators provided by MATFOR include both array and matrix operators. Array operators operate element-by-element on the mfArrays while matrix operators operate on whole mfArrays. The following lists the operators available with a brief description of their operation.

Operator	Descriptions
=	Assign `y = x` lets mfArray `y` equal mfArray `x`
+	Array addition or unary plus `x + y` adds mfArrays `x` and `y`. The two mfArrays must have the same size unless one is a scalar. A scalar can be added to mfArrays of any size.
-	Array subtraction or unary minus `x – y` subtracts mfArray `y` from `x`. The two mfArrays must have the same size unless one is a scalar. A scalar can be subtracted from mfArrays of any size.
*	Array multiplication `x * y` returns element-by-element the multiplication of mfArrays `x` and `y`. The two mfArrays must be in the same size unless one is a scalar. A scalar can be multiplied to a mfArray of any size.
/	Array division

`x / y` returns element-by-element the division of mfArrays `x` into `y`. The two mfArrays must be in the same size unless one is a scalar. A scalar can be division to an mfArray of any size.

-()

Array Negative

`-x` returns negative mfArray `x`

+=

Array addition and assignment

`x += y` adds `y` to `x` and store the result in `x`.

-=

Array subtraction and assignment

`x -= y` subtracts `y` from `x` and store the result in `x`.

*=

Array multiplication and assignment

`x *= y` multiplies `x` by `y` and store the result in `x`.

/=

Array division and assignment

`x /= y` divides `x` by `y` and store the result in `x`.

!

Bitwise not

`!x` performs the and operation on `x` and returns a logical mfArray. `x` must also be a logical mfArray.

|

Bitwise or

`x | y` performs the or operation on `x` and `y` and returns a logical mfArray. `x` and `y` must also be logical mfArrays.

&

Bitwise and

`x & y` performs the and operation on `x` and `y` and returns a logical mfArray. `x` and `y` must also be logical mfArrays.

^ Bitwise exclusive or

`x ^ y` performs the exclusive or operation on `x` and `y` and returns a logical mfArray. `x` and `y` must also be logical mfArrays.

() Parentheses are used to indicate precedence in arithmetic expressions

`(x+y)*z` returns the multiplication of mfArray z and the addition of mfArray `x` and `y`.

mfArray::Pow Power function

`z =y.Pow(x) returns` a mfArray `z` with elements computed from x raised to the power of each element of mfArray `y`, i.e., `z = y**x`.

mfArray::T Array transpose

`y = x.T()` returns a mfArray y containing the transpose of mfArray `x`. In a transpose operation, `aij` and `aji` are interchanged.

Complex elements are not conjugated.

You can use the `mfTranspose` function to perform the same operation.

```
y = x.T()
y = mfTranspose(x)
```

mfArray::H Complex conjugate transpose

`y = x.H()` returns a mfArray y containing the complex conjugate transpose of mfArray `x`. If `x` is real, this operation returns the same result as `x.T()`.

When performing linear algebra operations on complex matrices, it is almost always the complex conjugate

transpose that is needed.

mfArray::Mul

Matrix multiplication

`z = x.Mul(y)` returns matrix product of mfArrays x and y, where x is a m-by-p matrix and y is a p-by-n matrix. The product of the matrix multiplication is a m- by-n mfArray.

mfArray::LDiv

Matrix left division

`z = x.LDiv(y)` returns a mfArray `z` approximately equal to `mfInv(x)*y`.

mfArray::RDiv

Matrix right division

`z = x.RDiv(y)` returns a mfArray `z` approximately equal to `x*mfInv(y)`.

mfArray::HC

Horizontal concatenation

`z = x.HCat(y)` concatenates mfArray `x` and `y` horizontally. The two mfArrays must have the same number of rows. There is no limit on the number of mfArrays you concatenate in the same assignment.

mfArray::VC

Vertical concatenation

`z = x.VCat(y)` concatenates mfArray `x` and `y` vertically. The two mfArrays must have the same number of columns. There is no limit on the number of mfArrays you concatenate at the same assignment.

Relational Operators

Relational operators return true or false in comparing left hand side value with right hand side value.

Include

`cml.h/cml.lib`

Prototype

mfArray :: operator >

```
mfArray operator > (const mfArray& rhs) const;
```

mfArray :: operator >=

```
mfArray operator >=(const mfArray& rhs) const;
```

mfArray :: operator ==

```
mfArray operator ==(const mfArray& rhs) const;
```

mfArray :: operator !=

```
mfArray operator !=(const mfArray& rhs) const;
```

mfArray :: operator <

```
mfArray operator < (const mfArray& rhs) const;
```

mfArray :: operator <=

```
mfArray operator <=(const mfArray& rhs) const;
```

Descriptions

The relational operators ==, >=, >, <=, and != perform element-by-element comparisons between two mfArrays.

The returned logical mfArray records the status of the comparison. Each element records a logical true (1) if the relationship is true, and logical false (0) otherwise.

Note that the two mfArrays x and y must be in the same size, unless one is a scalar. The scalar will be expanded to a mfArray the same size as the other mfArray.

Operator	Descriptions
==	mfArray equality comparision x == y checks if each element of x equals the corresponding element of y. The operator tests in both real and imaginary parts of the elements. When you use the equality (==) operator in an if statement, use the all function to enclose it. The all function ensures that the equality condition is satisfied by all elements of the returned mfArray.
>=	mfArray greater than or equal to comparison x >= y checks if each element of x is more than or equal to the corresponding element of y. The operator compares only the real part of the elements.
>	mfArray greater than comparison x > y checks if each element of x is greater than the corresponding element of y. The operator compares only the real part of the elements.
<=	mfArray less than or equal to comparison

`x <= y` checks if each element of `x` is less than or equal to the corresponding element of `y`. The function returns 1 if the relation is true and 0 otherwise. The operator compares only the real part of the elements.

`<` mfArray less than comparison

`x < y` checks if each element of `x` is less than the corresponding element of `y`. The function returns 1 if the relation is true and 0 otherwise. The operator compares only the real part of the elements.

`!=` mfArray inequality comparision

`x != y` checks if each element of `x` for inequality to the corresponding element of `y`. The operator tests in both real and imaginary parts of the elements.

Operator Precedence

The table below lists the MATFOR operator precedence. Operators sharing the same symbol as Fortran operators, share also the same precedence level. All MATFOR-defined operators occupy the lowest precedence level.

Note that operators having the same precedence level are evaluated from left to right.

Member function	Descriptions
`mfArray :: Pow`	mfArray power
`mfArray :: T`	mfArray transpose
`mfArray :: H`	mfArray complex transpose
`mfArray :: Mul`	mfArray multiplication
`mfArray :: Ldiv`	mfArray right division
`mfArray :: Rdiv`	mfArray lift division
`mfArray :: HC`	Horizontal concatenation
`mfArray :: VC`	Vertical concatenation

Operator	Descriptions	Precedence
`mfArray :: operator -()`	mfArray negative	Highest
`mfArray :: operator *`	mfArray multiplication	
`mfArray :: operator /`	mfArray right division	
`mfArray :: operator +`	mfArray addition or unary plus	
`mfArray :: operator -`	mfArray subtraction or unary minus	

`mfArray :: operator >`	mfArray greater than comparison	
`mfArray :: operator <`	mfArray less than comparison	
`mfArray :: operator >=`	mfArray greater than or equal to	
`mfArray :: operator <=`	mfArray less than or equal to	
`mfArray :: operator ==`	mfArray equality comparison	
`mfArray :: operator !=`	mfArray inequlity comparison	
`mfArray :: &operator =`	mfArray assign	Lowest

Member Function

ToDouble

Convert mfArray to double.

Module

`cml.h/cml.lib`

C++ Prototype

```
double mfArray :: ToDouble(mfArray x) const;
```

Descriptions

The member function `ToDouble` converts an mfArray to a double.

See Also

%ToString

ToString

Convert mfArray to string.

Module

`cml.h/cml.lib`

C++ Prototype

```
char* mfArray :: ToString(mfArray x) const;
```

Descriptions

The member function `ToString` converts an mfArray to a string.

See Also

%ToDouble

mfArray Access

mfMatSub, mfS

Retrieve rows and columns of elements from vector or matrix mfArray.

Module

`ess.h/ess.lib`

C++ Prototype

```
mfArray mfMatSub(mfArray& p, const mfIndexArray& idx1);
mfArray mfMatSub(mfArray& p, const mfIndexArray& idx1, const
mfIndexArray& idx2);
mfArray mfMatSub(mfArray& p, const mfIndexArray& idx1, const
mfIndexArray& idx2, const mfIndexArray& idx3);
mfArray mfMatSub(mfArray& p, const mfIndexArray& idx1, const
mfIndexArray& idx2, const mfIndexArray& idx3, const mfIndexArray& idx4);
mfArray mfMatSub(mfArray& p, const mfIndexArray& idx1, const
mfIndexArray& idx2, const mfIndexArray& idx3, const mfIndexArray& idx4,
const mfIndexArray& idx5);
mfArray mfMatSub(mfArray& p, const mfIndexArray& idx1, const
mfIndexArray& idx2, const mfIndexArray& idx3, const mfIndexArray& idx4,
const mfIndexArray& idx5, const mfIndexArray& idx6);
mfArray mfMatSub(mfArray& p, const mfIndexArray& idx1, const
mfIndexArray& idx2, const mfIndexArray& idx3, const mfIndexArray& idx4,
const mfIndexArray& idx5, const mfIndexArray& idx6, const mfIndexArray&
idx7);
```

Descriptions

Function `mfMatSub` retrieves elements from an mfArray. The arguments (except the first one) specify the subscripts of the elements

Notice that MATFOR also provides the abbreviated name `mfS` for this Function.

For example, ifA is a vector mfArray, `mfMatSub(A, "1:100:2")` would return a vector containing the 1st, 3rd, 5th, ..., and 99th elements of A.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfMatSub.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fml.h"
```

```
#include "fgl.h"

void main() {

    mfArray x, y, z;
    int i;

    x = mfReshape(mfColon(1, 24), 2, 3, 4);
    mfDisplay(x, "x");

    // Get x <= 12
    y = x(x <= 12);
    mfDisplay(y, "x(x <= 12)");

    // y = x(1:24) using 1x24 replace 2x3x4
    y = x(mfI(1,24));
    mfDisplay(y, "x(mfI(1,24))");

    // y = x(1:2, 1:12) using 2x12 replace 2x3x4
    y = x(mfI(1,2), mfI(1,12));
    mfDisplay(y, "x(mfI(1,2), mfI(1,12))");

    // y = x(1:2, 1:12:3)
    y = x(mfI(1,2), mfI(1,12,3));
    mfDisplay(y, "x(mfI(1,2), mfI(1,12,3))");

    // y = x(24:1)
    y = mfS(x,MF_COL);
    mfDisplay(y, "x(MF_COL)");

    // y = x(1, 2, 3)
    y = x(1, 2, 3);
    mfDisplay(y, "x(1, 2, 3)");

    // Get element using arbitrary index
    y = x(mfV(2, 1), mfV(3, 1), mfV(4, 1, 3));
    mfDisplay(y, "x(mfV(2, 1), mfV(3, 1), mfV(4, 1, 3))");

    // Set element in y which <= 12 and >= 6 to 0
    y = x;
    y( (y <= 12) & (y >= 6) ) = 0;
    mfDisplay(y, "y((y <= 12) & (y >= 6)) = 0");

    // y(1:24) = 0 using 1x24 replace 2x3x4
    y = x;
    y(mfI(1,24)) = 0;
    mfDisplay(y, "y(mfI(1,24)) = 0");

    // y(1:2, 1:12) = 0 using 1x24 replace 2x3x4
    y = x;
    y(mfI(1,2), mfI(1,12)) = 0;
    mfDisplay(y, "y(mfI(1,2), mfI(1,12)), = 0");

    // y(1:2, 1:12:3) = 0 using 1x24 replace 2x3x4
    y = x;
    y(mfI(1, 2), mfI(1,12,3)) = 0;
    mfDisplay(y, "y9mfI(1, 2), mfI(1,12,3)) = 0");

    // y(1:2, 1:3:2, 1:4:3) = reshape((/(i, i = 31, 38)/), (/2, 2, 2/))
    y = x;
    z = mfReshape(mfColon(31, 38), 2, 2, 2);
    y(mfI(1,2), mfI(1,3,2), mfI(1,4,3)) = z;
    mfDisplay(y, "y(mfI(1,2), mfI(1,3,2), mfI(1,4,3)) = z");


    // y(24:1) = .t.(/(i, i= 25, 48)/)
    y = x;
    y(MF_COL) =  mfColon(25, 48).T();
    mfDisplay(y, "y(MF_COL) =  mfColon(25, 48).T()");

    // y(1, 2, 3) = 60
    y = x;
```

```
    y(1, 2, 3) = 60;
    mfDisplay(y, "y(1, 2, 3) = 60");

    // Set element using arbitrary index

    y = x;
    z = mfReshape(mfColon(101, 112), 2, 2, 3);
    y(mfV(2,1), mfV(3,1), mfV(4,1,3)) = z;
    mfDisplay(y, "y(mfI(2,1), mfI(3,1), mfI(4,1,3)) = z");
}
```

Result

```
x(:,:,1) =

  1  3  5
  2  4  6

 x(:,:,2) =

   7   9  11
   8  10  12

 x(:,:,3) =

  13  15  17
  14  16  18

 x(:,:,4) =

  19  21  23
  20  22  24

 x(x <= 12) =

   1
   2
   3
   4
   5
   6
   7
   8
   9
  10
  11
  12

 x(mfI(1,24)) =

column  1  to  19 ...
   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19

column  20  to  24
  20  21  22  23  24

 x(mfI(1,2), mfI(1,12)) =

   1   3   5   7   9  11  13  15  17  19  21  23
   2   4   6   8  10  12  14  16  18  20  22  24

 x(mfI(1,2), mfI(1,12,3)) =

   1   7  13  19
   2   8  14  20

 x(MF_COL) =

   1
   2
   3
   4
   5
```

```
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24

x(1, 2, 3) =

 15

x(mfV(2, 1), mfV(3, 1), mfV(4, 1, 3))(:,:,1) =

 24  20
 23  19

x(mfV(2, 1), mfV(3, 1), mfV(4, 1, 3))(:,:,2) =

 6  2
 5  1

x(mfV(2, 1), mfV(3, 1), mfV(4, 1, 3))(:,:,3) =

 18  14
 17  13

y((y <= 12) & (y >= 6)) = 0(:,:,1) =

 1  3  5
 2  4  0

y((y <= 12) & (y >= 6)) = 0(:,:,2) =

 0  0  0
 0  0  0

y((y <= 12) & (y >= 6)) = 0(:,:,3) =

 13  15  17
 14  16  18

y((y <= 12) & (y >= 6)) = 0(:,:,4) =

 19  21  23
 20  22  24

y(mfI(1,24)) = 0(:,:,1) =

 0  3  5
 2  4  6

y(mfI(1,24)) = 0(:,:,2) =

  7   9  11
  8  10  12

y(mfI(1,24)) = 0(:,:,3) =

 13  15  17
 14  16  18
```

```
y(mfI(1,24)) = 0(:,:,4) =

 19  21  23
 20  22  24

y(mfI(1,2), mfI(1,12)), = 0(:,:,1) =

 0  0  0
 0  0  0

y(mfI(1,2), mfI(1,12)), = 0(:,:,2) =

 0  0  0
 0  0  0

y(mfI(1,2), mfI(1,12)), = 0(:,:,3) =

 0  0  0
 0  0  0

y(mfI(1,2), mfI(1,12)), = 0(:,:,4) =

 0  0  0
 0  0  0

y9mfI(1, 2), mfI(1,12,3)) = 0(:,:,1) =

 0  3  5
 0  4  6

y9mfI(1, 2), mfI(1,12,3)) = 0(:,:,2) =

  0   9  11
  0  10  12

y9mfI(1, 2), mfI(1,12,3)) = 0(:,:,3) =

  0  15  17
  0  16  18

y9mfI(1, 2), mfI(1,12,3)) = 0(:,:,4) =

  0  21  23
  0  22  24

y(mfI(1,2), mfI(1,3,2), mfI(1,4,3)) = z(:,:,1) =

 31   3  33
 32   4  34

y(mfI(1,2), mfI(1,3,2), mfI(1,4,3)) = z(:,:,2) =

  7   9  11
  8  10  12

y(mfI(1,2), mfI(1,3,2), mfI(1,4,3)) = z(:,:,3) =

 13  15  17
 14  16  18

y(mfI(1,2), mfI(1,3,2), mfI(1,4,3)) = z(:,:,4) =

 35  21  37
 36  22  38

y(MF_COL) =  mfColon(25, 48).T()(:,:,1) =

 25  27  29
 26  28  30

y(MF_COL) =  mfColon(25, 48).T()(:,:,2) =

 31  33  35
 32  34  36
```

```
y(MF_COL) =  mfColon(25, 48).T()(:,:,3) =

 37  39  41
 38  40  42

y(MF_COL) =  mfColon(25, 48).T()(:,:,4) =

 43  45  47
 44  46  48

y(1, 2, 3) = 60(:,:,1) =

 1  3  5
 2  4  6

y(1, 2, 3) = 60(:,:,2) =

  7   9  11
  8  10  12

y(1, 2, 3) = 60(:,:,3) =

 13  60  17
 14  16  18

y(1, 2, 3) = 60(:,:,4) =

 19  21  23
 20  22  24

y(mfI(2,1), mfI(3,1), mfI(4,1,3)) = z(:,:,1) =

 108   3  106
 107   4  105

y(mfI(2,1), mfI(3,1), mfI(4,1,3)) = z(:,:,2) =

  7   9  11
  8  10  12

y(mfI(2,1), mfI(3,1), mfI(4,1,3)) = z(:,:,3) =

 112  15  110
 111  16  109

y(mfI(2,1), mfI(3,1), mfI(4,1,3)) = z(:,:,4) =

 104  21  102
 103  22  101
```

See Also

Special Function

mfAll

Return logical true if all elements of mfArray are non-zeros.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfAll(const mfArray &op1[, const mfArray &dim]);
```

Descriptions

Function `mfAll` determines if all elements of an mfArray are non-zeros.

`l = mfAll(x)`

- `l` is a logical mfArray, containing logical element(s).
- If `x` is a vector, `mfAll` returns a scalar containing logical 1 if all elements of `x` are non-zeros, and logical 0 otherwise.
- If `x` is a matrix, `mfAll` is applied to each column of `x`, returning a row vector containing 0's or 1's.

`l = mfAll(x, IDIM)` operates on the dimension of `x` specified by `IDIM`.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfAll.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y, z;
    bool l;

    x = mfMagic(3)  > 2;

    // Check if All elements of x are greater than 0.
    l = mfAll(x);

    y = mfAll(x, 1);

    z = mfAll(x, 2);

    mfDisplay(x, "x", mf(l), "mfAll(x)", y, "mfAll(x, 1)",  z, "mfAll(x,
2)");
}
```

Result

```
x =

  1  0  1
  1  1  1
  1  1  0

 mfAll(x) =

  3

 mfAll(x, 1) =

  1  0  0

 mfAll(x, 2) =

  0
  1
  0
```

See Also

mfAny

mfAny

Return logical true if any element of mfArray is nonzero.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfAny(const mfArray &op1[, const mfArray &dim]);
```

Descriptions

Function `mfAny` determines if any element of an mfArray is nonzero.

`l = mfAny(x)`

- `l` is a logical mfArray, containing logical elements.
- If `x` is a vector, `mfAny` returns a scalar containing logical 1 if any element of `x` is nonzero, and logical 0 otherwise.
- If `x` is a matrix, `mfAny` is applied to each column of `x`, returning a row vector containing 0's and 1's.

`l = mfAny(x, IDIM)` operates on the dimension of `x` specified by `IDIM`.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfAny.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;
    bool l;

    x = mfMagic(3);

    // Check if all elements of x are more than 5.
    l = mfAny(x>5);
    mfDisplay(x, "x", l, "mfAny(x>5)");

    y = mfAny(x>5, 1);
    mfDisplay(y, "mfAny(x>5, 1)");
    y = mfAny(x>5, 2);
    mfDisplay(y, "mfAny(x>5, 2)");
}
```

Result

```
x =

  8  1  6
  3  5  7
  4  9  2

 mfAny(x>5) =

  5

 mfAny(x>5, 1) =

  1  1  1

 mfAny(x>5, 2) =

  1
  1
  1
```

See Also

mfAll

mfColon

Pick out selected rows, columns and elements of vector, matrices and mfArrays.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfColon(const mfArray &start, const mfArray &end);
mfArray mfColon(const mfArray &start, const mfArray &step,  const mfArray &end);
```

Descriptions

Function `mfColon` constructs a regularly spaced vector mfArray. The input arguments, `start`, `step` and `end` can be integers or real. For complex inputs, imaginary parts are ignored.

`x = mfColon(a, b, c)`

- Vector mfArray `x` is constructed with elements [a, a+b, ..., a+b*m, ...,c] where `m = mfFix((c-a)/b`.
- The function returns empty when `b>0, a>c`, or when `b<0, a<c`.

`x = mfColon(a, c)`

- Vector mfArray `x` is constructed with elements [a, a+1, ...,c].
- It returns empty if `a>c`.

Example

The following example constructs an mfArray x by using mfColon.

Code

```
// ************************************************
// AnCAD example file
// file: mfColon.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x;

    x = mfColon(1.0, 0.5, 2.0);
    mfDisplay (x, "x");

}
```

Result

```
x =

   1.0000   1.5000   2.0000
```

mfCreateVector, mfV

Create a vector mfArray.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfCreateVector(double v1, double v2[, double v3, ..., double
v32]);
mfArray mfCreateVector(dcomplex v1, dcomplex v2[, dcomplex v3, ...,
dcomplex v32]);
```

Descriptions

The function creates a vector mfArray.

`y = mfCreateVector(v1, v2, ...);`
Returns a vector mfArray `y` containing the data specified in arguments `v1`, `v2`, ...

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfCreateVector.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x;
    double a[5];
    int i;

    for(i=0;i<=4;++i)
    {
        a[i] = (i+1) * 0.1;
    }

    // Create Vector mfArray
    x = mfV(a[0], a[1], a[2], a[3], a[4]);
    mfDisplay(x, "x");

}
```

Result

```
x =

   0.1000   0.2000   0.3000   0.4000   0.5000
```

See Also

mfCreateMatrix

mfCreateMatrix, mfM

Create a matrix mfArray.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfCreateMatrix(int row, int col, double v1[, double v2, ...,
double v128])
mfArray mfCreateMatrix(int row, int col, dcomplex v1[, dcomplex v2, ...,
dcomplex v128])
```

Descriptions

The function creates a matrix mfArray.

`y = mfCreateMatrix(row, col, v1, v2, ...);`
Returns matrix mfArray `y` whose shape is specified by arguments `row` and `col`. The data`v1`, `v2`, ... are stored column-wise in the elements of the created mfArray.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfCreateMatrix.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x;
    double a[5], b[5], c[5];
    int i;

    for(i=0;i<=4;++i)
    {
        a[i] = i * 0.1;
        b[i] = i * (-0.1);
        c[i] = a[i] * b[i];
    }

    // Create matrix mfArray
    x = mfM(5, 3, a[0], a[1], a[2], a[3], a[4],
                  b[0], b[1], b[2], b[3], b[4],
                  c[0], c[1], c[2], c[3], c[4]);
    mfDisplay(x, "x");

}
```

Result

```
x =

   0.0000   0.0000   0.0000
   0.1000  -0.1000  -0.0100
   0.2000  -0.2000  -0.0400
   0.3000  -0.3000  -0.0900
   0.4000  -0.4000  -0.1600
```

See Also

mfCreateMatrix

mfCreateFromCArray

Convert row major C array into mfArray(column major).

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfCreateFromCArray (const double* p, int row, int col);
mfArray mfCreateFromCArray (const double* p, int row, int col, int depth);
mfArray mfCreateFromCArray (const double* p, int row, int col, int depth,
int idx4);
mfArray mfCreateFromCArray (const double* p, int row, int col, int depth,
int idx4, int idx5);
mfArray mfCreateFromCArray (const double* p, int row, int col, int depth,
int idx4, int idx5, int idx6);
mfArray mfCreateFromCArray (const double* p, int row, int col, int depth,
int idx4, int idx5, int idx6, int idx7);
mfArray mfCreateFromCArray (const float* p, int row, int col);
mfArray mfCreateFromCArray (const float* p, int row, int col, int depth);
mfArray mfCreateFromCArray (const float* p, int row, int col, int depth,
int idx4);
mfArray mfCreateFromCArray (const float* p, int row, int col, int depth,
int idx4, int idx5);
mfArray mfCreateFromCArray (const float* p, int row, int col, int depth,
int idx4, int idx5, int idx6);
mfArray mfCreateFromCArray (const float* p, int row, int col, int depth,
int idx4, int idx5, int idx6, int idx7);
mfArray mfCreateFromCArray (const dcomplex* p, int row, int col);
mfArray mfCreateFromCArray (const dcomplex* p, int row, int col, int
depth);
mfArray mfCreateFromCArray (const dcomplex* p, int row, int col, int depth,
int idx4);
mfArray mfCreateFromCArray (const dcomplex* p, int row, int col, int depth,
int idx4, int idx5);
mfArray mfCreateFromCArray (const dcomplex* p, int row, int col, int depth,
int idx4, int idx5, int idx6);
mfArray mfCreateFromCArray (const dcomplex* p, int row, int col, int depth,
int idx4, int idx5, int idx6, int idx7);
```

Descriptions

Function `mfCreateFromCArray` creates a new mfArray from C order array (row major).

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfCreateFromCArray.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x;
    double a[6]={1.1, 2.2, 3.3, 4.4, 5.5, 6.6};

    // creates a new mfArray from C order array
    x = mfCreateFromCArray(a, 2, 3);
    mfDisplay(x, "x");

}
```

Result

```
x =


   1.1000   2.2000   3.3000
   4.4000   5.5000   6.6000
```

See Also

mfCreateFromFArray

mfCreateFromFArray

Assign Fortran array to mfArray.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfCreateFromFArray (const double* p, int row, int col);
mfArray mfCreateFromFArray (const double* p, int row, int col, int depth);
mfArray mfCreateFromFArray (const double* p, int row, int col, int depth,
int idx4);
mfArray mfCreateFromFArray (const double* p, int row, int col, int depth,
int idx4, int idx5);
mfArray mfCreateFromFArray (const double* p, int row, int col, int depth,
int idx4, int idx5, int idx6);
mfArray mfCreateFromFArray (const double* p, int row, int col, int depth,
int idx4, int idx5, int idx6, int idx7);
mfArray mfCreateFromFArray (const float* p, int row, int col);
mfArray mfCreateFromFArray (const float* p, int row, int col, int depth);
mfArray mfCreateFromFArray (const float* p, int row, int col, int depth,
int idx4);
mfArray mfCreateFromFArray (const float* p, int row, int col, int depth,
int idx4, int idx5);
mfArray mfCreateFromFArray (const float* p, int row, int col, int depth,
int idx4, int idx5, int idx6);
mfArray mfCreateFromFArray (const float* p, int row, int col, int depth,
int idx4, int idx5, int idx6, int idx7);
mfArray mfCreateFromFArray (const dcomplex* p, int row, int col);
mfArray mfCreateFromFArray (const dcomplex* p, int row, int col, int
depth);
mfArray mfCreateFromFArray (const dcomplex* p, int row, int col, int depth,
int idx4);
mfArray mfCreateFromFArray (const dcomplex* p, int row, int col, int depth,
int idx4, int idx5);
mfArray mfCreateFromFArray (const dcomplex* p, int row, int col, int depth,
int idx4, int idx5, int idx6);
mfArray mfCreateFromFArray (const dcomplex* p, int row, int col, int depth,
int idx4, int idx5, int idx6, int idx7);
```

Descriptions

Function `mfCreateFromFArray` creates a new mfArray from Fortran order array (column major).

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfCreateFromFArray.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x;
    double a[6]={1.1, 2.2, 3.3, 4.4, 5.5, 6.6};


    // creates a new mfArray from Fortran order array
    x = mfCreateFromFArray(a, 2, 3);
    mfDisplay(x, "x");

}
```

Result

```
x =


   1.1000   3.3000   5.5000
   2.2000   4.4000   6.6000
```

See Also

mfCreateFromCArray

mfDisplay

Display mfArray data.

Module

`cml.h/cml.lib`

C++ Prototype

```
void mfDisplay(const mfArray& v1, const mfArray& t1="ans");
void mfDisplay(const mfArray& v1, const mfArray& t1[, const mfArray& v2,
const mfArray& t2, ..., const mfArray& v16, const mfArray& t16]);
```

Descriptions

Function `mfDisplay` shows the content of mfArrays on a MS-DOS console window.

`mfDisplay(x)` displays the content of mfArray `x`, with a caption 'ans ='.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfDisplay.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main(){

    double input[6]={1.0,2.0,3.0,4.0,5.0,6.0};
    mfArray b(input, 2,3);

    mfDisplay(b, "b");
}
```

Result

```
b =

  1  3  5
  2  4  6
```

See Also

mfFormat

mfFormat

Specify displaying format of mfArray.

Module

`cml.h/cml.lib`

C++ Prototype

```
void mfFormat(char *str);
```

Descriptions

Function `mfFormat` specifies display format of mfArray.

`mfFormat("long")`
Return 8 - byte precision with function `mfDisplay`

`mfFormat("shot")`
Return 4 - byte precision with function `mfDisplay`.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfFormat.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;

    x=2.02;
    y=mfCos(x);

    mfFormat("short");
   mfDisplay(y, "y (short format)");

   mfFormat("long");
   mfDisplay(y, "y (long format)");

}
```

Result

```
y (short format) =

  -0.4342

 y (long format) =
```

```
-0.434248346118300
```

See Also

mfDisplay

mfRDiv, mfLDiv

Matrix left divide and right divide operators.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfRDiv(const mfArray &a, mfArray &b);
mfArray mfLDiv(const mfArray &a, mfArray &b);
```

Descriptions

Function `mfLDiv` and `mfRDiv` are normally used in solving systems of linear equations represented by ax=b.

`x = mfLDiv(a, b)` is an approximation to `mfMul(mfInv(a), b))`.
`x = mfRDiv(b, a)` is an approximation to `mfMul(b, mfInv(a)`.

The two functions are related by

```
mfLdiv(a,                              b)                              =
mfTranspose(mfRDiv((mfTranspose(b)),(mfTranspose(a))))
```

Depending on the structure of the coefficient matrix `a`, MATFOR uses different algorithms to solve the simultaneous linear equations `mfLDiv(a, b)` and `mfRDiv(b, a)`. Figure 2.2 provides an overview of the different methods used for solving the linear equation, depending on the structure of matrix `a`.

Figure 2.2 Algorithms applicable for each type of matrix `a`.

If `a` is an n x n square matrix, and `b` is a n-by-p matrix, then `mfLDiv(a, b)` is solved by using Gaussian elimination. MATFOR performs a structural test on matrix `a` to select the optimum factorization method. Non-symmetry and non-positive definite systems are detected almost immediately, hence this does not take much of the computation time.

If `a` is an m-by-n rectangle matrix, and `b` is an m-by-p matrix, for m /= n, MATFOR uses the least square method for solving the under-determined or over-determined system. There are two approaches to solving a least square problem - QR and normal equations method. MATFOR uses the normal equations method as it requires half the arithmetic when m<n and much lesser storage space compared to the QR method.

Note that MATFOR uses LAPACK for solving Linear Algebra equations.

Example

Code

```
// ************************************************
```

```
// AnCAD example file
// file: mfRDiv.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y, z;
    double temp[9] = { 5.0, 1.0, 2.0, 2.0, 10.0, 3.0 , 3.0, 2.0, 5.0 };

    // Construct a 3-by-3 mfArray x and 3-by-1 mfArray y.
    x = mfArray(temp,3,3);
    y = mfOnes(3, 1);

    z = mfLDiv(x, y);

   // Display value of x, y
    mfDisplay(x, "x", y, "y", z, "z");
}
```

Result

```
x =

   5   2   3
   1  10   2
   2   3   5

 y =

  1
  1
  1

 z =


   0.1018
   0.0659
   0.1198
```

See Also

Arithmetic Operators

mfMul

Return matrix product of two mfArrays.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfMul(const mfArray &x, mfArray &y);
```

Descriptions

Function `mfMul(x, y)` returns matrix product of mfArrays `x` and `y`, where `x` is a m-by-p matrix and `y` is a p-by-n matrix. The product of the matrix multiplication is an m-by-n mfArray.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfMul.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y, m;

    x = mfMagic(3);
    y = mfOnes(3,2);

    m = mfMul(x,y);

    mfDisplay(x, "x", y, "y", m, "m");
}
```

Result

```
x =

  8  1  6
  3  5  7
  4  9  2

 y =

  1  1
  1  1
  1  1

 m =

  15  15
  15  15
  15  15
```

See Also

Arithmetic Operators

mfOut

Specify a list of mfArrays as output of a function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfOutArray mfOut();
mfOutArray mfOut( const mfArray& x1[, const mfArray& x2, ..., const
mfArray& x18] );
```

Descriptions

Function `mfOut` specifies a list of mfArrays as output of a subroutine. It differentiates the input arguments from the output arguments in a function call. Note that function `mfOut` encloses only mfArrays.

For example, you can use function `mfFind` to retrieve the row, column indices, and non-zero values of an mfArray as shown below:

`mfFind(mfOut(i, j, v), x)`

The function returns three mfArrays in this usage. To get the three mfArrays `i`, `j`, and `v`, you must enclose them with `mfOut` in your function call. This automatically triggers function `mfFind` to return the specified mfArrays.

See Also

mfTranspose, mfT

Return mfArray transpose.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfTranspose(const mfArray &x);
```

Descriptions

Function `mfTranspose(x)` returns an mfArray containing the transpose of mfArray `x`. In a transpose operation, x_{ij} and x_{ji} are interchanged.
Complex elements are not conjugated.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfTranspose.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, t;

    x = mfMagic(3);

    t = mfTranspose(x);

    mfDisplay(x, "x", t, "t");
}
```

Result

```
x =

  8  1  6
  3  5  7
  4  9  2

 t =

  8  3  4
  1  5  9
  6  7  2
```

mfHCat

Horizontal concatenation.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfHCat(const mfArray& v1, const mfArray& v2[, const mfArray&
v3, ..., const mfArray& v32);
```

Descriptions

Function `mfHCat` performs a horizontal concatenation of mfArrays.
mfHCat(x, y) concatenates mfArray `x` and `y` horizontally. The two mfArrays must have the same number of rows.

When you concatenate two arrays of different data types, the data type with higher precedence is used to store the data of the returned mfArray. Complex has higher precedence over real one, and the real one has higher precedence over integer type.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfHCat.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y, m;

    x = mfMagic(3);
    y = mfOnes(3,1);

    m = mfHCat(x, y);

    mfDisplay(x, "x", y, "y", m, "m");

}
```

Result

```
x =

  8  1  6
  3  5  7
  4  9  2

 y =

  1
```

```
  1
  1

 m =

  8  1  6  1
  3  5  7  1
  4  9  2  1
```

See Also

mfVCat

mfVCat

Horizontal concatenation.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfVCat(const mfArray& v1, const mfArray& v2[, const mfArray&
v3, ..., const mfArray& v32);
```

Descriptions

Function `mfVCat` performs a vertical concatenation of mfArrays. Using together with `mfHCat`, you can construct multi-dimensional mfArrays easily.
`mfVCat(x, y)` concatenates mfArray `x` and `y` vertically. The two mfArrays must have the same number of columns.

When you concatenate two arrays of different data types, the data type with higher precedence is used to store the data of the returned mfArray. Complex has higher precedence over real , and real has higher precedence over integer type.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfVCat.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y, m;

    x = mfMagic(3);
    y = mfOnes(2,3);
    m = mfVCat(x, y);

    mfDisplay(x, "x", y, "y", m, "m");
}
```

Result

```
x =

  8  1  6
  3  5  7
  4  9  2

 y =

  1  1  1
```

```
  1  1  1

 m =

  8  1  6
  3  5  7
  4  9  2
  1  1  1
  1  1  1
```

See Also

mfHCat

File IO

mfLoad

Load binary file into mfArray.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfLoad(char *fileName);
```

Descriptions

Function `mfLoad` reads data from a binary file, created by function `mfSave`, into an mfArray.

`x = mfLoad(filename)`

- Argument `filename` is a C string containing the name of the binary file. For example, `x = mfLoad("y.mfb")`.
- The MATFOR binary file uses ".mfb" as a file extension.

Example

The following example uses function `mfLoad`.

Code

```
// ************************************************
// AnCAD example file
// file: mfLoad.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;

    x = mfReshape(mfColon(1, 6), 2, 3);
    // x    is 1 3 5
    //         2 4 6

    mfSave("x.mfb", x);
    y = mfLoad("x.mfb");
    mfDisplay(x, "x", y, "y");

}
```

Result

```
x =

  1  3  5
  2  4  6

 y =
```

```
1  3  5
2  4  6
```

See Also

mfSave, mfSaveAscii, mfLoadAscii

mfLoad.m

Load MATFOR binary file into MATLAB workspace.

Module

`mfLoad.dll`

Descriptions

Function `mfLoad` retrieves data in a MATFOR *.mfb binary file into a MATLAB matrix `x`.

```
x = mfLoad(filename)
```

- Argument `filename` is a string containing the name of the target binary file. An extension of .mfb is assumed if not specified.
- You can use this function to load data saved using `mfSave` in MATFOR and mfSave.m in MATLAB.

Note that to use this function, ensure that the files mfLoad.m and mfLoad.dll are in your MATLAB working directory. The two files are installed in your MATFOR directory <MATFOR>/common/tools/matlab when MATFOR is installed.

See Also

mfSave, mfLoad, mfSave.m

mfLoadAscii

Load ASCII file into mfArray.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfLoad(char *fileName);
```

Descriptions

Function `mfLoadAscii` reads in data from an ASCII file into an mfArray.

`x = mfLoadAscii(filename)`

- The ascii file must be in 2-D matrix format with m rows and n columns corresponding to an m x n matrix mfArray.
- Argument `filename` is a string containing the filename of the ASCII file. For example, `x = mfLoadAscii("y.txt")`.

You can use function `mfSaveAscii` to produce an ASCII file for loading by using function `mfLoadAscii`.

Example

The following example uses the `mfSaveAscii` function.

Code

```
// ************************************************
// AnCAD example file
// file: mfLoadAscii.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;

    x = mfReshape(mfColon(1, 6), 2, 3);
    //  x    is  1 3 5
    //           2 4 6

    mfSaveAscii("x.txt", x);
    y = mfLoadAscii("x.txt");
    mfDisplay(x, "x", y, "y");

}
```

Result

```
x =
```

```
  1  3  5
  2  4  6

 y =

  1  3  5
  2  4  6
```

See Also

mfSave, mfLoad, mfSaveAscii

mfSave

Save mfArray as binary file.

Module

`cml.h/cml.lib`

C++ Prototype

```
void mfSave(char *fileName, const mfArray &x);
```

Descriptions

Function `mfSave` writes data contained in an mfArray to a binary file with the extension 'mfb.'

`mfSave(filename, x)`

- Argument `filename` is a C string containing the name of the target binary file. For example, `mfSave("x.mfb",x)`.
- Argument `x` is the name of the mfArray targeted for output.

Example

The following example uses the function `mfSave`.

Code

```
// ************************************************
// AnCAD example file
// file: mfSave.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x;

    x = mfReshape(mfColon(1, 6), 2, 3);
    // x    is 1 3 5
    //         2 4 6

    mfSave("x.mfb", x);

}
```

See Also

mfLoad, mfSaveAscii, mfLoadAscii

mfSave.m

Save MATLAB matrix data as MATFOR binary file.

Module

`mfSave.dll`

Descriptions

Function `mfSave(filename, x)` saves data contained in MATLAB matrix `x` in MATFOR *.mfb binary file format.

- Argument `filename` is a string containing the name of the target binary file. An extension of .mfb is assumed if not specified.
- The data saved by using this function can be reloaded into MATLAB workspace using function `mfLoad` or retrieved in C++ using function `mfLoad`. This facilitates exchange of data between C++ and MATLAB environment.

Note that to use this function, ensure that the files mfSave.m and mfSave.dll are in your MATLAB working directory. The two files are installed in your MATFOR directory <MATFOR>/common/tools/matlab when MATFOR is installed.

See Also

mfSave, mfLoad, mfLoad.m

mfSaveAscii

Save mfArray as ASCII file.

Module

`cml.h/cml.lib`

C++ Prototype

```
void mfSaveAscii(char *fileName, const mfArray &x);
```

Descriptions

Function `mfSaveAscii` writes data contained in an mfArray to an ASCII file with the extension ".dat".

`mfSaveAscii(filename, x)`

- The data is written in a 2-D matrix format, with m rows and n columns, corresponding to an m x n matrix mfArray.
- Argument `filename` is a string containing the name of the target binary file. For example, `mfSaveAscii("x.dat", x)`.
- Argument `x` is the name of the mfArray targeted for output.

Example

The following example uses the `mfSaveAscii` function.

Code

```
// ************************************************
// AnCAD example file
// file: mfSaveAscii.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {
    mfArray x, y;

    x = mfReshape(mfColon(1, 6), 2, 3);
    //  x    is  1 3 5
    //           2 4 6

    mfSaveAscii("x.txt", x);
}
```

See Also

mfSave, mfLoad, mfLoadAscii

CHAPTER 3

Data Manipulation Functions

mfMax

Find the maximum value.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfMax(mfArray x);
mfArray mfMax(mfArray x1, mfArray x2);
mfArray mfMax(mfArray x, mfArray Empty, int IDIM);
```

Descriptions

`a = mfMax(x)`
For vectors, `mfMax` returns the largest element in `x`. For matrices, `mfMax` returns a row vector containing the maximum elements from each column.

`y = mfMax(x, mf(), IDIM)` operates along the dimension `IDIM`.

`a = mfMax(x, y)` returns an array the same size as `x` and `y` with the larger elements taken from `x` or `y`. Either one can be a scalar.

For complex number, its absolute value is used for comparison.

Example

The following example uses the `mfMax` function.

Code

```
// ************************************************
// AnCAD example file
// file: mfMax.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, c, y;
    int i;

    x = mfReshape(mfColon(1, 6), 2, 3);
    //  x    is  1 3 5
    //           2 4 6

    y = mfReshape(mfV(3,5,4,6,2,1), 2, 3);
    //  y    is  3 4 2
    //           5 6 1

    i = mfMax(mfColon(1,3)).ToDouble();
    mfDisplay(mf(i), "mfMax(mfColon(1,3))");
```

```
    c = mfMax(x, mf(), 1);
    mfDisplay(x, "x", y, "y", c,"mfMax(x, mf(), 1)");
    c = mfMax(x, mf(), 2);
    mfDisplay(x, "x", y, "y", c,"mfMax(x, mf(), 2)");
    c = mfMax(x, y);
    mfDisplay(c, "mfMax(x, y)");

}
```

Result

```
mfMax(mfColon(1,3)) =

  3

 x =

  1  3  5
  2  4  6

 y =

  3  4  2
  5  6  1

 mfMax(x, mf(), 1) =

  2  4  6

 x =

  1  3  5
  2  4  6

 y =

  3  4  2
  5  6  1

 mfMax(x, mf(), 2) =

  5
  6

 mfMax(x, y) =

  3  4  5
  5  6  6
```

See Also

mfMin

mfMin

Find the minimum value.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfMin(mfArray x);
mfArray mfMin(mfArray x1, mfArray x2);
mfArray mfMin(mfArray x, mfArray Empty, int IDIM);
```

Descriptions

`a = mfMin(x)`
For vectors, `mfMin` returns the largest element in `x`. For matrices, `mfMin` returns a row vector containing the minimum elements from each column.

`y = mfMin(x, mf(), IDIM)` operates along the dimension `IDIM`.

`a = mfMin(x, y)` returns an array the same size as `x` and `y` with the smaller elements taken from `x` or `y`. Either one can be a scalar.

For complex number, its absolute value is used for comparison.

Example

The following example uses the `mfMin` function.

Code

```
// ************************************************
// AnCAD example file
// file: mfMin.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, c, y;
    int i;

    x = mfReshape(mfColon(1, 6), 2, 3);
    //  x    is  1 3 5
    //           2 4 6
    y = mfM(2, 3, 3,5,4,6,2,1);
    //  y    is  3 4 2
    //           5 6 1

    i = (int) mfMin(mfColon(1,3)).ToDouble();
    mfDisplay(i, "mfMin(mfColon(1,3))");

    c = mfMin(x, mf(), 1);
```

```
    mfDisplay(x, "x", y, "y", c, "mfMin(x, mf(), 1)");
    c = mfMin(x, mf(), 2);
    mfDisplay(c, "mfMin(x, mf(), 2)");
    c = mfMin(x, y);
    mfDisplay(c, "mfMin(x,y) ");
}
```

Result

```
mfMin(mfColon(1,3)) =

  1

 x =

  1  3  5
  2  4  6

 y =

  3  4  2
  5  6  1

 mfMin(x, mf(), 1) =

  1  3  5

 mfMin(x, mf(), 2) =

  1
  2

 mfMin(x,y)  =

  1  3  2
  2  4  1
```

See Also

mfMax

mfProd

Product of elements.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfProd(mfArray x[, int IDIM]);
```

Descriptions

`a = mfProd(x)`

- For matrices, `mfProd(x)` returns a row vector containing the product over each column of mfArray x.
- For N-dimension arrays, `mfProd(x)` operates on the first non-singleton dimension.

`a = mfProd(x, IDIM)`

- Argument `IDIM` is an int scalar specifying the dimension where function `mfProd(x)` works along.

Example

The following example uses the `mfProd` function.

Code

```
// ************************************************
// AnCAD example file
// file: mfProd.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y, c;
    int i;

    x = mfColon(1, 3);
    y = mfReshape(mfColon(1, 6), 2, 3);
    //  x  is  1 3 5
    //         2 4 6

    i = mfProd(x).ToDouble();     // returns 6
    mfDisplay(x, "x", i, "mfProd(x)");
    c = mfProd(y, 1);  // returns 2 12 30
    mfDisplay(y, "y", c, "mfProd(y, 1)");

}
```

Result

```
x =

  1  2  3

 mfProd(x) =

  6

 y =

  1  3  5
  2  4  6

 mfProd(y, 1) =

   2  12  30
```

See Also

mfSum

mfSort

Sort in ascending order.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfSort(mfArray x[, int IDIM]);
```

Descriptions

`y = mfSort(x), y=mfSort(x, IDIM)`

- The function returns mfArray `y` containing the elements of mfArray `x` sorted in ascending order along different dimensions.
- For vector `x`, the elements are sorted in ascending order.
- For matrix `x`, the elements in each column of `x` are sorted in ascending order accordingly. In effect, the function returns an mfArray `y` assuming the shape of `x`, with the elements in each column sorted in ascending order.
- For n-dimensional `x`, the elements are sorted along the first non-singleton dimension of `x`.
- Argument `IDIM` is an integer specifying the dimension to be sorted along. By default, `IDIM = 1`, corresponding to sorting along the column.
- When `x` is complex, the function returns mfArray y containing sorted elements of `mfAbs(x)`. Complex matches are further sorted according to `mfAtan2(mfImag(x), mfReal(x))`.

