

## 第15章 MATLAB与其他编程语言结合

MATLAB可以和其他编程语言一起使用，可以调用 FORTRAN或C程序。反过来FORTRAN或C也可以调用MATLAB程序。这样，快速的编译程序就可以利用 MATLAB中强大的矩阵或图形命令，通过编写部分的 C或FORTRAN程序，并进行编译，就可以避免 MATLAB程序的瓶颈现象。

MATLAB还可以结合使用其他的应用程序，如 Microsoft Word for Windows。这将在本章的最后一节讨论，这主要取决于计算机的系统 and 安装的应用程序。

### 15.1 介绍MATLAB和FORTRAN或C

MATLAB可以被FORTRAN或C语言程序调用，它也可以调用 FORTRAN或C语言程序。如果MATLAB程序运行速度很慢，后者对此很有用。因为 MATLAB是一个解释性语言，所以当运行程序时就是解释它的命令。这样有时会导致程序的运行速度很慢，如 for-loops循环。

在FORTRAN 77和C中可以使用MATLAB库，也可以用FORTRAN 90或C++对它们进行链接。

除非特别需要，一般不推荐编写 FORTRAN或C程序。MATLAB的优点在于可以用高级的形式描述出操作，而程序员不必担心循环的次数和一些其他细节问题。

被MATLAB调用的程序必须在编译后转换成 MEX文件，这样才能被MATLAB调用。在编译时它们和M文件一样使用。

在2.8节中讲到了由MATLAB创建的二进制文件。它们是以 MAT文件形式被调用的，在 C或FORTRAN语言的库中有用来读和写二进制文件的程序。注意，这些文件可以在不同的平台间传递，例如，用户可以读取在 Windows环境下建立的MAT文件到UNIX环境中。在15.4节中介绍了如何在MATLAB中读或写其他的二进制文件。这对有特殊格式要求的程序很有用。

MATLAB编译器、C数学库和C++数学库可以从 MathWorks公司买到。首先可以作为自动MEX文件生成器或C源代码生成器使用，结合C数学库一起生成应用程序。

在C中编写MATLAB程序，数据通过指针来访问。在其他编程语言中调用 MATLAB程序，就要求使用指针。

在MATLAB 5中，所有变量类型，如标量、向量、矩阵、字符串、细胞矩阵和结构，都以mxArrays形式来保存，所有的数据操作都通过这些 mxArrays来完成。

MATLAB 5中新的数据类型，也就是多维数组、细胞矩阵和结构只能在 C中使用，而不能在FORTRAN中使用。

在C或FORTRAN中使用的MATLAB程序主要分四类：

- |     |                            |
|-----|----------------------------|
| mx  | 可操作的mxArrays。              |
| mat | MAT文件。                     |
| eng | MATLAB工程文件。                |
| mex | MEX程序，在MATLAB环境中完成一些操作的程序。 |

在下面几节中将举一些例子。这些例子基本上说明了 MATLAB和C或FORTRAN是如何相

互调用的。它们已在运行 Sun OS 5.5.1 的工作站上和 Soloais CDE 1.0.2 版的 Windows 系统中编译通过。对于每一种系统而言，编程的思想都是一样的。然而还是有一些重要的细节方面是不相同的。这就是为什么 MATLAB 中的例程很有趣的原因，它们可以在库中找到：

```
.../matlab52/extern/examples
```

路径中的三个点，...，表示这部分路径与系统有关。

例程的文档可以用 MATLAB 命令 `helpdesk` 获得。而且还有 MATLAB 手册《应用程序接口指南》。

## 15.2 MATLAB和C

为了使 C 和 MATLAB 混合编程，重要的是使用的 C 编译器以 ANSI C 标准进行编译。

### 15.2.1 C 中对 mxArray 的操作

用下面描述的程序可以对 mxArray 进行操作。为了使用这些程序，在程序中必须嵌入头文件 `matrix.h`，也就是在程序的开始包含下面一行：

```
#include "matrix.h"
```

下面表中的程序用来分配和释放内存。一个好的编程习惯就是及时释放不再使用的内存。不必使用 MATLAB 程序来创建数据结构，因为在程序结束时 MATLAB 会自动地完成（可见命令集 195 中程序 `mexMakeArrayPersistent` 和 `mexMakeMemoryPersistent`）。

#### 命令集 175 C 中的内存管理

```
void *mxCalloc(size_t n, size_t size);
```

分配内存。参数 *n* 表示分配的元素个数，*size* 表示每个元素的字节数。如果分配成功，返回一个指向已分配内存的开始位置的指针；否则返回 `NULL`。在程序中必须嵌入库文件 `<stdlib.h>`。当不再使用时用 `mxFree` 来释放内存。

```
void mxSetAllocFcns(calloc_proc callocfcn, free_proc  
freefcn, realloc_proc reallocfcn, malloc_proc mallocfcn);
```

在非 MEX 程序中用来释放内存。使用 `helpdesk` 可得更多信息。

```
void mxFree(void *ptr);
```

释放 *ptr* 指向的内存空间。

```
void *mxRealloc(void *ptr, size_t size);
```

重新分配用 `mxCalloc` 分配的内存。参数 *ptr* 是指向内存开始位置的指针，*size* 是分配元素的个数。如果分配成功，返回得到指向分配内存开始位置的指针；否则返回 `NULL`。在程序中必须嵌入库文件 `<stdlib.h>`。用 `mxFree` 来释放不再使用的内存。

```
void mxDestroyArray(mxArray *array_ptr);
```

释放 *array\_ptr* 指向的 mxArray 内存。

下面的常用程序用来管理和检查 mxArray，如命名、重构和检查它们的类型。

#### 命令集 176 C 中处理 mxArray 的常用程序

```
mxComplexity
```

是一个枚举数据类型，用来表示 mxArray 的虚数元素。它的值可以为 mxCOMPLEX(复数mxArray)或mxREAL(其他)。

mxClassID

是一个枚举数据类型，用来表示 mxArray 的类型。有下列选项：

mxCELL_CLASS,	细胞类型。
mxSTRUCT_CLASS,	结构类型。
mxOBJECT_CLASS,	用户自定义类型。
mxCHAR_CLASS,	字符串类型。
mxSPARSE_CLASS,	稀疏矩阵。
mxDOUBLE_CLASS,	双精度浮点小数。
mxSINGLE_CLASS,	单精度浮点小数。
mxINT8_CLASS,	8位整数。
mxUINT8_CLASS,	8位无符号整数。
mxINT16_CLASS,	16位整数。
mxUINT16_CLASS,	16位无符号整数。
mxINT32_CLASS,	32位整数。
mxUINT32_CLASS,	32位无符号整数。
mxUNKNOWN_CLASS,	未知类型。

```
mxClassID mxGetClassID(const mxArray *array_ptr);
```

返回array\_ptr指向的mxArray类型；见上。

```
const char *mxGetClassName(const mxArray *array_ptr);
```

同上，返回字符串形式的类型。

```
bool mxIsClass(const mxArray *array_ptr, const char *name);
```

如果array\_ptr指向的mxArray有字符串name表示的类型，则返回真。字符串name相对应于上面的类型(见mxClassID):“cell”、“struct”、“char”、“sparse”、“double”、“single”、“int8”、“uint8”、“int16”、“uint16”、“int32”和“uint32”。它还可以是自定义的类型名。

```
const char *mxGetName(const mxArray *array_ptr);
```

返回包含array\_ptr指向的mxArray名字的字符串。

```
double mxGetScalar(const mxArray *array_ptr);
```

返回array\_ptr指向的mxArray的第一个实数元素的值。总是返回一个double型值。如果mxArray是一个结构或细胞类型，则返回0.0；如果mxArray是一个稀疏矩阵类型，则返回第一个非零实数元素的值；如果mxArray为空，则返回一个不确定值。

```
mxArray *mxDuplicateArray(const mxArray *in);
```

复制in指向的mxArray，并返回指向复制mxArray的指针。当它不再使用时，用mxDestroyArray来释放它；见命令集175。

```
int mxGetNumberOfElements(const mxArray *array_ptr);
```

返回array\_ptr指向的mxArray的元素个数。使用mxGetClassID来找出元素类型。

```
int mxGetElementSize(const mxArray *array_ptr);
```

返回保存array\_ptr指向的mxArray中一个元素需要的字节数。如果mxArray是细胞或结构类型，则返回指向它们的指针大小。如果操作失败，返回0。

```
int mxGetNumberOfDimensions(const mxArray *array_ptr);
```

返回`array_ptr`指向的`mxArray`中的维数，这个数总是不小于2。

```
const int *mxGetDimensions(const mxArray *array_ptr);
```

返回一个整数向量的指针，包含`array_ptr`指向的`mxArray`的每一维的元素个数。

```
int mxSetDimensions(mxArray *array_ptr, const int *size, int ndims);
```

用来重构或增加/减少`array_ptr`指向的`mxArray`的元素。参数`ndims`表示维数范围，`size`表示一个整数向量的指针，包含每维中需要的元素个数。如果操作成功，返回0；否则返回1。如果要增加或减少元素，则必须进行分配/释放内存。用`helpdesk`可得更多信息。

```
int mxGetM(const mxArray *array_ptr);
```

返回‘行’数，也就是`array_ptr`指向的`mxArray`的第一维中元素的个数。

```
void mxSetM(mxArray *array_ptr, int m);
```

用来重构或增加/减少`array_ptr`指向的`mxArray`中的‘行’数。参数`m`表示规定的‘行’数，见`mxSetDimensions`。

```
int mxGetN(const mxArray *array_ptr);
```

返回‘列’数，也就是`array_ptr`指向的`mxArray`的第二维中元素的个数。

```
void mxSetN(mxArray *array_ptr, int n);
```

用来重构或增加/减少`array_ptr`指向的`mxArray`中的‘列’数。参数`n`表示规定的‘列’数，见`mxSetDimensions`。

```
bool mxIsEmpty(const mxArray *array_ptr);
```

如果`array_ptr`指向的`mxArray`为空，就返回真。

```
bool mxIsFromGlobalWS(const mxArray *array_ptr);
```

如果`array_ptr`指向的`mxArray`是从MATLAB全局工作区中复制得到，则返回真。

```
bool mxIsNumeric(const mxArray *array_ptr);
```

如果`array_ptr`指向的`mxArray`是数字或字符串类型，则返回真。

```
bool mxIsInt8(const mxArray *array_ptr);
```

8位整数。

```
bool mxIsUint8(const mxArray *array_ptr);
```

8位无符号整数。

```
bool mxIsInt16(const mxArray *array_ptr);
```

16位整数。

```
bool mxIsUint16(const mxArray *array_ptr);
```

16位无符号整数。

```
bool mxIsInt32(const mxArray *array_ptr);
```

32位整数。

```
bool mxIsUint32(const mxArray *array_ptr);
```

32位无符号整数。

```
bool mxIsSingle(const mxArray *array_ptr);
```

单精度浮点小数。

```
bool mxIsDouble(const mxArray *array_ptr);
```

双精度浮点小数。

```
bool mxIsComplex(const mxArray *array_ptr);
```

复数。如果 *array\_ptr* 指向的 *mxArray* 按函数指定的格式存储数据，则返回真。

```
int mxCalcSingleSubscript((const mxArray *array_ptr, int nbus, int *subs)
```

将多维中的坐标向量转换成字典序中的标量下标。参数 *nsubs* 通常表示 *array\_ptr* 指向的 *mxArray* 中的维数，*subs* 表示要转换坐标向量的指针。用 *helpdesk* 可得更多信息。

下面的程序用来创建和处理二维  $m \times n$  满矩阵，矩阵的元素是双精度浮点小数。

#### 命令集177 C中满矩阵的处理

```
mxArray *mxCreateDoubleMatrix(int m, int n, mxComplexity Complexflag);
```

和 *mxCreateCellMatrix* 相似(见命令集181)，但是这里创建的是二维  $m \times n$  双精度浮点小数矩阵。如果矩阵中元素有复数，则参数 *Complexflag* 是 *mxCOMPLEX* 类型，否则是 *mxREAL* 类型。

```
double *mxGetPr(const mxArray *array_ptr);
```

返回 *array\_ptr* 指向的 *mxArray* 中第一个实数元素的指针。如果矩阵中没有任何实数元素，则返回 *NULL*。

```
void mxSetPr(mxArray *array_ptr, double *pr);
```

设置 *array\_ptr* 指向的 *mxArray* 中的实数元素。参数 *pr* 包含应该使用值的向量指针，这个向量必须用 *mxCalloc* 来动态地分配；见命令集 175。

```
double *mxGetPi(const mxArray *array_ptr);
```

