目　录

第 1 章　MATLAB 入门

MATLAB 产品是用来解决工程与科学实际问题的工程软件，而外部接口编程是该软件的一项基本功能。掌握 MATLAB 的基本使用方法是学习 MATLAB 外部接口编程的基础，所以在正式学习使用 MATLAB 外部接口编程之前，首先回顾一下 MATLAB 软件的基本环境及其使用方法，并了解一下 MATLAB 的产品体系。

本章重点内容

- MATLAB 产品简介；
- MATLAB 基本环境；
- MATLAB 的 M 语言编程。

1.1　MATLAB 产品简介

MATLAB 的名称源自 Matrix Laboratory，它的首创者是在数值线性代数领域颇有影响的 Cleve Moler 博士，同时他也是生产和经营 MATLAB 产品的美国 Mathworks 公司的创始人之一。MATLAB 本身是一种科学计算软件，专门以矩阵的形式处理数据。MATLAB 将高性能的数值计算和可视化集成在一起，并提供了大量的内置函数，还提供了一种高级的解释型编程语言——M 语言。MATLAB 产品具有良好的开放性和扩充性，利用 MATLAB 提供的科学计算能力和 M 语言编程能力开发了很多专业函数并组成了不同的工具箱，而这些工具箱产品被广泛地应用于科学计算、控制系统、信息处理等领域的分析、仿真和设计工作中。大多数的 MATLAB 工具箱中所包含的代码都是开放的，这使得工程师在使用 MATLAB 软件解决工程问题，不断深化对问题认识的同时，还可以非常容易地对 MATLAB 的功能进行扩充，从而不断完善 MATLAB 产品以提高产品自身的竞争能力。

目前 MATLAB 产品族主要应用于以下领域：

- 数值分析；
- 数值和符号计算；
- 工程与科学绘图；
- 控制系统的设计与仿真；
- 数字图像处理；
- 数字信号处理；
- 财务与金融工程；
- 图形化的用户界面开发。

MATLAB 产品由若干个模块组成，不同的模块完成不同的功能，其中包括：

- MATLAB；

- MATLAB Toolboxes;
- MATLAB Compiler;
- Simulink;
- Simulink Blocksets;
- Real-Time Workshop (RTW);
- Stateflow;
- Stateflow Coder;

由这些模块构成的 MATLAB 产品体系如图 1-1 所示。其中，MATLAB 是 MATLAB 产品家族的基础，它提供了基本的数学算法，例如矩阵运算、数值分析算法等，MATLAB 集成了 2D 和 3D 图形功能，以完成相应的数值可视化工作，同时 MATLAB 还提供了一种交互式的高级编程语言——M 语言，利用 M 语言可以通过编写脚本或者函数文件实现用户自己的算法。

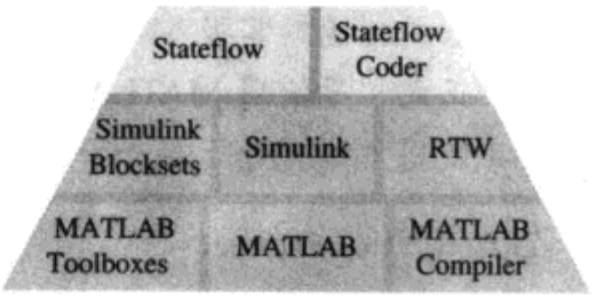


图 1-1 MATLAB 的产品体系

MATLAB 不仅能够和 C/C++语言进行集成开发，而且还提供了和 Java 语言接口的能力，另外它还支持 COM 标准，能够和任何支持 COM 标准的软件协同工作。

目前，MATLAB 整个产品的最新发布版本为 Release 13，表示 MATLAB 产品体系的第 13 次发布，整个产品族不仅包含了 MATLAB 的基本功能模块，还包含了应用于各种专业的工具箱等产品。MATLAB 基本功能模块的最新版本为 6.5.1。

Simulink 产品是用来对复杂动态系统进行建模和仿真的图形化交互式平台，利用该产品，用户只要通过简单的鼠标操作就可以建立起各种复杂的动态系统，其中包括简单系统、离散系统、连续系统，或者由几种系统共同组成的复杂系统，它是 MATLAB 产品族中重要的组成部分。目前 Simulink 产品的最新版本为 5.1。

Simulink Blocksets 是 Simulink 的扩展，包括了应用于不同行业(专业)的功能模块集合，目前 MATLAB 产品中包含的主要功能模块集合如表 1-1 所示。

表 1-1 MATLAB 中包含的功能模块集合

Aerospace	应用于航空航天飞行器系统建模的功能模块集合
DSP Blockset	应用于数字信号处理系统开发、建模和仿真的功能模块集合
Communication	应用于通信系统建模仿真的功能模块集合
Dials & Gauges	以图形化形式显示 Simulink 信号和仿真参数的功能模块集合
CDMA	基于 IS-95A 标准的无线通信系统建模仿真的功能模块集合
SimPowerSystem	针对电力电子系统进行建模仿真的功能模块集合
SimMechanics	针对机械系统进行建模仿真的功能模块集合

MATLAB 产品族包含的功能模块的详细信息请参阅 MATLAB 的相关文档，或查阅网上信息 www.mathworks.com 或者 www.hirain.com。

RTW 为 Real-Time Workshop 的缩写，该产品是将 Simulink 框图模型转变成为标准 C 语言的工具，这样的 C 源程序结合具体的实时软件和硬件，可以完成实时条件下的动态系统测试仿真，例如快速控制原型仿真或者硬件在回路中的仿真。在众多实时仿真系统中，目前最流行、性能最出色的产品为德国 dSPACE 公司研发的 dSPACE 系统，有关 dSPACE 系

统的详细信息可以查阅网上信息 www.dspaceinc.com 或者 www.hirian.com。

Stateflow 产品是以 Simulink 产品为基础的图形化建模仿真环境，它是基于有限状态机理论对事件驱动模型进行建模和仿真的图形化环境，可以用于复杂逻辑控制、状态切换系统的建模和仿真。利用 Stateflow Coder 可以将 Stateflow 模型转变成为标准 C 代码，这样就可以结合 RTW 生成的代码共同完成实时系统的仿真。

MATLAB 整个产品体系中共有 70 余个产品模块，如果需要了解这些产品的详细信息可以参阅 MATLAB 的相关文档。

在介绍 MATLAB 外部接口编程之前，先介绍一下 MATLAB 基本环境的使用方法，以及利用 MATLAB 提供的高级编程语言——M 语言进行编程的基本方法。如果读者对 MATLAB 基本使用方法比较了解，可以快速浏览本小节的内容，或者直接略过本章内容。而对于那些对 MATLAB 不甚熟悉的读者，最好仔细阅读一下本小节的内容，为学习后面章节的内容打好基础。

1.2　MATLAB 桌面环境

MATLAB 的桌面环境可以包含多个窗口，这些窗口分别为历史命令窗口(Command History)、命令行窗口(Command Window)、当前目录浏览器(Current Directory Browser)、工作空间浏览器(Workspace Browser)、目录分类窗口(Launch Pad)、数组编辑器(Array Editor)、M 文件编辑器/调试器(Editor/Debugger)、超文本帮助浏览器(Help Navigator/Browser)。这些窗口都可以内嵌在 MATLAB 主窗体下，组成 MATLAB 的用户界面。

当 MATLAB 安装完毕并首次运行时，展示在用户面前的界面为 MATLAB 运行时的缺省界面，如图 1-2 所示。

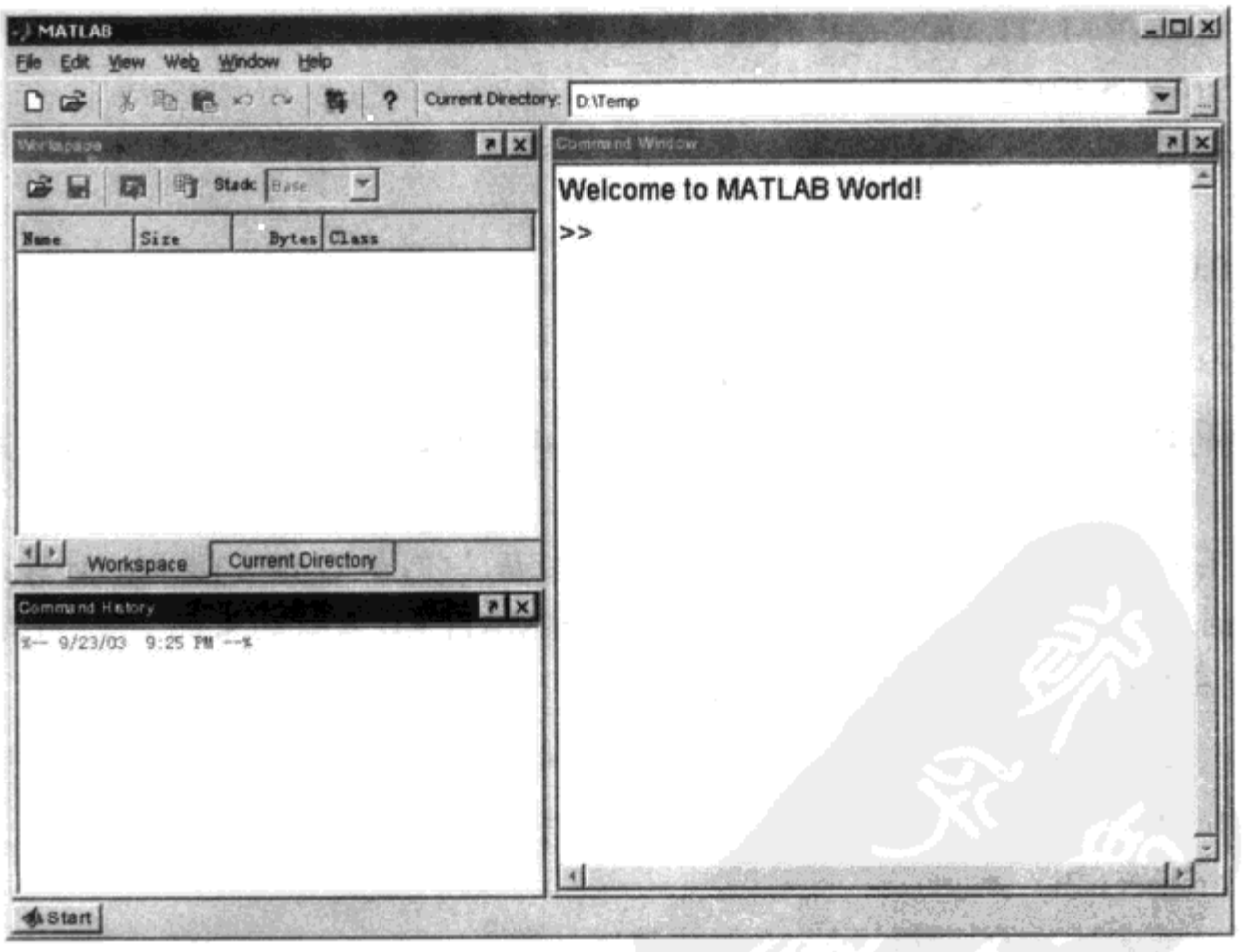


图 1-2　MATLAB 启动之后默认的界面

MATLAB 的桌面环境可以通过 View 菜单中 Desktop Layout 子菜单下的命令进行切换，这些命令分别为：

- Default：缺省的界面，如图 1-2 所示，其中包含历史命令窗口(Command History)、命令行窗口(Command Window)，此外工作空间浏览器(Workspace)和当前目录浏览器(Current Directory)两个窗口层叠在一起。
- Command Windows Only：仅包含命令行窗口(Command Window)，此时 MATLAB 界面的外观类似于旧版本的 MATLAB。
- Simple：包含两个窗口——命令行窗口(Command Window)和历史命令窗口(Command History)，两个窗口并列在界面中。
- Short History 和 Tall History：这两个菜单命令包含的窗口类型和数量同默认的界面完全一致，只是排放的顺序不同。
- Five Panel：包含所有的 MATLAB 桌面窗口，在该界面中各个窗口处于平铺状态。

在 MATLAB 用户界面的 View 菜单下还有一些菜单命令可以用来选择显示在图形界面中的窗口，用户可以根据自己的喜好选择配置用户界面，推荐使用 Command Windows Only 的外观样式，此时仅显示一个窗口——MATLAB 的命令行窗口，这样相对占用的系统资源少，启动 MATLAB 的速度较快，同时执行 MATLAB 程序的效率也较高。

在上述各种 MATLAB 窗口中，最常用的就是 MATLAB 的命令行窗口，它最具特色的就是其命令回调的功能，也就是说在 MATLAB 的命令行窗口键入任意算术表达式，系统将自动解算，并给出结果，见下面的例子。

例 1-1　计算算术表达式 $\frac{-5}{(4.8+5.32)^2}$。

只要直接在 MATLAB 的命令行窗口中键入：

```
>> -5/(4.8+5.32)^2↙
```

系统将直接计算表达式的计算结果，并且给出答案：

```
ans =
    -0.0488
```

注意：

这里的符号“>>”为 MATLAB 的命令行提示符；这里的符号“↙”表示键入表达式之后按回车键。

MATLAB 的数学运算符同其它的计算机高级语言(例如 C 语言)类似。计算得到的结果显示为 ans，ans 是英文单词“answer”的缩写，它是 MATLAB 默认的系统变量。所有 MATLAB 的计算结果和数值都默认使用双精度类型显示。

例 1-2　复数的运算 $(1+2i)\times(1-3i)$。

在 MATLAB 命令行窗口中键入：

```
>> (1+2i)*(1-3i) ↙
```

系统直接计算表达式的计算结果，并给出答案：

```
ans =
    7.0000 - 1.0000i
```

注意：

在 MATLAB 中表示复数按照例 1-2 中所示的样式，即其中 x 和 y 都是双精度的数字。在这里，i 作为复数单位存在，同样也可以使用 j 表示复数单位。

在上面的两个例子中都是将 MATLAB 直接作为计算器来使用的，在 MATLAB 的命令窗口中还可以定义相应的 MATLAB 数据对象和变量以及调用函数。

例 1-3　调用函数。

```
>> cos(pi/2)
ans =
   6.1232e-017
>> exp(acos(0.3))
ans =
    3.5470
```

在例 1-3 中调用了余弦函数来求 π/2 的余弦值。数学知识告诉我们 π/2 的余弦应该为 0，但是 MATLAB 求的数值不是 0，而是一个近似为 0 的数值，这是由 MATLAB 浮点数计算精度造成的。在调用函数的时候，需要注意括号的作用，它会使计算的优先级发生变化。在例 1-3 中，首先计算反余弦函数，然后再计算指数函数。

MATLAB 的基本运算单位是矩阵，所以在 MATLAB 命令行中创建矩阵是最基本的操作。在例 1-4 中演示了创建矩阵和索引矩阵元素的方法。

例 1-4　创建矩阵。

```
>> A = [1 2 3;4 5 6;7 8 90]
A =
     1     2     3
     4     5     6
     7     8    90
>> A(1,3)
ans =
     3
>> A(7)
ans =
     3
```

在例 1-4 中，首先创建了一个三阶的方阵，然后分别使用全下标方式和单下标方式索引了矩阵中的同一个元素。创建矩阵时，可以使用逗号“,”或者空格作为列与列元素之间的间隔，使用分号“;”表示行与行之间的间隔。在使用单下标进行元素索引时需要注意，MATLAB 的矩阵索引是以列元素优先的。关于全下标和单下标方式索引矩阵元素的方法，请参阅《MATLAB 基础与编程入门》一书或者 MATLAB 的帮助文档。

MATLAB 的功能是通过大量的 M 语言函数或者 MATLAB 内建的函数来完成的，在命令行窗口中，调用这些函数的方法就是直接键入函数或者指令，并且根据不同的函数提供相应的参数列表。MATLAB 的命令行窗口具有命令行记忆的功能，也就是说，在命令行窗口中，使用上下光标键就可以重复以前键入的指令，这对使用 MATLAB 是非常便利的。MATLAB 还可以具有局部记忆的功能，例如在 MATLAB 的命令行窗口中曾经执行了一个名为 testcommandwindows 的函数，那么再次运行该函数时，只要在命令行中键入 test，然后按光标上键(↑)，整条命令会出现在命令行窗口中，再按回车键就可以执行该指令了。

1.3 使 用 帮 助

任何 MATLAB 的使用者，都应当学会使用 MATLAB 的帮助系统。因为 MATLAB 和相应的工具箱包含了上万条不同的指令，每条指令函数都对应着一种不同的操作或者算法，没有哪个人能够将这些指令都清楚地记忆在脑海中，MATLAB 的帮助系统则为用户使用 MATLAB 指令提供了便捷的索引环境。MATLAB 的帮助系统是学习 MATLAB 最好的教科书，它的讲解清晰、透彻，所以养成良好的使用 MATLAB 帮助系统的习惯，对于使用 MATLAB 的用户来说是非常必要的。

在 MATLAB 中提供了两种类型的帮助系统：在线帮助和窗口帮助。

1.3.1 在线帮助

所有的 MATALB 函数都具有自己的帮助信息，这些帮助信息都保存在相应的函数文件注释区中，这些帮助信息是由那些编写函数的工程人员在编写函数的同时添加在函数内的，所以，这些信息能够最直接地说明函数的用途，或者函数需要的一些特殊的输入参数，以及函数的返回变量等。甚至在有些函数中，将函数采用的算法也在这里加以说明。另外，获取在线帮助的过程也非常快捷，因此，MATLAB 的用户最常用的帮助就是在线帮助。获取在线帮助的指令是 help 或者 helpwin。

例 1-5 获取在线帮助。

在 MATLAB 命令行窗口中，键入下面的指令：

```
>> %获取帮助主题
>> help
HELP topics:
matlab\general           -  General purpose commands.
matlab\ops               -  Operators and special characters.
matlab\lang              -  Programming language constructs.
matlab\elmat             -  Elementary matrices and matrix manipulation.
matlab\elfun             -  Elementary math functions.
... ...
>> %获取帮助主题下的函数列表
>> help elfun
```

```
    Elementary math functions.
    Trigonometric.
      sin          - Sine.
      sinh         - Hyperbolic sine.
      asin         - Inverse sine.
      asinh        - Inverse hyperbolic sine.
... ...
>> %获取具体函数的帮助
>> help sin
 SIN      Sine.
     SIN(X) is the sine of the elements of X.
 Overloaded methods
     help sym/sin.m
```

在例 1-5 中，使用的省略符号是为了缩减篇幅而用，在实际的 MATLAB 中，将给出全部内容。

在线帮助不仅可以显示在命令行窗口中，还可以显示在 MATLAB 的帮助窗口中，内容仍然是在线帮助的内容，例如：

```
>> %在窗口中显示在线帮助信息
>> helpwin sin
```

这时 sin 函数的在线帮助将显示在帮助窗口中，如图 1-3 所示。

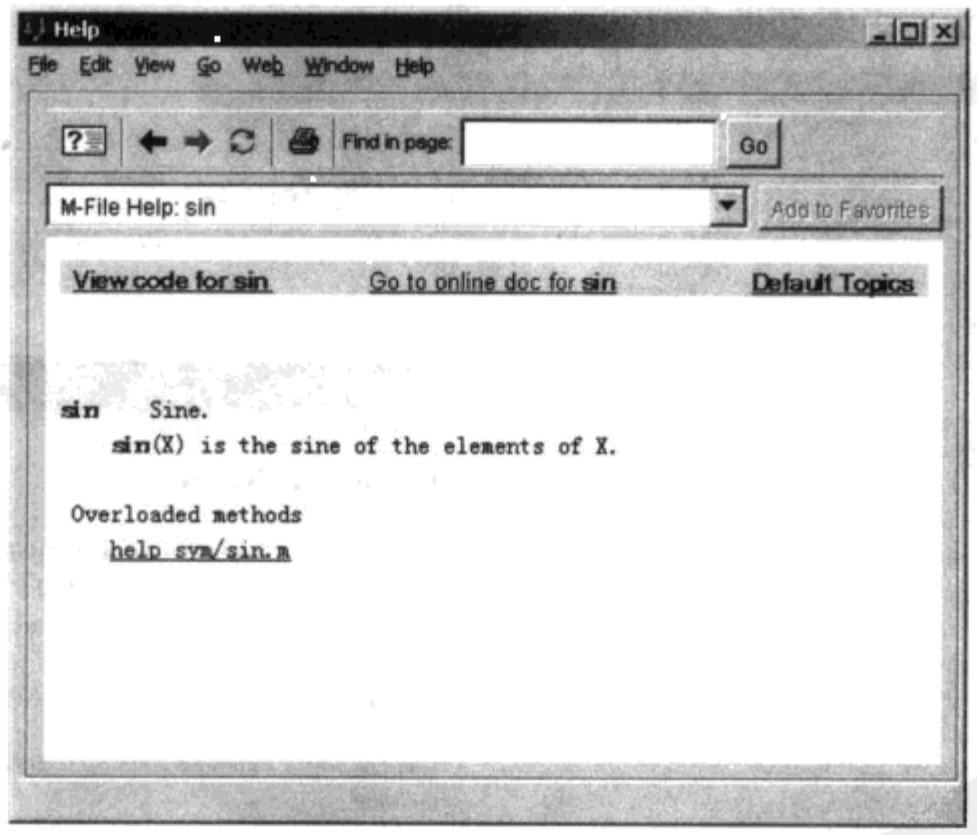


图 1-3　显示在窗口中的在线帮助

所有的 MATLAB 函数还具有一类在线帮助，叫作 H1 帮助行，这部分内容在每一个 M 语言函数文件的在线帮助的第一行，它能够被 lookfor 函数搜索查询。在 H1 帮助行中，往往是言简意赅的说明性语言，它在所有的帮助中相对最重要。例如，在 MATLAB 命令行窗口中键入：

```
>> %使用 H1 帮助行
>> lookfor Fourier
FFT Discrete Fourier transform.
FFT2 Two-dimensional discrete Fourier Transform.
FFTN N-dimensional discrete Fourier Transform.
IFFT Inverse discrete Fourier transform.
IFFT2 Two-dimensional inverse discrete Fourier transform.
IFFTN N-dimensional inverse discrete Fourier transform.
... ...
```

这时 MATLAB 将所有有关傅立叶变换的函数罗列在命令行窗口中，这些函数的 H1 帮助行都有关键字 Fourier。

关于编写 MATLAB 的在线帮助和 H1 帮助行的方法，将在第 4 章进行详细讲述。

1.3.2 窗口帮助

尽管在线帮助使用起来简便、快捷，但是在线帮助能够提供的信息毕竟有限，而且并不是所有与函数有关的内容都可以用在线帮助的形式表示，比如数学公式，图形等，因此，MALTLAB 还提供了内容更加丰富的帮助文档，作为 MATLAB 的用户指南出现。目前 MATALB 的帮助文档有英文版和日文版，而在中国地区使用的 MATLAB 只有英文版的帮助文档。

MATLAB 的帮助文档显示在 MATLAB 的帮助窗口中，单击 MATLAB 用户界面上的 ? 按钮，将打开 MATLAB 的帮助文档界面，如图 1-4 所示。

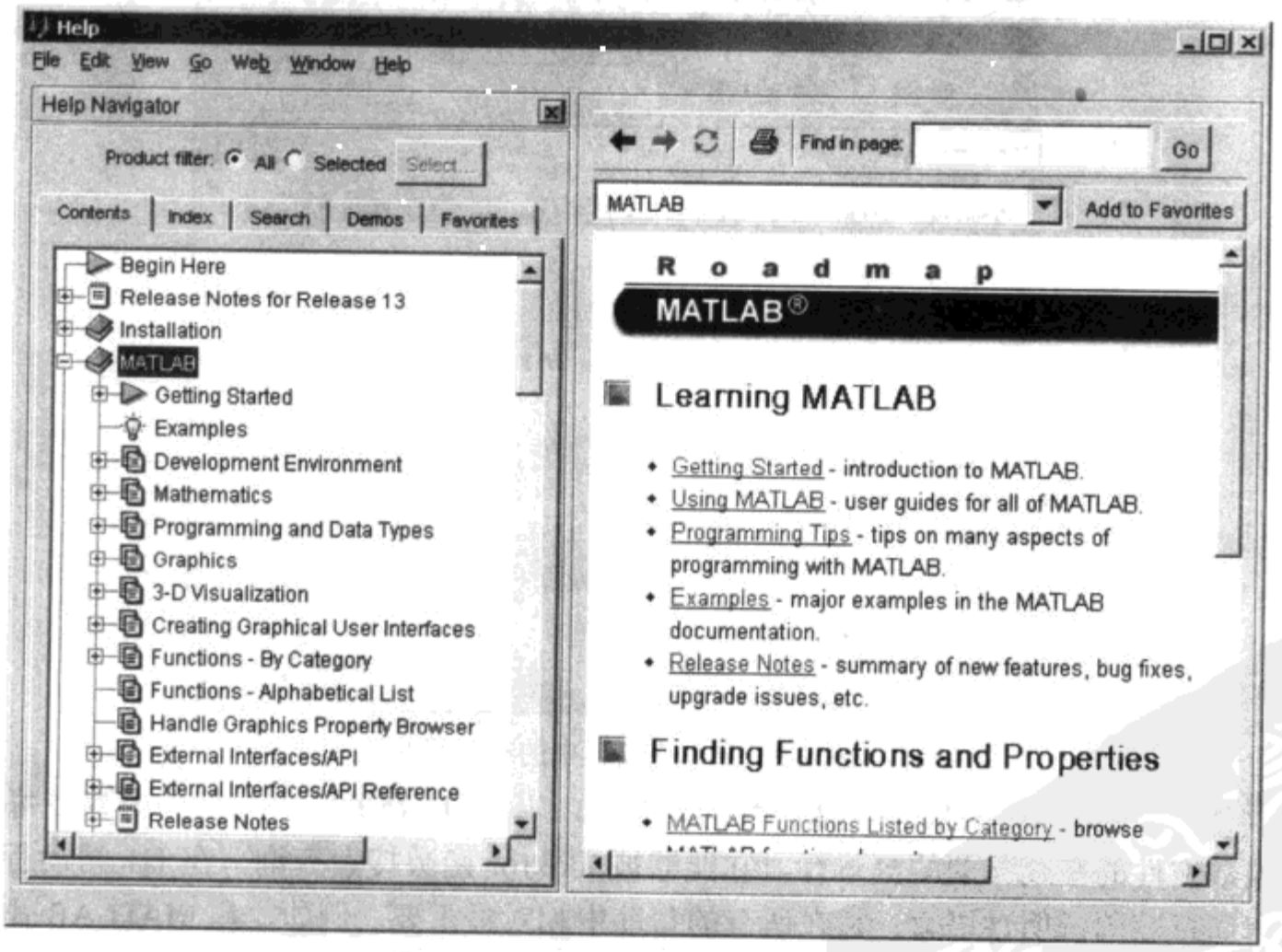


图 1-4 MATLAB 的帮助文档界面

这里看到的 MATLAB 帮助文档是跟随 MATLAB 产品一同发布的文档光盘经过安装之后的超文本内容。界面中的 Contents 标签页罗列了所有产品帮助文档的目录，单击这些目录以及目录下面的文章标题，就可以在右边的窗体中具体浏览帮助信息。除此之外，在帮助文档界面还具有下面几个标签页：

Index 标签页：关键字索引查询。

Search 标签页：关键字全文搜索。

Demos 标签页：MATLAB 演示例子。

Favorites 标签页：个人喜好的书签页。

在这些标签页中，用户使用频率最高的就是 Contents 标签页，一般地，学习 MATLAB 不可避免地需要阅读帮助文档，而就笔者的经验而言，直接阅读帮助文档是学习 MATLAB 最有效的方法。

此外，使用频率最高的就是 Demos 标签页了。MATLAB 为每一个工具箱或者模块都设计了很多演示示例，通过这些例子学习 MATLAB 往往能够起到事半功倍的效果。这些演示程序的作用非常独特，往往连帮助文档都无法替代其功用，所以对于初学者来说，在阅读帮助文档的基础上，多研习 MATLAB 的 Demos，是一种学习 MATLAB 的最佳方法。

MATLAB 的帮助文档除了超文本格式以外，还具有 PDF 格式的帮助文档，这些帮助文档与 MATLAB 的产品手册(纸版)一一对应，甚至在新版的 MATLAB 中，PDF 文件格式的帮助文档内容要多于超文本格式的文档，更是多于纸版的手册，所以在必要的情况下，可以将部分 PDF 格式的文档打印出来，作为手册保存。MATLAB 6.5 的 PDF 格式文档可以从 MATLAB 产品的第三张光盘上直接拷贝。阅读这些文档，需要安装 Adobe Acrobat Reader 4.0 以上版本的阅读器。

尽管 MATLAB 的帮助文档比较详实、规范，用户在使用 MATLAB 的过程中，不可避免地还是会遇到一些问题，这个时候可以使用 MATLAB 的网上资源。MATLAB 在互联网上的资源非常丰富，不仅在 Mathworks 公司的主页上可以找到很多有用的信息，而且在国内的网站上也有一定规模的信息资源，特别是在北京九州恒润科技有限公司(简称九州恒润科技公司)的主页上，拥有一个以 MATLAB 应用为主的论坛，大家在使用 MATLAB 的过程中若是遇到了问题，可以直接在该论坛上发表自己的问题，九州恒润公司的技术人员会很快为大家解决相应的问题。

此外，还可以通过 E- mail 向 Mathworks 公司的技术人员进行技术问题的询问，不过在提出问题的同时需要提供用户产品的信息，这些信息是可以通过在 MATLAB 命令行中键入 ver 指令来获得的，将出现在命令行窗口中的 MATLAB License Number 内容提供给 Mathworks 公司即可。

1.3.3　操作帮助的函数

MATLAB 还提供了一些函数用于操作帮助和帮助浏览器，如表 1-2 所示。

表 1-2 帮助函数

函 数	说 明
help	在 MATLAB 命令行中显示在线帮助
helpwin	在帮助浏览器中显示在线帮助
helpbrowser	打开帮助浏览器，并显示超文本帮助文档
helpdesk	与函数 helpbrowser 的功能一致，在早期版本的 MATLAB 中可以打开帮助界面
doc	打开帮助浏览器，并显示指定的内容
docroot	帮助文档存在的根目录
demo	打开帮助浏览器并显示 Demos 标签页
dbtype	显示 M 文件内容，同时包括文件代码行号
lookfor	搜索 H1 帮助行
web	打开帮助浏览器并显示指定的超文本链接内容

在这些函数中最常用的是 help 函数和 doc 函数，两者命令行语法基本相同，只不过 doc 函数将打开 MATLAB 的帮助浏览器，并显示相应函数的超文本帮助文档。其它函数的具体用法就不再一一叙述了，有兴趣的读者可以查阅帮助文档或者在线帮助。

1.4 MATLAB 的数据类型

作为一种编程语言——M 语言同样提供了各种数据类型，这些数据类型同样可使用 MATLAB 工具箱函数。M 语言类似于 C 语言，提供了包括整数类型、双精度类型、布尔类型、字符串类型等多种数据类型，还提供了像元胞数组这样的特殊类型。利用 MATLAB 面向对象的编程能力，还可以自定义用户的特殊数据类型。在图 1-5 中，对 MATLAB 的数据类型进行了总结。

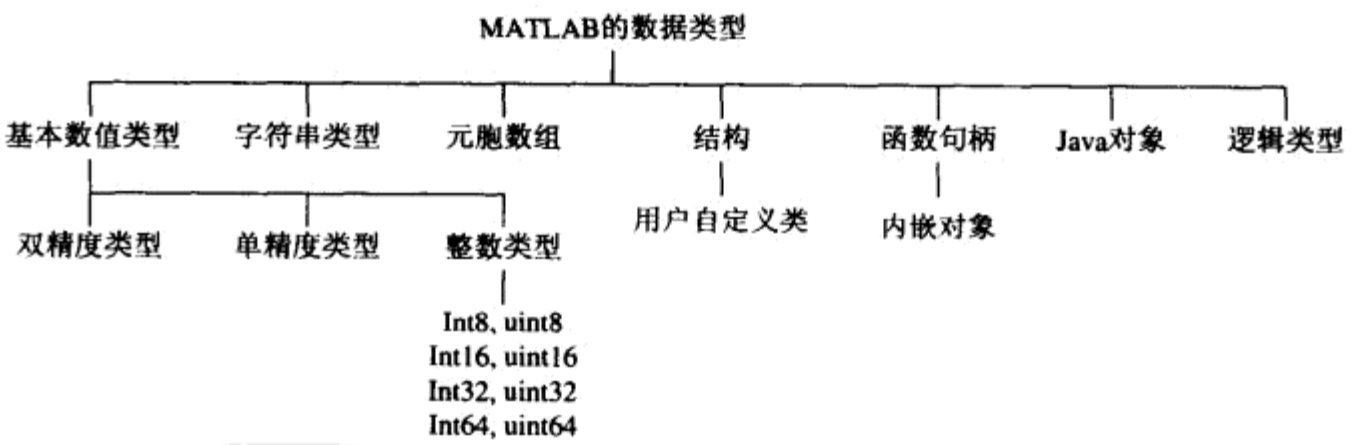


图 1-5 MATLAB 的数据类型

在图 1-5 所示的各种数据类型中，字符串类型和双精度类型是所有 MATLAB 数据类型的基础。在 MATLAB 命令行中创建矩阵或者对变量赋数值时，都默认使用双精度类型。在例 1-6 中演示了使用双精度类型数据和字符串类型数据的方法。

例 1-6　双精度类型数据和字符串类型数据。

在 MATLAB 命令行中，键入下面的指令：

```
>> A = [ 1 2 3];
>> class(A)
ans =
double
>> whos
  Name        Size                    Bytes  Class
  A           1x3                        24  double array
  ans         1x6                        12  char array
Grand total is 9 elements using 36 bytes
```

在 MATLAB 命令行中，键入下面的指令：

```
>> a = 127
a =
    127
>> class(a)
ans =
double
>> size(a)
ans =
     1     1
>> b = '127'
b =
127
>> class(b)
ans =
char
>> size(b)
ans =
     1     3
```

创建字符串时，只要将字符串的内容用单引号括起来就可以了，若需要在字符串内容中包含单引号，则需要在键入字符串内容时，连续键入两个单引号即可，例如：

```
>> c = 'Isn''t it?'
c =
Isn't it?
```

在 MATLAB 中包含了多种操作双精度类型和字符串类型数据的函数，具体的可以参阅《MATLAB 基础与编程入门》一书，或者查阅 MATLAB 的帮助文档。

在 MATLAB 中同样可以定义多维数组，见下面的例子。

例 1-7　多维数组的创建。

在 MATLAB 的命令行中，键入下面的指令：

```
>> A = pascal(4)
A =
     1     1     1     1
     1     2     3     4
     1     3     6    10
     1     4    10    20
>> A(:,:,2) = eye(4)
A(:,:,1) =
     1     1     1     1
     1     2     3     4
     1     3     6    10
     1     4    10    20
A(:,:,2) =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

创建多维数组的方法非常简单，在例 1-7 中，首先对 A 进行赋值，这里使用的是 pascal 函数创建四阶方阵，然后利用另外一个函数 eye 创建四阶方阵，同时对 A 的第二页进行赋值即可。创建多维数组的方法就是直接对相应的维进行赋值即可。

例 1-8　元胞数组和结构数组的创建。

在 MATLAB 命令行中，键入下面的指令：

```
>> A = {zeros(2,2,2) , 'Hello' ; 17.35,1:100}
A =
    [2x2x2 double]    'Hello'
    [       17.3500]    [1x100 double]
>> B = [{zeros(2,2,2)},{'Hello'};{17.35},{1:100}]
B =
    [2x2x2 double]    'Hello'
    [       17.3500]    [1x100 double]
>> C   = {1}
C =
    [1]
>> C(2,2) = {3}
C =
    [1]      []
```

```
    []    [3]
>> isequal(A,B)
ans =
     1
>> whos
  Name        Size                    Bytes  Class
  A           2x2                      1122  cell array
  B           2x2                      1122  cell array
  C           2x2                       144  cell array
  Ans         1x1                         1  logical array
Grand total is 243 elements using 2389 bytes
```

这里演示了创建元胞数组的基本方法，注意创建元胞数组时“{}”的使用方法。元胞数组是 MATLAB 特有的数据类型，可以将其看作无所不包的通用矩阵。在早期版本的 MATLAB 中元胞数组扮演着非常重要的角色，很多函数的输入、输出参数都使用了元胞数组。但是在新版本的 MATLAB 中元胞数组逐渐被结构数组替代，下面演示创建结构的方法。

在 MATLAB 命令行中，键入下面的指令：

```
>> Student.name = 'Way';
>> Student.age = 26;
>> Student.grade = uint16(1);
>> whos
  Name        Size                    Bytes  Class
  Student     1x1                       388  struct array
Grand total is 8 elements using 388 bytes
>> Student
Student =
    name: 'Way'
     age: 26
   grade: 1
```

结构是由相应的字段和记录共同组成的，在 MATLAB 中创建结构的方法非常简单，只需直接对结构的相应字段进行赋值就可以完成结构数组的创建。

有关 MATLAB 各种数据类型的详细操作方法请参阅 MATLAB 的帮助文档，或者阅读《MATLAB 基础与编程入门》一书。

1.5　M 语言编程入门

MATLAB 提供了完整的编写应用程序的能力，这种能力通过一种被称为 M 语言的高级解释性语言来实现。利用该语言编写的代码仅能被 MATLAB 接受，并被 MATLAB 解释和执行。

从编程语言的角度上看，M 语言同 C 语言等高级编程语言非常类似，如果读者对 C 语

言比较熟悉，则学习 M 语言的编程将没有任何的障碍。和 C 语言文件类似，M 语言文件都是标准的纯文本格式的文件，其文件的扩展名为.m。在本小节，将介绍 M 语言编程的基本方法。

提示：

所有的 M 语言文件都可以使用任何一种纯文本编辑器进行编辑，也可以使用 MATLAB 提供的 meditor 编辑。在 MATLAB 中启动 meditor 的方法是在命令行窗口中键入指令：

```
>> edit filename
```

这时将启动 meditor，同时打开或创建名为 filename 的文件，如图 1-6 所示。

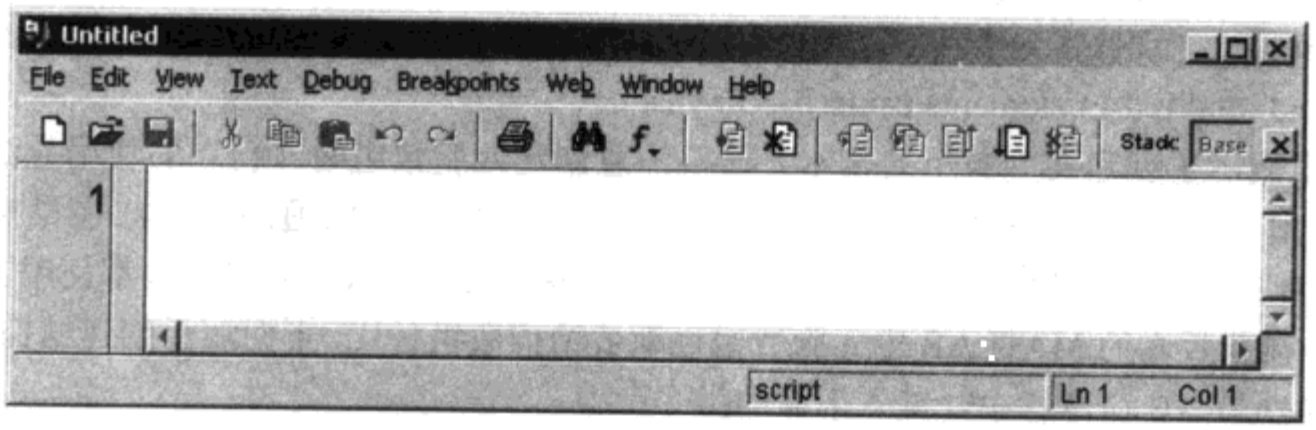


图 1-6 MATLAB 的编辑器窗口

1.5.1 流程控制

程序流程控制包含控制程序流程的基本结构和语法。结构化的程序主要有三种基本的程序结构：顺序结构、选择结构、循环结构。

顺序结构就是指所有组成程序源代码的语句按照由上至下的次序依次执行，直到程序的最后一个语句，也就是程序语句简单的罗列；而选择结构是依照不同的判断条件进行判断，然后根据判断的结果选择某一种方法来解决某一个问题，这样的结构就是选择结构；循环结构就是在程序中某一条语句或者多条语句重复多次的运行结构。

上述的三种程序结构以及相应结构的组合足以处理各种各样的复杂问题，在 MATLAB 中，M 语言文件都是由上述三种结构的 MATLAB 指令构成的。顺序结构相对容易理解，这里仅向大家介绍一下 MATLAB 的选择结构和循环结构。

1. 选择结构

如前所述，当人们判断某一条件是否满足，根据判断的结果来选择不同的解决问题的方法时，就需要使用选择结构。MATLAB 的选择结构需要使用 if 语句或者 switch 语句。

if 语句组成选择结构的基本语法结构有三种，分别如下：

(1) if (关系运算表达式)

    MATLAB 语句

end

这种形式的选择结构表示当关系运算表达式计算的结果为逻辑真的时候，执行 MATLAB 语句，这里的 MATLAB 语句可以是一个 MATLAB 表达式，也可以是多个 MATLAB 表达式。在 MATLAB 语句的结尾处，必须有关键字 end。

(2) if(关系运算表达式)

　　MATLAB 语句 A

else

　　MATLAB 语句 B

end

这种选择结构表示当关系运算表达式的计算结果为逻辑真的时候，则执行 MATLAB 语句 A，否则执行 MATLAB 语句 B，在语句 B 的结尾处必须具有关键字 end。

(3) if (关系运算表达式 a)

　　MATLAB 语句 A

 Elseif(关系运算表达式 b)

　　MATLAB 语句 B

 Else (关系运算表达式 c)

 ……

end　(　　)

这种选择结构可以判断多条关系运算表达式的计算结果，然后按照执行的逻辑关系执行相应的语句。读者可以根据类似的 C 语言知识或者前面两种选择结构的介绍判断这种结构的运算执行方式。

和 C 语言类似，if-elseif-else 的语句结构可以嵌套使用，也就是说，可以存在这样的语句结构：

if(关系表达式 a)

　　if(关系表达式 b)MATLAB 语句 A

　　else MATLAB 语句 B

　　end

else

　　if(关系表达式 c) MATLAB 语句 C

　　else MATLAB 语句 D

　　end

end

注意，在使用这种结构的选择结构时，需要小心 if 语句和 end 关键字的配对。

另外一种构成选择结构的关键字就是 switch。在处理实际问题的时候，往往要处理多个分支，这时如果使用 if-else 语句处理多分支结构往往使程序变得十分冗长，从而降低了程序的可读性，而 switch 语句，就可以用于处理这种多分支的选择。它的基本语法结构如下：

switch(表达式)

　　case 常量表达式 a：MATLAB 语句 A

　　case 常量表达式 b：MATLAB 语句 B

　　……　……

　　case 常量表达式 m：MATLAB 语句 M

　　otherwise　　：MATLAB 语句 N

end

注意：

MATLAB 的 switch 语句和 C 语言的 switch 语句结构不同，在 C 语言中，每一个 case 后面的语句中必须包含类似 break 语句的流程控制语句，否则程序会依次执行符合条件的 case 语句后面的每一个 case 分支，但是在 MATLAB 中就不必如此，程序仅仅执行符合条件的 case 分支。

在 switch 语句之后的表达式可以是一个表达式或者一个变量，当这个表达式的值同 case 后面的某一个常量表达式相等时，则执行该 case 常量表达式后面的语句。另外，在处理以字符串参与的关系判断操作时，使用 switch 结构要比 if-else 结构效率好一些。由于 MATLAB 的 switch 结构没有 C 语言的 fall-through 特性，所以，如果要针对多个条件来使用同一个 case 分支的时候，就需要使用元胞数组与之配合。例如下面的代码片段：