`mfSort(mfOut(y, i), x)`

- This format returns an additional index mfArray `i`, containing the indices of elements in `x`, corresponding to the sorted elements in `y`.

For example,

```
x = 8   1   6
    3   5   7
    4   9   2
```

`mfSort(mfOut(y, i), x)` returns mfArray `y` and `i` containing,

```
y = 3   1   2
    4   5   6
    8   9   7

i = 2   1   3
```

```
    3   2   1
    1   3   2
```

- For two values of the same magnitude, their original orders are preserved.

Example

The following example uses the `mfSort` function.

Code

```
// ************************************************
// AnCAD example file
// file: mfSort.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, a;

    x = mfReshape(mfV(3.0, 5.0, 4.0, 6.0, 2.0, 1.0), 2, 3);
    //  x    is  3 4 2
    //           5 6 1

    a = mfSort(x) ;
    mfDisplay(x, "x", a, "mfSort(x)");
    a = mfSort(x,2);
    mfDisplay(x, "x", a, "mfSort(x,2)");

}
```

Result

```
x =

  3  4  2
  5  6  1

 mfSort(x) =

  3  4  1
  5  6  2

 x =

  3  4  2
  5  6  1

 mfSort(x,2) =

  2  3  4
  1  5  6
```

See Also

mfSortRows

mfSortRows

Sort rows in ascending order.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfSortRows(mfArray x[, int col]);
void mfSortRows(mfOutArray outArray, mfArray x[, int col]);
```

Descriptions

`y = mfSortRows(x), y = mfSortRows(x, col)`

- The function returns mfArray `y` containing the rows of matrix mfArray `x` sorted in ascending order as a group.
- If `x` contains strings, the function performs a dictionary sort.
- When complex `x`, the rows are sorted according to `mfAbs(x)`. Complex matches are further sorted according to `mfAtan2(mfImag(x), mfReal(x))`.
- If argument `col` is specified, the rows are sorted according to the columns specified in vector mfArray `col`. For example,

```
   x = 8   1   6
       3   5   7
       4   9   2

col = 3

y = mfSortRows(x, col) returns,

   y = 4   9   2
       8   1   6
       3   5   7
```

`mfSortRows(mfOut(y,i),x)`

- This format returns an additional index mfArray `i`, which contains the row indices after the corresponding rows are being swapped.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfSortRows.cpp
//
```

```
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"


void main() {

    mfArray x, y, i;

    x = mfMagic(5);

    // Sort according to rows and retrieve index
    mfSortRows(mfOut(y, i), x);
    mfDisplay(x, "x", y, "Sort rows according to first column");
    mfDisplay(i, "Corresponding index");

}
```

Result

```
x =

  17  24   1   8  15
  23   5   7  14  16
   4   6  13  20  22
  10  12  19  21   3
  11  18  25   2   9

 Sort rows according to first column =

   4   6  13  20  22
  10  12  19  21   3
  11  18  25   2   9
  17  24   1   8  15
  23   5   7  14  16

 Corresponding index =

  1
  2
  3
  4
  5
```

See Also

mfSort

mfSum

Find the sum of elements.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfSum(mfArray x[, int IDIM]);
```

Descriptions

`a = mfSum(x), a = mfSum(x, IDIM)`
`a = mfSum(x)`
For vectors, the function returns the sum of the elements of `x`. For matrices, `a` is a row vector with the sum over each column. For N-Dimension arrays, `a` operates along the first non-singleton dimension.

`a = mfSum(X, IDIM)` sums along the dimension `IDIM`.

Example

The following example uses the `mfSum` function.

Code

```
// ************************************************
// AnCAD example file
// file: mfSum.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y, c;
    int i;

    x = mfColon(1, 3);
    y = mfReshape(mfColon(1, 6), 2, 3);
    //  x    is  1 3 5
    //           2 4 6

    i = mfSum(x).ToDouble();            // returns 6
    mfDisplay(x, "x", mf(i), "mfSum(x)");
    c = mfSum(y, 1);          // returns 3 7 11
    mfDisplay(y, "y", c, "mfSum(y, 1)");

}
```

Result

```
x =

  1  2  3
```

```
mfSum(x) =

 6

y =

 1  3  5
 2  4  6

mfSum(y, 1) =

  3   7  11
```

See Also

mfProd

CHAPTER 4

ELFUN

It contains a set of elementary math functions including trigonometric, exponential, complex, rounding and remainder.

The functions are listed below:

Trigonometric	
mfACos	Inverse cosine function
mfACosh	Inverse hyperbolic cosine function
mfACot	Inverse cotangent function
mfACoth	Inverse hyperbolic cotangent function
mfACsc	Inverse cosecant function
mfACsch	Inverse hyperbolic cosecant function
mfASec	Inverse secant function.
mfASech	Inverse hyperbolic secant function.
mfASin	Inverse sine function.
mfASinh	Inverse hyperbolic sine function.
mfATan	Inverse tangent function.
mfATan2	Four quadrant arctangent function
mfATanh	Inverse hyperbolic tangent function
mfCos	Cosine function
mfCosh	Hyperbolic cosine function
mfCot	Cotangent function
mfCoth	Hyperbolic cotangent function
mfCsc	Cosecant function
mfCsch	Hyperbolic cosecant function
mfSec	Secant function
mfSech	Hyperbolic secant function

`mfSin`	Sine function
`mfSinh`	Hyperbolic sine function
`mfTan`	Tangent function
`mfTanh`	Hyperbolic tangent function

Exponential

`mfExp`	Exponential function
`mfLog`	Natural logarithm
`mfLog10`	Common logarithm (base 10)
`mfLog2`	Base 2 Logarithmic and floating point dissection
`mfPow2`	Base 2 power and floating point number scaling
`mfSqrt`	Square Root function

Complex

`mfAbs`	Absolute of real and complex value
`mfAngle`	Phase angle of complex
`mfComplex`	Convert input number to complex
`mfConj`	Conjugate of complex
`mfImag`	Imaginary part of complex
`mfReal`	Real part of complex

Rounding and Remainder

`mfCeil`	Round towards positive infinity
`mfFix`	Round towards zero
`mfFloor`	Round towards minus infinity
`mfMod`	Modulus (signed remainder after division)
`mfRem`	Remainder after division
`mfRound`	Round towards nearest integer
`mfSign`	Signum function

Trigonometry

mfACos

Request inverse cosine function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfACos(mfArray x);
```

Descriptions

Function `mfACos(x)` returns the arccosine of the mfArray `x`, in radians, where

$\cos^{-1}(x) = -i*\log[x + i*(1-x^2)^{1/2}]$
The function domain and ranges include real and complex data.
For |x| <= 1, the function result is real and lies in the range 0 to π.
For |x| > 1, the function returns a complex value.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfACos.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;

    x = 2.02;
    y = mfACos(x);

    mfDisplay(x, "x", y, "mfACos(x)");

}
```

Result

```
x =

   2.0200

 mfACos(x) =

 0.0000 +1.3284i
```

See Also

mfACosh, mfCos

mfACosh

Request inverse hyperbolic cosine function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfACosh(mfArray x);
```

Descriptions

Function `mfACosh(x)` returns element-by-element the inverse hyperbolic cosine of the mfArray `x`, in radians, where

$\cosh^{-1}(x) = \log[x + (x^2 - 1)^{1/2}]$

The function domain and ranges include real and complex data.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfACosh.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {
    mfArray x, y;

    x = dcomplex(2, -2);
    y = mfACosh(x);

    mfDisplay(x, "x", y, "mfACosh(x)");
}
```

Result

```
x =

 2.0000 -2.0000i

 mfACosh(x) =

 1.7343 -0.8165i
```

See Also

mfCos, mfCosh

mfACot

Request inverse cotangent function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfACot(mfArray x);
```

Descriptions

Function `mfACot(x)` returns element-by-element the arccotangent of the mfArray `x`, in radians, where

$\cot^{-1}(x) = \tan^{-1}(1/x)$

The function domain and range includes real and complex data.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfACot.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {
    mfArray x, y;

    x = dcomplex(-5, -5);
    y = mfACot(x);

    mfDisplay(x, "x", y, "mfACot(x)");
}
```

Result

```
x =

-5.0000 -5.0000i

 mfACot(x) =

-0.1007 +0.0993i
```

See Also

mfACoth, mfCot, mfCoth

mfACoth

Request inverse hyperbolic cotangent function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfACoth(mfArray x);
```

Descriptions

Function `mfACoth(x)` returns element-by-element the inverse hyperbolic cotangent of the mfArray `x` in radians, where

$\coth^{-1}(x) = \tanh^{-1}(1/x)$

The function domain and range includes real and complex data. For |x| <= 1, the function result is complex or infinity.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfACoth.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;

    x = 0.5;
    y = mfACoth(x);

    mfDisplay(x, "x", y, "mfACoth(x)");
}
```

Result

```
x =

   0.5000

 mfACoth(x) =

 0.5493 -1.5708i
```

See Also

mfACot, mfCot, mfCoth

mfACsc

Request inverse cosecant function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfACsc(mfArray x);
```

Descriptions

Function `mfACsc(x)` returns element-by-element the inverse cosecant of the mfArray `x` in radians, where

$\csc^{-1}(x) = \sin^{-1}(1/x)$

The function domain and range includes real and complex data.
For $|x| < 1$, the function result is complex or NaN.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfACsc.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;

    x = 3.0;
    y = mfACsc(x);

    mfDisplay(x, "x", y, "mfACsc(x)");
}
```

Result

```
x =

  3

 mfACsc(x) =

   0.3398
```

See Also

mfACsch, mfCsc, mfCsch

mfACsch

Request inverse hyperbolic cosecant function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfACsch(mfArray x);
```

Descriptions

Function `mfACsch(x)` returns element-by-element the inverse hyperbolic cosecant of the mfArray `x` in radians, where

$\text{csch}^{-1}(x) = \sinh^{-1}(1/x)$
The function domain and range includes real and complex data.
For x = 0, the function result is infinity.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfACsch.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;
    x = 0.5;
    y = mfACsch(x);

    mfDisplay(x, "x", y, "mfACsch(x)");
}
```

Result

```
x =

   0.5000

 mfACsch(x) =

   1.4436
```

See Also

mfACsc, mfCsc, mfCsch

mfASec

Request inverse secant function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfAsec(mfArray x);
```

Descriptions

Function `mfASec(x)` returns element-by-element the inverse secant of the mfArray `x` in radians, where

$\sec^{-1}(x) = \cos^{-1}(1/x)$

The function domain and range includes real and complex data.
For $|x| < 1$, the function result is complex or NaN.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfASec.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;

    x = 0.1;
    y = mfASec(x);

    mfDisplay(x, "x", y, "mfASec(x)");
}
```

Result

```
x =

   0.1000

 mfASec(x) =

 0.0000 +2.9932i
```

See Also

mfASech, mfSec, mfSech

mfASech

Request inverse hyperbolic secant function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfASech(mfArray x);
```

Descriptions

Function `mfASech(x)` returns element-by-element the inverse hyperbolic secant of the mfArray `x` in radians, where

$\text{sech}^{-1}(x) = \cosh^{-1}(1/x)$

The function domain and range includes real and complex data.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfASech.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;
    x = 0.1;
    y = mfASech(x);

    mfDisplay(x, "x", y, "mfASech(x)");
}
```

Result

```
x =

   0.1000

 mfASech(x) =

   2.9932
```

See Also

mfASec, mfSec, mfSech

mfASin

Request inverse sine function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfASin(mfArray x);
```

Descriptions

Function `mfASin(x)` returns element-by-element the arcsine of the mfArray `x` in radians, where

$\sin^{-1}(x) = -i*\log[i*x+(1-x^2)^{1/2}]$
The function domain and range includes real and complex data.
For real $|x| < 1$, the function result is real and lies in the range $[-\pi/2, \pi/2]$.
For real $|x| > 1$, the function returns a complex value.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfASin.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {
    mfArray x, y;

    x = 0.5;
    y = mfASin(x);

    mfDisplay(x, "x", y, "mfASin(x)");
}
```

Result

```
x =

   0.5000

 mfASin(x) =

   0.5236
```

See Also

mfASinh, mfSin, mfSinh

mfASinh

Request inverse hyperbolic sine function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfASinh(mfArray x);
```

Descriptions

Function `mfASinh(x)` returns element-by-element the inverse hyperbolic sine of the mfArray `x` in radians, where

$\sinh^{-1}(x) = \log[x + (x^2+1)^{1/2}]$

The function domain and range includes real and complex data.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfASinh.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {
    mfArray x, y;

    x = dcomplex(2.0, -2.0);
    y = mfASinh(x);

    mfDisplay(x, "x", y, "mfASinh(x)");
}
```

Result

```
x =

 2.0000 -2.0000i

 mfASinh(x) =

 1.7343 -0.7542i
```

See Also

mfASin, mfSin, mfSinh

mfATan

Request inverse tangent function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfATan(mfArray x);
```

Descriptions

Function `mfATan(x)` returns element-by-element the inverse tangent of the mfArray `x` in radians, where

$\tan^{-1}(x) = (i/2)* \log((i+x)/(i-x))$
The function domain and range include real and complex data.
For real x, the result lies in the range $[-\pi/2, \pi/2]$.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfATan.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;

    x = 0.154;
    y = mfATan(x);

    mfDisplay(x, "x", y, "mfATan(x)");
}
```

Result

```
x =

   0.1540

 mfATan(x) =

   0.1528
```

See Also

mfATan2, mfATanh, mfTan, mfTanh

mfATan2

Request four quadrant arctangent function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfATan2(mfArray x, mfArray y);
```

Descriptions

Function `mfATan2(y, x)` returns an mfArray `z`, containing element-by-element the principal value of the complex number defined by mfArrays `y` and `x`, where `y` is the imaginary part and `x` is the real part.

For x approaching to 0, the function result is approximately mfATan(y/x). However, in contrast to mfATan(y/x), which is limited to the range $-\pi/2 <= mfATan(y/x) <= \pi/2$, the function result lies in the range of - mfATan2(y,x) in four quadrant.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfATan2.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;

    x = 0.5;
    y = mfATan2(x, x);

    mfDisplay(x, "x", y, "mfATan2(x, x)");
}
```

Result

```
x =

   0.5000

 mfATan2(x, x) =

   0.7854
```

See Also

mfATan, mfATanh, mfTan, mfTanh

mfATanh

Request inverse hyperbolic tangent function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfATanh(mfArray x);
```

Descriptions

Function `mfATanh(x)` returns element-by-element the inverse hyperbolic tangent of the mfArray `x` in radians, where

$\tan^{-1}(x) = (1/2)* \log((1+x)/(1-x))$
The function domain and range include real and complex data.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfATanh.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {
    mfArray x, y;

    x = 1.5782;
    y = mfATanh(x);

    mfDisplay(x, "x", y, "mfATanh(x)");
}
```

Result

```
x =

   1.5782

 mfATanh(x) =

 0.7475 -1.5708i
```

See Also

mfATan, mfATan2, mfTan, mfTanh

mfCos

Request cosine function.

Module

`elfun.h/elfun.lib`

C++ Prototype

```
mfArray mfCos(mfArray x);
```

Descriptions

Function `mfCos(x)` returns element-by-element the circular cosine of mfArray `x`, where `x` is in radians.

The function domain and range include real and complex data.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfCos.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;

    x = dcomplex(3.14, -3.14);
    y = mfCos(x);

    mfDisplay(x, "x", y, "mfCos(x)");
}
```

Result

```
x =

 3.1400 -3.1400i

 mfCos(x) =

-11.5736  +0.0184i
```

See Also

mfACos, mfACosh, mfCosh

mfCosh

Request hyperbolic cosine function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfCosh(mfArray x);
```

Descriptions

Function `mfCosh(x)` returns element-by-element the hyperbolic cosine of mfArray `x` where `x` is in radians.

The function domain and range include real and complex data.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfCosh.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;

    x = 0.1;
    y = mfCosh(x);

    mfDisplay(x, "x", y, "mfCosh(x)");
}
```

Result

```
x =


   0.1000

 mfCosh(x) =

   1.0050
```

See Also

mfACos, mfACosh, mfCos

mfCot

Request cotangent function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfCot(mfArray x);
```

Descriptions

Function `mfCot(x)` returns element-by-element the cotangent of mfArray `x` in radians where

mfCot(x) = 1/mfTan(x).
The function domain and range include real and complex data.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfCot.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {
    mfArray x, y;

    x = 0.7;
    y = mfCot(x);

    mfDisplay(x, "x", y, "mfCot(x)");
}
```

Result

```
x =

   0.7000

 mfCot(x) =

   1.1872
```

See Also

mfCoth, mfACot, mfACoth

mfCoth

Request hyperbolic cotangent function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfCoth(mfArray x);
```

Descriptions

Function `mfCoth(x)` returns element-by-element the hyperbolic cotangent of the mfArray `x` in radians, where

mfCoth(x) = 1/mfTanh(x).
The function domain and range include real and complex data.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfCoth.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {
    mfArray x, y;

    x = 0.5;
    y = mfCoth(x);

    mfDisplay(x, "x", y, "mfCoth(x)");
}
```

Result

```
x =

   0.5000

 mfCoth(x) =

   2.1640
```

See Also

mfCot, mfACot, mfACoth

mfCsc

Request cosecant function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfCsc(mfArray x);
```

Descriptions

Function `mfCsc(x)` returns the element-by-element cosecant of mfArray `x`, where

mfCsc(x) = 1/mfSin(x)
The function domain and range include real and complex data. All angles are in radians.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfCsc.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {
    mfArray x, y;

    x = 2.0;
    y = mfCsc(x);

    mfDisplay(x, "x", y, "mfCsc(x)");
}
```

Result

```
x =

  2

 mfCsc(x) =

   1.0998
```

See Also

mfCsch, mfACsc, mfACsch

mfCsch

Request hyperbolic cosecant function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfCsch(mfArray x);
```

Descriptions

Function `mfCsch(x)` returns element-by-element the hyperbolic cosecant of mfArray `x`, where

mfCsch(x) = 1/mfSinh(x)
The function domain and range include real and complex data. All angles are in radians.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfCsch.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {
    mfArray x, y;

    x = 0.5;
    y = mfCsch(x);

    mfDisplay(x, "x", y, "mfCsch(x)");
}
```

Result

```
x =

   0.5000

 mfCsch(x) =

   1.9190
```

See Also

mfCsc, mfACsc, mfACsch

mfSec

Request secant function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfSec(mfArray x);
```

Descriptions

Function `mfSec(x)` returns element-by-element the secant of mfArray `x`, where

mfSec(x) = 1/mfCos(x).
The function domain and range include real and complex data. All angles are in radians.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfSec.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;

    x = MF_PI;
    y = mfSec(x);

    mfDisplay(x, "x", y, "mfSec(x)");

}
```

Result

```
x =

   3.1416

 mfSec(x) =

 -1
```

See Also

mfSech, mfASec, mfASech

mfSech

Request hyperbolic secant function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfSech(mfArray x);
```

Descriptions

Function `mfSech(x)` returns element-by-element the hyperbolic secant of mfArray `x`, where

mfSech(x) = 1/mfCosh(x).
The function domain and range include real and complex data. All angles are in radians.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfSech.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {
    mfArray x, y;

    x = dcomplex(-1, 4);
    y = mfSech(x);

    mfDisplay(x, "x", y, "mfSech(x)");
}
```

Result

```
x =

-1.0000 +4.0000i

 mfSech(x) =

-0.5578 -0.4918i
```

See Also

mfSec, mfASec, mfASech

mfSin

Request sine function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfSin(mfArray x);
```

Descriptions

Function `mfSin(x)` returns element-by-element the circular sine of mfArray `x`.
The function domain and range include real and complex data. All angles are in radians.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfSin.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {
    mfArray x, y;

    x = 0.5236;
    y = mfSin(x);

    mfDisplay(x, "x", y, "mfSin(x)");
}
```

Result

```
x =

   0.5236

 mfSin(x) =

   0.5000
```

See Also

mfSinh, mfASin, mfASinh

mfSinh

Request hyperbolic sine function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfSinh(mfArray x);
```

Descriptions

Function `mfSinh(x)` returns element-by-element the hyperbolic sine of mfArray `x` where `x` is in radians.
The function domain and range include real and complex data.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfSinh.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;

    x = 3;
    y = mfSinh(x);

    mfDisplay(x, "x", y, "mfSinh(x)");

}
```

Result

```
x =

  3

 mfSinh(x) =

   10.0179
```

See Also

mfSin, mfASin, mfASinh

mfTan

Request tangent function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfTan(mfArray x);
```

Descriptions

Function `mfTan(x)` returns element-by-element the tangent of mfArray `x`.
The function domain and range include real and complex data. All angles are in radians.
mfTan(x) = mfSin(x)/mfCos(x)

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfTan.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;

    x = 1.0;
    y = mfTan(x);

    mfDisplay(x, "x", y, "mfTan(x)");

}
```

Result

```
x =

  1

 mfTan(x) =

   1.5574
```

See Also

mfTanh, mfATan, mfATanh

mfTanh

Request hyperbolic tangent function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfTanh(mfArray x);
```

Descriptions

Function `mfTanh(x)` returns element-by-element the hyperbolic tangent of mfArray `x`, where

mfTanh(x) = mfSinh(x)/mfCosh(x)
The function domain and range include real and complex data. All angles are in radians.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfTanh.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {
    mfArray x, y;

    x = 1;
    y = mfTanh(x);

    mfDisplay(x, "x", y, "mfTanh(x)");
}
```

Result

```
x =

  1

 mfTanh(x) =

   0.7616
```

See Also

mfTan, mfATan, mfATanh

Exponential

mfExp

Request exponential function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfExp(mfArray x);
```

Descriptions

Function `mfExp(x)` returns element-by-element the exponential of mfArray `x`. The function domain and range include real and complex data.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfExp.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;

    x = dcomplex(2.0, 3.0);
    y = mfExp(x);

    mfDisplay(x, "x", y, "mfExp(x)");
}
```

Result

```
x =

 2.0000 +3.0000i

 mfExp(x) =

-7.3151 +1.0427i
```

See Also

mfLog, mfLog10

mfLog

Request natural logarithm.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfLog(mfArray x);
```

Descriptions

Function `mfLog(x)` returns element-by-element the natural logarithm of mfArray `x`. The function domain and range include real and complex data.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfLog.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;

    x = MF_E;
    y = mfLog(x);

    mfDisplay(x, "x", y, "mfLog(x)");

}
```

Result

```
x =

   2.7183

 mfLog(x) =

  1
```

See Also

mfExp, mfLog10, mfLog2

mfLog10

Common logarithm (base 10).

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfLog10(mfArray x);
```

Descriptions

Function `mfLog10(x)` returns element-by-element the natural logarithm of mfArray `x`. The function domain and range include real and complex data.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfLog10.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;

    x = 5.0;
    y = mfLog10(x);

    mfDisplay(x, "x", y, "mfLog10(x)");

}
```

Result

```
x =

  5

 mfLog10(x) =

   0.6990
```

See Also

mfExp, mfLog, mfLog2

mfLog2

Base 2 logarithm and floating-point dissection.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfLog2(mfArray x);
```

Descriptions

Function `mfLog2` computes base 2 logarithm or extract mantissa and exponent of a floating-point number.

`y = mfLog2(x)` returns the elemental base 2 logarithm of mfArray `x`.

`mfLog2(mfOut(f,e),x)` dissects each element of mfArray `x` into the binary floating-point format consisting of a mantissa, mfArray `f`, and an exponent, mfArray `e`, where x = f*(2**e) for real `x`. MfArray `f` contains real values lying in the range of 0.5 <= mfAbs(f) < 1.
For elements of x=0, the corresponding elements of `f` and `e` are equal to zero.

Example

The example below evaluates the `mfLog2` of mfArray `x`.

Code

```
// ************************************************
// AnCAD example file
// file: mfLog2.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y, f, e;

    // MF_PI is MATFOR intrinsic parameter for pi.

    x = mfV(MF_PI, 2* MF_PI, 3* MF_PI);
    y = mfLog2(x);

    mfLog2(mfOut(f, e), x);
    mfDisplay(y, "mfLog2(x)");
    mfDisplay(f, "the mantissa", e, "the exponent");

}
```

Result

```
mfLog2(x) =
```

```
  1.6515   2.6515   3.2365

 the mantissa =

  0.7854   0.7854   0.5890

 the exponent =
  2  3  4
```

See Also

mfLog, mfPow2, mfLog10

mfPow2

Base 2 power and floating point number scaling.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfPow2(mfArray x1[, mfArray x2]);
```

Descriptions

Function `mfPow2` computes base 2 power or floating point number scaling.

`y = mfPow2(x)` returns an mfArray `y` with elements computed from two raised to the power of each element of mfArray `x`, i.e., y = 2**x.

`y = mfPow2(f, e)` returns the mfArray `y` containing elements computed from y = f*(2**e). The result is equivalent to scaling each element of `f` by exponent `e` or adding each element of `e` to the corresponding floating-point exponent of `e`.

Example

The example below evaluates the `mfPow2` of mfArray `x`, a 1-by-10 vector.

Code

```
// ************************************************
// AnCAD example file
// file: mfPow2.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y, z, f, e;

    // Initialize the mfArrays
    x = 3;
    f = mfV(1.0, 0.5, -0.75);
    e = mfV(2, 3, 5);

    // Compute base 2 power y = 2**x
    y = mfPow2(x);

    // Perform floating point scaling z = f*(2**e)
    z = mfPow2(f, e);

    // Display the values
    mfDisplay(x, "x", y, "mfPow2(x)");
    mfDisplay(f, "f", e, "e", z, "mfPow2(f, e)");
}
```

Result

```
x =
 3
mfPow2(x) =
 8
f =

  1.0000   0.5000  -0.7500

e =
 2  3  5
mfPow2(f, e) =
  4   4 -24
```

See Also

mfLog2, mfExp

mfSqrt

Square root function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfSqrt(mfArray x);
```

Descriptions

Function `mfSqrt(x)` returns element-by-element the square root of mfArray `x`. The function domain includes real and complex data.
For negative and complex elements of `x`, complex results are returned.

Example

The example below evaluates `mfSqrt(-4)`.

Code

```
// ************************************************
// AnCAD example file
// file: mfSqrt.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;

    x = -4.0;
    y = mfSqrt(x);

    mfDisplay(x, "x", y, "mfSqrt(x)");

}
```

Result

```
x =

 -4

 mfSqrt(x) =

 0.0000 +2.0000i
```

See Also

mfExp, mfLog, mfLog2, mfLog10

Complex

mfAbs

Absolute of real and complex value.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfAbs(mfArray x);
```

Descriptions

Function `mfAbs(x)` returns element-by-element the absolute of mfArray `x`.

For real `x`, `mfAbs(x)` returns |x|.

For complex z = x + iy, `mfAbs(z)` returns the magnitude of the complex computed as sqrt(x**2+y**2).

Example

The example below evaluates the absolute of complex (2, -2)

Code

```
// ************************************************
// AnCAD example file
// file: mfAbs.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;

    x = dcomplex(2, -2);
    y = mfAbs(x);

    mfDisplay(x, "x", y,"mfAbs(x)");
}
```

Result

```
x =

 2.0000 -2.0000i

 mfAbs(x) =

   2.8284
```

See Also

mfAngle, mfSign

mfAngle

Phase angle of complex.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfAngle(mfArray x);
```

Descriptions

Function `mfAngle(z)` returns an mfArray `p` containing the element-by-element phase angle of complex mfArray `z`.

Example

The example below evaluates the phase angle of complex mfArray `z` over a range of value.

Code

```
// ************************************************
// AnCAD example file
// file: mfAngle.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray theta, z, x, y;

    x = mfColon(0, 5, 10);
    y = mfColon(-2, 5, 8);
    z = mfComplex(x, y);
    theta = mfAngle(z);

    mfDisplay(z, "z", theta, "mfAngle(z)");

}
```

Result

```
z =

column   1  to   3

  0.0000  -2.0000i    5.0000  +3.0000i   10.0000  +8.0000i

 mfAngle(z) =

  -1.5708   0.5404   0.6747
```

See Also

mfAbs

mfComplex

Convert input numbers to complex.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfComplex(mfArray x[, mfArray y]);
```

Descriptions

Function `mfComplex(x, y)` returns a complex mfArray whose real part is specified by the elements of mfArray `x` and imaginary part is specified by elements of mfArray `y` ie z = x + yi. `z = mfComplex(x)` returns a complex mfArray whose real part is specified by the elements of mfArray `x` and imaginary part is zero ie z = x + 0i.

Example

The example below constructs a complex mfArray `z` from two real mfArrays `x` and `y`.

Code

```
// ************************************************
// AnCAD example file
// file: mfComplex.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray z, x, y;

    x = mfV(-5, 7, 0);
    y = mfV( 3, 1, 2);
    z = mfComplex(x, y);

    mfDisplay(x, "x", y, "y", z, "mfComplex(x, y)");

}
```

Result

```
x =

 -5  7  0

 y =

  3  1  2

 mfComplex(x, y) =

-5.0000 +3.0000i   7.0000 +1.0000i   0.0000 +2.0000i
```

See Also

mfImag, mfReal

mfConj

Conjugate of complex.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfConj(mfArray x);
```

Descriptions

Function `mfConj(z)` returns an mfArray containing the element-by-element conjugate of complex mfArray `z`.

Example

The example below evaluates the conjugate of complex mfArray `z`.

Code

```
// ************************************************
// AnCAD example file
// file: mfConj.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray c, z;

    z = dcomplex(-2, 2);
    c = mfConj(z);

    mfDisplay(z, "z", c, "mfConj(z)");
}
```

Result

```
z =

-2.0000 +2.0000i

 mfConj(z) =

-2.0000 -2.0000i
```

See Also

mfImag, mfReal

mfImag

Imaginary part of complex.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfImag(mfArray x);
```

Descriptions

Function `mfImag(z)` returns element-by-element the imaginary part of complex mfArray `z`.

Example

The example below gets the imaginary part of complex mfArray `z`.

Code

```
// ************************************************
// AnCAD example file
// file: mfImag.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray x, y;

    x = dcomplex(2, -2);
    y = mfImag(x);

    mfDisplay(x, "x", y, "mfImag(x)");

}
```

Result

```
x =

 2.0000 -2.0000i

 mfImag(x) =

 -2
```

See Also

mfReal, mfConj

mfReal

Real part of complex.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfReal(mfArray x);
```

Descriptions

Function `mfReal(z)` returns the elemental real part of complex mfArray `z`.

Example

The example below gets the real part of complex mfArray `z`.

Code

```
// ************************************************
// AnCAD example file
// file: mfReal.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray z, x;

    z = dcomplex(-2, 2);
    x = mfReal(z);
    mfDisplay(z, "z", x, "mfReal(z)");
}
```

Result

```
z =

-2.0000 +2.0000i

 mfReal(z) =

 -2
```

See Also

mfImag, mfConj

Rounding and Remainder

mfCeil

Round towards positive infinity.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfCeil(mfArray x);
```

Descriptions

Function `mfCeil(x)` returns an mfArray containing elements of mfArray `x` rounded to the nearest integer `x` towards infinity. The real and imaginary parts of a complex number are rounded independently.

Example

The example below performs the `mfCeil` operation on mfArray `x`.

Code

```
// ************************************************
// AnCAD example file
// file: mfCeil.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y, u, v;

    x = mfV(-1.6, 2.3);
    u = dcomplex(3.2, 2.2);
    y = mfCeil(x);
    v = mfCeil(u);

    mfDisplay(x, "x", y, "mfCeil(x)");
    mfDisplay(u, "u", v, "mfCeil(u)");
}
```

Result

```
x =

  -1.6000   2.3000

 mfCeil(x) =

 -1  3

 u =

 3.2000 +2.2000i
```

```
 mfCeil(u) =

 4.0000 +3.0000i
```

See Also

mfFix, mfFloor, mfRound

mfFix

Round the elements towards zero.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfFix(mfArray x);
```

Descriptions

Function `mfFix(x)` rounds the elements of mfArray `x` towards zero, returning an mfArray of integers. The real and imaginary parts of a complex number are rounded towards the nearest integer independently.

Example

The example below performs the `mfFix` operation on mfArray `x`.

Code

```
// ************************************************
// AnCAD example file
// file: mfFix.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y, u, v;

    x = mfV(-1.6, 2.3);
    y = mfFix(x);
    u = dcomplex(3.2, 2.2);
    v = mfFix(u);

    mfDisplay(x, "x", y, "mfFix(x)");
    mfDisplay(u, "u", v, "mfFix(u)");

}
```

Result

```
x =

  -1.6000   2.3000

 mfFix(x) =

 -1  2

 u =
```

```
 3.2000 +2.2000i

 mfFix(u) =

 3.0000 +2.0000i
```

See Also

mfCeil, mfFloor, mfRound

mfFloor

Round towards minus infinity.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfFloor(mfArray x);
```

Descriptions

Function `mfFloor(x)` rounds the elements of mfArray `x` towards nearest integer less than or equal to `x` and towards minus infinity. The real part and imaginary part of a complex number are rounded towards the nearest integer independently.

Example

The example below performs the `mfFloor` operation on mfArray `x`.

Code

```
// ************************************************
// AnCAD example file
// file: mfFloor.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y, u, v;

    x = mfV(-1.6, 2.3);
    u = dcomplex(3.2, 2.2);
    y = mfFloor(x);
    v = mfFloor(u);

    mfDisplay(x, "x", y, "mfFloor(x)");
    mfDisplay(u, "u", v, "mfFloor(u)");

}
```

Result

```
x =

  -1.6000   2.3000

 mfFloor(x) =

 -2  2

 u =

 3.2000 +2.2000i
```

```
mfFloor(u) =

3.0000 +2.0000i
```

See Also

mfCeil, mfFix, mfRound

mfMod

Return modulus (signed remainder after division).

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfMod(mfArray x, mfArray y);
```

Descriptions

Function `mfMod(x, y)` returns an mfArray containing element-by-element the signed remainder of `x`, `y` division.
The function uses the following algorithm:

$y \neq 0$, mfMod(x, y) = x - y*mfFloor(x/y)
$y = 0$, mfMod(x, y) = x

Note that :

- The shape of the input arguments `x` and `y` must conform.
- `mfMod(x, y)` always differ from `x` by a multiple of `y`.
- `mfMod(x, y)` has the same sign as `y` while `mfRem(x,y)` has the same sign as `x`.
- `mfMod(x, y)` and `mfRem(x, y)` are equal if `x` and `y` are of the same sign. They differ if the sign of `x` and `y` are different.

Limitations

Arguments `x` and `y` should be integers. Due to the inexact representation of floating-point numbers on a computer, real (or complex) inputs may lead to unexpected results.

Example

The example below finds the modulus of `x` and `y`.

Code

```
// ************************************************
// AnCAD example file
// file: mfMod.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y, m;

    x = mfV(-5, 7, -15);
    y = mfV(2, -3, -4);
    m = mfMod(x, y);
```

```
    mfDisplay(x, "x", y, "y", m, "mfMod(x,y)");
}
```

Result

```
x =
  -5   7 -15
 y =
  2 -3 -4
 mfMod(x,y) =
  1 -2 -3
```

See Also

mfRem

mfRem

Remainder after division.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfRem(mfArray x, mfArray y);
```

Descriptions

Function `mfRem(x, y)` returns an mfArray containing the element-by-element remainder of `x/y` division. The result lies between 0 and `mfSign(x)*mfAbs(y)`. The input `x` and `y` are either scalars or conformable arrays.

Note that :

- `mfRem(x,y) = x - y*mfFix(x/y)` for `y ≠ 0`
- `mfFix(x,y)` is the integer of the quotient `x/y`.
- If `y` is zero, `mfRem` returns `MF_NAN`.

Limitations

Arguments `x` and `y` should be integers. Due to the inexact representation of floating-point numbers on a computer, real (or complex) inputs may lead to unexpected results.

Example

The example below finds the remainder of x/y.

Code

```
// ************************************************
// AnCAD example file
// file: mfRem.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y, r;

    x = mfV(-5, 7, -15);
    y = mfV( 2, -3, -4);

    r = mfRem(x,y);
    mfDisplay(x, "x", y, "y", r, "mfRem(x,y)");

}
```

Result

```
x =
  -5   7 -15
 y =
  2 -3 -4
 mfRem(x,y) =
 -1  1 -3
```

See Also

mfMod

mfRound

Round towards nearest integer.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfRound(mfArray x);
```

Descriptions

Function `mfRound(x)` rounds the elements of mfArray `x` to the nearest integer. The real part and imaginary part of a complex number is rounded towards the nearest integer independently.

Example

The example below performs the round operation on mfArray `x`.

Code

```
// ************************************************
// AnCAD example file
// file: mfRound.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y, u, v;

    x = mfV( -1.6, 2.3);
    u = dcomplex(3.2, 2.2);

    y = mfRound(x);
    v = mfRound(u);

    mfDisplay(x, "x",  y, "mfRound(x)");
    mfDisplay(u, "u",  v, "mfRound(u)");
}
```

Result

```
x =

  -1.6000   2.3000

 mfRound(x) =

 -2  2

 u =

 3.2000 +2.2000i
```

```
mfRound(u) =

3.0000 +2.0000i
```

See Also

mfCeil, mfFix, mfFloor

mfSign

Signum function.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfSign(mfArray x);
```

Descriptions

Function `mfSign(x)` returns an mfArray `y` containing the element-by-element information on the sign of each element of mfArray `x`.

Note that :
For elements `x > 0`, corresponding element of `y = 1`.
For elements `x = 0`, corresponding element of `y = 0`.
For elements `x < 0`, corresponding element of `y = -1`.

For nonzero elements of complex `x`, `mfSign(x) = x/mfAbs(x)`

Example

The example below finds the sign of elements of mfArray `x`.

Code

```
// ************************************************
// AnCAD example file
// file: mfSign.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;

    x = mfV(-5, 7, 0);
    y = mfSign(x);

    mfDisplay(x, "x", y, "mfSign(x)");

}
```

Result

```
x =

 -5  7  0

 mfSign(x) =

 -1  1  0
```

See Also

mfAbs, mfConj, mfImag, mfReal

CHAPTER 5

Elementary Matrix-manipulating Functions

It contains a set of elementary matrix manipulation functions. This chapter describes the functions available, as listed below:

Matrices	
`mfEye`	Identity matrix
`mfLinspace`	Constructs linearly spaced vectors.
`mfMagic`	Constructs magic matrix.
`msMeshgrid`	Constructs grids for solving functions of 2 variables.
`mfRepmat`	Replicate and tile an array
`mfOnes`	Arrays containing ones.
`mfRand`	Arrays containing random numbers.
`mfZeros`	Arrays containing zeros.
`mfSize`	Size of a mfArray.

Matrix Manipulation	
`mfDiag`	Diagonal matrices and diagonals of a matrix.
`mfFind`	Find indices and values of nonzero elements.
`mfReshape`	Change shape of an array.

`mfTril`	Extracts lower triangular of a mfArray.
`mfTriu`	Extracts upper triangular of a mfArray.
`mfLogical`	Converts numerical values to logical.

Matrices

mfEye

Construct identity matrix.

Module

`elmat.h/elmat.lib`

C++ Prototype

```
mfArray mfEye(int x1[, int x2]);
```

Descriptions

Function `mfEye` generates an identity matrix.

`a = mfEye(m)` returns an m-by-m identity matrix for scalar `m`. If mfArray `m` contains information about the shape of an array, `mfEye` returns mfArray whose shape is specified by `m`. For example, `a = mfEye(mf(Shape(b)))`.

`a = mfEye(m, n)` returns an m-by-n identity matrix.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfEye.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {
    mfArray a;
    a = mfEye(3,3);
    mfDisplay(a, "mfEye(3,3)");
}
```

Result

```
mfEye(3,3) =

  1  0  0
  0  1  0
  0  0  1
```

See Also

cml

mfLinSpace

Construct linearly spaced vector.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfLinspace(int x1, int x2[, int x3]);
```

Descriptions

Function `mfLinSpace` generates linearly spaced row vectors.

`a = mfLinSpace(l, u)` returns a row vector mfArray with 100 linearly and equally spaced points between `l` and `u`, where `l` is the initial value and `u` is the last value.

`a = mfLinSpace(l, u, n)` generates `n` linearly and equally spaced points between `l` and `u`.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfLinspace.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {
    mfArray a;
    a = mfLinspace(3, 4, 5) ;
    mfDisplay(a, "mfLinspace(3, 4, 5)");
}
```

Result

```
mfLinspace(3, 4, 5) =

   3.0000   3.2500   3.5000   3.7500   4.0000
```

See Also

mfColon, mfMeshgrid

mfMagic

Construct magic square.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfMagic(int x);
```

Descriptions

Function `mfMagic` creates a magic square mfArray. A magic square is a special matrix with equal row, column and diagonal sum.

`a = mfMagic(m)` generates an m-by-m magic matrix constructed from the integers1 through `m**2`. This function produces valid magic squares for all `m > 0`, except for `m = 2`.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfMagic.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray a, rsum, csum;

    a = mfMagic(3);
    rsum = mfSum(a, 2);
    csum = mfSum(a, 1);

    mfDisplay(a, "mfMagic(3)", rsum, "row sum", csum, "column sum");

}
```

Result

```
mfMagic(3) =

  8  1  6
  3  5  7
  4  9  2

 row sum =

  15
  15
  15

 column sum =
```

```
  15  15  15
```

See Also

mfZeros, mfOnes

mfMeshgrid

Generate x and y matrices for three-dimensional plots.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfMeshgrid(mfArray x);
mfArray mfMeshgrid(mfArray x1, mfArray x2);
mfArray mfMeshgrid(mfArray x1, mfArray x2, mfArray x3);

void mfMeshgrid(mfOutArray outArray, mfArray x);
void mfMeshgrid(mfOutArray outArray, mfArray x1, mfArray x2);
void mfMeshgrid(mfOutArray outArray, mfArray x1, mfArray x2, mfArray x3);
```

Descriptions

Function `mfMeshgrid` generates grids from vectors for solving functions of two variables and plotting three-dimensional graphs.

`x = mfMeshgrid(m)` returns `x` is equivalent to `mfMeshgrid(mfOut(a, b), m)` returns output `a`.

`x = mfMeshgrid(m, n)` returns `x` is equivalent to `mfMeshgrid(mfOut(a, b), m, n)` returns output `a`.

`x = mfMeshgrid(m, n, k)` returns `x` is equivalent to mfMeshgrid(mfOut(a, b, c), m, n, k) returns output `a`.

`mfMeshgrid(mfOut(a, b), m, n)` transforms the domain specified by vectors `m` and `n` into matrix mfArrays `a` and `b`. The rows of the output matrix `a` are copies of the vector `m` and the columns of the output matrix `b` are copies of the vector `n`.

`mfMeshgrid(mfOut(a, b), m)` is an abbreviation for `mfMeshgrid(mfOut(a, b), m, m)`.

`mfMeshgrid(mfOut(a, b, c), m, n, k)` returns a three-dimensional arrays that can be used to evaluate functions of three variables and three-dimensional volumetric plots.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfMeshgrid.cpp
//
// Copyright 2003 AnCAD
```

```
// ************************************************

#include "cml.h"


void main() {

    mfArray x, y, a, b;

    // Generate vectors a and b using the colon function.
    a = mfColon(-1.0, 0.5, 1.0);
    b = mfColon(-1.5, 0.3, 1.5);

    // Use the meshgrid function to transform the domain
    // specified by vectors a and b into a two-dimensional function domain.
    mfMeshgrid(mfOut(x, y), a, b);

    // Display the generated matrices.
    mfDisplay(x, "x", y, "y");

}
```

Result

```
x =


  -1.0000  -0.5000   0.0000   0.5000   1.0000
  -1.0000  -0.5000   0.0000   0.5000   1.0000
  -1.0000  -0.5000   0.0000   0.5000   1.0000
  -1.0000  -0.5000   0.0000   0.5000   1.0000
  -1.0000  -0.5000   0.0000   0.5000   1.0000
  -1.0000  -0.5000   0.0000   0.5000   1.0000
  -1.0000  -0.5000   0.0000   0.5000   1.0000
  -1.0000  -0.5000   0.0000   0.5000   1.0000
  -1.0000  -0.5000   0.0000   0.5000   1.0000
  -1.0000  -0.5000   0.0000   0.5000   1.0000


 y =


  -1.5000  -1.5000  -1.5000  -1.5000  -1.5000
  -1.2000  -1.2000  -1.2000  -1.2000  -1.2000
  -0.9000  -0.9000  -0.9000  -0.9000  -0.9000
  -0.6000  -0.6000  -0.6000  -0.6000  -0.6000
  -0.3000  -0.3000  -0.3000  -0.3000  -0.3000
   0.0000   0.0000   0.0000   0.0000   0.0000
   0.3000   0.3000   0.3000   0.3000   0.3000
   0.6000   0.6000   0.6000   0.6000   0.6000
   0.9000   0.9000   0.9000   0.9000   0.9000
   1.2000   1.2000   1.2000   1.2000   1.2000
   1.5000   1.5000   1.5000   1.5000   1.5000
```

See Also

mfLinSpace

mfOnes

Construct a matrix of ones.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfOnes(int x1[,int x2, int x3, int x4, int x5, int x6, int x7]);
```

Descriptions

Function `mfOnes` generates a matrix containing ones.

`a = mfOnes(m)` returns a m-by-m matrix of ones.

`a = mfOnes(m, n)` returns a m-by-n matrix of ones.