和 *mxGetPr* 相似，但是是对虚数元素。

```
void mxSetPi(mxArray *array_ptr, double *pi);
```

和 *mxSetPr* 相似，但是是对虚数元素。

下面的程序用来创建和处理二维  $m \times n$  的稀疏矩阵，矩阵元素是双精度浮点小数。

#### 命令集178 C中稀疏矩阵的处理

```
mxArray *mxCreateSpares(int m, int n int nzmax, mxComplexity ComplexFlag);
```

创建一个二维  $m \times n$  的稀疏矩阵。参数 *nzmax* 表示矩阵中非零元素的个数。如果矩阵中有复数元素，则参数 *ComplexFlag* 是 *mxCOMPLEX* 类型；否则是 *mxREAL* 类型。如果创建成功，返回指向这个矩阵的指针；否则返回 *NULL*。当它不再使用时，用 *mxDestroyArray* 来释放所占内存；见命令集 175。

```
int mxGetNzmax(const mxArray *array_ptr);
```

返回 *array\_ptr* 指向的稀疏矩阵 *mxArray* 中的 *nzmax* 值(见上)。如果发生任何错误，都返回一个不确定数。

```
void mxSetNzmax(mxArray *array_ptr, int nzmax);
```

设置`array_ptr`指向的稀疏矩阵 `mxArray` 中的 `nzmax` 值(见上)。如果 `nzmax` 改变, 那么向量 `ir`、`pr` 和 `pi` 的大小(如果它们存在)也将随着改变。用 `helpdesk` 可得更多信息。

```
int *mxGetIr(const mxArray *array_ptr);
```

返回一个包含有行数的整数向量指针, 其中第一行有数字 0, `array_ptr` 指向的稀疏矩阵 `mxArray` 中有非零元素。如果操作失败, 返回 `NULL`。

```
void mxSetIr(mxArray *array_ptr, int *ir);
```

定义 `array_ptr` 指向的稀疏矩阵 `mxArray` 中有非零元素的行。参数 `ir` 是一个整数向量指针, 包含使用的行数, 这些行必须按列序来存储。在 0 处开始行计数。用 `helpdesk` 可得更多信息。

```
int *mxGetJc(const mxArray *array_ptr);
```

和 `mxGetIr` 相似, 但是返回的整数向量指针直接表示有非零元素的列来。用 `helpdesk` 可得更多信息。

```
void mxSetJc(mxArray *array_ptr, int *jc);
```

和 `mxSetIr` 相似, 但是设置直接表示有非零元素列的向量。用 `helpdesk` 可得更多信息。

```
bool mxIsSparse(const mxArray *array_ptr);
```

如果 `array_ptr` 指向的 `mxArray` 是稀疏矩阵类型, 返回真。

下面的程序用来创建和处理字符串 `mxArrays`。

#### 命令集179 C中字符串的处理

`mxChar`

被字符串 `mxArray` 用来存储数据元素的数据类型。

```
mxArray *mxCreateCharArray(int ndim, const int *dims);
```

和 `mxCreateCellArray` 相似, 但是是创建  $n$  维的字符矩阵, 见命令集 181。

```
mxArray *mxCreateCharMatrixFromStrings(int m, char **str);
```

和 `mxCreateCellMatrix` 相似(见命令集 181), 但是是用 `str` 指向的字符串向量创建二维字符矩阵;  $m$  是字符串向量中的字符串数。

```
mxArray *mxCreateString(const char *str);
```

用字符串 `str` 创建一个字符串矩阵 `mxArray`。如果创建成功, 则返回指向这个字符串 `mxArray` 的指针; 否则返回 `NULL`。当字符串 `mxArray` 不再使用时, 应用 `mxDestroyArray` 来释放所占内存; 见命令集 75。

```
int mxGetString(const mxArray *array_ptr, char *buf, int buflen);
```

复制 `array_ptr` 指向的字符串 `mxArray`, 得到的字符串保存在 `buf` 中。 `buflen` 是 `buf` 中可以存放的最大字符数。如果复制成功, 返回 0; 否则返回 1。

```
bool mxIsChar(const mxArray *array_ptr);
```

如果 `array_ptr` 指向的 `mxArray` 是字符串类型, 则返回真。

MATLAB 5 中一个新数据类型是多维数组; 见 2.2 节。用下面的程序来处理这种类型的 `mxArray`。

注意：C中可以使用 8、16或32位的带符号或不带符号的整数和单精度浮点小数的 `mxArray` 来创建和计算。然而现在已不能在 MATLAB 环境中使用它们了。

#### 命令集180 C中多维数组的处理

```
mxArray *mxCreateNumericArray(int ndim, const int *dims, mxClassID class,
MxComplexity ComplexFlag);
```

和 `mxCreateCellArray` 相似，但是这里是创建  $n$  维的数字矩阵。数字类型为 `class`，见命令集 176 中的 `mxClassID`。如果有复数，则 `ComplexFlag` 设为 `mxCOMPLEX`；否则为 `mxREAL`。

```
void *mxGetData(const mxArray *array_ptr);
```

和 `mxGetPr` 相似，见命令集 177，但是返回一个 `void` 指针。更多的是用在除双精度浮点小数以外的其他类型数字矩阵中。

```
void mxSetData(mxArray *array_ptr, void *data_ptr);
```

和 `mxSetPr` 相似，见命令集 177，但是返回一个 `void` 指针。更多的是用在除双精度浮点小数以外的其他类型数字矩阵中。

```
void *mxGetImagData(const mxArray *array_ptr);
```

和 `mxGetPi` 相似，见命令集 177，但是返回一个 `void` 指针。更多的是用在除双精度浮点小数以外的其他类型数字矩阵中。

```
void mxSetImagData(mxArray *array_ptr, void *pi);
```

和 `mxSetPi` 相似，见命令集 177，但是返回一个 `void` 指针。更多的是用在除双精度浮点小数以外的其他类型数字矩阵中。

```
void mxSetLogical(mxArray *array_ptr);
```

在 `array_ptr` 指向的数字 `mxArray` 中设置逻辑标识符。MATLAB 就会把 `mxArray` 的数据当作逻辑变量来对待，也就是 0 是假，其他值是真。

```
void mxClearLogical(mxArray *array_ptr);
```

去掉数字 `mxArray` 中的逻辑标识符，见上。

```
bool mxIsLogical(const mxArray *array_ptr);
```

检查数字 `mxArray` 中的逻辑标识符的设置，见上。如果设置了，就返回真；否则返回假。

MATLAB 5 中一新数据类型是细胞矩阵，也称细胞数组；见 5.5 节。下面的程序用来处理这种类型的 `mxArray`：

#### 命令集181 C中细胞矩阵的处理

```
mxArray *mxCreateCellArray(int ndim, const int *dims);
```

创建一个  $n$  维的空细胞矩阵。参数 `ndim` 是维数，`dims` 是表示每维大小的向量指针。如果创建成功，就返回指向细胞矩阵的指针；否则返回 `NULL` 或停止程序运行。

```
mxArray *mxCreateCellMatrix(int m, int n);
```

和上个函数相似，但是是用来创建一个二维  $m \times n$  的细胞矩阵。

```
mxArray *mxGetCell(const mxArray *array_ptr, int index);
```

从细胞 `mxArray` 中复制一个细胞。参数 `array_ptr` 是指向细胞 `mxArray` 的指针，`index` 表



示第一个细胞与被复制细胞之间的细胞数；见命令集176中mxCalcSingleSubscript。如果复制成功，返回指向细胞mxArray的指针；否则返回NULL。

```
void mxSetCell(mxArray *array_ptr, int index, mxArray *value);
```

设置细胞mxArray中的一个细胞。参数 *index* 表示第一个细胞与被设置细胞之间的细胞数；见命令集176中mxCalcSingleSubscript。参数 *value* 是细胞指针，细胞的值将被设置在 *array\_ptr* 指向的mxArray中。

```
bool mxIsCell(const mxArray *array_ptr);
```

如果 *array\_ptr* 指向的mxArray是细胞类型，则返回真。

MATLAB 5 中另一新数据类型是结构；见 12.5 节。下面的程序用来处理这种类型的mxArray：

#### 命令集182 C中结构的处理

```
mxArray *mxCreateStructArray(int ndim, const int *dims, int nfields, const char **field_names);
```

和mxCreateCellArray相似(见命令集181)，但是是创建 *n* 维的结构矩阵。参数 *nfields* 表示每个元素中的域数，*field\_names* 是字符串向量指针，表示域名。

```
mxArray *mxCreateStructMatrix(int m, int n, int nfields, const char **field_names);
```

和上个函数相似，但是是用来创建二维  $m \times n$  的结构矩阵。

```
int mxGetNumberOfFields(const mxArray *array_ptr);
```

返回 *array\_ptr* 指向的结构mxArray中的域数。如果操作失败，则返回0。

```
mxArray *mxGetField(const mxArray *array_ptr, int index, const char *field_name);
```

返回 *array\_ptr* 指向的结构mxArray中一个元素的一个域值。参数 *index* 表示第一个元素和返回的元素之间的元素个数；见命令集176中mxCalcSingleSubscript。参数 *field\_name* 表示元素中域名的字符串，如果操作成功，返回指向这个域的指针；否则返回NULL。

```
void mxSetField(mxArray *array_ptr, int index, const char *field_name, mxArray *value);
```

和上个函数相似，但是是用指针 *value* 指向的值来设置域值。

```
int mxGetFieldNumber(const mxArray *array_ptr, const char *field_name);
```

返回 *array\_ptr* 指向的结构mxArray中一个域中的域数。字符串 *field\_name* 表示域名字，如果操作成功，则返回结构中域的个数(从0开始)；否则返回 -1。

```
const char *mxGetFieldNameByNumber(mxArray *array_ptr, int field_number);
```

返回 *array\_ptr* 指向的结构mxArray中的域名。参数 *field\_number* 表示结构中域的一个数字序列(从0开始)。

```
mxArray *mxGetFieldByNumber(const mxArray *array_ptr, int index, int field_number);
```



和mxGetField相似，但是返回的是域名，*field\_number*表示结构中域的一个数字序列(从0开始)。

```
void mxSetFieldByNumber(mxArray_ptr index, int field_number,
mxArray *value);
```

和mxSetField相似，但是返回的是域名，*field\_number*表示结构中域的一个数字序列(从0开始)。

```
bool mxIsStruct(const mxArray *array_ptr);
```

如果*array\_ptr*指向的mxArray是结构类型，则返回真。

```
int mxSetClassName(mxArray *array_ptr, const char *classname);
```

将*array\_ptr*指向的MATLAB结构转换成*classname*指定的MATLAB对象。如果转换成功，返回0；否则返回一个非零数。当用load读入对象到MATLAB中时，要检查类*classname*是否存在。如果不存在，则不能将对象反转换成结构。

下面的程序用来在C中取一些特殊常数值，比如机器无穷小正数和无穷大正数。还有一些程序用来检查变量的值是否等于这些常数。

### 命令集183 C中特殊常数

```
double mxGetEps(void);
```

返回MATLAB中机器无穷小正数。

```
double mxGetInf(void);
```

返回MATLAB中inf的值，也就是无穷大的正数。

```
bool mxIsInf(double value);
```

如果*value*是无穷大正数，就返回真；否则返回假。

```
double; mxGetNaN(void);
```

返回MATLAB中的NaN值。

```
bool mxIsNaN(double value);
```

如果*value*是一个NaN，返回真；否则返回假。

```
bool mxIsFinite(double value);
```

如果*value*不是一个inf或NaN，就返回真；否则返回假。

下面的程序用来调试C语言程序。

### 命令集184 C中调试程序

```
void mxAssert(int expr, char *error_message);
```

在调试时使用，如果*expr*为假，则程序停止，并打印出*expr*、文件名、行数和错误信息。如果*expr*为真，则对程序没有影响。

```
void mxAssertS(int expr, char *error_message);
```

同上，但是*expr*为假时不打印出*expr*。

MATLAB 4.2中的一些程序已被新程序所代替。虽然这些旧程序不应该再在新的C程序中

使用,但是它们还存在,以便 MATLAB 5能向下兼容。

#### 命令集185 C中旧的矩阵程序

```
mxCreateFull  mxIsFull  mxIsString  mxFreeMatrix
```

### 15.2.2 C中对MAT文件的处理

下面的例子说明了如何写和读一个 MAT文件,也就是 MATLAB以内部二进制格式存储的数据文件。

#### 例15.1

假设C程序中要用到一个服从正态分布的随机矩阵。这个简单的矩阵在 C程序中生成很困难,但是在 MATLAB中只需用一个命令。先定义一个  $10 \times 10$ 的服从正态分布的矩阵,并用 save命令保存:

```
OldMatrix=randn(10);           % 创建一个随机矩阵
save data OldMatrix;           % 保存这个矩阵到文件data.mat中
```