```
switch var
    case 1
        disp('1')
    case {2,3,4}
        disp('2 or 3 or 4')
    case 5
        disp('5')
    otherwise
        disp('something else')
end
```

在上面的代码片段中，当 var 取值为 2、3、4 时，都是用同一个 case 分支。

2. 循环结构

在解决很多问题的时候都需要使用循环结构，例如求解数列的和或者利用某种迭代法求解数值方程时，都需要循环结构配合完成计算。

在 MATLAB 中包含两种循环结构，一种是循环次数不确定的 while 循环，而另一种是循环次数确定的 for 循环。

while 语句可以用来实现“当”型的循环结构，它的一般形式如下：

```
while(表达式)
    MATLAB 语句
end
```

当表达式为真时，循环将执行由语句构成的循环体，其特点是先判断循环条件，如果循环条件成立，即表达式运算结果为“真”，再执行循环体。循环体执行的语句可以是一条也可以是多条，在 MATLAB 语句之后必须使用关键字 end 作为整个循环结构的结尾。另外，在循环过程中一定要能够改变关系表达式或者布尔类型变量的值，或者使用其它方法来跳出循环，否则会陷入死循环(无法正常退出的循环叫作死循环)。

使用 for 语句构成循环是最灵活、简便的方法，不过，使用 for 语句循环需要预先知道循环体执行的次数，所以这种循环一般叫作确定循环。在 MATLAB 中 for 循环的基本结构

如下：

```
for index = start:increment:end
    MATLAB 语句
end
```

其中，index 的取值取决于 start 和 end 的值，一般地，这里通常使用等差的数列向量。

在 for 循环语句中，不仅可以使用行向量进行循环迭代的处理，也可以使用矩阵作为循环次数的控制变量，这时循环的索引值将直接使用矩阵的每一列，循环的次数为矩阵的列数，例如下面的代码片段：

```
A = rand(3,4);
for i = A
    sum = mean(i)
end
```

上面的代码片段中使用了一个矩阵作为循环的索引值，于是循环结果就分别计算矩阵的每一列元素的均值。

和其它高级语言类似，MATLAB 的循环结构也可以进行嵌套使用，使用嵌套的循环需要注意 for 关键字和 end 关键字之间的配对使用，请读者根据高级语言的一般特性来推断，这里就不再赘述。

在读者使用 M 语言进行编程的时候，要充分利用 MATLAB 以向量(矩阵)为基本运算单位的特点，使用向量化的代码不仅可以缩短程序的长度，提高代码的可读性，还能够提高程序的执行效率，见例 1-9。

例 1-9　向量化的代码提高程序的执行效率。

```
Mass = rand(5,10000);
Length = rand(5,10000);
Width = rand(5,10000);
Height = rand(5,10000);

[rows, cols] = size(Mass);

disp([char(10), '使用数组运算：'])
tic
Density = Mass./(Length.*Width.*Height);
toc

disp([char(10), '使用循环结构：'])
tic;
for I = 1:rows
    for J = 1:cols
        Density(I) = Mass(I,J)/(Length(I,J)*Width(I,J)*Height(I,J));
```

```
    end
end
toc
```

例 1-9 比较了循环结构和数组运算的执行效率，程序的运行结果如下：

```
>> array_vs_loops
使用数组运算的结果为
elapsed_time =
        0

使用循环结构的结果为
elapsed_time =
    0.0100
```

通过程序运行的结果可以看出，数组运算和循环迭代结构在计算效率方面的差距，特别是在循环迭代层次较多的时候，数组运算的速度优势越明显。

M 语言尽管是一种解释型的语言，执行效率上无法和 C 语言这种编译型语言相比，但是随着 MATLAB 版本的升级，M 语言代码执行的效率也在不断提高。特别是 MATLAB Release 13 中包含的 MATLAB JIT 加速功能，将 M 语言中针对标量、循环等结构的处理速度提高了很多，一般的总会有 10 倍，甚至 100 倍速度的提升。MATLAB 性能加速器在 MATLAB 6.5 中就默认设置为开启(On)状态，广大 MATLAB 的用户可以充分利用加速器带来的好处。

有关如何提高 M 语言文件的执行效率以及 MATLAB 性能加速器的内容请参阅 MATLAB 的相关文档。

1.5.2 脚本文件

所谓脚本文件，就是由一系列的 MATLAB 指令和命令组成的纯文本格式的 M 文件，执行脚本文件时，文件中的指令或者命令按照出现在脚本文件中的顺序依次执行。脚本文件没有输入参数也没有输出参数，执行起来就像早期的 DOS 操作系统的批处理文件一样，而脚本文件处理的数据或者变量必须在 MATLAB 的公共工作空间中，如下例所示。

例 1-10 脚本文件示例。

```
% 注释行
%  M 脚本文件示例
theta = -pi:0.01:pi;
rho(1,:) = 2*sin(5*theta).^2;
rho(2,:) = cos(10*theta).^3;
rho(3,:) = sin(theta).^2;
rho(4,:) = 5*cos(3.5*theta).^3;
for k = 1:4
    % 图形输出
```

```
    subplot(2,2,k)
    polar(theta,rho(k,:))
end
disp('程序运行结束!')
```

上面的例子中使用了 MATLAB 的图形功能在图形窗口中绘制数据。读者可以在 MATLAB 的编辑器——meditor 中编辑该文件，并在当前的工作路径下保存文件名为 script_example.m，然后执行该文件，在 MATLAB 命令行窗口中键入：

```
>> script_example
```

得到的结果如图 1-7 所示。

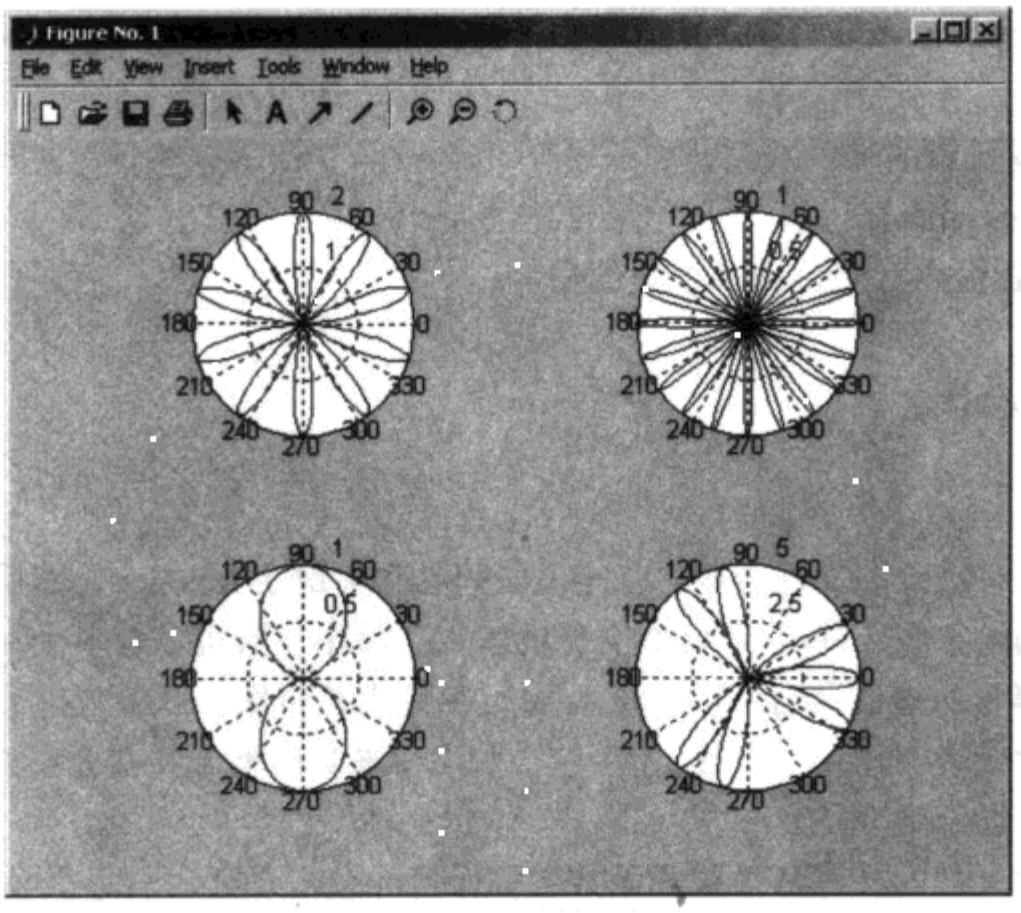


图 1-7　脚本文件执行的结果

在上述脚本文件中，“%”后面的内容为注释行，可以用来对程序进行解释，这些注释行的内容不会被显示或执行。

1.5.3　函数文件

函数文件是 M 文件最重要的组成部分，M 语言函数文件能够接受用户的输入参数，计算并将计算结果作为函数的返回值返回给调用者。在 MATLAB 中具有不同类型的函数，分别为内建函数、系统 M 函数、系统 MEX 函数文件、用户自定义 MEX 函数文件和用户自定义的 M 文件。用户自定义的 M 函数文件相对于脚本文件复杂得多，不仅具有输入、输出参数，还必须具有特殊的文件结构。本小节将结合具体的例子介绍函数文件的有关知识。

例 1-11　简单函数文件。

```
function y = average(x)
% AVERAGE 求向量元素的均值
% 语法:
```

```
% Y = average(X)
% 其中，X 是向量,Y 为计算得到向量元素的均值
% 若输入参数为非向量则出错

% 代码行
[m,n] = size(x);
% 判断输入参数是否为向量
if (~((m == 1) | (n == 1)) | (m == 1 & n == 1))
    % 若输入参数不是向量，则出错
    error('Input must be a vector')
end
% 计算向量元素的均值
y = sum(x)/length(x);
```

在 MATLAB 命令行中，键入下面的指令：

```
>> z = 1:99;
>> y = average(z)
y =
    50
```

函数文件具有特殊的文件结构，首先是必须具有关键字 function，然后一般的函数都具有输入参数和输出参数，并且 M 函数文件的输入、输出参数可以有多个。以函数定义行的等号为分界，等号右边的所有参数都成为输入参数或者叫作右手参数，等号左边的所有参数都是输出参数或者叫作左手参数。左手参数和右手参数的概念将在介绍 C 语言 MEX 函数文件的时候再次详细地介绍。

另外，函数文件的文件名必须和函数名称保持一致，否则会引起不必要的错误。

例 1-12　多个输入、输出参数的函数文件。

```
function varargout=varargout_example(varargin)
%VARARGOUT_EXAMPLE 不确定个数的输出参数

% 判断输出参数的个数
str = sprintf('输出参数的个数 :=%d',nargout);
disp(str);
if(nargout <= nargin)
    for k=1:nargout
        varargout{k} = varargin{nargin-k+1};
    end
end
```

例 1-12 所示的代码就可以使用多个输入参数和多个输出参数，代码中包含的 varargout 和 varargin 分别用来处理输出参数和输入参数，而 nargout 和 nargin 函数可以用来判断输入

参数和输出参数的个数。

关于 MATLAB 函数文件的详细内容介绍读者可以参阅 MATLAB 的帮助文档或者《MATLAB 基础与编程入门》一书。

1.6 本 章 小 结

在本章中重点介绍了 MATLAB 的产品体系，回顾了 MATLAB 的基本操作以及 M 语言编程等内容。MATLAB 是用于解决实际工程问题强大的数学分析、设计以及仿真软件。该软件目前被广泛地应用于科学计算、控制系统设计与仿真、数字信号处理系统的设计与仿真，甚至金融财经等领域。该软件以矩阵为基本运算单位，以其良好的开放性和灵活性为不同领域的工程师提供了完整的解决方案。如果读者从未接触或者使用过 MATLAB 软件，仅仅通过本章的学习是无法全面掌握 MATLAB 的基本应用和概念的，特别是本书后面章节将详细讲述外部接口编程，需要读者对 M 语言编程有充分的认识和了解。所以，建议对 MATLAB 比较生疏的读者应该仔细阅读一下 MATLAB 的帮助文档中有关基本操作和 M 语言编程方面的内容，或者阅读一下《MATLAB 基础与编程入门》一书。从下一章内容开始，笔者均假设读者已经基本掌握了 MATLAB 的应用，并且对 C 语言编程有充分的认识和了解。

第 2 章　MATLAB 外部接口概述

尽管利用MATLAB提供的高级编程语言——M语言进行程序开发已经基本能够满足一般算法开发的需要了，但是在很多情况下，MATLAB 的用户仍需要将 MATLAB 同其它的软件或者开发语言结合在一起共同完成算法开发的任务，例如在工程中广泛应用的 C 语言或者 Fortran 语言。广大 MATLAB 的用户提出的需求包括：

- 在 MATLAB 中调用已有的 C 语言或者 Fortran 语言的代码。
- 在 C 语言或者 Fortran 语言中调用 MATLAB 的算法。
- 利用 COM 标准进行客户端/服务器模式开发，例如在 Visual Basic 程序或者 Microsoft Excel 中调用 MATLAB 的算法。
- 在 MATLAB 中直接加载动态链接库。
- 在 M 语言中使用 Java 类。

在解决这些问题的时候，都可以使用 MATLAB 的外部接口功能。本章将对 MATLAB 的外部接口应用进行简要的入门介绍。

本章重点内容

- MATLAB 外部接口应用的类型；
- mxArray 数据结构；
- mx 函数；
- MATLAB 的环境配置。

2.1　外部接口应用的类型

MATLAB 应用范围非常广泛，但是在不同专业领域都有适合自己的开发软件或者语言，例如 C 语言、Fortran 语言、Visual Basic，甚至 Microsoft Excel 软件等。为了解决 MATLAB 用户提出的各种与其它开发工具或者语言交互的需求，MATLAB 提供了不同的工具用来完成相应的功能，如表 2-1 所示。

通过表 2-1 可以看到，MATLAB 同其它软件开发工具或者语言交互的能力是非常强大的，特别是 MATLAB 在 Windows 平台上支持 Microsoft 提出的 COM 标准，同时支持 Java 语言，所以 MATLAB 几乎可以同 Windows 平台上任何一种软件或者开发语言进行交互。不过，使用不同的工具解决不同的问题，往往一类问题又有很多种解决方法，但不是每一种都适合或者最佳。例如在解决利用 C 语言调用 MATLAB 算法的问题上，可以分别通过计算引擎和 MATLAB 函数库的方法解决，但是不同的方法各有优点和缺点，需要针对用户的需要来选择。

在解决各种软件交互问题的方法中，外部接口是相对容易实现也是最重要的一种方法。

MATLAB 的外部接口应用包括如下内容。

表 2-1　MATLAB 与其它开发工具语言交互的能力

问　题	解决方法	需要工具
在 MATLAB 中调用 C 或者 Fortran 语言代码	MEX 文件	MATLAB
在 C 语言中调用 MATLAB 算法	计算引擎	MATLAB
	MATLAB 函数库	MATLAB、MATLAB Compiler
在 Fortran 语言中调用 MATLAB 算法	计算引擎	MATLAB
MATLAB COM 客户端/服务器开发 Visual Basic/Excel 程序中调用 MATLAB	COM 应用	MATLAB
	MATLAB 组件打包	MATLAB、MATLAB Compiler、 MATLAB COM Builder、 MATLAB Excel Builder
在 C 语言/Fortran 语言中读写 MAT 数据文件	MAT 应用	MATLAB
M 语言中调用 Java 类	Java 语言	MATLAB
加载动态链接库函数	直接加载	MATLAB

1. 使用 MEX 文件调用已有的 C 代码或者 Fortran 代码

创建 MEX 文件是外部接口应用的重点，也是本书所讲述内容的重点。MEX 从字面上是MATLAB Executable 两个单词的缩写。MEX 文件类似于 M 文件，是一种能够在 MATLAB 环境中可以被 MATLAB 的解释器解释并执行的函数文件。MEX 文件可以直接使用 C 语言或者 Fortran 语言来编写，在相应的 C 语言或者 Fortran 语言代码中，就可以调用已有的大量 C 语言或者 Fortran 语言算法。在 MATLAB 中调用 MEX 文件时，就好像调用 MATLAB 的内建函数或者 M 文件函数一样，语法结构和输入、输出参数都符合 MATLAB 的标准。利用 MEX 文件可以完成：

- 在 MATLAB 中使用已有的 C 语言或者 Fortran 语言代码，避免重复劳动。
- 解决 M 语言运行速度的瓶颈。
- 隐藏算法的细节。
- 通过 C 语言对计算机硬件设备进行操作，扩展 MATLAB 的能力。

 **注意：**

由于 MATLAB 版本的更新，MEX 文件解决 M 语言运行速度瓶颈的能力有所削弱。另外，M 语言是一种高效率的编程语言，在大多数的情况下，推荐使用 M 文件来完成算法开发的工作。

通过 C 语言编写 MEX 函数文件的方法将在第 3 章中详细讲述，而 Fortran 语言 MEX 文件的编写将在第 4 章中详细讲述。

2. 通过 MATLAB 计算引擎在 C 代码或者 Fortran 代码中调用 MATLAB

MATLAB 计算引擎应用程序是那些通过特殊的通信方式与 MATLAB 进程相互交互的 C 语言或者 Fortran 语言应用程序。计算引擎在不同的平台上使用不同的通信方法，例如在 UNIX 平台上使用管道(pipes)，而在 Windows 平台上使用 COM 接口——IEngine。

MATLAB 计算引擎从应用上与 MEX 文件正好相对，它提供了在 C/Fortran 语言应用程序中调用 MATLAB，将 MATLAB 作为后台计算处理平台的能力。利用 MATLAB 计算引擎可以完成：

■ 利用 C 语言或者 Fortran 语言开发前台应用程序，调用 MATLAB 丰富的算法库，充分利用不同开发工具的优势和长处。

■ 在 UNIX 平台中，用户不仅可以在本地计算机上调用 MATLAB 计算引擎服务，而且还可以通过网络调用其它计算机平台上的 MATLAB 计算引擎服务，从而充分利用网络上的计算资源。

MATLAB 的计算引擎应用简单、灵活，但是惟一的缺点就是 C 语言或者 Fortran 语言开发的应用程序无法脱离 MATLAB 环境，所以使得应用计算引擎必须安装一套 MATLAB。MATLAB 计算引擎的应用将在第 5 章中详细讲述。

3. C 语言或者 Fortran 语言应用程序读写 MAT 数据文件

MAT 数据文件是 MATLAB 独有的一种数据文件格式，这种数据文件是一种二进制文件，能够跨平台使用。MAT 数据文件的后缀名称是.mat。

一般地，MAT 数据文件都是在 MATLAB 环境中进行读写的，不过可以利用 MATLAB 提供的函数库完成 C 语言或者 Fortran 语言读写 MAT 数据文件的程序开发，这样就能够充分利用 MAT 数据文件跨平台应用的优势了。

有关 MAT 文件以及在 C 语言或者 Fortran 语言应用程序中读写 MAT 文件的方法将在第 6 章中详细讲述。

4. 在 M 语言中调用 Java 类

从 MATLAB 5.3 开始，MATLAB 环境中就包含了 Java 虚拟机，其软件的图形化用户界面也利用 Java 语言进行了开发。在 MATLAB 环境中，M 语言和 Java 语言可以混合编程，结合两种开发语言的优势，完成丰富的功能。

MATLAB 的 Java 接口包括：

■ 调用 Java API 类(class)和包(package)，完成 Java 核心功能。

■ 调用第三方定义的 Java 类(class)。

■ 在 MATLAB 环境下创建 Java 对象。

■ 使用 Java 语法或者 MATLAB 语法使用 Java 对象的方法。

■ 在 Java 对象和 MATLAB 之间交互数据。

Java 语言是一种高级编程语言，目前被广泛利用在网络开发等领域。将 M 语言和 Java 语言结合起来可以为 M 语言开发更丰富的代码，例如网络 I/O、文件 I/O、图形用户界面等。

MATLAB 的 Java 接口能力应用将在第 7 章中详细讲述。

1. 在 MATLAB 中加载动态链接库

Windows 平台下的动态链接库包含了一组函数，这些函数可以被不同的应用程序来调用，这些文件就是共享库在 Windows 平台下的实现，这些文件的扩展名为.dll。在应用程序运行的时候，动态库被自动加载到内存中，利用库函数完成计算或者某些操作。MATLAB 提供了相应的接口函数，可以将通用的动态链接库加载到 MATLAB 环境中，这样在 M 语言程序中就可以直接调用动态链接库包含的函数了。

注意：

在 MATLAB 中加载动态链接库是 MATLAB 最新版 6.5.1 新增加的功能，MATLAB 早期的版本还不具备这些能力。如果需要在 MATLAB 6.5 版本中实现加载动态链接库的功能，则需要在 Mathworks 公司的网站上下载一个补丁文件，该文件的超链接如下：

ftp://ftp.mathworks.com/pub/tech-support/solutions/s33513/GenericDll_1p1.exe

将该文件下载并安装后，就可以在 MATLAB 6.5 中实现该功能了。

MATLAB 直接加载动态链接库的内容将在附录 B 中详细讲述。

6. MATLAB 的 COM 应用

Windows 平台上的 MATLAB 可以完成 COM 客户端或者服务器应用程序的开发，尽管 MATLAB 的 COM 应用也是外部接口的内容，但是由于 COM 应用相对难度较大，涉及的软件工具种类多，所以关于 COM 应用的内容将在《MATLAB 应用程序集成与发布》一书中详细讲述。同样有关动态数据交换(DDE)的内容也将在《MATLAB 应用程序集成与发布》一书中详细讲述。在上述各种外部接口应用中，最重要也是最常用的就是 MEX 文件，所以在本书中，将重点讲解 MEX 文件的创建。

2.2　mxArray 数 据 结 构

MATLAB 软件是以矩阵为基本运算单位的，这与 C 语言或者 Fortran 语言不同，在 C 语言或者 Fortran 语言中，只有基本的双精度类型、整数类型、字符类型的变量定义，它们和 MATLAB 中相应类型的变量定义是不一样的。为了能够在 C 语言中表示 MATLAB 的数据，MATLAB 提供了一个特殊的 C 语言结构——mxArray，这个结构就是用来在 C 语言中表示 MATLAB 数据的，在 MATLAB 的 C 语言外部接口编程中将大量使用该结构。

2.2.1　mxArray 的定义

mxArray 本身是一个 C 语言的结构，该结构的定义在 Matrix.h 文件中，它的定义如下：

```
struct mxArray_tag {
    void        *reserved;
    int          reserved1[2];
    void        *reserved2;
    int          number_of_dims;
    unsigned int reserved3;
    struct {
        unsigned int    scalar_flag : 1;
        unsigned int    flag1 : 1;
        unsigned int    flag2 : 1;
        unsigned int    flag3 : 1;
        unsigned int    flag4 : 1;
        unsigned int    flag5 : 1;
```

```
        unsigned int      flag6 : 1;
        unsigned int      flag7 : 1;
        unsigned int      private_data_flag : 1;
        unsigned int      flag8 : 1;
        unsigned int      flag9 : 1;
        unsigned int      flag10 : 1;
        unsigned int      flag11 : 4;
        unsigned int      flag12 : 8;
        unsigned int      flag13 : 8;
    }   flags;
    unsigned int reserved4[2];
    union {
        struct {
            void   *pdata;
            void   *pimag_data;
            void   *reserved5;
            int     reserved6[3];
        }   number_array;
    }   data;
};
```

mxArray 结构由很多个字段组成，这些字段完成了记录 MATLAB 数据的工作。一般地，mxArray 在记录一个数据时，需要保存下列信息：

- 数据类型。
- 数组维数。
- 与数组相关的数据(尺寸和数据)。
- 如果是数值对象，则保存数据对象是实数还是虚数。
- 如果是稀疏矩阵，则保存矩阵中非零元素的个数和索引。
- 如果是结构和对象，则保存字段的数量和相应的名称。

在图 2-1 中显示了 mxArray 保存双精度矩阵的情况。

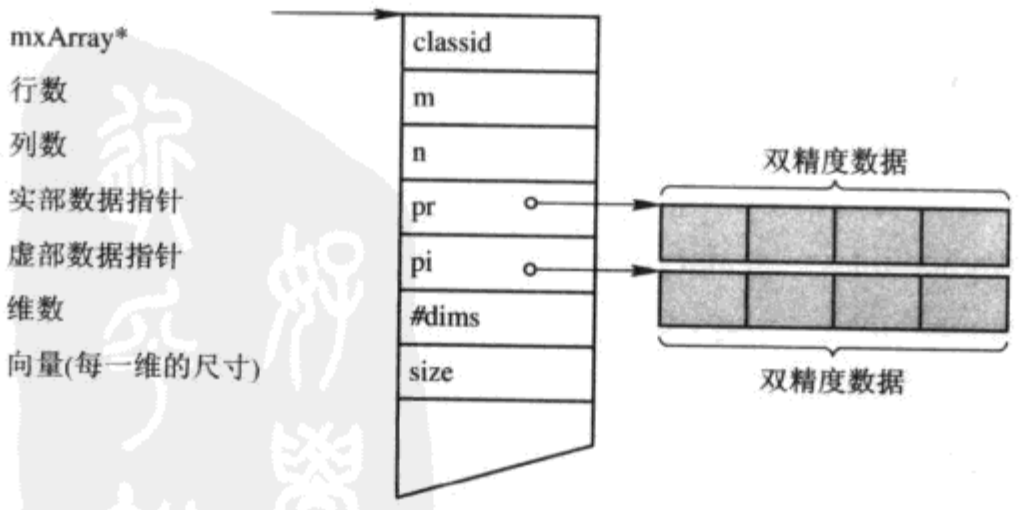


图 2-1　mxArray 保存双精度矩阵

从图 2-1 可以看出，mxArray 将 MATLAB 数据的信息分别保存在不同的字段里，例如 m 和 n 分别表示二维矩阵的行数和列数；对于多维数组，m 是矩阵的行数，而 n 是数组中每一个矩阵的列数、页面数和其它数据的乘积。维数保存着数据对象实际的维数，而 size 字段存储着每个维存储数据元素的个数。pr 为实部数据的指针，pi 是虚部数据的指针，如果没有虚部数据，则 pi 为 NULL。但是，在 mxArray 数据结构的定义中是不能直接找到这些字段的对应关系的，如果需要操作该数据对象，则需要使用专门的函数。

一般的情况下，直接操作 mxArray 对象比较困难，而且在使用 mxArray 数据对象时，都必须使用 mxArray 数据类型的指针来完成工作。为了便于创建和访问 mxArray 数据对象，MATLAB 提供了相应的函数，这些函数就是 MATLAB 提供的 mx 函数，该函数具有 C 语言的形式和 Fortran 语言的形式，利用这些函数不需要直接对 mxArray 结构对象的字段进行赋值或者读取了。关于 C 语言的 mx 函数创建数据对象的方法将在下一小节进行介绍，而详细的 C 语言和 Fortran 语言 mx 函数的解释请参阅 MATLAB 的帮助文档。

在进行外部接口编程时，获取数据类型信息是经常需要的一种操作。在 mxArray 结构中同样也保存着这样的信息，这类信息是通过 mxClassID 的枚举类型变量来保存的，该枚举类型定义如下：

```
typedef enum {
    mxUNKNOWN_CLASS = 0,
    mxCELL_CLASS,
    mxSTRUCT_CLASS,
    mxLOGICAL_CLASS,
    mxCHAR_CLASS,
    mxSPARSE_CLASS,          /* OBSOLETE! DO NOT USE */
    mxDOUBLE_CLASS,
    mxSINGLE_CLASS,
    mxINT8_CLASS,
    mxUINT8_CLASS,
    mxINT16_CLASS,
    mxUINT16_CLASS,
    mxINT32_CLASS,
    mxUINT32_CLASS,
    mxINT64_CLASS,       /* place holder - future enhancements */
    mxUINT64_CLASS,          /* place holder - future enhancements */
    mxFUNCTION_CLASS,
    mxOPAQUE_CLASS,
    mxOBJECT_CLASS
} mxClassID;
```

在 MEX 文件中进行数据类型判断时，将返回 mxClassID 类型的变量。此外，针对复数和实数也定义了一个枚举类型，该枚举类型的变量将在 C 语言应用程序中加以定义。它的定义如下：

```
typedef enum {
    mxREAL,
    mxCOMPLEX
} mxComplexity;
```

在 C 语言中创建 mxArray 数值类型对象需要指定数据是复数类型还是实数类型。

为了能够保存字符串类型的变量，在 Matrix.h 文件中还专门定义了相应的数据类型——mxChar。mxChar 是 mxArray 存储字符类型数据时使用的变量类型，该数据类型使用了 16 位的无符号整数，定义如下：

```
typedef uint16_T mxChar;
```

这里 unit16_T 是 MATLAB 自定义的数据类型符号，相当于 C 语言中 unsigned int 的定义。

为了便于用户学习外部接口编程，MATLAB 也提供了相应的示例。在 Windows 平台下，这些示例都保存在%MATLABROOT\extern\examples 目录中，在 mex 子目录下都是相应的 MEX 函数文件例子，其中有一个实例文件名为 explore.dll，它是一个 C 语言的 MEX 函数文件，功能是显示 MATLAB 数据对象的维数、尺寸和类型信息，将该文件拷贝到当前的工作路径下，然后在 MATLAB 中运行下面的指令：

```
>> x = 3;
>> explore(x)
-----------------------------------------
Name: prhs[0]
Dimensions: 1x1
Class Name: double
-----------------------------------------
    (1,1) = 3
```

请用户在自己的 MATLAB 中运行下面的指令：

```
>> explore ([1 2 3 4 5])
>> explore 1 2 3 4 5
>> explore ({1 2 3 4 5})
>> explore (int8([1 2 3 4 5]))
>> explore {1 2 3 4 5}
>> explore (sparse(eye(5)))
>> explore (struct('name','Joe Jones','ext', 7332))
>> explore (1, 2, 3, 4, 5)
```

可查看 MATLAB 相应的输出。

提示：

在相同的子目录下，explore.c 文件为该 MEX 文件的源代码，在本小节还暂时不需要关心代码的内容，等学习完第 3 章之后，有兴趣的读者可以阅读该代码。

2.2.2　外部接口函数

前一小节曾经提及为了完成创建、访问 mxArray 数据对象的操作，MATLAB 提供了相应的函数，这些函数是 mx 函数。mx 函数是 MATLAB 外部接口函数的一种，针对不同外部接口应用，MATLAB 提供了不同类型的函数，其中 mx 是这些函数共有的前缀。在表 2-2 中对 C 语言和 Fortran 语言外部接口应用程序中常用的函数进行了分类总结。

表 2-2　MATLAB 外部接口函数类型

函数类型	说　明
mat 函数	在 C 语言或者 Fortran 语言应用程序中完成 MAT 文件读写的函数，此类函数具有 mat 前缀
引擎函数	在 C 语言或者 Fortran 语言应用程序中完成引擎操作的函数，此类函数具有 eng 前缀
mex 函数	在 C 语言或者 Fortran 语言 MEX 函数源文件中完成系统操作的函数，此类函数具有 mex 前缀
mx 函数	在各种 C 语言或者 Fortran 语言外部接口函数中用来操作 mxArray 数据对象的函数，此类函数具有 mx 前缀

此外还有进行 Java 对象操作的函数和加载动态链接库的函数，这两种函数没有特别的前缀，分别用来操作不同的对象。

MATLAB 的外部接口应用程序的主要工作就是这些函数的使用方法，所以在有些英文文档中将 MATLAB 外部接口称之为 MATLAB API，表明了外部接口应用的基本范畴——接口函数的使用。本书主要就是介绍这些函数的基本使用方法，而掌握了这些 API 函数的使用方法，就基本掌握了外部接口的使用方法。

在上述函数中，数量最多也是相对最重要的函数就是 mx 函数，因为该函数不仅可以用于相应的外部接口应用程序中，而且还可以应用于 MATLAB Compiler 打包的 C 函数中。所以，本书重点、详细地解释了所有 mx 函数的使用方法，请读者注意查阅本书各章对 mx 函数的应用实例，以及 MATLAB 帮助文档中对 mx 函数的解释。


提示：

MATLAB 的外部接口函数在相应的 MATLAB 帮助文档中有详尽的说明，而这些函数的使用方法将在本书相应的章节中通过具体的示例进行讲解。此外进行 COM 应用、DDE 操作的函数将在《MATLAB 应用程序集成与发布》一书中详细讲述。

2.3　mx 函数应用

如前所述，mx 函数是所有外部接口函数中最重要的一种，所以本小节将详细介绍 mx 函数操作不同类别 MATLAB 数据的方法，作为学习后面外部接口应用的基础，请读者仔细阅读本小节，掌握 MATLAB 数据在高级语言中的表示方法。

 注意：

本小节以 MATLAB 数据 C 语言的实现为例子说明函数的使用，对于 Fortran 语言的应用，由于函数参数的相似性，这里就不进行详细的介绍了。相应的函数将在第 4 章中结合具体的应用加以讲解，函数的解释说明请参阅 MATLAB 的帮助文档。需要说明的是，本小节没有涵盖所有 mx 函数。

2.3.1 数值矩阵

MATLAB 中基本的数据类型有两种，一种是双精度数据，另外一种是字符串，其中双精度数据可以是标量、向量或者矩阵，也可以是多维的数组。MATLAB 都将其看作矩阵或者数组，又由于这些数组或者矩阵的元素都是数字，因此又将其称之为数值矩阵或者数值数组。本小节将介绍外部接口编程中通过 mx 函数创建使用数值矩阵的方法。

1. 标量

所谓标量，就是具有一个元素的矩阵，也就是 1×1 的矩阵。利用 mx 函数创建标量的方法非常简单，只需要用函数 mxCreateDoubleScalar 就可以了，该函数的 C 语言定义如下：

```
mxArray *mxCreateDoubleScalar(double value);
```

函数的输入参数是双精度类型的数据，可以是常量或者变量，而输出参数是 mxArray 数据结构的对象。使用该函数的方法参见下面的代码片段：

```
double realdata = 1.0;
mxArray *Data;
/* 创建 mxArray 数据对象-> 双精度标量 */
Data = mxCreateDoubleScalar(realdata);
```

 注意：

在早期的版本中创建双精度类型标量可以使用函数 mxCreateScalarDouble，该函数在 MATLAB 6.5 中依然可以使用，但是在未来版本中，该函数将不再存在。所以推荐读者在编写程序时，使用函数 mxCreateDoubleScalar。

如果需要创建其它数据类型的标量，则需要使用创建数值数组的函数来创建。使用标量数据和使用其它数据类型数值矩阵的函数都一样，访问数据的函数在后面详细讲述。

2. 向量和矩阵

由于在 C 语言中向量和矩阵都可以看作为二维数组，不过向量是 1×n 或者 n×1 的二维数组(矩阵)，所以在 C 语言中创建向量和矩阵都是使用同一个函数。

创建双精度类型矩阵可以使用 mxCreateDoubleMatrix 函数，该函数的定义如下：

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

该函数的输入参数是矩阵的行数 m 和列数 n，以及数据是复数或者实数的标志。该函数若成功执行，则返回变量为 mxArray 数据结构的对象指针。若不能成功执行，则在独立可执行应用程序中返回 NULL，在 MEX 函数中返回 MATLAB 命令行窗口。通过该函数创建的双精度类型矩阵还没有实际的元素，而矩阵具体的元素的创建还需要通过另外的操作

才能完成。创建双精度类型矩阵的完整过程参阅下面的代码片段：

```
double realdata[]={1,2,3,4},imagdata[]={1,2,3,4};
double *pr, *pi;
mxArray *Data;
int flag;

/* 创建 mxArray 数据对象-> 2×2 复数矩阵 */
Data = mxCreateDoubleMatrix(2,2,mxCOMPLEX);
/*获取双精度类型矩阵的实部和虚部数据指针*/
pr = mxGetPr(Data);
pi = mxGetPi(Data);
/*复制数据完成赋值*/
memcpy(pr,realdata,4*sizeof(double));
memcpy(pi,imagdata,4*sizeof(double));
```

在上面的代码片段中完成了双精度类型矩阵的创建，其中使用 mxGetPr 函数和 mxGetPi 函数获取了矩阵实部和虚部的指针，然后通过 memcpy 函数通过复制内存数据的方法完成整个矩阵数据的赋值。上述 mxArray 数据类型对象创建的过程完整地说明了在 C 语言中创建 MATLAB 数据对象的完整过程：首先使用 mx 函数创建指针完成 mxArray 数据类型指针的内存分配；然后通过 mxGetPr 和 mxGetPi 函数获取 mxArray 数据类型的实部数据指针和虚部数据指针；最后是用内存复制的方法完成数据的赋值。而操作 mxArray 类型数据指针的过程是通过 mxGetPr 和 mxGetPi 函数获取 mxArray 数据对象的指针，然后通过内存复制的方法将具体的数据赋值给相应的 C 语言双精度类型变量。

和创建标量数据类似，如果需要创建其它数据类型的矩阵变量，则需要通过创建数值数组的函数完成创建。

3. 任意数值数组

众所周知，除了双精度类型，MATLAB 还支持整数类型、单精度类型等其它数值类型，也可以创建三维甚至更多维的数组。在外部接口应用中，通过不同函数可以完成同样的操作。如果需要创建任意类型数据的矩阵可以使用 mxCreateNumericMatrix 函数，该函数的定义如下：

```
mxArray *mxCreateNumericMatrix(int m, int n, mxClassID class, mxComplexity
    ComplexFlag);
```

该函数的输入参数是矩阵的行数 m 和列数 n、矩阵的数据类型 class 以及实数或者复数的标志。这里定义的变量的数据类型是在 2.2.1 小节介绍的 mxClassID 枚举变量，返回的参数同样为 mxArray 数据类型的指针。

下面的代码片段说明了创建 32 位整数类型复数矩阵的方法：

```
/* 实部数据和虚部数据 */
long realdata [] = {1,2};
long imagdata [] = {1,2};
```

```
/* 实部虚部数据的指针 */
long *pr,*pi;
mxArray *Data;

/* 创建 mxArray 数据对象-> 双精度标量 */
Data = mxCreateNumericMatrix(1,2,mxINT32_CLASS,Data);
/* 获取实部数据和虚部数据的指针 */
pr = mxGetPr(Data);
pi = mxGetPi(Data);
/* 通过内存复制的方法完成赋值 */
memcpy(pr,realdata,2*sizeof(long));
memcpy(pi,imagdata,2*sizeof(long));
```

上述代码片段创建的数据为

```
ans =
    1.0000 + 1.0000i     2.0000 + 2.0000i
```

其数据类型为

```
Name          Size                    Bytes    Class
  ans         1x2                        16    int32 array (complex)
Grand total is 2 elements using 16 bytes
```

注意：

在一般情况下，设置非双精度类型的数值数组数据时，可以使用 mxSetData 函数，该函数详细的解释请参阅 MATLAB 的帮助文档。

如果需要创建任意类型数据的多维数组可以使用 mxCreateNumericArray 函数，该函数的定义如下：

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

该函数的输入参数分别为数组的维数 ndim、每一维的尺寸*dims、数组的数据类型 class 以及实数或者复数的标志，其中每一维的尺寸需要保存在一个数组中 ndims，返回的参数为 mxArray 数据类型的指针。

使用 mxCreateNumericArray 函数创建多维数组的过程参见下面的代码片段：

```
/* 数组的实际数据 */
double realdata [] = {0,1,2,3,4,5,6,7,8,9,10,11};
/* 多维数组的每一维尺寸 */
int ndims [] = {2,2,3};
double *pr;
mxArray *Data;

/* 创建 mxArray 数据对象-> 双精度三维数组 */
```

```
Data = mxCreateNumericArray(3,ndims,mxDOUBLE_CLASS,mxREAL);
/* 获取数组的实部数据指针 */
pr = mxGetPr(Data);
/* 通过内存赋值的方法完成数据的赋值 */
memcpy(pr,realdata,12*sizeof(double));
```

上述代码片段创建的多维数组是

```
ans(:,:,1) =
     0     2
     1     3
ans(:,:,2) =
     4     6
     5     7
ans(:,:,3) =
     8    10
     9    11
```

这里主要需要说明一点，C 语言中数组的序列是以行元素优先的，而 MATLAB 的数组是以列元素优先的，所以在上述代码创建的数组中，realdata 数组数据出现在 MATLAB 的顺序为列元素排列。另外，C 语言数组的起始元素序号为 0，而 MATLAB 数组元素起始序号为 1。

在创建多维数据的时候需要注意，当数组数据的个数不满足实际多维数组的个数时，不足的数据将自动使用数据 0 来初始化。

2.3.2　字符串

MATLAB 另外一种的基本数据类型就是字符串类型。同 C 语言不同的是，MATLAB 中只有字符串一种数据类型来表示文本信息，而 C 语言中有字符类型和字符串类型两种不同的数据表示文本信息。

在外部接口应用程序中创建 mxArray 数据类型对象表示字符串数据可以通过不同的函数来完成，其中，将 C 语言的字符串转变为 MATLAB 字符串最简单的函数是 mxCreateString 函数，它的定义如下：

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

该函数的输入参数是字符串的内容，输入参数是 C 语言的字符串常量，而输出参数就是 mxArray 数据类型的对象，该对象就是 MATLAB 的字符串。使用 mxCreateString 函数创建字符串的方法参见下面的代码片段：

```
mxArray *Data;
/* 创建 mxArray 数据对象-> 字符串 */
Data = mxCreateString("Isn't MATLAB Great?");
```

通过函数 mxCreateString 创建的字符串数组是一个行向量，向量中包含的元素是字符类型数据。

MATLAB 还提供了其它的函数用来创建字符数组，这里的字符数组不仅可以是向量，

也可以是矩阵，甚至多维数组，这时可以使用 mxCreateCharArray 函数，该函数的定义如下：

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

该函数的输入参数是数组的维数 ndim 和每一维的尺寸*dims。数组具体的数据需要通过 mx 函数内存复制的方法完成赋值。使用该函数创建字符数组的方法参见下面的代码片段：

```
unsigned short data [] = {65,66,67,68,69,70,71,72,73,74,75,76};
int ndims [] = {2,2,3};
unsigned short *pr;
mxArray *Data;