`a = mfOnes(m, n, d3, d4, d5, d6, d7)` returns an m-by-n-by-d3-by-d4-by-d5-by-d6-by-d7 array of ones.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfOnes.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {
    mfArray a;
    a = mfOnes(3, 2);
    mfDisplay(a, "mfOnes(3, 2)");
}
```

Result

```
mfOnes(3, 2) =

  1  1
  1  1
  1  1
```

See Also

mfZeros, mfEye

mfRand

Uniformly distribute random numbers.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfRand(int x1[, int x2, int x3, int x4, int x5, int x6, int x7]);
```

Descriptions

Function `mfRand` randomly generates an mfArray.

`a = mfRand(m)` returns a m-by-m matrix with random entries chosen from a uniform distribution in the interval (0,1).

`a = mfRand(m, n)` generates a m-by-n matrix with random entries chosen from a uniform distribution in the interval (0,1).

`a = mfRand(m, n, d3, d4, d5, d6, d7)` generates a m-by-n-by-d3-byd4-by-d5-by-d6-by-d7 matrix with random entries chosen from a uniform distribution in the interval (0,1).

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfRand.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {
    mfArray a;
    a = mfRand(3, 2);
    mfDisplay(a, "mfRand(3, 2)");
}
```

Result

```
mfRand(3, 2) =

   0.9267   0.8342
   0.6032   0.7617
   0.1321   0.6148
```

See Also

mfZeros, mfOnes, mfMagic

mfRepmat

Replicate and tile an array.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfRepmat(mfArray x, int m[, int n]);
void mfRepmat(mfOutArray outArray, mfArray x, int m[, int n]);
```

Descriptions

Function `mfRepmat` generates matrix mfArrays by replicating copies of an array into a larger block array.

`x = mfRepmat(a, m, n)`, `mfRepmat(mfOut(x), a, m, n)` generate an mfArray `x` consisting of m-by-n tiling of copies of vector mfArray a.

`x = mfRepmat(a, p)`, `mfRepmat(mfOut(x), a, p)` generate an mfArray `x` containing `p` block copies of mfArray `a`. Vector `p` contains information about the number of blocks mfArray `x` in each dimension. The information is in the form of [m, n], or [m, n, d3, ..., d7].

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfRepmat.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray  a;

    // Create a 3-by-2 mfArray consisting of ones.
    a = mfRepmat(1, 3, 2);
    // This is similar to mfOnes(3,2) but is much faster.
    mfDisplay (a, " mfRepmat(1, 3, 2)");

    // Create  3-by-2 block copies of mfMagic(3)
    a = mfRepmat(mfMagic(2), 3,2);
    mfDisplay (a, "mfRepmat(mfMagic(2), 3,2)");

}
```

Result

```
mfRepmat(1, 3, 2) =

  1  1
```

```
 1  1
 1  1

mfRepmat(mfMagic(2), 3,2) =

 1  3  1  3
 4  2  4  2
 1  3  1  3
 4  2  4  2
 1  3  1  3
 4  2  4  2
```

See Also

mfMeshgrid

mfSize

Show the total number of elements in an array.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfSize(mfArray x[, int y]);
void mfSize(mfOutArray outArray, mfArray x[, int y]);
```

Descriptions

Function `mfSize` returns the total number of elements in an array.

`s = mfSize(a)` returns the number of elements in `a`.

`b = mfSize(a, IDIM)` returns the length of the dimension specified by the scalar `IDIM`.

`mfSize(mfOut(m1, m2, ..., mn), a)` returns the lengths of the first `n` dimensions of `a`.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfSize.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray a;
    int s;

    a = mfRand(3);

    s = mfSize(a);
    mfDisplay(a, "mfRand(3)", mf(s), "mfSize(a)");


    s = mfSize(a, 1);
    mfDisplay(mf(s), "mfSize(a, 1)");

    s = mfSize(a, 2);
    mfDisplay(mf(s), "mfSize(a, 2)");

}
```

Result

```
mfRand(3) =
```

```
  0.7850   0.9132   0.6183
  0.0866   0.4634   0.6053
  0.9386   0.0296   0.4254

mfSize(a) =

 9

mfSize(a, 1) =

 3

mfSize(a, 2) =

 3
```

See Also

mfArray::Shape

mfZeros

Show matrix with zeros.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfZeros(int m[, int n, int d3, int d4, int d5, int d6, int d7]);
```

Descriptions

Function `mfZeros` generates the mfArrays of zeros. It used to allocate memory for mfArrays.

`a = mfZeros(m)` returns a m-by-m mfArray of zeros.

`a = mfZeros(m, n)` returns a m-by-n mfArray of zeros.

`a = mfZeros(m, n, d3, ..., d7)` returns a m-by-n-by-d3-by-...-by-d7 array of zeros.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfZeros.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {
    mfArray a;
    a = mfZeros(2, 2);
    mfDisplay(a, "mfZeros(2, 2)");
}
```

Result

```
mfZeros(2, 2) =

  0  0
  0  0
```

See Also

mfOnes, mfEye

Matrix Manipulation

mfDiag

Request diagonals of a matrix.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfDiag(mfArray x1[, mfArray x2]);
```

Descriptions

Function `mfDiag(a)` returns a vector mfArray `d`, containing the elements extracted from the main diagonal of matrix mfArray `a`.

`a = mfDiag(d)` returns a diagonal matrix mfArray `a`, with its main diagonal composed of member elements of vector `d`.

`d = mfDiag(a, k)` returns a vector mfArray `d`, containing the elements extracted from the kth diagonal of matrix mfArray `a`.

`a = mfDiag(d, k)` returns a diagonal matrix `a` of order mfLength(d) + mfAbs(k) whose kth diagonal is composed of elements from vector mfArray `d`. k = 0 represents the main diagonal, k > 0 is above the main diagonal, and k < 0 is below the main diagonal.

Figure 4.8: The kth diagonal of a matrix.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfDiag.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray a, d, b;

    // Construct an mfArray using the magic function.
    a = mfMagic(3);

    // Extract the 1st diagonal of mfArray a.
    d = mfDiag(a, 1);

    // Construct an mfArray b, whose main diagonal is
    // composed of d.
    b = mfDiag(d);

    // Display the resulting mfArrays.
```

```
    mfDisplay(a, "mfMagic(3)", d, "k = 1 diagonal");
    mfDisplay(b, "mfDiag(d)");

}
```

Result

```
mfMagic(3) =

  8  1  6
  3  5  7
  4  9  2

 k = 1 diagonal =

  1
  7

 mfDiag(d) =

  1  0
  0  7
```

See Also

mfTriu, mfTril

mfFind

Find indices of nonzero elements.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfFind(mfArray x);
void mfFind(mfOutArray outArray, mfArray x);
```

Descriptions

Function `mfFind` generates indices of nonzero elements.

`y = mfFind(x)` returns an mfArray `y`, containing long column indices of nonzero entries in the mfArray `x`. If none is found, `mfFind` returns an empty matrix.

`mfFind(mfOut(i, j), x)` returns two mfArrays `i` and `j` containing the row and column indices of nonzero entries in matrix mfArray `x`.

`mfFind(mfOut(i, j, v), x)` returns three mfArrays `i`, `j`, and `v`, containing the row indices, column indices, and nonzero entries of matrix mfArray `x` respectively.

Example

The example below retrieves the indices of non-zero elements of a matrix mfArray.

Code

```
// ************************************************
// AnCAD example file
// file: mfFind.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray a, i, j;

    a = mfMagic(3) > 7;
    mfFind(mfOut(i,j), a);

    mfDisplay(a, "a", i, "i", j, "j");
}
```

Result

```
a =

  1  0  0
  0  0  0
  0  1  0
```

```
i =

 1
 3

j =

 1
 2
```

See Also

mfColon, relational_operators

mfLogical

Convert numeric values to logical.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfLogical(mfArray x);
```

Descriptions

Function `mfLogical(x)` returns an logical mfArray. An element of mfArray `l` is assigned logical "true" if the corresponding element in mfArray `x` is nonzero, otherwise it is assigned "false".

Notice that most arithmetic operations remove the logical characteristic from an array. For example, adding zero to a logical array.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfLogical.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray l, x, y, z;

    x = mfEye(3);
    l = mfLogical(x);
    y = mfVCat(
        mfV(1.0, 2.0, 3.0),
        mfV(4.0, 5.0, 6.0),
        mfV(7.0, 8.0, 9.0));
    z = mfS(y, l);

    mfDisplay(y, "y",z, "z");

}
```

Result

```
y =

  1  2  3
  4  5  6
  7  8  9

 z =
```

```
1
5
9
```

See Also

mfZeros, mfOnes, mfMagic

mfReshape

Change size of a matrix.

Module

cml.h/cml.lib

C++ Prototype

```
mfArray mfReshape(mfArray x, int m[, int n, int d3, int d4, int d5, int d6, int d7]);
```

Descriptions

Function mfReshape reshapes elements.

y = mfReshape(x, m, n) returns the m-by-n matrix mfArray y whose elements are taken column wise from mfArray x. An error occurs if x is not an m-by-n matrix.

a = mfReshape(b, m, n, d3, ..., d7) returns the m-by-n-by-d3-by-d4-by-d5-by-d6-by-d7 matrix mfArray a whose elements are taken column wise from mfArray b. An error occurs if b is not an m-by-n-by-d3-by-d4-by-d5-by-d6-by-d7 matrix.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfReshape.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;

    x = mfColon(1, 6);
    y = mfReshape(x, 3, 2);
    mfDisplay(x, "x", y, "mfReshape(x, 3, 2)");

}
```

Result

```
x =

  1  2  3  4  5  6

 mfReshape(x, 3, 2) =

  1  4
  2  5
  3  6
```

See Also

mfSize, mfArray::Shape

mfTril

Lower triangular part of a matrix.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfTril(mfArray x[, int k]);
```

Descriptions

Function `mfTril` generates the lower triangular part of a matrix

`l = mfTril(a)` returns the mfArray `l` containing the elements on and below the main diagonal of mfArray `a`.

`l = mfTril(a,k)` returns the mfArray `l` containing the elements on and below the kth diagonal of mfArray `a`. k = 0 is the main diagonal, k > 0 is above the main diagonal, and k < 0 is below the main diagonal.

Figure 4.12: The kth diagonal of a matrix.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfTril.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray a, l, l1;

    // Construct an mfArray using the magic function.
    a = mfMagic(3);

    // Get the lower triangular of mfArray a from the main
    // diagonal downwards.
    l = mfTril(a);

    // Get the lower triangular a from the 1sdiagonal downwards.
    l1 = mfTril(a, 1);

    // Display the resulting mfArrays.
    mfDisplay(a, "mfmagic(3)", l, "lower triangular");
    mfDisplay(l1, "lower triangular from k = 1");

}
```

Result

```
mfmagic(3) =

  8  1  6
  3  5  7
  4  9  2

 lower triangular =

  8  0  0
  3  5  0
  4  9  2

 lower triangular from k = 1 =

  8  1  0
  3  5  7
  4  9  2
```

See Also

mfTriu, mfDiag

mfTriu

Upper triangular part of a matrix.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfTriu(mfArray x1[, mfArray x2]);
void mfTriu(mfOutArray outArray, mfArray x1[, mfArray x2]);
```

Descriptions

Function `mfTriu` generates the upper triangular part of a matrix.

`u = mfTriu(a)` returns an mfArray `u` containing the elements on and above the main diagonal of mfArray `a`.

`u = mfTriu(a, k)` returns an mfArray `u` containing the elements on and above the kth diagonal of mfArray `a`. k = 0 is the main diagonal, k > 0 is above the main diagonal, and k < 0 is below the main diagonal.

Figure 4.13 The kth diagonal of a matrix.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfTriu.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray a, u, u1;

    // Construct an mfArray using the magic function.
    a = mfMagic(3);

    // Extract the upper triangular of mfArray a from the main
    // diagonal upwards.
    u = mfTriu(a);

    // Extract the upper triangular a from the 1sdiagonal upwards.
    u1 = mfTriu(a, 1);

    // Display the resulting mfArrays.
    mfDisplay(a, "mfMagic(3)", u, "upper triangular");
    mfDisplay(u1, "upper triangular from k = 1");

}
```

Result

```
mfMagic(3) =

  8  1  6
  3  5  7
  4  9  2

 upper triangular =

  8  1  6
  0  5  7
  0  0  2

 upper triangular from k = 1 =

  0  1  6
  0  0  7
  0  0  0
```

See Also

mfTril, mfDiag

CHAPTER 6

Matrix Functions

It contains a set of matrix functions for solving numerical algebra problems.

This chapter describes the functions available, as listed below:

Matrix Analysis

mfDet

mfNorm

mfRank

mfTrace

Linear Equations

mfChol

mfCond

mfInv

mfRcond

mfLu

mfQr

Eigenvalues and singular values

```
mfEig

mfHess

mfQz

mfSchur

mfSvd
```

Factorization Utilities

```
mfBalance
```

Matrix Analysis

mfDet

Matrix determinant.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfDet(mfArray x);
```

Descriptions

Function `mfDet(x)` returns a scalar mfArray containing the determinant of square matrix mfArray `x`. For matrices of modest order with small integer entries, it can be used as a test for matrix singularity.

Example

The following example uses the `mfDet` function to compute the determinant of a non-singular square matrix mfArray `x`.

Code

```
// ************************************************
// AnCAD example file
// file: mfDet.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, d;

    // Construct a 3-by-3 mfArray x using vertical concatenation.
    x = mfVCat(
        mfV(1.0, 2.0, 3.0),
        mfV(7.0, 8.0, 9.0),
        mfV(1.0, 2.0, 4.0));

    // Compute determinant of mfArray x
    d = mfDet(x);

    // Display value of x and determinant of x
    mfDisplay(x, "x", d, "mfDet(x)");

}
```

Result

```
x =

  1  2  3
  7  8  9
  1  2  4

 mfDet(x) =
```

- 6

See Also

mfCond, mfInv, mfLu

mfNorm

Matrix or vector norm.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfNorm(mfArray x[, mfArray p]);
```

Descriptions

`n = mfNorm(x), n = mfNorm(x, p)`
Function `mfNorm` generates norm value differently for matrices and vectors.

For matrices:

- `mfNorm(x)` returns a scalar mfArray containing the largest singular value of mfArray `x`.
- `mfNorm(x, p)` returns a different kind of norm, depending on the value of `p`.
- `mfNorm(x, 1)` returns the largest column sum of `x`.
- `mfNorm(x, 2)` is equiv. to `mfNorm(x)`. It returns the largest singular value of mfArray `x`.
- `mfNorm(x, MF_INF)` returns the largest row sum of `x`, which is also the infinity norm of `x`.
- `mfNorm(x, 'fro')` returns the Frobenius norm.

For vectors:

- `mfNorm(v, 1)` returns a scalar mfArray equal to the sum of elements of mfArray `v`.
- `mfNorm(v)` is the same as `mfNorm(v, 2)`, and returns the value of `mfSum(mfAbs(v).^2)^(1/2)`.

Example

The following example uses the `mfNorm` function to compute the norm of a non-singular square matrix mfArray `x`.

Code

```
// ************************************************
// AnCAD example file
// file: mfNorm.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {
    mfArray x, n;
```

```
    // Construct a 3-by-3 mfArray x
    x = mfReshape(mfV(
        1.0, 7.0, 1.0,
        2.0, 8.0, 2.0,
        3.0, 9.0, 4.0), 3, 3);

    // Compute norm of mfArray x
    n = mfNorm(x);

    // Display value of x and norm n of x
    mfDisplay(x, 'x', n, 'norm');

}
```

Result

```
=

  1  2  3
  7  8  9
  1  2  4

  =


   15.0130
```

See Also

mfCond

mfRank

Estimate for the number of linearly independent rows or columns.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfRank(mfArray x[, mfArray tol]);
```

Descriptions

For matrices, function `mfRank(x)` is used as an estimation for the number of linearly independent rows and columns of a matrix `x`.

`mfRank(x), mfRank(x[, tol])`

- Function `mfRank(x)` returns a scalar mfArray containing the number of independent rows or columns of a matrix `x`.
- Function `mfRank(x)` uses the default `tol = mfMax(mfSize(x)) * mfNorm(x) * MF_EPS`.
- Function `mfRank(x, tol)` returns the singular values of matrix `x` that are larger than `tol`.

Example

The following example uses the `mfRank` function to compute the rank of a square matrix mfArray `x`.

Code

```
// ************************************************
// AnCAD example file
// file: mfRank.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, r;

    // Construct a 3-by-3 mfArray x using vertical
    // concatenation.
    x = mfVCat(
        mfV(1.0, 2.0, 3.0),
        mfV(7.0, 8.0, 9.0),
        mfV(2.0, 4.0, 6.0));

    // Compute the rank of mfArray x
    r = mfRank(x);

    // Display value of x and the rank r of x
```

```
    mfDisplay(x, "x", r, "mfRank(x)");
}
```

Result

```
x =

  1  2  3
  7  8  9
  2  4  6

 mfRank(x) =

  2
```

See Also

mfSize

mfTrace

Return the sum of diagonal elements.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfTrace(mfArray x);
```

Descriptions

For matrices, function `mfTrace(x)` returns the sum of diagonal elements of mfArray `x`, which is equivalent to the sum of eigenvalues of mfArray `x`.

Example

The following example uses the `mfTrace` function to compute the sum of the diagonal elements of the square matrix mfArray `x`.

Code

```
// ************************************************
// AnCAD example file
// file: mfTrace.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, s;

    // Construct a 3-by-3 mfArray x using vertical
    // concatenation.
    x = mfVCat(
        mfV(10.0, 0.0, 6.0),
        mfV(0.0, -3.0, 9.0),
        mfV(0.0, 2.0, 4.0));

    // Compute the trace of mfArray x
    s = mfTrace(x);

    // Display value of x and trace s of x
    mfDisplay(x, "x", s, "mfTrace(x)");

}
```

Result

```
x =

  10   0   6
   0  -3   9
   0   2   4

 mfTrace(x) =
```

```
11
```

Linear Equations

mfChol

Cholesky factorization.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfChol(mfArray x);
void mfChol(mfOutArray outArray, mfArray x);
```

Descriptions

Function `mfChol(x)` uses only the diagonal and upper triangle of `x`.

`r = mfChol(x)`

For matrices, if mfArray `x` is positively definite, then Function `mfChol(x)` returns an upper triangular mfArray `r` so that `.h.r * r = x`. If `x` is not a positive definite mfArray, an error is occurred.

`mfChol(mfOut(r, p), x)`

Error is prevented from occurring when `mfChol(mfOut(r, p), x)` is used. If `x` is positively definite, `p` is 0 and `r` is the same as above. Otherwise, `p` is a positively scalar mfArray and `r` is an upper triangular mfArray such that:`.h.r*r = mfGet(x, mfColon(1, p-1),mfColon(1,p-1))`

Example

The following example uses the `mfChol` function to derive the Cholesky factorization of a non-positive square matrix mfArray `x`.

Code

```
// ************************************************
// AnCAD example file
// file: mfChol.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, r ,p;

    // Construct a 3-by-3 mfArray x using vertical concatenation.
    x = mfVCat(mfV(10.0, 2.0, 6.0), mfV(6.0, 3.0, 9.0), mfV(3.0, 2.0,
4.0));

    // Compute mfChol of mfArray x
    mfChol(mfOut(r, p), x);

    // Display value of x, r and p
    mfDisplay(x, "x", r, "r", p, "p");
```

```
}
```

Result

```
x =

  10   2   6
   6   3   9
   3   2   4

 r =


   3.1623   0.6325
   0.0000   1.6125

 p =

  3
```

See Also

mfLu

mfCond

Return condition number of a matrix.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfCond(mfArray x1[, mfArray p]);
```

Descriptions

For matrices, Function `mfCond` returns the p-norm condition number of `x`.

`c = mfCond(x)`, `c = mfCond(x, p)`

- Function `mfCond(x)` returns the 2-norm condition number. It is also the ratio of the largest singular value of `x` to the smallest. A large condition number indicates a nearly singular mfArray `x`.
- Specifying argument `p`, it returns a p-norm condition number of `x`, which is equal to `mfNorm(x, p) * mfNorm(mfInv(x), p)`, where `p` is `1, 2, MF_INF` or `'fro.'`

[p=1	The mfCond returns the 1-norm condition number
p=2	The mfCond returns the 2-norm condition number
p='fro'	Return the Frobenius norm condition number
p=MF_INF	Return the Infinity norm condition number]

Example

The following example uses the `mfCond` function to compute the condition number with respect to inversion in the 2-norm of a square matrix mfArray `x`.

Code

```
// ************************************************
// AnCAD example file
// file: mfCond.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, c;

    // Construct a 3-by-3 mfArray x using vertical concatenation.
    x = mfVCat(
        mfV(10.0, 8.0, 6.0),
        mfV(5.0, 15.0, 5.0),
        mfV(6.0, 7.0,  8.0));
```

```
    // Compute the condition number with respect to inversion
    // of mfArray x
    c = mfCond(x);

    // Display value of x and c of x
    mfDisplay(x, "x", c, "mfCond(x)");

}
```

Result

```
x =

  10   8   6
   5  15   5
   6   7   8

 mfCond(x) =

   8.2853
```

See Also

mfNorm

mfInv

Matrix inverse.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfInv(mfArray x);
```

Descriptions

For matrices, Function `mfInv(x)` returns the inverse of mfArray `x`.

`y = mfInv(x)`

Function `mfInv` is applicable only for square matrix `x`. A warning message will be printed when `x` is badly scaled or nearly singular.

Example

The following example uses the `mfInv` function to compute inverse of a matrix mfArray `x`.

Code

```
// ************************************************
// AnCAD example file
// file: mfInv.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, y;
    double temp[9] = { 5.0, 1.0, 2.0, 2.0, 10.0, 3.0 , 3.0, 2.0, 5.0 };

    // Construct a 3-by-3 mfArray x.
    x = mfArray(temp,3,3);

   // Compute the inverse of mfArray x
    y = mfInv(x);

   // Display value of x, y
    mfDisplay(x, "x", y, "y");
}
```

Result

```
x =

   5   2   3
   1  10   2
   2   3   5

 y =


   0.2635  -0.0060  -0.1557
```

```
  -0.0060   0.1138  -0.0419
  -0.1018  -0.0659   0.2874
```

See Also

mfCond

mfRcond

LINPACK reciprocal condition estimator.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfRcond(mfArray x);
```

Descriptions

For matrices, Function `mfRcond(x)` uses the LAPACK condition estimator to get an estimate for the reciprocal of the condition of `x` in the 1-norm.

`c = mfRcond(x)` returns a value near 1.0 when matrix `x` is well conditioned. And it returns a value near 0.0 when `x` is badly conditioned.

Example

The following example uses the `mfRcond` function to compute the reciprocal of the condition of x in the 1-norm of a square matrix mfArray `x`.

Code

```
// ************************************************
// AnCAD example file
// file: mfRcond.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x,c;
    double temp[9] = { 4.0, 7.0, 5.0, 5.0, 8.0, 2.0 , 6.0, 9.0, 4.0 };

    // Construct a 3-by-3 mfArrays x.
    x = mfArray(temp,3,3);

    // Compute the LINPACK reciprocal condition of mfArray x.
    c = mfRcond(x);

    // Display value of x,c
    mfDisplay(x, "x", c, "c");
}
```

Result

```
x =

  4  5  6
  7  8  9
  5  2  4

 c =
```

```
0.0139
```

See Also

mfCond, mfNorm

mfLu

LU factorization.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfLu(mfArray x);
void mfLu(mfOutArray outArray, mfArray x);
```

Descriptions

Function `mfLu(x)` returns LU decomposition of a square mfArray `x`.

`mfLu(mfout(l, u), x)`
When it is used, the function returns mfArray `u` containing the upper triangular matrix, and mfArray `l` containing product of the lower triangular matrix and permutation array, so that `x = l.Mul(u)`.

`mfLu(mfout(l, u, p), x)`
When it is used, the function returns an upper triangular mfArray `u`, lower triangular mfArray `l`, and permutation mfArray `p`, such that `p.Mul(x) = l.Mul(u).`
When `y = mfLu(x)` is used, the function returns the one output from LINPACK'S ZGEFA routine.

Example

The following example uses the `mfLu` function to compute the LU decomposition of matrix mfArray `x`.

Code

```
// ************************************************
// AnCAD example file
// file: mfLu.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray x, l, u, p;

    // Construct a 3-by-3 mfArray x using vertical concatenation.
    x = mfVCat(
        mfV(1.0, 2.0, 3.0),
        mfV(7.0, 8.0, 9.0),
        mfV(1.0, 2.0, 4.0));

    // Compute lu decomposition of mfArray x
    mfLu(mfOut(l, u, p), x);
```

```
    // Display value of x, l, u and p
    mfDisplay(x, "x", l, "l" ,u, "u", p, "p");

}
```

Result

```
x =

  1  2  3
  7  8  9
  1  2  4

 l =


   1.0000   0.0000   0.0000
   0.1429   1.0000   0.0000
   0.1429   1.0000   1.0000


 u =


   7.0000   8.0000   9.0000
   0.0000   0.8571   1.7143
   0.0000   0.0000   1.0000


 p =

  0  1  0
  1  0  0
  0  0  1
```

See Also

mfQr

mfQr

Orthogonal-triangular decomposition.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfQr(mfArray x1[, mfArray x2]);
void mfQr(mfOutArray outArray, mfArray x1[, mfArray x2]);
```

Descriptions

Function `mfQr` returns the orthogonal-triangular decomposition of a matrix.

`mfQr(mfOut(q, r), a)`
This function returns an upper triangular mfArray `r` of the same shape as `a` and a unitary mfArray matrix `q`, such that `a = q*r`.

`mfQr(mfOut(q, r), a, 0)`
This function performs an "economy size" decomposition. If `a` is a m-by-n mfArray with `m > n`, only the first `n` columns of `q` will be computed.

`mfQr(mfOut(q, r, e), a)`
This function returns an upper triangular mfArray `r`, a unitary mfArray `q`, and a permutation matrix mfArray `e`, so that `a*e = q*r`.

`mfQr(mfOut(q, r, e), a, 0)`
This function performs an "economy size" decomposition, returning a permutation vector `e`, so that `q*r = mfGet(a, MF_COL, e)`. The column permutation `e` is chosen so that `mfAbs(mfDiag(r))` is decreasing.

Example

The following example uses the `mfQr` function to compute the orthogonal -triangular decomposition of matrix mfArray `x`.

Code

```
// ************************************************
// AnCAD example file
// file: mfQr.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {
    mfArray q, r, a, e;
```

```
    // Construct a 3-by-3 mfArray a using vertical
    // concatenation.
    a = mfVCat(
        mfV(1.0, 2.0, 3.0),
        mfV(7.0, 8.0, 9.0),
        mfV(1.0, 2.0, 4.0));

    // Compute qr decomposition of mfArray a
    mfQr(mfOut(q, r, e), a);

    // Display value of a, q, r and e
    mfDisplay(a, "a", q, "q" ,r, "r", e, "e");

}
```

Result

```
a =

  1  2  3
  7  8  9
  1  2  4

 q =


  -0.2914   0.4491  -0.8447
  -0.8742  -0.4836   0.0445
  -0.3885   0.7513   0.5335


 r =


  -10.2956   -6.7990   -8.3531
    0.0000   -2.1849   -1.4681
    0.0000    0.0000   -0.2667


 e =

  0  1  0
  0  0  1
  1  0  0
```

See Also

mfLu

Eigenvalues and Singular Values

mfEig

Eigenvalues and eigenvectors.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfEig(mfArray x[, mfArray flag]);
mfArray mfEig(mfArray x, mfArray y[, mfArray flag]);
void mfEig(mfOutArray outArray, mfArray x[, mfArray flag]);
void mfEig(mfOutArray outArray, mfArray x, mfArray y[, mfArray flag]);
```

Descriptions

Function `mfEig` computes the eigenvalues and eigenvectors of a matrix mfArray `a`.

```
e = mfEig(a)
e = mfEig(a, flag)
```

- This function returns a vector `e` containing the eigenvalues of square mfArray `a`.
- Argument `flag` can be a string containing 'nobalance', so that the function can perform the computation with balancing switched off. This usually produces more accurate results for specific problems.

```
mfEig(mfOut(v, d), a)
mfEig(mfOut(v, d), a, flag)
```

- This function returns a diagonal matrix `d` of eigenvalues, and a full matrix mfArray `v` whose columns are the corresponding eigenvectors such that `a*v = v*d`.

```
e = mfEig(a, b)
e = mfEig(a, b, flag)
```

- This function returns a vector containing the generalized eigenvalues of square matrix mfArrays `a` and `b`.
- The function specifies the algorithm used to compute eigenvalues and eigenvectors through the argument `flag`, which is specified in More Detail below.

```
mfEig(mfOut(v, d), a, b)
mfEig(mfOut(v, d), a, b ,flag)
```

- This function returns a diagonal matrix mfArray `d` of generalized eigenvalues and a full matrix mfArray `v` whose columns are the corresponding eigenvectors so that `a*v = b*v*d`.

- The function specifies the algorithm used to compute eigenvalues and eigenvectors

through the argument `flag`, which is specified in More Detail below.

More Detail

Argument `flag` can be:

'chol'

This is the default for symmetric(Hermitian) `a` and symmetric(Hermitian) positive definite `b`. The generalized eigenvalues of `a` and `b` is computed by using Cholesky factorization of `b`.

'qz'

This uses the mfQz algorithm to compute eigenvlaues as for nonsymmetrical (non-Hermitian) `a` and `b`.

Example

The following example uses the `mfEig` function to compute the eigenvalues and eigenvectors of a square matrix mfArray `a`.

Code

```
// ************************************************
// AnCAD example file
// file: mfEig.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray v, d, a;

    // Construct a 3-by-3 mfArray a using vertical concatenation.
    a = mfVCat(
        mfV(5.0, 4.0, 3.0),
        mfV(2.0, 4.0, 6.0),
        mfV(10.0, 15.0, 25.0));

    // Compute eigenvalues and eigenvectors of mfArray a
    mfEig(mfOut(v, d), a);

    // Display value of a, v and d
    mfDisplay(a, 'a', v, 'v' ,d, 'd');

}
```

Result

```
=

   5   4   3
   2   4   6
  10  15  25

  =


  -0.1512  -0.9354   0.5446
  -0.2317   0.2120  -0.7954
  -0.9610   0.2830   0.2660

  =


   30.1904    0.0000    0.0000
    0.0000    3.1857    0.0000
```

```
    0.0000    0.0000    0.6238
```

See Also

mfBalance, mfHess, mfQz, mfSchur

mfHess

Hessenberg form.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfHess(mfArray x);
void mfHess(mfOutArray outArray, mfArray x);
```

Descriptions

Function `mfHess` returns the Hessenberg form of an mfArray.

`h = mfHess(a)`

This function returns an mfArray `h` with zeros below the first sub diagonal and has the same eigenvalues as `a`. If the original mfArray `a` is symmetric or Hermitian, `h` will be tridiagonal.

`mfHess(mfout(p, h), a)`

This function produces an unitary matrix `p` and a Hessenberg matrix `h` so that `a = p*h*.h.p` and `.h.p*p` is an identity matrix.

Example

The following example uses the `mfHess` function to compute the Hessenberg form of a square matrix mfArray `a`.

Code

```
// ************************************************
// AnCAD example file
// file: mfHess.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray a, p, h;

    // Construct a 3-by-3 mfArray a using vertical concatenation.
    a = mfVCat(
        mfV(5.0, 4.0, 3.0),
        mfV(9.0, 8.0, 7.0),
        mfV(5.0, 1.0, 2.0));

    // Compute the Hessenberg form of mfArray a
    mfHess(mfOut(p, h), a);

    // Display value of a, p and h
    mfDisplay(a, "a", p, "p" , h, "h");

}
```

Result

```
a =

  5  4  3
  9  8  7
  5  1  2

 p =


   1.0000   0.0000   0.0000
   0.0000  -0.8742  -0.4856
   0.0000  -0.4856   0.8742

 h =


    5.0000   -4.9536    0.6799
  -10.2956    9.9811   -2.5660
    0.0000    3.4340    0.0189
```

See Also

mfQz

QZ factorization for generalized eigenvalues.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfQz(mfArray x1, mfArray x2[, flag]);
void mfQz(mfOutArray outArray, mfArray x1, mfArray x2[, flag]);
```

Descriptions

Function `mfQz` performs qz factorization for generalized eigenvalues of square mfArrays.

`mfQz(mfOut(aa, bb, q, z, v), a, b)`

This function returns upper triangular mfArrays `aa` and `bb`, the left and right transformed mfArrays `q` and `z`, and the generalized eigenvector mfArray `v`, such that `q*a*z = aa`, and `q*b*z = bb`.

`mfQz(mfOut(aa, bb, q, z, v), a, b[, flag])`

This function depends on the value of flag:

'complex' produces a possibly complex decomposition with a triangular `aa`. This is the default option.

'real' produces a real decomposition with a quasitriangular `aa`, containing 1-by-1 and 2-by-2 blocks on its diagonal.

Example

The following example uses the `mfQz` function to compute the qz factorization of square matrices in mfArray `a` and `b`.

Code

```
// ************************************************
// AnCAD example file
// file: mfQz.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray aa, bb, q, z, v, a, b;

    // Construct a 3-by-3 mfArray a and b using vertical
    // concatenation.
    a = mfVCat(
            mfV(4.0, 2.0, 3.0),
            mfV(7.0, 8.0, 9.0),
            mfV(5.0, 2.0, 4.0));

    b = mfVCat(
```

```
            mfV(3.0, 5.0, 7.0),
            mfV(2.0, 1.0, 1.0),
            mfV(3.0, 2.0, 5.0));

    // Compute qz factorization of mfArray a and b
    mfQz(mfOut(aa, bb, q, z, v ), a, b);

    // Display values of aa, bb, q, z and v
    mfDisplay(aa, "aa", bb, "bb" ,q, "q", z, "z", v, "v");

}
```

Result

```
aa =

column   1  to   3

  6.3661  +0.0000i   -9.8642  +0.0000i   10.4110  +0.0000i
  0.0000  +0.0000i   -1.4632  +0.0000i    4.2403  +0.0000i
  0.0000  +0.0000i    0.0000  +0.0000i    1.2882  +0.0000i


 bb =


 2.3112 +0.0000i  -6.1276 +0.0000i   6.0482 +0.0000i
 0.0000 +0.0000i   4.3735 +0.0000i  -4.9870 +0.0000i
 0.0000 +0.0000i   0.0000 +0.0000i   1.8797 +0.0000i


 q =


-0.4555 +0.0000i  -0.7072 +0.0000i  -0.5407 +0.0000i
 0.6410 +0.0000i  -0.6820 +0.0000i   0.3521 +0.0000i
-0.6178 +0.0000i  -0.1862 +0.0000i   0.7640 +0.0000i


 z =


-0.9116 +0.0000i   0.4095 +0.0000i   0.0353 +0.0000i
-0.1805 +0.0000i  -0.3219 +0.0000i  -0.9294 +0.0000i
 0.3693 +0.0000i   0.8536 +0.0000i  -0.3674 +0.0000i


 v =


-1.0000 +0.0000i  -0.7564 +0.0000i   0.0489 +0.0000i
-0.1980 +0.0000i  -0.4240 +0.0000i  -1.0000 +0.0000i
 0.4051 +0.0000i   1.0000 +0.0000i   0.8466 +0.0000i
```

See Also

mfEig

mfSchur

Perform Schur decomposition.

Module

cml.h/cml.lib

C++ Prototype

```
mfArray mfBalance(mfArray x[, mfArray flag]);
void mfSchur(mfOutArray outArray, mfArray x[, mfArray flag]);
```

Descriptions

Function `mfSchur` performs the Schur decomposition of a square matrix mfArray.

```
qt = mfSchur(a)
qt = mfSchur(a, flag)
```

- This function returns a quasi-triangular Schur matrix mfArray `qt`. If `a` is complex, `mfSchur` returns the complex Schur form in matrix `qt`. The complex Schur form is an upper triangular matrix containing the eigenvalues of `a` on the diagonal.
- This function returns a Schur matrix `qt` in one of two forms for real matrix `a`, depending on the value of flag:
- If `flag` is 'complex', `qt` is triangular. If `a` has complex eigenvalues, `qt` is complex. If `flag` is 'real', `qt` has real eigenvalues on the diagonal. Complex eigenvalues are located in 2-by-2 blocks on the diagonal. By default, `flag` is 'real'.

```
mfSchur(mfOut(u, qt), a)
```

- This function returns a unitary matrix `u` and a quasi-triangular Schur matrix mfArray `qt` such that `a = u*qt*.h.u` and `.h.u*u` is an identity matrix given by `mfEye(Shape(u))`.

Example

The following example uses the `mfSchur` function to compute the Schur decomposition of a square matrix mfArray `a`.

Code

```
// ************************************************
// AnCAD example file
// file: mfSchur.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {
```

```
    mfArray u,  qt,  a;

    // Construct a 3-by-3 mfArray a using vertical
    // concatenation.
    a = mfVCat(
        mfV(3.0, 4.0, 3.0),
        mfV(2.0, 1.0, 4.0),
        mfV(6.0, 2.0, 2.0));

        // Compute the Schur decomposition of mfArray a
    mfSchur(mfOut(u, qt), a);

    // Displays values of a, u and qt
    mfDisplay(a, "a", u, "u" , qt, "qt");

}
```

Result

```
a =

  3  4  3
  2  1  4
  6  2  2

 u =


   0.6118   0.7878  -0.0707
   0.4634  -0.4295  -0.7751
   0.6410  -0.4415   0.6279


 qt =


   9.1729   1.2110  -0.5957
   0.0000  -1.5865  -1.4934
   0.0000   2.4025  -1.5865
```

See Also

mfSvd

Perform singular value decomposition.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfSvd(mfArray x1[, 0]);
void mfSvd(mfOutArray outArray, mfArray x1[, 0]);
```

Descriptions

Function `mfSvd` performs the singular value decomposition on matrix mfArray `a`.

`s = mfSvd(a)` returns a vector in mfArray, containing singular values.

`mfSvd(mfout(u, s, v), a)` returns two unitary matrices `u` and `v`, and a diagonal matrix `s` such that `a = u*s*.h.v`. Diagonal matrix `s` has the same shape as `a` and contains nonnegative elements in decreasing order.

`mfSvd(mfout(u, s, v), a, 0)` returns the "economy size" decomposition. If `a` is m-by-n and m > n, then only the first `n` columns of `u` are computed. `s` is n-by-n.

Example

The following example uses the `mfSvd` function to compute the singular value decomposition of a square matrix mfArray `a`.

Code

```
// ************************************************
// AnCAD example file
// file: mfSvd.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray a, u, s, v;

    // Construct a 3-by-3 mfArray a using vertical
    // concatenation.
    a = mfVCat(
        mfV(10.0, 20.0, 30.0),
        mfV(7.0, 8.0, 9.0),
        mfV(5.0, 15.0, 25.0));

    // Compute singular value decomposition of mfArray a
    mfSvd(mfOut(u, s, v), a);

    // Display value of a, u, s and v
    mfDisplay(a, "a", u, "u" ,s, "s", v, "v");
```

```
}
```

Result

```
a =

  10  20  30
   7   8   9
   5  15  25

 u =


  -0.7568  -0.1344  -0.6397
  -0.2688  -0.8280   0.4921
  -0.5958   0.5444   0.5905

 s =


   49.4333    0.0000    0.0000
    0.0000    5.0350    0.0000
    0.0000    0.0000    0.0000

 v =


  -0.2514  -0.8776   0.4082
  -0.5305  -0.2279  -0.8165
  -0.8095   0.4219   0.4082
```

See Also

Factorization Utilities

mfBalance

Perform diagonal scaling to improve eigenvalue accuracy.

Module

`cml.h/cml.lib`

C++ Prototype

```
mfArray mfBalance(mfArray x);
void mfBalance(mfOutArray outArray, mfArray x);
```

Descriptions

For matrices, function `mfBalance` performs diagonal scaling to improve the eigenvalue accuracy.

`b = mfBalance(a)` returns the balanced matrix mfArray `b`.

`mfBalance(mfOut(t, b), a)` returns a similarity transformation `t`, such that `b = mfLDiv(t, a*t)` has as closely as possible, approximately equal row and column norms.

Example

The following example uses the `mfBalance` function to find the balanced matrix mfArray `b` of the original mfArray `a`.

Code

```
// ************************************************
// AnCAD example file
// file: mfBalance.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"

void main() {

    mfArray a, b;

    // Construct a 3-by-3 mfArray a using vertical concatenation.
    a = mfVCat(mfV(100.0, 200.0, 300.0), mfV(4.0, 5.0, 6.0), mfV(7.0, 8.0,
9.0));

    // Compute the balanced matrix b of mfArray a
    b = mfBalance(a);

    // Display values of a and b
    mfDisplay(a, "a", b, "mfBalance(a)");
}
```

Result

```
a =

  100  200  300
```

```
   4    5    6
   7    8    9

mfBalance(a) =

  100.0000   25.0000   37.5000
   32.0000    5.0000    6.0000
   56.0000    8.0000    9.0000
```

See Also

mfEig

CHAPTER 7

MATFOR Visualization Routines

Windows Frame and Figure

Figure

`msFigure`	Create figure in Graphics Viewer.
`msCloseFigure`	Close figure in Graphics Viewer.
`mfFigureCount`	Number of figures in graphics viewer.

Window Frame

`mfWindowCaption`	Graphics Viewer title.
`mfWindowSize`	Graphics Viewer frame size.
`mfWindowPos`	Graphics Viewer frame position.

Display

`msGDisplay`	Display mfArray data on a MATFOR Data Viewer.
`msDrawNow`	Draw all pending graphs in current figure.
`msViewPause`	Pause program execution.

Recording

`mfRecordStart`	Record animation as avi file or MATFOR mfa file.
`mfRecordEnd`	Stop recording animation.

`mfExportImage`	Save figure graph as picture file.

Subplot

Plot Creation and Control

`mfSubplot`	Create subplot in active figure.
`msClearSubplot`	Remove all draws in specified subplot.
`msHold`	Hold previous graph on plot space.
`mfIsHold`	Return status of plot space.

Plot Annotation and Appearance

`mfTitle`	Graph title.
`mfXLabel`	X-axis label.
`mfYLabel`	Y-axis label.
`mfZLabel`	Z-axis label.
`mfText`	2-D text annotation
`mfAnnotation`	3-D text annotation
`msShading`	Shading methodology of surface object.
`msColorbar`	Display color scale.
`msColormap`	Colormap type.
`mfColormapRange`	Range of colormap.
`mfBackgroundColor`	Background color of plot space.

Axis Control

`mfAxis`	Manipulate axis object.
`msAxisWall`	Manipulate the axis wall object.
`msAxisGrid`	Display grid lines.

Object & Camera

Object Manipulation

`msObjRotateX`	Rotate draw object in degrees about the x-axis using the right hand rule.

`msObjRotateY`	Rotate draw object in degrees about the y-axis using the right hand rule.
`msObjRotateZ`	Rotate draw object in degrees about the z-axis using the right hand rule.
`msObjRotateWXYZ`	Rotate draw object in degrees about an arbitrary axis.
`mfObjScale`	Scale of draw object.
`mfObjPosition`	Position of draw object in world coordinates.
`mfObjOrigin`	Origin of draw object.
`mfObjOrientation`	Return WXYZ orientation of draw object.

Camera Manipulation

`mfView`	Viewpoint specification.
`mfCamZoom`	Zoom in/out.
`mfCamPan`	Move the camera horizontally and vertically.
`mfCamProj`	Set the camera projection mode.

Graphics

Linear Graphs

`mfPlot`	2-D linear plot.
`mfPlot3`	3-D linear graphs.
`mfRibbon`	3-D ribbons.
`mfTube`	3-D tubes.

Surface Graphs

`mfSurf`	Surface plot.
`mfMesh`	Mesh plot.
`mfSurfc`	Combined plot of suface and contour3.
`mfMeshc`	Combined plot of mesh and contour3.
`mfPColor`	Pseudocolor plot of a matrix.
`mfContour`	2-D contour.
`mfContour3`	3-D contour.
`mfSolidContour`	2-D solid contour.
`mfSolidContour3`	3-D solid contour.

`mfOutline`	Wireframe outline corners.
`mfIsoSurface`	3-D plot isovalue surface from volume data.

Slice Graphs

`mfSliceXYZ`	Display orthogonal slice-planes through volumetric data.
`mfSliceIJK`	Display orthogonal slice-planes through volumetric data.
`mfSlicePlane`	Display orthogonal slice-planes along i, j or k indices.
`mfGetSliceXYZ`	Display orthogonal slice-planes along arbitrary direction.
`mfGetSliceIJK`	Retreive orthogonal slice-plane(s) through volumetric data.
`mfGetSlicePlane`	Retreive orthogonal slice-plane(s) along i, j or k indices.

Streamline Graphs

`mfStreamLine`	Streamlines from 2-D or 3-D vector data.
`mfStreamDashedLine`	Stream of dashed lines from 2-D or 3-D vector data.
`mfStreamRibbon`	Stream of ribbons in ribbon from 2-D or 3-D vector data.
`mfStreamTube`	Stream of tubes from 2-D or 3-D vector data.

Triangular Surface Graphs

`mfTriSurf`	Polygonal surface plot.
`mfTriMesh`	Polygonal mesh plot.
`mfTriContour`	Contour on polygonal plot.
`mfPatch`	Add patch on 2-D or 3-D coordinates.

Unstructured Grids

`mfTetSurf`	Polyhedral surface plot.
`mfTetMesh`	Polyhedral mesh plot.

`mfTetContour`	Contour on polyhedral plot.
`mfTetIsoSurface`	Poyhedral isosurface plot.

Unstructured Point Set

`mfPoint`	Display input points in 3-D space.
`mfDelaunay`	2-D Delaunay triangulation of input points.
`mfDelaunay3`	3-D Delaunay triangulation of input points.
`mfGetDelaunay`	2-D Delaunay triangulation of input points.
`mfGetDelaunay3`	3-D Delaunay triangulation of input points.

Velocity Vectors

`mfQuiver`	3-D velocity vectors.
`mfQuiver3`	3-D velocity vectors.

Image

`mfImage`	Display image file.
`mfImRead`	Read in image file.
`mfImWrite`	Write to image file.

3-D Objects

`mfMolecule`	Draw stick and ball model of molecules.
`mfSphere`	Draw stick and ball model of molecules.
`mfCube`	Draw a sphere.
`mfCylinder`	Draw Cube.
`mfCone`	Draw Cylinder.
`mfAxisMark`	3-directional mark on arbitrary point.

Property Setting

`msGSet`	Set property of specified graph.

`msDrawMaterial`	Set draw object's transparency reflectance, ambient reflectance, diffuse reflectance and specular reflectance.
`msDrawTexture`	Texture mapping.
`mfIsValidDraw`	Check validity of draw object.
`mfGetCurrentDraw`	Return handle of current draw object.
`msRemoveDraw`	Remove draw object from plot space.
`msSetDrawName`	Name of draw object.

Simple GUI

`msShowMessage`	Pop up message dialog box.
`mfInputString`	Pop up string insertion dialog box.
`mfInputValue`	Pop up value insertion dialog box.
`mfInputVector`	Pop up vector insertion dialog box.
`mfInputMatrix`	Pop up matrix insertion dialog box.
`mfFileDialog`	Pop up file open dialog box.
`mfInputYesNo`	Pop up yes-no query dialog box.

Figure

mfFigure

Create figure in Graphics Viewer.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfFigure([int figure_id, char* figure_name]);
```

Descriptions

Function `mfFigure` creates a new figure with ID specified in argument `figure_id` and with name specified in argument `figure_name.` The ID and the name of the figure is displayed on the figure tab.