再编写一个C程序,用来读这个随机矩阵。矩阵的所有元素乘以 2,并生成一个新矩阵保存到文件data.mat中。C程序保存在文件matex.c中。

```
#include "mat.h"

void main()
{
    MATFile *mfp;
    mxArray *A_ptr, *B_ptr;
    double *A, *B;
    int M, N, i, j;

    /*从文件中读矩阵*/
    mfp = matOpen("data.mat", "u");
    A_ptr = matGetArray(mfp, "OldMatrix");
    M = mxGetM(A_ptr);
    N = mxGetN(A_ptr);
    A = mxGetPr(A_ptr);

    /*创建新矩阵*/
    B_ptr = mxCreateDoubleMatrix(M, N, mxREAL);
    mxSetName(B_ptr, "NewMatrix");
    B = mxGetPr(B_ptr);

    /*将原矩阵乘以2后保存到新矩阵中*/
    for (i = 0; i < M; i++)
    {
        for (j = 0; j < N; j++)
        {
            B[i + M * j] = 2 * A[i + M * j];
        }
    }
}
```

```
/*保存新矩阵到文件中, 程序结束*/  
    matPutArray(mfp, B_ptr);  
    matClose(mfp);  
    mxDestroyArray(A_ptr);  
    mxDestroyArray(B_ptr);  
    exit(0);  
}
```

这个程序在UNIX下进行编译(在系统提示符下输入连续的一行):

```
gcc -ansi -I/opt/matlab52/extern/include -o matex matex.c  
-L/opt/matlab52/extern/lib/sol2 -R/opt/matlab52/extern/lib/sol2  
-lmat -lmx -lmi -lut
```

注意, 所有的路径都和系统有关。当程序运行时, 将会在文件 data.mat中添加矩阵 NewMatrix, 它等于2\*OldMatrix。

下面列出了与MAT文件有关的程序, 用它们可以读和写 MATLAB的二进制文件, 也可以在MATLAB和C程序之间传递数据。为了使用这些程序, 必须在程序中嵌入头文件 mat.h, 也就是在程序开始包含下面一行:

```
#include "mat.h"
```

在读或写MAT文件之前, 必须先打开这个文件, 在完成之后关闭文件。用下列程序来完成打开和关闭文件。

#### 命令集186 C中打开和关闭MAT文件

MATFile

二进制MAT数据文件。

```
MATFile *matOpen(const char *filename, const char *mode);
```

以mode方式打开文件filename, 打开文件的方式有四种: “r”表示读方式, “w”表示写方式, “u”表示读/写方式, “w4”表示写MATLAB 4 的MAT文件。如果打开成功, 返回MAT文件的指针; 否则返回NULL。

```
FILE *matGetFp(MATFile *mfp);
```

返回mfp指向的MAT文件的C文件指针。比如在C的函数ferror()中将会用到。

```
int matClose(MATFile *mfp);
```

关闭mfp指向的MAT文件。如果关闭成功, 返回0; 否则返回EOF。

当MAT文件处于打开状态时, 就可用下面的程序来读和写文件。

#### 命令集187 C中读和写MAT文件

```
char **matGetDir(MATFile *mfp, int *num);
```

给出保存在 mfp指向的 MAT文件中mxArrays的名字列表指针。参数 num是保存mxArrays数的变量地址。如果操作失败, 返回 NULL, 并且num变成一个负数。当不再使用mxArrays列表时, 用mxFree释放所占内存; 见命令集175。

```
mxArray *matGetArray(MATFile *mfp, const char *name);
```

从mfp指向的MAT文件中复制name指定的mxArray。如果复制成功, 返回一个

mxArray指针；否则返回NULL。当不再使用时，用mxDestroyArray释放mxArray所占内存；见命令集175。

```
mxArray *matGetArrayHeader(MATFile *mfp, const char *name);
```

和上个函数相似，但是复制数组开始部分的信息。

```
mxArray *matGetNextArray(MATFile *mfp);
```

复制mfp指向的MAT文件中下个mxArray。如果复制成功，返回一个mxArray指针；否则返回NULL。当不再使用时，用mxDestroyArray释放它所占的内存；见命令集175。

```
mxArray *matGetNextArrayHeader(MATFile *mfp);
```

和上个函数相似，但是复制数组开始部分的信息。

```
int matPutArray(MATFile *mfp, const mxArray *mp);
```

将mp指向的mxArray写入到mfp指向的MAT文件中。如果文件中已有这个mxArray，那么就被覆盖。如果写操作成功，返回0；否则返回一个非零数。

```
int matPutArrayAsGlobal(MATFile *mfp, const mxArray *mp);
```

和上个函数相似，但是当mxArray读入到MATLAB中时，它被存放到全局工作区中。在局部工作区中也可以使用它。

```
int matDeleteArray(MATFile *mfp, const char *name);
```

从mfp指向的MAT文件中删除name指定的mxArray。如果删除成功，返回0；否则返回一个非零数。

MATLAB 4.2中的一些函数已被一些新函数所取代，旧函数的保留是为了使MATLAB 5能向下兼容，但是已不在C程序中使用。

#### 命令集188 C中与MAT文件操作有关的旧程序

matGetFull	matGetMatrix	matGetNextMatrix	matGetString
matPutFull	matPutMatrix	matPutString	matDeleteMatrix

使用下面的命令来编译和链接使用MAT文件的程序：

- UNIX：在系统提示符下输入一不间断行：

```
gcc -ansi -I/.../matlab/extern/include -o programname sourcecode.c
-L/.../matlab/extern/lib/... -R/.../matlab/extern/lib/...
-lmat -lmx -lmi -lut
```

上面路径中的三个点，...，表示这部分路径是系统安装路径，programname是用户调用的程序名，sourcecode.c是要进行编译的C源代码文件列表。这里使用的编译器是gcc，当然，其他的编译器也可以使用，只要它们按ANSI标准进行编译旧可以。如果需要，可以设置调试和优化标识；见编译器文档。

- Windows：在MATLAB提示符下输入：

```
mex sourcecode.c -f optfil
```

其中optfil表示批处理文件watengmatopts.bat(Watcom C)、msvcengmatopts.bat(Microsoft Visual C)或bccengmatopts.bat(Borland C)。参数sourcecode.c是要进行编译的C源代码文件列表。

• Macintosh : 见MATLAB 5手册《应用程序接口指南》。

### 15.2.3 C调用MATLAB

为了使C能调用MATLAB, 首先要打开一个MATLAB工程。通过调用命令 `engOpen` 就能很简单地打开一个工程; 见命令集 189。

下一步就是将 `mxArray` 转换成在MATLAB中可操作的形式。这可以分成两步来完成:

1) 第1步是将 `mxArray` 转换成MATLAB可理解的形式, 这又有两种不同的方式。一是用程序 `mxCreate` 来创建矩阵, 之后用 `mxSetName` 对它们进行命名。这些程序的描述在 15.2.1 节中。另一种方式是选择将一个自定义的数据结构复制到 `mxArray` 中。然而值得注意的一点是MATLAB在存储矩阵时是按列序来保存的, 而在C中是按行序来保存的, 所以必须分清下标。

2) 第2步是将矩阵放入MATLAB工作区中, 可以用以 `engPut` 开头的程序来完成; 见命令集 189。

现在MATLAB已准备好接收命令了。这些命令可以在普通命令窗口中给出, 但是是以字符串的形式传递给函数 `engEvalString`。

最后, 从MATLAB到C的转换和传递也是有必要的。

这听起来相当的复杂, 但是用下面的这个例子来说明就显得清楚多了。

#### 例15.2

假设C程序中有一个矩阵, 这个程序是有关计算机图形使用的。使用 MATLAB 就能很好地将图形显示出来。编写下面C程序并将它保存在文件 `plotm.c` 中:

```
# include "engine.h"

void main()
{
    Engine *ep;
    mxArray *A_ptr;
    double* A;
    int i, j;

    /*创建一个新矩阵*/
    A_ptr = mxCreateDoubleMatrix(10, 10, mxREAL);
    mxSetName(A_ptr, "A");
    A = mxGetPr(A_ptr);
    for (i = 0; i < 10; i++)
    {
        for (j = 0; j < 10; j++)
        {
            A[i + 10 * j] = (j + 1) * (j + 1) * (i + 1) * (i + 1);
        }
    }

    /*打开一个MATLAB工程*/
    ep = engOpen("/opt/matlab52/bin/matlab");
    /*传递新矩阵*/
    engPutArray(ep, A_ptr);
    /*画出图形并保存*/
}
```

```

engEvalString(ep, "mesh(A);");
engEvalString(ep, "print picture.eps -deps;");

/*结束*/
engClose(ep);
mxDestroyArray(A_ptr);
exit(0);
}

```

在UNIX环境下编译这个程序，可以在系统提示符下输入下面一不间断行命令：

```

gcc -ansi -I/opt/matlab52/extern/include -o plotm plotm.c
-L/opt/matlab52/extern/lib/sol2 -R/opt/matlab52/extern/lib/sol2
-leng -lmat -lmx -lmi -lut

```

注意，所有的路径都和系统有关，程序运行结果如图 15-1所示。

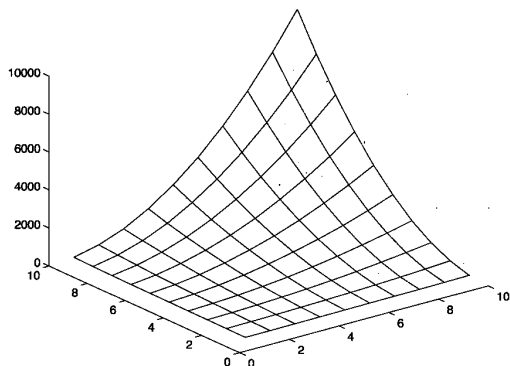


图15-1 将C程序创建的矩阵用MATLAB画出其图形

可以用下面的工程程序来处理 MATLAB的调用。注意，这些程序现在已不适用于 Macintosh系统。要使用这些程序，必须在程序开始包含头文件 **engine.h**，也就是在程序开始处包含下面一行：

```
#include "engine.h"
```

#### 命令集189 C中MATLAB工程程序

Engine

表示MATLAB工程。

```
Engine *engOpen(const char *startcmd);
```

打开一个 MATLAB工程，其中 startcmd是一个包含打开命令的字符串，通常是“matlab”。如果打开成功，返回指向 MATLAB工程的指针；否则返回 NULL。

```
int engOutputBuffer(Engine *ep, char *p, int n);
```

给 ep指向的 MATLAB工程定义一个大小为 n的文本缓冲区 p，通常屏幕上显示的文本保存在这里。

```
int engEvalString(Engine *ep, const char *string);
```

在 ep指向的 MATLAB工程中对字符串 string中的 MATLAB命令求值。如果求值成功，

返回0；否则返回一个非零数。

```
mxArray *engGetArray(Engine *ep, const char *name);
```

在 $ep$ 指向的MATLAB工程中从工作区复制 $name$ 表示的 $mxArray$ 。如果复制成功，返回指向 $mxArray$ 的指针；否则返回NULL。当 $mxArray$ 不再使用时，用 $mxDestroyArray$ 来释放它所占的内存；见命令集175。

```
int engPutArray(Engine *ep, const mxArray *mp);
```

将 $mp$ 指向的 $mxArray$ 复制到 $ep$ 指向的MATLAB工程的工作区中。如果复制成功，返回0；否则返回1。

```
int engClose(Engine *ep);
```

关闭 $ep$ 指向的MATLAB工程。如果关闭成功，返回0；否则返回1。

MATLAB 4.2中的一些函数已被新函数所取代，旧函数的保留是为了使MATLAB 5能向下兼容，但是已不在C程序中使用。

#### 命令集190 C中旧的MATLAB工程程序

用下面的命令来编译和链接调用MATLAB的C程序。

```
engGetFull      enGetMatrix      engPutFull      engPutMatrix
engSetEvalCallback  engSetEvalTimeout  engWinInit
```

• UNIX：在系统提示符下输入一不间断行：

```
gcc -ansi -I/.../matlab/extern/include -o programname
sourcecode.c
-L/.../matlab52/extern/lib/...
-R/.../matlab/extern/lib/...
-leng -lmat -lmx -lmi -lut
```

上面路径中的三个点，...，表示这部分路径是系统有关， $programname$ 是用户调用的程序名， $sourcecode.c$ 是要进行编译的C源代码文件列表。这里使用的编译器是gcc，当然，其他的编译器也可以使用，只要它们按ANSI标准进行编译就可以。如果需要，可以设置调试和优化标识；见编译器文档。

• Windows：在MATLAB提示符下输入：

```
mex sourcecode.c -f optfil
```

其中 $optfil$ 表示批处理文件 $watengmatopts.bat$ (Watcom C)、 $msvcengmatopts.bat$ (Microsoft Visual C)或 $bccengmatopts.bat$ (Borland C)。参数 $sourcecode.c$ 是要进行编译的C源代码文件列表。

### 15.2.4 MATLAB调用C

本节开始用一个简单的例子来说明MATLAB对C的调用。在MATLAB中调用C和调用普通的函数文件即M文件是一样的。



MEX文件中使用的程序和矩阵格式都和C中调用MATLAB是一样的。只不过是现在也是使用MEX文件。

文件必须有一个叫做mexFunction的主函数，mexFunction是MATLAB调用的函数。这个程序有四个参数：输入参量的个数、输出参量的个数和指向这些参量的两个指针数组。所以这个函数可以调用任何计算程序。