/* 创建 mxArray 数据对象-> 字符类型多维数组 */
Data = mxCreateCharArray(3,ndims);
/*获取数据的指针*/
pr = mxGetPr(Data);
/* 通过内存赋值的方法完成数据的赋值 */
memcpy(pr,data,12*sizeof(short));
```

从上述的代码片段可以看出，创建字符类型数组的过程和创建普通的数值类型数组没有什么区别，这里得到的字符类型数组为

```
ans(:,:,1) =
AC
BD
ans(:,:,2) =
EG
FH
ans(:,:,3) =
IK
JL
```

 **注意：**

在 MATLAB 中每个字符数组元素占用两个字节的内存空间，所以在 C 语言代码中，使用 unsigned shot 类型的数据为字符数组进行赋值。数组元素的顺序是以列元素优先的。

如果创建字符串矩阵，还可以使用函数 mxCreateCharMatrixFormString，该函数可以用来创建二维的字符串矩阵，它的定义如下：

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

该函数的输入参数为字符串矩阵的行数以及矩阵中的内容，该函数能够自动判断输入的字符串内容的长度，并且使用所有字符串中最长的长度来分配 MATLAB 字符串占用的内存。使用该函数创建字符串矩阵的方法参考下面的代码片段：

```
/*字符串矩阵的内容
char *string[] = {"MATLAB","外部接口","编程"};
```

```
mxArray *Data;
/* 创建 mxArray 数据对象-> 字符串矩阵 */
Data = mxCreateCharMatrixFromStrings(3,string);
```

上述代码片段创建的字符串矩阵为

```
ans =
MATLAB
外部接口
编程
>> whos
  Name        Size                     Bytes  Class
  ans         3x6                         36  char array
```

如果需要获取字符串类型的 mxArray 数据对象中包含的内容，则需要使用 mxGetChars 或者 mxGetString 函数，关于这两个函数具体的解释请参阅 MATLAB 的帮助文档，具体的使用过程将在第 3 章中进行演示。

2.3.3　逻辑数组

逻辑数组在 MATLAB 中是一类特殊的数据类型，一般进行任何关系运算得到的结果就是逻辑类型的数据。在其它的高级语言中，一般将这种数据称之为布尔类型变量，这种变量只有两种可能的取值。在 MATLAB 中，用 1 代表逻辑真，用 0 表示逻辑假。逻辑数组每一个元素占用一个字节的内存。

外部接口应用程序同样提供了不同的函数来创建逻辑数组。和创建数值矩阵类似，逻辑数组也分为创建标量、矩阵和多维数组的函数。

创建逻辑类型的标量需要使用函数 mxCreateLogicalScalar，该函数的定义如下：

```
mxArray *mxCreateLogicalScalar(mxLOGICAL value);
```

该函数输入参数的类型为 mxLOGICAL，这种数据类型就是 C 语言中表示布尔类型的数据类型，一般为 bool。该函数的输出参数是 mxArray 数据类型的对象，它表示逻辑量。

如果创建逻辑类型的矩阵，则需要使用函数 mxCreateLogicalMatrix，该函数的定义如下：

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

该函数的输入参数是逻辑矩阵的行数 m 和列数 n，函数的输出参数是 mxArray 数据类型的对象，它表示逻辑量矩阵。

如果创建逻辑类型的多维数组，则可以使用函数 mxCreateLogicalArray，该函数的定义如下：

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

该函数的输入参数是数组的维数 ndim 和每一维的尺寸*dims，函数的输出参数是 mxArray 数据类型的对象，它表示逻辑数组。

下面的代码片段说明了使用 mxCreateLogicalArray 函数创建逻辑数组的过程：

```
/*逻辑数组的数据*/
bool data [] = {0,1,0,1,0,1,0,1,0,1,0,1};
```

```
/*数组的尺寸*/
int ndims [] = {3,4};
/*数据的指针*/
mxLogical *pr;
mxArray *Data;
/* 创建 mxArray 数据对象-> 逻辑类型数组 */
Data = mxCreateLogicalArray(2,ndims);
/*获取数据的指针*/
pr = mxGetLogicals(Data);
/* 通过内存赋值的方法完成数据的赋值 */
memcpy(pr,data,12*sizeof(mxLogical));
```

在创建逻辑数组时，不能使用 mxGetPr 函数来获取数据的指针，而是使用 mxGetLogicals 函数获取逻辑数组数据的指针，该函数的定义如下：

```
mxLogical *mxGetLogicals(const mxArray *array_ptr);
```

函数的输入参数是表示逻辑数组的 mxArray 数据类型对象，返回的输出参数是 mxLogical 类型的数据指针，为 mxArray 数据类型对象中所包含的数据首地址。

上述代码片段得到的逻辑类型矩阵如下：

```
ans =
     0     1     0     1
     1     0     1     0
     0     1     0     1
>> whos
  Name        Size                    Bytes  Class
  ans         3x4                        12  logical array
Grand total is 12 elements using 12 bytes
```

在 MATLAB 中存在大量的以 is 为前缀的函数，这些函数主要是完成逻辑判断操作的函数，函数返回的变量类型也是逻辑类型数组。在外部接口编程中同样也存在相应的函数可以完成逻辑判断操作。这些函数就是以 mxIs 为前缀的 mx 函数，这些函数多数是用来完成数据类型的判断，但是返回的变量不是逻辑类型的 mxArray 数据对象，而是 bool 类型的数据，也就是 mxLogical 类型的数据。

这里就不给出创建逻辑类型标量和普通矩阵的实例代码片段了，关于 mxIs 函数的相关内容请参阅 MATLAB 的帮助文档。

2.3.4 元胞数组

元胞数组是 MATLAB 特有的一种数据类型，用户可以将元胞数组看作成为广义矩阵，组成元胞数组的元素可以是各种类型的 MATLAB 数据，既可以是标量、向量，也可以是矩阵甚至是多维数组。在外部接口应用程序中通过 mx 函数创建元胞数组的过程比创建普通的数据要略微繁琐：首先创建元胞数组的各个元胞数据；然后创建元胞类型的数据对象指针；最后将元胞数据赋值给元胞数组的各个元素。

创建其它类型数据的函数在其它小节都已经介绍了，而创建元胞数组的函数有两个，分别用来创建元胞类型的矩阵和元胞类型的数组。

创建元胞类型矩阵的函数定义如下：

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

该函数的输入参数是矩阵的行数 m 和列数 n，返回的输出参数是元胞矩阵的 mxArray 数据类型对象指针。

创建元胞类型数组的函数定义如下：

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

该函数的输入参数是数组的维数 ndim 以及每一维的尺寸*dims，返回的输出参数是元胞数组的 mxArray 数据类型对象指针。

创建了元胞数组的 mxArray 数据类型对象指针后，需要使用 mxSetCell 函数将已经创建好的元胞数据赋值给具体的元胞数组。mxSetCell 函数的定义如下：

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

该函数没有返回的输出参数，输入参数分别为元胞数组的 mxArray 数据类型对象指针 array_ptr、元胞元素的索引 index 和元胞数据的 mxArray 数据类型对象指针 value。

下面的代码片段说明了创建元胞数组的过程：

```
/* 元胞数组的指针 */
mxArray *Data;
/*实际数据的指针 */
mxArray *string;
mxArray *multi;
mxArray *scalar;
int ndims [] = {2,2,2};
double realdata [] = {1,2,3,4,5,6,7,8};
double *pr;
/* 创建字符串类型变量*/
string = mxCreateString("Isn't MATLAB Great?");
/*创建多维数值数组*/
multi = mxCreateNumericArray(3,ndims,mxDOUBLE_CLASS,mxREAL);
pr = mxGetPr(multi);
memcpy(pr,realdata,8*sizeof(double));
/*创建标量*/
scalar = mxCreateDoubleScalar(10);
/*创建元胞数组*/
Data = mxCreateCellMatrix(1,3);
/*使用 mxSetCell 函数设置元胞的数据*/
mxSetCell(Data,0,string);
mxSetCell(Data,1,multi);
mxSetCell(Data,2,scalar);
```

上述代码片段创建的元胞数组为

```
ans =
    'Isn't MATLAB Great?'    [2x2x2 double]    [10]
>> whos
  Name        Size                    Bytes  Class
  ans         1x3                       290  cell array
Grand total is 31 elements using 290 bytes
```

需要注意的是，在 C 语言中数组的起始元素序号为 0，所以，使用 mxSetCell 函数设置元胞数组数据的时候，第一个元胞数据应该用数字 0 作为序号。

如果需要从已经存在的元胞数组中获取元胞数据，则需要使用 mxGetCell 函数，该函数的具体定义和使用方法在这里就不再赘述了，请读者查阅 MATLAB 的帮助文档。

2.3.5 结构数组

结构类型的数据和元胞类型的数据非常类似，在 MATLAB 中还有专门的函数完成两者之间的转换。结构类型的数据是由若干记录组成的，每一个记录保存在结构的字段中，结构保存的数据可以是任意类型的 MATLAB 数据。在外部接口应用程序中，创建结构数组的过程和创建元胞数组的过程也非常类似，基本过程如下：

- 首先，准备不同字段的数据。
- 其次，创建结构类型的数据对象指针。
- 最后，使用 mx 函数完成结构数据的赋值。

创建其它类型数据的函数在其它小节都已经介绍了，而创建结构数组的函数有两个，分别用来创建结构类型的矩阵和结构类型的数组。

创建结构类型矩阵的函数定义如下：

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

创建结构矩阵的函数输入参数比较多，分别为矩阵的行数 m 和列数 n，字段的个数 nfields，字段名称的字符串数组 field_names。函数的输出参数是 mxArray 数据类型的数据对象指针表示一个结构矩阵。

创建结构类型数组的函数定义如下：

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

创建结构数组函数的输入参数分别为数组的维数 ndim，数组每一维的尺寸*dims，字段的个数 nfileds，字段名称的字符串数组 field_names。函数的输出参数是 mxArray 数据类型的数据对象指针，表示一个结构类型的数组。

```
/* 结构类型数据的指针 */
mxArray *Data;
/* 结构中实际数据对象 */
mxArray *string;
mxArray *scalar;
```

```
int field_num = 0;
/* 字段名称 */
char Fields[] ={"String","Scalar"};
/*创建结构的数据 */
string = mxCreateString("Isn't MATLAB Great?");
scalar = mxCreateDoubleScalar(10);
/* 创建结构矩阵 */
Data = mxCreateStructMatrix(1,1,2,Fields); /* 创建结构 */
/*设置结构的字段数据 */
field_num = mxGetFieldNumber(Data, Fileds[0]);
mxSetFieldByNumber(Data,0, field_num, string);
/*设置结构的字段数据 */
field_num = mxGetFieldNumber(Data, Fileds[1]);
mxSetFieldByNumber(Data,0, field_num, scalar);
```

上述代码创建的结构矩阵如下：

```
ans =
    String: 'Isn't MATLAB Great?'
    Scalar: 10
```

在上面的代码片段中，首先使用 mxGetFieldNumber 函数获取字段序号，然后利用 mxSetFiledByNumber 函数设置了不同字段的数据。其实上面的代码中的操作只要使用一个函数就能够完成，这个函数就是 mxSetFiled，例如下面的代码：

```
mxSetField(Data, 0, Fields[0],string); /*设置字段的数值*/
mxSetField(Data, 0, Fields[1],scalar); /*设置字段的数值*/
```

上面两行代码同样完成了设置结构数据的工作。

在使用 mxSetField 函数和 mxSetFiledByNumber 函数的时候需要注意，函数的第二个参数是整个结构数组的元素序号，所以在上面的代码片段中，这个参数都被设置成为 0，表示结构数组的第一个元素。

有关 mxSetFiled 函数和 mxSetFieldByNumber 函数的详细解释请参阅 MATLAB 的帮助文档。

2.3.6　稀疏矩阵

稀疏矩阵是一类特殊的数值矩阵，这些矩阵中多数数据都是 0，只有一小部分的元素为非零数据。在一般情况下，矩阵中的零元素没有什么特殊意义，但是，如果编程中使用矩阵的完整表示，则这些零元素将会占用很多内存空间，所以，在数据结构上提出了保存稀疏矩阵的算法，做到既能够有效地完成稀疏矩阵的存储和运算，又能够达到节约内存空间的目的。

在 MATLAB 中保存稀疏矩阵是通过以下三个数组来完成的：

- 第一个数组保存所有的非零元素，一般该数组为双精度类型，数组的长度为 nzmax。
- 第二个数组为整数类型数组，保存每个非零元素的行序号索引。

● 第三个数组是长度为 N+1 的整数类型数组，其中 N 为矩阵的列数。该数组保存非零元素的列索引信息。该数组最后一个元素是稀疏矩阵的非零元素个数。

有关稀疏矩阵的存储算法可以参阅有关的数据结构数据或者 MATLAB 的帮助文档。

在外部接口程序中，可以使用不同的函数创建不同数据类型的稀疏矩阵，其中最常用的就是创建双精度类型的稀疏矩阵，其函数为 mxCreateSparse，它的定义如下：

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

该函数的输入参数分别为矩阵的行数 m 和列数 n，矩阵中非零元素的个数 nzmax，以及矩阵数据的复数或者实数标志。函数的输出参数是表示稀疏矩阵的 mxArray 数据类型指针。

在创建了稀疏矩阵数据指针之后，需要通过其它的函数向数据指针进行赋值，从而完成矩阵数据的创建，这些函数分别如下：

- mxGetPr 和 mxGetPi，获取矩阵数据的指针。
- mxGetIr 获取表示稀疏矩阵的第二个数组的指针。
- mxGetJc 获取表示稀疏矩阵的第三个数组的指针。

然后就可以通过拷贝内存的方法完成数据的赋值。

创建双精度类型稀疏矩阵的示例代码片段如下：

```
double data[]    = { 3, 1, 10, -7, 2, -2 }; /* 稀疏矩阵数据 */
int      irdata[] = { 0, 3, 0, 1, 2, 3 }; /* ir 向量 */
int      jcdata[] = { 0, 2, 2, 5, 6 }; /* jc 向量 */
mxArray *Data
/*数据的指针*/
double *pr;
int      *ir, *jc;
/* 创建稀疏矩阵-> 4×4 稀疏矩阵，矩阵中为 6 个数据*/
Data = mxCreateSparse( 4, 4, 6, mxREAL );
/*获取实际数据的指针 */
pr = mxGetPr( Data );
/* ir 向量的指针 */
ir = mxGetIr( Data );
/* jc 向量的指针 */
jc = mxGetJc( Data );
   /*复制数据 */
memcpy( pr, data, 6*sizeof(double) );
/*复制 ir 向量的数据 */
memcpy( ir, irdata, 6*sizeof(int) );
/*复制 jc 向量的数据*/
memcpy( jc, jcdata, 5*sizeof(int) );
```

上述代码片段创建的稀疏矩阵如下：

```
ans =
```

```
    (1,1)        3
    (4,1)        1
    (1,3)       10
    (2,3)       -7
    (3,3)        2
    (4,4)       -2
>> whos
  Name        Size                    Bytes  Class
  ans         4x4                        92  double array (sparse)
Grand total is 6 elements using 92 bytes
```

与之对应的满阵为

```
>> full(ans)
ans =
     3     0    10     0
     0     0    -7     0
     0     0     2     0
     1     0     0    -2
```

在外部接口函数中，还存在一个函数用来创建逻辑类型的稀疏矩阵，该函数的定义如下：

```
mxArray *mxCreateSparseLogicalMatrix(int m, int n, int nzmax)
```

该函数的输入、输出参数和前面介绍的创建双精度类型稀疏矩阵函数的输入、输出参数的意义是一致的，这里就不再赘述了。下面的代码片段说明了创建逻辑类型稀疏矩阵的过程。

```
mxLogical data[]     = { 1, 1, 1}; /*  稀疏矩阵数据  */
int       irdata[] = { 1,3,0 }; /* ir  向量  */
int       jcdata[] = { 0, 1, 2, 3 }; /* jc  向量  */
mxArray *Data;
mxLogical *pr;
int       *ir , *jc;

/*  创建稀疏矩阵-> 4×3 矩阵*/
Data = mxCreateSparseLogicalMatrix(4,3,3);

pr = mxGetLogicals( Data ); /*获取实际数据的指针  */
ir = mxGetIr( Data ); /* ir  向量的指针  */
jc = mxGetJc( Data ); /* jc  向量的指针  */

/*复制数据  */
memcpy( pr, data, 3*sizeof(mxLogical) );
```

```
/*复制 ir 向量的数据 */
memcpy( ir, irdata, 3*sizeof(int) );
/*复制 jc 向量的数据*/
memcpy( jc, jcdata, 4*sizeof(int) );
```

请读者参照前面创建双精度类型稀疏矩阵的代码自学这部分代码，这部分代码创建的稀疏矩阵为

```
ans =
    (2,1)        1
    (4,2)        1
    (1,3)        1
>> whos
  Name        Size                    Bytes  Class
  ans         4x3                        31  logical array (sparse)
>> full(ans)
ans =
     0     0     1
     1     0     0
     0     0     0
     0     1     0
```

从实际的效果上看，上面例子得到的稀疏矩阵并没有节约内存，所以逻辑类型的稀疏矩阵要酌情使用。此外，在创建稀疏矩阵的时候，还可以直接使用 mxSetIr、mxSetJc、mxSetNzmax 等函数直接完成稀疏矩阵的属性设置，这些函数的说明请参阅 MATLAB 的帮助文档。

2.3.7 内存管理操作

由于所有使用 mxArray 数据类型对象的操作都使用了指针操作，所以内存的分配和管理在外部接口应用程序中显得尤为重要。特别是 C 语言的应用程序不像 MATLAB 的 M 语言应用程序，在 M 语言应用程序中，程序员可以不关心内存的分配和释放，这些工作都由 MATLAB 的解释器来完成，但是在 C 语言程序中，内存的分配和释放就需要程序员编写代码来完成操作了。

在 mx 函数中，同样包含了若干数据内存分配和释放的函数。在前面几个小节中介绍了很多 mxCreate 函数，分别用来创建不同类型的 mxArray 数据类型对象，这些数据对象分别表示不同类型的 MATLAB 数据。其实这些 mxCreate 函数是在 C 语言的数据内存中分配工作的，当内存分配成功后，则返回相应的指针(内存地址)，否则出现不同类型的错误提示信息。

如果在程序中分配了若干内存，当不使用这些已经分配的内存时，程序员应该养成一个良好的编程习惯，将这些内存空间及时地释放。在 mx 函数中用于释放内存的函数为 mxDestroyArray，该函数的定义如下：

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

该函数的输入参数是需要释放的数据指针。该函数一旦操作成功，则将 array_ptr 指针指向的数据全部释放。所以请读者注意，在编写 MEX 函数文件时，如果函数内部创建的变量作为函数的输出参数，则千万不要在程序中将输出参数的内存释放了。

除了分配和释放 mxArray 数据类型的数据内存的函数以外，还存在若干用于释放其它类型的数据内存的函数，这些函数分别为 mxMalloc、mxRealloc、mxCalloc 和 mxFree 函数，这些函数类似于标准 C 语言中的 malloc、realloc、calloc 和 free 函数，使用这些函数的主要用途是避免在 MEX 文件中使用标准 C 语言的函数。不过这里需要读者注意，如果用户定义的变量使用了 mxCreate 函数来分配内存，则释放该数据内存的函数一定要使用 mxDestroyArray 函数；如果用户定义的变量使用 mxMalloc、mxReallo 或者 mxCalloc 函数来分配内存，则需要使用 mxFree 函数来进行内存的释放。例如下面的代码：

```
mxArray *data;
char *str;
data=mxCreateString("This was created using mxCreateString");
str=(char*)mxMalloc(32*sizeof(char));
str="This was created using mxMalloc";
...
mxDestroyArray(data); /* 释放使用 mxCreate 分配的内存空间*/
mxFree(str); /* 释放使用 mxMalloc 分配的内存空间*/
```

此外，在 mx 函数中还存在一些 mxSet 函数用来完成数据的赋值，例如 mxSetPr、mxSetPi、mxSetData 和 mxSetImagData 等函数，在使用这些函数时也需要注意，避免发生不必要的错误，参见下面的 MEX 函数代码片段：

```
/*创建 mxArray 数据类型指针*/
mxArray *temp = mxCreateDoubleMatrix(0,0,mxREAL);
/*实际包含的数据*/
double data[5] = {1,2,3,4,5};
/*设置矩阵的行数和列数*/
mxSetM(temp,1); mxSetN(temp,5);
/*数据的赋值*/
mxSetPr(temp, data); /* 错误 */
```

在上述的代码段中，直接使用 mxSetPr 函数是错误的，原因如下：当该 MEX 函数运行完毕退出时，MATLAB 将试图释放那些没有使用 mx 函数分配的内存空间，所以上述代码创建的 data 数据将被释放，而 mxArray 数据类型指针的数据同样也会被释放掉，这样就造成了错误。所以正确的方法是通过 memcpy 函数完成内存数据的拷贝复制，这样就不会出现因内存数据丢失而引起的不必要的错误，正确的做法是：

```
mxArray *temp = mxCreateDoubleMatrix(0,0,mxREAL);
double data[5] = {1,2,3,4,5};
mxSetM(temp,1); mxSetN(temp,5);
memcpy(mxGetPr(temp), data, 5*sizeof(double)); /* 正确 */
```

这就是为什么前面小节所有的代码示例中都通过 memcpy 函数来完成赋值的原因。

2.4 MATLAB 的环境配置

很多外部接口应用程序，包括 MEX 文件、MATLAB 计算引擎应用程序、MAT 应用程序都可以在 MATLAB 环境中完成其编译过程，不过完成编译之前需要在系统环境中进行相应的配置工作。本小节将介绍在 MATLAB 中进行环境配置的方法。

2.4.1 基本配置

在进行基本配置之前，一定要在操作系统中安装相应的 C 语言或者 Fortran 语言编译器。例如在 Windows 平台中，安装流行的 Microsoft Visual Studio，如果需要编写 Fortran 语言的应用程序，则需要安装相应的 Fortran 开发环境。不同的操作系统平台可以选择不同的语言编译器，MATLAB 支持的 C/C++、Fortran 语言编译器的详细列表可以参阅 MATLAB 的相关帮助文档。

 注意：

在 MATLAB 产品体系中不同的产品模块需要使用的 C/C++语言编译器或者 Fortran 语言编译器不尽相同，请读者仔细阅读附录 A 的内容，在自己的工作平台中选择安装正确的编译器。

在本小节以 Windows 平台为例，讲解编译器的配置过程。配置 MATLAB 编译环境的方法非常简单，在 MATLAB 的命令行窗口中键入下面的指令：

```
>> mex -setup
```

这时，在 MATLAB 命令行窗口中将出现下面的提示：

```
Please choose your compiler for building external interface (MEX) files:
Would you like mex to locate installed compilers [y]/n?
```

用户可以让 MATLAB 自己搜索当前系统平台下已经安装完毕的编译器或者开发环境，也可以自己来挑选，如果在上面的提示下，按“N”键，则由用户自己选择编译器：

```
Select a compiler:
[1] Borland C++Builder version 6.0
[2] Borland C++Builder version 5.0
[3] Borland C++Builder version 4.0
[4] Borland C++Builder version 3.0
[5] Borland C/C++ version 5.02
[6] Borland C/C++ version 5.0
[7] Borland C/C++ (free command line tools) version 5.5
[8] Compaq Visual Fortran version 6.1
[9] Compaq Visual Fortran version 6.6
[10] Digital Visual Fortran version 6.0
[11] Digital Visual Fortran version 5.0
```

[12] Lcc C version 2.4

[13] Microsoft Visual C/C++ version 7.0

[14] Microsoft Visual C/C++ version 6.0

[15] Microsoft Visual C/C++ version 5.0

[16] WATCOM C/C++ version 11

[17] WATCOM C/C++ version 10.6

[0] None

这时，在 MATLAB 的命令行窗体中显示的是当前所有 MATLAB 支持的 C/C++语言编译器或者 Fortran 语言编译器。在 Windows 平台上，MATLAB 几乎支持所有主流的 C/C++语言编译器。在这里，需要用户根据自己的判断选择一个已经安装在当前操作系统下的 C/C++语言编译器或者 Fortran 语言编译器。

如果在前一步骤直接按下回车键，则系统仅显示已经安装在当前操作系统中的编译器，例如：

Select a compiler:

[1] Lcc C version 2.4 in E:\MATLAB6P5P1\sys\lcc

[2] Microsoft Visual C/C++ version 6.0 in E:\Program Files\Microsoft Visual Studio

用户需要键入编译器列表中的序号来选择不同的编译器，例如当显示全部列表的时候，选择[14] Microsoft Visual C/C++ version 6.0 ，这时 MEX 将使用 Visual C++ 6.0 的编译器来完成应用程序的创建。

Compiler: 14

选择编译器后，将出现下面的提示信息，询问当前编译器的安装路径是否正确：

Your machine has a Microsoft Visual C/C++ compiler located at

E:\Program Files\Microsoft Visual Studio. Do you want to use this compiler [y]/n?

如果不正确，则需要用户键入“n”键，并给出正确的安装路径。接下来，MATLAB 将确认用户选择的编译器信息，请大家仔细确认，以保证 MATLAB 选择了正确的编译器。

Please verify your choices:

Compiler: Microsoft Visual C/C++ 6.0

Location: E:\Program Files\Microsoft Visual Studio

Are these correct?([y]/n):

确认之后，MATLAB 将完成系统文件的配置，提示如下：

The default options file:

"C:\Documents and Settings\Way\Application Data\MathWorks\MATLAB\R13\mexopts.bat"

is being updated from E:\MATLAB6P5P1\BIN\WIN32\mexopts\msvc60opts.bat...

Installing the MATLAB Visual Studio add-in ...

Updated E:\Program Files\Microsoft Visual Studio\common\msdev98\template\

```
MATLABWizard.awx
    from E:\MATLAB6P5P1\BIN\WIN32\MATLABWizard.awx
  Updated E:\Program Files\Microsoft Visual Studio\common\msdev98\template\
MATLABWizard.hlp
    from E:\MATLAB6P5P1\BIN\WIN32\MATLABWizard.hlp
  Updated E:\Program Files\Microsoft Visual Studio\common\msdev98\addins
MATLABAddin.dll
    from E:\MATLAB6P5P1\BIN\WIN32\MATLABAddin.dll
  Merged E:\MATLAB6P5P1\BIN\WIN32\usertype.dat
    with E:\Program Files\Microsoft Visual Studio\common\msdev98\bin\usertype.dat
```

此时 MATLAB 将在 Visual Studio 中安装 MATLAB 的插件，利用 MATLAB 的插件可以在 Visual Studio 的集成开发环境(IDE)中完成 MEX 文件的创建，并且可以进行 MEX 文件的调试。有关 Visual Studio 的 MATLAB 插件将在第 3 章中详细讲解。最后，MATLAB 将提示下列信息：

```
Note: If you want to use the MATLAB Visual Studio add-in with the MATLAB C/C++
      Compiler, you must start MATLAB and run the following commands:

      cd(prefdir);
      mccsavepath;

      (You only have to do this configuration step once.)
```

上面的提示信息表示如果希望在 Visual Studio 中正确使用 MATLAB Compiler，则需要执行上述提示的两个指令，这已经超出了本书的讨论范围，有兴趣的读者可以参阅 MATLAB 的帮助文档，或者《MATLAB 应用程序集成与发布》一书中关于 MATLAB Compiler 的内容。

至此 MEX 文件和 C/C++语言编译器之间的配置工作就完成了。本小节是以 Windows 平台下的 Visual Studio 6.0 为例讲解配置过程的。不同的编译器的配置过程大体一致，请读者根据自己的喜好选择编译器类型进行安装配置。

2.4.2 选项文件

在 MATLAB 中 MEX 是怎样完成程序的编译的呢？它主要是通过若干编译选项文件来完成第三方编译器编译程序的操作的。在 Windows 操作系统中，这些编译选项文件都保存在%MATLABROOT%\bin\win32\mexopts 目录下。在该目录下面，主要发挥编译作用的是.bat 文件，并且不同的编译器针对不同的编译选项文件，而且该目录下主要存在两类选项文件，其中一类为 MEX 文件编译选项文件，文件名的特征是 xxxxxxopts.bat，其中 xxxxxx 为不同的编译器，例如 Visual C++6.0 对应的编译选项文件为 msvc60opts.bat 文件。在表 2-3 中总结了不同编译器的 MEX 选项文件。

表 2-3　MEX 选项文件

编译器	版本	文件名的特征
Borland C++ Builder	6	bcc56opts.bat
	5	bcc55opts.bat
	4	bcc54opts.bat
	3	bcc53opts.bat
Borland C/C++ Compiler	5.5	bcc55freeopts.bat
	5.2	bccopts.bat
	5	
Compaq Visual Fortran	6.6	df66opts.bat
	6.1	
Digital Visual Fortran	6.0	df60opts.bat
	5.0	df50opts.bat
Lcc-win32	2.4.1	lccopts.bat
Microsoft Visual C/C++	.NET	msvc70opts.bat
	6	msvc60opts.bat
	5	msvc50opts.bat
Watcom C/C++	11	watopts.bat
	10.6	

有关 MATLAB 产品支持的编译器以及编译器与相应产品之间的关系，请参阅本书的附录 A。

前面进行编译器配置的主要过程，就是选择与用户使用的编译器对应的 MEX 选项文件，然后将选定的选项文件以文件名 mexopts.bat 的形式保存在 MATLAB 系统路径下。在 Windows 2000/XP 环境下，MATLAB R13 的系统路径一般为 C:\Documents and Settings\%USER%\Application Data\MathWorks\MATLAB\R13，其中%USER%为当前的系统用户，例如 Administrator。

除了 MEX 选项文件以外，MAT 文件应用程序和 MATLAB 计算引擎应用程序也具有相应的选项文件，这些文件的共同特征为 xxxxxxengmatopts.bat，其中 xxxxxx 为具体的编译器名称。不同的编译器具有不同的 MAT 和计算引擎应用程序选项文件，这里就不具体列出了，请读者结合 MEX 文件的选项文件进行查看，其它不同选项文件的内容将在讲解具体应用程序的时候进行讲解。

2.5　本 章 小 结

本章首先讲解了 MATLAB 外部接口应用程序的类型，并且针对不同的外部接口应用作出了解释。然后着重介绍了 MATLAB 外部接口应用程序中常用的 mx 函数，通过大量的代码示例介绍了利用 mx 函数创建不同类型 MATLAB 数据的方法。最后，还介绍了 MATLAB

配置外部接口应用程序编译器的方法以及相应的选项文件。通过本章的学习，读者能够基本掌握 mx 函数创建不同类型 MATLAB 数据的方法和外部接口编译器的配置，这些操作都是学习 MATLAB 外部接口编程，以及 MATLAB 应用程序集成与发布 C 语言应用程序的基础。所以，读者务必对本章的内容仔细学习，这样能够在后续的学习过程中达到事半功倍的效果。

从第 3 章开始，将陆续介绍 MATLAB 外部接口应用程序的具体应用，同时，还将进一步扩展介绍部分 mx 函数的使用方法。

练　习

1. 熟悉 MATLAB 的数据类型，并尝试在 MATLAB 中完成编译器的配置。
2. MATLAB 外部接口总共有几种类型，分别可以完成什么样的功能？
3. 编写程序片段完成下面的功能：

创建无符号 16 位整数类型矩阵，矩阵的内容如下：

$$\begin{bmatrix} 1 & 0 & 3 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 2 & 0 & 1 & 0 & 2 \end{bmatrix}$$

通过 mxIsSparse、mxIsUint16 函数来判断矩阵，将判断的结果创建成逻辑类型向量。

第 3 章　创建 C 语言 MEX 文件

从本章开始介绍 MATLAB 外部接口的编程方法。首先向读者介绍的就是创建 C 语言的 MEX 文件的方法。MEX 文件是 MATLAB 外部接口中非常常用的一种外部接口应用程序，通过 MEX 文件不仅能够集成已有的 C 语言或者 Fortran 语言的代码到 MATLAB 应用中，而且还能够为 MATLAB 带来一些系统特性，例如对计算机硬件设备的操作。

 注意：

MATLAB 提供了一种高效率的快速原型编程语言——M 语言，大多数 MATLAB 应用程序应该使用这种语言来开发，一般地，除非系统需要，不要使用 MEX 文件。

本章重点内容

- MEX 源文件结构；
- MEX 函数；
- MEX 文件的调试。

3.1　MEX 文件简介

MEX 从字面上是 MATLAB 和 Executable 两个单词的缩写。一般地 MEX 文件使用 C 语言或者 Fortran 语言进行开发，通过适当的编译之后，生成的目标文件能够被 M 语言解释器调用执行。MEX 文件在使用上和 M 函数文件非常类似，但是由于 M 语言解释器解析指令具有优先级特性，所以 MEX 文件总是被优先执行。

 提示：M 语言解释器指令解析优先级

当用户在 MATLAB 命令行窗口键入一个命令或者 M 语言文件执行一条语句时，MATLAB 解释器就负责解析输入或者需要被执行的语句，并且给出相应的答案。MATLAB 解释器解析命令按照一定优先级进行：首先判断该命令是否为变量，若不是内存中的变量，判断该命令是否为 MATLAB 的内建函数；若不是内建函数，则判断该命令是否为子函数；若不是子函数，则判断该命令是否为私有函数；若不是私有函数，则判断该命令是否为 MATLAB 搜索路径中所包含的某个文件或函数；若在同一个路径下发现可以被执行的文件具有同名的三种类型：MEX 文件、P 代码文件和 M 代码文件，则优先执行 MEX 文件，其次是 P 代码文件，最低的优先级是 M 语言文件。

若需要了解具体调用的是哪一个对象，则可以使用 which 命令获取相应的信息。

在详细介绍 MEX 源文件编写方法之前，首先查看一个简单的 C 语言 MEX 源文件，通过该文件了解 MEX 文件的基本创建过程。

例 3-1 简单 MEX 文件示例——mexHelloWorld.c。

```
001    /* 必需的头文件 */
002    #include "mex.h";
003    /* MEX 函数的入口函数 */
004    void mexFunction(int nlhs, mxArray *plhs[],
005                     int nrhs, const mxArray *prhs[])
006    {
007        /*函数的内容，调用其它 C 函数 */
008        mexPrintf("Hello MATALB World!");
009    }
```

首先请读者使用任何一种能够编辑纯文本文件的编辑器，例如 Windows 平台下的记事本，UNIX 平台下的 vi 等，也可以使用 MATLAB 提供的 meditor，将上面的代码键入到文件中。注意，不要将行号(001～009)也敲进去，在这里设置行号的主要目的是为了便于讲解和分析程序。

然后将该文件保存，设置文件名称为 mexHelloWorld.c，将该文件拷贝或者保存到 MATLAB 的当前工作路径下。

接着，在 MATLAB 的命令行窗口下键入下面的指令：

```
>> mex mexHelloWorld.c↙
```

其中，“↙”表示敲回车键。这个命令行有两个部分，mex 为编译器指令，mexHelloWorld.c 就是用户创建的 C 文件。注意，在这里需要给出 C 文件的扩展名。

如果编辑的应用程序没有任何错误，则 MATLAB 命令行中不会显示任何消息；否则，在 MATLAB 命令行中将显示相应的错误消息提示。一般常见的错误是拼写错误，例如没有区分大小写字母、语句结尾没有分号“;”等。如果出现编译错误，请仔细查看键入的源程序代码，修改错误直到编译通过为止。

编译的结果是创建出 MATLAB 的 MEX 文件，可以在 MATLAB 命令行下键入 what 指令查看当前路径下是否具有 MEX 文件。

```
>> what
MEX-files in the current directory D:\Temp
mexHelloWorld
```

接着，运行创建的 MEX 文件，在 MATLAB 命令行中键入下面的指令：

```
>> mexHelloWorld↙
Hello MATALB World!
```

例 3-1 的 MEX 文件的功能就是在 MATLAB 命令行下输出一条文本信息。还可以在 MATLAB 命令行窗口中键入下面的指令：

```
>> which mexHelloWorld
D:\Temp\mexHelloWorld.dll
```

which 指令可以查看当前执行的指令或者函数具体的位置或者类型。可以发现，在 Windows 平台下 MEX 文件的扩展名称为.dll。不同操作系统平台下的 MEX 文件扩展名称不尽相同，读者可以在自己的平台下，在 MATLAB 命令行中使用 mexext 指令获取当前平台

下 MEX 文件的扩展名，在表 3-1 中总结了这些文件的扩展名类型。

表 3-1　MEX 文件的扩展名

平　台	MEX 文件扩展名
Alpha	.mexaxp
HPUX Version 11.x	.mexhpux
HP700 Version 10.20	.mexhp7
IBM RS/6000	.mexrs6
Linux	.mexglx
SGI,SGI64	.mexsg
Solaris	.mexsol
Windows	.dll

上述编译 MEX 文件的过程同样可以在操作系统的控制台下完成，例如在 Windows 平台下，在命令行提示符窗口中，直接键入 mex mexFilename.c 也可以完成 MEX 文件的编译。注意，mexFilename.c 文件一定在当前的路径下。在命令行提示符窗口中同样可以完成系统编译器的配置。

提示：控制台方式

控制台方式这个名词应该是来自于 UNIX/Linux 操作系统。而在 Windows 9x 操作系统中，为了照顾大多数从 MS-DOS 操作系统过渡到 Windows 操作系统的用户，Windows 提供了一个可以运行在 Windows 图形界面中的字符界面——MS-DOS 窗口，叫作 MS-DOS 方式。在 Windows 2000 系统中叫作命令提示符。

3.2　MEX 源文件的结构

这里稍微花费一点时间来查看一下例 3-1 的代码文件，熟悉一下 C 语言 MEX 源文件的基本结构和 MEX 函数文件输入、输出参数。

3.2.1　源文件的基本结构

C 语言 MEX 源文件是标准的 C 语言源文件，其程序的基本语法完完全全是 ANSI C 的语法结构。所以，请读者注意，某些 C/C++编译器不仅仅支持标准 C 语言的语法，可能还支持一些特殊的 C 语法结构，例如 Microsoft Visual C/C++ 6.0 的编译器支持以“//”的形式添加程序注释，这些特殊的程序特性最好不要出现在 C 语言 MEX 源文件中，以免影响了程序的可移植特性。

在例 3-1 中程序的第 002 行包含了一个头文件——mex.h，在所有 C 语言的 MEX 源文件中必须包含该头文件，该头文件位于%MATLABROOT%\extern\include 路径下。在这个头文件中完成了所有 C 语言 MEX 函数的原型声明，同时还包含了 matrix.h 文件。matrix.h 文

件包含了第 2 章介绍的若干 mx 函数和数据类型的声明和定义，所以在编写 C 语言 MEX 源文件的时候，就不必在 C 文件中再次包含 matrix.h 文件了。

例 3-1 的源文件第 004 行和 005 行为 C 语言 MEX 源文件的入口函数部分。这两行代码是所有 C 语言 MEX 源文件必须包含的内容，而且书写的内容也必须按照固定的格式，也就是说必须按照例子中的书写方法来表示这两行代码：

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

mexFunction 函数的作用类似 C 语言源文件的 main 函数，它是 MEX 文件的入口函数。当在 MATLAB 命令行中执行 MEX 函数时，MATLAB 解释器将从此函数开始执行 MEX 代码。

该入口函数的输入参数有四个，其意义分别是：

nlhs：表示输入参数的个数。

Plhs：mxArray 类型的指针数组，表示 MEX 函数的输入参数。

Nrhs：表示输出参数的个数。

Prhs：mxArray 类型的指针数组，表示 MEX 函数的输出参数。

其实上述输入、输出参数的名称非常容易记忆，lhs 代表 Left hand parameters，rhs 代表 right hand parameters，n 代表 number，p 代表 pointer。这里的左手和右手如何划分呢？例如在 MATLAB 中调用求伯特图的函数时，可以这样表示：

```
[mag,phase,w] = bode(sys)
```

这里以“=”为分界，左边的参数 mag、phase 和 w 为输出参数，即左手参数，而等号右边的参数 sys 为输入参数，即右手参数。

3.2.2 MEX 文件的参数

在 MEX 应用程序中判断用户的输入参数和输出参数是否满足要求是非常重要的一个操作，参见下面的例子。

例 3-2 判断输出参数——simple.c。

```
001    #include "mex.h"
002    /* 函数声明，在 MEX 入口函数内部调用 */
003    int AssignNumericData(double *pr, int size);
004
005    /* MEX 文件入口函数*/
006    void mexFunction(int nlhs, mxArray *plhs[],
007                     int nrhs, const mxArray *prhs[])
008    {
009        /*变量声明 */
010        mxArray * mystr;
011        int flag;
012        char myprintchar[100],mychar[]= "This is a simple MEX-File ";
```

```
013
014         /* 判断输出参数个数，必须具有一个输出参数*/
015         if(nlhs != 1)
016             mexErrMsgTxt("必须给一个输出参数！ ");
017
018         /* 创建字符串 mxArray 数据对象
019            创建数据对象的同时赋初值 */
020         mystr = mxCreateString(mychar);
021         /* 创建双精度类型 mxArray 数据对象
022            该数据对象是函数的输出
023            创建时，数据对象为空，该数据将被 AssignNumericData 函数赋值*/
024         plhs[0] = mxCreateDoubleMatrix(2,3,mxREAL);
025
026         /* 双精度类型 mxArray 数据对象赋值 */
027         flag = AssignNumericData(mxGetPr(plhs[0]),
028                                  mxGetNumberOfElements(plhs[0]));
029
030         /* 输出字符串 */
031         mxGetString(mystr,myprintchar,
032                     mxGetNumberOfElements(mystr)*mxGetElementSize(mystr));
033         mexPrintf(myprintchar);
034
035         /* 退出 mexFunction 函数时，释放必要的内存空间*/
036         mxDestroyArray(mystr);
037     }
038     /* 算法函数 */
039     int AssignNumericData(double *pr, int size)
040     {
041         double data[]={1.,2.,3.,4.,5.,6.};
042         memcpy(pr,data,size*sizeof(double));
043         return(EXIT_SUCCESS);
044     }
```

例 3-2 的代码尽管比较长，却是一个完整的 C 语言 MEX 应用程序。请读者注意程序的 014～016 行，在这三行代码中对输出参数的个数进行了判断，要求执行该函数的时候必须给定一个输出。在 MATLAB 中编译并执行该程序：

```
>> mex simple.c
>> simple
??? 必须给一个输入参数！
```

```
Error in ==> D:\Temp\simple.dll
>> [a b] = simple
??? 必须给一个输入参数!
Error in ==> D:\Temp\simple.dll
>> a = simple
This is a simple MEX-File
a =
     1     3     5
     2     4     6
```

在执行 simple.c 程序时，命令行中必须给定合适的输出参数，否则根据程序中的判断将给出错误的信息，该信息是通过 mexErrMsgTxt 函数给出的，该函数的详细解释可以参阅 MATLAB 的帮助文档，在 3.5 小节中，还会再次使用该函数来控制程序的错误。

请读者留意程序的 021～024 行，在 MEX 函数源文件中，必须对输出参数占用的内存空间进行合理的分配，否则会造成内存泄漏等异常错误。所谓合理的分配就是设置符合要求的数据类型和数组元素个数。那么 mexFunction 入口函数的参数和 MATLAB 函数的输入、输出参数之间究竟是怎样对应的呢？查看下面的 C 语言 MEX 文件代码片段：

```
/* MEX - File myfun.c */
# include "mex.h"
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[])
{
  /* 对输入输出参数的个数进行判断，防止程序出错 */
  ......
  plhs[0]= mxCreate...
  plhs[1]= mxCreate...
  plhs[2]= mxCreate...
  ......
  /* 使用输入参数 prhs[0],prhs[1],prhs[2]和 prhs[3]
     完成相应的计算 */
  /* 将必要输出参数赋值给 plhs[0],plhs[1],和 plhs[2]* /
  /* 在程序退出之前释放必要的内存 */
}
```

假设在 MATLAB 命令行中这样调用该 MEX 文件：

```
>> [x,y,z] = myfunc(a,b,c,d);
```

则在函数调用时输出参数的个数 nlhs 为 3，输入参数的个数为 4，输入参数的指针 prhs[0]对应输入参数 a，prhs[1]对应输入参数 b，prhs[2]对应输入参数 c，prhs[3]对应输入参数 d。在函数的内部，必须对输出参数进行内存分配，于是使用 mxCreate 函数分配内存，注意分

配内存时需要针对具体的数据类型选择 mxCreate 函数，然后在 MEX 文件中完成相应的计算和处理。

当 MEX 函数文件退出时，M 解释器自动将输出参数的指针依次赋值给实际的输出参数，既 prhs[0]赋值给参数 x，prhs[1]赋值给参数 y，prhs[2]赋值给参数 z，所以不要将分配给输出参数的内存释放掉。

在 simple.c 源文件中也给出了编写使用 MEX 文件的基本原则：在入口函数 mexFunction 中调用其它 C 语言函数，从而完成 C 语言在 MATLAB 应用中的集成。

3.3　创建 MEX 文件

创建 MEX 文件的时候，在 MATLAB 命令行中需要使用 MEX 指令，该指令是 MATLAB 产品的一部分，它主要完成以下功能：

- 清除内存中的已经加载的 MEX 函数；
- 调用系统脚本(在操作系统中可以独立使用)，完成 MEX 文件生成。

其中不同的操作系统平台上的系统脚本类型不一样，例如在 Windows 平台上是 Perl 语言脚本，而在 UNIX 平台上则是 B-shell 脚本。

除了使用 MEX 指令在 MATLAB 命令行创建 MEX 文件的方法外，对于 C 语言的 MEX 文件，还可以使用 MATLAB 提供的 Visual Studio 插件完成创建工作。

本小节将具体介绍 MEX 指令以及在 Visual Studio 中创建 MEX 文件的方法。

3.3.1　MEX 指令

使用 MEX 指令不仅能够创建 C 语言的 MEX 文件，而且还能够创建 Fortran 语言的 MEX 文件，具体创建的 MEX 文件类型是通过预先配置的编译器类型决定的。完整地执行 MEX 指令的命令行如下：

```
>> mex [option1 ... optionN] sourcefile1 [... sourcefileN] …
[objectfile1 ... objectfileN] [libraryfile1 ... libraryfileN]
```

其中，option1……optionN 是 MEX 指令的命令行参数选项；sourcefile 为参与编译生成 MEX 文件的所有 C 文件；objectfile 和 libraryfile 分别为对象文件和函数库文件。

熟悉 C 语言或者 Fortran 语言的读者比较清楚，一般的高级编程语言从源代码到最终的可执行程序的生成过程包含两个步骤：第一个步骤是对源代码文件进行编译，这一步主要检查程序源代码是否有语法或者词法错误，将源文件编译成为目标文件；第二个步骤是链接过程，将源文件生成的目标文件和必要的库文件、其它的目标文件链接在一起，生成最终的可执行程序文件。在生成 MEX 文件时也是如此，只不过系统的 MEX 脚本将这一过程一次完成了，并且隐藏了其中的细节，如果需要对编译链接过程进行控制，则要通过设置相应的选项文件和命令行开关来完成。在表 3-2 中总结了 MEX 脚本的命令行参数。

表 3-2 MEX 脚本命令参数

参 数	说 明
@<rsp_file>	将 rsp_file 中包含的内容作为 MEX 脚本的命令参数
-argcheck	检测库函数的输入、输出参数
-c	仅完成编译，不进行链接过程
-D<name>[#<def>]	定义 C 语言预处理宏<name>
-f <file>	指定编译选项文件，如果该选项文件没有保存在当前的路径下，则 file 项需要使用完整的路径名称和文件名称。当指定该参数时，默认的选项文件 mexopts.bat 则不起作用了
-g	编译生成的 MEX 文件中包含必要的调试(debug)信息
-h[elp]	列出 MEX 指令所有的帮助信息，包含本表中的参数说明
-I<pathname>	将指定的路径 pathname 添加到系统的 include 路径中
-inline	内嵌函数文件中包含的 mx 函数，注意利用此参数生成的 MEX 文件也许和未来版本的 MATLAB 不兼容
-l<file>	将 MEX 文件与指定的库文件链接(仅在 UNIX 平台上适用)
-L<pathname>	将指定的路径 pathname 添加到库函数的搜索路径中(仅在 UNIX 平台上适用)
<name>#<def>	使用 def 内容替代选项文件中 name 选项的内容
<name>=<def>	使用 def 内容替代选项文件中 name 选项的内容(仅在 UNIX 平台上适用)
-O	创建代码优化的文件
-outdir <name>	指定文件的输出路径
-output <name>	指定创建文件的文件名称
-setup	设置系统默认的编译器以及编译器的选项文件
-U<name>	取消 C 预处理程序中指定的宏定义
-v	显示详细的编译过程
-V5	创建与 MATLAB 5 兼容的 MEX 文件

使用上述命令行参数可以控制 MEX 文件的生成，例如创建包含下面内容的一个纯文本文件：

-c -v -O -argcheck

然后将该文件保存为 test_rsp.txt，如果利用该文件完成 MEX 文件的编译，则可以在 MATLAB 命令行窗口中键入下面的指令：

```
>> mex @test_rsp.txt simple.c
 This is mex, Copyright 1984-2002 The MathWorks, Inc.