`mfFigure()`

- If argument `figure_id` is not provided, it creates a new figure with an automatically selected figure ID.

`id = mfFigure(...)`

- It will return the figure ID once the figure is created.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfFigure.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray x, y1, y2, y3;
    int num;

    x = mfLinspace(-MF_PI, MF_PI, 100);
    y1 = mfSin(x);
    y2 = mfCos(x);
    y3 = mfLinspace(-1, 1, 100);

    // Create figure 1 and plot x, y1
    mfFigure(1);
    mfPlot(x, y1);
    mfAxis(mfV(-MF_PI, MF_PI, -1.0, 1.0));

    // Returns number of figures
    num = mfFigureCount().ToDouble();
    mfDisplay(mf(num), "mfFigureCount()");
    mfViewPause();
```

```
    // Create figure 2 and plot x, y2
    mfFigure(2);
    mfPlot(x, y2, "r");
    mfAxis(mfV(-MF_PI, MF_PI, -1.0, 1.0));

    // Returns number of figures
    num = mfFigureCount().ToDouble();
    mfDisplay(mf(num), "mfFigureCount()");
    mfViewPause();

    // Create figure 3 and plot x, y3
    mfFigure(3);
    mfPlot(x, y3, "g");
    mfAxis(mfV(-MF_PI, MF_PI, -1.0, 1.0));

    // Returns number of figures
    num = mfFigureCount().ToDouble();
    mfDisplay(mf(num), "mfFigureCount()");
    mfViewPause();

    // Close figure 2
    mfCloseFigure(2);

    // Returns number of figures
    num = mfFigureCount().ToDouble();
    mfDisplay(mf(num), "mfFigureCount()");
    mfViewPause();
}
```

See Also

mfCloseFigure, mfFigureCount

mfCloseFigure

Close figure in Graphics Viewer.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfCloseFigure(int figure_id);
```

Descriptions

Function `mfCloseFigure` closes the target figure specified by argument `figure_id.`

See Also

mfFigure, mfFigureCount

mfFigureCount

Number of figures in graphics viewer.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfFigureCount();
```

Descriptions

Function `mfFigureCount` returns the number of figures that are in Graphics Viewer.

See Also

mfFigure, mfCloseFigure

Window Frame

mfWindowCaption

Graphics Viewer title.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfWindowCaption();
void mfWindowCaption(char* title);
```

Descriptions

Function `mfWindowCaption` sets the caption on the top window panel of the Graphics Viewer. The default caption is "MATFOR 3.0".

`title = mfWindowCaption()`

It can also be used as an inquiry function if given no argument.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfWindowCaption.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray x, y;

    x = mfLinspace(0, 2*MF_PI, 101);
    y = mfSin(x);

    mfPlot(x, y);
    mfWindowCaption("Example Change WindowCaption to 2D Plot");

    mfViewPause();

}
```

See Also

mfWindowSize, mfWindowPos

mfWindowSize

Graphics Viewer frame size.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfWindowSize();
void mfWindowSize(int width, int height);
```

Descriptions

Function `mfWindowSize` sets the frame size of the Graphics viewer.

`mfWindowSize(width, height)`

Arguments `width` and `height` can be integer scalars or mfArrays containing integer scalars.

`size = mfWindowSize()`

It can be used as inquiry function for the frame size if no argument is specified. The return argument `size` is a double vector in the format of [width, height].

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfWindowSize.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray x, y;

    x = mfLinspace(0, 2*MF_PI, 101);
    y = mfSin(x);

    mfPlot(x, y);
    mfWindowSize(600, 600);
    mfViewPause();

}
```

See Also

mfWindowPos

Graphics Viewer frame position.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfWindowPos();
void mfWindowPos(int x, int y);
```

Descriptions

Function `mfWindowSize` sets the frame position of the Graphics viewer.

`mfWindowPos(x, y);`

Arguments `x` and `y` can be integer scalars or mfArrays containing integer scalars.

`pos = mfWindowPos()`

It can be used as inquiry function for the frame position if no argument is specified. The return argument `pos` is a integer vector in the format of [x, y].

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfWindowPos.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray x, y;

    x = mfLinspace(0, 2*MF_PI, 101);
    y = mfSin(x);

    mfPlot(x, y);
    mfWindowPos(220, 0);
    mfViewPause();

}
```

See Also

mfWindowSize

Display

mfGDisplay

Display mfArray data on a MATFOR Data Viewer.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfGDisplay(mfArray x[, char* name]);
void mfGDisplay(mfArray x1, char* n1[, mfArray x2, char* n2, …]);
```

Descriptions

Function `mfGDisplay` displays your mfArray data on a Data Viewer.

You can output single or multiple mfArray data to the Data Viewer.

```
mfGDisplay(x)
mfGDisplay(x , 'name')
```

- Display data of mfArray `x` on Data Viewer.
- Character string 'name' specifies the label on the spreadsheet tab displaying data of `x`.

```
mfGDisplay(x, 'name', x1, 'name1', ...)
```

- Display multiple datasets `x, x1, ...`, on the Data Viewer, labeled with 'name', 'name1', ... respectively. The arguments must be specified in pairs.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfGDisplay.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray x, y, z;

    x = mfRand(5);
    y = mfRand(5);
    z = mfComplex(x, y);

    //Display the data of X, Y and Z on a MATFOR Data Viewer.
    mfGDisplay(x, "x", y, "y", z, "z");

    // Pause program to display Data Viewer
    mfViewPause();
```

```
}
```

See Also

mfDrawNow

Draw all pending graphs.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfDrawNow();
```

Descriptions

Function `mfDrawNow` draws all pending graphics on the current Figure.

The function is used mainly for animation. It does not pause program execution. As a result, the Figure is displayed, updated, and flushed almost immediately. Animation results when `mfDrawNow` is used within a do loop in which the graphics object is continuously being updated.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfDrawNow.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray x, y, h;
    int i;

    x = mfLinspace(-MF_PI, MF_PI, 50);
    y = mfSin(x);

    // Create an initial copy of the graph to be animated.
    // This is recommended as you can obtain the current
    // graphics handle and use erase mode for the animation.
    h = mfPlot(x, y);

    // Use a Do Loop to animate the sin(x) curve. Note,
    // mfGSet continuously updates the specified data and
    // sleep slows down the program execution.
    for(i = 1; i <= 100; i++)
    {
        y = mfSin(x+0.1*i);
        mfGSet(h, "ydata", y);
        mfDrawNow();
    }

    // Pause the program to continue displaying the
    // Graphics Viewer after the Do Loop ends.
    mfViewPause();
```

```
}
```

See Also

mfViewPause

mfViewPause

Pause program execution.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfViewPause();
```

Descriptions

Function `mfViewPause` pauses program execution for graphical display.
Use this function wherever you wish to pause a program to visualize your data. To continue your program execution, you can either close the Graphics Viewer, or click on the "Continue" button located at the top right corner of the Graphics Viewer.

You must add at least one line of `mfViewPause()`after each set of graphical creation routines. If the function were left out, you would only see a flash as the Graphics Viewer is opened and closed almost immediately by the program.

See Also

Recording

mfRecordStart, mfRecordEnd

Record animation as avi file, MATFOR mfa file or bitmap files.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfRecordStart(char* filename[, char* property1, int value1, ...]);
void mfRecordEnd();
```

Descriptions

Functions `mfRecordStart` and `mfRecordEnd` are MATFOR build-in functions for recording visualized data as avi animation files or mfa files.

`mfRecordStart(filename, property1, value1, property2, value2, property3, value3)`

- Records the animations as avi files or MATFOR mfa files.
- Argument `filename` can take the following values.

Value	Meaning
"*.avi"	Example: "filename.avi". Record the current animation in an avi file. Avi or video recording uses frame capturing method to capture the animation playing on the current Graphics Viewer. The recorder automatically detects the type of video compression utilities available in your system and presents a drop-list for you to choose.
"*.mfa"	Example: "filename.mfa". Record the current animation as an mfa file. The mfa file format is a MATFOR-specific record of all data used for generating the current animation on the Graphics Viewer. You can playback the animation by using MATFOR mfPlayer at a later time. Using mfPlayer, you can perform graphical manipulations on the animation, such as zoom in/out, rotation, colormap adjustment, etc.
"*.bmp"	Example: "filename.bmp". Save each frame of the animation into a bitmap file. The name of each bitmap file is set to be the input file name followed by four digits starting from 0000 counting up. In the example, the names of the first two bitmap files would be "filename0000.bmp" and "filename0001.bmp".
"*.tif"	Example: "filename.tif". Save each frame of the animation into a TIFF file. The naming method of the saved files is similar to saving as bitmap files.

- The optional arguments property1, property2 and property3 and the corresponding arguments value1, value2 and value3 can take the following values.

Property	Meaning
"framerate"	Specify the number of frames captured per second. The argument applies only when the animation file is saved in avi format. By default, MATFOR records avi file at 15 frames per second. The recommended range is 5 to 30 frames per second. The higher the frame rate, the faster the frames are looped through. Likewise, smaller frame rate slows down the animation.
"width"	Specify the width of the captured figure frame. The argument applies only when the animation is saved in avi format or bmp format. Without specifying the argument, the captured figure frame will have the exact same width as the displaying figure frame.
"height"	Specify the height of the captured figure frame. The argument applies only when the animation is saved in avi format or bmp format. Without specifying the argument, the captured figure frame will have the exact same height as the displaying figure frame.

```
mfRecordEnd()
```

- Stop the recording.

The general syntax of the recording functions is as follows:

```
mfRecordStart('animation.avi')
```

or

```
mfRecordStart('animation.bmp')
```

------- <animation codes>

```
mfRecordEnd()
```

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfRecord.cpp
```

```
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray a, b, c, x, y, z, indxi, indxj, h;
    int i;

    a = mfLinspace(-3, 7, 51);
    b = mfLinspace(-2, 8, 51);
    mfMeshgrid(mfOut(x, y), a, b);

    // Next, initializ indxi and indxj using Meshgrid.
    // Compute z using indxi and indxj.
    // Note, a ".0" is added to the integers, to ensure
    // double precision. MATFOR uses only double precision
    // data.
    c = mfColon(1, 51);
    mfMeshgrid(mfOut(indxi, indxj), c);
    z = 3.0*mfSin((indxi+1)/10.0)*mfCos((indxj+1)/10.0)
        + 2.0*mfSin((indxi+indxj)/10.0);

    // Plot a mesh grid using mfArray x, y and z for the grid
    // intersections.
    h = mfMesh(x, y, z);

    // Start record of an animation using avi file format
    // Records scarf.avi in the Debug directory
    mfRecordStart("scarf.avi");

    // Animate the mesh using a do loop.
    for(i = 1; i <= 30; i++)
    {
        z = 3.0*mfSin((indxi+i+1)/10.0)*mfCos((indxj+1-i)/10.0)
            + 2.0*mfSin((indxi+indxj+i)/10.0);

        // Update z
        mfGSet(h, "zdata", z);
        // Update Graphics Viewer
        mfDrawNow();
    }

    // end video record
    mfRecordEnd();

    // Pause to display the graph.
    mfViewPause();
}
```

See Also

mfGSet, mfFigure

mfExportImage

Save figure graph as picture file.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfExportImage(char* filename[, int width, int height]);
```

Descriptions

Function `mfExportImage` saves the graph(s) in current figure as a picture file. You can choose the picture format by specifying the extension in argument `filename`. For example, `"filename.bmp"` would save it as a bitmap file. The supported formats are: BMP, JPEG, TIFF, PS and PNG.

See Also

Plot Creation and Control

mfSubplot

Create subplot in active figure.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfSubplot(int row, int col, int index);
```

Descriptions

Function `mfSubplot` divides the plot space of a Graphics Viewer into m-by-n rectangular sub-plot spaces, as specified by the `rol, col` arguments.

Each sub-plot space is numbered row-wise so that a subplot space at position (1,2) is numbered 2 and (2,2) is numbered 4.

The current subplot is set to the subplot number `index.` Any subsequent operations will be performed on the current subplot.

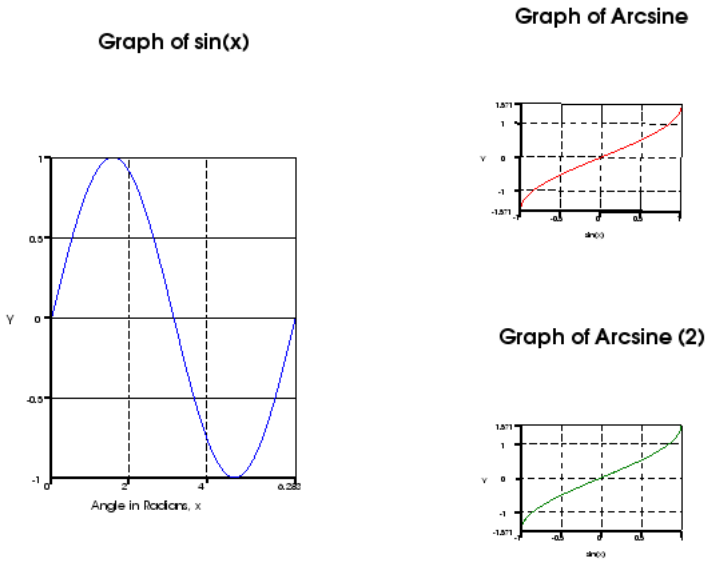
Example

Code

```
// ************************************************
// AnCAD example file
// file: mfSubplot.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray x, y1, y2;

    x = mfLinspace(0, 2*MF_PI, 101);
    y1 = mfSin(x);
    y2 = mfASin(y1);

    // Divide the plotting space into 1 on left 2 on right sub-plot spaces,
    // and specify the subplot space 1 or left sub-plot as current.
    mfSubplot("5, 5[5, 5]", 1);
    // Plot and label the graph.
    mfPlot(x, y1);
    mfAxis(mfV(0.0, 2*MF_PI, -1.0, 1.0));
    mfTitle("Graph of sin(x)");
    mfXLabel("Angle in Radians, x");
    mfCamZoom(0.80);

    // Next, specify subplot space p=2, or the right-top subplot
    // space as current.
    mfSubplot("", 2);
    // Again, plot and label the graph.
    mfPlot(y1, y2, "r");
```

```
    mfAxis(mfV(-1.0, 1.0, -MF_PI/2, MF_PI/2));
    mfTitle("Graph of Arcsine");
    mfXLabel("sin(x)");
    mfCamZoom(0.80);

    // Finally, specify subplot space p=3, or the right-bottom subplot
    // space as current.
    mfSubplot("", 3);
    // Again, plot and label the graph.
    mfPlot(y1, y2, "g");
    mfAxis(mfV(-1.0, 1.0, -MF_PI/2, MF_PI/2));
    mfTitle("Graph of Arcsine (2)");
    mfXLabel("sin(x)");
    mfCamZoom(0.80);

    // Pause the program to display the graphs.
    mfViewPause();

}
```

Result



See Also

mfClearSubplot

mfClearSubplot

Remove all draws in specified subplot.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfClearSubplot(int subplot_id);
```

Descriptions

Function `mfClearSubplot` removes all draws in current or specified subplot if argument `subplot_id` is specified.

See Also

mfSubplot

mfHold

Hold previous graph on plot space.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void msHold(char* mode);
```

Descriptions

Function `mfHold` holds the current graph in plot space so that it would not be overwritten by the next graph creation. Subsequent creations of graphs would be appended one after another in the plot space.

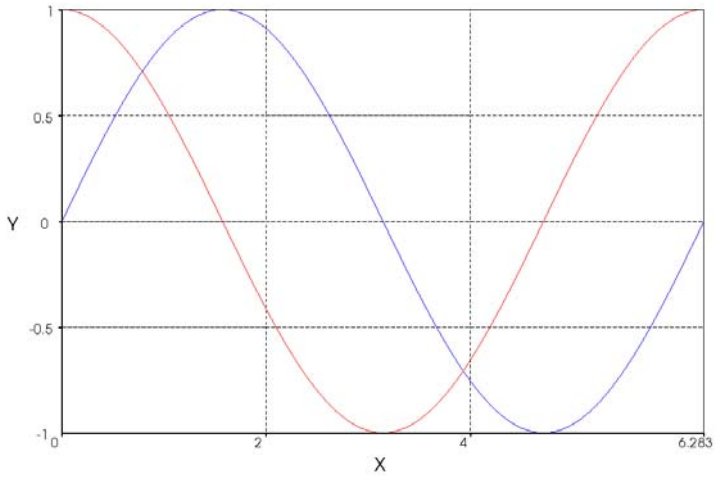
Argument `mode` is a string containing "on" or "off".

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfHold.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"


void main() {

    mfArray x, y1, y2, a;

    x = mfLinspace(0, 2*MF_PI, 100);
    y1 = mfSin(x);
    y2 = mfCos(x);

    // Plot y1 = mfSin(x)
    mfPlot(x, y1);
    mfAxis(mfV(0.0, 2*MF_PI, -1.0, 1.0));

    // Hold the figure for plotting
    mfHold("on");

    // Plot y2 = mfCos(x) to the same figure
    mfPlot(x, y2, "r");

    mfHold("off");
    // Pause the figure for viewing
    mfViewPause();

}
```

Result



See Also

mfIsHold

mfIsHold

Return status of plot space.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfIsHold();
```

Descriptions

Function `mfIsHold` returns the status of current plot space. The output is an mfArray containing logical data. It returns true if the plot space is held on, false otherwise.

See Also

Plot Annotation and Appearance

mfTitle, mfXLabel, mfYLabel, mfZLabel

Label the axis objects.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfTitle();
mfArray mfXLabel();
mfArray mfYLabel();
mfArray mfZLabel();

void mfTitle(char* title[, mfArray color, int font_size]);
void mfXLabel(char* xlabel);
void mfYLabel(char* ylabel);
void mfZLabel(char* zlabel);
```

Descriptions

Functions `mfTitle, mfXLabel, mfYLabel` and `mfZLabel` annotate a graph with title, x-axis label, y-axis label and z-axis label respectively. By default, x-axis is labeled as "x", y-axis is labeled as "y" and z-axis label is labeled as "z".

You can also annotate the graph through the Axis menu ->Title, XLabel, YLabel and ZLabel functions of the Graphics Viewer.

```
title = mfTitle()
xlabel = mfXLabel()
ylabel = mfYLabel()
zlabel = mfZLabel()
```

They can also be used as inquiry functions to retrieve the title and labels that are set by users.

`mfTitle(title, color, font_size)`

- Set the rgb color code and the font size of the title by specifying arguments `color` and `font_size`. The rgb color code is specified as [r, g, b] where 0 < r, g, b < 1.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfTitle.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"
```

```
void main() {

    mfArray x, y;

    x = mfLinspace(0, 2*MF_PI, 51);
    y = mfSin(x);

    // Plot the x, y curve using mfPlot.
    mfPlot(x, y, "rx");

    // Annotate the graph with title, x-axis label and y-axis label.
    mfTitle("Graph of sin(x)");
    mfXLabel("x in radians");
    mfYLabel("sin(x)");

    // Pause program execution.
    mfViewPause();

}
```

See Also

mfCaption

mfText

2-D text annotation.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfText(char* text[, mfArray loc, mfArray color, mfArray
font_size]);
```

Descriptions

Function `mfText` places a two-dimensional text on the current subplot.

`mfText(text, loc, color, font_size)`

- Argument `text` can be a string or an mfArray containing a string.
- Argument `loc` is a 1-by-2 vector in the format [m, n]. Each element contains a value ranging from 0 to 1. Specifying m as 0 would place the text annotation to the left-most position of the subplot window and specifying n as 0 would place the text annotation on the bottom of the subplot window. For example, the vector `[0, 0]`would place the text annotation on the bottom-left corner of the subplot and vector [0.5, 0.5] would place the text annotation in the center of the plot space.

- You can set the color and the font size through arguments `color` and `font_size.` Argument `color` contains the rgb color code which is specified as [r, g, b] where 0 < r, g, b < 1.

`h = mfText(...)`

- Handle `h` retrieves a handle to the text annotation created by `mfText(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()`to retrieve the handle of the current graphics object.

You can specify properties of the text annotation through handle `h` with function `mfGSet.`

The properties available are:

1. text 2. location 3. color 4. font_size 5. bold 6. italic 7. shadow 8. just (justification): "left", "right" or "center"

See Also

mfAnnotation

3-D text annotation.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfAnnotation(char* text[, mfArray loc, mfArray color, int
font_size]);
```

Descriptions

Function `mfAnnotation` places a three-dimensional text annotation on the current subplot.

`mfAnnotation(text, loc, color, font_size)`

- Argument `text` can be a string or an mfArray containing a string.
- Argument `loc` is a 1-by-3 vector in the format [m, n, p]. Each element contains a value ranging from 0 to 1. Specifying m as 0 would place the text annotation to the left-most position of the subplot window and specifying n as 0 would place the text annotation on the bottom of the subplot window. For example, the vector `[0, 0, 0]`would place the text annotation on the bottom-left corner of the subplot window and vector [0.5, 0.5] would place the text annotation in the center of the plot space.

`h = mfAnnotation(...)`

- Handle `h` retrieves a handle to the text annotation created by `mfAnnotation(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()`to retrieve the handle of the current graphics object.

You can specify properties of the text annotation through handle `h` with function `mfGSet.`

The properties available are:

1. text 2. location 3. color 4. font_size 5. bold 6. italic 7. shadow // 8. just (justification): "left", "right" or "center", not open 9. offset, (x,y) 2x1 array

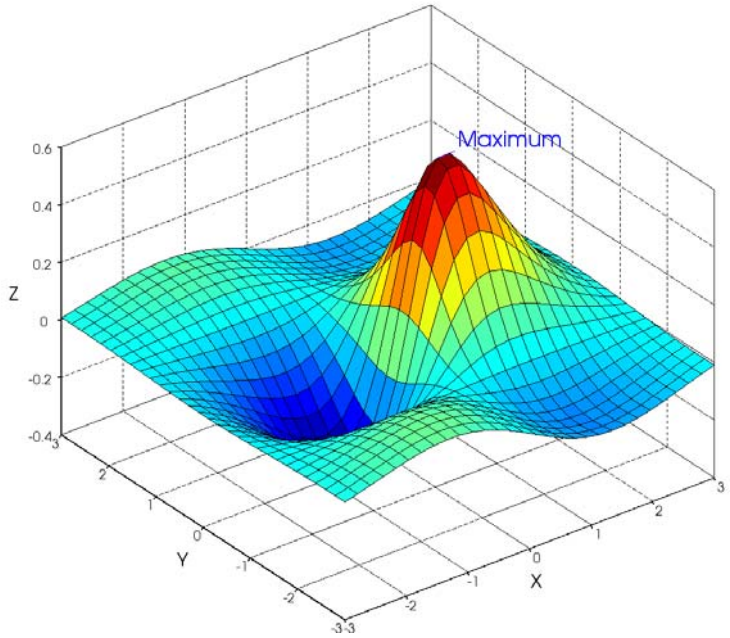
Example

Code

```
// ************************************************
// AnCAD example file
// file: mfAnnotation.cpp
//
// Copyright 2003 AnCAD
// ************************************************
```

```
#include "cml.h"
#include "fgl.h"

void main() {

    mfArray m, x, y, z, cm;

    m = mfLinspace(-3, 3, 30);
    mfMeshgrid(mfOut(x, y), m);
    z = mfSin(x) * mfCos(y) / ( x*(x-0.5) + (y+0.5)*y + 1);

    mfSurf(x, y, z);
    mfAxis(-3.0, 3.0, -3.0, 3.0, -0.4, 0.6);
    mfAnnotation("Maximum", mfV(0.7241, -0.1034, 0.5877), mfV(0, 0, 1));
    mfViewPause();
}
```

Result



See Also

mfText

mfShading

Shading methodology of surface object.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfShading(char* mode);
void mfShading(int draw_id, char* mode);
```

Descriptions

Function `mfShading` sets the shading mode for all draws in plot space.

`mfShading(mode)`

- Specify shading type for functions `mfSurf` and `mfMesh.` The options available are listed in the table below.

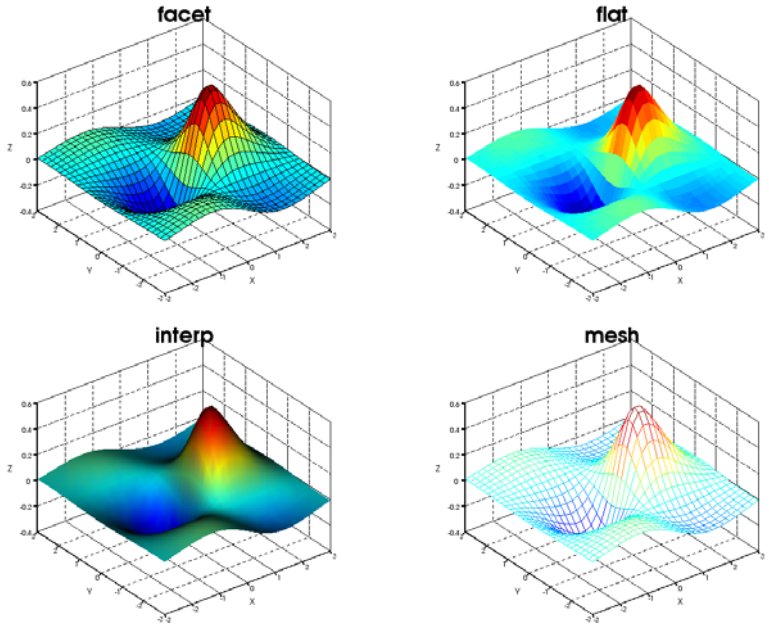
Example

Code

```
// ************************************************
// AnCAD example file
// file: mfShading.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray a, x, y, z, h;

    a = mfLinspace(-3, 3, 30);
    mfMeshgrid(mfOut(x, y), a);
    z = mfSin(x) * mfCos(y) / ( x*(x-0.50) + (y+0.50)*y + 1);

    mfSubplot(2, 2, 1);
    mfSurf(x, y, z);
    mfShading("facet");
    mfTitle("facet");
    mfCamZoom(1.2);

    mfSubplot(2, 2, 2);
    mfSurf(x, y, z);
    mfShading("flat");
    mfTitle("flat");
    mfCamZoom(1.2);

    mfSubplot(2, 2, 3);
    mfSurf(x, y, z);
    mfShading("interp");
    mfTitle("interp");
    mfCamZoom(1.2);
```

```
    mfSubplot(2, 2, 4);
    mfSurf(x, y, z);
    mfShading("mesh");
    mfTitle("mesh");
    mfCamZoom(1.2);

    mfViewPause();

}
```

Result



See Also

mfColorbar

Display color scale.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfColorbar(char* mode);
void mfColorbar(char* property, mfArray value);
```

Descriptions

Function `mfColorbar` controls the colorbar through specification of argument `mode.`

A colorbar displays the current color map and acts as a color scale showing the relationship between graphics data and color. In the case of a surface object, it shows the relationship between color and height of the surface object. You can also select the colorbar setting from the menu and toolbar functions of the Graphics Viewer.

```
mfColorbar(mode)
mfColorbar(property, value)
```

Argument `mode` can be:

mode	Meaning
`"on"`	Display a colorbar on the Graphics Viewer.
`"off"`	Hide the colorbar.
`"vert"`	Display a vertical colorbar.
`"horz"`	Display a horizontal colorbar.

Argument `property` can be:

Property	Meaning
`label_count`	Number of labels displayed on the colorbar.
`label_color`	Color of the colorbar labels. Corresponding argument `value` is an mfArray containing a string that specifies the color, e.g.

	“y”, or a 1-by-3 mfArray contains the rgb codes.

See Also

mfColormap

Colormap type.

Module

```
fgl.h/fgl.lib
```

C++ Prototype

```
void mfColormap(char* type);
void mfColormap(mfArray colormap);
```

Descriptions

Function `mfColormap` specifies the colormap type used for drawing surface objects.

`mfColormap(type)`

- Argument `type` specifies the type of colormap used. The argument can be an mfArray containing a string specifying the type of colormap or a character string. MATFOR provides the following types of colormap:

Value	Meaning
"jet"	Range from blue to red, and pass through the colors cyan, yellow, and orange.(default)
"gray"	Return a linear grayscale colormap.
"hot"	Vary smoothly from black, through shades of red, orange, and yellow, to white.
"cool"	Vary smoothly from cyan to magenta.
"copper'	Vary smoothly from black to bright copper.
"hsv"	Vary the hue component of the hue-saturation-value color model.
"spring"	Consist of colors that are shades of magenta and yellow.
"summer"	Consist of colors that are shades of green and yellow.
"autumn"	Vary smoothly from red, through orange, to yellow.

"winter"	Consist of colors that are shades of blue and green.

`mfColormap(colormap)`

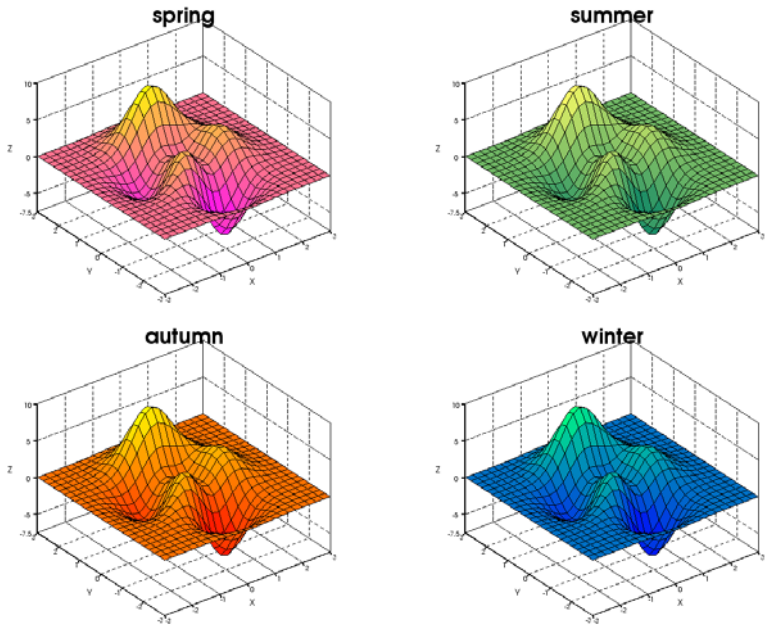
- Allows users to customize the colormap through the argument `colormap` which is a m-by-3 matrix consists of m sets of rgb color code. The rgb color code is specified as [r, g, b] where 0 < r, g, b < 1.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfColormap.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray x, u, v, z, h;

    x = mfLinspace(-3, 3, 25);
    mfMeshgrid(mfOut(u, v), x);
    z = 3*mfPow(1-u,2)*mfExp(-mfPow(u,2)-mfPow(v+1,2))
        -
(10*(u/5-mfPow(u,3)-mfPow(v,5))*mfExp(-mfPow(u,2)-mfPow(v,2)))
        - mfExp(-mfPow(u+1,2)- mfPow(v,2))/3;

    mfSubplot(2, 2, 1);
    mfTitle("spring");
    mfSurf(u, v, z);
    h = mfColormap("spring");
    mfCamZoom(1.2);

    mfSubplot(2, 2, 2);
    mfTitle("summer");
    mfSurf(u, v, z);
    h = mfColormap("summer");
    mfCamZoom(1.2);

    mfSubplot(2, 2, 3);
    mfTitle("autumn");
    mfSurf(u, v, z);
    h = mfColormap("autumn");
    mfCamZoom(1.2);

    mfSubplot(2, 2, 4);
    mfTitle("winter");
    mfSurf(u, v, z);
    h = mfColormap("winter");
    mfCamZoom(1.2);

    mfViewPause();
}
```

Result



See Also

mfColormapRange

Range of color map.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfColormapRange();
void mfColormapRange(double min, double max);
void mfColormapRange(mfArray range);
void mfColormapRange("auto");
```

Descriptions

Function `mfColormapRange` sets the range of color map.

```
mfColormapRange(min, max)
mfColormapRange([min, max])
```

- Arguments `min` and `max` specify the minimum and maximum of the range.
- The range can also be input as a vector containing the min and max of the range.

```
range = mfColormapRange()
```

- It can also be used as an inquiry function to retrieve the color map range in the format of [min, max] if no argument is specified.

See Also

mfBackgroundColor

Background color of plot space.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfBackgroundColor();
void mfBackgroundColor(double r, double g, double b);
void mfBackgroundColor(mfArray colorCode);
```

Descriptions

Function `mfBackgroundColor` set the background color of plot space.

```
mfBackgroundColor(r, g, b)
mfBackgroundColor(colorCode)
```

- Arguments `r, g, b` contain a real number within the range 0 to 1 specifying the rgb code of the background color.
- Argument `colorCode` is a vector containing the rgb color code in the format of [r, g, b] where 0 < r, g, b < 1.

```
colorCode = mfBackgroundColor()
```

- Retrieves the color code of current plot space in the format of [r, g, b] where 0 < r, g, b < 1.

As an example, `mfBackgroundColor(1, 1, 1)`sets the background color to white.

See Also

Axis Control

mfAxis

Manipulate the axis object.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfAxis();
void mfAxis(mfArray xyzrange);
void mfAxis(int x_min, int x_max, int y_min, int y_max[, int z_min, int
z_max]);
void mfAxis(char* mode);
void mfAxis(char* property, mfArray value);
```

Descriptions

Function `mfAxis` sets the properties of the x-axis, y-axis and z-axis, such as the range, mode and color. It can also be used an inquiry function for the ranges of the axes.

`xyzrange = mfAxis()`

- Retrieves the ranges of the axis objects. The output argument `xyzrange` is a vector in the format [x_min, x_max, y_min, y_max, z_min, z_max].

```
mfAxis(xyzrange)
mfAxis(x_min, x_max, y_min, y_max)
mfAxis(x_min, x_max, y_min, y_max, z_min, z_max)
```

- Sets the ranges of the axis objects. The input data can be provided in two ways. One is through a vector `xyzrange` in which the ranges of the axes objects are specified; whereas the other one is specified in the element-by-element way.
- Argument `xyzrange` is a vector in the format [x_min, x_max, y_min, y_max, z_min, z_max].
- Arguments `x_min, x_max, y_min, y_max, z_min, z_max` specify the displaying ranges of x-axis, y-axis and z-axis.

```
mfAxis(mode)
mfAxis(property, value)
```

- Sets the mode and property of the axis objects.

Argument `mode` can be:

mode	Meaning

“on”	Display the tick marks, labeling and background of the current axis object.
“off”	Remove the tick marks, labeling and background of the current axis object.
“Normal”	Restore the current axis object to its full size and remove any restrictions on scaling.
“equal”	Use the same aspect ratio for each axis of the axis object. In other words, tick marks of equal increments have the same size on all axes.
“Auto”	Set axes scaling to automatic mode. MATFOR sets the limits, minimum and maximum of each axis based on the extents of the graphs plotted. It is the default setting.

Argument `property` can be:

property	Meaning
“axis_color”	Color of all three axes. Corresponding argument `value` is an mfArray containing a string that specifies the color, e.g. “y”, or a 1-by-3 mfArray contains the rgb codes.
"xaxis_color"	Color of the x-axis. Corresponding argument `value` is an mfArray containing a string that specifies the color, e.g. “y”, or a 1-by-3 mfArray contains the rgb codes.
"yaxis_color"	Color of the y-axis. Corresponding argument `value` is an mfArray containing a string that specifies the color, e.g. “y”, or a 1-by-3 mfArray contains the rgb codes.
“zaxis_color”	Color of the z-axis. Corresponding argument `value` is an mfArray containing a string that specifies the color, e.g. “y”, or a 1-by-3 mfArray contains the rgb codes.
“xaxis_ticks”	Ticks on the x-axis. Corresponding argument `Value` is a vector of integers. Each of the elements corresponds to a tick displayed on x-axis.

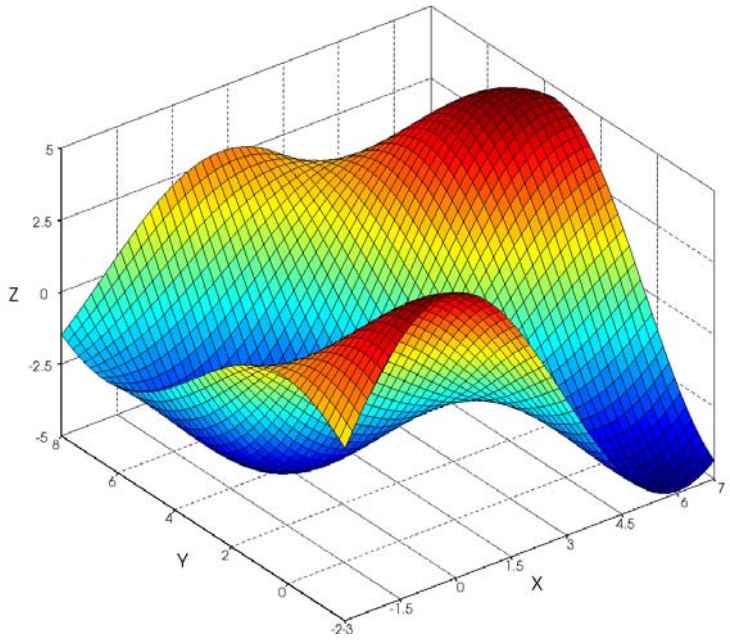
“yaxis_ticks”	Ticks on the y-axis. Corresponding argument `Value` is a vector of integers. Each of the elements corresponds to a tick displayed on y-axis.
“zaxis_ticks”	Ticks on the z-axis. Corresponding argument `Value` is a vector of integers. Each of the elements corresponds to a tick displayed on z-axis.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfAxis.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"
#include "fml.h"


void main() {

    mfArray a, b, c, x, y, z, indxi, indxj;

    a = mfLinspace(-3, 7, 51);
    b = mfLinspace(-2, 8, 51);
    c = mfColon(1, 51);
    mfMeshgrid(mfOut(x, y), a, b);
    mfMeshgrid(mfOut(indxi, indxj), c);
    z = 3*mfSin((indxi+1)/10)*mfCos((indxj+1)/10) +
2*mfSin((indxi+indxj)/10);

    mfSurf(x, y, z);
    mfAxis(-3, 7, -2, 8, -5, 5);
    mfAxis("xaxis_ticks", mfV(-3.0, -1.5, 0.0, 1.5, 3.0, 4.5, 6.0));
    mfAxis("yaxis_ticks", mfV(-2.0, 0.0, 2.0, 4.0, 6.0, 8.0));
    mfAxis("zaxis_ticks", mfV(-5.0, -2.5, 0.0, 2.5, 5.0));
    mfViewPause();
}
```

Result



See Also

mfAxisWall, mfAxisGrid

mfAxisWall

Manipulate the axis wall object.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfAxisWall(char* mode);
void mfAxisWall(char* property, mfArray value);
```

Descriptions

Function `mfAxisWall` sets the color of the three axis-wall objects and switches them on or off. The axis-wall object represents the three axis planes....

`mfAxisWall(mode)`

- Switches the three axis-planes on or off. Argument `mode` is either "on" or "off".

`mfAxisWall(property, value)`

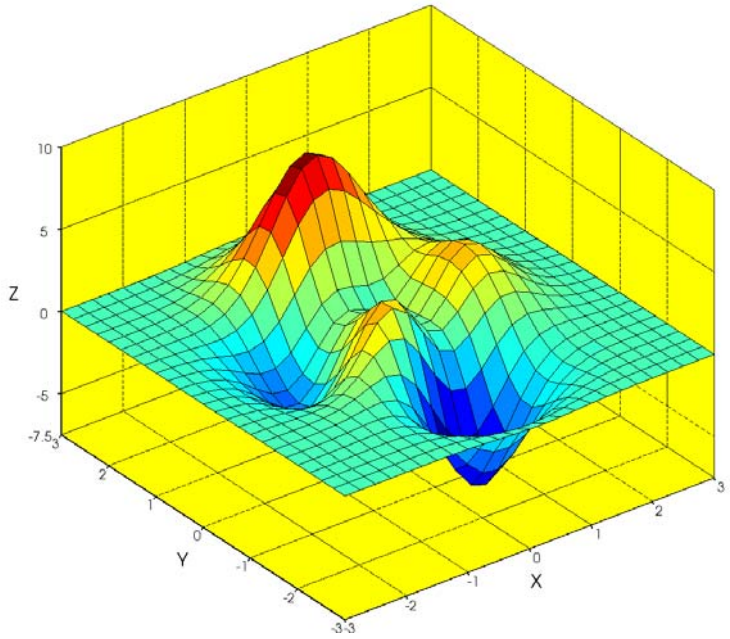
- Sets the color of the axis-wall object. Argument `property` can be a string specified as "color" and argument `value` contains the rgb color code which is specified as [r, g, b] where 0 < r, g, b < 1.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfAxisWall.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray u, v, x, z;

    x = mfLinspace(-3, 3, 25);
    mfMeshgrid(mfOut(u, v), x);
    z = 3*mfPow(1-u,2)*mfExp(-mfPow(u,2)-mfPow(v+1,2))
        -
(10*(u/5-mfPow(u,3)-mfPow(v,5))*mfExp(-mfPow(u,2)-mfPow(v,2)))
        - mfExp(-mfPow(u+1,2)- mfPow(v,2))/3;

    mfSurf(u, v, z);
    mfAxisWall("color", mfV(1, 1, 0));
    mfViewPause();

}
```

Result



See Also

mfAxis

mfAxisGrid

Display grid lines.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfAxisGrid(char* axis, char* mode);
void mfAxisGrid(char* property, mfArray value);
```

Descriptions

Function `mfAxisGrid` sets the properties of the axis grid objects, such as the width, color and pattern.

`mfAxisGrid(axis, mode)`

- Switch an axis on or off. Argument `axis` can be "xaxis", "yaxis" or "zaxis" which corresponds to the three axes respectively. Argument `mode` is either "on or "off".

`mfAxisGrid(property, value)`

- Sets a property of the axis grid objects.

Argument `property` can be:

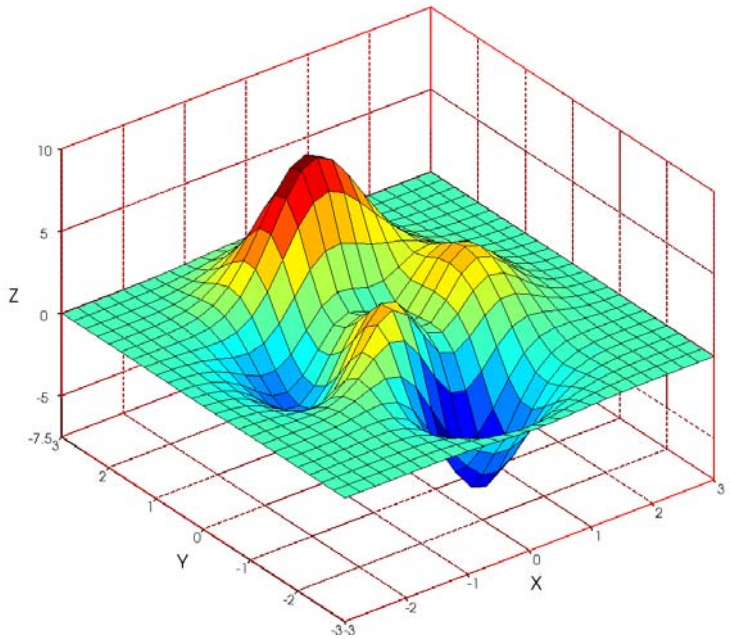
property	Meaning
`“width”`	Line width. Corresponding argument `value` is a scalar integer value.
`“color”`	Line color. Corresponding argument `value` can be an mfArray containing a string specifies the color, e.g. “y”, or a 1-by-3 mfArray contains the rgb codes.
`“pattern”`	Line pattern. Corresponding argument `value` can be an mfArray containing the string that is specified as “`solid", "dashed", "dotted" or "dashdot"`.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfAxisGrid.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray u, v, x, z;

    x = mfLinspace(-3, 3, 25);
    mfMeshgrid(mfOut(u, v), x);
    z = 3*mfPow(1-u,2)*mfExp(-mfPow(u,2)-mfPow(v+1,2))
        -
(10*(u/5-mfPow(u,3)-mfPow(v,5))*mfExp(-mfPow(u,2)-mfPow(v,2)))
        - mfExp(-mfPow(u+1,2)- mfPow(v,2))/3;

    mfSurf(u, v, z);
    mfAxisGrid("width", 2);
    mfAxisGrid("color", mfV(1, 0, 0));
    mfAxisGrid("pattern", "dashed");
    mfViewPause();

}
```

Result



See Also

mfAxis, mfAxisWall

Object Manipulation

mfObjRotateX, mfObjRotateY, mfObjRotateZ

Rotate draw object in degrees about the x-axis, y-axis and z-axis using the right hand rule.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfObjRotateX(mfArray handle, int angle);
void mfObjRotateY(mfArray handle, int angle);
void mfObjRotateZ(mfArray handle, int angle);
```

Descriptions

Functions `mfObjRotateX`, `mfObjRotateY` and `mfObjRotateZ` rotate the draw object that is associated with argument `handle` in degrees about the x-, y-, z- axes respectively using the right hand rule. The axes are the draw object's axes.