### 例15.3

创建一个MEX文件，将给出的矩阵中元素和它自己的行下标值相乘生成一个新矩阵。下面的C程序保存在文件rmult.c中：

```
# include "mex.h"

/* 完成元素和它自己的行下标相乘的程序*/
void radMult(double *Out, double *In, int M, int N)
{
    int i, j;

    for (i = 0; i < M; i++)
    {
        for (j = 0; j < N; j++)
        {
            Out[i + M * j] = (i + 1) * In[i + M * j];
        }
    }
}

/* MATLAB调用的程序*/
void mexFunction(int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[])
{
    mxArray *In_ptr, *Out_ptr;
    double *In, *Out;
    int M, N;

    /* 检查参量个数及其类型*/
    In_ptr = prhs[0];
    In = mxGetPr(In_ptr);
    if (nrhs != 1)
        mexErrMsgTxt("Only one input argument allowed!");
    else if (nlhs != 1)
        mexErrMsgTxt("Only one output argument allowed!");
    if ( !mxIsNumeric(In_ptr) || mxIsComplex(In_ptr) ||
         mxIsSparse(In_ptr) || !mxIsDouble(In_ptr) )
        mexErrMsgTxt
            ("Input argument must be a full floating point matrix!");

    /* 创建一个新矩阵*/
    M = mxGetM(In_ptr);
    N = mxGetN(In_ptr);
```

```
Out_ptr = mxCreateDoubleMatrix(M, N, mxREAL);
Out = mxGetPr(Out_ptr);

/*调用矩阵运算程序*/
radMult(Out, In, M, N);

/*返回新矩阵...*/
plhs[0] = Out_ptr;
}
```

在UNIX下编译这个程序(在系统提示符是输入):

```
/opt/matlab52/bin/mex rmult.c
```

注意,路径和系统有关。

当程序编译完后,就可以在MATLAB提示符下输入下面内容来运行程序:

```
New =
     1     2     3
     2     4     6
     3     6     9
```

在下面的命令表中给出了所有 MEX 程序。它们可以在 MEX 文件中使用,也可以看作是用 C 编写的并已编译过的 M 文件。调用这些程序,必须在程序开始嵌入头文件 `mex.h`,也就是在程序开始处包含下面一行:

```
#include "mex.h"
```

为了使 MATLAB 能调用 C 编写的程序,必须编写一个接口函数来调用 `mxFunction`。从 MEX 文件中也可以有其他的调用,也就是用程序 `mexCallMATLAB` 和 `mexEvalString` 来实现从 C 到 MATLAB 的调用。

### 命令集 191 C 与 MATLAB 的接口

```
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]);
```

是 MATLAB 接口函数,参数 `nlhs` 表示输出参量的个数, `plhs` 表示指向这些参量的指针向量的指针(设置为 NULL)。同样, `nrhs` 表示输入参量的个数, `prhs` 表示指向这些参量的指针向量的指针。输入参量不能在 C 程序中改变。

```
int mexCallMATLAB(int nlhs, mxArray *plhs[], int nrhs, mxArray *prhs[],
const char *command_name);
```

调用一个 MATLAB 函数、M 文件或 MEX 文件。参数 `nlhs` 是输出参数个数(必须不大于 50), `plhs` 是这些参数指针向量的指针(设置为 NULL)。同样, `nrhs` 是输入参数个数(必须不大于 50), `prhs` 是这些参数指针向量的指针。字符串 `command_name` 表示调用函数的名字,如果它是一个运算符,就用一个带单引号的字符,如 `' + '`。如果操作成功,返回 0;否则返回一个非零数。当不再使用时,用 `mxDestroyArray` 来释放 `plhs` 指向的 `mxArray` 所占的内存;见命令集 175。

```
void mexSetTrapFlag(int trap_flag);
```

用来处理调用 `mexCallMATLAB` 时发生的错误。如果 MATLAB 在调用 `mexCallMATLAB`

时发生错误，就停止运行MEX文件，并回到MATLAB提示符下。

将`trap_flag`设为0表示发生同样的错误；相反将它设为1，MEX文件恢复控制。

```
int mexEvalString(const char *command);
```

在调用工作区执行MATLAB命令`command`，不将结果传送回MEX程序。`command`的所有输入参数都可以在调用工作区中找到。如果操作成功，返回0；否则返回1。

```
int mexAtExit(void (*ExitFcn)(void));
```

定义一个在MEX程序退出时调用的函数。比如用来关闭文件，这个函数总是返回0。

另外在`mexFunction`中可以将数据作为参数来传递，这样就可以在MATLAB工作区中直接读和写数据。

#### 命令集192 C和MATLAB数据交换

```
mxArray *mexGetArray(const char *name, const char *workspace);
```

从工作区复制一个`mxArray`，字符串`name`是`mxArray`的名字，`workspace`表示要复制的`mxArray`所在工作区，有下列字符串可用：

“base”表示在MATLAB当前工作区内搜索变量`name`。

“caller”表示在调用函数的工作区内搜索变量`name`。

“global”表示在全局变量列表中搜索变量`name`。

如果复制成功，返回复制的`mxArray`的指针；否则返回NULL。当它不再使用时，用`mxDestroyArray`释放所占内存；见命令集175。

```
int mexPutArray(mxArray *array_ptr, const char *workspace);
```

和上个函数相似，但是是将`mxArray`复制到工作区中。如果复制成功，返回0；否则返回1。

```
const mxArray *mexGetArrayPtr(const char *name, const char *workspace);
```

返回一个不同工作区中`mxArray`的只读指针，字符串`name`是`mxArray`的名字，`workspace`是要复制`mxArray`(见`mexGetArray`)的工作区。如果操作失败，返回NULL。注意，这只能从`mxArray`中读它的值，而不能改变。

下面的程序用来在MATLAB命令窗口中输出错误信息、警告和其他文本。

#### 命令集193 C中的错误处理和打印

```
int mexPrintf(const char *formatg1, arg2,...);
```

在MATLAB窗口中以ANSI C输出格式输出字符串。参数`format`是ANSI C格式字符串，`arg1`，`arg2`，...是输出的可选参数。

```
void mexErrMsgTxt(const char *error_msg);
```

在MATLAB窗口中输出错误信息`error_msg`，并且停止运行MEX文件。

```
void mexWarnMsgTxt(const char *warning_msg);
```

在MATLAB窗口中显示警告信息`warning_msg`，但是不停止运行MEX文件。

在MATLAB环境中，下列C程序可以完成不同的工作。例如，可以用它们来处理图形对

象；见14.2节。

#### 命令集194 C中其他MEX程序

```
bool mexIsGlobal(const mxArray *array_ptr);
```

如果`array_ptr`指向的`mxArray`是全局变量，则返回真。

```
const mxArray *mexGet(double handle, const char *property);
```

用来获取MATLAB中图形对象的属性值。参数`handle`是图形对象的指针(句柄)，字符串`property`表示属性。如果操作成功，返回一个包含属性值的 `mxArray`的指针，否则返回NULL。也可参见14.2节。

```
int mexSet(double handle, const char *property, mxArray *value);
```

和上个函数相似，但是是用来设置 MATLAB中图形对象的属性。参数`value`是包含要设置的属性值的`mxArray`指针。如果设置成功，返回0；否则返回1。也可参见14.2节。

```
void mexAddFlops(int count);
```

将`count`加到MATLAB内部计算器上，用来计算浮点操作的次数。

下列的程序用在MEX文件中来管理内存。这些程序可以使变量在 MEX程序的连续调用之间保存变量的值。

#### 命令集195 C中内存处理

```
void mexMakeArrayPersistent(mxArray *array_ptr);
```

通常，在MEX程序退出时要对 `mxArray`所占的内存进行释放。如果不释放所占的内存，也就是要在随后的 MEX程序调用中保存 `mxArray`的值，那么这个函数还可以使用。参数`array_ptr`表示`mxArray`指针。

```
void mexMakeMemoryPersistent(void *ptr);
```

和上个函数相似，但是是管理用 `mxCalloc`分配的内存。参数`ptr`是指向分配内存区开始部分的指针。

```
void mexLock(void);
```

锁住MEX文件，也就是不能从内存中将它删除。

```
void mexUnlock(void);
```

对MEX文件解锁；见上。

```
bool mexIsLocked(void);
```

如果MEX文件被锁，返回真；见上。

MATLAB 4.2中的一些函数已被一些新函数所取代，旧函数的保留是为了使 MATLAB 5能向下兼容，但是已不在C程序中使用。

#### 命令集196 C中旧的MEX程序

<code>mexGetMatrix</code>	<code>mexGetGlobal</code>	<code>mexPutMatrix</code>	<code>mxGetMatrixPtr</code>
<code>mexGetFull</code>	<code>mexGetGlobal</code>	<code>mexPutMatrix</code>	<code>mxGetMatrixPtr</code>
<code>mexIsInf</code>	<code>mexGetNaN</code>	<code>mexIsNaN</code>	<code>mexIsFinite</code>

使用下面的命令来对调用 MATLAB 的 C 程序进行编译和链接：

- **UNIX**：在系统提示符下输入一不间断行：

```
.../matlab52/bin/mex sourcecode.c
```

上面路径中的三个点，...，表示这部分路径与系统有关，programname 是用户调用的程序名，sourcecode.c 是要进行编译的 C 源代码文件列表。可能的话，用户要配置编译或链接环境；见 MATLAB 5 《应用程序接口指南》。

- **Windows**：在 MATLAB 提示符下输入：

```
mex sourcecode.c
```

可能的话，用户要配置编译或链接环境；见 MATLAB 5 《应用程序接口指南》。

- **Macintosh**：参考 MATLAB 5 《应用程序接口指南》来进行配置。配置完后可以在 MATLAB 提示符下输入：

```
Mex sourcecode.c
```

在不同的系统下创建出的 MEX 文件的后缀名也不一样。可以用 MATLAB 命令 mexext 来检查系统使用的后缀名类型。

可以对 MEX 程序进行调试，但是添加一个编译调试符—g。在 UNIX 环境下，编译完后，当 MATLAB 已运行，就可在系统提示符下输入—Ddbx 来运行程序。在 MATLAB 提示符下输入 dbmex on 来运行要调试的 MEX 文件。然而，在 MEX 文件运行前，返回到调试器中来列程序、设置断点等。从 MATLAB 提示符下可以输入 dbmex stop 到调试器中。在调试器中可以输入 continue 返回到 MATLAB 中。在 Windows 和 Macintosh 中调试可以参考 MATLAB 5 手册《应用程序接口指南》。

## 15.3 MATLAB和FORTRAN

FORTRAN 和 C 的不同之处在于 MATLAB FORTRAN 库只能用来处理字符串和二维矩阵、满矩阵和稀疏矩阵，而且矩阵的元素要求是双精度浮点小数。

### 15.3.1 FORTRAN中对mxArray的操作

可以用下面表中列出的程序来对 mxArray 进行操作。这里的 ‘mxArray’ 可以是字符串或二维矩阵、满矩阵和稀疏矩阵，而且要求矩阵元素是双精度浮点小数。

C 中的许多操作都通过指针来完成。FORTRAN 中没有指针，所以操作起来很不方便。在 FORTRAN 中所有的平台上都用的是 MATLAB 的 4 字节的整数指针，除了在 Alpha 和 SGI64 中用的是 8 字节的整数指针（因此对表中的命令调用必须根据不同的平台进行相应的调整）。矩阵的所有运算都必须通过以 mxCopy 开头的转换程序来进行。

有一些编译器也支持 %val 结构，这是对 FORTRAN 77 和 FORTRAN 90 的一个扩充。也就是可以在调用的子程序间传递变量值，也称为 ‘值调用’。如果这种结构可用，就不必将指针转换成数据，也就是可以用 mxGetPr 和 mxGetPi 来返回指针；见命令集 200。将 %val 代替这些指针传递到子程序中，在子程序中将这些指针声明为包含双精度浮点小数的 FORTRAN 矩阵；见下例 15.6。参考 FORTRAN 编译手册，可知它是否支持 %val 结构。

如果要动态分配内存也要用到 %val 结构；见 MATLAB 5 手册《应用程序接口指南》和下例 15.6。

## 命令集197 FORTRAN中指针处理

```
subroutine mxCopyPtrToInteger4(px, y, n)
integer*4 y(n)
integer*4 px, n
```

从 $px$ 指向的MATLAB整数向量中复制 $n$ 个整数到FORTRAN中标准整数向量 $y$ 中。参数 $px$ 可以是 $ir$ 或 $jc$ 向量的指针；见命令集201中 $mxGetIr$ 和 $mxGetJc$ 。

```
subroutine mxCopyInteger4ToPtr(y, px, n)
integer*4 y(n)
integer*4 px, n
```

从FORTRAN普通整数向量 $y$ 中复制 $n$ 个整数到MATLAB  $px$ 指向的整数向量中。参数 $px$ 可以是 $ir$ 或 $jc$ 向量的指针；见命令集201中 $mxGetIr$ 和 $mxGetJc$ 。

```
subroutine mxCopyPtrToPtrArray(px, y, n)
integer*4 y(n)
integer*4 px, n
```