-> Default options filename found in C:\Documents and Settings\Administrator\Application
Data\MathWorks\MATLAB\R13
----------------------------------------------------------------
->      Options file                = C:\Documents and Settings\Administrator\Application
```

```
Data\MathWorks\MATLAB\R13\mexopts.bat
        MATLAB                  = E:\MATLAB6p5p1
->      COMPILER                = cl
->      Compiler flags:
            COMPFLAGS           = -c -Zp8 -G5 -W3 -DMATLAB_MEX_FILE -nologo
            OPTIMFLAGS          = /MD -O2 -Oy- -DNDEBUG
            DEBUGFLAGS          = /MDd -Zi -Fd"simple.pdb"
            arguments           =  -DARGCHECK
            Name switch         = /Fo
->      Pre-linking commands    =
->      LINKER                  = link
->      Link directives:
            LINKFLAGS                                   = /dll/export:mexFunction/MAP
/LIBPATH:"E:\MATLAB6p5p1\extern\lib\win32\microsoft\msvc60"  libmx.lib  libmex.lib
libmatlb.lib libmat.lib /implib:_lib7928.x
            LINKFLAGSPOST       =
            Name directive      = /out:"simple.dll"
            File link directive =
            Lib. link directive =
            Rsp file indicator  = @
->      Resource Compiler       = rc /fo "mexversion.res"
->      Resource Linker         =
-------------------------------------------------------------
... ...
```

上面的编译过程利用了 test_rsp.txt 文件中定义的 MEX 命令行参数，于是在 MATLAB 命令行中将给出详细的编译过程，并且生成的文件不是 MEX 文件，而是编译之后得到的目标文件 simple.obj。

在使用 MEX 脚本编译、链接多个文件时需要注意，第一个源文件的名字将成为 MEX 函数文件的名字，并且该源文件中必须包含 MEX 函数的入口函数 mexFunction。

3.3.2　在 Visual Studio 中创建 MEX 文件

前面创建 C 语言 MEX 文件的方法相对是比较麻烦的，首先需要利用任何一种纯文本编辑器创建 C 语言源文件，然后回到 MATLAB 环境下利用 MEX 指令将 C 语言源文件编译生成为 MEX 文件。这对那些习惯利用集成开发环境进行开发的程序员来说是非常不习惯的，所以在本小节介绍利用集成开发环境——Visual Studio 创建 MEX 函数文件的方法。

Visual Studio 是由 Microsoft 开发的一套非常流行的集成开发环境，尽管有很多程序员认为微软公司推出的 C/C++编译器不是最优秀的，但是不能否认 Visual Studio 是 Windows 平台下最流行的 C/C++ Windows 32 位开发工具。目前 Visual Studio 的最新版本是 Visual Studio .NET，也就是常说的 Visual Studio 7。

MATLAB 为了便于利用 Visual Studio 开发 MATLAB 应用程序，特别提供了 Visual Studio 的应用程序插件，不过 Visual Studio 插件仅支持 Visual Studio 5 或者 6 版本。

本书中首先介绍 Visual Studio 插件的使用方法，然后再介绍利用 Visual Studio .NET 的集成开发环境创建 MEX 文件的方法。

1. 使用 MATLAB Add-in

MATLAB Add-in 是为了方便 Visual Studio 用户在自己熟悉的集成开发环境中创建 MATLAB 应用程序而开发的，它可以创建 C 语言 MEX 文件，并将其独立为执行程序等。

注意：

MATLAB Add-in for Visual Studio 是随 MATLAB Compiler 产品一同发布的，所以只有当用户购买的 MATLAB 产品包含 MATLAB Compiler 时，才能够使用 Visual Studio 的 MATLAB Add-in。

使用 Visual Studio 6 的插件开发 MEX 程序的过程比较简单，下面是主要的操作步骤：

首先在 Visual Studio 6 中添加 MATLAB 的插件。运行 Visual Studio 6，执行“Tools”菜单下的“Customize”命令，在弹出的对话框中选择“Add-ins and Macro Files”页，如果前面配置 MEX 外部编译器时选择了 Visual C/C++6 或者 5，则 MATLAB Add-in 会出现在对话框中。选择“MATLAB Add-in”，如图 3-1 所示。

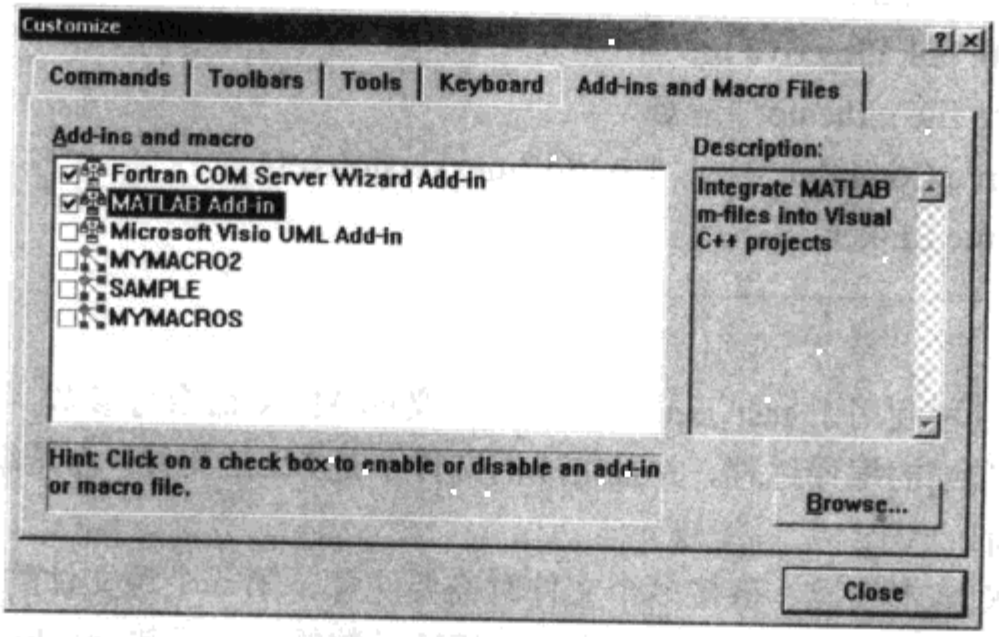


图 3-1 选择 MATLAB Add-in

关闭“Customize”对话框，这时在 Visual Studio 开发环境中将出现 MATLAB 工具条，如图 3-2 所示。

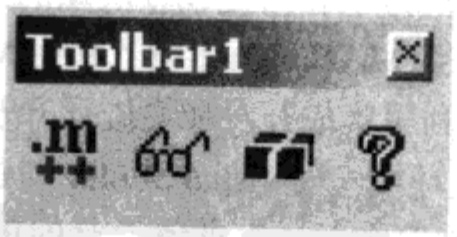


图 3-2 MATLAB Add-in 工具条

执行“File”菜单下的“New”命令，在弹出的新建对话框中选择“Projects”页，选择“MATLAB Project Wizard”项，如图 3-3 所示。

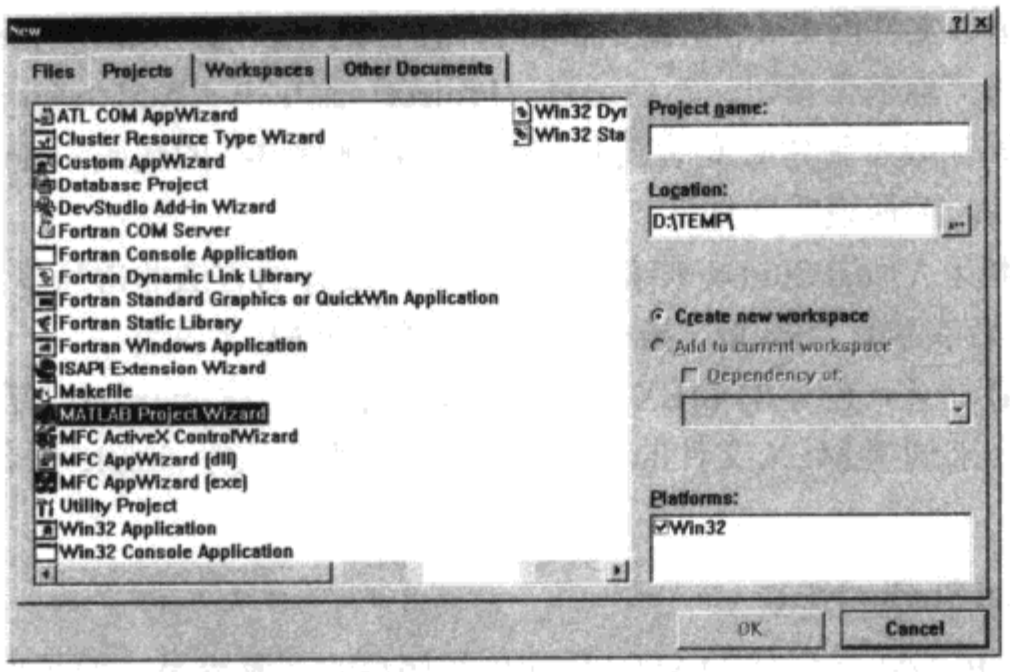


图 3-3　选择 MATLAB Project Wizard

在“Project name”文本框中输入项目的名称，例如 mexTest，然后单击“OK”按钮。这时将弹出“MATLAB Project Wizard”对话框，在该对话框的“Visual MATLAB Application Type”下拉框中选择“C-MEX DLL”，再选中“No main file”单选框，如图 3-4 所示。

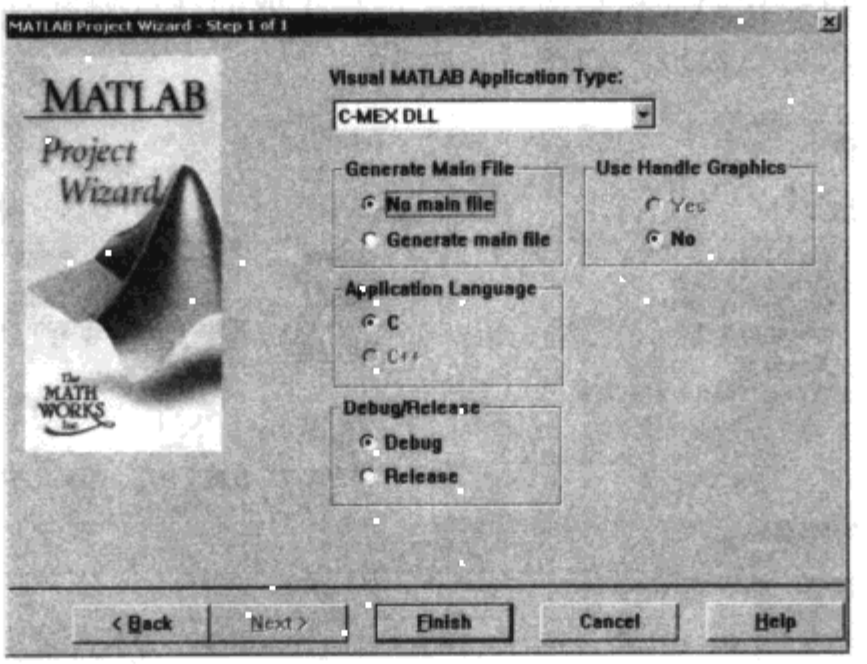


图 3-4　MATLAB Project 的设置

单击“Finish”按钮，这时将弹出“Project Information”对话框，提示用户刚刚完成的项目信息，再次单击“OK”按钮，这时 Visual Studio 将创建相应的项目，选择 C 文件对话框，如图 3-5 所示。

图 3-5　选择 C 文件对话框

如果 MEX 函数文件的 C 代码源文件已经存在，则可以通过该对话框选择相应的 C 文件。否则可以单击“取消”按钮，然后通过“Project”菜单下“Add to Project”子菜单下的命令向项目文件添加或者新建 C 语言源代码。例如添加例 3-2 的代码 simple.c，这时 Visual Studio 将创建项目的编译说明文件。

接着就可以利用 Visual Studio 来编译 C 文件了，编译的方法非常简单，直接按 F7 键或者通过“Build”菜单下的“Build”指令就可以编译生成 MEX 文件了。

不过需要读者注意，创建的 MEX 文件名称是项目的名称 mexTest。在 MATLAB 中是通过 mexTest 指令来创建 MEX 文件的，而不是像前面那样执行 simple 指令，具体的请读者自己尝试完成。

2. 使用 Visual Studio.NET

使用 MATLAB Add-in 创建 MATLAB 应用程序尽管非常容易，但是也有其局限性。并不是每一个程序员都使用 Visual Studio 5 或者 6 来开发应用程序，也不是每个 MATLAB 的用户都拥有 MATLAB Add-in，这里以 Visual Studio .NET 为例，演示创建 MEX 文件的过程。

首先，在 Microsoft Visual Studio .NET 中通过“文件”菜单下的“新建”命令创建一个新的项目，创建项目时，在新建项目对话框中选择生成的项目类型为“Visual C++项目”，并且选用“生成文件项目”模板，如图 3-6 所示。

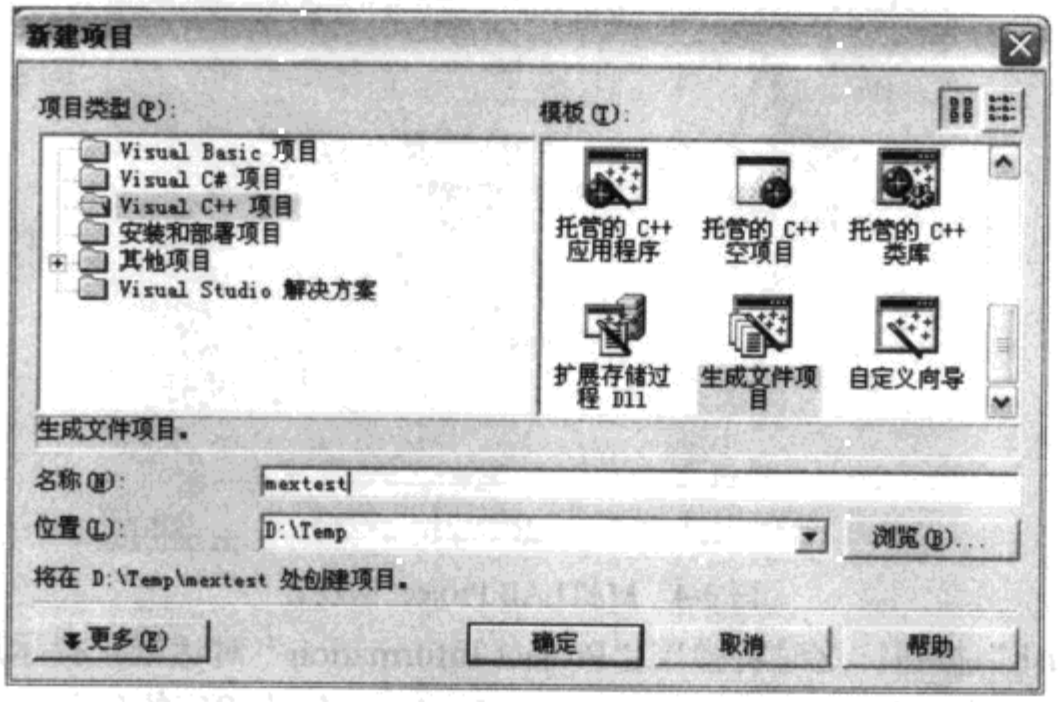


图 3-6 新建项目

在项目名称处给自己的项目取个名字，例如 mextest，单击“确定”按钮，Visual Studio 将创建新的项目。

创建新的项目之后，首先向项目添加资源文件——mexversion.rc，该文件位于%MATLABROOT%\extern\include 路径中，该文件定义了 MATLAB 的 MEX 文件的版本信息，然后再添加 MEX 源文件，例如 simple.c。

第三步，修改项目属性。将项目生成的文件类型，从默认的可执行程序(.exe)修改为动态链接库(.DLL)。执行“项目”菜单下的“属性”命令，在弹出的对话框中，设置配置类型为“动态库”，如图 3-7 所示。

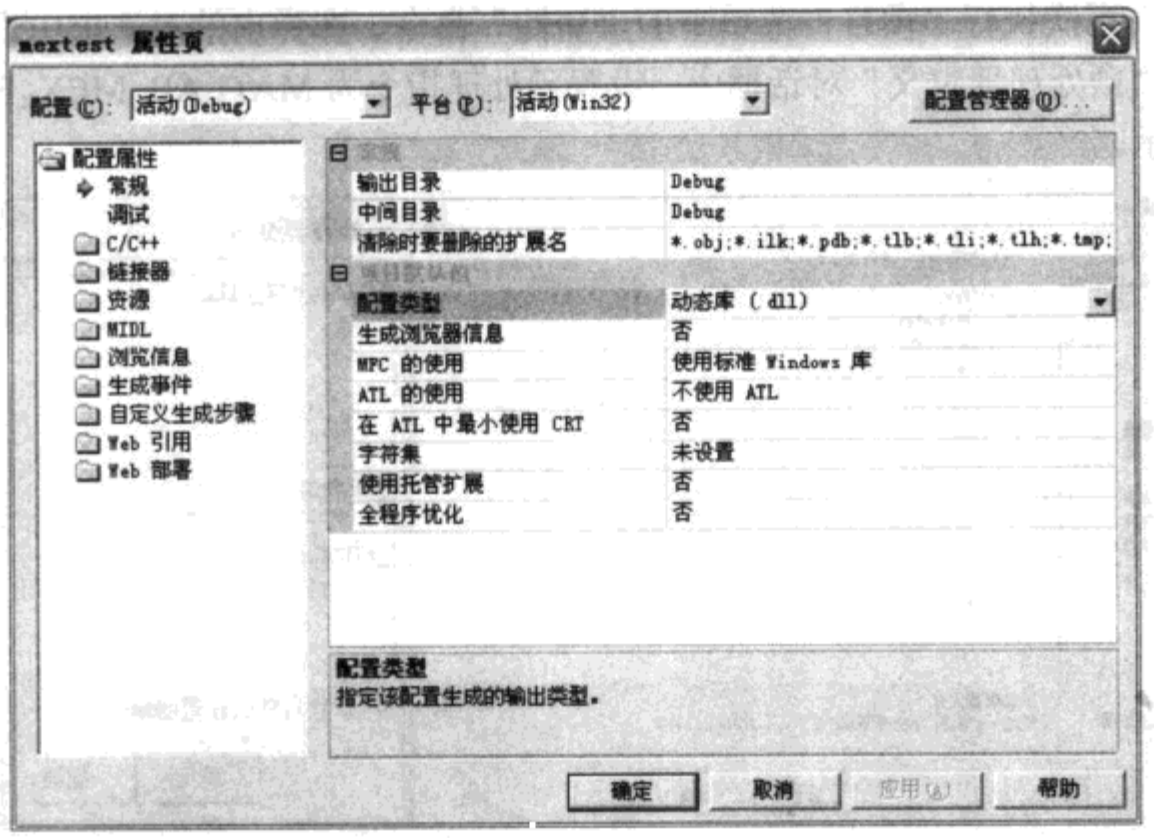


图 3-7　设置项目的配置类型

第四步，创建 DEF 文件。该文件的名称最好同项目工程名称一致，例如 mextest.def，在项目文件中添加下面的内容：

```
LIBRARY     mextest.DLL
EXPORTS     mexFunction
```

第五步，设置 include 文件和 Lib 文件的路径。C 语言的 MEX 文件需要特殊的头文件和库文件配合才能正确完成编译链接工作。执行“工具”菜单下的“选项”命令，在弹出的对话框中选择 VC++项目栏，分别在“包含文件”和“库文件”页中添加 MATLAB 的 include 文件的和 Lib 文件的路径，其中 include 路径设置为%MATLABROOT%\extern\include，Lib 路径设置为$\MATLAB6p5p1\extern\lib\win32\microsoft\msvc70，如图 3-8 所示。

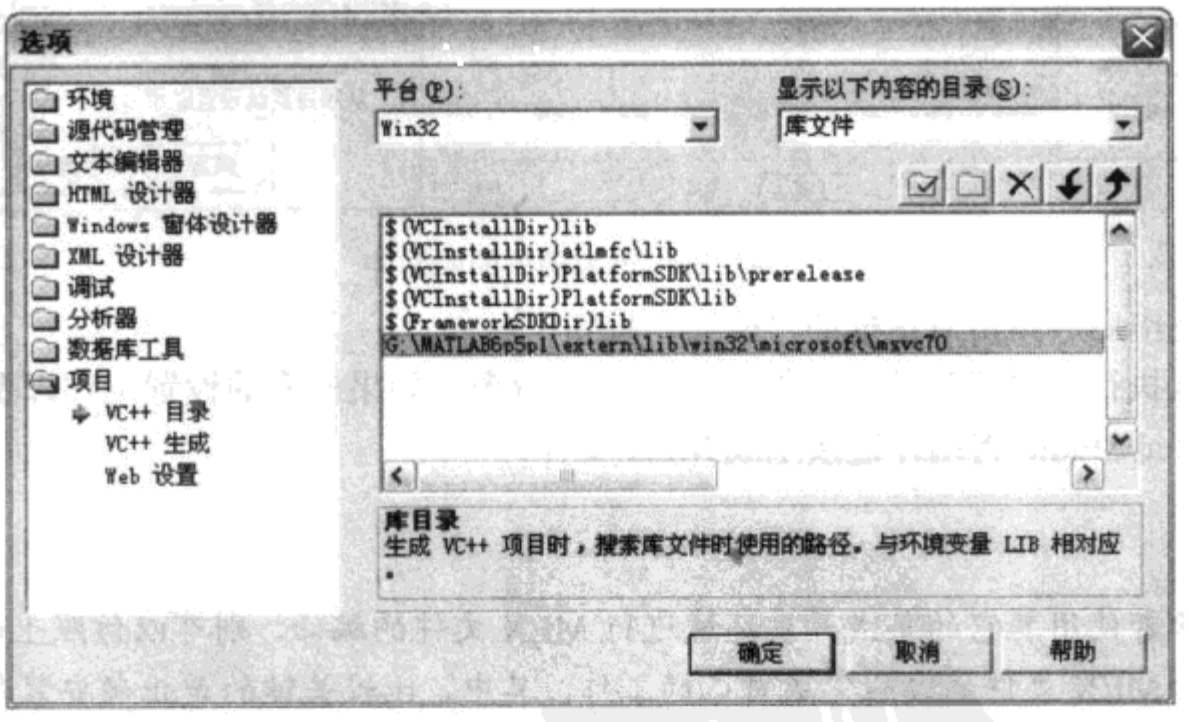


图 3-8　设置库文件路径

注意，在选择库文件时，需要针对不同的编译器选择不同的库文件目录。例如 Visual C++ 6.0 的库文件在%MATLABROOT%\extern\lib\win32\microsoft\msvc60 路径下。

第六步，再次执行“项目”菜单下的“属性”命令，在弹出的对话框中设置 C/C++预编译宏，在“预处理器定义”对话框中，设置预处理指令为 MATLAB_MEX_FILE，对话框如图 3-9 所示。

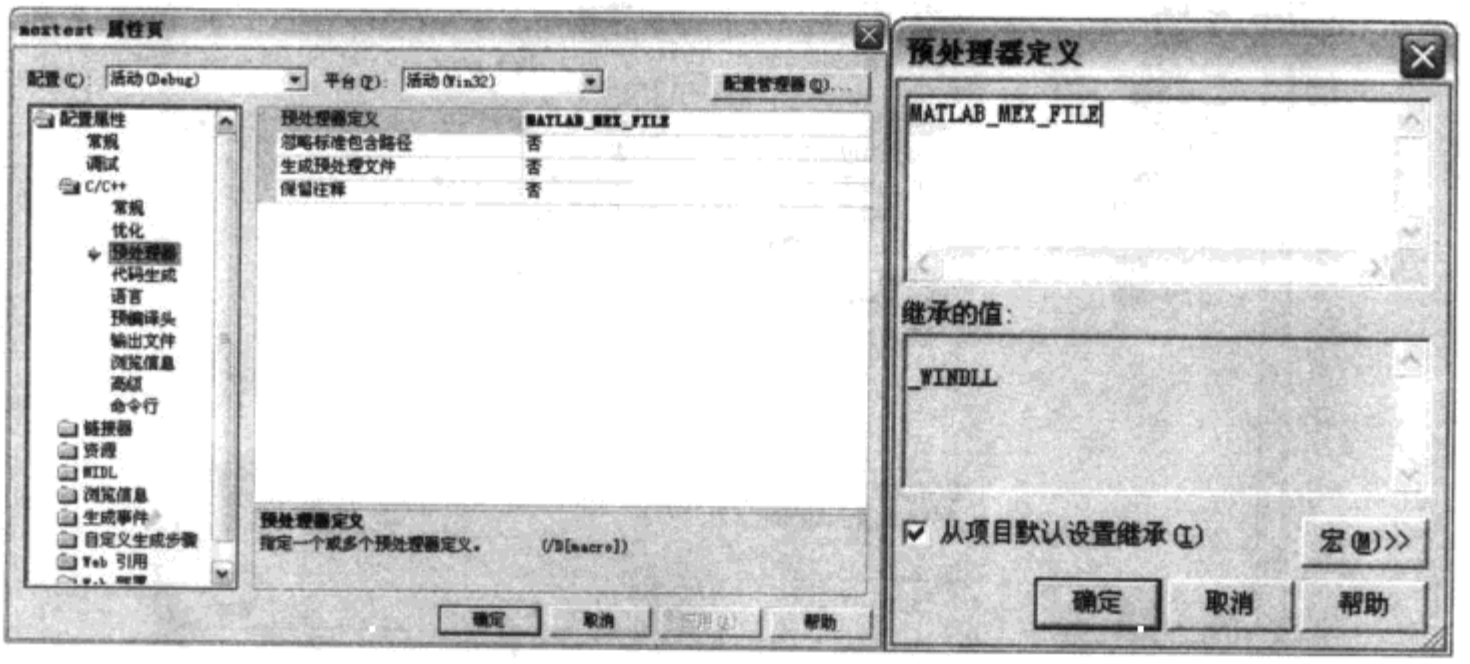


图 3-9 设置预处理器定义对话框的参数

同样，在项目属性对话框中需要设置链接库文件的参数，在链接器参数设置的输入项目中，设置需要额外增加的库文件：libmat.lib、libmex.lib 和 libmx.lib，如图 3-10 所示。

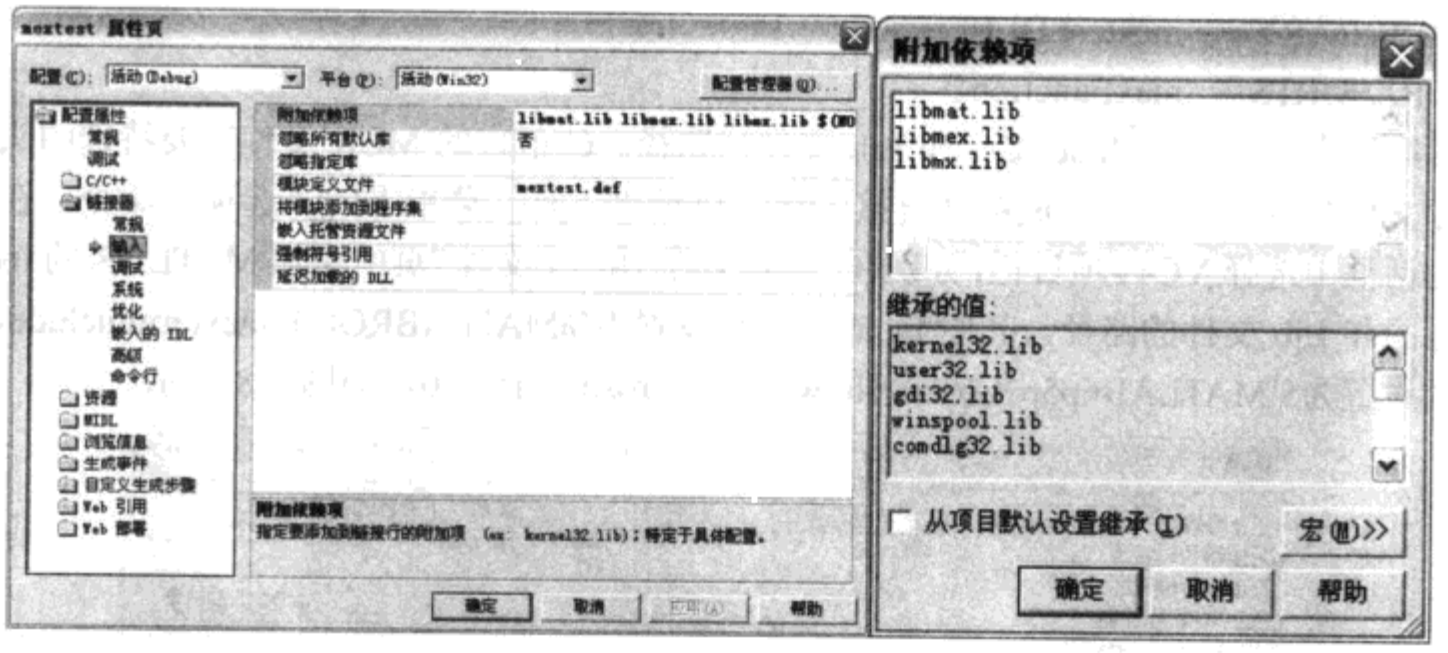


图 3-10 添加额外的库文件

设置完毕后，单击对话框的“确认”按钮。

最后，执行“生成”菜单下的“mextest”命令，如果前面的设置工作以及代码没有错误，则 Visual Studio 将编译链接生成 MEX 文件。

提示：

如果读者使用其它的集成开发环境进行 MEX 文件的编译，则可以仿照上面使用 Visual Studio 创建 MEX 文件的过程完成自己的工作。其中，比较关键的是正确安装集成开发环境并且向 Windows 正确注册系统环境变量(头文件、库文件的路径等)。按照上面提示的步骤，正确设置项目类型(生成动态库)、DEF 文件、库文件和预编译器定义，从而链接正确的库文件。

如果读者需要详细了解在集成开发环境创建下 MEX 文件的方法，或者需要了解在 UNIX 环境下自定义编译链接 MEX 文件的方法，请参阅 MATLAB 的帮助文档。

3.4　MEX 文件的内存管理

在第 2 章中介绍了用于管理 mxArray 数据对象的内存管理函数，并且强调指出了在外部接口应用程序中进行良好的内存管理是非常重要的工作。在本小节中，将继续讨论有关在 MEX 文件中进行内存管理的话题。

3.4.1　内存自动释放机制

自动内存释放机制是 MATLAB 提供的一种内存管理机制，当 MEX 执行完毕退出时，系统将自动释放临时分配的变量，而只有通过 plhs[]传递给 MATLAB 的输出参数的那部分内存不会被自动释放掉。即使内存变量是通过 mxMalloc、mxCalloc 和 mxRealloc 函数分配的，只要 MEX 文件退出后这些内存不再被使用，它们也会被自动地被释放掉。尽管如此，MATLAB 还是推荐用户在编写 MEX 应用程序时，显性地在代码中释放那些不再使用的临时变量占用的内存空间，因为这样做比依赖 MATLAB 自己完成要有效率。

不过很多事情都有例外发生，例如在 MEX 文件中发生了下面几种情况时，用户自己编写的内存释放代码就有可能不会起作用了：

- 当 MEX 函数文件中执行了 mexErrMsgTxt 函数时。
- 当 MEX 函数文件中调用了其它的 MATLAB 指令而出现错误时。
- 当用户通过 Ctrl+C 终止当前的程序运行时。
- 当系统内存被 MATLAB 耗尽而引起当前 MEX 文件终止运行时。

在这几种情况下就要依赖于内存自动释放机制来完成内存的释放了。不过笔者这里提醒读者，不要过分依赖 MATLAB 提供的内存释放机制，当应用程序中使用了不恰当的内存操作时内存泄漏的情况就会时有发生，这时 MATLAB 的内存释放机制也是无能为力的，下面来看一个例子。

例 3-3　内存管理示例——change.c。

```
001    # include "mex.h"
002    /* MEX 文件的入口函数 */
003    void mexFunction(int nlhs, mxArray *plhs[],
004                     int nrhs, const mxArray *prhs[])
005    {
006        /* 临时 mxArray 数据对象 */
007        mxArray * temp;
008        /* 实际数据指针 */
009        double *pr;
010        /* 使用 prhs[0]为临时变量赋值 */
011        temp = prhs[0];
```

```
012         /* 获取实际数据的指针 */
013         pr = mxGetPr(temp);
014         /* 赋新的数值 */
015         *pr = 1000;
016     }
```

例 3-3 中包含的代码表面上没有什么问题，在 MATLAB 命令行中进行如下操作：

```
>> mex change.c
>> x = 10;
>> y = x;
>> change(y)
>> x,y
x =
        1000
y =
        1000
```

在上面的操作中针对变量 y 进行了修改，变量 x 的数值随之也被修改了，这显然不是程序开发人员或者用户所希望的。为什么会出现这种错误呢？主要是由 MATLAB 自身的内存管理机制造成的。在上面的操作中，由于 x 和 y 引用的内容是一样的(都是数字 10)，x 和 y 实际上是两个指针，指向了同一个数据，如果通过一个指针 y 修改了数据，则另一个指针 x 指向的数据就会跟着发生变化。正确的做法是通过 mxDuplicateArray 函数完成数据的复制，该函数的定义如下：

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

该函数的输入参数是需要复制的数据，输出参数则为输入数据的副本。例 3-3 修改之后的代码见例 3-4。

例 3-4 内存复制操作——change_duplicate.c。

```
001     /* mxDuplicateArray 函数的重要作用*/
002     # include "mex.h"
003     void mexFunction(int nlhs, mxArray *plhs[],
004                      int nrhs, const mxArray *prhs[])
005     {
006         mxArray * temp;
007         double *pr;
008         /* 利用 prhs[0]为临时变量赋值 */
009         temp = mxDuplicateArray(prhs[0]);
010         pr = mxGetPr(temp);
011         *pr = 1000;/* 赋新的数值 */
012         mxDestroyArray(temp); /*好习惯！ */
013     }
```

再次对例 3-4 执行如下操作：

```
>> mex change_duplicate.c
>> x = 10;
>> y = x;
>> change_duplicate(y)
>> x,y
x =
     10
y =
     10
```

这时 x 和 y 的数据没有真正的修改，即使程序的 012 行代码不存在也是一样。

在第 2 章通过例子说明了不恰当使用 mxSet 函数是造成内存泄漏的一个原因，那么在 MEX 函数中如果不恰当使用 PRHS，则也有可能会造成内存的泄漏，它分为以下两种情况：

第一种情况，从 PRHS 直接对临时变量赋值，例如下面的代码片段：

```
void mexFunction(int nlhs, mxArray * plhs[],
                 int nrhs, const mxArray * prhs[])
{    mxArray * Temp;
     ...
     Temp = prhs[0];   /* 错误 */
     Temp = mxDuplicateArray(prhs[0]) /* 正确 */
     ...
}
```

直接从 PRHS 向临时变量赋值是错误的，这是因为利用内存自动释放机制，临时变量在 MEX 文件退出时会自动地释放，当临时变量的内存空间被释放的时候，输入参数占用的内存也会跟着被释放，所以正确的做法是使用 mxDuplicateArray 函数进行临时变量的赋值。

第二种情况，从 PRHS 直接对 PLHS 赋值，例如下面的代码片段：

```
void mexFunction(int nlhs, mxArray * plhs[],
                 int nrhs, const mxArray * prhs[])
{    mxArray * Temp;
     ...
     plhs[0] = prhs[0];   /* 错误 */
     plhs[0] = mxDuplicateArray(prhs[0]) /* 正确 */
     ...
}
```

直接从 PRHS 向 PLHS 赋值是错误的，主要原因是输出参数必须通过内存分配才能够存在，如果直接进行赋值会造成不可预料的错误，因此正确的做法还是应该使用 mxDuplicateArray 函数对输出参数完成赋值。

3.4.2 内存保留变量

内存保留变量可以是一个矩阵或者数组，也可以是内存中的一个片段，程序员可以在MEX 文件中声明一段内存为保留(persistent)，方法是使用 mexMakeArrayPersistent 或者mexMakeMemoryPersistent 函数。由于自动内存释放机制就不会释放被声明为保留(persistent)的内存，所以在使用内存保留变量的时候一定千万小心，一旦在彻底释放变量内存之前 MEX 文件就被从内存中清除了，被保留的内存就不再会被释放，这样就造成了内存泄漏。所以，稍微可靠一点的方法是通过 mxAtExit 函数注册一个内存释放的函数，在那个函数中完成保留内存的释放。mexAtExit 函数的作用是向 MATLAB 注册一个子函数，该子函数将在该 MEX 文件从内存中清除时自动执行，这样可以在注册的函数中完成资源的整理等工作，参见下面的例子。

例 3-5 保留的内存空间——persistentmemory.c。