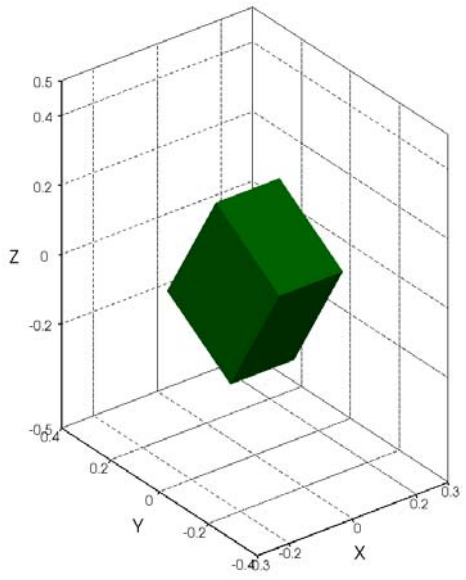
If you want to rotate about the world x-, y- and z- axes, use `mfObjRotateWXYZ(handle, angle, 1, 0, 0)`, `mfObjRotateWXYZ(handle, angle, 0, 1, 0)`and `mfObjRotateWXYZ(handle, angle, 0, 0, 1)`.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfObjRotateX.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray center, cubesize, h;

    center = mfV(0, 0, 0);
    cubesize = mfV(0.2, 0.3, 0.4);

    h = mfCube(center, cubesize, "g");
    mfAxis("equal");
    mfAxis(-0.3, 0.3, -0.4, 0.4, -0.5, 0.5);
    mfViewPause();
    mfObjRotateX(h, 30);
    mfViewPause();
    mfObjRotateY(h, 30);
    mfViewPause();
    mfObjRotateZ(h, 10);
    mfViewPause();

}
```

Result



See Also

mfObjRotateWXYZ

Rotate draw object in degrees about an arbitrary axis.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfObjRotateWXYZ(mfArray handle, int angle, int x, int y, int z);
```

Descriptions

Function `mfObjRotateWXYZ` rotates the draw object that is associated with argument `handle` in degrees about an arbitrary axis specified by the last three arguments `x, y` and `z.`

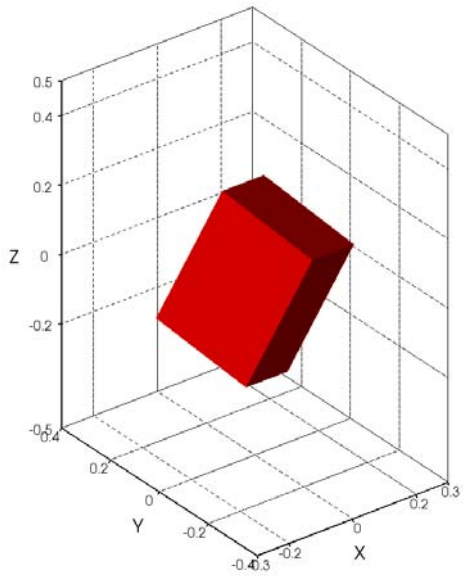
In other words, (x,y,z) is the axis that the rotation will be performed around. If you want to rotate about the object's axes, use `mfObjRotateX`, `mfObjRotateY` and `mfObjRotateZ`.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfObjRotateWXYZ.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray center, cubesize, h;

    center = mfV(0, 0, 0);
    cubesize = mfV(0.2, 0.3, 0.4);

    h = mfCube(center, cubesize, "r");
    mfAxis("equal");
    mfAxis(-0.3, 0.3, -0.4, 0.4, -0.5, 0.5);
    mfViewPause();
    mfObjRotateWXYZ(h, 30, 1, 1, 1);
    mfViewPause();

}
```

Result



See Also

mfObjScale

Scale of draw object.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfObjScale(mfArray handle);
void mfObjScale(mfArray handle, mfArray scale);
void mfObjScale(mfArray handle, double x, double y, double z);
```

Descriptions

Function `mfObjScale` resets the scale independently on the x-, y- and z- axes. A scale of zero is illegal and will be replaced with one.

`scale = mfObjScale(handle)`

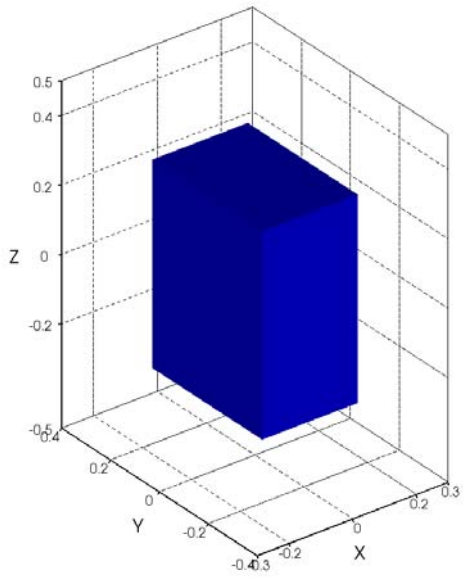
It can also be used as an inquiry function to retrieve the scale of the draw object if given only the handle that associates with the draw object.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfObjScale.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray center, cubesize, h;

    center = mfV(0.0, 0.0, 0.0);
    cubesize = mfV(0.2, 0.3, 0.4);

    h = mfCube(center, cubesize, "b");
    mfAxis("equal");
    mfAxis(-0.3, 0.3, -0.4, 0.4, -0.5, 0.5);
    mfViewPause();
    mfObjScale(h, 1.5, 1.5, 1.5);
    mfViewPause();

}
```

Result



See Also

mfObjPosition

Position of draw object in world coordinates.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfObjPosition(mfArray handle);
void mfObjPosition(mfArray handle, mfArray position);
```

Descriptions

Function `mfObjPosition` sets the position of the draw object that is associated with argument `handle` in world coordinates specified in argument `[x, y, z]`.

`position = mfObjPosition(handle)`

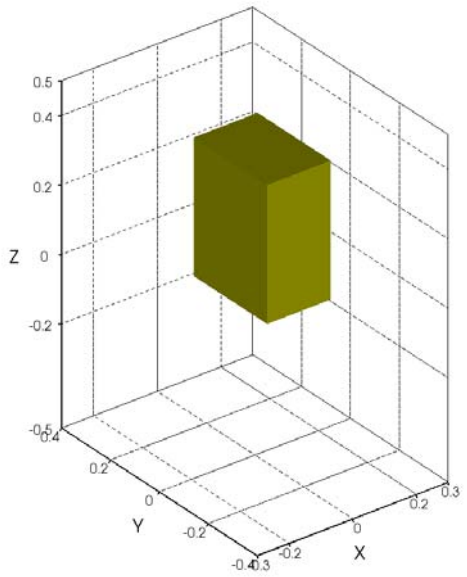
It can also be used as an inquiry function to retrieve the position of the draw object if given only the handle that associated with the draw object.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfObjPosition.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray center, cubesize, h;

    center = mfV(0.0, 0.0, 0.0);
    cubesize = mfV(0.2, 0.3, 0.4);

    h = mfCube(center, cubesize, "y");
    mfAxis("equal");
    mfAxis(-0.3, 0.3, -0.4, 0.4, -0.5, 0.5);
    mfViewPause();
    mfObjPosition(h, 0.1, 0.1, 0.1);
    mfViewPause();

}
```

Result



See Also

mfObjOrigin

Origin of the draw object.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfObjOrigin(mfArray handle);
void mfObjOrigin(mfArray handle, mfArray position);
```

Descriptions

Function `mfObjOrigin` sets the origin of the draw object. All rotations perform on the draw object take place against the origin. Notice that the origin is relative to the position of the object. Whenever the object moves, the origin moves with the position of the object so that they maintain a relative relationship.

`origin = mfObjOrigin(handle)`

It can also be used as an inquire function to retrieve the origin of the draw object if given only the handle that associates with the draw object.

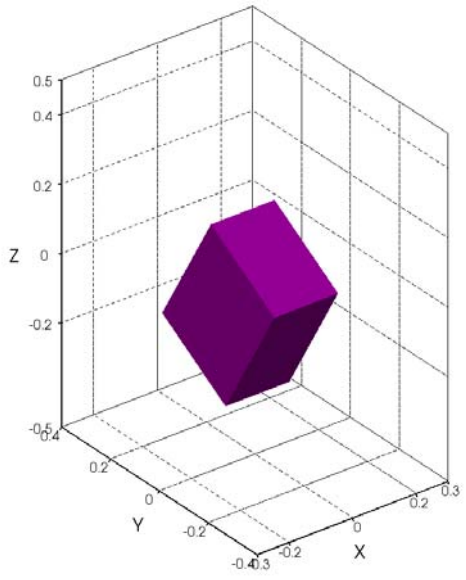
Example

Code

```
// ************************************************
// AnCAD example file
// file: mfObjOrigin.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray center, cubesize, h;

    center = mfV(0.0, 0.0, 0.0);
    cubesize = mfV(0.2, 0.3, 0.4);

    h = mfCube(center, cubesize, "m");
    mfAxis("equal");
    mfAxis(-0.3, 0.3, -0.4, 0.4, -0.5, 0.5);

    // Set origin (0.0, 0.15, 0.0)
    mfObjOrigin(h, 0.0, 0.15, 0.0);
    mfViewPause();

    mfObjRotateX(h, 30);
    mfViewPause();
    mfObjRotateY(h, 30);
    mfViewPause();
    mfObjRotateZ(h, 10);
    mfViewPause();
```

```
}
```

Result



See Also

mfObjOrientation

WXYZ orientation of the draw object.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfObjOrientation(mfArray handle);
void mfObjOrientation(mfArray handle, mfArray orientation);
```

Descriptions

Function `mfObjOrientation` sets the WXYZ orientation of the draw object as a vector of x, y and z rotation. The ordering in which these rotations are performed is Rotate z, Rotate x and then Rotate y.

`orientation = mfObjOrientation(handle)`

It can also be used as an inquiry function to retrieve the orientation of the draw object if given only the handle that associates with the draw object.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfObjOrientation.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"


void main() {

    mfArray center, cubesize, h;

    center = mfV(0.0, 0.0, 0.0);
    cubesize = mfV(0.2, 0.3, 0.4);

    h = mfCube(center, cubesize, "c");
    mfAxis("equal");
    mfAxis(-0.3, 0.3, -0.4, 0.4, -0.5, 0.5);
    mfViewPause();
    mfObjOrientation(h, 15, 15, 15);
    mfViewPause();

}
```

See Also

Camera Manipulation

mfView

Viewpoint specification.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfView(int az, int el);
void mfView(mfArray az_el);
void mfView(char* mode);
```

Descriptions

Function mfView specifies the orientation of an axis object. The orientation of the axis object is determined by the azimuth `az` and elevation `el` of the viewing angle from a viewpoint or can be determined by the setting the view mode.

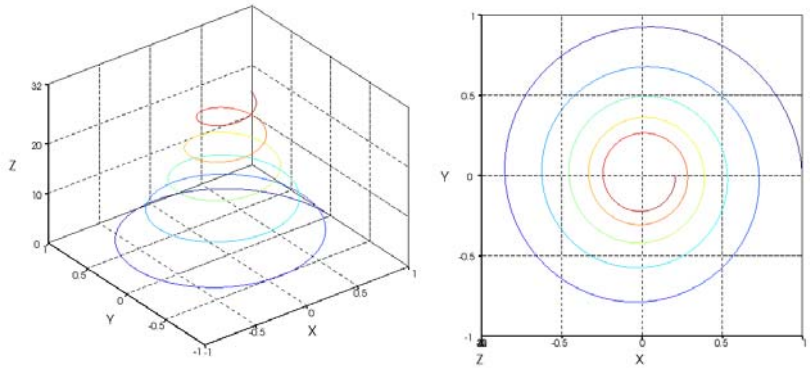
Argument `mode` can be `"2"`, `"3"`, `"home"`, `"top"`, `"bottom"`, `"front"`, `"back"`, `"left"` or `"right"`.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfView.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray x, y, z;

    z = mfLinspace(0, 10*MF_PI, 315);
    x = mfExp(-z/20)*mfCos(z);
    y = mfExp(-z/20)*mfSin(z);

    // Plot a 3-D line graph using mfPlot3() routine and title it.
    mfSubplot(1, 2, 1);
    mfPlot3(x, y, z);
    mfAxis(mfV(-1, 1, -1, 1, 0, 32));

    mfSubplot(1, 2, 2);
    mfPlot3(x, y, z);
    mfView(0, 90);
    mfAxis(mfV(-1, 1, -1, 1, 0, 32));

    // Pauses the program to display graph.
    mfViewPause();

}
```

Result



See Also

mfCamZoom

Zoom in/out.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfCamZoom(int zf);
```

Descriptions

Function `mfCamZoom` zooms the displaying object in or out. In perspective mode, it decreases the view angle by the specified zoom factor `zf.` In parallel mode, it decreases the parallel scale by the specified zoom factor `zf.` A value greater than 1 is a zoom-in whereas a value less than 1 is a zoom-out.

See Also

mfCamPan

Move the camera horizontally and vertically.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfCamPan(int dx, int dy);
```

Descriptions

Function `mfCamPan` moves the camera along the horizon and vertical. Arguments `dx` and `dy` specify the horizontal and vertical distances of the displacement respectively.

See Also

mfview, mfCamZoom

mfCamProj

Set the camera projection mode.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfCamProj();
void mfCamProj(char* mode);
```

Descriptions

Function `mfCamProj` sets the camera projection mode to be either perspective or parallel projection. Argument `mode` can be `"orthographic"` or `"perspective"`.

`mode = mfCamProj()`

- It can also be used an inquiry function to retrieve the camera projection mode. The output argument is a logical mfArray whose value is true if the projection mode is orthographic, false otherwise.

See Also

mfView, mfCamZoom

Linear Graphs

mfPlot

Two-dimensional linear graphs.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfPlot(mfArray y[, char* linespec]);
mfArray mfPlot(mfArray x, mfArray y[, char* linespec]);
mfArray mfPlot(mfArray x1, mfArray y1, char* linespec1, mfArray x2,
mfArray y2, char* linespec2, ...);
void mfPlot(mfOutArray OutArray, mfArray x1, mfArray y1, char* linespec1,
mfArray x2, mfArray y2, char* linespec2, ...);
```

Descriptions

Function `mfPlot` generates two-dimensional line graphs.

```
mfPlot(y)
mfPlot(y, linespec)
```

- Plot elements of vector `y` against their indices. If `y` is a matrix, multiple lines are plotted from each column of `y.`
- Argument `linespec` contains special characters that specify line color and marker type of the graph. For example, `"yo",` specifies a graph drawn from yellow-colored, circular markers. `linespec` can be an mfArray containing the special characters or a character string. Refer to linespec for a list of special characters applicable for line spec.

Linespec

The table below lists the `linespec` characters.

Character	Line color	Character	Marker Type	Character	Line Type
y	yellow	.	point	-	solid
m	magenta	o	circle	:	dotted
c	cyan	x	x-mark	-.	dashdot
r	red	+	plus	--	dashed

g	green	*	star	
b	blue	s	square	
w	white	d	diamond	
k	black	v	triangle down	
		^	triangle up	
		<	triangle left	
		>	triangle right	
		p	pentagram	
		h	hexagram	

`mfPlot(mfOut(h1, h2, h3, ...), x1, y1, linespec1, x2, y2, linespec2, ...)`

- Handles `h1, h2, h3,` ... retrieve the handles to the plot objects created by `mfPlot(...)` respectively.
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the plot object through handle `h` with function `mfGSet.`

The property available is:

1. linespec

See Also

mfPlot3

mfPlot3

Three-dimensional linear graphs.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfPlot3(mfArray x, mfArray y, mfArray z[, mfArray c]);
mfArray mfPlot3(mfArray xyz[, mfArray c]);
void mfPlot3(...)
```

Descriptions

Function `mfPlot3` draws three-dimensional linear graphs.

```
mfPlot3(x, y, z)
mfPlot3(x, y, z, c)
```

- If arguments `x, y` and `z` are vectors, `mfPlot3` draws a line whose x-, y-, and z-coordinates are elements of arguments `x, y` and `z` respectively.
- If arguments `x, y` and `z` are matrices, `mfPlot3` draws multiple lines from the columns of `x, y` and `z` matrices.
- Shape of each argument must be conformed.
- Complex data are not supported.
- Argument `c` contains the corresponding scalar values of the corresponding coordinates `x, y` and `z.` By default, `c` = `z.`

```
mfPlot3(xyz)
mfPlot3(xyz, c)
```

- Vertex vectors are defined in the n-by-3 matrix `xyz.`

```
h = mfPlot3(...)
```

- Handle `h` retrieves a handle to the three-dimensional linear graph objects created by `mfPlot3(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()`to retrieve the handle of the current graphics object.

You can specify properties of the graphics objects through handle `h` with function `mfGSet.`

The property available is:

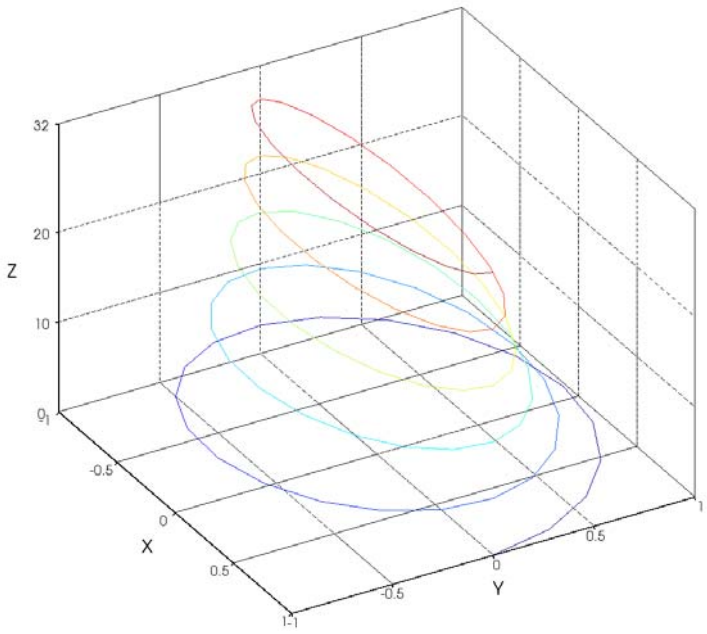
1. xyz: Vertex vectors.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfPlot3.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray x, y, z;

    z = mfLinspace(0, 10*MF_PI, 101);
    x = mfCos(z);
    y = mfExp(-z/20)*mfSin(z);

    mfAxis(mfV(-1.0, 1.0, -1.0, 1.0, 0.0, 32.0));

    // Plot the three-dimensional graph
    mfPlot3(x, y, z);

    // Specify the viewpoint
    mfView(60,30);

    // Pause program for graphics display
    mfViewPause();
}
```

Result



See Also

mfViewPause, mfAxis, mfSubplot, mfView, mfSurf, mfMesh

mfRibbon

Three-dimensional ribbons.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfRibbon(mfArray x, mfArray y, mfArray z[, mfArray c]);
mfArray mfRibbon(mfArray xyz[, mfArray c]);
```

Descriptions

Function `mfRibbon` draws three-dimensional ribbons.

```
mfRibbon(x, y, z)
mfRibbon(x, y, z, c)
```

- If arguments `x, y` and `z` are vectors, `mfRibbon` draws a ribbon whose x-, y-, and z-coordinates are elements of arguments `x, y` and `z` respectively.
- If arguments `x, y` and `z` are matrices, `mfRibbon` draws multiple ribbons from the columns of `x, y` and `z` matrices.
- Shape of each argument must be conformed.
- Complex data are not supported.
- Argument `c` contains the corresponding scalar values of the corresponding coordinates `x, y` and `z.` By default, `c` = `z.`

```
mfRibbon(xyz)
mfRibbon(xyz, c)
```

- Vertex vectors are defined in the n-by-3 matrix `xyz.`

```
h = mfRibbon(...)
```

- Handle `h` retrieves a handle to the three-dimensional ribbon objects created by `mfRibbon(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the graphics objects through handle `h` with function `mfGSet.`

The properties available are:

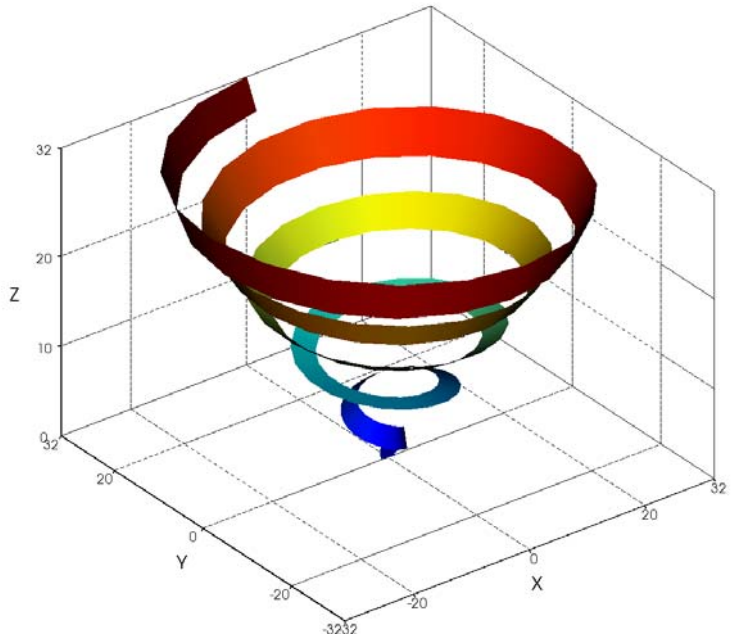
1. sizefactor: The width of the ribbon objects. By default, sizefactor is 1.
2. xyz: Vertex vectors.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfRibbon.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray x, y, z;

    z = mfLinspace(0, 10*MF_PI, 100);
    x = z*mfSin(z);
    y = z*mfCos(z);

    mfRibbon(x, y, z);
    mfAxis(-32, 32, -32, 32, 0, 32);
    mfViewPause();

}
```

Result



See Also

mfTube

Three-dimensional tubes.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfTube(mfArray x, mfArray y, mfArray z[, mfArray c]);
mfArray mfTube(mfArray xyz[, mfArray c]);
```

Descriptions

Function `mfTube` draws three-dimensional tubes.

```
mfTube(x, y, z)
mfTube(x, y, z, c)
```

- If arguments `x, y` and `z` are vectors, `mfTube` draws a tube whose x-, y-, and z-coordinates are elements of arguments `x, y` and `z` respectively.
- If arguments `x, y` and `z` are matrices, `mfTube` draws multiple tubes from the columns of `x, y` and `z` matrices.
- Shape of each argument must be conformed.
- Complex data are not supported.
- Argument `c` contains the corresponding scalar values of the corresponding coordinates `x, y` and `z.` By default, `c` = `z.`

```
mfTube(xyz)
mfTube(xyz, c)
```

- Vertex vectors are defined in the n-by-3 matrix `xyz.`

```
h = mfTube(...)
```

- Handle `h` retrieves a handle to the three-dimensional tube objects created by `mfTube(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the graphics objects through handle `h` with function `mfGSet.`

The properties available are:

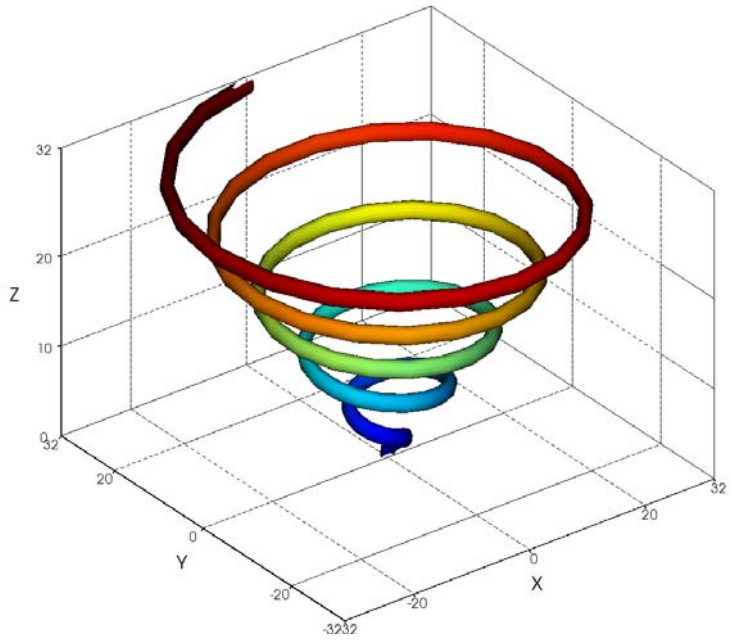
1. sizefactor: The diameter of the tube objects. By default, sizefactor is 1.
2. xyz: Vertex vectors.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfTube.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray x, y, z, h;

    z = mfLinspace(0, 10*MF_PI, 100);
    x = z*mfSin(z);
    y = z*mfCos(z);

    h = mfTube(x, y, z);
    mfAxis(-32, 32, -32, 32, 0, 32);
    mfViewPause();

}
```

Result



See Also

Surface Graphs

mfSurf

Surface plot.

Module

```
fgl.h/fgl.lib
```

C++ Prototype

```
mfArray mfSurf(mfArray x, mfArray y, mfArray z[, mfArray c]);
mfArray mfSurf(mfArray z[, mfArray c]);
```

Descriptions

Function `mfSurf` creates three-dimensional graphs composed of colored quadrilateral surfaces. You can choose several shading options including mesh, flat, faceted, and interpolated. The options can be set using function `mfShading` or through the menu and toolbar functions of the Graphics Viewer. Note that mesh surface can also be plotted function `mfMesh.`

```
mfSurf(x, y, z)
mfSurf(x, y, z, c)
```

- Plot surface objects from arguments `x, y` and `z.` The arguments `x, y` and `z` contain the x-, y-, and z- coordinates respectively of the surface object's grid intersections.
- If `x, y` and `z` are matrices, their shapes should conform. The grid intersections are given by (x(i,j), y(i,j), z(i,j)).
- If `x` and `y` are vectors, and Shape(z)=[m, n], then if mfLength(x) = n, and mfLength(y)= m, the grid intersections are given by (x(i), y(j), z(i,j)).
- By default, the edge color of the wire-frame grid is proportional to the `z` coordinates of the surface object. Specifying argument `c` would override the default color scale.
- By default, `mfSurf` draws surface with faceted shading.

```
mfSurf(z)
mfSurf(z, c)
```

- Create a three-dimensional surface from the m-by-n matrix `z.` Arguments `x` and `y` are set to the default `x=mfColon(1, n)`, `y=mfColon(1, m)`. Argument `z` is a single-valued function defined over a rectangular grid formed by `x` and `y.`

```
h = mfSurf(...)
```

- Handle `h` retrieves a handle to the surface object created by `mfSurf(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()`to retrieve the handle of the current graphics object.

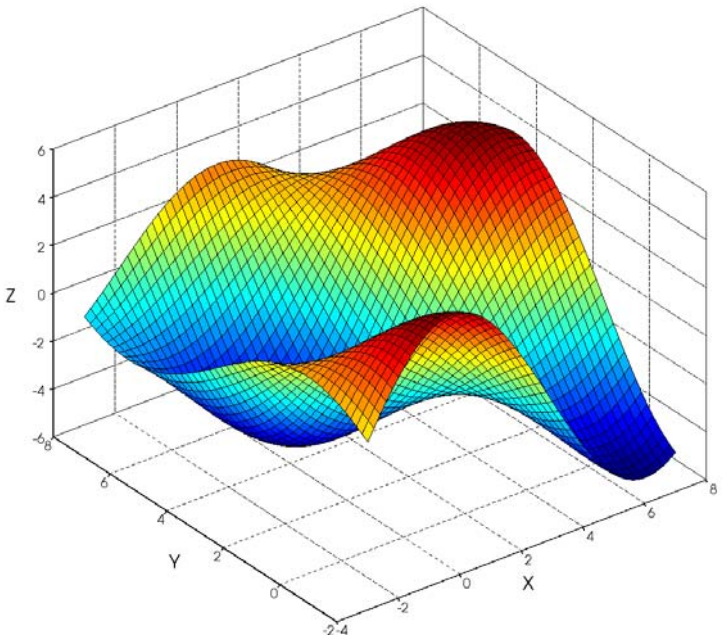
You can specify properties of the surface objects through handle h with function `mfGSet.`

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfSurf.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray a, b, c, x, y, z, indxi, indxj;

    a = mfLinspace(-3, 7, 51);
    b = mfLinspace(-2, 8, 51);
    c = mfColon(1, 51);
    mfMeshgrid(mfOut(x, y), a, b);
    mfMeshgrid(mfOut(indxi, indxj), c);
    z = 3*mfSin((indxi+1)/10)*mfCos((indxj+1)/10)
        + 2*mfSin((indxi+indxj)/10);

    // Plot a surf using mfArray x, y and z
    mfSurf(x, y, z);

    // Pause to display the graph
    mfViewPause();

}
```

Result



See Also

mfViewPause, mfAxis, mfSubplot, mfView, mfMesh, mfPlot3

mfMesh

Mesh plot.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfMesh(mfArray x, mfArray y, mfArray z[, mfArray c]);
mfArray mfMesh(mfArray z[, mfArray c]);
```

Descriptions

Function `mfMesh` plots a three-dimensional mesh surface consisting of criss-crossed lines that looks like a net draped over the surface defined by your data.

```
mfMesh(x, y, z)
mfMesh(x, y, z, c)
```

- Plot surface objects from arguments `x, y` and `z.` The arguments `x, y` and `z` contain the x-, y-, and z- coordinates respectively of the surface object's grid intersections.
- If `x, y` and `z` are matrices, their shapes should conform. The grid intersections are given by (x(i,j), y(i,j), z(i,j)).
- If `x` and `y` are vectors, and Shape(z)=[m, n], then if mfLength(x) = n, and mfLength(y)= m, the grid intersections are given by (x(i), y(j), z(i,j)).
- By default, the edge color of the wire-frame grid is proportional to the `z` coordinates of the surface object. Specifying argument `c` would override the default color scale.

```
mfMesh(z)
mfMesh(z, c)
```

- Create a three-dimensional surface from the m-by-n matrix `z.` Arguments `x` and `y` are set to the default `x=mfColon(1, n)`, `y=mfColon(1, m)`. Argument `z` is a single-valued function defined over a rectangular grid formed by `x` and `y.`

```
h = mfMesh(...)
```

- Handle `h` retrieves a handle to the mesh surface object created by `mfMesh(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()`to retrieve the handle of the current graphics object.

You can specify properties of the surface objects through handle `h` with function `mfGSet.`

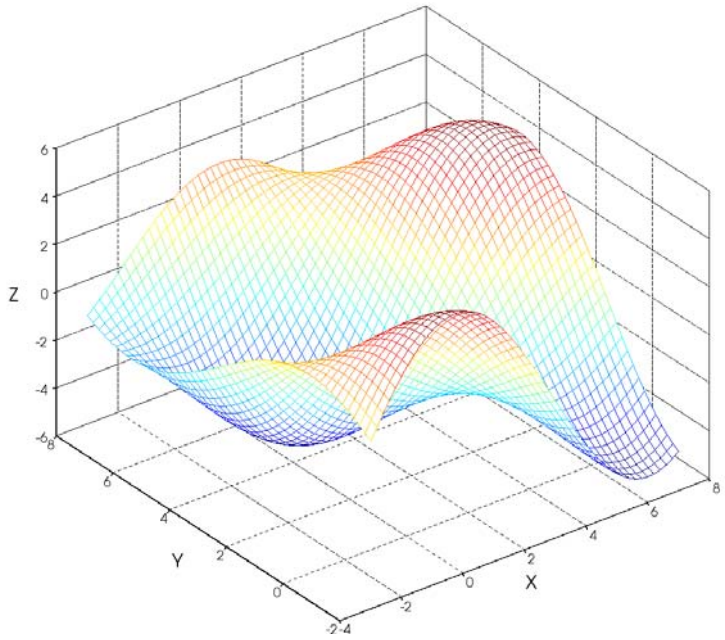
Example

Code

```
// ************************************************
```

```
// AnCAD example file
// file: mfMesh.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"


void main() {

    mfArray a, b, c, x, y, z, indxi, indxj;

    a = mfLinspace(-3, 7, 51);
    b = mfLinspace(-2, 8, 51);
    c = mfColon(1, 51);
    mfMeshgrid(mfOut(x, y), a, b);
    mfMeshgrid(mfOut(indxi, indxj), c);
    z = 3*mfSin((indxi+1)/10)*mfCos((indxj+1)/10)
        + 2*mfSin((indxi+indxj)/10);

    // Plot a mesh grid using mfArray x, y and z for the grid
    // intersections.
    mfMesh(x, y, z);

    // Pause to display the graph
    mfViewPause();
}
```

Result



See Also

mfSurfc

Combined plot of surface and contour3.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfSurfc(mfArray x, mfArray y, mfArray z[, mfArray c]);
mfArray mfSurfc(mfArray z[, mfArray c]);
mfSurfc(mfOutArray OutArray, mfArray x, mfArray y, mfArray z[, mfArray
c]);
mfSurfc(mfOutArray OutArray, mfArray z[, mfArray c]);
```

Descriptions

Function `mfSurfc` draws a contour below the surface object.

`mfSurfc(z)` draws a contour below the surface object created by `mfSurf(z)`. `mfSurfc(x, y, z)` draws a contour below the surface object created by `mfSurf(x, y, z)`.

`mfSurfc(mfOut(h1, h2), ...)`

- Handles `h1` and `h2` retrieve the handles to the surface object and contour object that are created by `mfSurfc(...)` respectively.
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the surface object through handle `h` with function `mfGSet`.

For the properties, see the description on `mfSurf`.

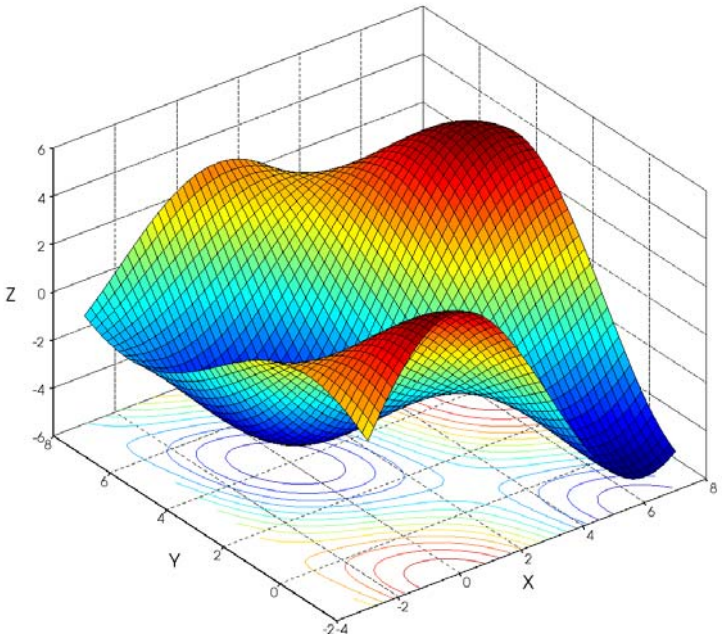
Example

Code

```
// ************************************************
// AnCAD example file
// file: mfSurfc.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray a, b, c, x, y, z, indxi, indxj;

    a = mfLinspace(-3, 7, 51);
    b = mfLinspace(-2, 8, 51);
    c = mfColon(1, 51);
```

```
    mfMeshgrid(mfOut(x, y), a, b);
    mfMeshgrid(mfOut(indxi, indxj), c);
    z = 3*mfSin((indxi+1)/10)*mfCos((indxj+1)/10)
        + 2*mfSin((indxi+indxj)/10);

    // Plot a surf using mfArray x, y and z and Plot Contour below the
surf
    mfSurfc(x, y, z);

    // Pause to display the graph
    mfViewPause();

}
```

Result



See Also

mfSurf

mfMeshc

Combined plot of mesh and contour3.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfMeshc(mfArray x, mfArray y, mfArray z[, mfArray c]);
mfArray mfMeshc(mfArray z[, mfArray c]);
mfMeshc(mfOutArray OutArray, mfArray x, mfArray y, mfArray z[, mfArray
c]);
mfMeshc(mfOutArray OutArray, mfArray z[, mfArray c]);
```

Descriptions

Function `mfMeshc` draws a contour below the meshed surface object.

`mfMeshc(z)` draws a contour below the meshed surface object created by `mfSurf(z)`. `mfMeshc(x, y, z)` draws a contour below the meshed surface object created by `mfSurf(x, y, z)`.

`mfSurfc(mfOut(h1, h2), ...)`

- Handles `h1` and `h2` retrieve the handles to the meshed surface object and contour object that are created by `mfMeshc(...)` respectively.
- If only one handle `h` is given, the function returns the two handles in a vector mfArray.
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the surface object through handle `h` with function `mfGSet`.

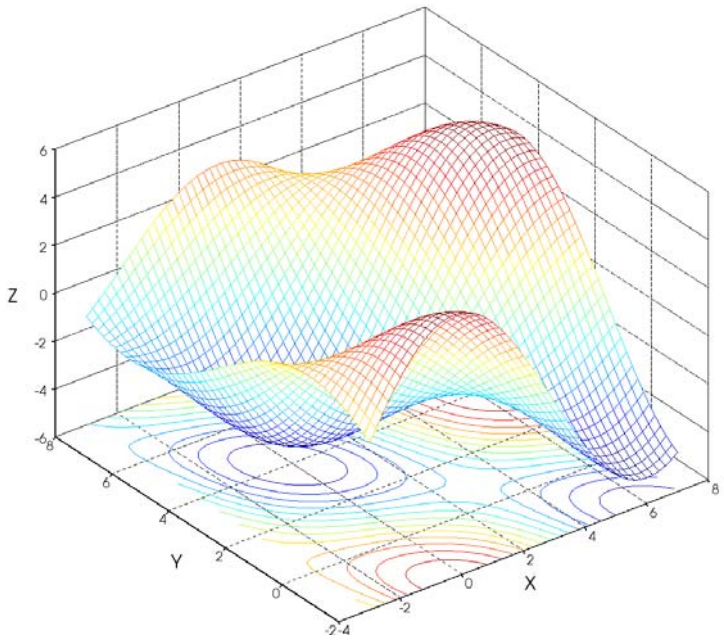
Example

Code

```
// ************************************************
// AnCAD example file
// file: mfMeshc.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray a, b, c, x, y, z, indxi, indxj;

    a = mfLinspace(-3, 7, 51);
    b = mfLinspace(-2, 8, 51);
```

```
    c = mfColon(1, 51);
    mfMeshgrid(mfOut(x, y), a, b);
    mfMeshgrid(mfOut(indxi, indxj), c);
    z = 3*mfSin((indxi+1)/10)*mfCos((indxj+1)/10)
        + 2*mfSin((indxi+indxj)/10);

    // Plot a mesh grid using mfArray x, y and z for the grid
    // intersections and Plot Contour below the mesh
    mfMeshc(x, y, z);

    // Pause to display the graph
    mfViewPause();

}
```

Result



See Also

mfMesh

mfPColor

pseudocolor plot.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfPColor(mfArray c);
mfArray mfPColor(mfArray x, mfArray y, mfArray c);
```

Descriptions

Function `mfPColor` produces pseudocolor plot of a matrix mfArray `c` by mapping the elements of `c` to the current colormap. This function is equivalent to a top-view of `mfSurf.`

`mfPColor(c)`

- The function displays matrix mfArray `c` as a checker-board plot with elements of `c` specifying each cell of the plot, mapped to the index of the current colormap.
- The smallest and largest elements of matrix `c` correspond to the minimum and maximum indices of the colormap.
- By default, the shading is `"faceted",` with each cell containing a constant color. Each element of matrix `c` specifies the color of a rectangular patch of the image.

`mfPColor(x, y, c)`

- It plots the checker-board plot on the grid defined by arguments `x` and `y.` The arguments `x` and `y` can be vectors or matrices.

`h = mfPColor(...)`

- Handle `h` retrieves a handle to the pseudocolor plot object created by `mfPColor(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the pseudocolor plot object through handle `h` with function `mfGSet.`

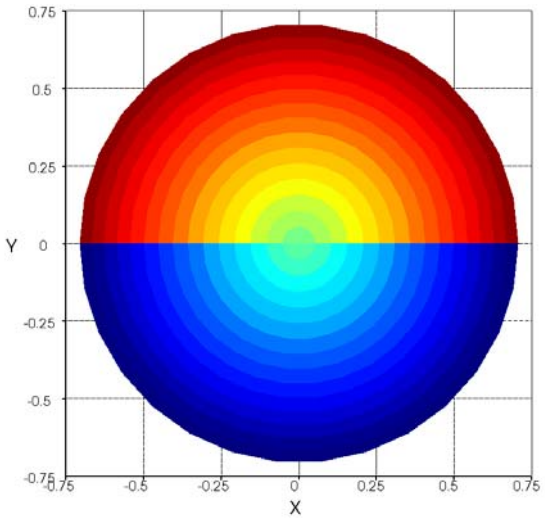
Example

Code

```
// ************************************************
// AnCAD example file
// file: mfPColor.cpp
//
```

```
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"


void main() {

    mfArray theta, phi, x, y, z;

    theta = mfLinspace(-MF_PI, MF_PI, 31);
    phi = mfT(theta /4);
    x = mfSin(phi).Mul(mfCos(theta));
    y = mfSin(phi).Mul(mfSin(theta));
    z = mfSin(phi).Mul(mfOnes(1,31));

    // Plot the surface object using X, Y and Z
    mfPColor(x, y, z);
    mfAxis("equal");
    mfCamZoom(0.8);

    // Pause the program for display
    mfViewPause();


}
```

Result



See Also

mfSurf

Module

.h/.lib

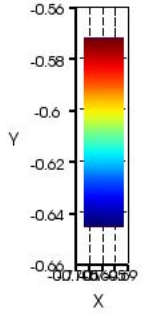
Descriptions

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfPColor.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"


void main() {

    mfArray theta, phi, x, y, z, extent;

    theta = mfLinspace(-MF_PI, MF_PI, 31);
    phi = mfT(theta /4);
    x = mfSin(phi).Mul(mfCos(theta));
    y = mfSin(phi).Mul(mfSin(theta));
    z = mfSin(phi).Mul(mfOnes(1,31));
    extent = mfHCat(mfMin(x), mfMax(x), mfMin(y), mfMax(y));

    // Plot the surface object using X, Y and Z
    mfFastPColor(z, extent);
    mfAxis("equal");
    mfCamZoom(0.8);

    // Pause the program for display
    mfViewPause();


}
```

Result



See Also

mfContour

Two-dimensional contour.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfContour(mfArray x, mfArray y, mfArray c);
mfArray mfContour(mfArray c);
```

Descriptions

Function `mfContour` plots constant value lines in two-dimensional space.

`handle = mfContour(x, y, c)`

- Generate two-dimensional contour lines of matrix `c.` The values plotted are selected automatically.
- Argument `c` is assumed to contain data representing the scalar values.
- Arguments `x` and `y` specify the corresponding x, y coordinates of the scalar values.
- Similar to `mfSurf` and `mfMesh,` the edge colors of the contour lines are selected based on the current colormap.

`mfContour(c)`

- The scalar values `c` is assumed to be defined over a geometrically rectangular grid where `x = mfColon(1, n)` and `y = mfColon(1, m)`

.

`h = mfContour(...)`

- Handle `h` retrieves a handle to the contour object created by `mfContour(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the contour object through handle `h` with function `mfGSet.`

See mfContour3 for available properties.

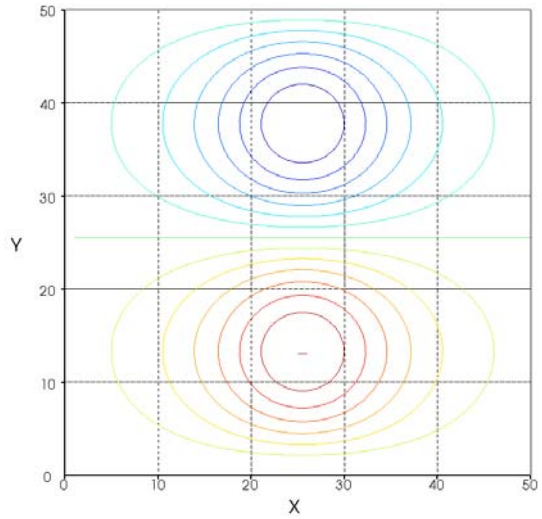
Example

Code

```
// ************************************************
// AnCAD example file
// file: mfContour.cpp
//
// Copyright 2003 AnCAD
// ************************************************
```

```
#include "cml.h"
#include "fgl.h"


void main() {

    mfArray a, x, y, z;

    a = mfLinspace(-MF_PI, MF_PI, 50);
    mfMeshgrid(mfOut(x, y) ,a);
    z = (1/mfCosh(x))*mfCos(y+MF_PI/2);

    // Draw a 2-D contour plot
    mfContour(z);
    mfAxis("equal");
    mfCamZoom(0.8);

    // Pause the program to display the graphics
    mfViewPause();
}
```

Result



See Also

mfSolidContour

mfContour3

Three-dimensional Line Contour.

Module

fgl.h/fgl.lib

C++ Prototype

```
mfArray mfContour3(mfArray x, mfArray y, mfArray z[, mfArray c]);
mfArray mfContour3(mfArray z[, mfArray c]);
```

Descriptions

Function mfContour3 plots constant value lines of matrix z in three-dimensional space. The values plotted are selected automatically.

```
mfContour3(x, y, z, c)
mfContour3(z, c)
```

- This function is similar to mfContour. The contour lines are presented in three-dimensional perspective reflecting their scalar values. The values plotted are selected automatically.