复制指针 $px$ 到FORTRAN的普通整数向量 $y$ 中，参数 $px$ 可以是 $ir$ 或 $jc$ 向量的指针；见命令集201中 $mxGetIr$ 和 $mxGetJc$ 。

```
subroutine mxCopyPtrToReal8(px, y, n)
real*8 y(n)
integer*4 px, n
```

从MATLAB中的 $px$ 指向的浮点小数向量中将 $n$ 个浮点小数复制到FORTRAN中的普通浮点小数向量 $y$ 中，参数 $px$ 可以是 $pr$ 或 $pi$ 向量的指针；见命令集200中 $mxGetPr$ 和 $mxGetPi$ 。

```
subroutine mxCopyReal8ToPtr(y, px, n)
real*8 y(n)
integer*4 px, n
```

从FORTRAN的普通浮点小数向量 $y$ 中将 $n$ 个浮点小数复制到MATLAB的 $px$ 指向的浮点小数向量中，参数 $px$ 可以是 $pr$ 或 $pi$ 向量的指针；见命令集200中 $mxGetPr$ 和 $mxGetPi$ 。

```
subroutine mxCopyPtrToComplex16(pr, pi, y, n)
complex*16 y(n)
integer*4 pr, pi, n
```

将 $pr$ (实数部分)和 $pi$ (虚数部分)指向的MATLAB向量中的 $n$ 个复数复制到FORTRAN的普通复数向量 $y$ 中，见命令集200中 $mxGetPr$ 和 $mxGetPi$ 。

```
subroutine mxCopyComplex16ToPtr(y, pr, pi, n)
complex*16 y(n)
integer*4 pr, pi, n
```

将FORTRAN的普通复数向量 $y$ 中的 $n$ 个复数复制到 $pr$ (实数部分)和 $pi$ (虚数部分)指向的MATLAB向量中，见命令集200中 $mxGetPr$ 和 $mxGetPi$ 。

```
subroutine mxCopyPtrToCharacter(px, y, n)
character*(*) y
integer*4 px, n
```

将 $px$ 指向的MATLAB字符向量中的 $n$ 个字符复制到FORTRAN的普通字符向量 $y$ 中，见命令集202中 $mxGetString$ 。

```
subroutine mxCopyCharacterToPtr(y, px, n)
character*(*) y
```

```
integer*4 px, n
```

将FORTRAN的普通字符向量 $y$ 中的 $n$ 个字符复制到 $px$ 指向的MATLAB字符向量中，见命令集202中mxGetString。

下面表中的程序用来分配和释放内存。及时地释放不再使用的内存是一个好的编程经验。如果只用MATLAB程序来创建数据结构，就不必释放内存，因为在程序结束时 MATLAB会自动完成。为了以防万一，不管哪种情况都可用这些程序来释放内存。

#### 命令集198 FORTRAN内存管理

```
integer*4 function mxCalloc(n, size)
integer*4 n, size
```

分配内存，参数 $n$ 是要分配的元素个数， $size$ 是每个元素的字节数。如果分配成功，返回指向分配内存的起始位置的指针；否则返回0或程序停止。当元素不再使用时，用mxFree来释放分配的内存。

```
subroutine mxFree(ptr)
integer*4 ptr
```

释放 $ptr$ 指向的内存。

```
subroutine mxFreeMatrix(pm)
integer*4 pm
```

使用mxCreateFull或mxCreateSparse分配的内存。参数 $pm$ 是一个mxArray的指针。

#### 命令集199 FORTRAN中处理mxArray的常用程序

```
character*32 function mxGetName(pm)
integer*4 pm
```

返回一个字符向量(最大32个字符)，包含 $pm$ 指向的mxArray名字。如果操作失败，返回0。

```
subroutine mxSetName(pm, name)
integer*4 pm
character*(32) name
```

将 $pm$ 指向的mxArray赋予名字 $name$ (最大32个字符)。

```
real*8 function mxGetScalar(pm)
integer*4 pm
```

返回 $pm$ 指向的mxArray的第一个实数元素值。如果mxArray是稀疏矩阵，则返回它的第一个非零元素值；如果是空矩阵，则返回一个不确定数。

```
integer*4 function mxGetM(pm)
integer*4 pm
```

返回 $pm$ 指向的mxArray中行数。

```
subroutine mxSetM(pm, m)
integer*4 pm, m
```

用来对 $pm$ 指向的mxArray重构，增加/减少mxArray中的行数，参数 $m$ 是要求的行数。如果增加/减少mxArray中的行数，则必须分配/释放内存；见helpdesk可得更多信息。

```
integer*4 function mxGetN(pm)
```



```
integer*4 pm
```

返回 $pm$ 指向的mxArray的列数。

```
subroutine mxSetN(pm,n)
```

```
integer*4 pm,n
```

用来对 $pm$ 指向的mxArray重构，增加/减少mxArray中的列数，参数 $n$ 是要求的列数。

如果增加/减少mxArray中的列数，则必须分配/释放内存；见helpdesk可得更多信息。

```
integer*4 function mxIsNumeric(pm)
```

```
integer*4 pm
```

如果 $pm$ 指向的mxArray包含数字，则返回1，否则返回0。

```
integer*4 function mxIsDouble(pm)
```

```
integer*4 pm
```

如果 $pm$ 指向的mxArray包含有双精度浮点小数，则返回1；否则返回0。如果返回0，则这个mxArray不能被FORTRAN程序使用。

```
integer*4 function mxIsComplex(pm)
```

```
integer*4 pm
```

如果 $pm$ 指向的mxArray包含有复数，则返回1；否则返回0。

下面的程序用来创建和处理二维 $m \times n$ 的满矩阵，矩阵的元素是双精度浮点小数。

## 命令集200 FORTRAN中满矩阵的处理

```
integer*4 function mxCreateFull(m, n, ComplexFlag)
```

```
integer*4 m, n, ComplexFlag
```

用双精度浮点小数创建一个二维 $m \times n$ 矩阵(mxArray)。如果矩阵中有复数，则参数ComplexFlag设为1；否则设为0。当矩阵不再使用时，用mxFreeMatrix来释放矩阵所占的内存空间；见命令集198。

```
integer*4 function mxGetPr(pm)
```

```
integer*4 pm
```

返回 $pm$ 指向的mxArray中指向第一个实数元素的指针。如果mxArray中没有实数元素，则返回0。

```
subroutine mxSetPr(pm, pr)
```

```
integer*4 pm, pr
```

在 $pm$ 指向的满矩阵mxArray中设置实数元素。参数 $pr$ 是包含实数值的向量指针，这个向量必须用mxCalloc来动态分配；见命令集198。

```
integer*4 function mxGetPi(pm)
```

```
integer*4 pm
```

和mxGetPr相似，但是是用于虚数元素。

```
subroutine mxSetPi(pm, pi)
```

```
integer*4 pm, pi
```

和mxSetPr相似，但是是用于虚数元素。

```
integer*4 function mxIsFull(pm)
```

```
integer*4 pm
```

$pm$ 指向的mxArray是一个满矩阵，则返回1；如果是稀疏矩阵，则返回0。

下面的程序用来创建和处理二维  $m \times n$  的稀疏矩阵，矩阵的元素是双精度浮点小数。

#### 命令集201 FORTRAN中稀疏矩阵的处理

```
integer*4 function mxCreateSparse(m, n, nzmax, ComplexFlag)
integer*4 m,n,nzmax, ComplexFlag
```

创建一个二维  $m \times n$  的稀疏矩阵，参数  $nzmax$  表示矩阵中非零元素的个数。如果矩阵中有复数元素，则参数  $ComplexFlag$  设为1，否则设为0。如果创建成功，返回指向矩阵的指针；否则返回0。

```
integer*4 function mxGetNzmax(pm)
integer*4 pm
```

返回  $pm$  指向的稀疏矩阵 mxArray 中非零元素的个数  $nzmax$  (见上)。如果发生错误，返回一个不确定数。

```
subroutine mxSetNzmax(pm, nzmax)
integer*4 pm, nzmax
```

设置  $pm$  指向的稀疏矩阵 mxArray 中非零元素的个数  $nzmax$  (见上)。如果改变  $nzmax$  值，并且如果存在向量 **ir**、**pr** 和 **pi**，则它们也会发生改变。使用 helpdesk 可得更多信息。

```
integer*4 function mxGetIr(pm)
integer*4 pm
```

返回一个整数向量指针，向量中包含  $pm$  指向的稀疏矩阵中有非零元素的行数，其中第一行有数字0。如果操作失败，返回0。

```
subroutine mxSetIr(pm, ir)
integer*4 pm, ir
```

定义  $pm$  指向的稀疏矩阵中有非零元素的行数，参数  $ir$  是整数向量指针。向量中包含要使用的行数，这些行必须按列序来保存。行从0开始。使用 helpdesk 可得更多信息。

```
integer*4 function mxGetJc(pm)
integer*4 pm
```

和  $mxGetIr$  相似，但是返回的是直接表示有非零元素的列的整数向量指针。使用 helpdesk 可得更多信息。

```
subroutine mxSetJc(pm, jc)
integer*4 pm, jc
```

和  $mxSetIr$  相似，但是是用来设置有非零元素的列。使用 helpdesk 可得更多信息。

```
integer*4 function mxIsSparse(pm)
integer*4 pm
```

如果  $pm$  指向的矩阵是稀疏矩阵，则返回1；否则返回0。

下面的程序用来创建和处理字符串矩阵 mxArray。

#### 命令集202 FORTRAN中字符串的处理

```
integer*4 function mxCreateString(str)
character*(*) str
```

从字符串 **str** 创建一个字符串矩阵 **mxArray**。如果创建成功，则返回这个字符串 **mxArray** 的指针；否则返回 0。

```
integer*4 function mxGetString(pm, str, strlen)
integer*4 pm, strlen
character*(*) str
```

从 **pm** 指向的字符串 **mxArray** 中复制一个字符串。字符串被保存在 FORTRAN 的字符串向量 **str** 中，**strlen** 是 **str** 中可以保存的最大字符个数。如果复制成功，返回指向字符串 **mxArray** 的指针；否则返回 0。

```
integer*4 function mxIsString(pm)
integer*4 pm
```

如果 **pm** 指向的 **mxArray** 是字符串矩阵，则返回 1；否则返回 0。

### 15.3.2 FORTRAN 中 MAT 文件的处理

下面的例子给出了如何来读和写 MAT 文件，也就是 MATLAB 以二进制格式存储的数据文件。

#### 例 15.4

假设 FORTRAN 程序中要用到一个服从正态分布的随机矩阵。在程序中来生成相当的困难，但是在 MATLAB 中只需一个命令就可以创建出来。先定义一个  $10 \times 10$  的正态分布的矩阵，并用 **save** 命令保存：

```
OldMatrix=randn(10);      % 创建一个随机矩阵
Save data OldMatrix      % 将矩阵OldMatrix保存到文件data.mat中
```

现在编写 FORTRAN 程序来读取这个随机矩阵且所有的元素乘以 2，并将结果作为一个新矩阵保存到文件 **data.mat** 中。FORTRAN 程序保存在文件 **matex.for** 中。

```
program main
implicit none

integer mfp
integer A_ptr, B_ptr
double precision Temp(10, 10)
integer A, B
integer i, j
integer matOpen, matGetMatrix, mxCreateFull
integer mxGetPr, matPutMatrix, stat, matClose, stat
```

```
C    Read matrix from file.
mfp = matOpen("data.mat", "u")
A_ptr = matGetMatrix(mfp, "OldMatrix")
A = mxGetPr(A_ptr)
call mxCopyPtrToReal8(A, Temp, 100)
```

```
C    Create a new matrix.
B_ptr = mxCreateFull(10, 10, 0)
call mxSetName(B_ptr, "NewMatrix")
B = mxGetPr(B_ptr)
```

```

C      Set the new matrix to the old*2.
      do 100 i = 1, 10
        do 110 j = 1, 10
          Temp(i, j) = 2.0 * Temp(i, j)
110      continue
100     continue
      call mxCopyReal8ToPtr(Temp, B, 100)

C      Save the new matrix and finish.
      stat = matPutMatrix(mfp, B_ptr)
      stat = matClose(mfp)
      call mxFreeMatrix(A_ptr)
      call mxFreeMatrix(B_ptr)
      stop

      end

```

UNIX下编译这个程序(在系统提示符下输入—不间断行)：

```

f77 -I/opt/matlab52/extern/include -o matex matex.for
-L/opt/matlab52/extern/lib/sol2 -R/opt/matlab52/extern/lib/sol2
-lmat -lmx -lmi -lut

```

注意，所有的路径都和系统有关。程序运行后，在文件 data.mat中增加一个矩阵 NewMatrix，它等于2\*OldMatrix。

下面列出对MAT文件操作的程序，使用这些程序可以来读和写 MATLAB二进制文件，并且可以在MATLAB和FORTRAN程序之间传递数据。注意，这些程序不能用在Windows环境下，不过它们仍能用在C程序中。