```
001     #include "mex.h"
002
003     static int initialized = 0;
004     static mxArray *persistent_array_ptr = NULL;
005     static char *persistent_memory_ptr = NULL;
006
007     void cleanup(void) {
008         mexPrintf("退出 MEX 函数，清除内存！\n");
009         mxDestroyArray(persistent_array_ptr);
010         mxFree(persistent_memory_ptr);
011     }
012
013     void mexFunction(int nlhs,mxArray *plhs[],
014                      int nrhs,const mxArray *prhs[])
015     {
016         char ptr[] = "Isn't MATLAB Great?!";
017         if (!initialized) {
018             mexPrintf("初始化 MEX 函数，分配内存！\n");
019
020             /* 创建保留内存的变量 */
021             persistent_array_ptr = mxCreateDoubleMatrix(1, 1, mxREAL);
022             mexMakeArrayPersistent(persistent_array_ptr);
023             /* 创建保留内存空间 */
024             persistent_memory_ptr = mxMalloc(100);
025             mexMakeMemoryPersistent(persistent_memory_ptr);
026             /* 注册清除函数 */
```

```
027                 mexAtExit(cleanup);
028                 initialized = 1;
029
030                 /* 对保留变量和内存空间赋值 */
031                 *mxGetPr(persistent_array_ptr) = 1.0;
032                 memcpy(persistent_memory_ptr,ptr,sizeof(char)*21);
033             } else {
034                 mexPrintf("执行了 MEX 文件，保留的变量为:%g\n",
035                                 *mxGetPr(persistent_array_ptr));
036                 mexPrintf("保留的内存为：%s",persistent_memory_ptr);
037             }
038         }
```

在 MATLAB 命令行窗口中编译并执行该 MEX 文件：

```
>> mex persistentmemory.c
>> persistentmemory
初始化 MEX 函数，分配内存。
>> %再次执行 MEX 函数文件
>> persistentmemory
执行了 MEX 文件，保留的变量为 1
保留的内存为 Isn't MATLAB Great?!。
>> %清除当前的工作空间和内存占用
>> clear all
退出 MEX 函数，清除内存！
```

 **提示：**

本小节使用了三个 mex 函数，分别为 mexMakeArrayPersistent、mexMakeMemoryPersistent 和 mexAtExit 函数，这三个函数的详细解释请参阅 MATLAB 的帮助文档。

3.4.3　复合数组

在第 2 章中介绍了创建 mxArray 数据结构类型的函数，以及可以直接对 mxArray 数据结构对象进行赋值的函数，例如 mxSetPr，mexSetPi，mxSetCell 等。释放由这些函数创建或者赋值的 mxArray 数据结构对象占用的内存需要使用 mxDetroyArray 函数来完成，但是，这种内存释放的方法不见得可靠，因为不是任何 mxArray 数据对象都可以被 mxDetroyArray 函数删除，以释放占用的内存的，复合数组(hybrid Array)就是其中的一种。

复合数组是一种特殊的数组，该数组中有一些数据可以被释放，而有一些数据却不能被释放，例如下面的代码片段：

```
mxArray *pArray = mxCreateDoubleMatrix(0, 0, mxREAL);
double data[10];
```

```
mxSetPr(pArray, data);
mxSetM(pArray, 1);
mxSetN(pArray, 10);
```

这里创建的 pArray 指针得到的矩阵就是复合数组，因为 mxFree 函数不能释放 data 占用的内存，更不能使用标准 C 语言的 free 函数来释放内存，所以 pArray 矩阵是一个复合数组。

还有一种情况就是当元胞数组或者结构数组的某个元素是只读的情况时，该元胞数组或者结构数组就是复合数组。

由于复合数组所占用的内存空间不能清除，所以尽管复合数组使用起来比较方便，在 MEX 应用程序中还是应该尽量避免出现复合数组。像前面的代码，完全可以使用 memcpy 来完成 pArray 数据的赋值。

3.5 MEX 文件示例

在第 2 章曾经提及，不同的外部接口应用程序对应着不同的外部接口函数，那么 MEX 函数文件对应的就是 mex 函数。例如在 3.4.2 小节中，使用了 mexMakeArrayPersistent 函数、mexMakeMemoryPersistent 函数和 mexAtExit 函数。本小节将通过一些 MEX 文件的示例说明其它一些比较重要的 mex 函数的使用方法。

注意：

MEX 函数只能够用在 MEX 应用程序的源文件中，本小节只简要介绍 MEX 函数的使用方法，MEX 函数的具体解释请读者参阅 MATLAB 的帮助文档。

例 3-6　编译多个文件。

在 Istranglechk.c 文件中包含这样的代码：

```
001     int Istrianglechk(double side1, double  side2, double  side3)
002     {
003       /* 检查边长 */
004       int flag;
005       if ((side1+side2 > side3) && (side2+side3 > side1) &&
006           (side1+side3 > side2))
007           flag = 1;
008       else
009           flag = 0;
010      /* 检测是否为等腰三角形或者等边三角形 */
011       if (flag == 1)
012       {
013               if (side1 == side2 && side2 == side3)
014                   flag = 3;
```

```
015                else if (side1 == side2 || side2 == side3 || side3 == side1)
016                    flag = 2;
017            }
018        return(flag);
019    }
```

上面的代码是用来判断当前输入的三个数据是否能够构成一个三角形，如果可以，判断是等腰三角形还是等边三角形。这部分 C 代码是已有的一段算法，现在就通过 C 语言的 MEX 文件来调用这部分代码，实现已有的 C 代码和 MATLAB 的结合。实现的 MEX 源文件 Istriangle.c 所包含的代码如下：

```
001    #include "mex.h"
002    /* 算法函数声明 */
003    int Istrianglechk(double, double, double);
004    /* MEX 函数入口*/
005    void mexFunction (int nlhs, mxArray *plhs[],
006                      int nrhs, const mxArray *prhs[])
007    {
008        double *s1,*s2,*s3;
009        int f;
010
011        s1=mxGetPr(prhs[0]);
012        s2=mxGetPr(prhs[1]);
013        s3=mxGetPr(prhs[2]);
014
015        if(*s1 <= 0 || *s2 <= 0 || *s3 <= 0 || nrhs != 3)
016            f=5; /* 除了 0, 1, 2, or 3 则为错误代码 */
017        else
018           f=Istrianglechk(*s1,*s2,*s3);
019        switch(f){
020        case 0:
021            plhs[0]=mxCreateString("无法创建三角形. ");break;
022        case 1:
023            plhs[0]=mxCreateString("能够创建三角形. ");break;
024        case 2:
025            plhs[0]=mxCreateString("能够创建三角形，并且该三角形是等腰三角
形！ ");
026            break;
027        case 3:
028            plhs[0]=mxCreateString("能够创建三角形，并且该三角形是正三角形！
");
```

```
029             break;
030         default:
031             plhs[0]=mxCreateString("错误！请检查输入参数！");
032         }
033     }
```

在 Istriagnle.c 中，018 行调用了 Istrianglechk.c 文件中包含的算法代码。本例子主要向大家演示编译多个文件的方法，在 MATLAB 命令行中键入下面的指令：

```
>> mex Istriangle.c Istrianglechk.c
```

编译生成 MEX 文件时，如果需要有多个文件，则直接将 C 文件或者库文件的名称添加在 mex 命令行中就可以了。但是须强调一点，MEX 函数的入口函数必须在第一个源文件中，而且生成的 MEX 文件名称也和第一个 C 源文件的名称一致。

```
>> what
MEX-files in the current directory D:\Temp
Istriangle
```

运行生成的 MEX 文件：

```
>> Istriangle(1,2,3)
ans =
无法创建三角形.
>> Istriangle(3,2,3)
ans =
能够创建三角形，并且该三角形是等腰三角形!
>> Istriangle(3,3,3)
ans =
能够创建三角形，并且该三角形是正三角形!
```

例 3-7　警告和错误信息。

当用户使用 MEX 文件时，对用户输入的参数或者程序的执行进行监控是程序员必须在代码中完成的工作。每个程序员都必须为自己的程序“修一道栏杆”，避免出现不必要的错误或者无法挽回的损失，那么在程序中进行监测，并根据测试的结果输出必要的信息是非常重要的手段。在 MEX 函数中，可以使用 mexErrMsgTxt 函数和 mexWarnMsgTxt 函数完成警告和错误信息的输出。MEX 文件例子的源代码 processingerr.c 如下：

```
001     include "mex.h"
002     /* MEX 函数入口 */
003     void mexFunction(int nlhs, mxArray *plhs[],
004                      int nrhs, const mxArray *prhs[])
005     {
006         const char *ClassName;
007         char *string = "该函数无法处理元胞数组数据或者结构数据!";
008         double *flag;
```

```
009         /* 判断输入输出参数 */
010         if (nlhs != 0 || nrhs != 2)
011             mexErrMsgTxt("错误：输入输出参数不正确！");
012         /* 设置是否现实错误或者警告
013            当 flag 为 1 时，显示警告信息，
014            否则显示错误信息 */
015         flag = mxGetPr(prhs[0]);
016         /* 获取输入参数的类型 */
017         ClassName = mxGetClassName(prhs[1]);
018         if (strcmp(ClassName,"cell")==0 ||
019             strcmp(ClassName,"struct")==0){
020             if(*flag == 1)
021                 /* 仅显示警告信息 */
022                 mexWarnMsgTxt(string);
023             else
024                 /* 显示错误信息 */
025                 mexErrMsgTxt(string);
026         }else{
027             mexPrintf("类型 : %s\n", ClassName);
028                 plhs[0]=mxCreateString("数据类型判断成功！");
029             }
030     }
```

上面的代码 015 行获取函数的第一个参数，根据这个参数的数值来判断是显示警告还是错误。如果函数输入的第二个参数是元胞数组或者结构类型数组，且第一个参数是 1，则显示警告信息，否则显示错误信息。警告和错误对于程序来说是不同的，警告信息仅仅是一种提示，程序不会因为警告的出现而退出执行，而且可以通过 warning off 指令禁止控制警告在 MATLAB 命令行中的显示。但是错误就不一样，一旦错误发生，则当前的 MEX 文件就会退出执行，用红色字体在 MATLAB 命令行中显示错误信息。

编译并执行 processingerr.c：

```
>> mex processingerr.c
>> A = {rand(5),'String'};
>> %关闭警告显示
>> warning off
>> processingerr(1,A)
>> %上面程序执行了，但是没有任何显示
>> %打开警告显示
>> warning on
>> processingerr(1,A)
Warning: 该函数无法处理元胞数组数据或者结构数据!
```

```
>> %上面的程序执行，并显示了警告信息
>> %输出错误信息
>> processingerr(0,A)
??? 该函数无法处理元胞数组数据或者结构数据!
Error in ==> D:\Temp\processingerr.dll
>> %程序没有执行完毕，出现了错误
>> processingerr(1,10)
类型 : double
ans =
数据类型判断成功!
```

前面曾经指出，程序员在编写函数时对输入、输出参数的个数或者类型进行必要的判断是非常必要的，这样能够保证程序不会出现异常的错误。然而对于使用函数的用户，一个良好的习惯是在使用函数之前阅读一下函数的帮助文档，特别是 MATLAB 函数的在线帮助。但是这里有一点需要提醒读者，MEX 函数文件没有在线帮助，所以对于程序员来说，当开发 MEX 函数文件时，应该开发一个和 MEX 文件同名的 M 函数文件，在 M 文件中编写相应的在线帮助，在线帮助的主要内容就是对输入、输出参数以及函数执行功能的说明，还可以包括程序的版本信息、作者或者公司的信息等内容。当然不必在 M 函数文件中编写任何具体代码。这样，利用 M 函数文件就可以使 MEX 函数文件具有在线帮助了。

例 3-8　曲线拟合——调用 MATLAB 指令。

在 MEX 文件中大多数的操作都是通过 C 语言函数来完成的，但是，不可避免地需要调用一些 MATLAB 的指令，例如其它的 M 文件或者 MEX 文件。在 MEX 函数中，提供了两个函数可以完成调用 MATLAB 指令的功能，其中一个就是 mexEvalString 函数，该函数的定义如下：

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

该函数的输入参数是一个 C 语言的字符串变量，该变量包含的内容就是需要执行的 MATLAB 指令，而函数的返回值为一个整数，当返回值为 0 时，表示 MATLAB 指令成功的执行了，否则表示没有成功执行。在下面的代码中使用该函数完成了曲线拟合的工作。

```
001    /* MEX 函数示例 —— curvefitting.c */
002    #include "mex.h"
003    /* MEX 函数入口 */
004    void mexFunction(int nlhs,mxArray *plhs[],
005                     int nrhs,const mxArray *prhs[])
006    {
007        const char *filename = "curvedata.mat";
008        char *cmd;
009        mxArray *K;
010        /* 判断输入参数 */
011        if(nrhs != 1)
```

```
012             mexErrMsgTxt("必须给一个输入参数！ ");
013         /* 设置命令行 */
014         cmd = mxMalloc(256);
015         sprintf(cmd,"load %s;",filename);
016         /* 执行命令 */
017         mexEvalString(cmd);
018         mexEvalString("clf;plot(x,y);grid on;");
019         /* 暂停 */
020         mexEvalString("disp('按任意键继续.....');pause;");
021         /* 设置工作空间中的变量 */
022         mexPutVariable("base","Order",prhs[0]);
023         /* 执行曲线拟合 */
024         mexEvalString("K = polyfit(x,y,Order);");
025         /* 再次计算拟合数据 */
026         mexEvalString("yk = polyval(K,x);hold on;plot(x,yk,'r:*');");
027         sprintf(cmd,"legend('Origin','Fitted:%g')",
028                                 *mxGetPr(prhs[0]));
029         mexEvalString(cmd);
030         /* 获取拟合的结果 */
031             plhs[0] = mexGetVariable("base","K");
032     }
```

在 curvefitting.c 文件中还使用了 mexPutVariable 函数和 mexGetVariable 函数，这两个函数完成的是 MEX 函数文件的获取或者设置 MATLAB 工作空间变量的功能，关于这两个函数的具体解释请参阅 MATLAB 的帮助文档。

编译并执行 curvefitting.c：

```
>> mex curvefitting.c
>> k8 = curvefitting(8)
按任意键继续……
k8 =
  Columns 1 through 4
   -0.0002    0.0088   -0.1363    1.0872
  Columns 5 through 8
   -4.6983   10.6344  -11.0814    3.4641
  Column 9
    0.9442
>> %查看工作空间中的变量
>> whos
  Name          Size                    Bytes  Class
```

K	1x9	72	double array
Order	1x1	8	double array
k8	1x9	72	double array
x	1x101	808	double array
y	1x101	808	double array
yk	1x101	808	double array

Grand total is 322 elements using 2576 bytes

曲线拟合得到的结果就是多项式的系数，拟合得到的函数曲线如图 3-11 所示。

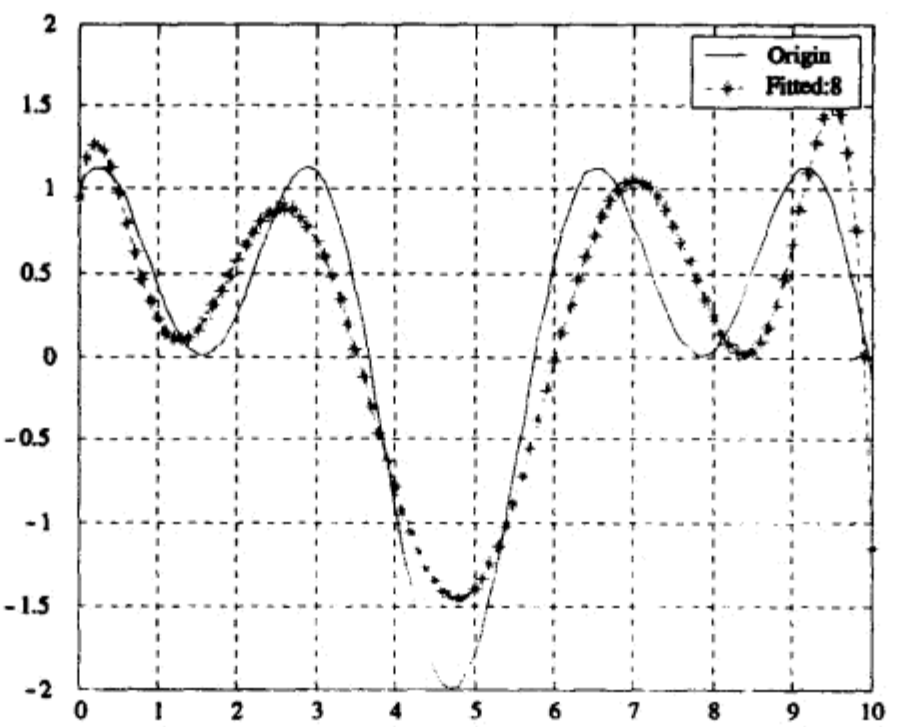


图 3-11　曲线拟合的结果

其实，通过 mexEvalString 函数执行 MATLAB 指令的效率是比较低的，所以不推荐经常使用 mexEvalString 函数，不过该函数也有优势，就是可以执行 MATLAB 的常用指令，例如 pause、keyboard、clear all 等，这些指令都可以通过该函数来完成调用。在例 3-9 的代码中，使用了另外一个 MEX 函数来完成对 MATLAB 指令的调用。

例 3-9　生成随机数——执行 MATLAB 函数。

```
/* MEX 函数示例 —— randomnum.c
   生成随机数 */
# include "mex.h"
/* MEX 函数入口 */
void mexFunction (int nlhs, mxArray * plhs[],
                  int nrhs, const mxArray * prhs[])
{
    double *m, *n, *flag;
    /* 错误检测 */
    if(nlhs >1 || nrhs != 3)
      mexErrMsgTxt("错误：输入输出参数个数错误！ ");

```

```
013             m = mxGetPr(prhs[0]);
014             n = mxGetPr(prhs[1]);
015             flag = mxGetPr(prhs[2]);
016
017             plhs[0] = mxCreateDoubleMatrix(*m,*n,mxREAL);
018             /* 设置运算结果标志 */
019             mexSetTrapFlag(*flag);
020             /* 错误的调用 --> 函数 rnd 并不存在 */
021             mexCallMATLAB(1,plhs,2,prhs,"rnd");
022             /*   程序继续运行 MEX 函数文件，
023                  于是 rand 函数被 MEX 文件调用。*/
024             mexPrintf(" 用户将 Flag 设置为 1. \n");
025              /* 正确调用 */
026             mexCallMATLAB(1,plhs,2,prhs,"rand");
027         }
```

在例 3-9 的源代码中使用 mexCallMATLAB 函数调用了 MATLAB 的随机数产生函数——rand。mexCallMATLAB 函数的定义如下：

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

其中，函数的输入参数 nlhs、plhs、nrhs、prhs 为需要输入到被调用函数的输入、输出参数设置，而 command_name 为被调用的函数名称。该函数的返回值为整数，如果返回值为 0，则代码函数被正确的执行了，否则函数的执行出现了错误。如果在执行 command_name 函数时出现了错误，则根据 mexSetTrapFlag 函数的设置进行错误处理，mexSetTarpFlag 函数的定义如下：

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

当函数的输入参数为 0 时，若 mexCallMATLAB 调用的函数发生了错误，则将程序的控制权返还给 MATLAB，也就是退出 MEX 函数的执行；当函数的输入参数为 1 时，若 mexCallMATLAB 调用的函数发生了错误，则将函数的控制权返还给 MEX 函数，也就是继续 MEX 函数的执行。

例如例 3-9 的 021 行和 026 行代码，分别使用 mexCallMATLAB 调用了两次求随机数的函数，其中 021 行的调用是错误的，因为在 MATLAB 中是没有 rnd 函数存在的。于是，根据 019 行设置的 flag 值，来判断是否退出当前 MEX 函数的执行。

编译并执行 randomnum.c：

```
>> mex randomnum.c
>> %运行 MEX 函数，flag = 0
>> y = randomnum(3,2,0)
??? Undefined function 'rnd'.
Error in ==> D:\Temp\randomnum.dll
>> %上面的函数调用没有成功
```

```
>> %再次运行 MEX 函数，flag = 1
>> y = randomnum(3,2,1)
??? Undefined function 'rnd'.
 用户将 Flag 设置为 1.
y =
    0.4057    0.4103
    0.9355    0.8936
    0.9169    0.0579
```

从上面的操作可以看出，mexSetTrapFlag 函数发挥了不同的作用。请读者根据自己的需要设置相应的错误控制方法。

例 3-10　设置图像对象属性。

众所周知，MATLAB 的数据可视化能力是非常强大的，通过简单的绘图指令就可以完成美观的数据分析、可视化的工作。特别是利用句柄图形的功能，还能够通过编程序完成图形对象属性的修改，从而响应用户的输入，完成图形用户界面应用程序的开发。

在 MEX 函数文件中，同样也可以完成图形对象属性的获取和修改，这就要通过 mexSet 函数或者 mexGet 函数来完成，这两个函数的作用就是 MATLAB 中 set 函数和 get 函数的功能。具体的函数意义和解释请参阅 MATLAB 的帮助文档。

下面是本例子的 C 源代码：

```
001     /* MEX 函数示例 —— modifyfig.c
002        定义图形对象属性 */
003     # include "mex.h"
004     /* MEX 函数入口 */
005     void mexFunction(int nlhs, mxArray * plhs[],
006                      int nrhs, const mxArray * prhs[])
007     {
008         const mxArray *mx_line, *mx_marker;
009         mxArray   *mx_newmarker;
010         double d_line;
011         char *marker, *newmarker;
012         int flag;
013
014         /* 错误检测 */
015         if (nlhs > 0 || nrhs != 1)
016             mexErrMsgTxt("错误：输入输出参数错误！");
017
018         /* 从工作空间中获取变量 */
019         mx_line = mexGetVariablePtr("base","h_line");
020         d_line = mxGetScalar(mx_line);
```

```
021
022         /* 获取当前的属性 */
023         mx_marker = mexGet(d_line,"marker");
024         /* 类型转换 */
025         marker = mxArrayToString(mx_marker);
026         /* 显示当前属性 */
027         mexPrintf("\n 当前的图形线条样式 : %s",marker);
028         /* 设置新的属性值 */
029         newmarker = mxGetChars(prhs[0]);
030         mx_newmarker = mxCreateString(newmarker);
031         /* 应用新的属性 */
032         flag = mexSet(d_line,"marker",mx_newmarker);
033         if (flag == 0)
034         {
035             mexPrintf("\n 新的图形线条样式 : %s",newmarker);
036             plhs[0] = mxCreateString("**** 图形线条样式设置操作成功 ****");
037         }
038     }
```

上面的代码主要说明了在 C 语言 MEX 文件中获取或者设置句柄图形属性的方法，为了正确执行上面的代码，需要在 MATLAB 中首先执行下面几条指令：

```
>> x = -10:0.1:10;
>> y = sin(x)+cos(2*x);
>> h_line = plot(x,y,'*')
h_line =
    3.0010
```

这时得到的图形如图 3-12 所示。

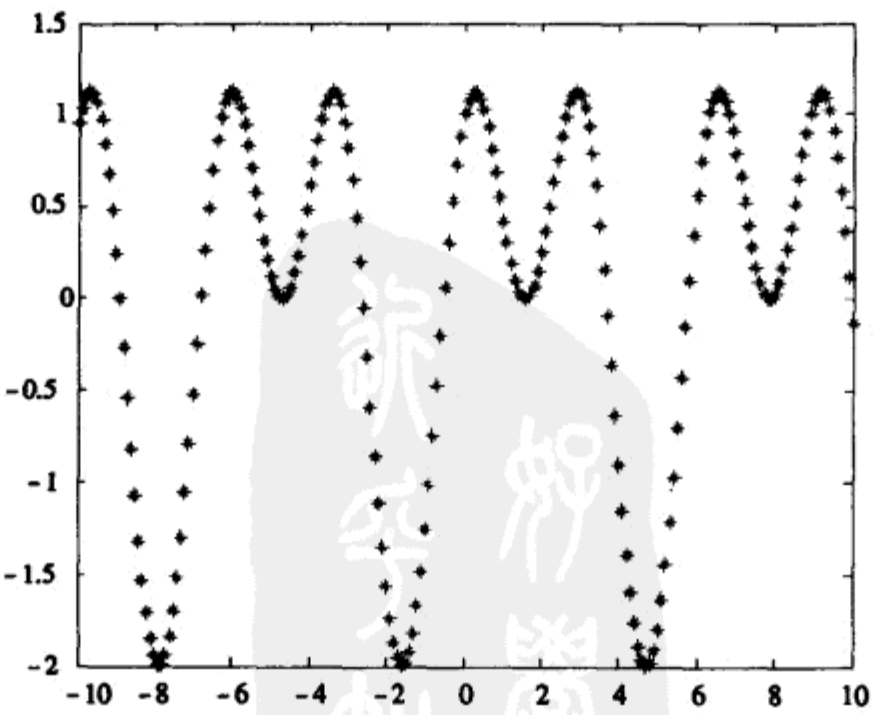


图 3-12　原始的图形

然后继续编译执行 modifyfig.c：

```
>> mex modifyfig.c
>> modifyfig('O')
```

当前的图形线条样式为：“*”，新的图形线条样式为“O”。

```
ans =
**** 图形线条样式设置操作成功 ****
```

得到的图形如图 3-13 所示。

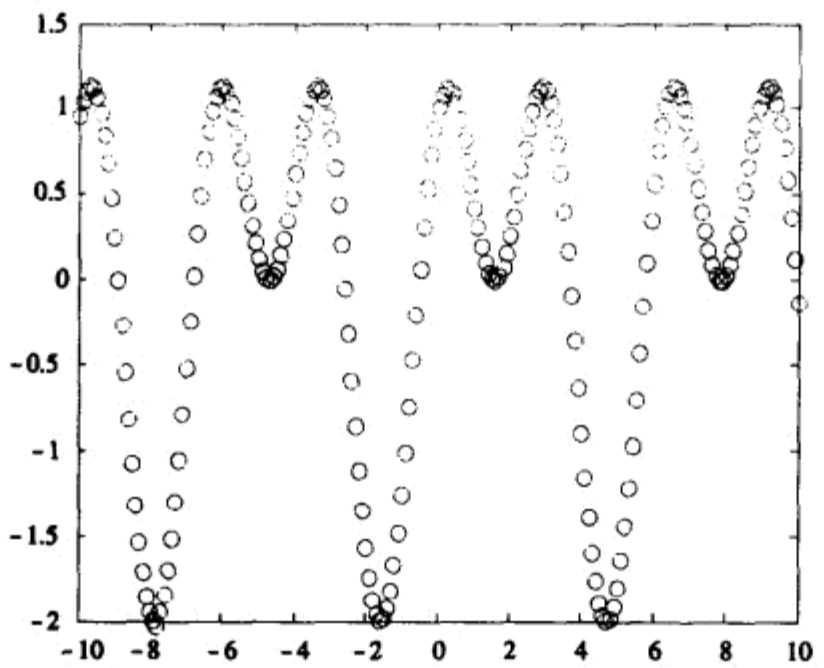


图 3-13　设置图形属性之后的图形

例 3-10 演示了在 MEX 文件中获取和设置图形对象属性的方法，相信读者已经基本了解在 MEX 文件中操作图形对象的方法了。

例 3-11　访问设备硬件。

创建 MEX 文件的目的之一就是为 MATLAB 增加访问系统设备硬件的能力，例 3-11 使用 C 语言的 MEX 文件使 MATLAB 能够访问 Windows 平台下的多媒体声音设备，函数将设备的一些信息通过 Windows 的多媒体接口函数进行获取并且显示出来，下面是该文件的代码：

```
001  /*************************************************************
002      snd_mixer_info.c
003      在 MATLAB 中获取 Windows 多媒体声音设备信息
004      使用了 Windows SDK 多媒体库函数
005
006      编译：
007      mex snd_mixer_info.c Winmm.lib
008  *************************************************************/
009  #include "mat.h" /* Must include mat.h before mex.h */
010  #include "mex.h" /* because mat.h #includes stdio.h */
011  #include <windows.h>
012  #include <mmsystem.h>
```

```
#include <stdio.h>
#include <string.h>
/* 入口函数 */
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
    UINT                    mixer_ID,i,j,k,l,connections;
    HMIXEROBJ               hmx;
    MIXERCAPS               mxcaps;
    MIXERLINE               mxl;
    MIXERLINECONTROLS   mxlc;
    MIXERCONTROL        pamxctrl[15];
    MIXERCONTROLDETAILS     mxcd;
    MIXERCONTROLDETAILS_LISTTEXT  padetailstext[30];
    MIXERCONTROLDETAILS_SIGNED        padetailssigned[30];

    /* 判断输入参数 */
    if (nrhs > 1)
        mexErrMsgTxt("Too many input parameter!");
    if (nrhs > 0){
        mixer_ID = (UINT)mxGetScalar(prhs[0]);
    }
    else{
        mixer_ID = 0;
    }

    if ((mixer_ID + 1) > mixerGetNumDevs()){
        mexErrMsgTxt("Remenber: MIXER_IDs start with 0!");
    }
    /* 获取混音器信息 */
    if (mixerOpen((LPHMIXER)&hmx,mixer_ID,0,0,0))
        mexPrintf("error opening mixer !!!!");
    /* 输出信息 */
    if (nlhs == 1){
        if (mixerGetDevCaps((UINT)hmx, &mxcaps, sizeof(mxcaps)))
            mexPrintf("!!!! error getting info !!!!");
        plhs[0]=mxCreateString(mxcaps.szPname);
        return;
```

```
051         }
052         else
053             mexPrintf("There are %d    active mixer devices in the system. \n \n",
054                     mixerGetNumDevs());
055         /* 设备的详细信息 */
056         if (mixerGetDevCaps((UINT)hmx, &mxcaps, sizeof(mxcaps)))
057             mexPrintf("!!!! error getting info !!!!");
058         mexPrintf("cDestinations is: %ld\n",mxcaps.cDestinations);
059         mexPrintf(strcat(mxcaps.szPname,"\n\n"));
060         mexPrintf("Driver Version: is: %d.%d \n",
061                     (char)(mxcaps.cDestinations >> 8),\
062                     (char)mxcaps.cDestinations);
063         /* 线路的详细信息 */
064         mxl.cbStruct = sizeof(mxl);
065         mxlc.cbStruct = sizeof(mxlc);
066         for (i=0; i < mxcaps.cDestinations; i++){
067             mxl.dwDestination = i;
068             if (mixerGetLineInfo(hmx, &mxl,
069                 MIXER_GETLINEINFOF_DESTINATION))
070                 mexPrintf("!!!! error getting info !!!!");
071             mexPrintf("\n======================\nDestination %ld:    ",i);
072             mexPrintf(strcat(mxl.szName,"    ("));
073             mexPrintf(strcat(mxl.szShortName,")\n"));
074             mexPrintf("    dwLineID is: %ld\n",mxl.dwLineID);
075             mexPrintf("    cChannels is: %ld\n",mxl.cChannels);
076             mexPrintf("    cConnections is: %ld\n",mxl.cConnections);
077             mexPrintf("    cControls is: %ld\n       ",mxl.cControls);
078
079             mxlc.dwLineID = mxl.dwLineID;
080             mxlc.cControls = mxl.cControls;
081             mxlc.cbmxctrl = sizeof(pamxctrl[0]);
082             mxlc.pamxctrl = pamxctrl;
083             if (mixerGetLineControls(hmx,&mxlc,
084                 MIXER_GETLINECONTROLSF_ALL))
085                 mexPrintf("!!!! error getting info !!!!");
086             for (j=0; j < mxl.cControls; j++){
087                 mexPrintf("\n\n - Control %ld:    ",j);
088                 mexPrintf(strcat(mxlc.pamxctrl[j].szName,"\n       "));
```

```
089                     mexPrintf("dwControlID is: %ld\n   ",mxlc.pamxctrl[j].dwControlID);
090                     mexPrintf("cMultipleItems is: %ld\n      ",
091                             mxlc.pamxctrl[j].cMultipleItems);
092                     mexPrintf("lMninmum is: %ld\n      ",
093                             mxlc.pamxctrl[j].Bounds.lMinimum);
094                     mexPrintf("lMaximum is: %ld\n      ",
095                             mxlc.pamxctrl[j].Bounds.lMaximum);
096                     mexPrintf("cSteps is: %ld\n      ",mxlc.pamxctrl[j].Metrics.cSteps);
097                     mexPrintf("fdwControl is: %ld\n          ",
098                             mxlc.pamxctrl[j].fdwControl);
099                 }
100             }
101             mixerClose((HMIXER)hmx);
102             return;
103         }
```

由于例 3-11 的代码使用了特殊的函数库——Winmm.lib，所以在编译该文件时需要在命令行中增加特殊的函数库，方法如下：

```
mex snd_mixer_info.c Winmm.lib
```

运行该 MEX 函数，将显示当前操作系统下安装的多媒体设备信息：

```
>> mixname = snd_mixer_info
mixname =
SoundMAX Digital Audio
>> snd_mixer_info
There are 1    active mixer devices in the system.
cDestinations is: 2
SoundMAX Digital Audio
Driver Version: is: 0.2
==========================================
Destination 0:    Volume Control    (Volume Control)
   dwLineID is: -65536
   cChannels is: 2
   cConnections is: 7
   cControls is: 2

 - Control 0:    Master Volume
      dwControlID is: 1
      cMultipleItems is: 0
      lMninmum is: 0
```

lMaximum is: 65535
cSteps is: 59
fdwControl is: 0

- Control 1: Mute
dwControlID is: 2
cMultipleItems is: 0
lMninmum is: 0
lMaximum is: 1
cSteps is: 0
fdwControl is: 1

==========

Destination 1: Recording Control (Recording Contr)
dwLineID is: -65535
cChannels is: 1
cConnections is: 6
cControls is: 1

- Control 0: Mux
dwControlID is: 0
cMultipleItems is: 6
lMninmum is: 0
lMaximum is: 5
cSteps is: 6
fdwControl is: 3

有兴趣的读者可以自己增加新的代码，使 MATLAB 能够直接读写以及播放各种多媒体的文件。

3.6 调试 MEX 文件

编写 MEX 文件的时候不可能一次成功，所以调试 MEX 文件就是非常必要的工作。现在 MATLAB 提供了在不同平台下调试 MEX 文件的能力，其中针对 UNIX 平台和 Windows 平台需要使用不同的方法，因为在 UNIX 平台上一般不具备可视化的集成开发环境，所以只有通过命令行进行调试，而在 Windows 平台上，则可以利用 Visual Studio 5 或者 6，也可以使用其它的 C/C++开发环境完成 MEX 文件的可视化调试。本小节将介绍 MEX 文件调试的方法。

注意：

如果需要生成可调试的 MEX 文件时，必须在命令行中使用下面的命令：

mex -g mexcfile.c

只有使用命令行参数 g 后，在 MEX 文件中才包含了调试信息，才能够进行函数调试。

3.6.1　在 Windows 平台上调试 MEX 文件

在 Windows 平台上进行 MEX 文件的调试可以利用可视化的集成开发环境来完成，不过首先要生成包含调试信息的 MEX 文件才可以。当 MEX 文件调试通过之后，才能再次编译 MEX 文件，生成可以发布的 MEX 文件。本小节以 Microsoft Visual Studio 6 为例说明在 Windows 平台下进行 MEX 文件调试的方法。这里利用的 MEX 文件是在例 3-2 中使用的 MEX 文件——simple.c。

首先，重新编译 simple.c，使用-g 命令行开关在生成的 MEX 文件中添加必要的调试信息：

```
>> mex -g simple.c
```

然后在 MATLAB 命令行窗口中，键入下面的指令，这条指令将启动 Microsoft Visual Studio，并且在 Visual Studio 中以项目工程的形式打开 Simple.dll：

```
>> !msdev simple.dll &
```

提示：

这一步操作也可以在 Windows 平台的控制台方式(命令行提示符)下完成。

接着执行“Project”菜单下的“Setting”命令，在弹出的对话框中选择“Debug”页，在该对话框中设置工程的调试属性，如图 3-14 所示。

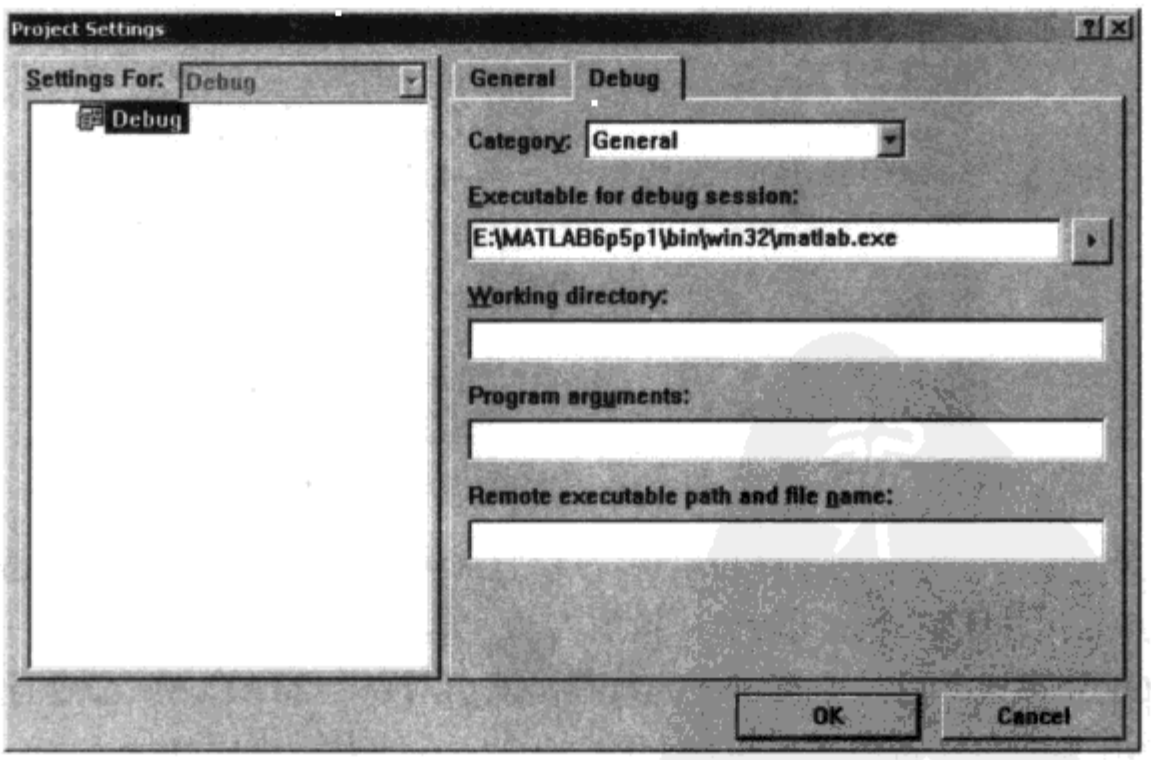


图 3-14　设置调试的属性

在“Executable for debug session”文本框中，设置属性为 MATLAB 的可执行程序——E: \MATLAB6p5p1\bin\win32\matlab.exe。设置完毕后，单击“OK”按钮结束工程调试属性的设置。

执行“File”菜单下的“Open”命令，在弹出的打开文件对话框中选择“simple.c”文件，simple.c 文件是需要被调试的源代码文件。打开的文件对话框如图 3-15 所示。

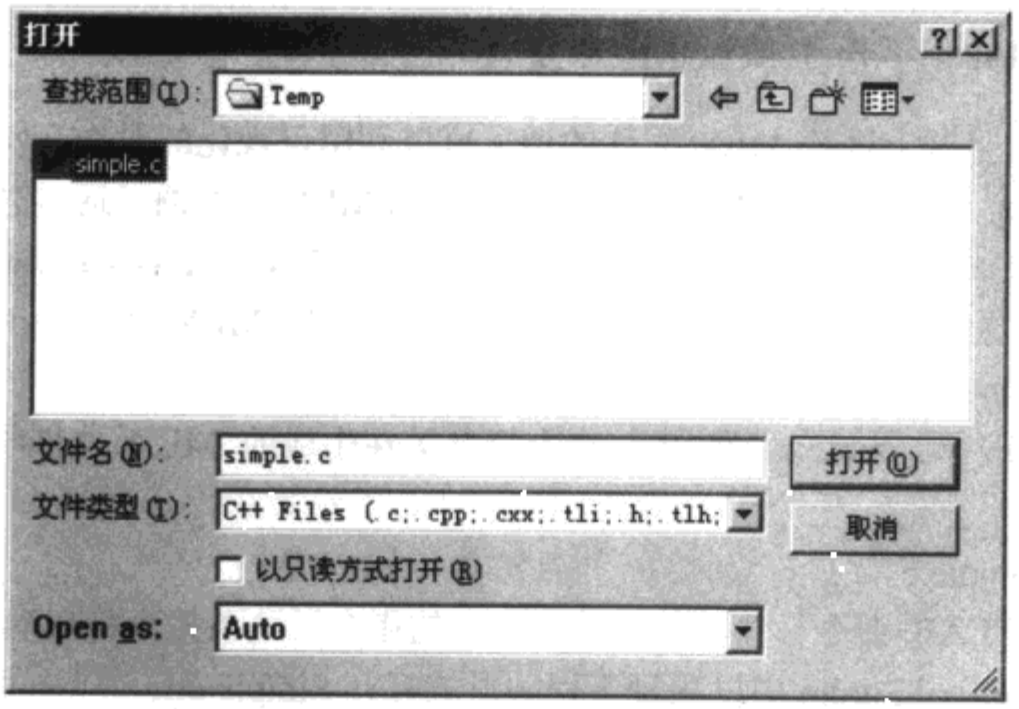


图 3-15　打开需要调试的源代码文件

单击对话框的“打开”按钮，确认打开源代码文件，然后在源代码文件必要的代码行中设置断点。设置断点的方法是将光标移动到需要设置断点的代码行上，然后按 F9 键就可以在代码行上设置断点了。

设置断点之后就可以开始进行程序的调试了。执行“Build”菜单下的“Start Debug”子菜单下的“Go”命令，或者直接使用 F5 快捷键，这时将弹出一个信息对话框，如图 3-16 所示。

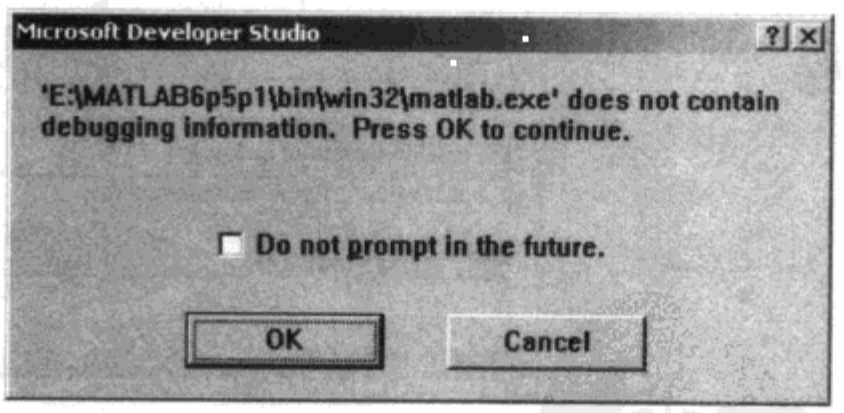


图 3-16　信息对话框

在此对话框上包含的信息是提示用户：MATLAB 的可执行程序 matlab.exe 本身不包含任何调试信息。用户不必理会此信息，按照提示，单击“OK”按钮继续。

单击“OK”按钮之后，Windows 系统将再次启动一个 MATLAB 命令行，在新启动的那个 MATLAB 命令行中按照平时执行 MEX 函数文件的方法执行 simple.dll：