```
h = mfContour3(...)
```

- Handle h retrieves a handle to the three-dimensional contour object created by mfContour3(...).
- Alternatively, you use function h = mfGetCurrentDraw()to retrieve the handle of the current graphics object.

You can specify properties of the three-dimensional contour object through handle h with function mfGSet.

The properties available are:

1. iso: iso-values, a vector containing iso-value set. Setting this property will replace default set of contour lines.
2. autolevel: given number of levels, it will generate the iso-value set automatically.
3. label: "on" or "off"

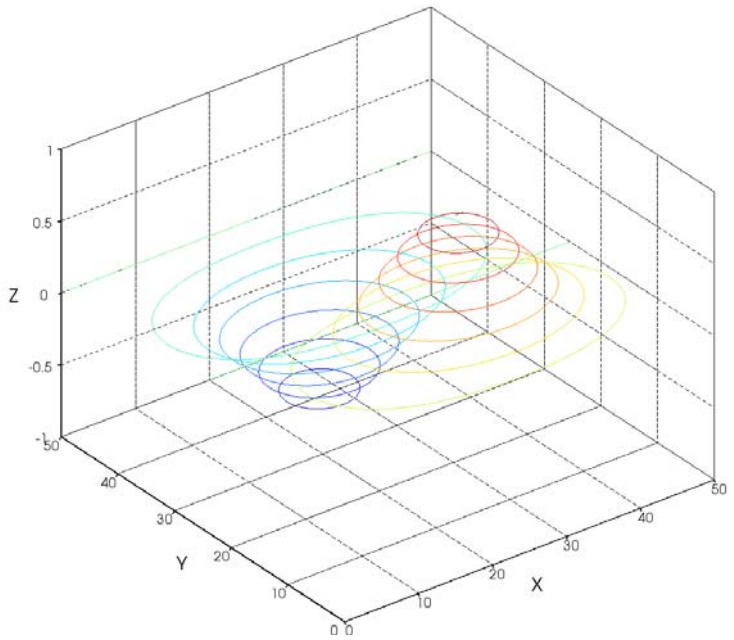
Example

Code

```
// ************************************************
// AnCAD example file
// file: mfContour3.cpp
//
// Copyright 2003 AnCAD
```

```
// ************************************************

#include "cml.h"
#include "fgl.h"


void main() {

    mfArray a, x, y, z;

    a = mfLinspace(-MF_PI, MF_PI, 50);
    mfMeshgrid(mfOut(x, y) ,a);
    z = (1/mfCosh(x))*mfCos(y+MF_PI/2);

    // Draw a 3-D contour plot
    mfContour3(z);

    // Pause the program to display the graphics
    mfViewPause();

}
```

Result



See Also

mfContour, mfSolidContour, mfSolidContour3, mfTriContour

mfSolidContour

Two-dimensional solid contour.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfSolidContour(mfArray x, mfArray y, mfArray c);
mfArray mfSolidContour(mfArray c);
```

Descriptions

Function `mfSolidContour` fills colors in the areas that are in between constant value lines in two-dimensional space.

`handle = mfSolidContour(x, y, c)`

- The values plotted are selected automatically.
- Argument `c` is assumed to contain data representing the scalar values.
- Arguments `x` and `y` specify the corresponding x, y coordinates of the scalar values.
- Similar to `mfSurf` and `mfMesh,` the surface colors are selected based on the current colormap.

`mfSolidContour(c)`

- The scalar values `c` is assumed to be defined over a geometrically rectangular grid where `x = mfColon(1, n)` and `y = mfColon(1, m)`

.

`h = mfSolidContour(...)`

- Handle `h` retrieves a handle to the contour object created by `mfSolidContour(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the contour object through handle `h` with function `mfGSet.`

See mfSolidContour3 for available properties.

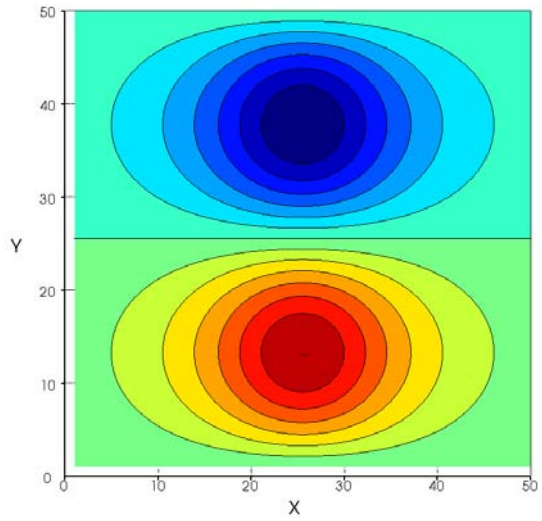
Example

Code

```
// ************************************************
// AnCAD example file
// file: mfSolidContour.cpp
//
// Copyright 2003 AnCAD
```

```
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray a, x, y, z;

    a = mfLinspace(-MF_PI, MF_PI, 50);
    mfMeshgrid(mfOut(x, y) ,a);
    z = (1/mfCosh(x))*mfCos(y+MF_PI/2);

    // Draw a 2-D solidcontour
    mfSolidContour(z);
    mfAxis("equal");
    mfCamZoom(0.80);

    // Pause the program to display the graphics
    mfViewPause();

}
```

Result



See Also

mfSoildContour3

Three-dimensional solid Contour.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfSolidContour3(mfArray x, mfArray y, mfArray z[, mfArray c]);
mfArray mfSolidContour3(mfArray z[, mfArray c]);
```

Descriptions

Function `mfSoildContour3` fills colors in the areas that are in between the constant value lines in three-dimensional space.

```
mfSolidContour3(x, y, z, c)
mfSolidContour3(z, c)
```

- This function is similar to `mfSolidContour.` The areas are presented in three-dimensional perspective reflecting their scalar values specified in argument `c.`

```
h = mfSolidContour3(...)
```

- Handle `h` retrieves a handle to the three-dimensional contour object created by `mfSolidContour3(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the three-dimensional contour object through handle `h` with function `mfGSet.`

The properties available are:

1. iso: iso-values, a vector containing iso-value sets.
2. autolevel: given number of levels, it will generate iso-value sets automatically.
3. label: `"on"` or `"off"`

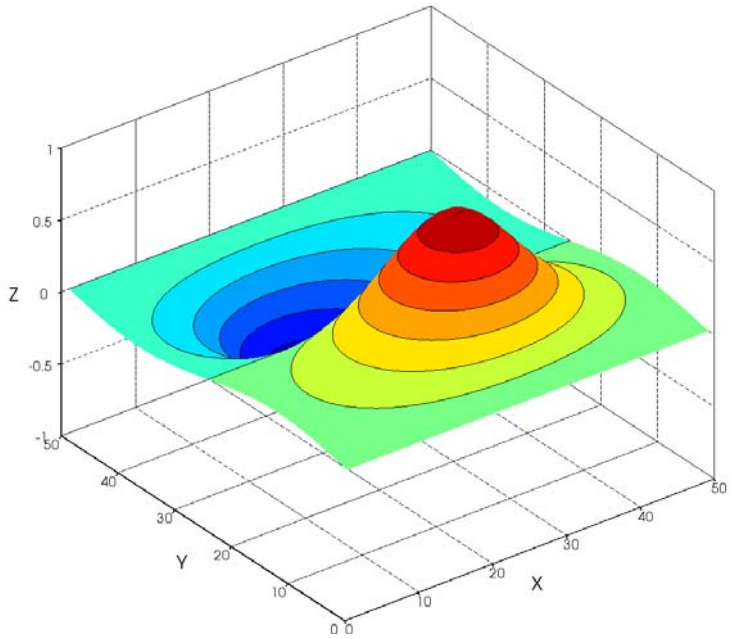
Example

Code

```
// ************************************************
// AnCAD example file
// file: mfSolidContour3.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
```

```
#include "fgl.h"

void main() {

    mfArray a, x, y, z;

    a = mfLinspace(-MF_PI, MF_PI, 50);
    mfMeshgrid(mfOut(x, y) ,a);
    z = (1/mfCosh(x))*mfCos(y+MF_PI/2);

    // Draw a 3-D solidcontour
    mfSolidContour3(z);

    // Pause the program to display the graphics
    mfViewPause();

}
```

Result



See Also

mfContour, mfContour3, mfSolidContour, mfSolidContour, mfTriContour

mfOutline

Wireframe outline boundary.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfOutline(mfArray x, mfArray y, mfArray z);
mfArray mfOutline(mfArray z);
```

Descriptions

Function `mfOutline` generates wireframe outline boundary for a given data set. Arguments `x, y` and `z` specify the corresponding coordinates of the points in the data set.

`handle = mfOutline(...)`

- Handle `h` retrieves a handle to the outline object created by `mfOutline(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the outline object through handle `h` with function `mfGSet.`

See Also

mfIsoSurface

Three-dimensional iso-value surface plot from volumetric data.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfIsoSurface(mfArray x, mfArray y, mfArray z, mfArray c, mfArray isovalue);
mfArray mfIsoSurface(mfArray c, mfArray isovalue);
```

Descriptions

Function `mfIsoSurface` creates 3-D graphs composed of isosurface data from the volumetric data `c` at the isosurface value specified in argument `isovalue.`

`mfIsoSurface(x, y, z, c, isovalue)`

- The arguments `x, y` and `z` define the coordinates for the volume `c.`

`mfIsoSurface(c, isovalue)`

- The coordinates for the volume `c` is of a geometrically rectangular grid where `x = mfColon(1, n)` and `y = mfColon(1, m) and z = mfColon(1, p)`.

`h = mfIsoSurface(...)`

- Handle `h` retrieves a handle to the isosurface object created by `mfIsoSurface(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the isosurface object through handle `h` with function `mfGSet.`

The properties available are:

1. iso: iso-value, a vector containing iso-value sets.

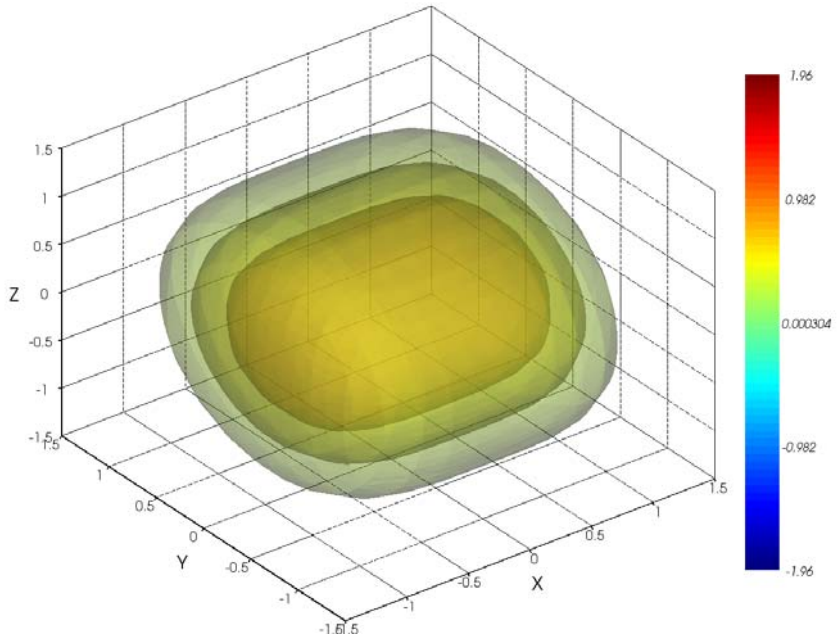
Example

Code

```
// ************************************************
// AnCAD example file
// file: mfIsoSurface.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
```

```
#include "fgl.h"

void main() {

    mfArray x, y, z, v, a, b, c, h;

    a = mfLinspace(-2, 2.2, 21);
    b = mfLinspace(-2, 2.25, 17);
    c = mfLinspace(-1.5, 1.6, 31);
    mfMeshgrid(mfOut(y, x, z), b, a, c);
    v = 2*mfCos(x*x)*mfExp(-(y*y)-(z*z));

    // Plot IsoSurface and Set Transparency to 70
    h = mfIsoSurface(x, y, z, v, mfV(1.0, 0.6, 0.3));
    mfDrawMaterial(h, "surf", "trans", 70);

    // Set Colorbar
    mfColorbar("on");

    // Pause the program for display
    mfViewPause();

}
```

Result



See Also

mfAxis, mfSubplot, mfView, mfMesh, mfSurf

Slice Graphs

mfSliceXYZ

Display orthogonal slice-planes through volumetric data.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfSliceXYZ([mfArray x, mfArray y, mfArray z, ]mfArray c, mfArray Sx,
mfArray Sy, mfArray Sz);
mfArray mfSliceXYZ([mfArray x, mfArray y, mfArray z, ]mfArray c, mfArray
Sx, mfArray Sy, mfArray Sz);
```

Descriptions

Function mfSliceXYZ displays orthogonal slice-planes of a specified set of volumetric data. The information on the slice-planes can be retrieved by using function `mfGetSliceXYZ.`

`mfSliceXYZ(x, y, z, c, Sx, Sy, Sz)`

- Displays orthogonal slice-planes along the x, y and z directions specified by points in vector mfArrays `Sx, Sy` and `Sz`
- Use `mf()` to substitute any of the direction vectors `Sx, Sy` or `Sz` if you are not drawing any slice along the respective direction. E.g. mfSliceXYZ(x, y, z, v, Sx, mf(), Sz) draws slices along the x-axis as specified by `Sx` and z-axis as specified by `Sz.`
- The arguments `x, y` and `z` define the corresponding coordinates of scalar values mfArray `c,` where `c` is an m-by-n-by-p three-dimensional array.
- The arguments `x, y` and `z` must be of the same shape as `c` and are monotonic and three-dimensional plaid as if produced by function `mfMeshgrid.`
- The color at each point is determined by three-dimensional interpolation into the elements of volume `c,` mapped to the current colormap.

`mfSliceXYZ(c, Sx, Sy, Sz)`

- Assumes `x` is composed of mfColon(1, n) vectors, `y` is composed of mfColon(1, m) vectors and `z` is composed of mfColon(1, p) vectors.

`h = mfSliceXYZ(...)`

- Handle `h` retrieves a handle to the volumetric slice object created by `mfSliceXYZ(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the volumetric slice objects through handle `h` with function

`mfGSet.`

The properties available are:

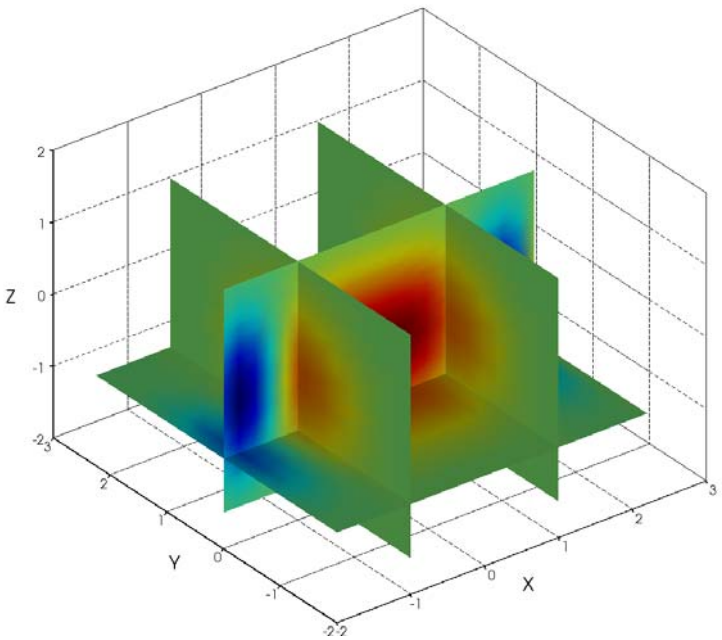
1. slicex: specifies the slice-planes along the x direction.
2. slicey: specifies the slice-planes along the y direction.
3. slicez: specifies the slice-planes along the z direction.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfSlicexyz.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray nx, ny, nz, x, y, z, c;

    nx = mfLinspace(-2, 2.2, 21.0);
    ny = mfLinspace(-2, 2.25, 17.0);
    nz = mfLinspace(-1.5, 1.6, 31.0);
    mfMeshgrid(mfOut(y, x, z), ny, nx, nz);
    c = 2*mfCos(x*x)*mfExp(-(y*y)-(z*z));

    mfSliceXYZ(x, y, z, c, mfV(-1.0, 1.0),  0, -0.75);
    mfViewPause();

}
```

Result



See Also

mfSliceIJK, mfSlicePlane, mfGetSliceXYZ

mfSliceIJK

Display orthogonal slice-planes along i, j or k indices.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfSliceIJK(mfArray x, mfArray y, mfArray z, mfArray c, mfArray
Si, mfArray Sj, mfArray Sk);
mfArray mfSliceIJK(mfArray c, mfArray Si, mfArray Sj, mfArray Sk);
```

Descriptions

Function `mfSliceIJK` displays slice-planes along i, j and k which are index of `x,` index of `y` and index of `z` respectively.

`mfSliceIJK(x, y, z, c, Si, Sj, Sk)`

- Displays slice-planes along arbitrary indices specified by mfArrays `Si, Sj` and `Sk.`
- Use `mf()` to substitute any of the index vectors `Si, Sj` or `Sk` if you are not drawing any slice along the respective index. For example, `mfSlice(x, y, z, c, Si, mf(), Sk)` draws slices along indices of `x` as specified by `Si` and `z` as specified by `Sz.`
- Arguments `x, y` and `z` define the corresponding coordinates of volumetric data `c,` where `c` is a m-by-n-by-p three-dimensional array.
- Arguments `x, y` and `z` must be of the same shape as `c` and are monotonic and three-dimensional plaid as if produced by function mfMeshgrid.

`h = mfSliceIJK(c, Sx, Sy, Sz)`

- Handle `h` retrieves a handle to the slice objects created by `mfSliceXYZ(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the slice objects through handle `h` with function `mfGSet.`

The properties available are:

1. slicei: specifies indices of x.
2. slicej: specifies indices of y.
3. slicek: specifies indices of z.

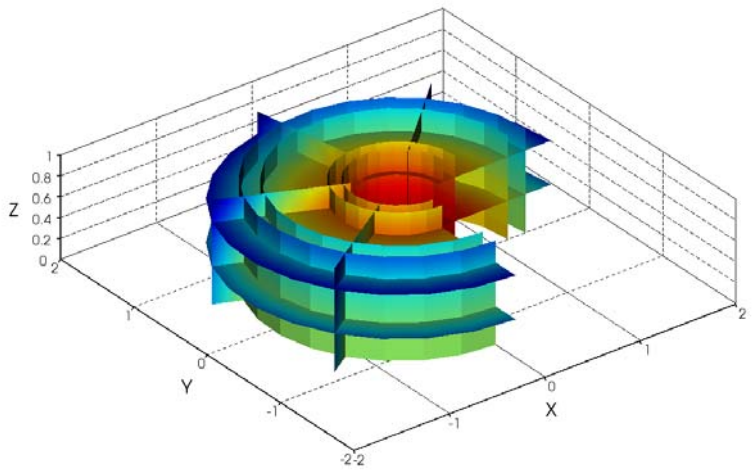
Example

Code

```
// ************************************************
```

```
// AnCAD example file
// file: mfSliceIJK.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray nu, nv, nw, u, v, w, x, y, z, c, h;

    nu = mfLinspace(0, 1.5*MF_PI, 20);
    nv = mfLinspace(0, 2*MF_PI, 20);
    nw = mfLinspace(0, 1, 20);
    mfMeshgrid(mfOut(u, v, w), nu, nv, nw );
    x = ( 1 + 0.6 * mfCos(v) ) * mfCos(u);
    y = ( 1 + 0.6 * mfCos(v) ) * mfSin(u);
    z = w;
    c = 1 - ( x * x + y * y + z * z);

    h = mfSliceIJK(x, y, z, c, mfV(4, 8, 12, 16), mfV(4, 8, 12, 16), mfV(8,
16));
    mfAxis("equal");
    mfCamZoom(1.5);
    mfViewPause();

}
```

Result



See Also

mfSlicePlane

Display orthogonal slice-planes along arbitrary direction.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfSlicePlane(mfArray x, mfArray y, mfArray z, mfArray c, mfArray plane);
mfArray mfSlicePlane(mfArray c, mfArray plane);
```

Descriptions

Function `mfSlice` displays orthogonal slice-planes of a specified set of volumetric data along arbitrary direction. The information on the slice-planes can be retrieved by using function `mfGetSlicePlane.`

`mfSlicePlane(x, y, z, c, plane)`

- Displays orthogonal slice-planes along the direction specified by `plane.`
- The arguments `x, y` and `z` are structured grid data which define the corresponding coordinates of scalar values specified in argument `c,` where `c` is a m-by-n-by-p three-dimensional array.
- Argument `plane` is a vector of size 4 representing the coefficients of the sliced plane equation, i.e. [a, b, c, d], where ax + by + cz + d = 0.

- The arguments `x, y` and `z` must be of the same shape as `c` and are monotonic and three-dimensional plaid as if produced by function `mfMeshgrid.`
- The color at each point is determined by three-dimensional interpolation into the elements of volume `c,` mapped to the current colormap.

`mfSlicePlane(c, plane)`

- Assumes `x` is composed of mfColon(1, n) vectors, `y` is composed of mfColon(1, m) vectors and `z` is composed of mfColon(1, p) vectors.

`h = mfSlicePlane(...)`

- Handle `h` retrieves a handle to the volumetric slice objects created by `mfSlicePlane(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the volumetric slice objects through handle `h` with function

`mfGSet.`

The property available is:

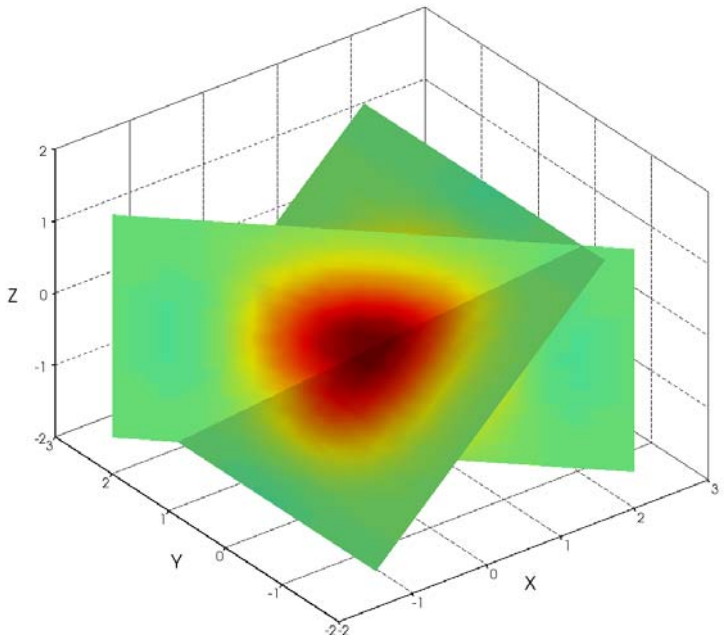
1. plane

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfSlicePlane.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray nx, ny, nz, x, y, z, c;

    nx = mfLinspace(-2, 2.2, 21);
    ny = mfLinspace(-2, 2.25, 17);
    nz = mfLinspace(-1.5, 1.6, 31);
    mfMeshgrid(mfOut(y, x, z), ny, nx, nz);
    c = 2*mfCos(x*x)*mfExp(-(y*y)-(z*z));

    mfSlicePlane(x, y, z, c, mfV(1, 0, -1, 0));
    mfHold("on");
    mfSlicePlane(x, y, z, c, mfV(1, 1, 0, 0));
    mfViewPause();

}
```

Result



See Also

mfSlice, mfGetSlicePlane

mfGetSliceXYZ

Retrieve orthogonal slice-plane(s) through volumetric data.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfGetSliceXYZ(mfOutArray OutArray, [mfArray x, mfArray y, mfArray
z, ]mfArray c, mfArray Sx, mfArray Sy, mfArray Sz);
```

Descriptions

Function `mfGetSliceXYZ` displays and retrieves orthogonal slice-planes of a specified set of volumetric data. It returns the triangular mesh `tri` and the vertex vectors of the sliced-planes defined in the n-by-3 matrix `xyz.`

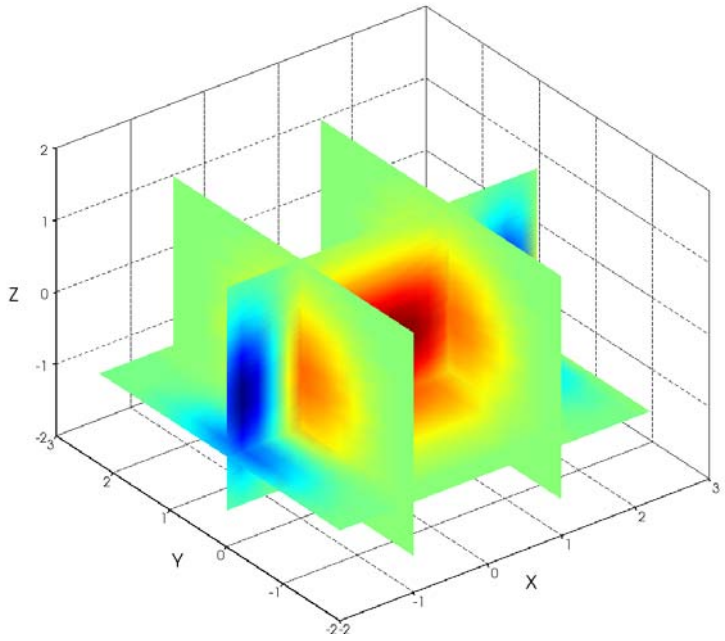
For details on the input arguments, refer to the description of function `mfSliceXYZ.`

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfGetSliceXYZ.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray nx, ny, nz, x, y, z, c, h, tri, xyz;

    nx=mfLinspace(-2, 2.2, 21);
    ny=mfLinspace(-2, 2.25, 17);
    nz=mfLinspace(-1.5, 1.6, 31);
    mfMeshgrid(mfOut(y, x, z), ny, nx, nz);
    c=2*mfCos(x*x)*mfExp(-(y*y)-(z*z));

    mfGetSliceXYZ(mfOut(tri, xyz), x, y, z, c, mfV(-1.0,1.0),0,-0.75);
    c=2*mfCos(mfPow(mfS(xyz,MF_COL,1),2))*mfExp(-(mfPow(mfS(xyz,MF_CO
L,2),2))-(mfPow(mfS(xyz, MF_COL, 3),2)));
    h = mfTriSurf(tri, xyz, c);
    mfDrawMaterial(h, "edge", "visible", "off");
    mfDrawMaterial(h, "surf", "smooth", "on");
    mfViewPause();

}
```

Result



See Also

mfSliceXYZ

mfGetSliceIJK

Retrieve orthogonal slice-plane(s) along i, j or k indices.

Module

`fgl.h/fgl.lib`

C++ Prototype

`mfArray mfGetSliceIJK(mfOutArray OutArray, [mfArray x, mfArray y, mfArray z, ]mfArray c, mfArray Si, mfArray Sj, mfArray Sk);`

Descriptions

Function `mfGetSliceXYZ` displays and retrieves orthogonal slice-planes along i, j and k which are index of `x`, index of `y` and index of `z` respectively.

It returns the triangular mesh `tri` and the vertex vectors of the sliced-planes defined in the n-by-3 matrix `xyz`.

For details on the input arguments, refer to the description of function `mfSliceIJK`.

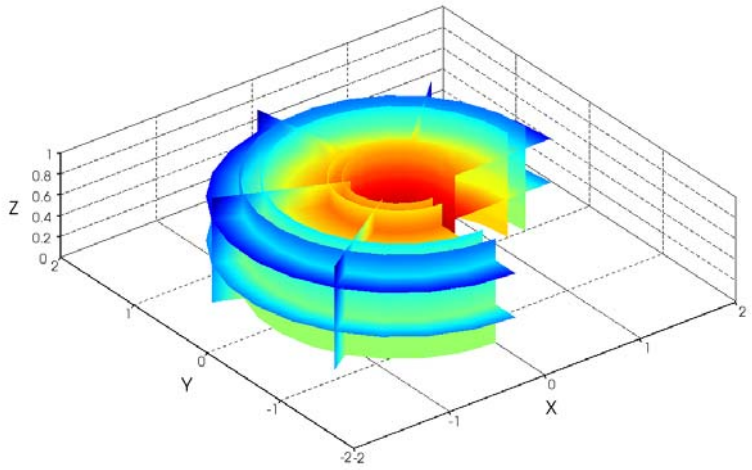
Example

Code

```
// ************************************************
// AnCAD example file
// file: mfGetSliceIJK.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray nu, nv, nw, u, v, w, x, y, z, c, h, tri, xyz;

    nu = mfLinspace(0, 1.5*MF_PI, 20);
    nv = mfLinspace(0, 2*MF_PI, 20);
    nw = mfLinspace(0, 1, 20);
    mfMeshgrid( mfOut(u, v, w), nu, nv, nw );
    x = (1+0.6* mfCos(v))*mfCos(u);
    y = (1+0.6* mfCos(v))*mfSin(u);
    z = w;
    c = 1-(x*x + y*y + z*z);

    mfGetSliceIJK(mfOut(tri,xyz),x,y,z,c,mfV(4,8,12,16),mfV(4,8,12,16
),mfV(8,16));
    c=1-(mfPow(mfS(xyz,MF_COL,1),2)+mfPow(mfS(xyz,MF_COL,2),2)+mfPow(
mfS(xyz,MF_COL,3),2));
    h=mfTriSurf(tri, xyz, c);
    mfDrawMaterial(h,"edge","visible","off");
    mfDrawMaterial(h,"surf","smooth","on");

    mfAxis("equal");
    mfCamZoom(1.5);
```

```
    mfViewPause();
}
```

Result



See Also

mfSliceIJK

mfGetSlicePlane

Retrieve orthogonal slice-plane(s) along arbitrary direction.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfGetSlicePlane(mfOutArray OutArray, [mfArray x, mfArray y,
mfArray z, ]mfArray c, mfArray Sx, mfArray Sy, mfArray Sz);
```

Descriptions

Function `mfGetSliceXYZ` displays and retrieves orthogonal slice-planes of a specified set of volumetric data along arbitrary direction. It returns the triangular mesh `tri` and the vertex vectors of the sliced-planes defined in the n-by-3 matrix `xyz.`

For details on the input arguments, refer to the description of function `mfSlicePlane.`

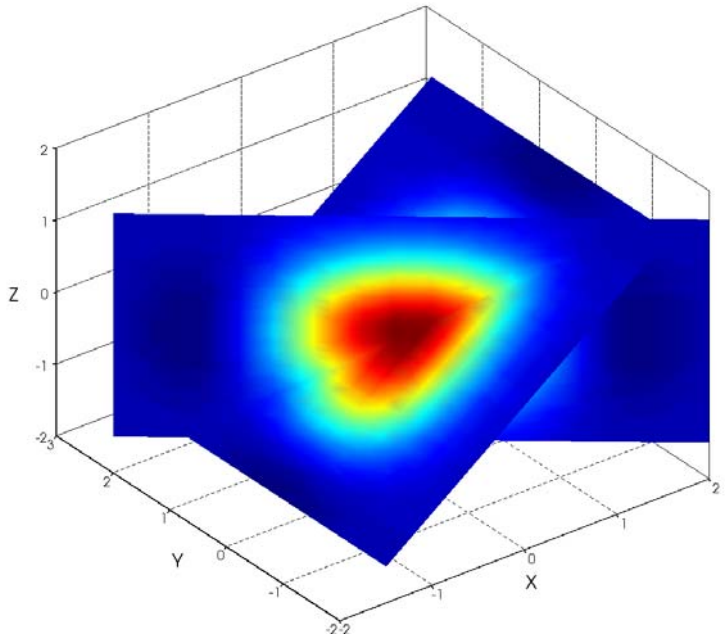
Example

Code

```
// ************************************************
// AnCAD example file
// file: mfGetSlicePlane.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray nx, ny, nz, x, y, z, c, h;
    mfArray tri, xyz, c1;

    nx=mfLinspace(-2, 2.2, 21);
    ny=mfLinspace(-2, 2.25, 17);
    nz=mfLinspace(-1.5, 1.6, 31);
    mfMeshgrid(mfOut(y, x, z), ny, nx, nz);
    c=2*mfCos(x*x)*mfExp(-(y*y)-(z*z));

    mfGetSlicePlane(mfOut(tri, xyz), x, y, z, c, mfV(1, 0, -1, 0));
    c1=2*mfCos(mfPow(mfS(xyz, MF_COL, 1),2))*mfExp(-(mfPow(mfS(xyz,
MF_COL, 2),2))-(mfPow(mfS(xyz, MF_COL, 3),2)));
    h=mfTriSurf(tri, xyz, c1);
    mfDrawMaterial(h, "edge", "visible", "off");
    mfDrawMaterial(h, "surf", "smooth", "on");
    mfHold("on");

    mfGetSlicePlane(mfOut(tri, xyz), x, y, z, c, mfV(1, 1, 0, 0));
    c1=2*mfCos(mfPow(mfS(xyz, MF_COL, 1),2))*mfExp(-(mfPow(mfS(xyz,
MF_COL, 2),2))-(mfPow(mfS(xyz, MF_COL, 3),2)));
    h=mfTriSurf(tri, xyz, c1);
    mfDrawMaterial(h, "edge", "visible", "off");
    mfDrawMaterial(h, "surf", "smooth", "on");
    mfViewPause();
```

```
}
```

Result



See Also

mfSlicePlane

Streamline Graphs

mfStreamLine

StreamLine from three-dimensional vector data.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfStreamLine(mfArray x, mfArray y, mfArray z, mfArray u, mfArray v, mfArray w, mfArray Sx, mfArray Sy, mfArray Sz);
mfArray mfStreamLine(mfArray u, mfArray v, mfArray w, mfArray Sx, mfArray Sy, mfArray Sz);
```

Descriptions

Function `mfStreamLine` creates streamlines from three-dimensional vector components `u, v` and `w.`

`mfStreamLine(x, y, z, u, v, w, Sx, Sy, Sz)`

- Arguments `u, v` and `w` are three-dimensional orthogonal vector components corresponding to the x, y and z-direction respectively.
- Arguments `x, y` and `z` define the coordinates for `u, v` and `w` and must be monotonic and three-dimensional plaid (as if produced by `mfMeshgrid).`
- Arguments `Sx, Sy` and `Sz` define the starting positions of the streamlines.

`mfStreamLine(u, v, w, Sx, Sy, Sz)`

- Arguments `x, y` and `z` are derived from `mfMeshgrid(mfOut(x, y, z), mfColon(1, n), mfColon(1, m), mfColon(1, p))`, where `n, m` and `p` are dimensions of `u.`

`h = mfStreamLine(...)`

- Handle `h` retrieves a handle to the streamline objects created by `mfStreamLine(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the streamline objects through handle `h` with function `mfGSet.`

The properties available are:

1. start: a n-by-3 matrix containing starting point coordinates.
2. sizefactor: a double value containing the width factor.

See Also

mfQuiver, mfQuiver3

mfStreamDashedLine

StreamLine from three-dimensional vector data.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfStreamDashedLine(mfArray x, mfArray y, mfArray z, mfArray u,
mfArray v, mfArray w, mfArray Sx, mfArray Sy, mfArray Sz);
mfArray mfStreamDashedLine(mfArray u, mfArray v, mfArray w, mfArray Sx,
mfArray Sy, mfArray Sz);
```

Descriptions

Function `mfStreamDashedLine` creates a stream of dashed lines from three-dimensional vector components `u, v` and `w.`

`mfStreamDashedLine(x, y, z, u, v, w, Sx, Sy, Sz)`

- Arguments `u, v` and `w` are three-dimensional orthogonal vector components corresponding to the x, y and z-direction respectively.
- Arguments `x, y` and `z` define the coordinates for `u, v` and `w` and must be monotonic and three-dimensional plaid (as if produced by `mfMeshgrid).`
- Arguments `Sx, Sy` and `Sz` define the starting positions of the stream of dashed lines.

`mfStreamDashedLine(u, v, w, Sx, Sy, Sz)`

- Arguments `x, y` and `z` are derived from `mfMeshgrid(mfOut(x, y, z), mfColon(1, n), mfColon(1, m), mfColon(1, p))`, where `n, m` and `p` are dimensions of `u.`

`h = mfStreamDashedLine(...)`

- Handle `h` retrieves a handle to the stream of dashed line objects created by `mfStreamDashedLine(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the stream of dashed line objects through handle `h` with function `mfGSet.`

The properties available are:

1. start: a n-by-3 matrix containing starting point coordinates.
2. sizefactor: a double value containing the width factor.

See Also

mfStreamRibbon

Stream of ribbons from three-dimensional vector data.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfStreamRibbon(mfArray x, mfArray y, mfArray z, mfArray u, mfArray v, mfArray w, mfArray Sx, mfArray Sy, mfArray Sz);
mfArray mfStreamRibbon(mfArray u, mfArray v, mfArray w, mfArray Sx, mfArray Sy, mfArray Sz);
```

Descriptions

Function `mfStreamRibbon` creates a stream of ribbon objects from three-dimensional vector components `u, v` and `w.`

`mfStreamRibbon(x, y, z, u, v, w, Sx, Sy, Sz)`

- Arguments `u, v` and `w` are three-dimensional orthogonal vector components corresponding to the x, y and z-direction respectively.
- Arguments `x, y` and `z` define the coordinates for `u, v` and `w` and must be monotonic and three-dimensional plaid (as if produced by `mfMeshgrid).`
- Arguments `Sx, Sy` and `Sz` define the starting positions of the stream of ribbon objects.

`mfStreamRibbon(u, v, w, Sx, Sy, Sz)`

- Arguments `x, y` and `z` are derived from `mfMeshgrid(mfOut(x, y, z), mfColon(1, n), mfColon(1, m), mfColon(1, p))`, where `n, m` and `p` are dimensions of `u.`

`h = mfStreamRibbon(...)`

- Handle `h` retrieves a handle to the stream of ribbon objects created by `mfStreamRibbon(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the stream ribbon objects through handle `h` with function `mfGSet.`

The properties available are:

1. start: a n-by-3 matrix containing starting point coordinates.
2. sizefactor: a double value containing the width factor.

pr

See Also

mfStreamTube

Stream of tubes from three-dimensional vector data.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfStreamTube(mfArray x, mfArray y, mfArray z, mfArray u, mfArray v, mfArray w, mfArray Sx, mfArray Sy, mfArray Sz);
mfArray mfStreamTube(mfArray u, mfArray v, mfArray w, mfArray Sx, mfArray Sy, mfArray Sz);
```

Descriptions

Function `mfStreamTube` creates a stream of tube objects from three-dimensional vector components `u, v` and `w.`

`mfStreamTube(x, y, z, u, v, w, Sx, Sy, Sz)`

- Arguments `u, v` and `w` are three-dimensional orthogonal vector components corresponding to the x, y and z-direction respectively.
- Arguments `x, y` and `z` define the coordinates for `u, v` and `w` and must be monotonic and three-dimensional plaid (as if produced by `mfMeshgrid).`
- Arguments `Sx, Sy` and `Sz` define the starting positions of the stream of tube objects.

`mfStreamTube(u, v, w, Sx, Sy, Sz)`

- Arguments `x, y` and `z` are derived from `mfMeshgrid(mfOut(x, y, z), mfColon(1, n), mfColon(1, m), mfColon(1, p))`, where `n, m` and `p` are dimensions of `u.`

`h = mfStreamTube(...)`

- Handle `h` retrieves a handle to the stream of tube objects created by `mfStreamTube(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the stream tube objects through handle `h` with function `mfGSet.`

The properties available are:

1. start: a n-by-3 matrix containing starting point coordinates.
2. sizefactor: a double value containing the width factor.

See Also

Triangular Surface Graphs

mfTriSurf

Polygonal surface plot.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfTriSurf(mfArray tri, mfArray x, mfArray y, mfArray z[, mfArray c]);
mfArray mfTriSurf(mfArray tri, mfArray xyz[, mfArray c]);
```

Descriptions

Function `mfTriSurf` displays polygons defined by a face matrix. The polygons must be convex polygons.

```
mfTriSurf(tri, x, y, z)
mfTriSurf(tri, x, y, z, c)
```

- Display the polygons defined by a m-by-n face matrix `tri` as a surface object, where m is the number of polygons to be drawn and n is the number of edges of each polygon. For example, a 4-by-3 face matrix `tri` draws a surface object of 4 triangles and a 3-by-4 face matrix `tri` draws a surface object of 3 quadrilaterals.
- Each row of face matrix `tri` contains indices into `x, y` and `z` vertex vectors to define a single polygonal face.
- As with function `mfSurf,` the color scale is assumed to be proportional to the surface height specified by vertex `z.`
- Argument `c` overrides the default color specification and defines the new edge color.
- The shapes of the four mfArrays `x, y, z` and `c` should be conformed.
- Note that you can use `mfSurf` to plot the polygons and switch the shading mode using the toolbar function **shading mode**.

```
mfTriSurf(tri, xyz)
mfTriSurf(tri, xyz, c)
```

- Vertex vectors are defined in the n-by-3 matrix `xyz.`

```
h = mfTriSurf(...)
```

- Handle `h` retrieves a handle to the polygonal surface object created by `mfTriSurf(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the polygonal surface object through handle h with function `mfGSet.`

The properties available are:

1. tri
2. xyz

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfTriSurf.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray tri, x, y, z, c;

    x = mfRand(400,1);
    y = mfRand(400,1);
    z = 1 - (mfPow((x-0.5),2) + mfPow((y-0.5),2));
    c = mfSin(z) * mfCos(z);
    tri = mfGetDelaunay(x, y);

    mfTitle("mfTriSurf");
    mfTriSurf(tri, x, y, z) ;
    mfViewPause();

}
```

See Also

mfTriMesh

Polygonal mesh plot.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfTriMesh(mfArray tri, mfArray x, mfArray y, mfArray z[, mfArray c]);
mfArray mfTriMesh(mfArray tri, mfArray xyz[, mfArray c]);
```

Descriptions

Function `mfTriMesh` displays polygons in mesh defined by a face matrix. The polygons must be convex polygons.

```
mfTriMesh(tri, x, y, z)
mfTriMesh(tri, x, y, z, c)
```

- Display the polygons defined by a m-by-n face matrix `tri` as a meshed surface object, where m is the number of polygons to be drawn and n is the number of edges of each polygon. For example, a 4-by-3 face matrix `tri` draws a surface object of 4 triangles and a 3-by-4 face matrix `tri` draws a surface object of 3 quadrilaterals.
- Each row of face matrix `tri` contains indices into `x, y` and `z` vertex vectors to define a single polygonal face.
- As with the function `mfSurf,` the color scale is assumed to be proportional to the surface height specified by vertex `z.`
- Argument `c` overrides the default color specification and defines the new edge color.
- The shapes of the four mfArrays `x, y, z` and `c` should be conformed.
- Note that you can use `mfMesh` to plot the polygons and switch the shading mode using the toolbar function **shading mode**.

```
mfTriMesh(tri, xyz)
mfTriMesh(tri, xyz, c)
```

- Vertex vectors are defined in the n-by-3 matrix `xyz.`

```
h = mfTriMesh(...)
```

- Handle `h` retrieves a handle to the polygonal meshed surface object created by `mfTriMesh.`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the polygonal meshed surface object through handle `h` with function `mfGSet.`

The properties available are:

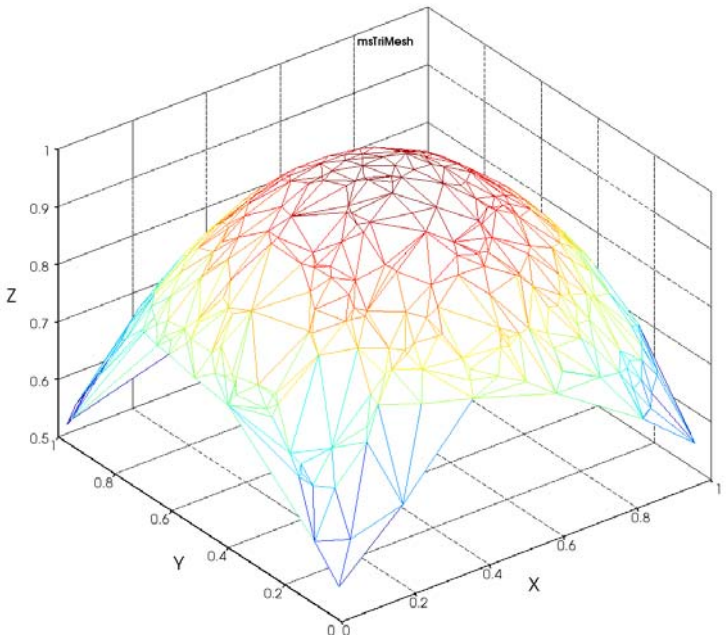
1. tri
2. xyz

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfTriMesh.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray tri, x, y, z, c;

    x = mfRand(400,1);
    y = mfRand(400,1);
    z = 1 - (mfPow((x-0.5), 2) + mfPow((y-0.5),2));
    c = mfSin(z) * mfCos(z);
    tri = mfGetDelaunay(x, y);

    mfTitle("mfTriMesh");
    mfTriMesh(tri, x, y, z) ;
    mfViewPause();

}
```

Result



See Also

mfTriContour

Contour on polygonal plot.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfTriContour(mfArray tri, mfArray x, mfArray y, mfArray z[,
mfArray c]);
mfArray mfTriContour(mfArray tri, mfArray xyz[, mfArray c]);
```

Descriptions

Function `mfTriSurface` plots contour lines of matrix `z` on the polygons defined by a face matrix. The polygons must be convex polygons.

```
mfTriContour(tri, x, y, z)
mfTriContour(tri, x, y, z, c)
```

- Generate contour lines on the polygons for selected scalar values. The values plotted are selected automatically.
- As with functions `mfTriSurf` and `mfTriMesh`, the polygons are defined by a m-by-n face matrix `tri`. Each row of face matrix `tri` contains indices into `x, y` and `z` vertex vectors to define a single polygonal face.
- As with the Surface Graphs functions, the color scale is assumed to be proportional to the surface height specified by vertex `z`.
- Argument `c` overrides the default edge color specification, and defines the new edge color.
- The shapes of the four mfArrays `x, y, z` and `c` should be conformed.

```
mfTriContour(tri, xyz)
mfTriContour(tri, xyz, c)
```

- Vertex vectors are defined in the n-by-3 matrix `xyz`.

```
h = mfTriContour(...)
```

- Handle `h` retrieves a handle to the contour line objects created by `mfTriContour(...)`.
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the contour line objects through handle `h` with function `mfGSet`.