#### 命令集203 FORTRAN中打开和关闭MAT文件

```

integer*4 function matOpen(filename, mode)
integer*4 mfp
character*(*) filename, mode

```

以mode方式打开文件filename，打开的方式有：“r”读方式、“w”写方式、“u”读/写方式、“w4”以MATLAB 4 MAT文件格式写。如果成功打开文件，返回MAT文件的指针mfp，否则返回0。

```

integer*4 function matClose(mfp)
integer*4 mfp

```

关闭mfp指向的MAT文件。如果成功关闭，返回0；否则返回1。

当MAT文件处于打开状态时，可以用下面的程序来写或读文件。

#### 命令集204 FORTRAN中MAT文件的读和写

```

integer*4 function matGetDir(mfp, num)
integer*4 mfp, num

```

给出一个向量指针，向量包含 *mfp* 指向的 MAT 文件中保存的 mxArray 的名字。参数 *num* 是存有 mxArray 个数的变量。如果操作失败，返回 0，并且 *num* 变成一个负数。当向量不再使用时 *mxFree* 来释放所占内存空间；见命令集 198。

```
integer*4 function matGetMatrix(mfp, name)
integer*4 mfp
character*(*) name
```

从 *mfp* 指向的 MAT 文件中复制 **name** 表示的 mxArray。如果复制成功，返回指向 mxArray 的指针；否则返回 0。用 *mxFreeMatrix* 来释放 mxArray 所占的内存；见命令集 198。

```
integer*4 function matGetNextMatrix(mfp)
integer*4 mfp
```

从 *mfp* 指向的 MAT 文件中复制下一个 mxArray。如果复制成功，返回指向 mxArray 的指针；否则返回 0。用 *mxFreeMatrix* 来释放 mxArray 所占的内存；见命令集 198。

```
integer*4 function matPutMatrix(mfp, name)
integer*4 mp, mfp
character*(*) name
```

将名为 **name** 的 mxArray 写到 *mfp* 指向的 MAT 文件中。如果 mxArray 已存在于文件中，则覆盖原来的 mxArray。如果写成功，返回 0；否则返回一个非零数。

```
integer*4 function matGetFull(mfp, name, m, n, pr, pi)
integer*4 mfp, m, n, pr, pi
character*(*) name
```

从 *mfp* 指向的 MAT 文件中复制名为 **name** 的  $m \times n$  满矩阵，矩阵的元素是双精度的浮点小数。参数 *pr* 是矩阵实数部指针，*pi* 是矩阵的虚数部指针。参数 *m*、*n*、*pr* 和 *pi* 由函数来设置。如果复制成功，返回 0；否则返回 1。当矩阵不再使用时，用 *mxFree* 来释放矩阵的实数部和虚数部所占的内存；见命令集 198。

```
integer*4 function matPutFull(mfp, name, m, n, pr, pi)
integer*4 mfp, m, n, pr, pi
character*(*) name
```

和上个函数相似，但是是将名为 **name** 的  $m \times n$  满矩阵写入到 MAT 文件中，矩阵的元素是双精度浮点小数。如果文件中已存在这个矩阵，就覆盖原来的矩阵。

```
integer*4 function matGetString(mfp, name, str, strlen)
integer*4 mfp, strlen
character*(*) name, str
```

从 *mfp* 指向的 MAT 文件中复制最长 *strlen* 的字符串 **str**。如果复制成功，返回 0；返回 1 表示 **str** 不是字符串；返回 2 表示 **str** 的长度大于 *strlen*；返回 3 表示发生文件错误。

```
integer*4 function matPutString(mfp, name, str)
integer*4 mfp
character*(*) name, str
```

将字符串 **str** 作为字符串矩阵 **name** 写入到 MAT 文件中。如果操作成功，返回 0；否则返回 1。

```
subroution matDeleteMatrix(mfp, name)
integer*4 mfp
```

```
character*(*) name
```

从`mfp`指向的MAT文件中删除名为的 `name`的mxArray。如果删除成功，返回0；否则返回一个非零数。

用下面的命令来编译和链接使用MAT文件的FORTRAN程序：

- UNIX：在系统提示符下输入一不间断行：

```
f77 -I/.../matlab/extern/include -o programname sourcecode.for  
-L/.../matlab/extern/lib/... -R/.../matlab/extern/lib/...  
-lmat -lmx -lmi -lut
```

上面路径中的三个点，...，表示这部分路径与系统有关，`programname`是用户调用的程序名，`sourcecode.for`是要进行编译的FORTRAN源代码文件列表。如果需要，可以设置调试和优化标识；见编译器文档。

- Macintosh：见MATLAB 5手册《应用程序接口指南》。

### 15.3.3 FORTRAN调用MATLAB

FORTRAN调用MATLAB，必须先启动一个MATLAB工程。这很简单，只需调用`engOpen`就可以完成；见命令集205。

下一步就是将mxArray转换成在MATLAB中可操作的形式。这可以分成两步来完成：

1) 第1步是将mxArray转换成MATLAB可理解的形式。用程`mxCreate`开头的程序来创建一个和要传递的数据类型大小相同的矩阵 `mxArray`。这些程序的描述在 15.3.1中。将FORTRAN格式的数据类型复制到MATLAB格式，在C中不必这样。用`mxCopy`开头的程序来完成这些操作；见命令集197。

2) 第2步是将矩阵放入MATLAB工作区中，可以用程序`engPutMatrix`或`engEvalString`来完成；见命令集205。

现在MATLAB已准备好接收命令了。这些命令可以在普通命令窗口中给出，但是是以字符串的形式传递给函数`engEvalString`。

此后保留其他的完成变换的过程。

这听起来相当复杂，但是用下面的例子来说明就显得清楚多了。

#### 例15.5

假设FORTRAN程序中有一个矩阵，这个程序是有关计算机图形使用的。使用MATLAB能很好地将图形显示出来。编写下面的FORTRAN程序并将它保存在文件`plotm.for`中：

```
program main  
implicit none  
  
integer ep  
integer A_ptr  
integer A  
integer i, j  
double precision Temp(10, 10)  
integer mxCreateFull, mxGetPr, engOpen
```

```

integer engPutMatrix, engEvalString, engClose, stat

C      Create a new matrix.
      A_ptr = mxCreateFull(10, 10, 0)
      call mxSetName(A_ptr, "A")
      A = mxGetPr(A_ptr)
      do 100 i = 1, 10
        do 110 j = 1, 10
          Temp(i, j) = j * j * i * i
110      continue
100     continue
      call mxCopyReal8ToPtr(Temp, A, 100)

C      Start the MATLAB Engine.
      ep = engOpen("/opt/matlab52/bin/matlab")

C      Transfer the new matrix.
      stat = engPutMatrix(ep, A_ptr)

C      Perform the command mesh(A) and save.
      stat = engEvalString(ep, "mesh(A);")
      stat = engEvalString(ep, "print picture.eps -deps;")

C      Finish.
      stat = engClose(ep)
      call mxFreeMatrix(A_ptr)
      stop
      end

```

在UNIX环境下编译这个程序，可以在系统提示符下输入下面一不间断行命令：

```

f77 -I/opt/matlab52/extern/include -o plotm plotm.for
-L/opt/matlab52/extern/lib/sol2 -R/opt/matlab52/extern/lib/sol2
-leng -lmat -lmx -lmi -lut

```

注意，所有的路径都和系统有关。程序运行的结果如图 15-1所示。

可以用下面的工程程序来处理 MATLAB 的调用。注意，这些程序现在已不适用于 Windows 或 Macintosh 系统，不过在 C 中它们仍可用。

#### 命令集 205 FORTRAN 中 MATLAB 工程程序

```

integer*4 function engOpen(startcmd)
integer*4 ep
character*(*) startcmd

```

启动一个 MATLAB 工程，字符串 **startcmd** 是启动命令，通常是 “matlab”。如果成功，返回指向 MATLAB 工程的指针 **ep**；否则返回 0

```

subroutine engOutputBuffer(ep, p, n)
integer*4 ep, n
character*(*) p

```

给 **ep** 指向的 MATLAB 工程定义一个大小为 **p** 的文本缓冲区。通常将屏幕上显示的文件保存到这里。



```
integer*4 function engEvalString(ep, command)
integer*4 ep
character*(*) command
```

在 $ep$ 指向的MATLAB工程中对字符串 **command** 中的MATLAB命令求值。如果求值成功，返回0；否则返回一个非零数。

```
integer*4 function engGetMatrix(ep, name)
integer*4 ep
character*(*) name
```

在 $ep$ 指向的MATLAB工程中从工作区复制名为 **name** 的mxArray。如果复制成功，返回指向mxArray的指针，否则返回0。当mxArray不再使用时，用mxDestroyArray来释放它所占的内存，见命令集198。

```
integer*4 function engPutMatrix(ep, mp)
integer*4 mp, ep
```

将 $mp$ 指向的mxArray复制到 $ep$ 指向的MATLAB工程的工作区中。如果在工作区中这个mxArray已存在，则覆盖它。如果复制成功，就返回0；否则返回1。

```
integer*4 function engGetFull(ep, name, m, n, pr, pi)
integer*4 ep, m, n, pr, pi
character*(*) name
```

从 $ep$ 指向的MATLAB工程工作区中复制名为 **name** 的 $m \times n$ 满矩阵，矩阵的元素是双精度浮点小数。参数 $pr$ 是矩阵实数部指针， $pi$ 是矩阵虚数部指针，参数 $m$ 、 $n$ 、 $pr$ 和 $pi$ 由函数来设置。如果复制成功，返回0；否则返回1。如果矩阵不再使用，就用mxFree来释放矩阵的实数部和虚数部所占的内存；见命令集198。

```
integer*4 function engPutFull(ep, name, m, n, pr, pi)
integer*4 ep, m, n, pr, pi
character*(*) name
```

和上个函数相似，但是将名为 **name** 的 $m \times n$ 满矩阵复制到工作区中，矩阵的元素是双精度浮点小数。如果工作区中已有这个矩阵，则覆盖它。

```
integer*4 ep integer*4 function engClose(ep)
```

关闭 $ep$ 指向的MATLAB工程。如果操作成功，返回0；否则返回1。

用下面的命令来编译和链接调用MATLAB的FORTRAN程序：

- UNIX：在系统提示符下输入一不间断行：

```
f77 -I/.../matlab/extern/include -o programname sourcecode.for
-L/.../matlab/extern/lib/... -R/.../matlab/extern/lib/...
-leng -lmat -lmx -lmi -lut
```

上面路径中的三个点，...，表示这部分路径与系统有关，**programname**是用户调用的程序名，**sourcecode.for**是要进行编译的FORTRAN源代码文件列表。如果需要，可以设置调试和优化标识；见编译器文档。

#### 15.3.4 MATLAB调用FORTRAN

在本节开始用一个简单的例子来描述 MATLAB调用FORTRAN，调用方式和在MATLAB

中调用普通函数M文件一样。

MEX文件使用的程序和矩阵格式和FORTRAN中调用MATLAB相同。然而现在也使用MEX文件。

文件中必须有一个叫做 `mexFunction` 的主函数，它是 MATLAB 调用的函数。这个程序有四个参数：输入参数的个数、输出参数的个数和两个指向这些参数的指针数组。所以这个函数可以调用任何计算程序。

#### 例15.6

创建一个 MEX 文件，返回用给定的矩阵元素和其下标相乘得到的一个新矩阵。下面的 FORTRAN 程序保存在文件 `rmult.for` 中：

```
C      A routine multiplying the elements with their row indices.
      subroutine radMult(Out, In, M, N)
      integer M, N
C      Dynamic memory in Fortran 77!
      real*8 Out(M, N), In(M, N)

      integer i, j

      do 100 i = 1, M
        do 110 j = 1, N
          Out(i, j) = real(i * In(i, j))
110      continue
100     continue

      return
      end
C      This routine is called by MATLAB.
      subroutine mexFunction(nlhs, plhs, nrhs, prhs)
      integer nlhs, nrhs
      integer plhs(*), prhs(*)

      integer In_ptr, Out_ptr
      integer In, Out
      integer M, N
      integer mxGetPr, mxGetM, mxGetN
      integer mxCreateFull, mxIsFull, mxIsDouble

C      Check the number of arguments and their type.
      In_ptr = prhs(1)
      In = mxGetPr(In_ptr)
      if (nrhs .ne. 1) then
        call mexErrMsgTxt("Only one input argument allowed!")
      elseif(nlhs .ne. 1) then
        call mexErrMsgTxt("Only one output argument allowed!")
      endif
      if ((mxIsFull(In_ptr) .ne. 1) .or.
$      (mxIsDouble(In_ptr) .ne. 1)) then
        call mexErrMsgTxt("Input argument must be a full
$      floating point matrix!")
      endif
```

```

C      Create a new matrix.
      M = mxGetM(In_ptr)
      N = mxGetN(In_ptr)
      Out_ptr = mxCreateFull(M, N, 0)
      Out = mxGetPr(Out_ptr)

C      Calls a routine that operates on the matrix.
      call radMult(%val(Out), %val(In), M, N)

C      Returns the new matrix ...
      plhs(1) = Out_ptr
      return
      end