```
>> simple
```

这时系统将自动进入到 Microsoft Visual Studio 中，并且在设置的断点处暂停程序的运行，

下面就可以利用 Visual Studio 提供的便利的调试手段来完成程序的调试了。当结束调试时，新启动的 MATLAB 会话行会自动关闭。

注意：

关于如何使用 Microsoft Visual Studio 进行代码调试已经超出了本书介绍的内容范围，感兴趣的读者可以阅读 Microsoft Visual Studio 的相关文档或者市面上介绍 Visual C++ 的书籍。

3.6.2　在 UNIX 平台上调试 MEX 文件

这里再简要介绍一下在 UNIX 平台上进行 MEX 文件调试的方法。如果需要在 UNIX 平台上调试 MEX 文件，则需要在 UNIX 的调试器中启动 MATLAB，然后在 MATLAB 环境下加载调试模型，启动调试工具就可以完成设置断点、查看内存变量等操作了。例如，在 Solaris 操作系统下，进行如下操作：

首先生成具有调试信息的 MEX 文件，mex –g filename.c。

然后在 Solaris 调试器中启动 MATLAB，matlab –Ddbx。

当调试器将 MATLAB 加载到内存之后，就可以像正常调用 MEX 文件一样执行 MEX 文件。打开 MEX 文件的源代码，设置断点，完成调试工作，也就是完成下面的工作：

```
>> my_CMEX_File   % 调用 my_CMEX_File.mexsol 进入 <dbx>
<dbx> cont   % 调试器继续命令
<dbx> stop   % 调试器终止命令
```

对于其它的 UNIX 平台，则需要使用 dbmex 指令，在 MATLAB 中设置断点模式：

```
>> dbmex on      % 打开调试模式
>> dbmex print   % 输出调试信息
>> dbmex stop    % 停止调试，返回到<dbx>
>> dbmex off     % 关闭调试器
```

有关在 UNIX 平台上完成 MEX 文件的调试请参阅 MATLAB 的帮助文档说明，同时还需要仔细阅读相应 UNIX 平台上的调试工具使用说明。

3.7　本 章 小 结

在本章详细介绍了 C 语言 MEX 文件的创建方法，仔细介绍了 MEX 文件的基本结构和输入、输出参数的设置，并且着重讲解了创建 MEX 文件的方法——使用 MEX 指令或者使用集成开发环境。本章通过大量详细的示例说明了创建 MEX 文件及使用 MEX 函数的基本方法，特别是讲述了在 MEX 文件内存管理中需要注意的问题，最后还简要介绍了 MEX 文件的调试方法。本章内容是全书的重点，希望读者仔细阅读本章包含的 C 代码示例程序，做到真正掌握 MATLAB 的 C 语言 MEX 文件的开发方法。

掌握 MEX 函数的使用方法和 MEX 文件的工作流程就基本掌握了 MEX 文件的使用，这里用图表的形式表示 C 语言的 MEX 文件的基本工作流程，如图 3-17 所示。

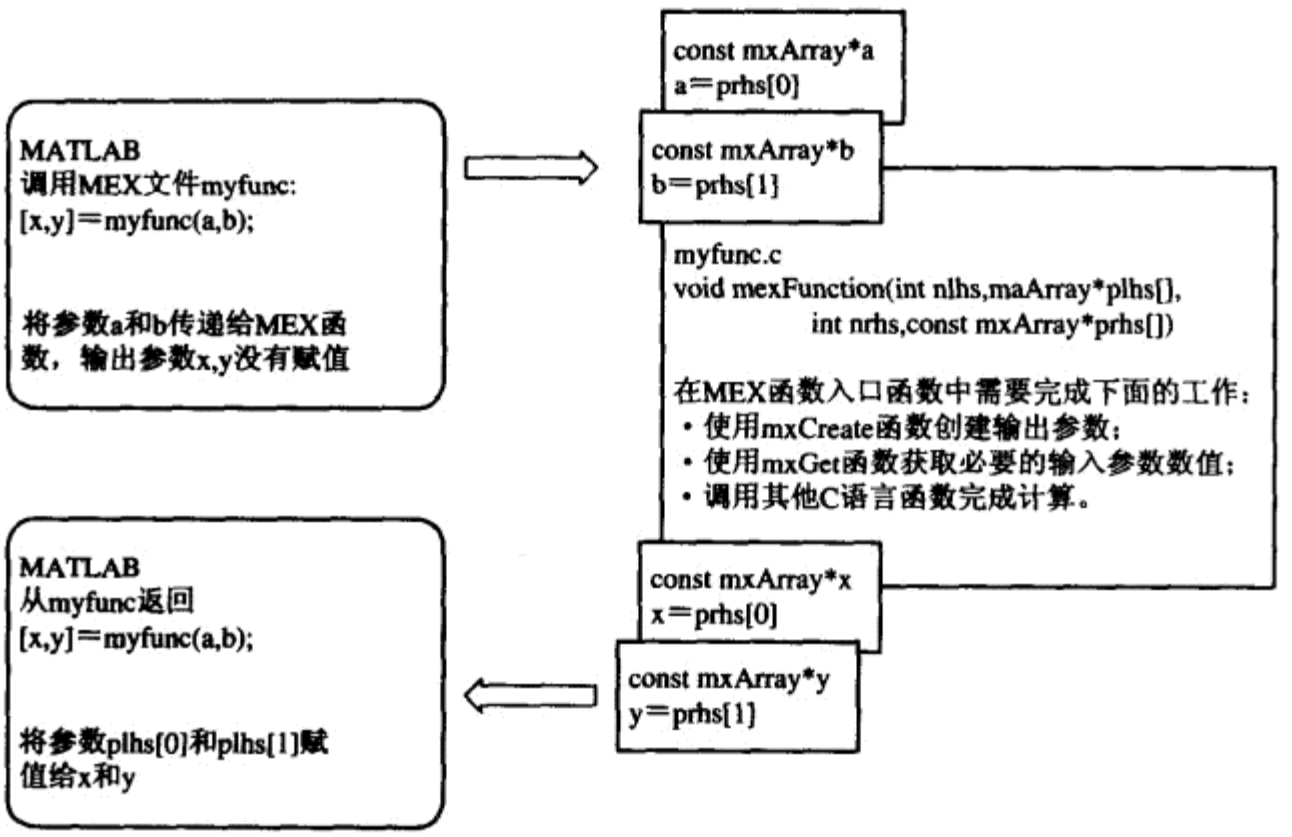


图 3-17　MEX 文件的工作流程

练　习

1. 比较使用 MEX 指令完成编译与在集成开发环境中进行 MEX 文件编译的异同。

2. 尝试使用自己熟悉的集成开发环境进行 MEX 文件的调试。

3. 编写程序完成下列功能：

编写函数，根据输入的变量输出不同的信息，需要输出的信息包括：

- 类型名称。
- ClassID。
- 行数。
- 列数。

该函数还需要完成下列功能：

- 返回操作是否成功的信息。
- 能够处理字符串数据和数值数组。

编译该函数时使用命令行编译和集成开发环境编译。

提示：

需要使用 mxGetClassName、mxGetClassID、mxGetM、mxGetN、mexErrMsgTxt 等函数。

4. 编写程序完成下列功能：

编写函数，根据输入的参数输出不同的信息，要求在 Windows 信息对话框中输出类型的名称，如图 3-18 所示。

图 3-18　参数的信息

提示：

使用 Windows 的 MessageBox 函数输出参数的信息，该函数的定义如下：

```
int MessageBox(HWND hWnd,
    LPCTSTR lpText,
    LPCTSTR lpCaption,
    UINT uType
);
```

具体的函数解释请参阅 Windows Platform SDK 的说明。

第 4 章　创建 Fortran 语言 MEX 文件

使用 Fortran 语言创建 MEX 文件的方法和过程与创建 C 语言 MEX 文件的方法和过程几乎一样，无非使用的高级编程语言有所不同而已。Fortran 语言在科学计算方面的应用比 C 语言要广泛，甚至 MATLAB 软件的前身也是使用 Fortran 语言编写的，比如 MATLAB 的数组元素排序和 Fortran 语言的数组元素排序就是一样的——以列元素优先。本章将详细介绍 Fortran 语言 MEX 文件的创建方法。

本章重点内容

- Fortran 语言 MEX 文件的结构；
- 在 MEX 文件中表示 MATLAB 数据；
- MEX 文件的可视化创建与调试。

提示：

如果读者对 C 语言 MEX 文件非常熟悉，那么学习 Fortran 语言 MEX 文件应该没有任何障碍，因为两者有很多地方非常相似。本章就忽略了有关内存管理等方面的话题，因为这些话题在第 3 章中已经作了详细的介绍。另外，如果读者不了解 Fortran 语言则需要阅读有关 Fortran 语言的教材或者文档，如果在日常工作中很少用到 Fortran 语言，则可以跳过本章的内容，继续后面 MATLAB 外部接口编程的学习。

4.1　MEX 文件简介

Fortran 语言的 MEX 文件顾名思义就是使用 Fortran 语言编写的 MEX 函数文件，和 C 语言的 MEX 文件相比较，两者功能非常相似，但是两种语言的优势决定了两种 MEX 文件应用范围各不相同。C 语言本身很灵活，是一种低级的高级编程语言，特别提供了指针用于内存的分配和管理，所以 C 语言可以完成各种应用，因此 C 语言 MEX 文件在 MATLAB 中的用处也最大，例如可以完成对计算机硬件或者操作系统任务的调度，还可以为 Real-Time Workshop 开发系统实时目标。Fortran 语言本身是一种科学计算的高级编程语言，用这种语言开发出来的程序多数用于数值计算方面，它的数值稳定性和多种数值变量是 C 语言无法匹敌的。但是 Fortran 语言本身对指针的操作比较繁琐，甚至有些 Fortran 语言编译器根本不支持指针操作，因此使用 Fortran 语言进行 MEX 文件编程时必须进行一些额外的操作，要求程序员必须按照严格的程序格式书写。

4.1.1　简单的 MEX 文件示例

在详细介绍 Fortran 语言 MEX 文件的编写方法之前，首先查看一个简单的 Fortran 语言

MEX 源文件，通过该文件了解 Fortran 语言 MEX 文件的基本创建过程。下面是该文件的源代码。

例 4-1　简单的 Fortran 语言 MEX 文件——mexHelloWorld.f。

```
001      C       入口函数
002              subroutine mexFunction(nlhs, plhs, nrhs, prhs)
003      C-------------------------------------------------------
004      C    参数声明
005              integer plhs(*), prhs(*)
006              integer nlhs, nrhs
007      C    代码行
008              call mexPrintf('Hello MATLAB World!')
009      C    函数文件尾部
010              return
011              end
```

和例 3-1 类似，例 4-1 的代码功能也是在 MATLAB 命令行窗口中输出一段文字。

首先请读者使用任何一种能够编辑纯文本文件的编辑器，将上面的代码键入到文件中。注意，不要将行号(001～011)也敲进去，在这里设置行号的主要目的是为了便于讲解和分析程序。

然后将该文件保存，设置文件名称为 mexHelloWorld.f (Fortran 语言源文件的文件扩展名为.f)，将该文件拷贝或者保存到 MATLAB 的当前工作路径下。

接着，在 MATLAB 的命令行窗口下键入下面的指令：

```
>> mex mexHelloWorld.f
```

 注意：

在进行 Fortran 语言 MEX 文件编译之前，一定在 MATLAB 环境下完成 MEX 文件编译器的配置，并且选择合适的 Fortran 语言编译器。在 Windows 平台下，MATLAB 支持的 Fortran 语言编译器类型远没有所支持的 C 语言编译器类型丰富，只有 Compaq Visual Fortran 和 Digital Visual Fortran 两种。在本书编写的过程中，使用的 Fortran 语言开发环境是 Compaq Visual Fortran。MATLAB 所支持的编译器种类请参阅本书的附录 A。

如果编辑的应用程序没有任何错误，则 MATLAB 命令行中不会显示任何消息；否则，在 MATLAB 命令行中将显示相应的错误消息提示。如果出现编译错误，请仔细查看键入的源程序代码，修改错误直到编译通过为止。另外，再次强调配置编译器的时候选择 MATLAB 支持的 Fortran 语言编译器。

编译的结果是创建出 MATLAB 的 MEX 文件，可以在 MATLAB 命令行下键入 what 指令查看当前路径下是否具有 MEX 文件。

```
>> what
MEX-files in the current directory D:\Temp
mexHelloWorld
```

接着，运行创建的 MEX 文件，在 MATLAB 命令行中键入下面的指令：

```
>> mexHelloWorld
Hello MATALB World!
```

4.1.2 MEX 源文件的基本结构

花费一点时间查看一下 Fortran 语言 MEX 源文件的基本结构。Fortran 语言 MEX 源文件是标准的 Fortran 语言源文件，建议读者在创建 Fortran 语言 MEX 源文件的时候使用 Fortran 77 或者 Fortran 90 的语言标准，以符合不同 Fortran 编译器的要求，这样编写出来的源代码就会具有比较好的可移植性。

从程序的第 001 行开始到 006 行，都是 Fortran 语言 MEX 源文件入口函数的声明部分：

```
C       入口函数
        subroutine mexFunction(nlhs, plhs, nrhs, prhs)
C-------------------------------------------------------
C    参数声明
        integer plhs(*), prhs(*)
        integer nlhs, nrhs
```

mexFunction 函数类似 C 语言源文件的 main 函数，它是 MEX 文件的入口函数。当在 MATLAB 命令行中执行 MEX 函数时，MATLAB 解释器将从此函数处开始执行 MEX 代码。该入口函数的输入参数有四个，其意义分别是：

nlhs：表示输入参数的个数。

Plhs：mxArray 类型的指针数组，表示 MEX 函数的输入参数。

Nrhs：表示输出参数的个数。

Prhs：mxArray 类型的指针数组，表示 MEX 函数的输出参数。

其实上述输入、输出参数的名称非常容易记忆，lhs 代表 Left hand parameters，rhs 代表 right hand parameters，n 代表 number，p 代表 pointer。这里的左手和右手如何划分呢？例如在 MATLAB 中调用求伯特图的函数时，可以这样调用：

```
[mag,phase,w] = bode(sys)
```

这里以“=”为分界，左边的参数 mag、phase 和 w 为输出参数，即左手参数，而等号右边的参数 sys，为输入参数，即右手参数。

入口函数之后就是 Fortran 语言 MEX 源文件的程序主体部分了，在这里需要完成 MATLAB 数据的获取和内存的分配，调用相应的计算子程序或者函数，完成 MEX 文件的计算工作，最后将必要的输出返回给 MATLAB。

程序的结尾是 Fortran 语言的特别要求，必须有 return 和 end 关键字作为 Fortran 语言源程序的结束。

通过前面两章的介绍读者应该已经了解，MATLAB 的数据在 C 语言中使用 mxArray 数据类型来表示，但是在 Fortran 语言中则没有显性地定义该数据结构，于是在 Fortran 语言的 MEX 文件中通过一种所谓的“指针”类型数据完成 Fortran 语言和 MATLAB 之间的数据传递。MATLAB 将需要传递给 Fortran 程序的 mxArray 数据指针保存成为一个整数类型的变量，例如在 mexFunction 入口函数中声明的 prhs 和 plhs，然后在 Fortran 程序中，通过能够访问

指针的 Fortran 语言 mx 函数访问 mxArray 数据，获取 mxArray 数据结构中包含的实际数据。有关在 Fortran 语言中操作 MATLAB 数据的方法，将在 4.2 小节详细讨论。

4.2　管理 MATLAB 数据

相比较而言在 C 语言中管理 MATLAB 的数据是比较容易的，而在 Fortran 中，由于受到编程语言本身的限制，就没有 C 语言那么灵活的方法，不过 MATLAB 还是提供了相应的 mx 函数用于 Fortran 语言程序的开发。注意，这些 Fortran 语言的 mx 函数不仅可以用于 MEX 文件的编写，而且还可以用于其它的应用程序开发。

首先，查看一个 Fortran 语言 MEX 文件的应用实例，该例子是对例 4-2——simple.c 使用 Fortran 语言重新编写而得的。

例 4-2　简单的 MEX 文件示例——simple.f。

```
001      C=============================================================
002      C     Simple.f
003      C     A simple mex file demo by Fortran
004      C     20031010
005      C=============================================================
006      C      Numberic Assignment
007            subrcutine AssignNumericData(pr)
008            integer pr
009            real*8 rdata(6)
010            data rdata /1. ,2. ,3. ,4. ,5. ,6. /
011
012            call mxCopyReal8ToPtr(rdata,pr,6)
013
014            return
015            end
016
017      C     The gateway routine
018            subroutine mexFunction(nlhs, plhs, nrhs, prhs)
019      C-------------------------------------------------------------
020      C     (pointer) Replace integer by integer*8 on the DEC Alpha
021      C     64-bit platform.
022
023            integer plhs(*), prhs(*)
024            integer mxGetPr, mxCreateDoubleMatrix
025            integer pr
026      C-------------------------------------------------------------
```

```
027             integer nlhs, nrhs
028
029     C       Check for proper number of arguments.
030             if(nlhs .ne. 1) then
031                 call mexErrMsgTxt('One output required.')
032             endif
033
034     C       Create matrix for the return argument.
035             plhs(1) = mxCreateDoubleMatrix(2, 3, 0)
036     C       Get the point to the Output argument
037             pr = mxGetPr(plhs(1))
038     C       Call the Assignment Subroutine
039             call AssignNumericData(pr)
040     C       Finished!
041             return
042             end
```

例 4-2 的代码是更完整的 Fortran 语言的 MEX 函数文件应用，此程序可以分为如下部分：首先从程序的 007~015 行是赋值运算的子函数，该函数在入口函数中调用，调用的代码在 038 行。与一般的 C 语言 MEX 文件不同，C 语言的 MEX 文件直接利用 memcpy 函数完成数据的拷贝(复制)，而 Fortran 语言本身没有提供这样的函数，于是在 simple.f 的源代码中，使用了 mxCopy 函数完成 mxArray 数据和 Fortran 语言数据的交互。这个函数就是在 012 行调用的 mxCopyReal8ToPtr，该函数的定义如下：

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

其中，y 为双精度类型的实数数组，其元素的个数为 n，而 px 则为 mxArray 数据对象的数据指针，在 Fortran 语言中使用整数类型表示。该函数的功能就是将实数数组 y 中的元素都赋值给 mxArray 数据对象 px，与之对应的函数是 mxCopyPtrToReal8，该函数的说明请参阅 MATLAB 的帮助文档。

编译并运行例 4-2 的代码：

```
>> mex simple.f
>> simple
??? One output required.
Error in ==> D:\Temp\simple.dll
>> y = simple
y =
     1     3     5
     2     4     6
```

与 C 语言的 mx 函数不同，在 MATLAB 提供的 Fortran 语言 mx 函数中专门有十几个函

数用来完成 MATLAB 数据指针的操作，分别可以用来操作字符串、双精度数组以及稀疏矩阵等。例如在例 4-2 的代码中使用了操作双精度数据的 mxCopy 函数。另外，在 MATLAB 的帮助文档中也提供了很多丰富的 Fortran 语言的应用实例，分别说明了操作各种数据的方法，请读者仔细阅读相应的帮助文档信息。

注意：

如果读者使用的 Fortran 语言编译器支持%VAL 运算符，则可以使用%VAL 运算符完成数据指针的操作，就不用使用 mxCopy 函数了，详细的信息请参阅 MATLAB 的帮助文档或者用户选择的 Fortran 编译器说明文档。

4.3　可视化创建 MEX 文件

可视化创建 Fortran 语言的 MEX 文件需要使用可视化的 Fortran 开发环境，这里笔者推荐使用 Compaq Visual Fortran，它的图形化开发界面和 Visual Studio 6 的界面保持一致，使用起来也和 Visual C++ 6 一样，几乎可以做到无缝的过渡，也就是熟悉使用 Visual C++ 6 的用户，基本上不需要什么额外的工作，就可以利用熟悉的 Fortran 语言开发应用程序了。

本小节介绍使用 Visual Fortran 创建 MEX 文件的方法，同时简要介绍了利用该图形界面工具调试 Fortran 语言 MEX 文件的方法。

首先运行 Compaq Visual Fortran，然后通过“File”菜单下的“New”命令，创建 Fortran 语言的动态链接库应用程序项目，如图 4-1 所示。

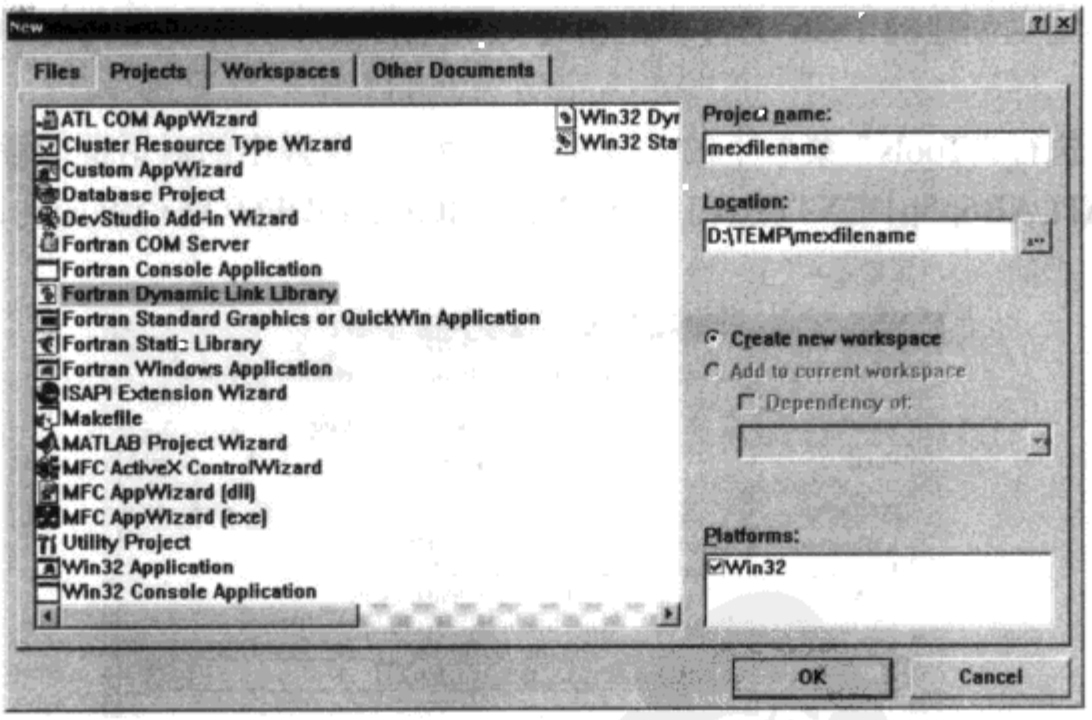


图 4-1　创建 Fortran 语言项目文件

在“Project name”处设置项目的名称为“mexfilename”，单击“OK”按钮创建新的工程项目。

注意：

如果用户的计算机上安装了 Visual C++和 Compaq Visual Fortran ，则两种开发工具共用一种开发界面，不过在具体开发应用程序的时候，分别使用自己的高级语言编译器。

然后，在弹出的对话框中选择“An empty DLL application”单选框，如图 4-2 所示。

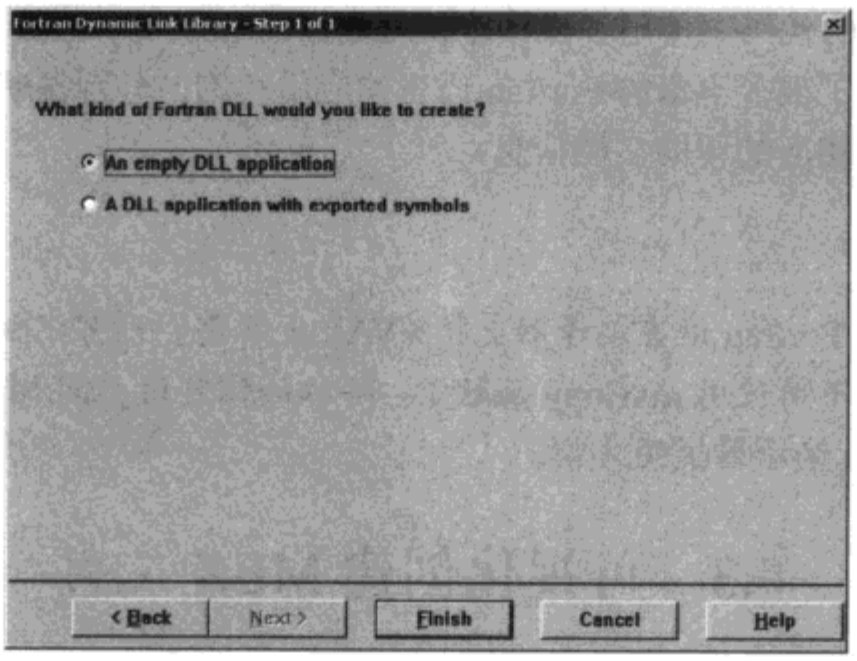


图 4-2 选择创建空 DLL 文件应用

单击“Finish”按钮，完成项目的创建。

创建新的项目之后，则需要向项目中添加资源文件——mexversion.rc，该文件位于%MATLABROOT%\extern\include 路径中，该文件定义了 MATLAB 的 MEX 文件版本信息，然后再添加 MEX 源文件，例如 simple.f。

第三步，创建 DEF 文件，该文件的内容如下：

```
LIBRARY MYFILE.DLL
EXPORTS _MEXFUNCTION@16
```

注意，在这里创建的 DEF 文件和在创建 C 语言 MEX 文件时定义的 DEF 文件的内容不同。

第四步，执行“Tools”菜单下的“Options”命令，在弹出的对话框中设置 Library 的路径，将 E:\MATLAB6p5p1\EXTERN\LIB\WIN32\DIGITAL\DF60 路径添加到该对话框中，如图 4-3 所示。

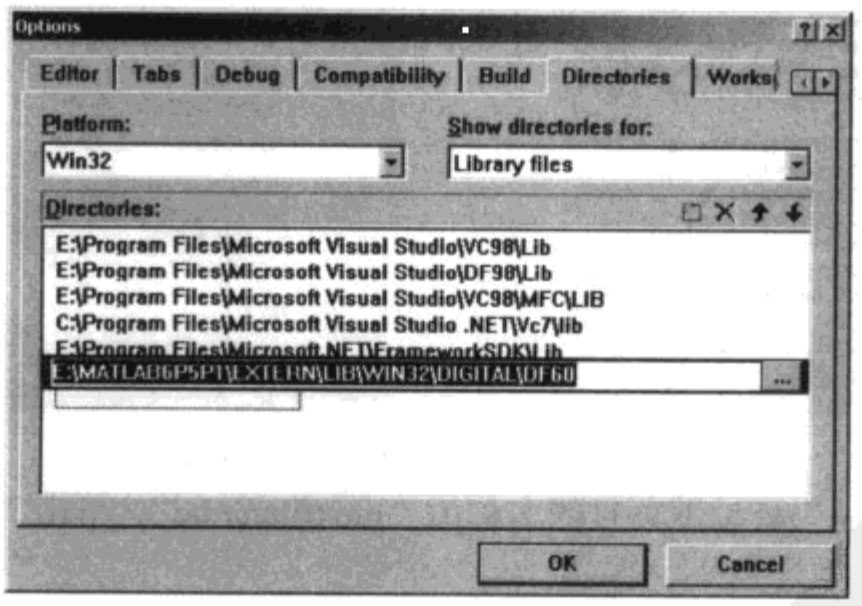


图 4-3 设置 Library 路径

按 Alt+F7 键，或者执行“Project”菜单下的“Setting”命令，在弹出的对话框中将必要的 Lib 文件添加在 Link 页的属性设置中。所谓必要的 Lib 文件，主要是 mx 函数的 libmx.lib 和 mex 函数的 libmex.lib，如果使用 MAT 数据文件应用函数，则还需要将 libmat.lib 文件添

加到属性中。关于 MAT 数据文件的应用将在第 5 章中详细讲解。设置 Link 属性对话框，如图 4-4 所示。

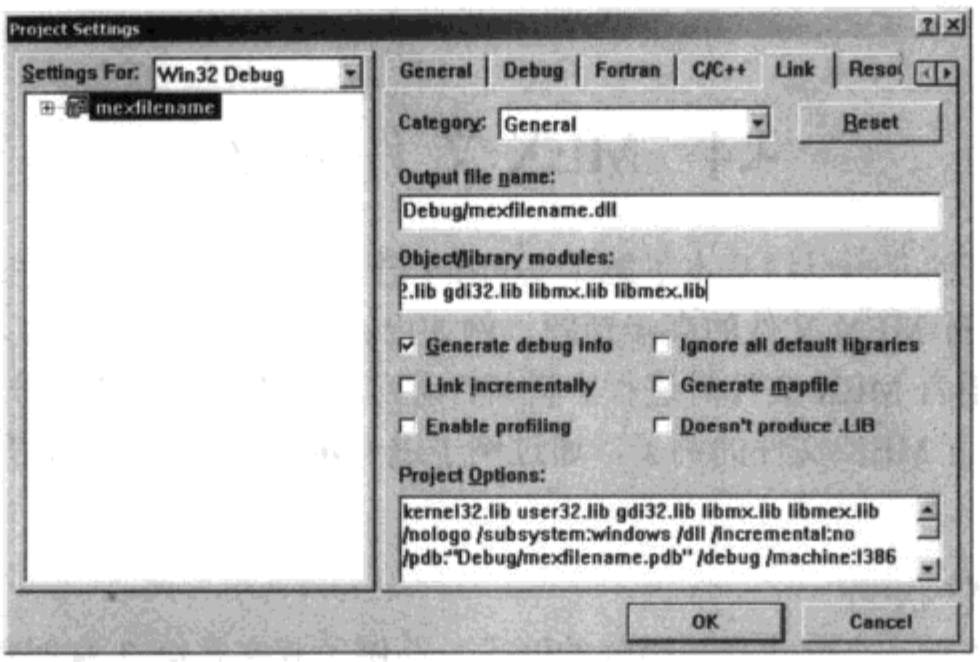


图 4-4　设置 Link 属性，添加必要的 Lib 文件

单击“OK”按钮结束属性的设置，然后，设置 C/C++ 预处理编译命令，设置该属性为 MATLAB_MEX_FILE。

接着就可以直接按 F7 键，或者执行“Build”菜单下的“Build mexfilename.dll”命令，来完成 MEX 文件的创建。

提示：

如果读者使用其它的集成开发环境进行 MEX 文件的编译，则可以仿照上面使用 Visual Studio 创建 MEX 文件的过程完成自己的工作。其中，比较关键的是需要正确安装集成开发环境，并且向 Windows 环境正确注册系统环境变量(头文件、库文件的路径等)。然后按照上面提示的步骤，正确设置项目类型(生成动态库)和 DEF 文件、库文件和预编译器定义，最后就是链接正确的库文件。一般的来说，可以使用其它的 Fortran 语言开发工具编译生成 MEX 文件，不过需要使用相应语言开发工具的 Lib 工具将 MATLAB 提供的相应的 DEF 文件编译成为库文件，然后添加到工程项目中。

如果读者需要详细了解在集成开发环境中创建 MEX 文件的方法，或者需要了解在 UNIX 环境下自定义编译链接 MEX 文件的方法，请参阅 MATLAB 的帮助文档。

如果用户需要在 Compaq Visual Fortran 的开发环境中进行 Fortran 语言的 MEX 文件可视化调试，则可以在创建完毕工程项目后，在源文件中需要设置断点的位置设置断点，然后执行“Build”菜单下“Start Debug”子菜单下的“Go”命令，或者直接按 F5 键，这时 Compaq Visual Fortran 将弹出对话框，如图 4-5 所示。

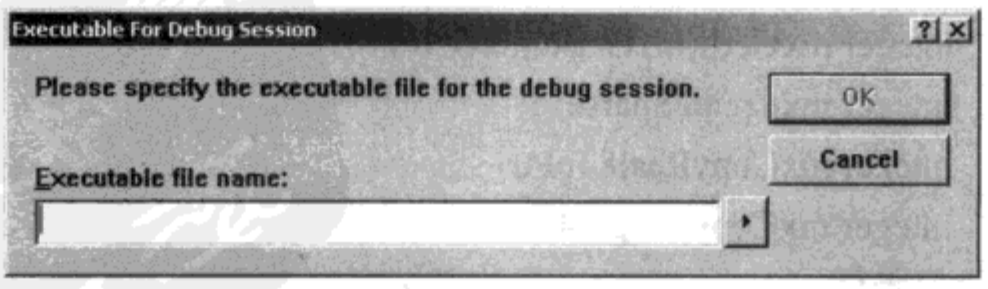


图 4-5　设置可执行文件的对话框

在此对话框中，将 MATLAB 的可执行文件输入进去，也就是在文本输入框中输入 %MATABROOT%\bin\win32\matlab.exe，单击“OK”按钮后，就可以开始 MEX 文件的调试了。

4.4 MEX 文件示例

通过前面小节的介绍已经基本了解了 Fortran 语言 MEX 文件的创建方法，本章没有详细介绍 Fortran 语言 MEX 文件的高级话题，例如内存管理等方面的内容，这些内容都已经在第 3 章介绍 C 语言 MEX 文件时进行了讲解，请读者参阅第 3 章的部分内容。本小节将演示一些 Fortran 语言 MEX 文件的例子，通过例子进一步学习 Fortran 语言 MEX 文件的创建方法。

例 4-3　创建稀疏矩阵。

首先向读者演示一个创建稀疏矩阵的例子，该例子其实是第 2 章创建稀疏矩阵 C 语言例子的翻版，这里使用 Fortran 语言将其重新写了一遍，通过这个例子重点了解一下 mxCopy 函数的具体使用方法，下面是该例子的源代码：

```
001      C   fcreatesparse.f
002      C   创建稀疏矩阵
003      C   入口函数
004          subroutine mexFunction(nlhs, plhs, nrhs, prhs)
005      C-----------------------------------------------------------
006      C   参数声明
007          integer plhs(*), prhs(*)
008          integer nlhs, nrhs
009      C   指针
010          integer ir, jc , pr
011      C   实际数据
012          real*8 pdata(6)
013          data pdata /3, 1, 10, -7, 2, -2/
014          integer      irdata(6)
015          data irdata / 0, 3, 0, 1, 2, 3 /
016          integer      jcdata(5)
017          data jcdata / 0, 2, 2, 5, 6 /
018      C-----------------------------------------------------------
019          integer mxGetIr,mxGetJc,mxGetPr
020          integer mxCreateSparse
021          integer mxCopyReal8ToPtr
022          integer mxCopyInteger4ToPtr
023      C   代码行
```

```
024          plhs(1) = mxCreateSparse(4 , 4 , 6 , 0)
025          ir = mxGetIr(plhs(1))
026          jc = mxGetJc(plhs(1))
027          pr = mxGetPr(plhs(1))
028      C   内存复制
029          call mxCopyReal8ToPtr(pdata,pr,6)
030          call mxCopyInteger4ToPtr(irdata,ir,6)
031          call mxCopyInteger4ToPtr(jcdata,jc,5)
032      C   函数文件尾部
033          return
034          end
```

本例子的核心部分从 024 行开始，首先使用 mxCreateSparse 函数创建了稀疏矩阵的 mxArray 数据对象，然后分别使用 mxGetIr、mxGetJc、mxGetPr 三个函数获取稀疏矩阵的三个重要的指针 ir、jc 和 pr，这里是实数的稀疏矩阵，所以没有 pi 指针。接着，使用 mxCopy 函数完成数据内存的复制。这里需要用户注意的是整数类型的数据拷贝，在 MATLAB 的外部接口函数库中，具有三个不同的函数用于整数类型数据的拷贝，分别是：

- mxCopyInteger1ToPtr。
- mxCopyInteger2ToPtr。
- mxCopyInteger4ToPtr。

同样，还有三个函数用于将指针数据拷贝给相应的整数类型数据，这里就不一一列举了。需要读者注意的是，针对具体的整数数据类型须选取不同的拷贝函数。下面编译运行该文件：

```
>> mex fcreatesparse.f
>> A = fcreatesparse
A =
    (1,1)         3
    (4,1)         1
    (1,3)        10
    (2,3)        -7
    (3,3)         2
    (4,4)        -2
>> full(A)
ans =
     3     0    10     0
     0     0    -7     0
     0     0     2     0
     1     0     0    -2
```

关于稀疏矩阵以及 MATLAB 对稀疏矩阵的操作，请参阅 MATLAB 的帮助文档，或者相应的计算机数据结构书籍资料。

例 4-4　复数数据。

Fortran 语言的数据类型是比较丰富的，其中，其独有的复数数据类型是该语言的一个优势，使其成为了早期计算机数值编程领域首选的编程语言。在 MATLAB 中，同样有相应的数据类型，不过 Fortran 语言的复数和 MATLAB 的复数之间进行交互的时候，只能够通过 mxCopy 函数完成数据的传递。例 4-4 演示了 Fortran 语言 MEX 文件处理复数的方法。下面是程序的源代码：

```
001     C=============================================================
002     C       times2.f
003     C       将输入的数据乘以 2.
004     C
005     C       简单的 MEX 文件示例.
006     C=============================================================
007     C       算法子程序
008             subroutine times2(y, x , size)
009             complex*16 x(*), y(*)
010             integer size
011             do 10, i = 1,size
012             y(i) = 2.0 * x(i)
013     10      continue
014             return
015             end
016
017     C       入口函数
018             subroutine mexFunction(nlhs, plhs, nrhs, prhs)
019     C-------------------------------------------------------------
020     C       (pointer) Replace integer by integer*8 on the DEC Alpha
021     C       64-bit platform.
022
023             integer plhs(*), prhs(*)
024             integer mxGetPr, mxGetPi,mxCreateDoubleMatrix
025             integer x_pr, x_pi , y_pr, y_pi
026     C-------------------------------------------------------------
027             integer nlhs, nrhs
028             integer mxIsComplex
029             integer m, n, size
030             complex*16 x(100), y(100)
031
032     C       检查输入参数是否满足需要
033             if(nrhs .ne. 1) then
```

```
034             call mexErrMsgTxt('One input required.')
035         elseif(nlhs .ne. 1) then
036             call mexErrMsgTxt('One output required.')
037         endif
038
039     C   获取输入的数据尺寸
040         m = mxGetM(prhs(1))
041         n = mxGetN(prhs(1))
042         size = m*n
043
044     C   确认输入的数据是否为合适的类型
045         if(mxIsComplex(prhs(1)) .eq. 0) then
046             call mexErrMsgTxt('Input must be a complex.')
047         endif
048     C   获取输入的数据指针
049         x_pr = mxGetPr(prhs(1))
050         x_pi = mxGetPi(prhs(1))
051         call mxCopyPtrToComplex16(x_pr,x_pi, x, size)
052
053     C   调用计算子程序
054         call times2(y, x, size)
055     C   创建返回的参数
056         plhs(1) = mxCreateDoubleMatrix(m, n, 1)
057         y_pr = mxGetPr(plhs(1))
058         y_pi = mxGetPi(plhs(1))
059
060     C   复制数据到输出参数
061         call mxCopyComplex16ToPtr(y, y_pr , y_pi, size)
062
063         return
064         end
```

在 Fortran 语言的 MEX 函数文件中，如果需要操作复数，则需要使用 mxCopyPtrToComplex16 函数将指针拷贝给复数数组，而使用 mxCopyComplex16ToPtr 函数将复数数组拷贝给指针，这两个函数的定义分别如下：

```
subroutine mxCopyPtrToComplex16(pr, pi, y, n)
complex*16 y(n)
integer*4 pr, pi, n
subroutine mxCopyComplex16ToPtr(y, pr, pi, n)
complex*16 y(n)
```

integer*4 pr, pi, n

在上述的两个函数中，y 代表 Fortran 语言的复数数组，而 pr 和 pi 为 mxArray 数据对象中保存数据的两个指针。在例 4-4 的代码中，分别在 051 行和 061 行调用了这两个函数，完成了复数矩阵的处理。

编译运行例 4-4：

```
>> mex times2.f
>> A = [1+2i,1-2i;2+1i,2-1i]
A =
    1.0000 + 2.0000i    1.0000 - 2.0000i
    2.0000 + 1.0000i    2.0000 - 1.0000i
>> B = times2(A)
B =
    2.0000 + 4.0000i    2.0000 - 4.0000i
    4.0000 + 2.0000i    4.0000 - 2.0000i
>> C = times2(B(1:4))
C =
  Columns 1 through 3
    4.0000 + 8.0000i    8.0000 + 4.0000i    4.0000 - 8.0000i
  Column 4
    8.0000 - 4.0000i
>> whos
  Name        Size                    Bytes  Class
  A           2x2                        64  double array (complex)
  B           2x2                        64  double array (complex)
  C           1x4                        64  double array (complex)
Grand total is 12 elements using 192 bytes
```

可以看到，例 4-4 的代码可以处理矩阵和向量，但输入的参数必须为复数。

例 4-5　处理字符串。

字符串永远是一类特殊的数据类型，在 Fortran 语言的 MEX 文件中也是如此。例 4-5 的代码演示了在 Fortran 语言的 MEX 函数文件中处理字符串的一些方法和函数。请读者注意，字符串在高级语言中一般占用一个字节的内存，而 MATLAB 的字符串占用两个字节的内存，因此，程序的输出比输入少一半也就不足为奇了。下面是例 4-5 程序的源代码：

```
001       C       parsestring.f
002       C       算法函数
003               subroutine revord(input_buf, strlen, output_buf)
004               character   input_buf(*), output_buf(*)
005               integer i, strlen
006       C       倒置字符串
```

```
007             do 10 i=1,strlen
008                 output_buf(i) = input_buf(strlen-i+1)
009     10      continue
010
011         return
012             end
013     C       入口函数
014             subroutine mexFunction(nlhs, plhs, nrhs, prhs)
015     C-------------------------------------------------------------
016     C   参数声明
017             integer plhs(*), prhs(*)
018             integer nlhs, nrhs
019             integer mxCopyCharacterToPtr,mxCreateCharArray,mxGetPr
020             integer mxGetM,mxGetN,mxGetString
021             character*32 input_buf
022             character*32 output_buf
023             integer dims(2)
024             integer pr
025             integer m,n,size
026     C-------------------------------------------------------------
027     C       代码行
028             if(nrhs .ne. 1) then
029                 call mexErrMsgTxt('One input required.')
030             elseif(nlhs .ne. 1) then
031                 call mexErrMsgTxt('One output required.')
032             endif
033     C       判断输入的参数尺寸
034             m = mxGetM(prhs(1))
035             n = mxGetN(prhs(1))
036             size = m*n
037             if(size .gt. 30) then
038               call mexErrMsgTxt('Too long of the input string!')
039             endif
040     C       获取字符串
041             call mxGetString(prhs(1),input_buf,size)
042
043             output_buf = ' '
044     C       调用算法函数
045             call revord(input_buf, size, output_buf)
```