The properties available are:

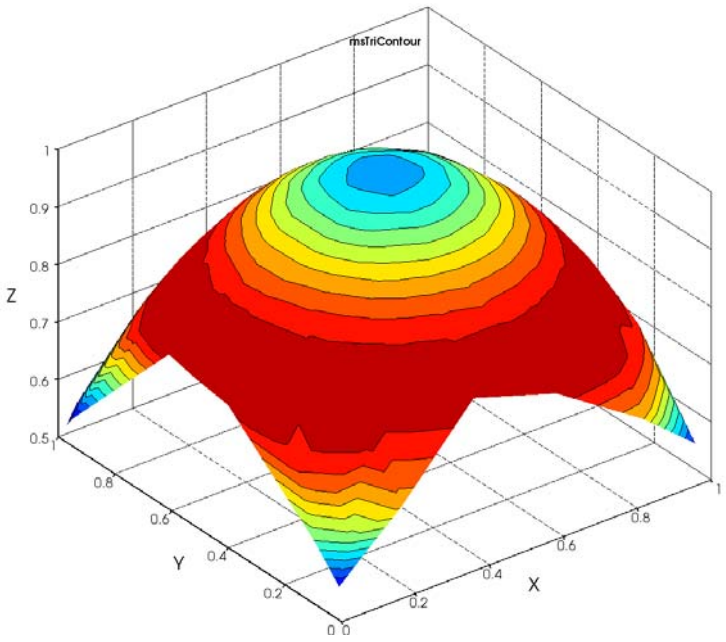
1. tri
2. xyz 3. iso: iso-values, a vector containing iso-value set. Setting this property will replace default set of contour lines. 4. autolevel: given number of levels, it will generate the iso-value set automatically.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfTriContour.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray tri, x, y, z, c;

    x = mfRand(400,1);
    y = mfRand(400,1);
    z = 1 - (mfPow((x-0.5),2) + mfPow((y-0.5),2));
    c = mfSin(z) * mfCos(z);
    tri = mfGetDelaunay(x, y);

    mfTitle("mfTriContour");
    mfTriContour(tri, x, y, z, c);
    mfViewPause();

}
```

Result



See Also

mfTriMesh

mfPatch

Add patch on 2-D or 3-D coordinates.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfPatch(mfArray x, mfArray y, mfArray c);
mfArray mfPatch(mfArray x, mfArray y, mfArray z, mfArray c);
```

Descriptions

Function `mfPatch` adds the patch (filled 2-D polygon) on the coordinates specified by arguments `x` and `y`. Notice that only convex polygons can be accepted.

```
mfPatch(x, y)
mfPatch(x, y, c)
```

- Add the patch on the vertices defined by arguments `x` and `y`.
- If arguments `x` and `y` are matrices, then each column defines a single patch.
- Argument `c` defines the color scale for the vertices that determine the interior color of the patch.
- The shapes of the three mfArrays `x, y` and `c` should be conformed.

```
mfPatch(x, y, z)
mfPatch(x, y, z, c)
```

- Add the patch on the 3-D coordinates defined by arguments `x, y` and `z`.
- If `x, y` and `z` are matrices of the same size, then each column defines a single patch.
- The color scale is assumed to be proportional to the surface height specified by vertex `z` and is used to determine the interior color of the added patch.
- If argument `c` is defined, it overrides the default color specification and defines the new color scale for the vertices.
- The shapes of the four mfArrays `x, y, z` and `c` should be conformed.

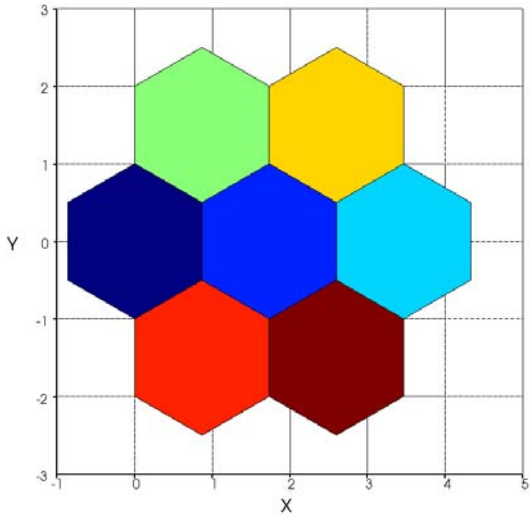
Example

Code

```
// ************************************************
// AnCAD example file
// file: mfPatch.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include <math.h>
#include "cml.h"
```

```
#include "fgl.h"

void main() {

    mfArray x, y, c, h;
    double p, q;

    p = 0.5*sqrt(3.0);
    q = 1.5;

    double px[] = {
        0.0, p, p, 0.0, -p, -p,
        2*p, 3*p, 3*p, 2*p, p, p,
        4*p, 5*p, 5*p, 4*p, 3*p, 3*p,
        p, 2*p, 2*p, p, 0.0, 0.0,
        3*p, 4*p, 4*p, 3*p, 2*p, 2*p,
        p, 2*p, 2*p, p, 0.0, 0.0,
        3*p, 4*p, 4*p, 3*p, 2*p, 2*p };

    double py[] = {
        -1.0, -0.5, 0.5, 1.0, 0.5, -0.5,
        -1.0, -0.5, 0.5, 1.0, 0.5, -0.5,
        -1.0, -0.5, 0.5, 1.0, 0.5, -0.5,
        -1.0+q, -0.5+q, 0.5+q, 1.0+q, 0.5+q, -0.5+q,
        -1.0+q, -0.5+q, 0.5+q, 1.0+q, 0.5+q, -0.5+q,
        -1.0-q, -0.5-q, 0.5-q, 1.0-q, 0.5-q, -0.5-q,
        -1.0-q, -0.5-q, 0.5-q, 1.0-q, 0.5-q, -0.5-q  };

    double pz[] = {
        1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
        2.0, 2.0, 2.0, 2.0, 2.0, 2.0,
        3.0, 3.0, 3.0, 3.0, 3.0, 3.0,
        4.0, 4.0, 4.0, 4.0, 4.0, 4.0,
        5.0, 5.0, 5.0, 5.0, 5.0, 5.0,
        6.0, 6.0, 6.0, 6.0, 6.0, 6.0,
        7.0, 7.0, 7.0, 7.0, 7.0, 7.0 };

    x = mfTranspose(mfCreateFromCArray(px, 7, 6));
    y = mfTranspose(mfCreateFromCArray(py, 7, 6));
    c = mfTranspose(mfCreateFromCArray(pz, 7, 6));

    h = mfPatch(x, y, c);
    mfView("2")   ;
    mfAxis("equal");
    mfViewPause();

}
```

Result



See Also

Unstructured Grids

mfTetSurf

Polyhedral surface plot.

Module

`fgl.h/fgl.lib`

C++ Prototype

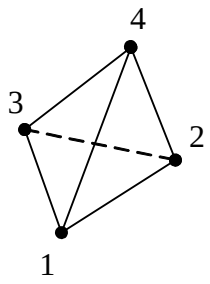
```
mfArray mfTetSurf(mfArray tet, mfArray x, mfArray y, mfArray z[, mfArray
c]);
mfArray mfTetSurf(mfArray tet, mfArray xyz[, mfArray c]);
```

Descriptions

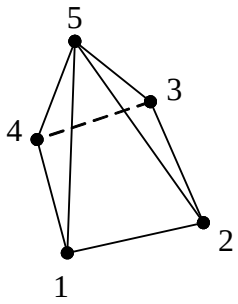
Function `mfTetSurf` displays polyhedrons defined by a cell matrix.

```
mfTetSurf(tet, x, y, z)
mfTetSurf(tet, x, y, z, c)
```

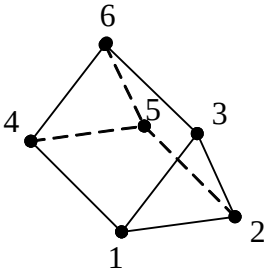
- Display the polyhedrons defined by a m-by-k cell matrix `tet` as a polyhedral object, where m is the number of polyhedrons to be drawn. There are four different types of polyhedrons depending on the value of k as illustrated below.



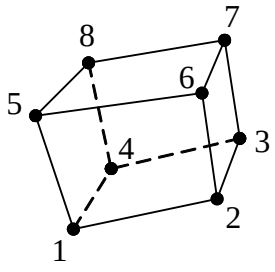
Tetrahedron (k=4)



Pyramid (k=5)



Wedge (k=6)



Hexahedron (k=8)

- Each row of cell matrix `tet` contains indices into `x, y` and `z` vertex vectors to define a single polyhedron.

- As with function `mfSurf,` the edge color is assumed to be proportional to the surface height specified by vertex `z.`
- Argument `c` overrides the default color specification and defines the new edge color.
- The shapes of the four mfArrays `x, y, z` and `c` should be conformed.
- Note that you can use either `mfTetSurf` or `mfTetMesh` to plot the polygons and switch the shading mode using the toolbar function **shading mode**.

```
mfTetSurf(tet, xyz)
mfTetSurf(tet, xyz, c)
```

- Vertex vectors are defined in the n-by-3 matrix `xyz` where n is the number of vertices.

```
h = mfTetSurf(...)
```

- Handle `h` retrieves a handle to the polyhedral object created by `mfTetSurf(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()`to retrieve the handle of the current graphics object.

You can specify properties of the polyhedral object through handle `h` with function `mfGSet.`

The properties available are:

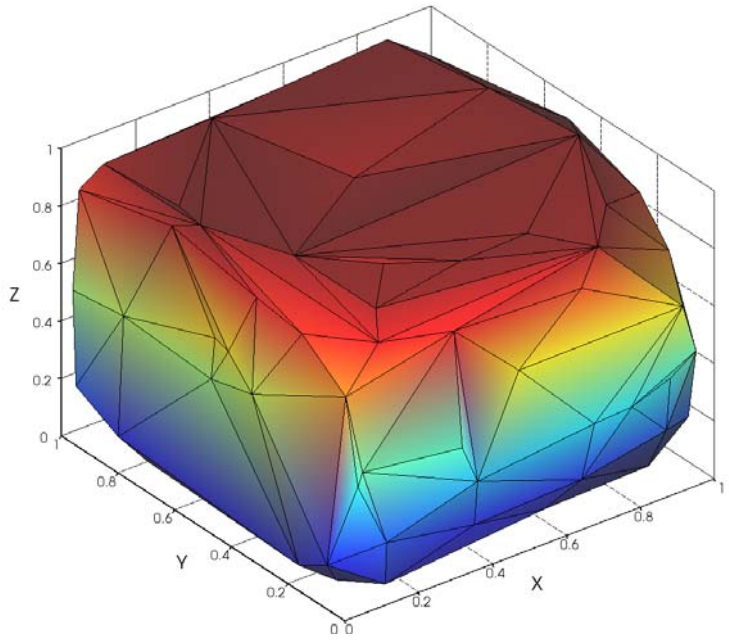
1. tet
2. xyz

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfTetSurf.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray tet, x, y, z, c;

    x = mfRand(500,1);
    y = mfRand(500,1);
    z = mfRand(500,1);
    c = 1 - (mfPow((x - 0.5), 2) + mfPow((y - 0.5), 2) + mfPow((z - 0.5),
2));

    tet = mfGetDelaunay3(x, y, z);

    mfTetSurf(tet, x, y, z);
    mfViewPause();

}
```

Result



See Also

mfTetMesh

Polyhedral mesh plot.

Module

`fgl.h/fgl.lib`

C++ Prototype

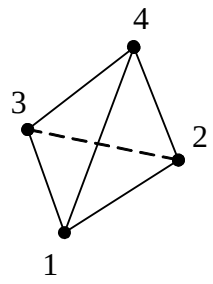
```
mfArray mfTetMesh(mfArray tet, mfArray x, mfArray y, mfArray z[, mfArray c]);
mfArray mfTetMesh(mfArray tet, mfArray xyz[, mfArray c]);
```

Descriptions

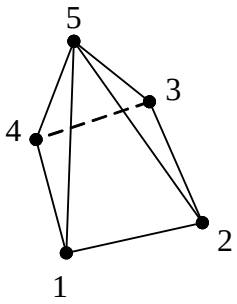
Function `mfTetSurf` displays polyhedrons defined by a cell matrix.

```
mfTetMesh(tet, x, y, z)
mfTetMesh(tet, x, y, z, c)
```

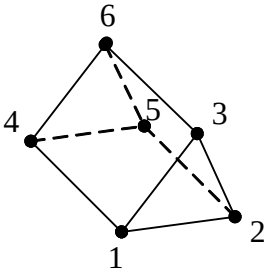
- Display the polyhedrons defined by a m-by-n cell matrix `tet` as a meshed polyhedral object, where m is the number of polyhedrons to be drawn. There are four different types of polyhedrons depending on the value of n as illustrated below.



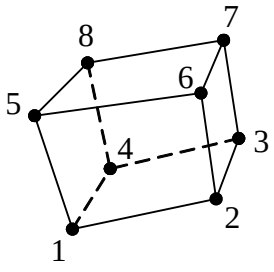
Tetrahedron (k=4)



Pyramid (k=5)



Wedge (k=6)



Hexahedron (k=8)

- Each row of cell matrix `tet` contains indices into `x, y` and `z` vertex vectors to define a single polyhedron.

- As with function `mfMesh,` the edge color is assumed to be proportional to the surface height specified by vertex `z.`
- Argument `c` overrides the default color specification and defines the new edge color.
- The shapes of the four mfArrays `x, y, z` and `c` should be conformed.
- Note that you can use either `mfTetSurf` or `mfTetMesh` to plot the polygons and switch the shading mode using the toolbar function **shading mode**.

```
mfTetMesh(tet, xyz)
mfTetMesh(tet, xyz, c)
```

- Vertex vectors are defined in the n-by-3 matrix `xyz.`

```
h = mfTetSurf(...)
```

- Handle `h` retrieves a handle to the polyhedral object created by `mfTetMesh(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the polyhedral object through handle `h` with function `mfGSet.`

The properties available are:

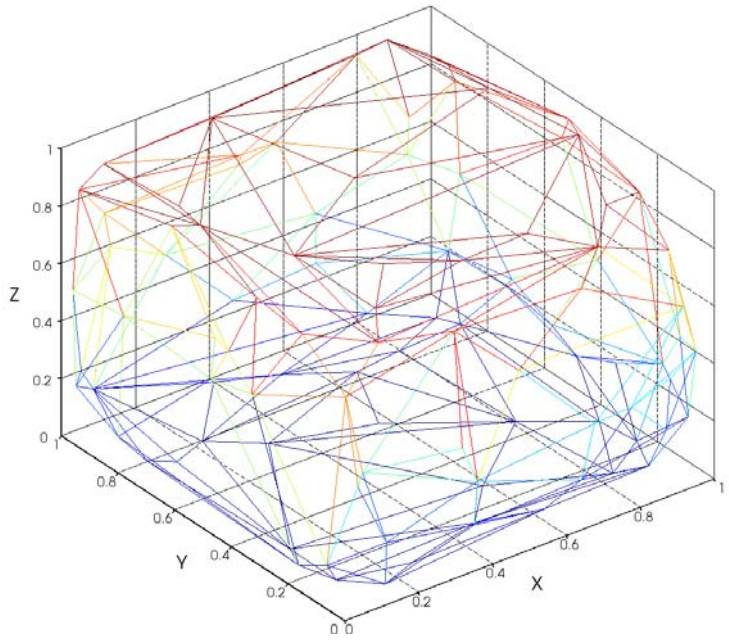
1. tet
2. xyz

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfTetMesh.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray tet, x, y, z, c;

    x = mfRand(500,1);
    y = mfRand(500,1);
    z = mfRand(500,1) ;
    c = 1 - (mfPow((x - 0.5), 2) + mfPow((y - 0.5), 2) + mfPow((z - 0.5),
2));

    tet = mfGetDelaunay3(x, y, z);

    mfTetMesh(tet, x, y, z);
    mfViewPause();

}
```

Result



See Also

mfTetContour

Contour on polyhedral plot.

Module

`fgl.h/fgl.lib`

C++ Prototype

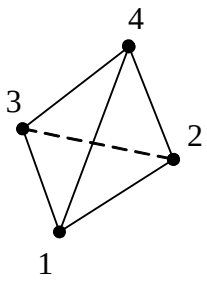
```
mfArray mfTetContour(mfArray tet, mfArray x, mfArray y, mfArray z[,
mfArray c]);
mfArray mfTetContour(mfArray tet, mfArray xyz[, mfArray c]);
```

Descriptions

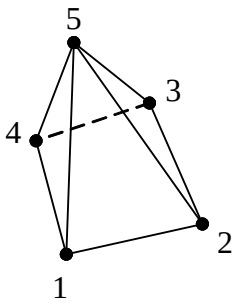
Function `mfTetSurface` plots contour lines on the surface of the polyhedral object defined by a face matrix.

```
mfTetContour(tet, x, y, z)
mfTetContour(tet, x, y, z, c)
```

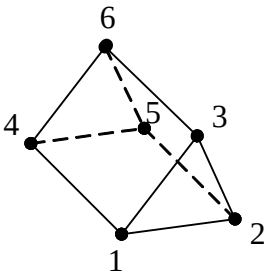
- Display the polyhedrons defined by a m-by-n cell matrix `tet` as a meshed polyhedral object, where m is the number of polyhedrons to be drawn. There are four different types of polyhedrons depending on the value of n as illustrated below.



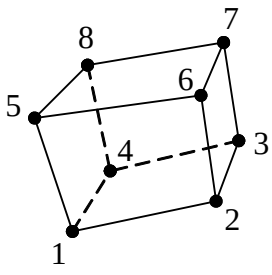
Tetrahedron (k=4)



Pyramid (k=5)



Wedge (k=6)



Hexahedron (k=8)

- Generate contour lines of matrix `z` on the surface of the polyhedral object for selected

scalar values. The values plotted are selected automatically.

- Similar to functions `mfTetSurf` and `mfTetMesh`, the polyhedral object is the combination of the polyhedrons defined by a m-by-n face matrix `tet`. Each row of face matrix `tet` contains indices into `x`, `y` and `z` vertex vectors to define a single polyhedron.
- The color scale is assumed to be proportional to the surface height specified by vertex `z`.
- Argument `c` overrides the default color specification and defines the new edge color.

- The shapes of the four mfArrays `x`, `y`, `z` and `c` should be conformed.

```
mfTetContour(tet, xyz)
mfTetContour(tet, xyz, c)
```

- Vertex vectors are defined in the n-by-3 matrix `xyz`.

```
h = mfTetContour(...)
```

- Handle `h` retrieves a handle to the contour line objects created by `mfTetContour(...)`.
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the contour line objects through handle `h` with function `mfGSet`.

The properties available are:

1. tet
2. xyz 3. iso: iso-values, a vector containing iso-value set. Setting this property will replace default set of contour lines. 4. autolevel: given number of levels, it will generate the iso-value set automatically.

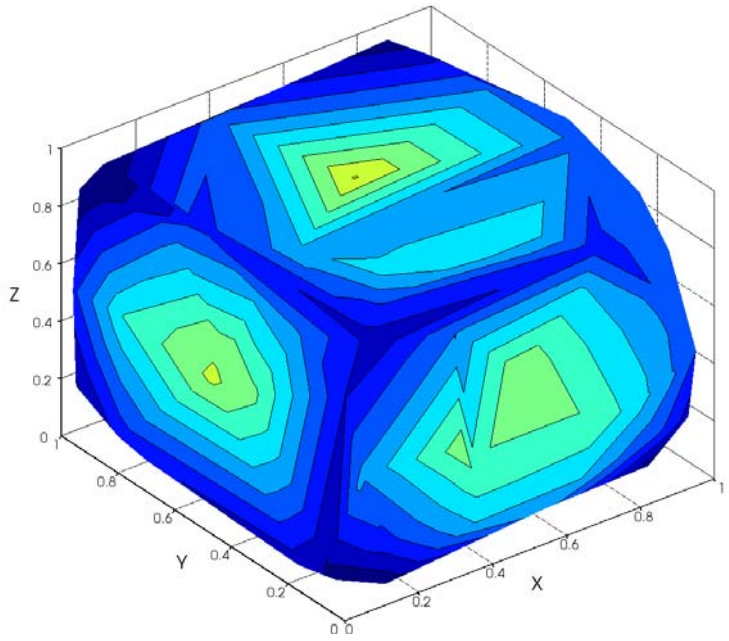
Example

Code

```
// ************************************************
// AnCAD example file
// file: mfTetContour.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray tet, x, y, z, c;

    x = mfRand(500,1);
    y = mfRand(500,1);
    z = mfRand(500,1) ;
    c = 1 - (mfPow((x - 0.5), 2) + mfPow((y - 0.5), 2) + mfPow((z - 0.5),
2));
```

```
    tet = mfGetDelaunay3(x, y, z);

    mfTetContour(tet, x, y, z, c);
    mfViewPause();
}
```

Result



See Also

mfTetMesh

mfTetIsoSurface

Polyhedral isosurface plot.

Module

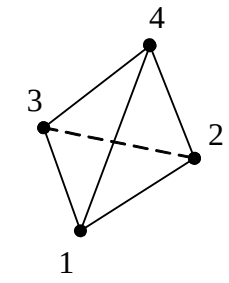
`fgl.h/fgl.lib`

Descriptions

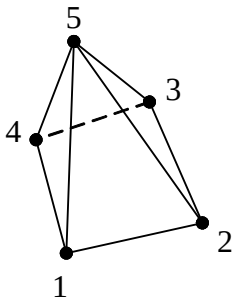
Function `mfTetIsoSurface` creates 3-D graphs composed of isosurface data from the volume data `c` at the isosurface value specified in argument `isovalue.`

`mfTetIsoSurface(x, y, z, c, isovalue)`

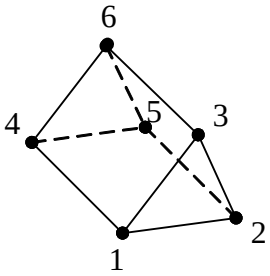
- Display the polyhedrons defined by a m-by-n cell matrix `tet` as a meshed polyhedral object, where m is the number of polyhedrons to be drawn. There are four different types of polyhedrons depending on the value of n as illustrated below.



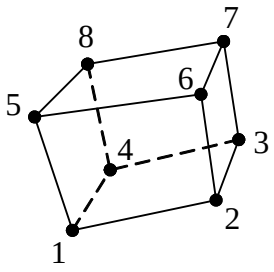
Tetrahedron (k=4)



Pyramid (k=5)



Wedge (k=6)



Hexahedron (k=8)

- The arguments `x, y` and `z` define the coordinates for the volume `c.`

`mfTetIsoSurface(c, isovalue)`

- The coordinates for the volume `c` is of a geometrically rectangular grid where `x = mfColon(1, n)`and `y = mfColon(1, m) and z = mfColon(1, p)`.

`h = mfTetIsoSurface(...)`

- Handle h retrieves a handle to the isosurface object created by `mfTetIsoSurface(...).`
- Alternatively, you use procedure `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the isosurface object through handle h with procedure `msGSet.`

The properties available are:

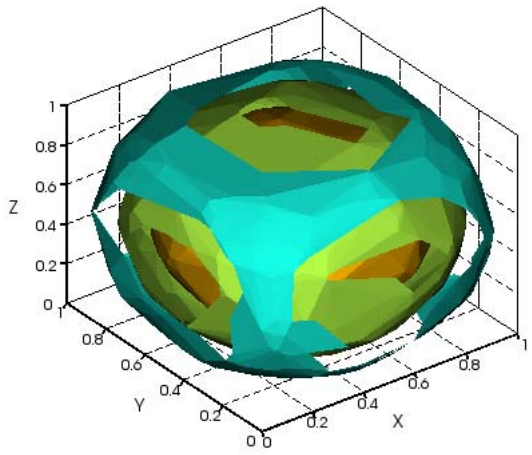
1. iso: iso-value, a vector containing iso-value sets.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfTetIsoSurface.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray tet, x, y, z, c;

    x = mfRand(500,1);
    y = mfRand(500,1);
    z = mfRand(500,1);
    c = 1 - ( mfPow((x - 0.5), 2) + mfPow((y - 0.5), 2) + mfPow((z - 0.5),
2) );

    tet = mfGetDelaunay3(x, y, z);

    mfTetIsoSurface(tet, x, y, z, c, mfV(0.8, 0.7, 0.6));
    mfViewPause();

}
```

Result



See Also

mfTetSliceXYZ

Display orthogonal slice-planes through volumetric data.

Module

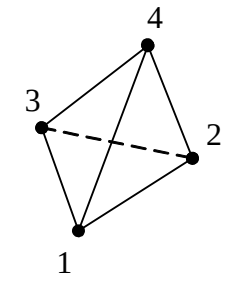
`fgl.h/fgl.lib`

Descriptions

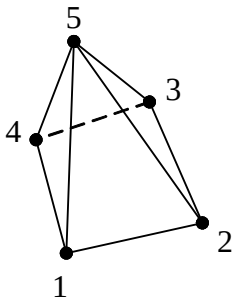
Function mfSliceXYZ displays orthogonal slice-planes of a specified set of volumetric data. The information on the slice-planes can be retrieved by using function `mfGetSliceXYZ.`

`mfTetSliceXYZ(tet, x, y, z, c, Sx, Sy, Sz)`

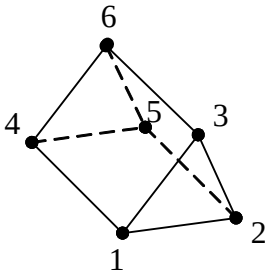
- Display the polyhedrons defined by a m-by-n cell matrix `tet` as a meshed polyhedral object, where m is the number of polyhedrons to be drawn. There are four different types of polyhedrons depending on the value of n as illustrated below.



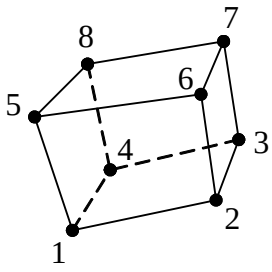
Tetrahedron (k=4)



Pyramid (k=5)



Wedge (k=6)



Hexahedron (k=8)

- Displays orthogonal slice-planes along the x, y and z directions specified by points in vector mfArrays `Sx, Sy` and `Sz`
- Use `mf()` to substitute any of the direction vectors `Sx, Sy` or `Sz` if you are not drawing any slice along the respective direction. E.g. mfSliceXYZ(x, y, z, v, Sx, mf(), Sz) draws slices along the x-axis as specified by `Sx` and z-axis as specified by `Sz.`
- The arguments `x, y` and `z` define the corresponding coordinates of scalar values mfArray `c,` where `c` is an m-by-n-by-p three-dimensional array.

- The arguments `x, y` and `z` must be of the same shape as `c` and are monotonic and three-dimensional plaid as if produced by procedure `mfMeshgrid.`
- The color at each point is determined by three-dimensional interpolation into the elements of volume `c,` mapped to the current colormap.

`mfTetSliceXYZ(tet, c, Sx, Sy, Sz)`

- Assumes `x` is composed of mfColon(1, n) vectors, `y` is composed of mfColon(1, m) vectors and `z` is composed of mfColon(1, p) vectors.

`h = mfTetSliceXYZ(...)`

- Handle `h` retrieves a handle to the volumetric slice object created by `mfSliceXYZ(...).`
- Alternatively, you use procedure `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the volumetric slice objects through handle `h` with procedure `msGSet.`

The properties available are:

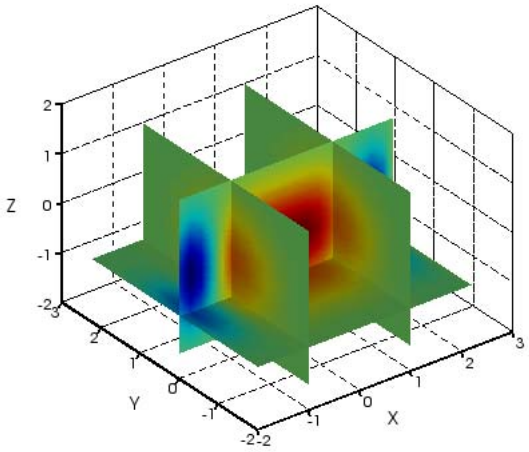
1. tet 2. slicex: specifies the slice-planes along the x direction.
3. slicey: specifies the slice-planes along the y direction.
4. slicez: specifies the slice-planes along the z direction.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfTetSliceXYZ.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray nx, ny, nz, x, y, z, c, tet;

    nx = mfLinspace(-2, 2.2, 21.0);
    ny = mfLinspace(-2, 2.25, 17.0);
    nz = mfLinspace(-1.5, 1.6, 31.0);
    mfMeshgrid(mfOut(y, x, z), ny, nx, nz);
    c = 2*mfCos(x*x)*mfExp(-(y*y)-(z*z));

    tet = mfGetDelaunay3(x, y, z);

    mfTetSliceXYZ(tet, x, y, z, c, mfV(-1.0, 1.0),  0, -0.75);
    mfViewPause();

}
```

Result



See Also

mfTetSlicePlane, mfTetIsoSurface

mfTetSlicePlane

Display orthogonal slice-planes along arbitrary direction.

Module

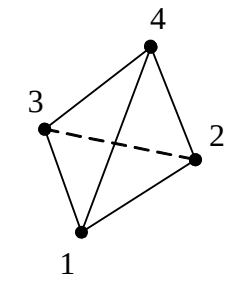
`fgl.h/fgl.lib`

Descriptions

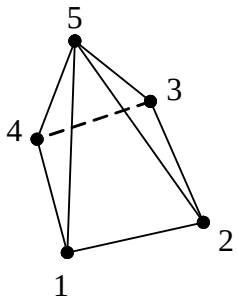
Function `mfSlice` displays orthogonal slice-planes of a specified set of volumetric data along arbitrary direction. The information on the slice-planes can be retrieved by using function `mfGetSlicePlane.`

`mfTetSlicePlane(tet, x, y, z, c, plane)`

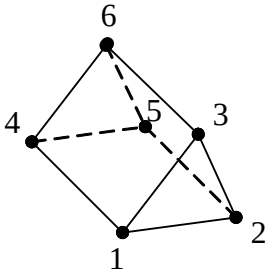
- Display the polyhedrons defined by a m-by-n cell matrix `tet` as a meshed polyhedral object, where m is the number of polyhedrons to be drawn. There are four different types of polyhedrons depending on the value of n as illustrated below.



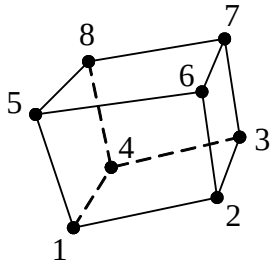
Tetrahedron (k=4)



Pyramid (k=5)



Wedge (k=6)



Hexahedron (k=8)

- Displays orthogonal slice-planes along the direction specified by `plane.`
- The arguments `x, y` and `z` are structured grid data which define the corresponding coordinates of scalar values specified in argument `c,` where `c` is a m-by-n-by-p three-dimensional array.
- Argument `plane` is a vector of size 4 representing the coefficients of the sliced plane equation, i.e. [a, b, c, d], where ax + by + cz + d = 0.

- The arguments x, y and z must be of the same shape as c and are monotonic and three-dimensional plaid as if produced by procedure mfMeshgrid.
- The color at each point is determined by three-dimensional interpolation into the elements of volume c, mapped to the current colormap.

mfTetSlicePlane(tet, c, plane)

- Assumes x is composed of mfColon(1, n) vectors, y is composed of mfColon(1, m) vectors and z is composed of mfColon(1, p) vectors.

h = mfTetSlicePlane(...)

- Handle h retrieves a handle to the volumetric slice objects created by mfSlicePlane(...).
- Alternatively, you use procedure h = mfGetCurrentDraw()to retrieve the handle of the current graphics object.

You can specify properties of the volumetric slice objects through handle h with procedure msGSet.

The property available is:

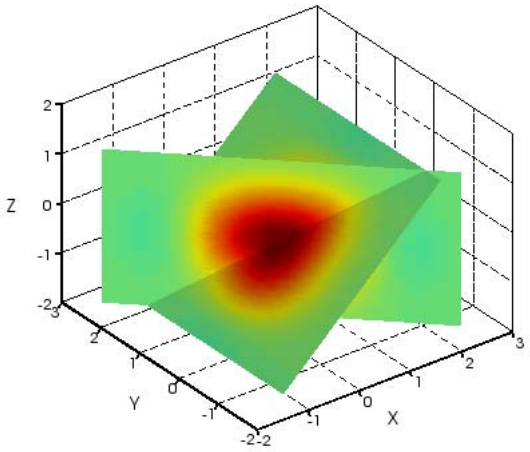
1. tet 2. plane

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfTetSlicePlane.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray nx, ny, nz, x, y, z, c, tet;

    nx = mfLinspace(-2, 2.2, 21);
    ny = mfLinspace(-2, 2.25, 17);
    nz = mfLinspace(-1.5, 1.6, 31);
    mfMeshgrid(mfOut(y, x, z), ny, nx, nz);
    c = 2*mfCos(x*x)*mfExp(-(y*y)-(z*z));

    tet = mfGetDelaunay3(x, y, z);

    mfTetSlicePlane(tet, x, y, z, c, mfV(1, 0, -1, 0));
    mfHold("on");
    mfTetSlicePlane(tet, x, y, z, c, mfV(1, 1, 0, 0));
    mfViewPause();

}
```

Result



See Also

mfSlicePlane, mfGetSlicePlane

Unstructured Point Set

mfPoint

Display input points in three-dimensional space.

Module

```
fgl.h/fgl.lib
```

C++ Prototype

```
mfArray mfPoint(mfArray x, mfArray y, mfArray z[, mfArray c]);
mfArray mfPoint(mfArray xyz[, mfArray c]);
```

Descriptions

Function `mfPoint` plots a set of input points in three-dimensional space.

```
mfPoint(x, y, z)
mfPoint(x, y, z, c)
```

- Arguments `x, y` and `z` contain the x-, y-, and z- coordinates of the points respectively. They can be either matrices or vectors of the same shape.
- By default, the scalar value of the each point is the height which is specified in argument `z.` Specifying the scalar vector `c` would override the default scalar values.

```
mfPoint(xyz)
mfPoint(xyz, c)
```

- Vertex vectors are defined in the n-by-3 matrix `xyz.`

```
h = mfPoint(...)
```

- Handle `h` retrieves a handle to the points created by `mfPoint(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the points through handle `h` with function `mfGSet.`

The property available is:

1. xyz

See Also

mfDelaunay, mfGetDelaunay

2-D Delaunay triangulation of input points.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfDelaunay(mfArray x, mfArray y[, mfArray bx1, mfArray bx2, ...]);
mfArray mfGetDelaunay(mfArray x, mfArray y[, mfArray p1, mfArray
p2, ...]);
void mfGetDelaunay(mfOutArray OutArray, mfArray x, mfArray y[, mfArray
p1, mfArray p2, ...]);
```

Descriptions

Function `mfDelaunay` is a filter that constructs a two-dimensional Delaunay triangulation from a set of input points. You may use function `mfGetDelaunay` to retrieve the output of the filter, the triangular mesh.

`h = mfDelaunay(x, y, bx1, by1, bx2, by2, bx3, by3)`

- Arguments `x` and `y` specify the x- and y- coordinates of the input points.
- You may define the boundaries which the Delaunay triangulation is constructed upon. Each boundary is defined by two vertex vectors specified in the arguments. For instance, arguments `bx1` and `by1` define the first boundary, arguments `bx2` and `by2` define the second boundary and so on.
- Notice that the order of the boundary points determines how the Delaunay triangulation is constructed. If the boundary points are specified counterclockwise, then the Delaunay triangulation is constructed within the boundary; if the boundary points are specified clockwise, then the Delaunay triangulation is constructed beyond the boundary. On the other hand, an unexpected result may arise if a false order of boundary points is given.

```
tri = mfGetDelaunay(x, y, bx1, by1, bx2, by2, bx3, by3)
mfGetDelaunay(mfOut(tri, xyz), x, y, bx1, by1, bx2, by2, bx3, by3)
```

- Retrieve the triangular mesh `tri.` With `tri` and the coordinates `xyz,` you may use `mfTriSurf` to plot the triangular surface.

`h = mfDelaunay(...)`

- Handle `h` retrieves a handle to the polygonal surface object created by `mfDelaunay(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the polygonal surface object through handle h with function mfGSet.

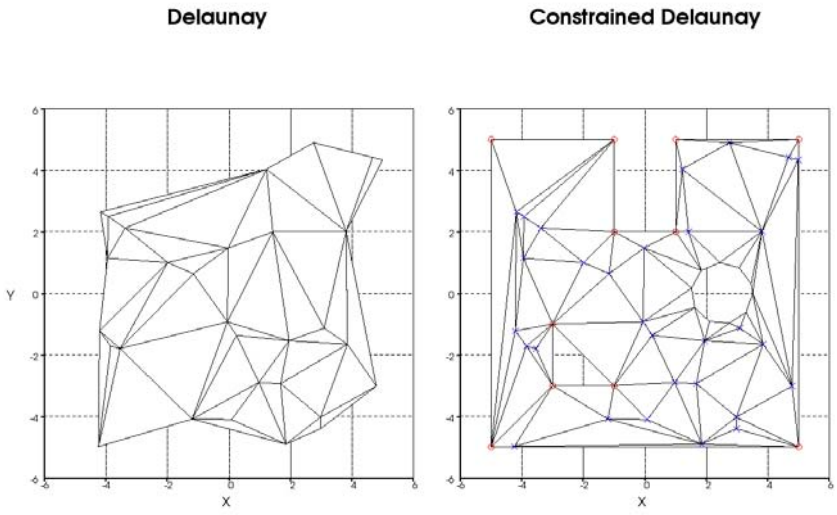
Example

Code

```
// ************************************************
// AnCAD example file
// file: mfDelaunay.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"
#include "stdlib.h"
#include "time.h"


void main() {

    mfArray x, y, n, h;
    mfArray bx1, by1, bx2, by2, bx3, by3;
    int i;
    double rx, ry;

    bx1 = mfV(-5, 5,  5, 1, 1, -1, -1, -5).T();
    by1 = mfV(-5, -5,  5, 5, 2, 2, 5, 5).T();
    bx2 = mfV( -3,  -3, -1 ).T();
    by2 = mfV( -3,  -1, -3 ).T();
    n = mfLinspace(0, -1.9*MF_PI, 10).T();
    bx3 = 2.5 + mfCos(n);
    by3 = mfSin(n);


    x = mfZeros(30,1);
    y = mfZeros(30,1);

    srand((unsigned)time(NULL));

    for(i = 1; i <= 30; i++)
    {
        while(true)
        {
            rx = ((rand()%1001)/ 1000.0);
            ry = ((rand()%1001)/ 1000.0);
            rx= rx*10 - 5;
            ry= ry*10 - 5;
            if ( ((rx<5) || (rx>-5)) && ((ry<5) || (ry>-5)) && ((rx<-1)
|| (rx>1) || (ry<2)) &&
               ((rx<-3) || (ry<-3) || (rx + ry > -2)) && ((rx-2.5)*(rx-2.5)
+ ry*ry > 1) )
            {
                break;
            }
        }

        x(i,1) = rx;
        y(i,1) = ry;
    }

    mfFigure("Delaunay");
    mfSubplot(1, 2, 1);
    mfTitle("Delaunay");
    h = mfDelaunay(x, y);
    mfAxis("equal");
```

```
    mfSubplot(1, 2, 2);
    mfTitle("Constrained Delaunay");
    h = mfDelaunay(x, y, bx1, by1, bx2, by2, bx3, by3);
    mfHold("on");
    h = mfPlot(bx1, by1, "or", bx2, by2, "or", x, y, "xb");
    mfAxis("equal");

    mfViewPause();
}
```

Result



See Also

mfDelaunay3, mfGetDelaunay3

3-D Delaunay triangulation of input points.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfDelaunay3(mfArray x, mfArray y, mfArray z);
mfArray mfDelaunay3(mfArray xyz);
mfArray mfGetDelaunay3(mfArray x, mfArray y, mfArray z);
mfArray mfGetDelaunay3(mfArray xyz);
void mfGetDelaunay3(mfOutArray OutArray, ...);
```

Descriptions

Function `mfDelaunay3` is a filter that constructs a three-dimensional Delaunay triangulation from a set of input points. You may use function to retrieve the output of the filter, the tetrahedral mesh.

```
h = mfDelaunay3(x, y, z)
tet = mfGetDelaunay3(x, y, z)
```

- Arguments `x, y` and `z` specify the x-, y- and z- coordinates of the input points.

```
h = mfDelaunay3(xyz)
[tet, xyz] = mfGetDelaunay3(xyz)
```

- Vertex vectors tet are defined in the n-by-3 matrix `xyz.`

```
tet = mfGetDelaunay3(xyz)
[tet, xyz] mfGetDelaunay3(xyz)
```

- Retrieve the tetrahedral mesh `tet.` With `tet` and the coordinates `xyz,` you may use `mfTetSurf` to plot the tetrahedral surface.

```
h = mfDelaunay3(...)
```

- Handle `h` retrieves a handle to the polyhedral object created by `mfDelaunay3(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()`to retrieve the handle of the current graphics object.

You can specify properties of the polyhedral object through handle `h` with function `mfGSet.`

The property available is:

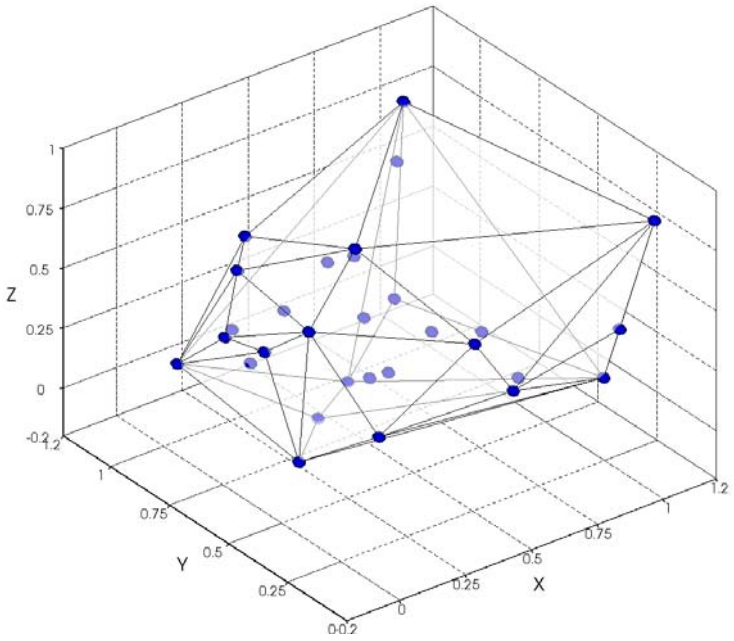
1. xyz

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfDelaunay3.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"


void main() {

    mfArray xyz, h;

    xyz = mfRand(30, 3);

    mfFigure("Delaunay 3D");
    mfTitle("Delaunay 3D");
    h = mfDelaunay3( xyz );
    mfDrawMaterial(h, "surf", "trans", 50);
    mfHold("on");
    h = mfSphere( xyz, 0.02, mfV(0, 0, 1) );
    mfViewPause();

}
```

Result



See Also

Velocity Vectors

mfQuiver

Two-dimensional velocity vectors.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfQuiver(mfArray x, mfArray y, mfArray u, mfArray v[, mfArray
scale]);
mfArray mfQuiver(mfArray u, mfArray v[, mfArray scale]);
```

Descriptions

Function `mfQuiver` plots velocity vectors as arrows with components `(u, v)` at the points `(x, y)`.

```
handle = mfQuiver(x, y, u, v)
handle = mfQuiver(x, y, u, v, scale)
```

- mfArrays `x` and `y` contain positions of the velocity vectors, while the mfArrays `u` and `v` contain the corresponding velocity components.
- You can control the vector scaling by specifying argument `scale.` Specifying `scale` as 0.5 would reduce the relative length of the vector by half.
- The shapes of the four mfArrays `x, y, u` and `v` must conform i.e. All are m-by-n mfArrays.

```
handle = mfQuiver(u, v)
handle = mfQuiver(u, v, scale)
```

- velocity vectors are plotted over a geometrically rectangular grid where `x = mfColon(1, n)` and `y = mfColon(1, m)`.

```
h = mfQuiver(...)
```

- Handle `h` retrieves a handle to the quiver object created by `mfQuiver(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the quiver object through handle `h` with function `mfGSet.`

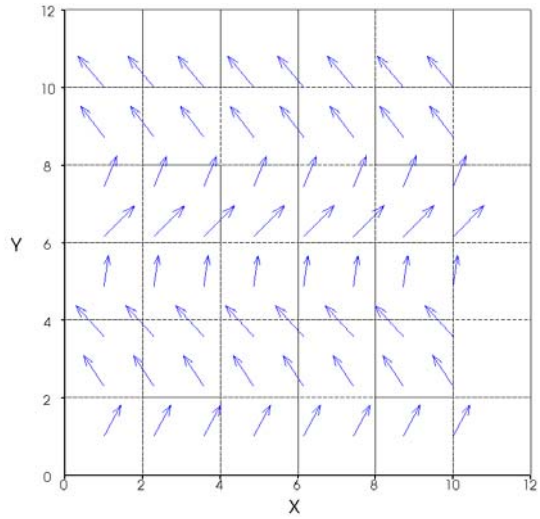
Example

Code

```
// ************************************************
// AnCAD example file
// file: mfQuiver.cpp
```

```
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray u, v, x, y, a;

    a = mfLinspace(1, 10, 8);
    mfMeshgrid(mfOut(x, y), a);
    u = 4*mfCos(y);
    v = 4*mfOnes(8, 8);

    mfQuiver(x, y, u, v, 0.2);
    mfAxis("equal");
    mfCamZoom(0.80);
    mfViewPause();

}
```

Result



See Also

mfQuiver3

mfQuiver3

Three-dimensional velocity vectors.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfQuiver3(mfArray x, mfArray y, mfArray z, mfArray u, mfArray v,
mfArray w, mfArray scale);
void mfQuiver3(...);
```

Descriptions

Function `mfQuiver3` plots velocity vectors as arrows with components `(u, v, w)` at the points `(x, y, z)` in three-dimensional space.

```
handle = mfQuiver3(x, y, z, u, v, w)
handle = mfQuiver3(x, y, z, u, v, w, scale)
```

- mfArrays `x, y` and `z` contain corresponding positions of velocity vectors, which are specified by mfArrays `u, v` and `w` corresponding to the three-dimensional orthogonal velocity components.
- You can control the vector scaling by specifying argument `scale`. Specifying `scale` as 0.5 would reduce the relative length of the vector by half. By default, `scale` = 0.
- The shapes of the four mfArrays `x, y, u` and `v` must be conformed, i.e. All are m-by-n mfArrays.

```
mfQuiver3(z, u, v, w)
mfQuiver3(z, u, v, w, scale)
```

- The height data is assumed to be defined over a geometrically rectangular grid where `x = mfColon(1, n)` and `y = mfColon(1, m)`

```
h = mfQuiver3(...)
```

- Handle `h` retrieves a handle to the velocity vector objects created by `mfQuiver3(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the velocity vector objects through handle `h` with function `mfGSet.`

The property available is:

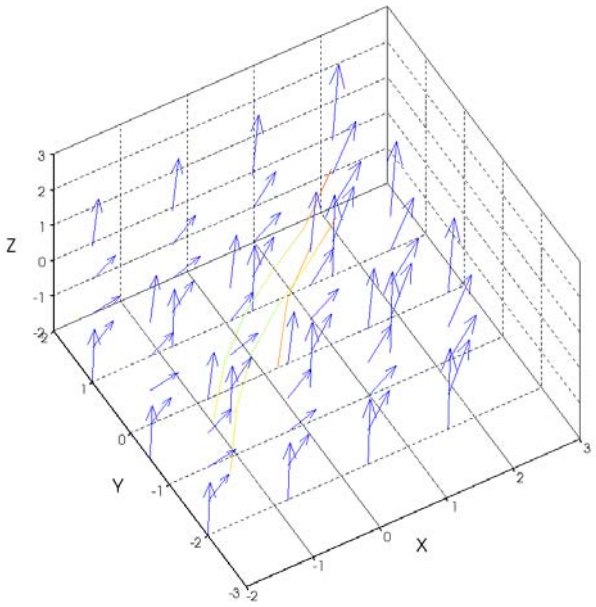
1. symbol: `"arrow"`, `"cone"` or `"flat_arrow"`

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfQuiver3.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray a, b, c, x, y, z, v, u, w;

    a = mfLinspace(-2, 1.6, 4);
    b = mfLinspace(-2, 1, 3);
    c = mfLinspace(-2, 1.84, 5);
    mfMeshgrid(mfOut(x, y, z), a, b, c);
    u = mfOnes(3, 4, 5);
    v = 0.4*(z*z);
    w = mfExp(0.5*x);

    mfQuiver3(x, y, z, u, v, w);
    mfHold("on");
    mfStreamLine(x,y,z,u,v,w, mfV(-1.2,0.0,0.2), mfV(-1.2,0.5,0.0),
mfV(-2.0,-1.0,-2.0));
    mfView(-30,50);
    mfViewPause();

}
```

Result



See Also

mfQuiver

Image

mfImage

Display image file.

Module

```
fgl.h/fgl.lib
```

C++ Prototype

```
mfArray mfImage(mfArray ax);
```

Descriptions

Function `mfImage` displays an image file in the plot space. Argument `ax` is an m-by-n-by-3 matrix containing the rgb color codes.

See Also

mfImRead, mfImWrite

mfImRead

Read in image file.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfImRead(char* filename);
```

Descriptions

Function `mfImRead` reads in an image file with the name specified by argument `filename` and stores it in argument `ax` which is an m-by-n-b-3 matrix storing the rgb color codes.