```

在UNIX环境中编译前，必须将文件

```
/opt/matlab52/bin/mexopts.sh
```

拷贝到相同的目录下。然后编辑这个文件，在标题 sol2)下设置：

```

LDFLAGS="$-G -M $MATLAB/extern/lib/sol2/$MAPFILE
          -R $MATLAB/extern/lib/sol2
          -L $MATLAB/extern/lib/sol2
          -leng -lmat -lmx -lmi -lut$"

```

现在就可以用下面的命令来编译程序(在系统提示符下输入)：

```
/opt/matlab52/bin/mex rmult.for
```

注意，路径和系统有关。

当程序编译完后，可以在MATLAB提示符下输入：

```
New = rmult([1 2 3;1 2 3;1 2 3])
```

```

New =
     1     2     3
     2     4     6
     3     6     9

```

在下面的命令表中描述了所有的 MEX程序。它们可以在 MEX文件中使用，并且可以看作 FORTRAN编写的已编译的 M文件。

为了让 MATLAB能够调用 FORTRAN编写的程序，必须编写一个调用 mexFunction的接口函数。从 MEX文件中，也可以反过来调用，也就是用程序 mexCallMATLAB和 mexEvalString来在 FORTRAN中调用 MATLAB。

#### 命令集206 FORTRAN中的MATLAB接口

```

subroutine mexFunction(nlhs, plhs, nrhs, prhs)
integer*4 nlhs, nrhs, plhs(*), prhs(*)

```

是 MATLAB接口函数。参数 *nlhs* 是输出参数的个数，*prhs* 是这些参数的指针向量。同样，*nrhs* 是输入参数的个数，*prhs* 是输入参数的指针向量。在 FORTRAN程序中输入参数不能改变。

```
integer*4 function mexCallMATLAB(nlhs, plhs, nrhs, prhs, name)
integer*4 nlhs, nrhs, plhs(*), prhs(*)
character*(*) name
```

调用一个 MATLAB 函数，M 文件或 MEX 文件。参数 *nlhs* 是输出参数的个数 (必须小于或等于 50)，*prhs* 是这些参数的指针向量。同样，*nrhs* 是输入参数的个数 (必须小于或等于 50)，*prhs* 是输入参数的指针向量。字符串 **name** 是被调用的函数名。如果它是一个运算符，那么它必须放在两单引号之间，如 ‘ + ’。如果操作成功，返回 0；否则返回一个非零数。当 *prhs* 指向的 *mxArrays* 不再使用时，用 *mxFree* 来释放它们所占的内存；见命令集 198。

```
integer*8 function mexCallMATLAB(nlhs, plhs, nrhs, prhs, name)
integer*4 nlhs, nrhs
integer*8 plhs(*), prhs(*)
character*(*) name
```

和上个函数相似，但是用于 Alpha 和 SGI64 系统中。

```
subroutine mexSetTrapFlag(trap_flag)
integer*4 trap_flag
```

用来处理在使用 *mexCallMATLAB* 时发生的错误。如果 MATLAB 在调用 *mexCallMATLAB* 中有一个错误，就停止运行 MEX 文件，并且退回到 MATLAB 提示符下。如果将 *trap\_flag* 设为 0，也会进行相同的操作。但另一方面，如果将它设为 1，那么 MEX 文件将再获控制。

```
integer*4 function mexEvalString(command)
character*(*) command
```

在调用工作区中执行 MATLAB 命令 **command**。不会将结果返传给 MEX 程序。命令 **command** 的所有输入参数都必须可以在调用工作区中找到。如果操作成功，返回 0；否则返回一个非零数。

```
integer*4 function mexAtExit(ExitFcn)
subroutine ExitFcn( )
```

定义一个 MEX 程序退出前调用的函数。例如用来关闭文件的函数。总是返回 0。

在 *mexFunction* 中可以将数据作为参数来传递，这样就可以在 MATLAB 工作区中直接读和写数据。

#### 命令集 207 FORTRAN 和 MATLAB 之间的数据传递

```
integer*4 function mexGetMatrix(name)
character*(*) name
```

从调用工作区中拷贝名为 **name** 的 *mxArray*。如果拷贝成功，返回指向这个 *mxArray* 的指针；否则返回 0。当 *mxArray* 不再使用时，用 *mxFree* 来释放它所占的内存；见命令集 198。

```
integer*4 function mexGetGlobal(name)
character*(*) name
```

和上个函数相似，但是从 MATLAB 的全局工作区中拷贝 *mxArray*。

```
integer*4 function mexPutMatrix(mp)
integer*4 mp
```

和函数mxGetMatrix相似，但是是将mxArray拷贝到工作区中。如果拷贝成功，返回0；否则返回1。

```
integer*4 function mexGetMatrixPtr(name)
character*(*) name
```

返回调用工作区中名为 **name** 的mxArray指针。这使得可以在 MEX 程序中读和修改mxArray。不能对这个mxArray进行释放内存操作。

```
integer*4 function mexGetFull(name, m, n, pr, pi)
integer*4 m, n, pr, pi
character*(*) name
```

从调用工作区中拷贝名为 **name** 的  $m \times n$  满矩阵，矩阵的元素是双精度浮点小数。参数 *pr* 是矩阵实数部指针，*pi* 是矩阵虚数部指针 (*m*, *n*, *pr* 和 *pi* 由函数来设定)。如果拷贝成功，返回0；否则返回1。当矩阵不再使用时，用mxFree释放矩阵实数部和虚数部所占的内存；见命令集198。

```
integer*4 function mexPutFull(name, m, n, pr, pi)
integer*4 m, n, pr, pi
character*(*) name
```

和上个函数相似，但是是将  $m \times n$  的满矩阵拷贝到调用工作区中，矩阵的元素是双精度浮点小数。

下面的程序用来在FORTRAN中获取特殊常数值，如机器无穷小数和正无穷大数。还有用来检查变量值是否等于这些常数的函数。

#### 命令集208 FORTRAN中的特殊常数

```
real*8 function mexGetEps( )
```

返回MATLAB中机器无穷小数。

```
real*8 function mexGetInf( )
```

返回MATLAB中inf值，也就是无穷大数。

```
integer*4 function mexIsInf(value)
real*8 value
```

如果value等于inf，返回1；否则返回0。

```
real*8 function mexGetNaN( )
```

返回MATLAB中NaN的值。

```
integer*4 function mexIsNaN(value)
real*8 value
```

如果value等于NaN，返回1；否则返回0。

```
integer*4 function mexIsFinite(value)
real*8 value
```

如果value不等于inf或NaN，返回1；否则返回0。

下面的程序用来在 MATLAB 命令窗口中输出错误信息和其他文本内容：

#### 命令集209 FORTRAN中错误处理和打印

```
subroutine mexPrintf(formatarg1, arg2,...)
character*(*) formatarg1, arg2,...
```

在 MATLAB 窗口中输出 ANSI C 输出格式的字符串。参数 **format** 是 ANSI C 格式字符串，**arg1**，**arg2**，... 是输出的可选参数。

```
subroutine mexErrMsgTxt(error_msg)
character*(*) error_msg
```

在 MATLAB 窗口中输出错误信息 **error\_msg**，并且停止运行 MEX 文件。

用下面的命令来编译和链接调用 MATLAB 的 FORTRAN 程序：

- UNIX：在系统提示符下输入一不间断行：

```
.../matlab52/bin/mex sourcecode.for
```

上面路径中的三个点，...，表示这部分路径与系统有关，**programname** 是用户调用的程序名，**sourcecode.for** 是要进行编译的 FORTRAN 源代码文件列表。如果可能用户可以配置编译或链接环境；参见 MATLAB 5 手册《应用程序接口指南》和例 15.6。

- Windows：在 MATLAB 提示符下输入：

```
mex sourcecode.for
```

如果可能，用户可以配置编译或链接环境；参见 MATLAB 5 手册《应用程序接口指南》和例 15.6。

- Macintosh：参见 MATLAB 5 手册《应用程序接口指南》来配置，然后在 MATLAB 提示符下输入如下命令：

```
mex sourcecode.for
```

创建 MEX 文件将会有有一个系统决定的后缀名，可以借助 MATLAB 命令 **mexext** 来检查系统的后缀名类型。

可以对 MEX 程序进行调试，但是添加一个编译调试符— **g**。在 UNIX 环境下，编译完后，当 MATLAB 已运行，就可在系统提示符下输入— **ldbmx** 来运行程序。在 MATLAB 提示符下输入 **ldbmx on** 来运行要调试的 MEX 文件。然而，在 MEX—文件运行前，返回到调试器中来列程序、设置断点等。在 MATLAB 提示符下输入 **ldbmx stop**，可以到调试器中；在调试器中输入 **continue**，可以返回到 MATLAB 中。在 Windows 和 Macintosh 中调试可以参考 MATLAB 5 手册《应用程序接口指南》。

## 15.4 MATLAB和高级文件管理

MATLAB 中的下面这些命令对文件管理很有用：

#### 命令集210 文件名

```
fullfile(dir1,
dir2,fname)
```

连接一个字符串。字符串 **dir1**，**dir2**，... (目录名) 表示文件 **fname** 的路径。返回的字符串表示带有完整路径的文件 **fname**。MATLAB 会在目录 **dir1**，**dir2**，... 之间插入一个目

[path,name,ext,ver]=fileparts(file)	分隔符。分隔符由系统来决定,比如UNIX下用/,PC上用。 在变量path、name和ext中返回文件的路径、文件名和扩展名。 在VMS中,变量ver包含文件的版本号。 返回系统使用的目录分隔符。
filesep	

有时在MATLAB中要用二进制文件,所以如果另一个程序需要或生成某种特定格式的文件,那么就必须能对各种格式的文件进行读和写。以下面的例子开始。

### 例15.7

假设要在文件中存储一个Hadamard(哈达马德)矩阵,这可以用在2.8节讨论过的命令save来完成。然而在使用命令save和load时不能控制矩阵的格式。使用MATLAB中低级文件处理命令可以控制文件的格式和精度。

编写如下的代码,将一个 $64 \times 64$ 的哈达马德矩阵保存到文件hada.mtl中:

```
fp      = fopen('hada.mtl','w');
antok   = fwrite(fp, hadamard(64), 'int8');
[msg, err] = ferror(fp);

if err ~= 0
    disp('An error occurred when writing to the file:')
    disp(msg)
end

err = fclose(fp);

if err ~= 0
    disp('Could not close the file.')
end
```

从这个例子中可以看出,文件必须先以写模式打开,然后要关闭文件。在使用命令save和load时就不必进行这些操作。

MATLAB中用命令fopen和fclose来打开和关闭文件。

### 命令集211 打开和关闭二进制文件

fopen(filename, op) 打开名为字符串filename的文件,返回一个文件指针。如果发生错误,返回-1。字符串op表示文件的打开模式,可以有下列模式:

- 'r' 只读模式。
- 'r+' 读写模式。
- 'w' 覆盖已存在的文件。或若文件不存在,则创建新文件。只读模式。
- 'w+' 覆盖已存在的文件。或若文件不存在,则创建新文件。读写模式。



	‘a’	追加已存在的文件。或若文件不存在，则创建新文件。只写模式。
	‘a+’	追加已存在的文件。或若文件不存在，则创建新文件。读写模式。
	字母 ‘r’、‘w’、和 ‘a’	分别代表读、写和追加。PC和VMS用户必须区别二进制文件和文本文件。在字符串 <b>op</b> 中加一个字母 ‘t’ 来区别；见 help fopen。
[fp,msg]=fopen (filename,op,ark)		同上。如果发生错误， <i>fp</i> 被赋予值 -1，字符串 <b>msg</b> 保存的是错误信息，用来解释错误信息，参考有关 C 语言的手册。字符串 <b>arch</b> 规定数据存储的机器格式；输入 help fopen 可得更多信息。
[filename,op,ark] =fopen(fp)		返回文件 <i>fp</i> 的有关信息。字符串 <b>filename</b> 是文件的名称，字符串 <b>op</b> 是文件的打开模式，字符串 <b>arch</b> 是文件打开的机器格式。
fclose(fp)		关闭文件 <i>fp</i> 。如果关闭失败，返回 -1；否则返回 0。
fclose(‘all’)		关闭所有打开的文件。如果关闭失败，返回 -1，否则返回 0。

## 例15.8

(a) 以读和追加模式打开文件 **hada.mtl**：

```
[fp,msg] = fopen('hada.mtl','a+');

if fp == -1
    disp(msg)
end
[f,op,ark] = fopen(fp)
err        = fclose(fp)

if err ~= 0
    disp('Could not close the file.')
end

f =
    hada.mtl

op =
    a+

ark =
    ieee-be

err =
    0
```