```
046      C      创建输出参数
047             dims(1) = 5
048             dims(2) = 3
049             plhs(1) = mxCreateCharArray(2,dims)
050      C      获取指针
051             pr = mxGetPr(plhs(1))
052      C      拷贝数据
053             call mxCopyCharacterToPtr(output_buf,pr,30)
054      C      函数文件的尾部
055             return
056             end
```

例 4-5 的函数文件对字符串的操作使用了两个不同的函数，分别为 mxGetString 函数和 mxCopyCharacterToPtr 函数，这两个函数以及 mxCopyPtrToCharacter 函数都可以处理字符串，但是，从使用的难易角度上看，mxGetString 函数更值得使用，这个函数相对简单容易。编译运行例 4-5：

```
>> mex parsestring.f
>> str = parsestring('A B C D E F G H I J K L M N !')
str =
!JE
NID
MHC
LGB
KFA
>> str = parsestring('ABCDEFGHIJKLMNOPQRSTUVWXYZ1234')
str =
4TJ
2RH
ZPF
XND
VLB
```

请读者注意查看两次运行结果的异同，比较 mxCopy 函数运行的结果。

例 4-6　清除 MEX 文件的操作。

mex 函数中提供了锁定和解锁函数，被锁定的 mex 函数不能从内存中清除，而解锁之后才能够从内存中将函数文件清除。另外，mexAtExit 函数可以用来向 MATLAB 注册函数，注册的函数将在 MEX 文件从内存中清除时被自动调用，从而可以完成一些清除内存，释放空间的操作。例 4-6 演示了锁定函数 mexLock 和解锁函数 mexUnlock 的使用方法，mexAtExit 函数注册的函数可以帮助读者了解 clear all 操作是否真的将函数清除。下面是程序的源代码：

```
001     C       mexlocked.f
002     C       Demo file about mexlock & mexunlock
003     C       入口函数
004             subroutine mexFunction(nlhs, plhs, nrhs, prhs)
005     C-----------------------------------------------------------
006     C       参数声明
007             external closeup
008             integer plhs(*), prhs(*)
009             integer nlhs, nrhs
010             integer mexIsLocked
011             real*8    mxGetScalar
012             real*8    lock
013             integer status
014     C       代码行
015             status = mexAtExit(closeup)
016             if(status .ne. 0) then
017                call mexErrMsgTxt(
018            *       'Register Exit subroutine error!')
019             endif
020     C       参数检查
021             if (nrhs .ne. 1) then
022                call mexErrMsgTxt(
023            *     'Input argument must be a real scalar double')
024             else if ((mxIsDouble(prhs(1)) .ne. 1) .or.
025            *       (mxGetN(prhs(1))*mxGetM(prhs(1)) .ne. 1)    .or.
026            *       (mxIsComplex(prhs(1)) .eq. 1)) then
027                call mexErrMsgTxt(
028            *     'Input argument must be a real scalar double')
029             end if
030             if (nlhs .gt. 0) then
031                call mexErrMsgTxt('No output arguments expected.')
032             end if
033
034             lock = mxGetScalar(prhs(1))
035     C       参数检查
036             if ((lock .ne. 0.0) .and. (lock .ne. 1.0) .and.
037            *       (lock .ne. -1.0)) then
038                call mexErrMsgTxt('Input argument must be either 1
039            *          to lock or -1 to unlock or 0 for lock status.')
```

```
040             end if
041       C     锁定操作
042             if (mexIsLocked() .eq. 1) then
043                 if (lock .gt. 0.0) then
044                     call mexWarnMsgTxt('MEX-file is already locked')
045                 else if (lock .lt. 0.0) then
046                     call mexUnlock()
047                     call mexPrintf('MEX-file is unlocked')
048                 else
049                     call mexPrintf('MEX-file is locked')
050                  end if
051             else
052                 if (lock .lt. 0.0) then
053                     call mexWarnMsgTxt('MEX-file is already unlocked')
054                 else if (lock .gt. 0.0) then
055                     call mexLock()
056                     call mexPrintf('MEX-file is locked')
057                 else
058                     call mexPrintf('MEX-file is unlocked')
059                  end if
060             end if
061       C     函数文件尾部
062             return
063             end
064       C     清除函数
065             subroutine closeup()
066             call mexPrintf('Exit the MEX file')
067             end
```

编译运行例 4-6：

```
>> mex mexlocked.f
>> % 锁定 MEX 文件
>> mexlocked(1)
MEX-file is locked
>> % 不允许再次锁定
>> mexlocked(1)
Warning: MEX-file is already locked
>> % 从内存中清除 MEX 文件操作
>> clear all
>> % 无法成功，因为 MEX 文件被锁定
```

```
>> mexlocked(1)
Warning: MEX-file is already locked
>> % 解锁
>> mexlocked(-1)
MEX-file is unlocked
>> % 再次清除
>> clear all
Exit the MEX file
>> % 操作成功!
```

4.5 本 章 小 结

本章通过丰富的例子介绍了 Fortran 语言 MEX 文件的编写方法。Fortran 语言和 C 语言的侧重点不一致，Fortran 语言主要是一种科学计算语言，其数值类型丰富，数据计算稳定，同时 MATLAB 的早期核心就是 Fortran 语言编写的算法包。通过本章的学习，相信读者已经全面掌握了 MEX 文件的编写方法。MEX 文件的主要目的是将用户已有的 C 语言或者 Fortran 语言的代码集成到 MATLAB 里面来，通过第 3 章和第 4 章的介绍，应该可以了解到编写 MEX 文件不是一件轻松的事情，因此只有在万不得已的情况下才考虑使用 MEX 文件。MATLAB 提供的 M 函数文件本身是一种非常好的快速开发的编程语言，大多数的操作都可以使用 M 文件来实现。

Fortran 语言的 MEX 文件和 C 语言的 MEX 文件是本书的重点，请读者针对自己的工作特点学习两种 MEX 文件的编写方法。另外，有关 MEX 文件的一些高级话题，例如内存管理、MEX 函数的应用等内容，在本章没有讲述，请读者仔细阅读第 3 章的相关章节，或者仔细阅读 MATLAB 的帮助文档。

练　　习

1. 比较 C 语言 MEX 文件和 Fortran 语言 MEX 文件的异同点。

2. 使用 Fortran 语言 MEX 文件，重新编写例 3-10 设置图像属性的代码，要求能够在 MEX 文件中直接创建曲线，并且设置曲线的属性，可以在集成开发环境中完成程序的编译过程。

3. 使用 Fortran 语言 MEX 文件编写第 2 章练习 3 的代码，创建整数类型的矩阵即可。

4. 编写程序完成下列功能：

调用 MATLAB 指令，根据输入的参数计算矩阵的特征值和特征向量(如果输入的参数是双精度类型矩阵，则利用 MATLAB 的矩阵运算指令)。

第 5 章　MAT 文件应用

MAT 文件是 MATLAB 用来存储数据的一种特殊的二进制文件格式，它的扩展名为.mat。这种文件能够跨平台应用，也就是说在 Windows 平台保存的 MAT 文件可以应用到 UNIX 平台上。为了便于用户使用 MAT 数据文件，在外部接口应用程序中，MATLAB 还提供了在 C 语言或者 Fortran 语言应用程序中读写 MAT 文件的接口函数。本章将针对 C 语言或者 Fortran 语言的 MAT 数据文件应用程序进行详细的介绍。

本章重点内容

- MAT 数据文件格式；
- 读写 MAT 数据文件；
- mat 函数。

提示：

MATLAB 本身提供了丰富的数据 I/O 能力，几乎可以读取各种数据文件，例如声音、图像、文本数据等。MATLAB 外部接口提供的 MAT 数据文件 I/O 能力是 MATLAB 数据文件 I/O 的扩展，它不仅可以用于 C 语言或者 Fortran 语言 MEX 文件中，而且还可以用于独立的可执行应用程序的开发。

5.1　MAT 文件入门

MATLAB 自己提供一种特殊的数据文件格式——MAT 文件，这种文件是一种二进制格式文件，扩展名为.mat，它为 MATLAB 提供了跨平台的数据交互能力。这些*.mat 文件之所以能够独立于各种平台的原因是在文件头带有设备的签名。MATLAB 在载入文件时将检查这个签名，如果发现文件来源不同于当前的系统，则进行必要的转换。目前 MAT 文件的版本为 5，它的文件格式如图 5-1 所示。

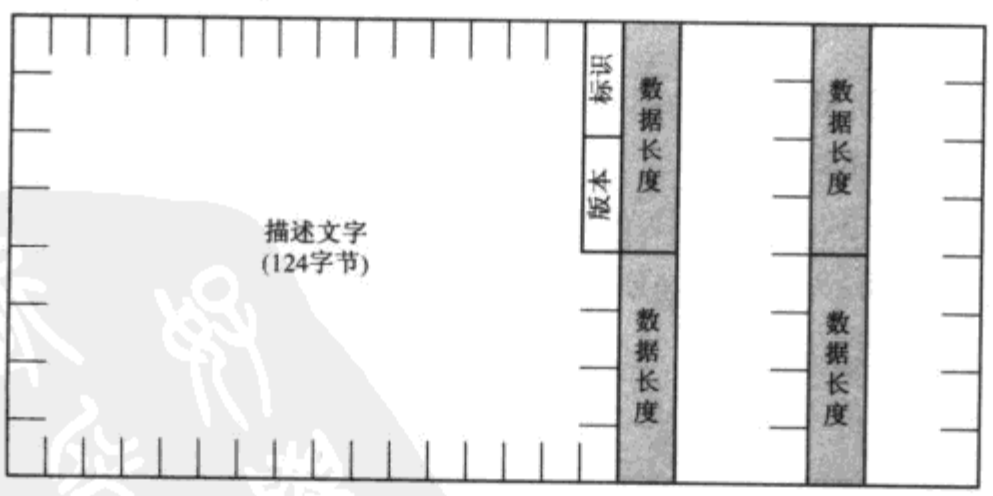


图 5-1　MAT 文件格式

一般 MAT 文件分为两个部分，文件头部和数据。其中在文件的头部主要包括一些描述性文字和相应的版本与标识，这部分占用了 128 个字节。此后依次是保存在 MAT 文件中的数据，数据是按照数据类型、数据长度和数据三个部分保存的。

一般的来说，MAT 文件都是在 MATLAB 环境下创建，并且完成数据的读写工作的，这些工作都是通过 save 指令或者 load 指令来完成的。

save 指令能够将当前工作空间中的变量保存到指定的数据文件中，其基本形式如下：

save	将当前工作空间中所有的变量保存到 matlab.mat 文件中
save filename var1 var2……	将当前工作空间中的 var1、var2 等变量保存到指定文件中
save filename data*	功能同上，其中*为通配符
save filename	将当前工作空间中所有的变量保存到指定的文件中
save ……option	按照 option 的不同取值保存数据
save('filename',……)	save 指令的函数格式用法

其中，option 可以有如下几种选项：

-append	在已有的数据文件尾部追加数据
-ascii	保存为 ASCII 文本格式，数据按照 8 位精度保存
-ascii -double	保存为 ASCII 文本格式，数据按照 16 位精度保存
-ascii -tabs	保存为 ASCII 文本格式，数据之间使用制表符作为间隔
-ascii -doube -tabs	上述几种选项的结合
-mat	保存为二进制的 MAT 文件格式(默认)
-v4	保存为版本 4 格式的数据文件

load 指令将数据文件的数据导入到 MATLAB 的工作空间，其基本的形式如下：

load	将 matlab.mat 文件中所有的变量加载到当前的工作空间
load filename	将指定文件中所有的变量加载到当前的工作空间
load filename var1 var2 ……	将指定文件中指定的变量加载到当前工作空间
load filename -ascii	将数据文件按照文本格式加载
load filename -mat	将数据文件按照 MAT 文件格式加载
S = load(……)	load 指令的函数格式用法

 注意：

■ 加载数据文件时，数据文件只要保存在 MATLAB 的搜索路径上即可。

■ 若不指明数据文件的扩展名，则数据文件默认按照二进制的 MAT 文件格式加载，否则都按照文本格式文件加载。

■ 版本 4 的 MAT 文件是早期的 MATLAB 数据文件格式，现在已经很少使用了。

■ 若保存数据为文本格式时不指定间隔符为制表符，则数据之间使用空格作为数据之间的间隔。

下面结合具体的操作实例来说明 save 和 load 指令的使用方法。

例 5-1　save 和 load 指令示例。

在 MATLAB 命令行窗口中，键入下面的指令：

```
>> clear all
>>%创建变量
>> x1 = 2; x2 = 3; x3 = 4;y1=0;
>> %保存数据
>> save xdata x1 x2
>> %查看当前路径下的 MAT 文件
>> dir *.mat
xdata.mat
>> clear all
>> %加载数据——默认加载二进制格式文件
>> load xdata
>> whos
  Name        Size                    Bytes  Class
  x1          1x1                         8  double array
  x2          1x1                         8  double array
Grand total is 2 elements using 16 bytes
```

例 5-1 演示了使用 save 和 load 指令保存加载数据的基本过程，需要注意，保存数据文件时的通配符“*”的使用，在保存文本格式文件时使用了该通配符，于是 MATLAB 将所有以 x 开头的变量保存了起来。

提示：

关于在 MATLAB 中进行数据文件 I/O 的内容请参阅《MATLAB 基础与编程入门》一书，或者阅读 MATLAB 的帮助文档。

5.2 MAT 文件应用

所谓 MAT 文件应用就是指读写 MAT 数据文件的 C 语言或者 Fortran 语言应用程序。MAT 文件应用程序既可以是 MEX 文件，也可以是独立的可执行应用程序，它可以使用 C 语言开发也可以使用 Fortran 语言开发。为了便于读写 MAT 文件，MATLAB 提供了相应的接口函数——mat 函数，这些函数都是以 mat 前缀开始的。MAT 文件应用程序就是利用这些 mat 函数完成 MAT 数据文件的读写工作。

5.2.1 简单 MAT 文件应用示例

下面看一个 C 语言 MAT 文件应用的例子，以便了解 MAT 文件应用程序的基本结构和 MAT 文件应用的基本过程。

例 5-2 在 C 语言 MEX 文件中创建 MAT 数据文件——mexcreatematfile.c。

```
001     /* 必要的头文件 */
002     #include "mex.h"
```

```
#include "mat.h"
/* MEX 函数文件入口 */
void mexFunction(int nlhs,mxArray *plhs[],
                 int nrhs,const mxArray *prhs[])
{
    mxArray *string,*scalar,*array;
    double img[] = {0,1,2,3,4,5,6,7,8};
    double real[] = {8,7,6,5,4,3,2,1,0};
    double *pr,*pi;
    mxChar *filename;
    MATFile *file;
    int buflen = 0;
    /* 对输入输出参数进行判断 */
    if(nrhs !=1)
        mexErrMsgTxt("必须给一个输入参数，表示文件名！");
    if(mxGetClassID(prhs[0]) != mxCHAR_CLASS)
        mexErrMsgTxt("输入参数必须为字符串，表示文件名！");
    /* 创建数据 */
    string = mxCreateString("MATLAB is Wonderful!");
    scalar = mxCreateDoubleScalar(2003);
    array = mxCreateDoubleMatrix(3,3,mxCOMPLEX);
    pr = mxGetPr(array);
    pi = mxGetPi(array);
    memcpy(pr,real,9*sizeof(double));
    memcpy(pi,img,9*sizeof(double));
    /* 获取文件名称 */
    buflen = mxGetNumberOfElements(prhs[0])+1;
    filename = mxMalloc(buflen);
    mxGetString(prhs[0],filename,buflen);
    /* 创建新的 MAT 文件 */
    file = matOpen(filename,"w");
    if(file == NULL)
        mexErrMsgTxt("无法创建指定的文件");
    /* 向 MAT 文件中写入数据 */
    matPutVariable(file, "String", string);
    matPutVariable(file, "Scalar", scalar);
    matPutVariable(file, "Array", array);
    /* 关闭 MAT 文件 */
```

```
041             matClose(file);
042             mexPrintf("成功创建 MAT 文件:%s",filename);
043             /* 好习惯! */
044             mxDestroyArray(string);
045             mxDestroyArray(scalar);
046             mxDestroyArray(array);
047             mxFree(filename);
048         }
```

编译链接生成 MEX 文件，然后运行：

```
>> mex mexcreatematfile.c
>> mexcreatematfile('matdemo.mat')
成功创建 MAT 文件:matdemo.mat。
>> %查看都生成了什么
>> what
MAT-files in the current directory D:\Temp
matdemo
MEX-files in the current directory D:\Temp
mexcreatematfile
>> %加载数据文件
>> load matdemo
>> whos
  Name            Size                     Bytes  Class
  Array           3x3                        144  double array (complex)
  Scalar          1x1                          8  double array
  String          1x20                        40  char array
Grand total is 30 elements using 192 bytes
>> Array
Array =
   8.0000            5.0000 + 3.0000i   2.0000 + 6.0000i
   7.0000 + 1.0000i  4.0000 + 4.0000i   1.0000 + 7.0000i
   6.0000 + 2.0000i  3.0000 + 5.0000i        0 + 8.0000i
>> Scalar
Scalar =
        2003
>> String
String =
MATLAB is Wonderful!
```

例 5-2 的代码比较长，演示了一个完整的 MAT 数据文件应用程序。首先，在所有的

MAT 数据文件应用程序 C 语言源代码中，必须包含头文件 mat.h，该头文件定义了 mat 函数的原型，并且包含了 matrix.h 头文件。这样在 MAT 数据文件应用程序的 C 语言源程序中就不用包含 matrix.h 头文件了。在例 5-2 的代码中，003 行就包含了 mat.h 文件。接下来的代码和普通的 MEX 文件应用程序的代码没有什么区别，用于处理和判断 MEX 函数文件的输入、输出，使用 mxCreate 函数创建必要的数据，然后在程序的 033 行利用 matOpen 函数打开了一个数据文件，文件名称是通过 MEX 文件的输入参数来创建的。

matOpen 函数返回的变量是 MAT 数据文件的指针，类型为 MATFile，例 5-2 使用的变量在第 013 行的代码中进行了声明。如果 matOpen 命令没有正常打开 MAT 数据文件，则返回值为 NULL，在 034 行~035 行的代码进行了相应的判断。

037 行~039 行的代码使用 matPutVariable 函数向 MAT 数据文件中写入数据。matPutVaraible 函数按照给定的变量名称(字符串)将相应的 mxArray 数据对象写入数据文件中。

在 041 行使用 matClose 函数将打开的 MAT 数据文件关闭。打开的数据文件一定要适时地关闭，否则会出现无法预料的错误。

程序的最后将函数中动态分配的内存都释放了，这是一种良好的编程习惯，请读者借鉴。

例 5-2 的代码详细说明了 MAT 数据文件应用的典型过程：

- 打开数据文件——matOpen；
- 从数据文件读入些数据——matGetVariable 或者向数据文件写入些数据——matPutVariable。
- 关闭数据文件——matClose。

5.2.2　常用的 mat 函数

为了完成 MAT 数据文件应用程序，MATLAB 提供了 mat 函数用于完成打开关闭数据文件、读写数据等操作，这些函数分别有 C 语言形式和 Fortran 语言形式的。本小节简要介绍常用的函数使用方法，以便于后面章节的学习。

 **注意：**

关于 mat 函数的详细说明请读者参阅 MATLAB 的帮助文档。

1. 打开数据文件——matOpen

打开数据文件的函数为 matOpen 函数，该函数 C 语言的定义如下：

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

Fortran 语言的定义如下：

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

matOpen 函数的作用是打开数据文件并返回文件的指针。MAT 数据文件根据函数的第二个参数 mode 取值的不同决定打开数据文件将用于哪些操作，在表 5-1 中对 mode 的取值

进行了总结。

表 5-1 打开数据文件的不同模式参数

参 数	说 明
R	以只读模式打开数据文件，这时只能从数据文件中读取文件中包含的数据，而不能向数据文件写入任何数据
U	打开数据文件，用于更新(update)模式，此模式下，既可从数据文件中读入数据，也可以向数据文件写入数据。如果需要打开的数据文件不存在，则函数不会创建新的数据文件
W	以写入模式打开数据文件，这时只能向数据文件写入数据，如果数据文件不存在，则创建新的数据文件
w_4	以版本 4 创建能够写入数据的数据文件

2. **关闭数据文件——matClose**

关闭 MAT 数据文件的函数为 matClose，它的定义比较简单，输入参数就是 MAT 数据文件的指针。在 C 语言中的定义如下：

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

在 Fortran 语言中的定义如下：

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

如果成功地关闭了 MAT 数据文件，则函数返回数值 0。

3. **获取变量——matGetVariable**

读入数据的函数为 matGetVariable，该函数在 C 语言中的定义如下：

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

在 Fortran 语言中的定义如下：

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

该函数的输入参数是 MAT 数据文件的指针和需要读入的变量名称(字符串)，返回变量是 mxArray 数据对象的指针。

4. **写入数据——matPutVariable**

写入数据的函数为 matPutVariable，该函数在 C 语言中的定义如下：

```
int matPutVariable(MATFile *mfp, const char *name, const mxArray *mp);
```

在 Fortran 语言中的定义如下：

```
integer*4 function matPutVariable(mfp, name, pm)
integer*4 mfp, pm
character*(*) name
```

该函数的输入参数分别是数据文件的指针 mfp、数据变量在 MAT 文件中的名称 name 字符串以及具体保存数据对象的指针。如果该函数成功操作，则返回整数 0。

注意：

所有 mat 函数都有具体的返回值，所以获取这些函数的返回值并且对其进行必要的处理和判断，以查看相应的函数是否运行成功，这些操作都是在 MAT 数据文件应用程序中非常重要的工作。在例 5-2 中忽略了这些操作，这是一种不好的编程习惯。

5.3　编译 MAT 文件应用程序

5.2 小节中介绍的 MAT 数据文件应用程序是 mex 函数文件，MAT 数据文件应用程序还可以是独立的可执行应用程序。不过，在编译生成独立的可执行应用程序时，需要针对选择的编译器及系统环境进行必要的设置，才能完成独立可执行应用程序的生成工作。本小节，将介绍生成独立可执行应用程序的编译方法。

5.3.1　命令行编译

生成独立的可执行应用程序可以在 MATLAB 命令行窗口或者操作系统的控制台方式下，利用 MEX 指令完成应用程序的编译。在前面的章节介绍了 MEX 指令，读者应该知道 MEX 指令是通过选项文件完成程序的编译工作的，在编译 MEX 文件时，需要针对不同的编译器选择不同的选项文件，MAT 数据文件应用程序也是如此。

生成独立的可执行应用程序和生成 MEX 文件不同。生成 MEX 文件时，只要在 MATLAB 中完成编译器的配置工作就可以直接使用 MEX 指令完成 MEX 文件的编译工作了。因为配置编译器的时候，将编译器的选项文件改名为 mexopts.bat 选项文件，并且拷贝到了系统环境目录下。不过在系统路径下 mexopts.bat 文件不能完成 MAT 数据文件应用程序的编译，必须通过 MEX 命令行指定具体的选项文件才可以。为了便于说明编译过程，这里使用一个 Fortran 语言的 MAT 数据文件应用程序作为示例。

例 5-3　Fortran 语言 MAT 文件示例——creatematfile.f。

```
001       C    创建 MAT 数据文件
002       C    独立的可执行应用程序
003            program creatematfile
004       C-----------------------------------------------------------------
005       C    声明函数或者子程序
006       C
007            integer mxCreateDoubleMatrix, mxCreateString
008            integer mxCreateScalarDouble
009            integer matOpen,matClose
010            integer matPutVariable
011       C
012       C    变量声明
013       C
```

```
      integer matfile
      integer pa0,pa1,pa2
      real*8 ral(9)
      data ral / 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0 /
      real*8 img(9)
      data img / 8.0, 7.0, 6.0, 5.0, 4.0, 3.0, 2.0, 1.0, 0.0 /

C
C     创建 MAT 数据文件
C
      write(6,*) '创建 MAT 文件 ...'
      mp = matOpen('filebyfor.mat', 'w')
      if (mp .eq. 0) then
          write(6,*) '无法创建指定的 MAT 文件 "file.mat".'
          stop
      end if
C
C     创建变量
C
      pa0 = mxCreateDoubleMatrix(3,3,1)
      call mxCopyReal8ToPtr(ral, mxGetPr(pa0), 9)
      call mxCopyReal8ToPtr(img, mxGetPi(pa0), 9)
      pa1 = mxCreateString('MATLAB: The language of computing')
      pa2 = mxCreateScalarDouble(ral(1))
C
C     写入数据
C
      call matPutVariable(mp, 'Array', pa0)
      call matPutVariable(mp, 'String', pa1)
      call matPutVariable(mp, 'Scalar', pa2)
C
C   关闭数据文件
C
      call matClose(mp)
      write(6,*) 'MAT 文件创建完毕！'
C
C   好习惯!
C
```

```
052          call mxDestroyArray(pa0);
053          call mxDestroyArray(pa1);
054          call mxDestroyArray(pa2);
055          stop
056          end
```

例 5-3 的代码完成的功能和例 5-2 的代码类似，不过程序的执行环境不尽相同。例 5-3 的程序是独立的可执行的应用程序，它没有 MEX 文件的入口函数，而是 Fortran 语言的入口 program creatematfile。

如果需要在 MATLAB 的命令行窗口中通过 MEX 指令完成可执行应用程序的创建，则首先需要在操作系统中进行编译器的配置，例如在环境变量中添加必要的对 LIB、Include 和 Path 等环境变量的说明，这样能够保证在编译的时候选择到正确的库文件或者文件。

然后，在 MEX 指令中，需要使用-f 命令行参数，定义必要的选项文件，进行 MAT 文件应用程序的编译使用的选项文件位于%MATLABROOT%\bin\win32\mexopts 路径下，这些选项文件都具有共同的后缀名称和扩展名——engmatopts.bat。例如 Msicrosoft Visual C++ 6 使用的选项文件就是 msvc60engmatopts.bat，而在例 5-3 中使用的 Fortran 语言的选项文件则是 df60engmatopts.bat。于是，在 MATLAB 命令行下键入下面的指令就可以创建独立的可执行 MAT 数据文件应用程序：

```
>> mex -f %MATLABROOT%\bin\win32\mexopts\df60engmatopts.bat creatematfile.f
```

这里需要读者将%MATLABROOT%替换成自己的 MATLAB 安装路径。如果应用程序源代码没有任何错误，则命令行窗口中不会显示任何信息，否则请仔细对照源代码文件，查找并修改相应的错误，直到程序能够完成编译为止。

运行得到的可执行应用程序：

```
>> dir *.exe
creatematfile.exe
>> !creatematfile
创建 MAT 文件 ...
MAT 文件创建完毕!
Stop - Program terminated.
```

可执行应用程序运行创建了 filebyfor.mat 文件，可以加载该数据文件并查看包含的数据：

```
>> load filebyfor
>> whos
  Name          Size                    Bytes  Class
  Array         3x3                       144  double array (complex)
  Scalar        1x1                         8  double array
  String        1x33                       66  char array
Grand total is 43 elements using 218 bytes
```

注意：

例 5-3 生成的可执行应用程序可以在相应操作系统的控制台方式下运行。

能否正确编译和生成 MAT 数据文件的独立可执行应用程序，关键就是选择正确的选项文件，并且代码不能有错误。请读者根据自己操作系统的配置情况完成命令行编译生成独立可执行应用程序的操作。

提示：

如果在编译过程中，出现无法确定 LIB 文件所在位置的情况，一般的原因是用户选择的编译器没有在系统的环境变量注册，特别是编译器的 LIB 路径是否添加到了环境变量 LIB 的定义中，此外还有 Include 和 Path 环境变量的设置。

另外，一种比较方便的方法是将与用户选择的编译器相对应的 engmatopts.bat 文件拷贝一份到当前的工作路径下，并且将起改名为 mexopts.bat，这样就可以直接在 MATLAB 命令行窗口中通过命令行完成 MAT 文件的编译了。

5.3.2 使用集成开发环境

由于是进行独立的可执行应用程序的开发，所以在集成开发环境中进行应用程序开发的机会更多。如果利用集成开发环境开发 MAT 应用程序，则需要按照一定的步骤完成应用程序的编译。这里使用例 5-3 说明在 Compaq Visual Fortran 的开发环境下进行程序开发的基本步骤。

首先，运行 Compaq Visual Fortran，执行“File”菜单下的“New”命令，在新建项目对话框中，选择“Fortran Console Application”，并且给一个项目工程的名称，例如 matdemo，如图 5-2 所示。

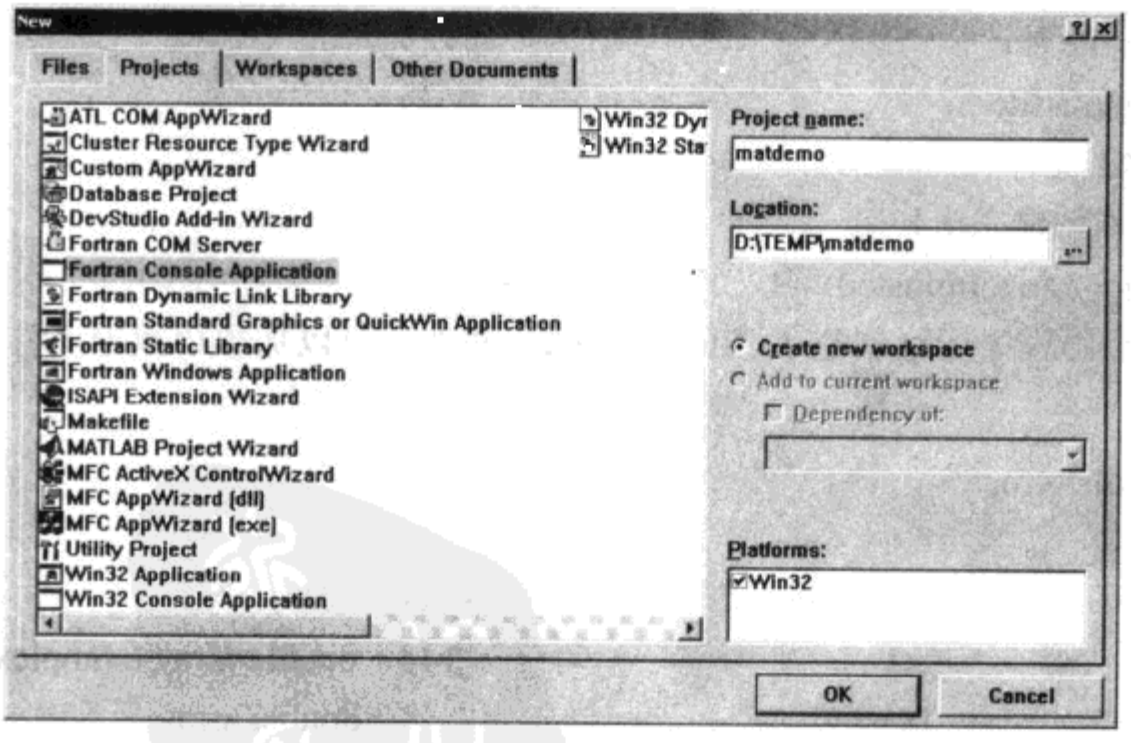


图 5-2 新建 Fortran 控制台应用项目

单击“OK”按钮后，在出现的对话框中选择创建空的项目文件选项，并且单击对话框的“Finish”按钮，这样就在 Visual Fortran 中创建完毕项目工程文件了。

执行“Tools”菜单下的“Options”命令，在弹出的对话框中，选择“Directories”属性页。在该属性页下设置 Include 路径属性和 LIB 路径属性，分别将 MATLAB 外部接口应用程序的 Include 路径和 LIB 路径添加到这里。在设置 LIB 路径时，需要根据不同的编译器选择不同的 LIB 路径，Visual Fortran 的 LIB 路径为 E:\MATLAB6p5p1\EXTERN\LIB\WIN32\DIGITAL\DF60，该路径下包含了 Visual Fortran 需要使用的 MATLAB 函数库文件，如图 5-3 所示。

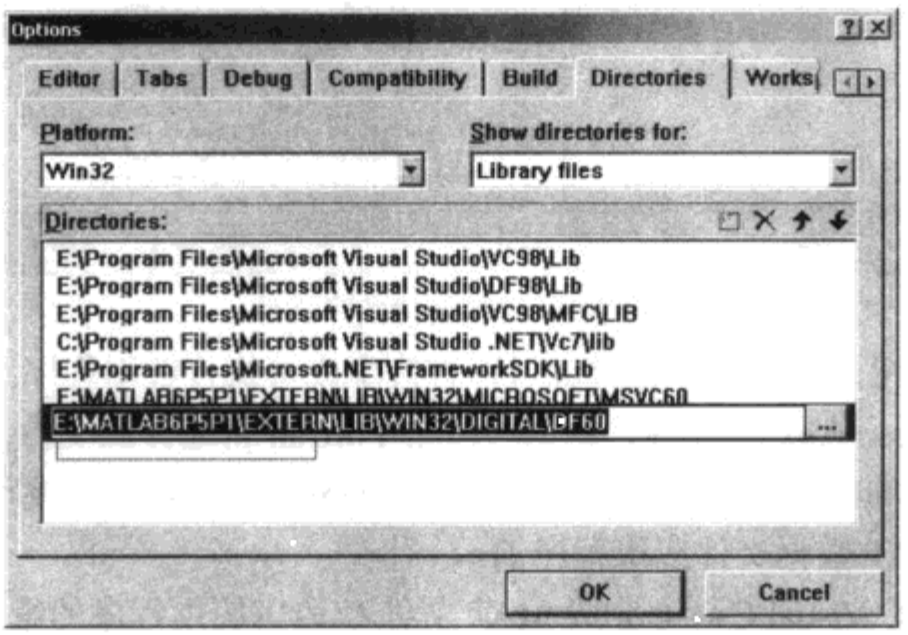


图 5-3　设置 Lib 属性

设置完毕后，单击“OK”按钮退出对话框。

接下来添加 Fortran 语言源文件，如果 Fortran 语言源文件已经存在，则通过“Project”菜单下“Add to Project”子菜单下的“Files”命令，在文件选择对话框中选择 Fortran 语言源代码文件为“creatematfile.f”，如图 5-4 所示。

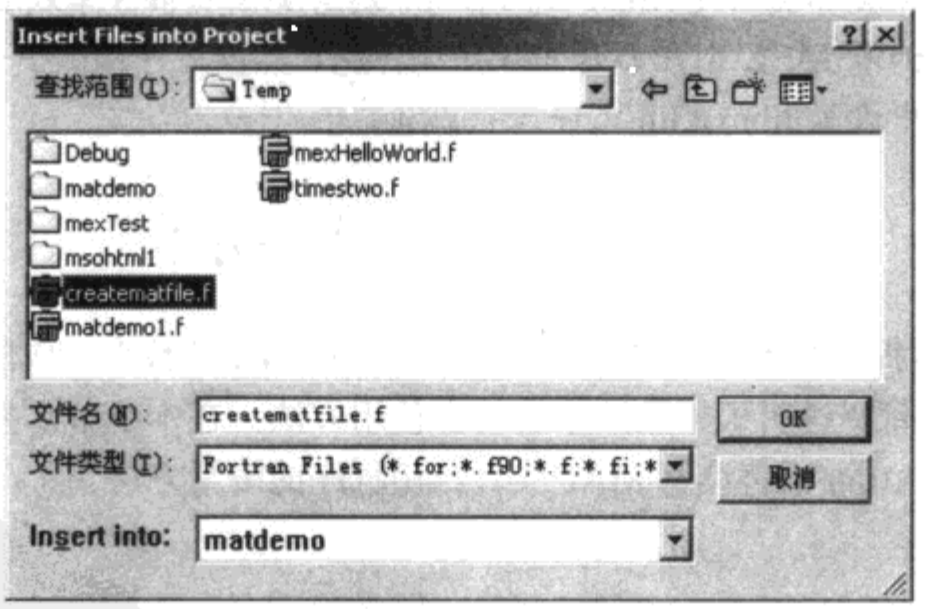


图 5-4　添加文件到项目工程中

单击“OK”按钮完成文件的添加，也可以通过“Project”菜单的“Add to Project”子菜单下的“New”命令将新的文件添加到工程中。

在进行编译之前，执行“Project”菜单下的“Setting”命令，在弹出的项目属性对话框中，设置 Link 属性页中链接库文件选项，将 libmx.lib 和 libmat.lib 两个文件添加到链接选项中，如图 5-5 所示。

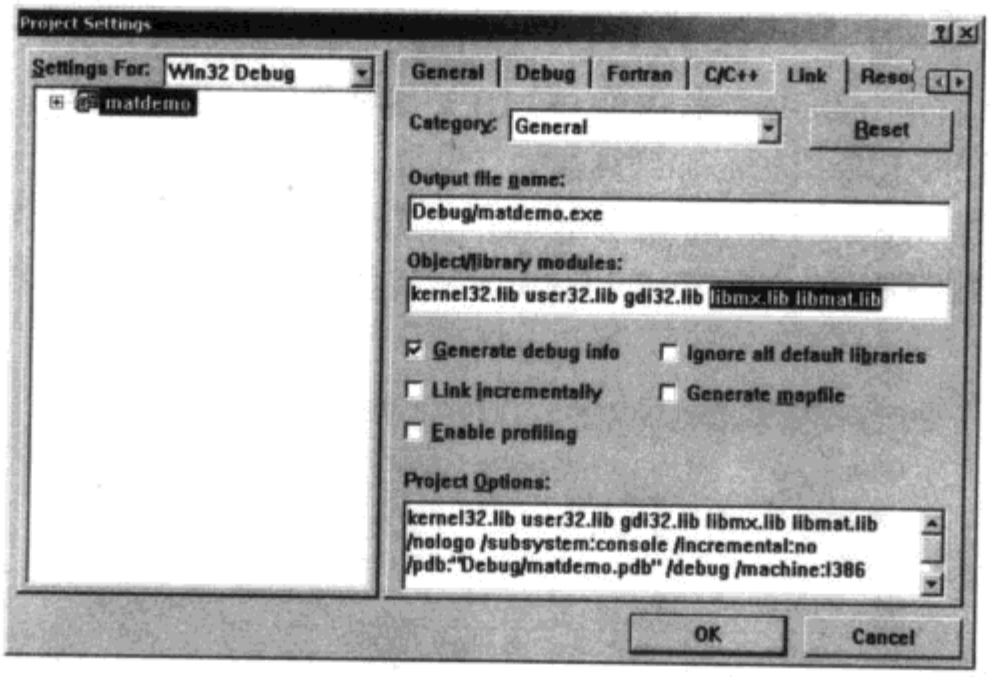


图 5-5 设置链接的库文件

单击“OK”按钮之后确认，这时就可以直接生成可执行应用程序了。执行“Build”菜单下的“Build matdemo.exe”命令，这时 Visual Fortran 将编译源文件，并生成可执行应用程序。再次执行“Build”菜单下的“Execute matdemo.exe”命令，就可以在 Visual Studio 环境下直接运行 MAT 数据文件应用程序了。

如果读者使用了 Visual C++ 6 完成 MAT 文件 C 语言应用程序的编译，则基本过程和设置方法和本小节介绍的内容完全一致，如果读者使用其它的集成开发环境，则需要根据自己使用的集成开发环境情况完成有关的操作，其中最关键的就是 LIB 库文件和 Include 头文件的设置，以及在相应的链接选项中添加必要的库文件。

提示：

如果使用集成开发环境创建 MEX 应用程序，则需要按照第 3 章或者第 4 章介绍的方法完成相应的操作。如果 MEX 引用程序中使用了 MAT 数据文件函数——mat 函数，则需要在相应的链接选项中设置 libmat.lib 文件。

5.4 MAT 文件应用示例

MAT 文件应用程序基本的数据文件操作都是通过 mat 函数来完成的，在前面几个小节的示例程序中已经简要说明了一些函数的基本使用方法。在本小节，将通过一些具体示例来说明 C 语言和 Fortran 语言读写 MAT 文件应用程序的编写方法。

提示：

mat 函数和 MAT 文件应用程序不仅可以是 mex 函数文件，也可以是独立可执行的应用程序。本小节将所有 Fortran 语言的应用程序都写成了 mex 函数，将 C 语言的应用程序写成了独立可执行应用程序。

例 5-4 全局变量的操作——mexcreateglobal.f。

作为一种编程语言，M 语言提供了全局变量和局部变量。所谓全局变量，就是在不同

的函数之间可以共享的数据变量，MATLAB 专门为这些变量设置了全局工作空间，和基本工作空间和函数工作空间不同，全局工作空间可以被所有 mex 函数共享，获取里面的数据。

在 MAT 数据文件应用程序中，可以直接将变量按照全局变量的范围写入 MAT 数据文件中，这个功能是通过 matPutVariableAsGlobal 函数完成的，它的 C 语言定义如下：

```
int matPutVariableAsGlobal(MATFile *mfp, const char *name, const mxArray *mp);
```

该函数的 Fortran 语言定义如下：

```
integer*4 function matPutVariableAsGlobal(mfp, name, pm)
integer*4 mfp, pm
character*(*) name
```

该函数的输入参数和 matPutVariable 函数的输入参数一样，分别为数据文件的指针 mfp、变量在数据文件中的名称 name 以及数据 mxArray 结构对象指针。如果该函数成功执行了，则返回数值为 0。

在下面的 Fortran 语言 mex 函数源代码中，将用户输入的参数 prhs 直接按照全局变量写入了数据文件 matglobal.mat 中，下面是程序的源代码：

```
001      C    mexcreateglobal.f
002      C    入口函数
003           subroutine mexFunction(nlhs,plhs,nrhs,prhs)
004      C-------------------------------------------------------------
005      C    Fortran 语言 MAT 数据文件应用的 MEX 函数
006      C-------------------------------------------------------------
007      C    参数声明
008           integer plhs(*), prhs(*)
009           integer nlhs, nrhs
010      C    需要调用的 API 函数
011           integer matOpen,matClose,matPutVarialbeAsGlobal
012           integer mxCreateString
013      C    必要的变量
014           integer matfile
015           integer status
016           character name(12)
017           data name /'A','B','C','D','E','F','G','H','I','J','K','L'/
018      C    判断输入输出参数
019           if(nrhs .eq. 0) then
020                call mexErrMsgTxt('必须有输入参数!')
021           elseif(nrhs .gt. 10) then
022                call mexErrMsgTxt('输入参数太多!')
023           endif
024      C
025      C    创建 MAT 文件
```

```
      C
            matfile = matOpen('matglobal.mat','w')
            if (matfile .eq. 0) then
                  call mexErrMsgTxt('无法创建 MAT 文件!')
            endif
      C
      C     写入数据
      C
            do 10 i=1,nrhs
                  status = matPutVariableAsGlobal(matfile,name(i),prhs(i))
                  if(status .ne. 0 ) then
                        call mexErrMsgTxt('写入数据错误!')
                  endif
      10    continue
      C
      C     关闭 MAT 文件
      C
            status = matClose(matfile)
            if(status .ne. 0) then
                  call mexErrMsgTxt('无法关闭 MAT 文件!')
            endif
            plhs(1) = mxCreateString('创建 matglobal.mat,并写入数据!')
            return
            end
```

在 MATLAB 命令行中，键入下面的指令完成程序的编译：