The supported file formats are: bmp, jpeg and png.

See Also

mfImage, mfImWrite

mfImWrite

Write to image file.

Module

```
fgl.h/fgl.lib
```

C++ Prototype

```
void mfImWrite(char* filename, mfArray ax);
```

Descriptions

Function `mfImWrite` writes the rgb color codes stored in the m-by-n-by-3 matrix `ax` into a file with the name specified by argument `filename.`

The supported file format are: bmp, jpeg, tiff, ps and png.

See Also

Elementary 3D Objects

mfMolecule

Draw stick and ball model of molecules.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfMolecule(mfArray loc, mfArray conn[, mfArray rad, mfArray color,
mfArray stick_rad, mfArray stick_col, mfArray resolution]);
```

Descriptions

Function `Molecule` enables you to create three-dimensional stick-and-model of molecules.

`mfMolecule(loc, conn, rad, color, stick_rad, stick_col, resolution)`

- Draw n balls specified by argument `loc` and m sticks specified by argument `conn.` Argument `rad` specifies the radius of each ball, argument `color` specifies the color of each ball, argument `stick_rad` specifies the cylindrical radius of each stick, argument `stick_col` specifies the color of the sticks and argument `resolution` specifies the smoothness of the ball objects.
- Argument `loc` is an n-by-3 array containing the three-dimensional Cartesian coordinates (x, y, z) of each ball center, hence specifying the spatial relationship of each ball. Each ball is numbered according to their respective row index. Thus row number 1 specifies ball number 1. The first column contains the x-coordinates, the second column contains the y-coordinates and the third the z-coordinates respectively. As an example, the array below specifies three balls whose center are located at (0,0,0), (1,1,1) and (0,1,0) respectively.

Argument `loc:`

	x	y	z
ball 1	0	0	0
ball 2	1	1	1
ball 3	0	1	0

- Argument `conn` is an m-by-2 array specifying m number of sticks and the balls connected by each stick. Each stick connects two balls and is labeled according to its row number. Columns of the argument `conn` contain the indices of the balls that each stick connects. For example, the array below specifies 3 sticks connecting ball 1 and ball 2, ball

3 and ball 1, ball 2 and ball 3.

Argument `conn:`

	ball index	ball index
stick 1	1	2
stick 2	3	1
stick 3	2	3

- Argument `rad` is a scalar specifying the radius of all balls or an n-by-1 array specifying the radius of each individual ball respectively. By default, all balls are drawn with a radius of 0.5. As an example, the array below specifies three balls of different sizes, with radius 1, 2 and 3 respectively.

Argument `rad:`

	radius
ball 1	1
ball 2	2
ball 3	3

- Argument `color` contains a string specifying the color of all the balls or an n-by-3 array containing the rgb color code of each ball. The rgb color code is specified as [r, g, b] where 0 < r, g, b < 1.
- The array below specifies three balls of different colors, red, green and blue respectively.

Argument `color:`

	r	g	b
ball 1	0.8	0.1	0.1
ball 2	0.1	0.8	0.1
ball 3	0.1	0.1	0.8

`h = mfMolecule(...)`

- Handle `h` retrieves a handle to the molecule objects created by `mfMolecule(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()`to retrieve the handle of the current graphics object.

You can specify properties of the molecule objects through handle `h` with function `mfGSet.`

The properties available are:

1. location
2. connective

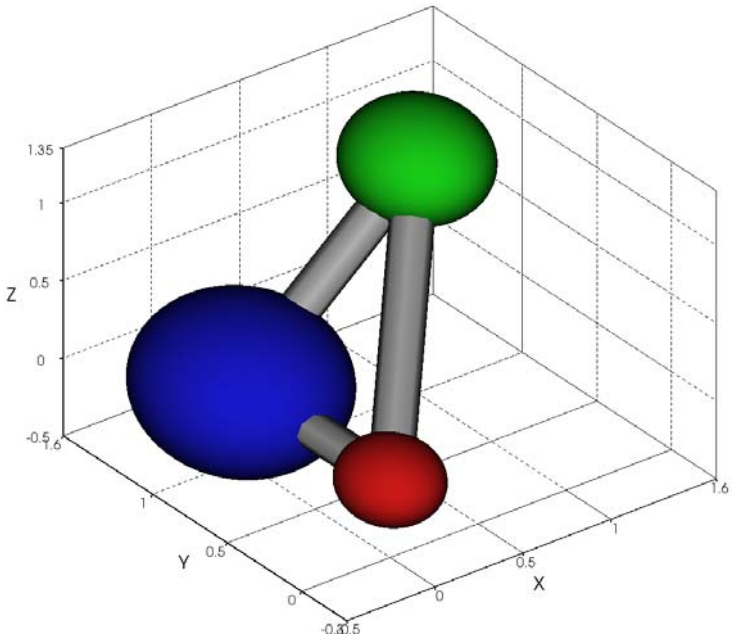
3. radius
4. color
5. stick_radius
6. stick_color
7. resolution

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfMolecule.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"


void main() {

    mfArray loc, conn, rad, color, stick_rad, stick_col, h;

    // Specify locations of three balls using vcat
    loc = mfReshape(mfV(0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0), 3,3);

    // Specify three sticks and their connections
    conn = mfReshape(mfV(1.0, 1.0, 2.0, 2.0, 3.0, 3.0), 3,2);

    // Specify the radius of each ball at radius=0.5,0.7,1.0
    rad = mfV(0.5, 0.7, 1.0);

    // Set the color of each ball to red, green and blue
    color = mfReshape(mfV(0.8, 0.1, 0.1, 0.1, 0.8, 0.1, 0.1, 0.1, 0.8),
3,3);

    // Set the cylindrical radius of each stick to 0.1,0.2,0.3
    stick_rad = mfV(0.1, 0.2, 0.3);

    // Set the color of the stick to grey
    stick_col = mfV(0.7, 0.7, 0.7);

    mfAxis(mfV(-0.5, 1.6, -0.3, 1.6, -0.5, 1.35));

    // Draw the molecules
    h = mfMolecule(loc, conn, rad, color, stick_rad, stick_col);

    // Pause program to view
    mfViewPause();

}
```

Result



See Also

mfSphere

mfFastMolecule

Draw stick and ball model of molecules.

Module

`fgl.h/fgl.lib`

Descriptions

Function `Molecule` enables you to create three-dimensional stick-and-model of molecules.

`mfMolecule(loc, conn, rad, color, stick_rad, stick_col, resolution)`

- Draw n balls specified by argument `loc` and m sticks specified by argument `conn.` Argument `rad` specifies the radius of each ball, argument `color` specifies the color of each ball, argument `stick_rad` specifies the cylindrical radius of each stick, argument `stick_col` specifies the color of the sticks and argument `resolution` specifies the smoothness of the ball objects.
- Argument `loc` is an n-by-3 array containing the three-dimensional Cartesian coordinates (x, y, z) of each ball center, hence specifying the spatial relationship of each ball. Each ball is numbered according to their respective row index. Thus row number 1 specifies ball number 1. The first column contains the x-coordinates, the second column contains the y-coordinates and the third the z-coordinates respectively. As an example, the array below specifies three balls whose center are located at (0,0,0), (1,1,1) and (0,1,0) respectively.

Argument `loc:`

	x	y	z
ball 1	0	0	0
ball 2	1	1	1
ball 3	0	1	0

- Argument `conn` is an m-by-2 array specifying m number of sticks and the balls connected by each stick. Each stick connects two balls and is labeled according to its row number. Columns of the argument `conn` contain the indices of the balls that each stick connects. For example, the array below specifies 3 sticks connecting ball 1 and ball 2, ball 3 and ball 1, ball 2 and ball 3.

Argument `conn:`

ball index	ball index

stick 1	1	2
stick 2	3	1
stick 3	2	3

- Argument `rad` is a scalar specifying the radius of all balls or an n-by-1 array specifying the radius of each individual ball respectively. By default, all balls are drawn with a radius of 0.5. As an example, the array below specifies three balls of different sizes, with radius 1, 2 and 3 respectively.

Argument `rad:`

	radius
ball 1	1
ball 2	2
ball 3	3

- Argument `color` contains a string specifying the color of all the balls or an n-by-3 array containing the rgb color code of each ball. The rgb color code is specified as [r, g, b] where 0 < r, g, b < 1.
- The array below specifies three balls of different colors, red, green and blue respectively.

Argument `color:`

	r	g	b
ball 1	0.8	0.1	0.1
ball 2	0.1	0.8	0.1
ball 3	0.1	0.1	0.8

`h = mfMolecule(...)`

- Handle `h` retrieves a handle to the molecule objects created by `mfMolecule(...).`
- Alternatively, you use procedure `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the molecule objects through handle `h` with function `mfGSet.`

The properties available are:

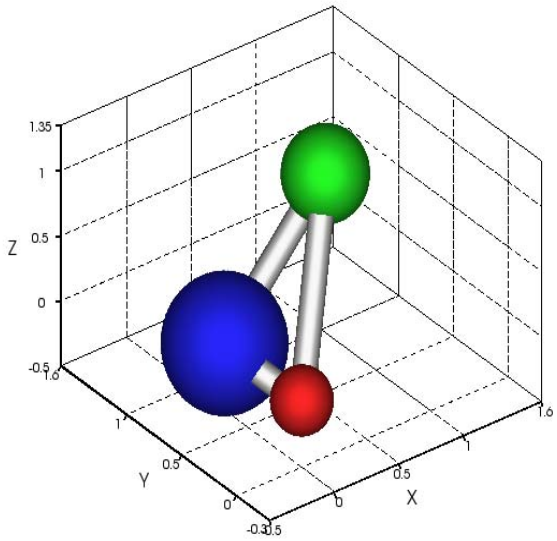
1. location
2. connective
3. radius
4. color
5. stick_radius
6. stick_color

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfFastMolecule.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"


void main() {

    mfArray loc, conn, rad, color, stickRad, stickColor, h;

    // Specify locations of three balls using vcat
    loc = mfReshape(mfV(0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0), 3,3);

    // Specify three sticks and their connections
    conn = mfReshape(mfV(1.0, 1.0, 2.0, 2.0, 3.0, 3.0), 3,2);

    // Specify the radius of each ball at radius=0.5,0.7,1.0
    rad = mfV(0.5, 0.7, 1.0);

    // Set the color of each ball to red, green and blue
    color = mfReshape(mfV(0.8, 0.1, 0.1, 0.1, 0.8, 0.1, 0.1, 0.1, 0.8),
3,3);

    // Set the cylindrical radius of all sticks to 0.2
    stickRad = mfV(0.2, 0.2);

    // Set the color of the stick to grey
    stickColor = mfV(0.7, 0.7, 0.7);

    mfAxis(mfV(-0.5, 1.6, -0.3, 1.6, -0.5, 1.35));

    // Draw the molecules
    h = mfFastMolecule(loc, conn, rad, color, stickRad, stickColor);

    // Pause program to view
    mfViewPause();

}
```

Result



See Also

mfSphere

mfSphere

Draw a sphere.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfSphere([mfArray loc][, mfArray rad][, mfArray color][, mfArray resolution]);
```

Descriptions

Function `mfSphere` draws a sphere with center at `loc`, radius specified by `rad`, color specified by `color` and transparency level specified by `resolution`. All arguments are optional.

```
mfSphere(loc, rad, color, resolution)
```

Argument	Meaning
`loc`	A 1-by-3 mfArray containing the x-, y- and z-coordinates of the sphere center. By default, argument `loc` is set to `[0,0,0]`.
`radius`	An mfArray containing a real number specifies the radius of the sphere. By default, argument `radius` is set to `0.5`.
`color`	An mfArray containing a string specifies the color, e.g. “y”, or a 1-by-3 mfArray contains the rgb codes. By default, argument `color` is set to grey.
`resolution`	An mfArray containing the number of polygons used for modeling the circular object. The higher the

	polygon number, the smoother a sphere appears. The lower the polygon number, the faster a sphere is rendered. By default, the sphere is set to a resolution = 64.

`h = mfSphere(...)`

- Handle `h` retrieves a handle to the sphere object created by `mfSphere(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the sphere object through handle `h` with function `mfGSet.`

The properties available are:

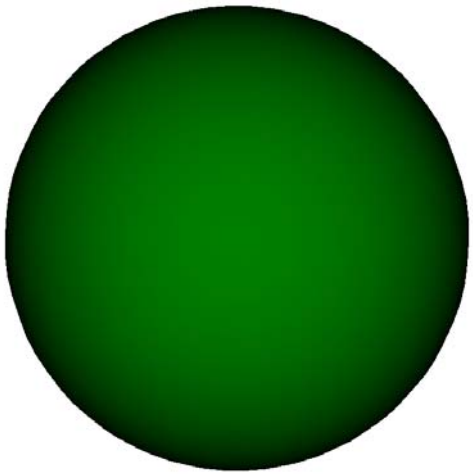
1. location
2. radius
3. color
4. resolution

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfSphere.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray zeros;

    zeros = mfV(0.0, 0.0, 0.0);

    // Center at (0,0,0), Radius = 0.5, Color = "green"
    mfSphere(zeros, 1, "g");

    // Remove current Axis
    mfAxis("off");
    mfAxis("equal");

    // Pause the program for display
    mfViewPause();

}
```

Result



See Also

mfCylinder, mfMolecule, mfCube

mfCube

Draw a Cube.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfCube([mfArray loc][, mfArray size][, mfArray color][, mfArray center]);
```

Descriptions

Function `mfCube` draws a cube with center at `loc,` size specified by `size,` color specified by `color` and transparency level specified by `resolution.` All arguments are optional.

```
mfCube(loc, size, color, center)
```

Argument	Meaning
`pos`	A 1-by-3 mfArray contains the x-, y- and z- coordinates of the cone center. By default, argument `pos` is set to `[0,0,0].`
`size`	A 1-by-3 mfArray specifies the length , width and height of a cube. By default, argument `Size` is set to `[1,1,1].`
`color`	An mfArray containing a string specifies the color, e.g. `"y"`, or a 1-by-3 mfArray containing the rgb codes. By default, argument `color` is set to `grey`.

```
h = mfCube(...)
```

- Handle `h` retrieves a handle to the cube object created by `mfCube(...).`
- Alternatively, you use function `h = mfGetCurrentDraw()`to retrieve the handle of the current graphics object.

You can specify properties of the cube object through handle `h` with function `mfGSet.`

The properties available are:

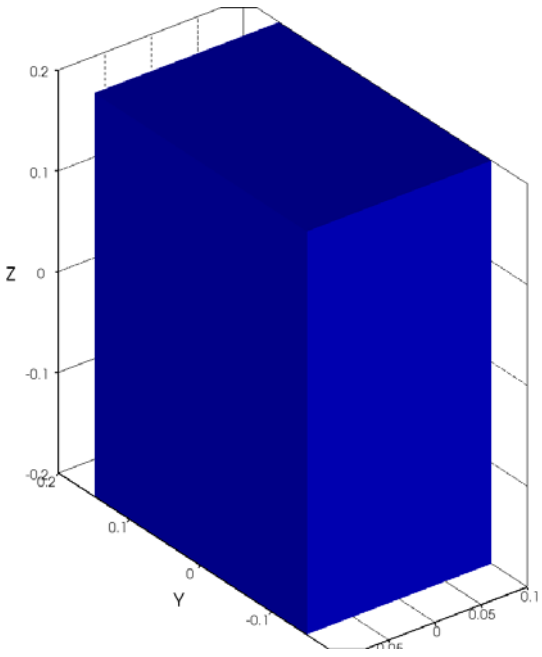
1. location
2. radius
3. color
4. center

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfCube.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"


void main() {

    mfArray zeros, cubesize;

    zeros = mfV(0, 0, 0);
    cubesize = mfV(0.2, 0.3, 0.4);

    // Cube with center at (0,0,0), size of 0.2x0.3x0.4
    // and blue in color
    mfCube(zeros, cubesize, "b");
    mfAxis("equal");

    // Pause the program for display
    mfViewPause();

}
```

Result



See Also

mfCylinder, mfSphere, mfCone

mfCylinder

Draw a cylinder.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfCylinder([mfArray loc][, mfArray rad][, mfArray height][,
mfArray color][, mfArray resolution]);
```

Descriptions

Function `mfCylinder` draws a cylinder with center at `loc,` radius specified by `rad,` color specified by `color` and transparency level specified by `resolution.` All arguments are optional.

```
mfCylinder(loc, rad, height, color, resolution)
```

Argument	Meaning
`pos`	A 1-by-3 mfArray contains the x-, y- and z- coordinates of the cylinder center. By default, argument `pos` is set to `[0, 0, 0].`
`radius`	An mfArray containing a real number specifies the radius of the cylinder. By default, argument `radius` is set to `0.5.`
`height`	An mfArray containing a real number specifies the height of the cylinder. By default, argument `height` is set to `1.0.`
`color`	An mfArray containing a string specifies the color, e.g. `"y"`, or a 1-by-3 mfArray contains the rgb codes. By default, argument `color` is set to `grey`.
`resolution`	An mfArray contains the number of polygons used for modeling the circular object. The higher the polygon number, the smoother a sphere appears. The lower the polygon number, the faster a sphere is rendered. By default, the sphere is set to a `resolution = 64.`

```
h = mfCylinder(...)
```

- Handle `h` retrieves a handle to the cylinder object created by `mfCylinder(...).`

- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the cylinder object through handle `h` with function `mfGSet.`

The properties available are:

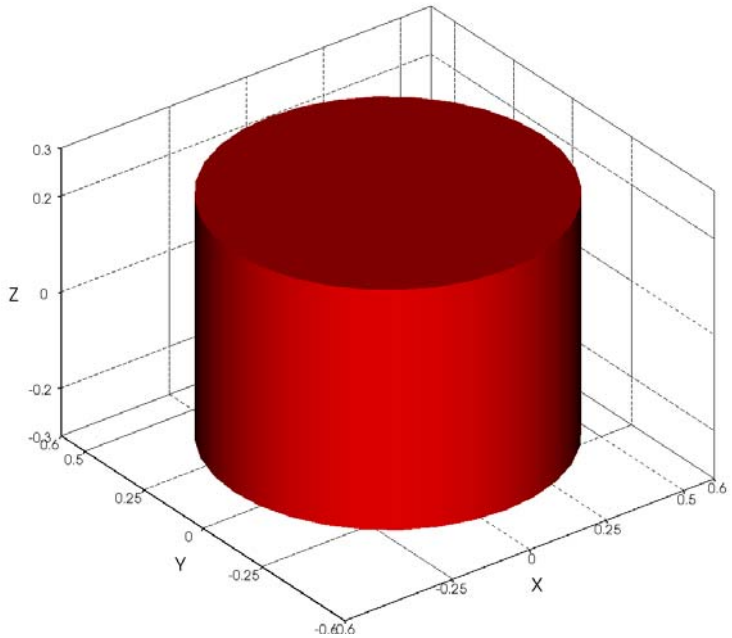
1. location
2. radius
3. height
4. color
5. resolution
6. center

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfCylinder.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray zeros;

    zeros = mfV(0, 0, 0);

    // Cube with center at (0,0,0), size of 0.2x0.3x0.4
    // and blue in color
    mfCylinder(zeros, 0.5, 0.5, "r");

    mfCamZoom(1.2);

    // Pause the program for display
    mfViewPause();

}
```

Result



See Also

mfSphere, mfCone, mfCube

mfCone

Draw a Cone.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfCone([mfArray loc][, mfArray rad][, mfArray height][, mfArray color][, mfArray resolution]);
```

Descriptions

Function `mfCone` draws a Cone with center at `loc`, radius specified by `rad`, color specified by `color` and transparency level specified by `resolution`. All arguments are optional.

```
mfCone(loc, rad, height, color, resolution)
```

```
h = mfCone(...)
```

- Handle `h` retrieves a handle to the cone object created by `mfCone(...)`.
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the cone object through handle `h` with function `mfGSet`.

The properties available are:

1.location 2.radius 3.height 4.color 5.resolution 6.center

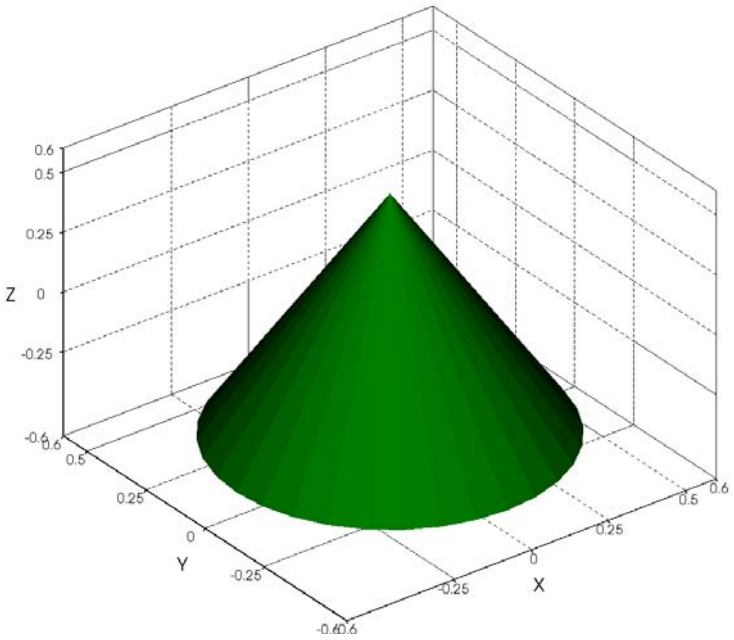
Example

Code

```
// ************************************************
// AnCAD example file
// file: mfCone.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {
    mfArray zeros;
    zeros = mfV(0.0, 0.0, 0.0);
    // Cone with location (0,0,0), radius = 0.5, height = 1.0,
```

```
    // color = green.
    mfCone(zeros, 0.5, 1.0, "g");

    mfCamZoom(1.2);

    // Pause the program for display
    mfViewPause();

}
```

Result



See Also

mfCylinder, mfSphere, mfCube

mfAxisMark

3-directional axis mark on arbitrary point.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfAxisMark([mfArray loc][, mfArray length][, mfArray radius]);
mfArray mfAxisMark([mfArray loc][, mfArray length][, mfArray radius]);
```

Descriptions

Function `mfAxisMark` draws a 3-directional axis mark on any arbitrary point in the plot space. Its purpose is to .

`mfAxisMark(loc, length, radius)`

- Draws a 3-directional axis mark on the point specified by argument `loc` with length specified by argument `length` and thickness specified by argument `radius`.
- By default, location is [0, 0, 0], length is 1 and radius is 0.1.

`h = mfAxisMark(...)`

- Handle `h` retrieves a handle to the axis mark object created by `mfAxisMark(...)`.
- Alternatively, you use function `h = mfGetCurrentDraw()` to retrieve the handle of the current graphics object.

You can specify properties of the molecules object through handle `h` with function `mfGSet`.

The properties available are:

1. symmetric = `"true"` or `"false"`
If symmetric is on, the axes extend to negative values.
2. location
3. length
4. radius

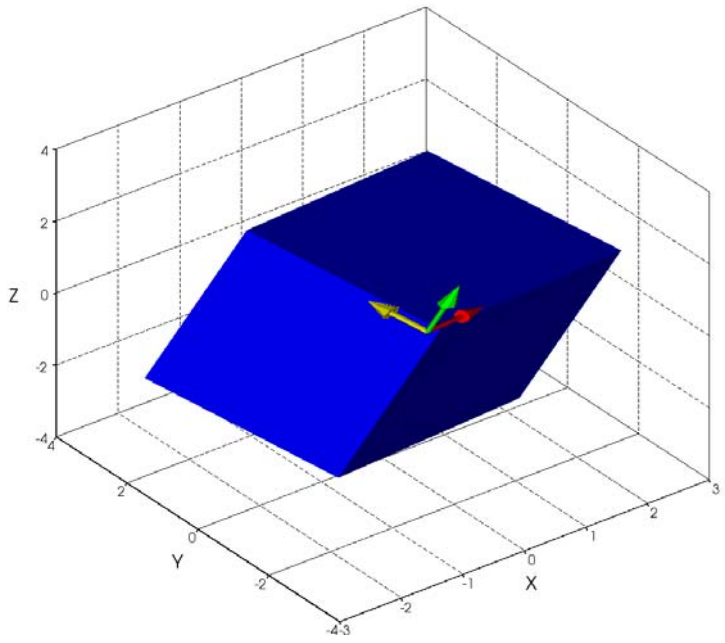
Example

Code

```
// ************************************************
// AnCAD example file
// file: mfAxisMark.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
```

```
#include "fgl.h"

void main() {

    mfArray cubesize, center, h, loc, length, radius;
    cubesize = mfV(4, 4, 4);
    center = mfV(0, 0, 0);
    loc = mfV(-1.0, -3.0, 2.1);
    length = 1 ;
    radius = 0.05;

    h = mfCube(center, cubesize, "b");
    mfObjOrientation(h, 15, 15, 15);
    mfHold("on");
    h = mfAxisMark(loc, length, radius);
    mfObjOrientation(h, 15, 15, 15);
    mfView("equal");
    mfViewPause();
}
```

Result



See Also

Property Setting

mfGSet

Set property of specified graph.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfGSet(mfArray handle, mfArray property1, mfArray value1[, mfArray
property2, mfArray value2, ...]);
```

Descriptions

Function `mfGSet` sets the property of a graphics object whose handle is given by `handle`. You can set various properties of a graph object through the function .

`mfGSet(handle, property, value)`

- Argument `property` is a string specifying the target property to be updated. Argument `value` is an mfArray containing the data to be updated. For example, you can input "xdata" for x-coordinate, "ydata" for y-coordinate, "zdata" for z-coordinate if you want to update the coordinates of a graphics object.

`mfGSet(handle, property, value, property2, value2, ...)`

- Multiple properties can be updated in one statement as above.

- The table below lists the common properties available for updating through function `mfGSet`.

Property	Description	Apply to
`Xdata`	Specify `x` data as target for updating	Almost all two-dimensional and three-dimensional graphics objects
`Ydata`	Specify `y` data as target for updating	Almost all two-dimensional and three-dimensional graphics objects
`Zdata`	Specify `z` data as target for updating	Almost all three-dimensional graphics objects

Cdata	Specify color vector, c as target for updating	Almost all two-dimensional and three-dimensional graphics objects
Udata	Specify velocity in x-direction, u as target for updating.	mfQuiver, mfQuiver3, mfStreamLine, etc.
Vdata	Specify velocity in y-direction, v as target for updating.	mfQuiver, mfQuiver3, mfStreamLine, etc.
Wdata	Specify velocity in z-direction, w as target for updating.	mfQuiver3, mfStreamLine

Note: Not all of the available properties are listed here, please refer to the description of each graphical function for a supplementary listing of properties.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfGSet.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"


void main() {

    mfArray x, y, h;
    int i;

    // Construct and initialize the mfArrays for plotting.
    x = mfLinspace(-MF_PI, MF_PI, 30);
    y = mfCos(x);

    // Plot the initial figure and get its handle.
    mfPlot(x, y, "ro");
    mfBackgroundColor(1, 1, 1);
    h=mfGetCurrentDraw();

    // Set up an iteration loop for the range of data you
    // wish to observe through animation.
    for(i = 1; i <= 1000; i++)
    {
        y=mfCos(x+0.02*i);
        // Within the iteration loop, use function mfGSet to update
        // the targeted data of the current draw.
        mfGSet(h, "ydata", y);
        // Update the current Graphics Viewer by using function
        // mfDrawNow.
        mfDrawNow();
    }

    // Pause the program to observe figure.
```

```
    mfViewPause();
}
```

See Also

mfDrawMaterial

Set transparency reflectance, ambient reflectance, diffuse reflectance and specular reflectance of a draw object.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfDrawMaterial(mfArray handle, char* target, char* property, mfArray value);
```

Descriptions

Function `mfDrawMaterial` sets the color component and the transparency reflectance, ambient reflectance, diffuse reflectance and specular reflectance of the draw object's surface and edge. Each reflectance is specified as a level of intensity ranging from 0 to 100. The resultant lighting effect is produced by applying the intensity levels of the reflectances to the color component.

For example, if the intensity of the draw object's ambient reflectance is set to be 50 and the color component is set to be [1, 1, 1], the draw object's ambient color component becomes [0.5, 0.5, 0.5].

`mfDrawMaterial(handle, target, property, value)`

- You can perform the operation on the draw object's surface, edge or both by specifying argument `target` as "surf", "edge" or "both".
- Arguments `property` and `value` can be:

Property	Meaning
“trans”	Transparency reflectance. The corresponding argument value can be an integer or an mfArray containing an integer that ranges from 0 to 100.
“ambient”	Ambient reflectance. The corresponding argument value can be an integer or an mfArray containing an integer that ranges from 0 to 100.
“diffuse”	Diffuse reflectance. The corresponding argument value can be an integer or an mfArray

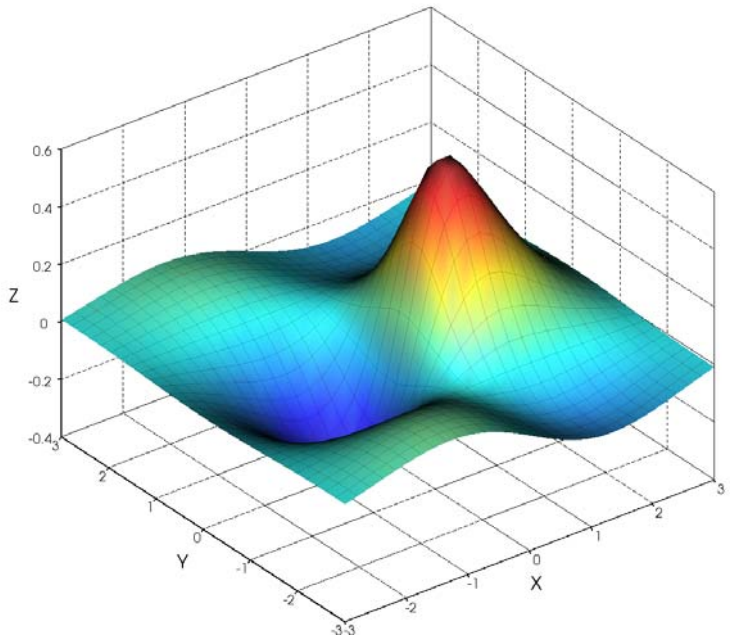
	containing an integer that ranges from 0 to 100.
“specular”	Specular reflectance. The corresponding argument value can be an integer or an mfArray containing an integer that ranges from 0 to 100.
“color”	Color component. The corresponding argument value can be an mfArray containing the rgb vector [r, g, b] or an mfArray containing the string that is specified as “on” or “off”.
“colormap”	Turn the colormap on or off. The corresponding argument value can be an mfArray containing the string that is specified as “on” or “off”.
“visible”	Turn the surface or edge on or off. The corresponding argument value can be an mfArray containing the string that is specified as “on” or “off”.
“smooth”	Interpolate to Gouraud shading. The corresponding argument value can be an mfArray containing the string that is specified as “on” or “off”.
“line_width”	Line width of edge. The corresponding argument value can be an integer or an mfArray containing an integer.
“line_style”	Line style of edge. The corresponding argument value can be an mfArray containing a string that is specified as "solid", "dashed", "dotted" or "dashdot".

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfDrawMaterial.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray a, x, y, z, h;

    a = mfLinspace(-3, 3, 30);
    mfMeshgrid( mfOut(x, y), a);
    z=mfSin(x)*mfCos(y)/(x*(x-0.5)+(y+0.5)*y+ 1);

    h = mfSurf(x, y, z);
    mfDrawMaterial(h, "surf",
                   "visible", "on",
                   "smooth", "on",
                   "colormap", "on",
                   "ambient", 0,
                   "diffuse", 75,
                   "specular", 25);
    mfDrawMaterial(h, "edge",
                   "color", mfV(1,0,0),
                   "smooth", "on",
                   "colormap", "off",
                   "ambient", 0,
                   "diffuse", 0,
                  "diffuse", 0,
                   "specular", 0,
                   "trans", 90);

    mfViewPause();
}
```

Result



See Also

mfDrawTexture

Texture mapping.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfDrawTexture(mfArray handle, char* property1, mfArray value1[,
char* property2, mfArray value2, ...]);
```

Descriptions

Function `mfDrawTexture` places a texture on a graphics object by mapping the texture coordinates to the object's coordinates. The texture coordinates comprise two coordinates namely s- and t-coordinates which are vectors of values ranging from 0 to 1. They correspond to the object's x- and y-coordinates in order to determine which texel in the texture is mapped to which vertex.

`mfDrawTexture(handle, property, value)`

- Arguments `property` and `value` can be:

Property	Meaning
"enable"	Enabling or disabling the texture-mapping. The corresponding argument value can be “on” or “off”.
"map"	Specifying the texture file. The corresponding argument value specifies the name of a bitmap file(e.g. texture.bmp).
"coord_s"	Texture’s s-coordinate. The corresponding argument value is a vector of values ranging from 0 to 1 which specifies the way of mapping.
"coord_t"	Texture’s t-coordinate. The corresponding argument value is a vector of values ranging from 0 to 1 which specifies the way of mapping.

-

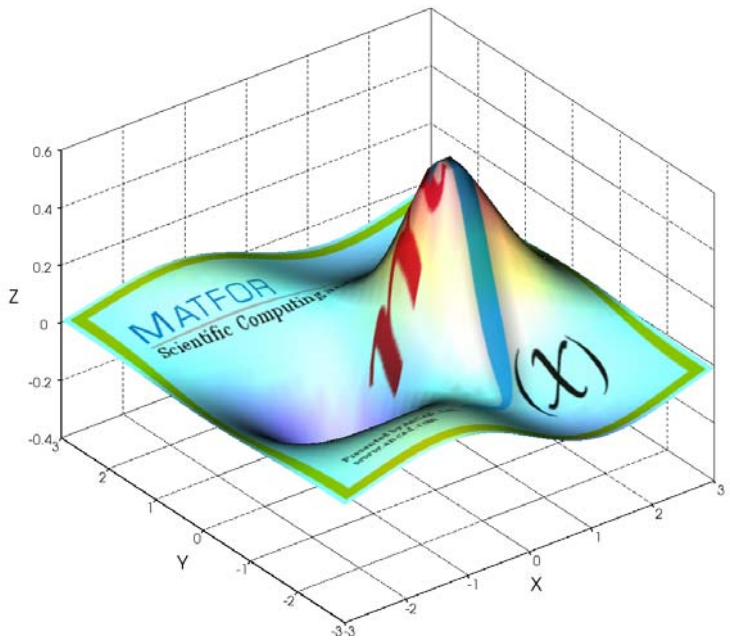
Example

Code

```
// ************************************************
```

```
// AnCAD example file
// file: mfDrawMaterial.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray a, x, y, z, h;

    a=mfLinspace(-3, 3, 30);
    mfMeshgrid( mfOut(x, y), a);
    z=mfSin(x)*mfCos(y)/(x*(x-0.5)+(y+0.5)*y+1);

    h = mfSurf(x, y, z);
    mfDrawTexture(h, "map", "ancad.bmp");
    mfDrawMaterial( h, "surf",
                    "smooth", "on",
                    "colormap", "on",
                    "ambient", 0,
                    "diffuse", 15,
                    "specular", 85);
    mfDrawMaterial(h, "edge", "visible", "off");
    mfViewPause();
}
```

Result



See Also

mfIsValidDraw

Check validity of draw object.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfIsValidDraw(mfArray handle);
```

Descriptions

Function `mfIsValidDraw` returns the validity of a draw object which is associated with argument `handle.` The output is an mfArray containing logical data. It returns true if the draw object still exists, false otherwise.

See Also

mfGetCurrentDraw

Return handle of current draw object.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfGetCurrentDraw();
```

Descriptions

Function `mfGetCurrentDraw` returns the handle of current draw object.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfGetCurrentDraw.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray a, b, c, x, y, z, theta, phi, h;
    int i;

    a = mfLinspace(-MF_PI/2, MF_PI/2, 31);
    b = mfLinspace(-MF_PI, MF_PI, 31);
    c = mfOnes(31, 31);
    mfMeshgrid(mfOut(phi, theta), a, b);
    x = mfCos(phi)*mfCos(theta);
    y = mfCos(phi)*mfSin(theta);
    z = mfSin(phi);

    // Plot the graph you wish to animate
    mfSurf(x, y, z);
    mfAxis(mfV(-MF_PI, MF_PI, -MF_PI, MF_PI, -MF_PI, MF_PI));
    mfShading("interp");
    mfAxis("off");

    // Get handle of current draw
    h = mfGetCurrentDraw();

    // Use a Do Loop to change the value of x, y and z.
    for(i = 1; i <= 30; i++)
    {
        x = mfCos(phi)*mfCos(theta+i*0.1);
        y = mfCos(phi)*mfSin(theta+i*0.1)+0.05*i;
        z = mfSin(phi)+mfSin(c*i);
        // Set the x,y, and z-data of the current graph
        mfGSet(h, "xdata", x, "ydata", y, "zdata", z);
        // Draw graph on Graphics Viewer
        mfDrawNow();
```

```
    }

    // Pause program to view graph. If this statement is
    // not added, the Graphics Viewer closes once animation
    // is completed.
    mfViewPause();

}
```

See Also

mfRemoveDraw

Remove draw object from plot space.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfRemoveDraw(mfArray handle1[, mfArray handle2, ...]);
```

Descriptions

Function `mfRemoveDraw` removes specific draw objects from plot space.

`mfRemoveDraw(handle1, handle2, ...)`
Removes the draw objects that are associated with handles specified in the arguments.

See Also

mfSetDrawName

Name draw object.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfSetDrawName(mfArray handle, char* name);
```

Descriptions

Function `mfSetDrawName` sets the name of a draw object that is associated with argument `handle.` By default, the name of a draw object is set to its draw type followed by an incremental integer.

The purpose of giving each draw object a name is for distinguishing between the draw objects. It allows you to perform operations (e.g. custom shading or view draw object data) on a specific draw object when there are two or more draw objects present in the same subplot.

See Also

Simple GUI

mfShowMessage

Pop up message dialog box.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
void mfShowMessage(char* msg);
```

Descriptions

Function `mfShowMessage` pops up a dialog box displaying a message.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfShowMessage.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "fgl.h"

void main() {

    mfShowMessage("Show Message Test");

}
```

Result

See Also

mfInputString

Pop up string insertion dialog box.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfInputString(char* msg, char* default_string);
```

Descriptions

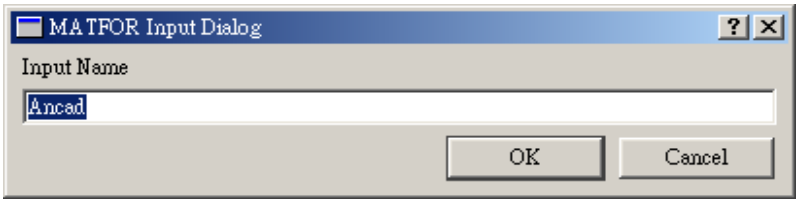
Function `mfInputString` pops up a dialog box displaying a message. The input string is passed back to the output argument `string.`

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfInputString.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "fgl.h"

void main() {

    mfArray str;
    str = mfInputString("Input Name", "Ancad");
    mfShowMessage(str);

}
```

Result



See Also

mfInputValue

Pop up value insertion dialog box.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfInputValue(char* msg, int default_value);
```

Descriptions

Function `mfInputValue` pops up a dialog box displaying a message. The input value is passed back to the output argument `value.`

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfInputValue.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray val;
    val = mfInputValue("Input Number", 10);

    mfDisplay(val, "Number");

}
```

See Also

mfInputVector

Pop up vector insertion dialog box.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfInputVector(char* msg, mfArray default_vector);
```

Descriptions

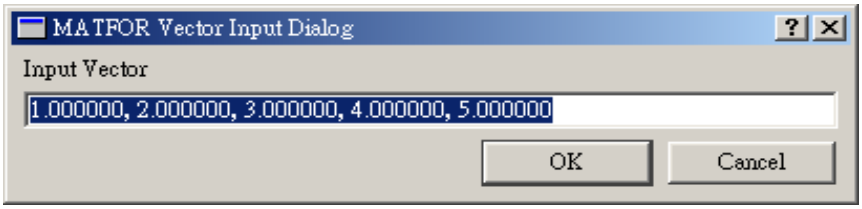
Function `mfInputVector` pops up a dialog box for inserting the entries of a vector. The input entries are passed back to the output argument `vector.`

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfInputVector.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray vec;
    vec = mfInputVector("Input Vector", mfV(1, 2, 3, 4, 5));
    mfDisplay(vec);

}
```

Result



See Also

mfInputMatrix

Pop up matrix insertion dialog box.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfInputMatrix(char* msg, mfArray default_matrix);
```

Descriptions

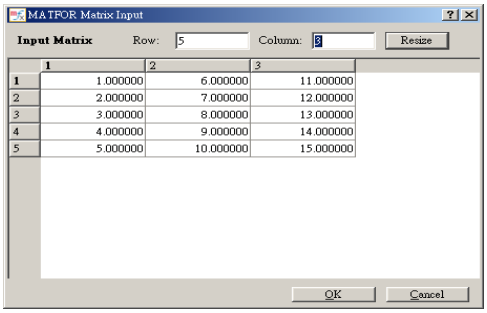
Function `mfInputVector` pops up a dialog box for inserting the entries of a matrix. The input entries are passed back to the output argument `matrix.`

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfInputMatrix.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray x, m;

    // Create matrix mfArray
    x = mfM(5, 3, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15);

    m = mfInputMatrix("Input Matrix", x);
    mfDisplay(m);

}
```

Result



See Also

mfFileDialog

Pop up file open dialog box.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfFileDialog(char* filename, mfArray filefilter);
```

Descriptions

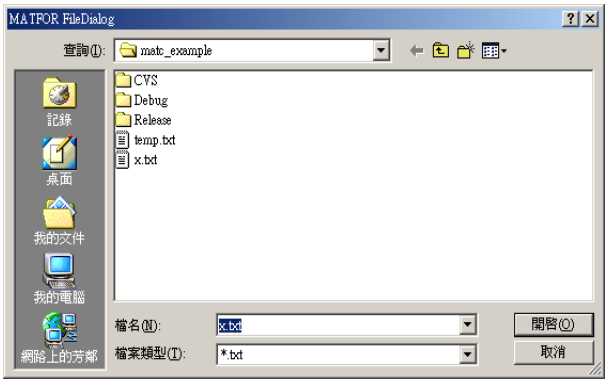
Function `mfFileDialog` pops up a file open dialog box for locating a file.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfFileDialog.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray a, str;

    str = mfFileDialog("x.txt", "*.txt");
    a = mfLoadAscii(str.ToString().c_str());
    mfDisplay(a, "a");
}
```

Result



See Also

mfInputYesNo

Pop up yes-no query dialog box.

Module

`fgl.h/fgl.lib`

C++ Prototype

```
mfArray mfInputYesNo(char* msg, double default_value);
```

Descriptions

Function `mfInputYesNo` pops up a yes-no dialog box. The chosen option is passed back to the output argument `value.`

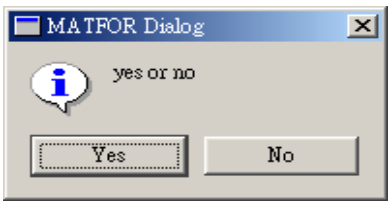
Argument `default_value` can be either 0 or 1.

Example

Code

```
// ************************************************
// AnCAD example file
// file: mfInputYesNo.cpp
//
// Copyright 2003 AnCAD
// ************************************************

#include "cml.h"
#include "fgl.h"

void main() {

    mfArray val;

    val = mfInputYesNo("yes or no");
    mfDisplay(val, "val");
}
```

Result



See Also

Index

U

V

W