(b) 关闭(a)中的文件：

```
err = fclose(fp)

err = 0
```

下面的两个命令 `fwrite` 和 `fread` 用来对二进制文件进行写和读操作。

#### 命令集212 写和读二进制文件

`fwrite(fp,A,prec)` 将矩阵 `A` 的列写入到文件 `fp` 中。函数返回写入元素的个数，字符串 `prec` 规定使用的精度，有 20 多种不同的精度可用。输入 `help fwrite` 可得更多信息。第 4 个参数可用在矩阵 `A` 的元素之间写空字节。

`fread(fp)` 从文件 `fp` 中读并返回数据。

`[A,c]=fread(fp,s,prec)` 从文件 `fp` 中读出数据到矩阵 `A` 中。矩阵 `A` 的大小由 `s` 来决定：

- `s=n` 一个长度为 `n` 的列向量。
- `s=inf` 文件 `fp` 中所有的数据读入到一个列向量中。
- `s=[m,n]` 从文件 `fp` 中的数据以列向量方式读入到  $m \times n$  矩阵中。

标量 `c` 是从文件中无错误地读出的元素个数。

`feof(fp)` 如果到 `fp` 所指的文件结尾，返回 1；否则返回 0。

#### 例15.9

下面对在例 15.7 中创建的文件 `hada.mtl` 进行读操作：

```
clear;

fp = fopen('hada.mtl','r');
A = fread(fp,[64,64],'int8');

fclose(fp);
whos
nnz(A-hadamard(64))    % Just checking...
```

结果显示：

Name	Size	Elements	Bytes	Density	Complex
A	64 by 64	4096	32768	Full	No
ans	1 by 1	1	8	Full	No
fp	1 by 1	1	8	Full	No

Grand total is 4098 elements using 32784 bytes

```
ans =
    0
```

这和写入到文件中的哈达马德矩阵相同。

MATLAB 可以创建和读带格式的文本文件。在这些操作中，MATLAB 以列向量的方式读和写矩阵的元素。然而 MATLAB 是以行向量的方式来读和写文本文件，所以有必要进行转换。

#### 命令集213 写和读带格式的文本文件

`fprintf(fp,fstr,A,...)` 依据字符串 `fstr` 给出的格式，将矩阵或数组 `A` 的元

素写入到文件 $fp$ 中。字符串 $fstr$ 包含的是象C程序语言中的格式字符。表15-1给出了最常用的代码列表,可见C语言手册或输入`help fprintf`,可得更多信息。函数最后返回写入元素的个数。

`fprintf(fstr,A,...)` 将格式数据输出到屏幕上。

`[A,c]=`

`fscanf(fp,fstr,s)` 从文件 $fp$ 中读出数据到矩阵 $A$ 中。如果 $s$ 是标量,那么 $A$ 就是一个列向量。如果 $s=[m,n]$ ,那么 $A$ 就是一个从 $fp$ 中以列向量方式读出的 $m \times n$ 矩阵。字符串 $fstr$ 规定数据的读出格式,和`fprintf`使用的格式字符相同。

`fgetl(fp)` 从文件 $fp$ 中读出一行到字符串中,函数返回这个字符串。

`fgets(fp)` 从文件 $fp$ 中读出包含`eol`(行结束)字符的下一行到字符串中,函数返回这个字符串。

在命令`fprintf`和`fscanf`以及5.1.2节中命令`sprintf`和`sscanf`的格式字符串中使用的字符列在表15-1中。

表15-1 命令FPRINTF和SPRINTF中使用的格式符

控制字符	格式符
<code>\n</code> 换行	<code>%e</code> 科学计数格式,小写e
<code>\r</code> 回车	<code>%E</code> 科学计数格式,大写E
<code>\b</code> 退格	<code>%f</code> 十进制格式
<code>\t</code> 横向跳格	<code>%s</code> 字符串
<code>\f</code> 新页	<code>%u</code> 整型数
<code>"</code> 单引号	<code>%i</code> 同上类型
<code>\\</code> 反斜杠	<code>%X</code> 十六进制,大写
<code>\a</code> 响铃	<code>%x</code> 十六进制,小写

除了这些外,还可以规定数据宽度、小数的位数和对齐方式。通常,MATLAB使用的是右对齐方式,但是在百分号和格式符之间的负号表示左对齐方式。数据的宽度和小数的位数也可以在百分号和格式符之间来规定,如`%[-][#. #]F`,括号内的部分是可选的,F是格式符;见表15-1。

#### 例15.10

(a) 用命令`sprintf`来输出不同格式的数值 $p$ :

```
twodec = sprintf('%4.2f',pi)    % 两位小数
```

```
twodec =
    3.14
```

```
ninedec = ...
    sprintf('The number pi = %11.9f',pi)    % 九位小数
```

```
ninedec =
    The number pi = 3.141592654
```

```
scfform = sprintf('%E',pi)    % 科学计数法
```

```
scfform =
    3.141593E+00
```

(b) 下面的程序将函数 $f(x)=1/x$ 在区间 $[0, 4]$ 上的值以格式表形式写入到文件 tab.txt 中。

```
% 写函数f(x) = 1/x 的格式表
% 左列是占四位的整数
% 右列占六位，且有多三位小数

x = 1:4; Y = zeros(4,2);

Y(:,1) = x';%           % Y的第1列
Y(:,2) = 1 ./ x';       % Y的第2列

fp = fopen('tab.txt','w+'); % 打开文件tab.txt

fprintf(fp,'%4.0f \t %6.3f \n',Y');
fclose(fp);
```

当程序运行时，会生成文件 tab.txt。可以用 type tab.txt 检查文件内容。

```
1      1.000
2      0.500
3      0.333
4      0.250
```

(c) 读(b)中创建的文件：

```
fp = fopen('tab.txt','r');
[Tab,c] = fscanf(fp,'%f %f',[2,4]);

fclose(fp);
Tab = Tab'

Tab =
    1.0000    1.0000
    2.0000    0.5000
    3.0000    0.3330
    4.0000    0.2500
```

注意，读取的矩阵必须进行转置，因为 MATLAB 以列方式创建矩阵，而文件是以行方式来读取的。

在每个文件操作中，都可能会发生错误。在每个文件操作之后，都应该进行错误处理，但在例 15.7 中没有进行错误处理。如果在最后一个文件操作中发生错误，可以通过调用产生错误代码和错误信息的函数 `ferror` 来进行错误处理。

#### 命令集214 错误信息

<code>msg=ferror(fp)</code>	如果在对文件 <code>fp</code> 的 I/O 操作过程中发生错误，返回一个错误信息。
<code>[msg,errn]=ferror(fp,</code>	返回上个命令中的错误信息，而且返回一个在 C

'clear')

语言参考手册中可查的错误代码。如果给出 'clear', 则清空错误信息缓冲区。

## 例15.11

在每个文件操作后都应该进行错误检查：

```
fp          = fopen('tab.txt','r');
[Tab,c]     = fscanf(fp,'%f %f',[3,4]);
[msg,errn] = ferror(fp);

if errn ~= 0
    disp('An error occurred when reading from tab.txt!')
    disp(msg)
end
```

```
fclose(fp);
Tab = Tab'
```

如果对例15.10(b)中创建的文件tab.txt进行操作，将会给出：

```
An error occurred when reading from tab.txt!
At end-of-file.
```

```
Tab =
    1.0000    1.0000    2.0000
    0.5000    3.0000    0.3330
    4.0000    0.2500         0
```

有三个函数可以来控制文件指针 *fp* 的位置：*fseek*、*ftell*和*frewind*。可以在文件内移动文件指针，检查文件指针的位置。

## 命令集215 文件指针的位置

<code>frewind(fp)</code>	将文件指针 <i>fp</i> 移动到文件开始处。
<code>fseek(fp,nb,u)</code>	移动文件指针到 <i>u</i> 和 <i>nb</i> 指定的位置。这里的 <i>nb</i> 是距离 <i>u</i> 规定的开始位置的字节数：
	- 1 文件起始处。
	0 当前位置。
	1 文件结尾。
	如果
	<i>nb</i> < 0 从 <i>u</i> 处向文件起始处移动指针 <i>nb</i> 个字节。
	<i>nb</i> = 0 移动指针到 <i>u</i> 处。
	<i>nb</i> > 0 从 <i>u</i> 处向文件结尾处移动指针 <i>nb</i> 个字节。
<code>ftell(fp)</code>	将从文件起始处读取的字节数作为文件指针的位置返回。负数表示已有错误发生。

## 15.5 和其他程序的结合

和其他程序结合使用 MATLAB 的可能性将由机器决定，可阅读手册和请教系统管理员。在本节中提到一些例子。

在 Lotus 1-2-3 的电子数据表格 WK1 中和 ASCII 电子数据表格中可以读和写文件。MATLAB 5.2 中有命令来读和写 hdf 文件和写 vrm1 文件。

### 命令集 216 读/写特殊文件格式

<code>dlmread(fname,st,r,c,v)</code>	从文件 <b>fname</b> 中读一个 ASCII 电子数据表格。表格中单元有分隔符 <b>st</b> ，例如 ‘\t’ 表示制表符。如果给出 <b>r</b> 和 <b>c</b> ，表示从 <b>r</b> 行和 <b>c</b> 列处开始读取表格单元（数字 10 开始计数）。如果给出向量 <b>v</b> = <b>[r1, c1, r2, c2]</b> ，表示读取 ( <b>r1, c1</b> ) 和 ( <b>r2, c2</b> ) 之间的部分表格单元。( <b>r1, c1</b> ) 表示左上单元，( <b>r2, c2</b> ) 表示右下单元。
<code>dlmwrite(fname,A,st,r,c)</code>	将 MATLAB 矩阵 <b>A</b> 转换成一个 ASCII 电子数据表格保存到文件 <b>fname</b> 中。表格中单元要有分隔符 <b>st</b> 。如果给定 <b>r</b> 和 <b>c</b> ，就从 <b>r</b> 行和 <b>c</b> 列的开始处写入矩阵 <b>A</b> （数字从 0 开始计数）。
<code>wklread(fname,r,c,v)</code>	从文件 <b>fname</b> 中读取 Lotus 1-2-3 WK1 电子数据表格。如果给定 <b>r</b> 和 <b>c</b> ，表示从 <b>r</b> 行和 <b>c</b> 列处开始读表格单元数据（数字从 0 开始）。如果给定的 <b>v</b> 是向量 <b>[r1, c1, r2, c2]</b> ，表示读取 ( <b>r1, c1</b> ) 和 ( <b>r2, c2</b> ) 之间的部分表格单元。( <b>r1, c1</b> ) 表示左上单元，( <b>r2, c2</b> ) 表示右下单元。也可以以字符串形式给出区间 <b>v</b> ，如 ‘a1...c5’ 或 ‘sales’。
<code>wklwrite(fname,A,st,r,c)</code>	将 MATLAB 矩阵 <b>A</b> 转换成一个 Lotus 1-2-3 WK1 电子数据表格保存到文件 <b>fname</b> 中。如果给定 <b>r</b> 和 <b>c</b> ，就从 <b>r</b> 行和 <b>c</b> 列的开始处写入矩阵 <b>A</b> （数字从 0 开始）。
<code>hdf</code>	对 hdf 格式文件进行读和写操作。使用 <code>help hdf</code> 可得更多信息。为了能在 MATLAB 中使用函数对 hdf 文件进行操作，必须熟悉 hdf 库；相关信息可在 <a href="http://hdf.ncsa.uiuc.edu">http://hdf.ncsa.uiuc.edu</a> 上找到。也可参见命令集 162 中的命令 <code>imread</code> 和 <code>imwrite</code> 。
<code>vrm1(h,filename)</code>	写一个 vrm1 2.0 文件，文件中包含有句柄 <b>h</b> 的图形对象和它的子对象。用字符串 <b>filename</b> 得到文件，如果没有给出后缀名，就加上后缀名 <code>.wrl</code> 。如果没有给出文件名，就使用 <code>matlab.wrl</code> 作为文件名。

如果 MATLAB 运行在 Microsoft Windows 环境下，可以使用 DDE (动态数据交换) 来和其他程序交换数据。这可以用在 MS Word 和 MATLAB 之间进行数据交换。对 MATLAB Notebook Suite 有访问权的用户可以在 Word 中交互地写他们自己的 MATLAB 命令，代码中可以混合有

MATLAB产生的无格式文本和图形。而且，可以通过矩阵编辑器来处理数组等。

在UNIX网络中，可以在其他计算机上启动 MATLAB工程，这样就可以使用计算机上的空闲资源来完成计算，而在自己本地计算机上有图形和命令窗口。

MATLAB有许多的工具箱，这里提到其中几个重要的工具箱：

- The Signal Processing Toolbox      用于信号处理。
- The Optimization Toolbox          用于优化。
- The Symbolic Math Toolbox        通过链接 Maple V用于符号数学。

有关这些和其他工具箱的更多信息可以输入 `expo`，然后选择 `TOOLBOXES` 来得到；也可参见附录C。总之，它们可以描述成 M文件的集合，每个集合都用于某个特定的领域。

MATLAB结合其他程序使用还存在很大的可能性，在将来的 MATLAB版本中会进行进一步的开发。