```
>> mex mexcreateglobal.f
```

如果程序没有任何错误，则 MATLAB 不会输出任何信息，并且创建 mex 函数文件。运行该文件，得到数据文件——matglobal.mat：

```
>> mexcreateglobal('String',10,rand(3))
ans =
创建 matglobal.mat 并写入数据!
>> what
MAT-files in the current directory D:\Temp
matglobal
MEX-files in the current directory D:\Temp
mexcreateglobal
>> load matglobal
>> whos
  Name          Size                    Bytes  Class
```

```
A       1x6          12  char array (global)
B       1x1           8  double array (global)
C       3x3          72  double array (global)
ans     1x21         42  char array
Grand total is 37 elements using 134 bytes
```

从 mex 函数文件运行的结果看，该函数文件将输入的参数分别以 A、B、C 等变量写入了数据文件，加载数据文件后，A、B、C 等变量都作为全局变量存在于工作空间中。整个程序中，核心的就是 033~038 行的代码，这部分代码将用户输入的参数 prhs 分别以全局变量的形式写入了数据文件中。在将数据写入文件时，适时地判断函数运行的返回值来获取程序是否成功运行的信息是非常重要的工作，希望读者在编写自己的应用程序时也遵循此原则，编写足够的代码来判断程序是否运行正常。

例 5-5　删除文件中的变量 —— mexdelvar.f。

除了能够直接向 MAT 数据文件写入变量以外，MATLAB 还提供了直接从 MAT 数据文件中删除指定数据变量的能力，这个功能是通过 matDeleteVariable 函数来实现的。该函数在 C 语言中的定义如下：

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

该函数在 Fortran 语言中的定义如下：

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

该函数有两个输入参数，分别是 MAT 数据文件的指针 mfp 和需要删除的变量名称 name，如果函数成功地删除了指定的变量，则函数返回值为 0。

下面的代码是 Fortran 语言的 MEX 文件，在该文件中使用了 matDeleteVariable 函数来删除指定数据文件中的变量，代码如下：

```
001     C   mexdelvariable.f
002     C   入口函数
003         subroutine mexFunction(nlhs,plhs,nrhs,prhs)
004     C-------------------------------------------------------------
005     C   Fortran 语言 MAT 数据文件应用的 MEX 函数
006     C-------------------------------------------------------------
007     C   参数声明
008         integer plhs(*), prhs(*)
009         integer nlhs, nrhs
010     C   需要调用的 API 函数
011         integer matOpen,matClose,matDeleteVariable
012         integer mxGetString,mxGetNumberOfElements
013     C   必要的变量
014         integer matfile
```

```
015         integer status,buflen
016         character*64 filename
017     C   判断输入输出参数
018         if(nrhs .ne. 1) then
019             call mexErrMsgTxt('There must be one input argument for the
020        *file''s name!')
021         endif
022     C   获取文件名
023         buflen = mxGetNumberOfElements(prhs(1))
024         status = mxGetString(prhs(1),filename,buflen)
025         if (status .ne. 0) then
026             call mexErrMsgTxt('Can not get the file''s name! ')
027         endif
028     C
029     C   打开 MAT 文件
030     C
031         matfile = matOpen(filename,'u')
032         if (matfile .eq. 0) then
033             call mexErrMsgTxt('Can not open the file!')
034         endif
035     C
036     C   删除变量 String
037     C
038         call mexPrintf('删除变量.....')
039         status = matDeleteVariable(matfile,'String')
040         if(status .ne. 0) then
041             call mexWarnMsgTxt('Can not delete the variable!')
042         endif
043     C
044     C   删除变量 Scalar
045     C
046         status = matDeleteVariable(matfile,'Scalar')
047         if(status .ne. 0) then
048             call mexWarnMsgTxt('Can not delete the variable!')
049         endif
050     C
051     C   关闭 MAT 文件
052     C
053         status = matClose(matfile)
```

```
054             if(status .ne. 0) then
055                 call mexErrMsgTxt('Can not close the MAT File!')
056             endif
057             call mexPrintf('Done!')
058             return
059             end
```

在例 5-5 的代码中，首先获取用户的输入信息，将用户输入的参数作为需要打开的文件名，并且将该数据文件打开，然后利用 035～046 行的代码将指定的两个变量 Scalar 和 String 删除了。在例 5-5 的程序中，使用了很多判断程序是否运行正常的代码，这是一种很好的编程习惯，请读者借鉴。

在 MATLAB 中编译运行该程序，运行该程序还需要使用例 5-2 的 mex 函数 mexcreatematfile.dll。首先使用该 mex 函数创建一个数据文件，然后再使用本例子的 mex 函数删除其中的变量。在 MATLAB 命令行中键入下面的指令：

```
>> mex   mexdelvariable.f
>> what
MEX-files in the current directory D:\Temp
mexcreatematfile
mexdelvariable
>> %创建 MAT 数据文件
>> mexcreatematfile('matfile1.mat');
成功创建 MAT 文件:matfile1.mat
>> %查看文件中包含的变量
>> load matfile1.mat
>> whos
  Name          Size                    Bytes  Class
  Array         3x3                       144  double array (complex)
  Scalar        1x1                         8  double array
  String        1x20                       40  char array
Grand total is 30 elements using 192 bytes
>> %删除变量
>> mexdelvariable('matfile1.mat')
删除变量.....Done!
>> %清除内存空间
>> clear
>> %再次查看文件中包含的变量
>> load matfile1.mat
>> whos
  Name          Size                    Bytes  Class
  Array         3x3                       144  double array (complex)
```

```
Grand total is 9 elements using 144 bytes
>> %如果再次删除，则出现警告
>> mexdelvariable('matfile1.mat')
删除变量.....Warning: Can not delete the variable!
Warning: Can not delete the variable!
Done!
```

通过例 5-5 使读者能够进一步加深对 MAT 数据文件应用的认识，同时还能够再次复习和掌握 Fortran 语言 MEX 函数文件的编写方法。

例 5-6　读写 MAT 文件数据——mutil.c、mdriver.c。

很多时候程序员编写的应用程序都是由不同的源代码文件组成的，在本例子中使用的 C 代码就是由两个文件组成的，分别为 mutil.c 和 mdriver.c。

在 mutil.c 中定义了两个函数——ReadArray 和 WriteArray，这两个函数分别用来从数据文件中读取数据和写入数据。另外，在 mutil.h 文件中声明了这两个函数，使用函数时，只要在应用程序中包含该头文件就可以在程序中具体利用这两个函数了。

在 mutil.h 文件中对 ReadArray 和 WriteArray 两个函数的声明的代码如下：

```
001    #ifndef __mutil_h__
002    #define __mutil_h__
003    /* 头文件
004     * ReadArray 和 WriteArray 函数的声明
005     */
006    #include "mat.h"
007
008    mxArray * ReadArray( const char *, const char * );
009    void WriteArray( const char *, const char *, mxArray * );
010
011    #endif
```

mutil.c 文件中是两个函数具体的定义，代码如下：

```
001    /*
002     * mutil.c
003     * ReadArray 和 WriteArray 函数定义
004     */
005    #include "mat.h"
006    /*
007     * 根据输入的文件名和变量名称，获取相应的 mxArray 数据对象*
008     */
009    mxArray * ReadArray( char *matfile, const char *vname )
010    {
011        MATFile *mfp;
```

```
012         mxArray *out;
013         /* 打开数据文件 */
014         mfp = matOpen( matfile, "r" );
015         if (mfp == (MATFile *)NULL)
016         {
017             fprintf( stderr, "错误：无法打开指定的 MAT 文件:'%s'.\n", matfile );
018             return( NULL );
019         }
020         /* 读数据 */
021         out = matGetVariable( mfp, vname   );
022         if (out == (mxArray *)NULL)
023         {
024             fprintf( stderr, "错误：无法读取指定的变量:%s.\n", vname );
025         }
026         /* 关闭数据文件 */
027         matClose( mfp );
028         return( out );
029     }
030
031     /*
032      * 根据输入的文件名和变量名称，将变量数值写入文件
033      */
034     void WriteArray( char *matfile, char *varname, mxArray *var )
035     {
036         MATFile *mfp;
037         int         status;
038         /* 打开数据文件 */
039         mfp = matOpen( matfile, "u" );
040         if (mfp == (MATFile *)NULL) {
041             fprintf( stderr, "错误：无法打开指定的 MAT 文件:'%s'.\n", matfile );
042         } else {
043             /* 好习惯：使用函数返回的状态值判断函数的执行状况*/
044             /* 写入数据 */
045             status = matPutVariable( mfp, varname, var );
046             if (status != 0) {
047                 fprintf( stderr, "无法写入变量%s 到指定的文件\n", varname );
048             }
049             /* 关闭数据文件 */
050             matClose( mfp );
```

```
051             }
052     }
```

ReadArray 和 WriteArray 的代码流程非常相似，两者是标准的 MAT 数据文件应用的过程：打开文件、读写数据、关闭文件。两个函数的输入参数也基本是一致的，分别为 MAT 数据文件的名称 matfile 和变量的名称 varname，读取数据的函数返回 mxArray 数据结构对象的指针，写入数据函数的第三个输入参数是 mxArray 数据结构对象的指针。

在 mdriver.c 文件中调用上述两个函数，首先读取了 wind.mat 中包含的变量，然后以一个新的名称写入到数据文件中，它的代码如下：

```
001     /*
002      * file: mdriver.c
003      *
004      *   本文件演示了从 MAT 文件中读取变量"wind"
005      *   然后又将该变量写回文件的过程
006      *
007      *   编译该文件时：:
008      *
009      *      >> mex -f optsfile mdriver.c mutil.c
010      */
011
012     /* 头文件，包含 ReadArray 和 WriteArray 函数声明 */
013     #include "mutil.h"
014     void main()
015     {
016         /* 主函数，程序被编译成为独立运行的应用程序*/
017         mxArray *wind;
018         char       matfile[] = "wind.mat";
019
020         /* 从 MAT 文件中获取变量 —— wind */
021         wind = ReadArray( matfile, "wind" );
022
023         /* 利用"wind"变量完成操作 */
024         /* wind = ..... */
025
026         /* 保存数据，变量名称为 newwind */
027         WriteArray( matfile, "newwind",wind );
028     }
```

在 mdriver.c 文件中，分别在 021 行和 027 行调用了 ReadArray 函数和 WriteArray 函数，完成了预先设计的功能。为了完成 mdriver.c 文件的功能，需要首先准备一个数据文件，在 MATLAB 命令行中，键入下面的指令：

```
>> wind = peaks(30);
>> save wind
>> what
MAT-files in the current directory D:\Temp
wind
```

提示：

wind 数据是在 MATLAB 中常见的三维表面数据，可以在 MATLAB 命令行中键入下面的指令查看 wind 数据的可视化结果：

```
>> surf(wind)
```

wind 包含的数据的可视化结果如图 5-6 所示。

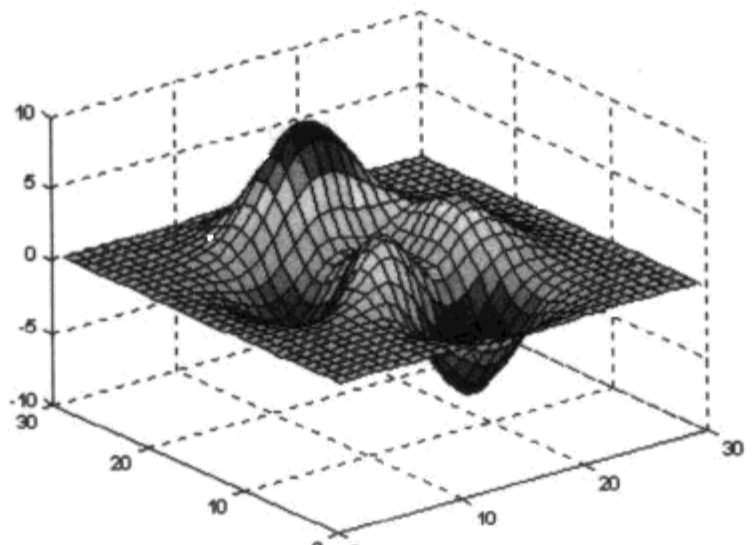


图 5-6　wind 包含的数据的可视化结果

在 MATLAB 的命令行窗口中完成应用程序的编译连接：

```
>> mex -f %MATLABROOT%\bin\win32\mexopts\msvc60engmatopts.bat mdriver.c mutil.c
```

编译连接得到可执行应用程序：

```
>> dir *.exe
mdriver.exe
```

运行 mdriver.exe：

```
>> !mdriver
```

成功运行应程序后再次加载数据文件，并且查看数据文件中包含的变量：

```
>> load wind
>> whos
  Name          Size                    Bytes  Class
  newwind       30x30                    7200  double array
  wind          30x30                    7200  double array
Grand total is 1800 elements using 14400 bytes
```

ReadArray 和 WriteArray 函数是比较有用的两个函数，读者可以在自己的应用程序中直接使用这两个函数完成自己的 MAT 文件应用程序开发。

例 5-7　获取变量信息——varinfo.c。

在前面的例子中，都是直接读取数据文件中的变量，并没有在读取变量之前判断一下是否指定的变量存在于数据文件中，这么做总是在冒风险——数据变量不存在则什么也读不到。在写入数据到数据文件时，也需要判断一下指定的数据变量是否已经存在于数据文件中了，因为写数据到文件时，如果写入的变量已经存在于数据文件中，则新的数据会覆盖文件中已经存在的数据。在 mat 函数中，函数 matGetVariableInfo 和 matGetDir 可以完成数据是否存在于数据文件中的判断。例 5-7 的代码中分别利用了 matGetDir 函数和 matGetVariableInfo 函数完成了变量信息的获取，同时完成了对数据文件是否包含指定变量的判断：

```
001    /*
002     * file: varinfo.c
003     *
004     *   获取指定 MAT 文件中变量的所在位置，
005     *   首先调用函数 matGetVariableInfo 然后，通过函数 matGetDir
006     *   获取变量的位置
007     *
008     */
009
010    /* 包含必要的头文件 */
011    #include "stdio.h"
012    #include "mat.h"
013    /* 函数声明 */
014    int CheckArrayByHeader( const char *, const char * );
015    int FindArrayByDir( const char *, const char * );
016    /* 主函数，生成独立的可执行应用程序 */
017    void main()
018    {
019        char matfile[] = "varinfo.mat";
020        char variable[10];
021        int location, flag;
022
023        fprintf(stdout,"\n 请输入变量名称: ");
024        fscanf(stdin,"%s",variable);
025        flag = CheckArrayByHeader( matfile, variable );
026        if (flag == 1)
027        {
028            location = FindArrayByDir( matfile, variable );
029            fprintf( stdout, "\n 变量 %s 存在于 MAT 文件中;",variable);
030            fprintf( stdout, "\n 它是数据文件中第 %d 个变量",location);
```

```
031         }else
032             fprintf( stdout, "\n 错误:变量 %s 不存在于 MAT 文件中",variable);
033     }
034     /*
035      * 从数据文件中获取变量信息
036      */
037     int FindArrayByDir( const char *matfile, const char *variable )
038     {
039         MATFile     *mfp;
040         const char **dir;
041         int         num, idx;
042         int         found = 0;
043         /* 以只读方式打开数据文件 */
044         mfp = matOpen( matfile, "r" );
045         if (mfp == (MATFile *)NULL)
046         {
047             fprintf( stdout, "错误：无法打开 MAT 文件:%s.\n",matfile );
048             return( -1 );
049         }
050         /* 获取文件中数据变量结构 */
051         dir = matGetDir( mfp, &num );
052         fprintf(stdout,"\n 变量的个数:= %d\n",num);
053         /* 输出信息 */
054         for( idx=0; idx<num; idx++ )
055         {
056             fprintf(stdout,"\n 序号 = %d",idx);
057             fprintf(stdout,"    名称 = %s",dir[idx]);
058             if( strcmp( dir[idx], variable ) == 0 )
059             {
060                 found = 1;
061                 break;
062             }
063         }
064         /* 关闭数据文件 */
065         matClose( mfp );
066         /* 返回变量 */
067         if( found ){
068             return( idx+1 );
069         } else {
```

```
070                 return( -1 );
071             }
072     }
073     /*
074      * 检查指定的变量是否存在与数据文件中
075      */
076     int CheckArrayByHeader( const char *matfile, const char *variable )
077     {
078         MATFile     *mfp;
079         mxArray     *tmp;
080         /* 以只读方式打开数据文件 */
081         mfp = matOpen( matfile, "r" );
082         if( mfp == (MATFile *)NULL )
083         {
084             fprintf( stderr, "错误：无法打开 MAT 文件:%s.\n",matfile );
085             return( -1 );
086         }
087         /* 获取变量的信息 */
088         tmp = matGetVariableInfo( mfp, variable );
089         if(tmp == NULL)
090         {
091             fprintf( stderr, "错误:matGetVariableInfo 函数运行错误\n" );
092             return (-1);
093         }
094         /* 关闭数据文件 */
095         matClose( mfp );
096         return(1);
097     }
```

例 5-7 的代码比较长，其中有三个函数，分别为应用程序的主函数、用来判断数据变量是否存在于数据文件中的函数 CheckArrayByHeader 以及获取数据变量在数据文件中信息的函数 FindArrayByDir。

在 CheckArrayByHeader 函数中，调用了 matGetVariableInfo 函数，该函数的 C 语言定义如下：

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

该函数的 Fortran 语言定义如下：

```
integer*4 function matGetVariableInfo(mfp, name);
integer*4 mfp
character*(*) name
```

该函数的输入参数是 MAT 数据文件的指针 mfp 和变量的名称 name。如果输入的变量

存在于数据文件中，则返回的变量包含数据的头信息，如果变量不存在，则返回值为 NULL，所以可以利用该函数来判断指定的变量是否存在于数据文件中，然后根据返回值完成相应的操作。

在 FindArrayByDir 函数中，主要调用了 matGetDir 函数获取数据文件中的变量信息，该函数的 C 语言定义如下：

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

该函数的 Fortran 语言定义如下：

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

该函数的输入参数是 MAT 数据文件的指针 mfp 和整数变量 num，如果成功执行了该函数，则返回字符串数组，同时 num 为文件中包含的数据个数。在例 5-7 的代码中，FindArrayByDir 函数通过循环一次获取每个变量的具体信息，然后比较变量名称和数据文件中的信息，从而完成数据变量是否存在于数据文件中的判断。

在编译并执行 varinfo.c 之前，首先准备数据文件 varinfo.mat，该文件包含的数据如下：

```
>> load varinfo.mat
>> whos
  Name          Size                    Bytes  Class
  A             5x5                        84  double array (sparse)
  C             1x4                       548  cell array
  Matrix        4x4                       128  double array
  Scalar        1x1                         8  double array
  String        1x44                       88  char array
Grand total is 136 elements using 856 bytes
```

由执行的结果可以看出，在文件中包含了稀疏矩阵 A、元胞数组 C、矩阵 Matrix、标量 Scalar 和字符串 String。

编译 varinfo.c 文件：

```
>> mex -f %MATLABROOT%\bin\win32\mexopts\msvc60engmatopts.bat varinfo.c
```

执行生成的可执行应用程序：

```
>> dir *.exe
varinfo.exe
```

在 Windows 的控制台方式下运行该程序：

```
D:\Temp>varinfo
请输入变量名称: PP
错误:matGetVariableInfo 函数运行错误
错误:变量 PP 不存在于 MAT 文件中
D:\Temp>varinfo
请输入变量名称: A
变量的个数:= 5
 序号 =0    名称 =A
```

```
变量 A 存在于 MAT 文件中;
它是数据文件中第 1 个变量
D:\Temp>varinfo
请输入变量名称: C
变量的个数:= 5
 序号 =0    名称 =A
 序号 =1    名称 =String
 序号 =2    名称 =Scalar
 序号 =3    名称 =Matrix
 序号 =4    名称 =C
变量 C 存在于 MAT 文件中;
它是数据文件中第 5 个变量
```

从上面三次运行 varinfo.exe 的结果可以看出判断数据信息的优势，就是在没有具体读取人和数据的情况下就可以完成数据的简要判断和处理，当确认数据可以被读取时，再完成数据的读取等操作，这样程序会更加安全、可靠。请读者留意数据变量在数据文件中保存的顺序。

5.5 本章小结

在本章中，重点介绍了 MAT 数据文件应用程序的编写方法。MAT 数据文件应用程序主要是在 C 语言或者 Fortran 语言开发的应用程序中，对 MATLAB 独有的二进制数据文件 MAT 文件进行读写操作。MAT 数据文件应用程序既可以是 mex 函数文件，也可以是独立的可执行应用程序。在本章分别针对不同类型的应用程序进行了介绍。通过本章的学习，读者能够基本掌握 MAT 数据文件应用程序的基本流程以及主要的 mat 函数的使用方法。MAT 数据文件应用程序能够脱离 MATLAB 环境使用，是外部接口应用程序中应用比较广泛的一种。读者可以结合本章介绍的示例程序，仔细学习 MAT 数据文件应用程序的开发、编译方法。

练　习

1. 集成开发环境的编译。

尝试将例 5-2 在 C 语言 MEX 文件中创建 MAT 数据文件——mexcreatematfile.c,在读者熟悉的 C 语言集成开发环境中进行编译链接，生成可执行应用程序。注意，该例子生成的目标文件是 C 语言的 mex 函数文件，请参考第 3 章介绍的相关内容。

2. 变量信息获取。

判断变量是否存在于 MAT 数据文件中是非常重要的操作，在例 5-7 中采用了 C 语言程序的编写方式，请读者利用 Fortran 语言将该程序重写一遍以巩固 Fortran 语言 MAT 数据文件应用程序的编写能力。

3. 读写数据。

有一纯文本格式的数据文件，该数据文件中包含的数据如下：

Broncos	14	2	0.8750	y
Falcons	14	2	0.8750	y
Lions	5	11	0.3125	n
Patriots	15	1	0.9375	y
Vikings	9	7	0.5625	y

编写程序实现下面的功能：将该纯文本格式的数据文件数据利用 C 语言或者 Fortran 语言的源程序读入；利用 MAT 的数据文件应用能力将这些数据写入一个 MAT 文件中，要求每一列数据保存在一个变量中，一共有五个变量，这些变量保存在一个数据文件中。

4. 读写数据。

例 5-2 生成的 MEX 文件执行的结果是生成包含三个变量的数据文件，其中一个变量为字符串类型的变量，它的内容是“MATLAB is Wonderful!”。请利用 C 语言或者 Fortran 编写程序实现下面的功能：将字符串从数据文件中读入，并且将该字符串内容完整地颠倒过来，也就是生成新的字符串“!lufrednoW si BALTAM”，并将新的字符串写入到 MAT 数据文件中。

第 6 章　MATLAB 计算引擎应用

在前面的章节中介绍的 MATLAB 外部接口应用都有一个共同点：基本都是在 MATLAB 中集成已有的 C 语言或者 Fortran 语言的代码。尽管 MATLAB 在科学计算、系统设计分析等方面有着无可比拟的优势，但是从计算机程序开发的角度上看，MATLAB 的 M 语言还是有些功能无法实现或者实现起来有一定困难，例如复杂的用户图形界面应用程序以及对计算机操作系统或者硬件设备操作的能力等。为了能够在其它的高级语言中使用 MATLAB 的基本算法，MATLAB 提供了多种手段来完成，其中计算引擎应用是相对常用也是最容易实现的一种。本章将详细介绍开发 MATLAB 计算引擎应用程序的方法。

本章重点内容

- 计算引擎应用；
- eng 函数；
- 应用程序的编译。

提示：

在其它的高级编程语言中调用 MATLAB 的算法可以使用的方法包括：

- MATLAB 的 COM 应用；
- MATLAB Compiler 打包函数库，生成可执行应用程序；
- MATLAB COM/Excel Builder 打包 COM 组件；
- MATLAB 计算引擎应用等。

其中 MATLAB COM 应用、Compiler 和 COM/Excel Builder 等内容将在《MATLAB 应用程序集成与发布》一书中进行介绍。

6.1　概　　述

MATLAB 计算引擎应用程序的思想和 MATLAB 的 MEX 文件相反，MEX 文件是在 MATLAB 环境下调用 C 或者 Fortran 语言程序的手段，而 MATLAB 计算引擎则是在 C 或者 Fortran 语言环境中调用 MATLAB 函数的方法。

MATLAB 的计算引擎应用实际上就是利用 MATLAB 提供的一组接口函数(API)，在用户开发的 C 语言或者 Fortran 语言应用程序中，通过某种通信机制后台调用 MATLAB 应用程序以完成复杂的系统任务。计算引擎应用程序是在 MATLAB 环境之外的可执行应用程序，在它们运行的过程中需要利用某种通信机制和另外一个 MATLAB 进程(会话)交互数据。在不同系统平台上，应用程序使用的通信机制是不一样的，例如在 UNIX 系统中使用管道(pipes)完成，而在 Windows 平台中，这种通信机制是利用 COM 应用接口 IEngine 来完成的，

图 6-1 表示了计算引擎应用在 Windows 系统中的情况。

图 6-1　Windows 平台上的计算引擎应用程序

在 Windows 平台上，用户开发的 C 语言或者 Fortran 语言应用程序作为 COM 应用的客户端，是整个应用程序的前端，而 MATLAB 作为 COM 应用的服务器运行在后台。两者利用 MATLAB 提供的计算引擎接口函数进行交互，例如在客户端从 MATLAB 进程中获取数据，或者在客户端调用 MATLAB 的指令等。

在 C 语言或者 Fortan 语言中使用 MATLAB 计算引擎应用集成 MATLAB 的算法与其它的集成方法比较，计算引擎应用程序具有如下几点优势：

■ 调用 MATLAB 数学函数完成繁重的数学计算，充分发挥 MATLAB 在数值计算上的强大优势，通过 C 语言或者 Fortran 语言编写的 GUI 来调用后台的 MATLAB 数学计算功能，完成特殊的需要，节约开发时间。

■ 在 UNIX 平台上，计算引擎的 MATLAB 进程不仅可以运行在本地计算机上，也可以运行在网络中的任何一台计算机上，这样可以将本地计算机作为前端的用户接口界面来直接调用后台复杂的算法。

■ 避免将庞大的 MATLAB 函数库链接到客户应用程序中。利用 MATLAB Compiler 打包的函数库时，需要将 MATLAB 的数学库或者图形库都链接到客户应用程序中，这样的操作会使客户应用程序变得庞大，而 MATLAB 计算引擎函数库仅仅采用十几个函数就可以完成这些复杂的操作了。

■ 不是所有的 MATLAB 函数都可以使用 MATLAB Complier 或者 COM Builder 打包成为函数库或者 COM 组件，而使用 MATLAB 计算引擎则没有此类限制。

不过相比之下，MATLAB 计算引擎也有缺点，MATLAB 计算引擎不能够脱离 MATLAB 环境使用，运行 MATLAB 计算引擎应用程序的 Windows 平台上必须安装有 MATLAB，否则计算引擎应用程序无法使用，因此 MATLAB 计算引擎不是一种真正脱离 MATLAB 的独立可执行应用程序。而利用其它的集成方法是可以开发出真正脱离 MATLAB 环境的应用程序的。

6.2　计算引擎应用

MATLAB 的计算引擎应用是 C 语言或者 Fortran 语言的可执行应用程序，它不能使用在 mex 函数文件中。为了便于程序员开发计算引擎应用程序，MATLAB 提供了相应的接口函数——eng 函数，这些函数都是以 eng 前缀开始的，这些 eng 函数将辅助程序完成启动计算引擎、获取 MATLAB 的计算结果等工作。

6.2.1　简单计算引擎应用示例

下面看一个简单的 MATLAB 计算引擎应用的例子，这个例子使用 C 语言开发。通过本

例子来了解一下 MATLAB 计算引擎应用程序的基本结构和编译方法。

例 6-1 简单的 C 语言计算引擎应用示例——simpleengdemo.c。

```
001    /* 包含必要的头文件 */
002    #include "engine.h"
003    #include "stdio.h"
004    #include "conio.h"
005    #define BUFFERLEN 256
006    /* 主函数 */
007    void main()
008    {
009        Engine *ep;
010        char cmd[BUFFERLEN];
011        int i = 0;
012        int status = 0;
013        /* 打开计算引擎 */
014        ep = engOpen(NULL);
015        if( ep == (Engine *)NULL ){
016            printf("错误：无法打开 MATLAB 计算引擎\n" );
017            exit(-1);
018        }
019        /* 执行 MATLAB 指令 */
020        engEvalString(ep,"A = zeros(1,10);");
021        /* 稍等片刻..... */
022        printf("\n 请在 MATLAB 会话中查看计算的结果！ \n");
023        printf("\n 按任意键继续.....\n");
024        getch();
025        /* 执行其它的 MATLAB 指令 */
026        for(i = 10; i < 20; i++){
027            /* 准备在 MATLAB 中执行的指令 */
028            sprintf(cmd,"A(%d) = fibonacci(%d);",i-9,i);
029            /* 运行该指令 */
030            engEvalString(ep,cmd);
031        }
032        /* 稍等片刻..... */
033        printf("\n 请在 MATLAB 会话中查看计算的结果！ \n");
034        printf("\n 按任意键继续.....\n");
035        getch();
036        /* 关闭 MATLAB 计算引擎 */
```

```
037             status = engClose(ep);
038             /* 好习惯！  */
039             if(status != 0){
040                 printf("无法正常关闭 MATLAB 计算引擎\n");
041                 exit(-1);
042             }
043             printf("\nMATLAB 计算引擎应用完毕!\n")
044         }
```

例 6-1 的代码演示了 MATLAB 计算引擎应用程序的基本结构和流程。在代码的第 002 行包含了头文件 engine.h，在该头文件中声明了所有 eng 函数的原型，并且也包含了 matrix.h 头文件，这样在计算引擎应用程序的 C 语言源程序中就不用包含 matrix.h 头文件了。在所有的 C 语言计算引擎应用程序中，都必须包含 engine.h 头文件。

在源代码的 009 行声明了 Engine 类型的指针，该指针类似于打开文件时的文件指针，相当于计算引擎的接口句柄。在 014 行，使用 engOpen 函数打开计算引擎，这个函数返回的变量就是计算引擎的指针。有了计算引擎的指针，就可以在 C 语言程序中执行 MATLAB 的指令，就像例 6-1 的 020 行和 030 行一样。执行 MATLAB 的指令通过 engEvalString 函数，该函数的输入参数是 MATLAB 的指令字符串。

最后，当所有的应用程序代码执行完毕后，一定要关闭计算引擎，方法是通过函数 engClose 函数，该函数直接关闭计算引擎指针，并且释放必要的内存。不过编译运行该源代码是比较复杂的工作，需要像前面第 5 章介绍的采用编译 MAT 数据文件引用程序的方法来编译计算引擎应用程序，这里暂时不过多讲解编译过程，具体的编译过程将在 6.3 小节中进行介绍。

运行例 6-1，在 MATLAB 的命令行窗口中键入下面的指令：

```
>> !simpleengdemo &
```

或者在 Windows 的命令行提示符下键入指令：simpleengdemo，这时操作系统将启动一个 MATLAB 进程，不过启动的 MATLAB 进程仅包含 MATLAB 的命令行窗口，如图 6-2 所示。

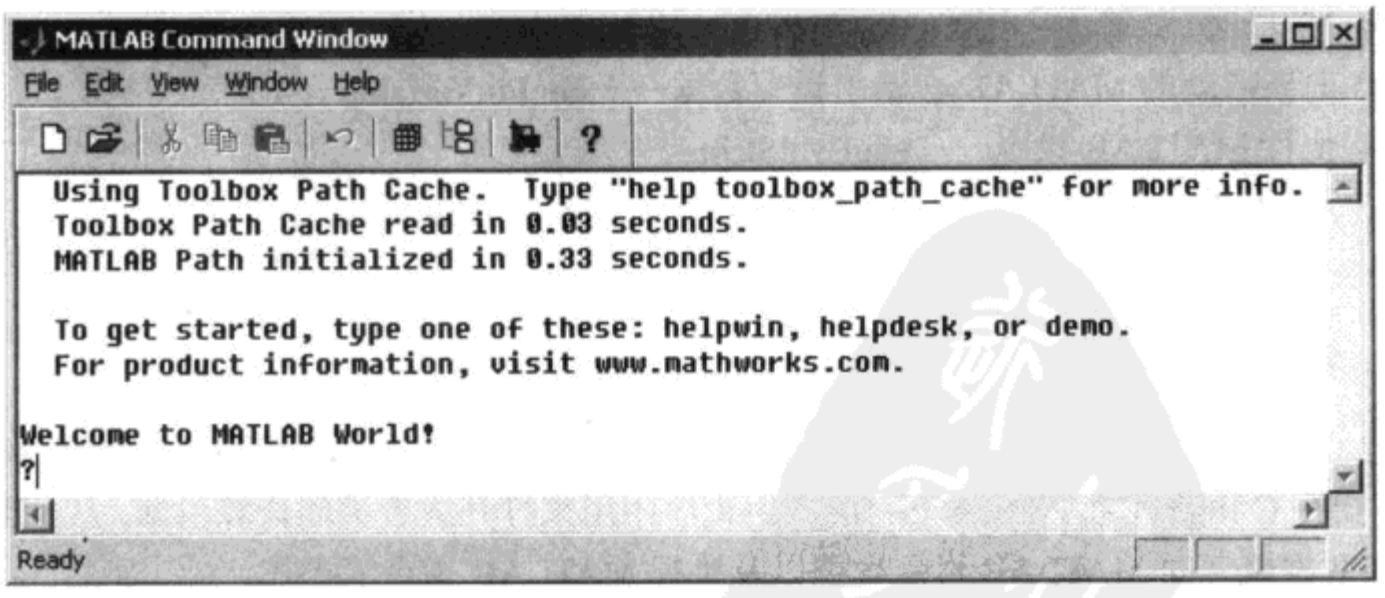


图 6-2　计算引擎启动的 MATLAB 会话窗口

这时在 Windows 的命令行提示符中显示如下信息：

请在 MATLAB 会话中查看计算的结果！

按任意键继续.....

大家可以在启动的 MATLAB 命令行中键入下面的指令：

```
?whos
  Name        Size                     Bytes  Class
  A           1x10                        80  double array
Grand total is 10 elements using 80 bytes
?A
A =
     0     0     0     0     0     0     0     0     0     0
```

在 MATLAB 会话的工作空间中有一个变量 A，该变量就是通过例 6-1 的第 020 行代码执行之后得到的。在命令行提示符中按任意键继续例 6-1 的代码执行，当重复出现相应的提示信息后，回到新启动的 MATLAB 会话中，键入下面的指令：

```
?A
A =
  Columns 1 through 6
        34          55          89         144         233         377
  Columns 7 through 10
       610         987        1597        2584
```

可以看到，由于例 6-1 的代码执行，MATLAB 工作空间中的变量 A 变成了 Fibonacci 数列的部分元素。

在 Windows 的命令行提示符中按任意键继续例 6-1 的执行，这时由于例子中正确地关闭了 MATLAB 计算引擎，则新启动的 MATLAB 会话会自动地退出运行，例 6-1 的 MATLAB 计算引擎应用实例就完整地执行完毕了。

例 6-1 的代码说明了 MATLAB 计算引擎应用 C 语言程序的基本结构以及相应的工作流程，只要包含了必要的头文件——engine.h，调用 eng 函数就可以完成计算引擎应用程序的工作。一般地，MATLAB 计算引擎应用程序的基本工作流程如下：

- 打开计算引擎——engOpen。
- 向新启动的 MATLAB 会话中设置数据——engPutVariable。
- 执行 MATLAB 指令——engEvalString。
- 从 MATLAB 会话中获取计算结果——engGetVariable。
- 关闭计算引擎——engClose。

6.2.2 常用的 eng 函数

为了完成计算引擎应用程序的开发，MATLAB 提供了若干 eng 函数，这些函数都以 eng 为前缀，在 C 语言或者 Fortran 语言的应用程序中用来打开或者关闭计算引擎，从 MATLAB 中获取数据或者向 MATLAB 设置数据以及执行 MATLAB 指令等操作。

注意：

关于 eng 函数的详细说明请读者参阅 MATLAB 的帮助文档.

1. 打开计算引擎——engOpen

打开 MATLAB 计算引擎的函数为 engOpen，该函数在 C 语言中的定义如下：

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

该函数在 Fortran 语言中的定义如下：

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

该函数具有一个输入参数——字符串，在 UNIX 平台上，如果输入参数是空字符串或者 NULL，则计算引擎打开本地计算机的 MATLAB 作为计算引擎的后台服务；如果输入参数是远程的计算机名称(hostname)，则将相应的计算机上的 MATLAB 启动作为计算引擎的后台服务。

在 Windows 平台上，该输入参数只能为 NULL。如果 engOpen 函数成功运行，则一个 MATLAB 进程作为后台服务启动，并且返回计算引擎的指针。

2. 关闭计算引擎——engClose

关闭计算引擎的函数为 engClose，该函数在 C 语言中的定义如下：

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

该函数在 Fortran 语言中的定义如下：

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

该函数只有一个输入参数——MATLAB 计算引擎指针，如果成功关闭了计算引擎，则作为计算引擎的 MATLAB 后台服务进程被关闭，然后返回整数 0 表示成功完成了操作。

3. 执行 MATLAB 指令——engEvalString

在 MATLAB 计算引擎应用程序中运行 MATLAB 指令的函数是 engEvalString，该函数在 C 语言中的定义如下：

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

该函数在 Fortran 语言中的定义如下：

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

该函数具有两个输入参数，分别为计算引擎的指针 ep 和需要执行的 MATLAB 指令 command。如果该函数返回了数值 0，则表明计算引擎已经成功处理了用户的指令。注意，在调用该函数时，请尽量保证参数 command 包含的 MATLAB 指令是合法的。

4. 获取数据——engGetVariable

获取数据的函数是 engGetVariable，它能够从当前作为计算引擎服务的 MATLAB 会话中获取指定的变量，它在 C 语言中的定义如下：

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

该函数在 Fortran 语言中的定义如下：

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

该函数的输入参数分别为 MATLAB 计算引擎的指针 ep 和需要读取的变量名 name，如果成功读取了数据，则函数返回 mxArray 数据类型对象指针。

5. 写入数据——engPutVariable

从计算引擎应用程序向 MATLAB 会话写入数据的函数是 engPutVariable，该函数在 C 语言中的定义如下：

```
int engPutVariable(Engine *ep, const char *name, const mxArray *mp);
```

该函数在 Fortran 语言中的定义如下：

```
integer*4 function engPutVariable(ep, name ,mp)
integer*4 ep, mp
character*(*) name
```

该函数具有三个输入参数，分别为计算引擎指针 ep、变量的名称 name 和在 C 语言或者 Fortran 源中创建 mxArray 数据类型对象的指针 mp。如果函数成功写入了数据，则返回整数数值 0。

6.3 编译应用程序

和编译 MAT 数据文件应用程序类似，编译 MATLAB 计算引擎应用程序不是一件非常容易的事情。由于 MATLAB 计算引擎应用程序都是可执行的应用程序，所以在编译生成 MATLAB 计算引擎应用程序时，需要针对选择的编译器及系统环境进行必要的设置才能完成应用程序的生成工作。本小节将介绍生成独立可执行应用程序的编译方法。

6.3.1 命令行编译

在生成可执行应用程序时，可以通过命令行完成程序的编译，也就是说，可以在 MATLAB 的命令行窗口下或者操作系统的控制台方式下，通过 MEX 指令完成应用程序的编译。在前面的章节介绍了 MEX 指令，读者应该知道 MEX 指令是通过选项文件完成程序的编译工作的，在编译 MEX 文件时，需要针对不同的编译器选择不同的选项文件，MAT 数据文件应用程序是这样，MATLAB 计算引擎应用程序也是如此。

和生成 MEX 文件不同，生成 MEX 文件时，只要在 MATLAB 中完成编译器的配置工作就可以直接使用 MEX 指令完成 MEX 文件的编译工作。因为配置编译器的时候，需要将编译器的选项文件改名为 mexopts.bat 选项文件，并且拷贝到了系统环境目录下。不过在系统路径下 mexopts.bat 文件不能完成 MATLAB 计算引擎应用程序的编译，必须通过 MEX 命令行，指定具体的选项文件才可以。为了便于说明编译过程，这里使用一个 Fortran 语言的计算引擎应用程序作为示例。

例 6-2 Fortran 语言计算引擎应用——fengdemo.f。

```
001       C
```

```
002     C     fengdemo.f
003     C
004           program main
005     C
006     C     调用的函数
007     C
008           integer engOpen, engGetVariable, mxCreateDoubleMatrix
009           integer mxGetPr
010     C
011     C     声明变量
012     C
013           integer ep, T, D, V
014           real*8 time(11), dist(11), finitV
015           integer engPutVariable, engEvalString, engClose
016           integer temp, status
017           data time / 0.0,0.5,1.0,1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0 /
018           write(6,*) '开始 MATLAB 计算引擎应用程序.....'
019     C
020     C     打开计算引擎
021     C
022           ep = engOpen('\0')
023           if (ep .eq. 0) then
024               write(6,*) '无法打开计算引擎!'
025               stop
026           endif
027     C     创建数据
028           T = mxCreateDoubleMatrix(1, 11, 0)
029           call mxCopyReal8ToPtr(time, mxGetPr(T), 11)
030           finitV = 20.0
031           V = mxCreateScalarDouble(finitV)
032     C
033     C     将变量 T,V 放置到 MATLAB 工作空间
034     C
035           status = engPutVariable(ep, 'T', T)
036           if (status .ne. 0) then
037               write(6,*) '无法设置变量 --- T!'
038               stop
039           endif
040           status = engPutVariable(ep, 'V', V)
```

```
041             if (status .ne. 0) then
042                 write(6,*) '无法设置变量 --- V!'
043                 stop
044             endif
045       C
046       C     执行 MATLAB 指令
047       C
048             if (engEvalString(ep, 'D = V.*T + 0.5.*(-9.8).*T.^2;') .ne. 0)
049            *  then
050                 write(6,*) '无法正确执行指令!'
051                 stop
052             endif
053       C     绘制结果
054             if (engEvalString(ep, 'plot(T,D);') .ne. 0) then
055              write(6,*) '无法执行指令!'
056                 stop
057             endif
058             if (engEvalString(ep, 'title("位置 vs. 时间")') .ne. 0) then
059                 write(6,*) '无法执行指令!'
060                 stop
061             endif
062             if (engEvalString(ep, 'xlabel("时间 (秒)")') .ne. 0) then
063                 write(6,*) '无法执行指令!'
064                 stop
065             endif
066             if (engEvalString(ep, 'ylabel("位置 (米)")') .ne. 0)then
067                 write(6,*) '无法执行指令!'
068                 stop
069             endif
070             write(6,*) '按回车键将继续.....
071             pause
072       C     获取计算结果
073             D = engGetVariable(ep, 'D')
074             call mxCopyPtrToReal8(mxGetPr(D), dist, 11)
075             write(6,*) 'MATLAB 的计算结果如下:'
076             write(6,*) '   time(s)   distance(m)'
077             do 10 i=1,11
078                 print 20, time(i), dist(i)
079       20        format(' ', G10.3, G10.3)
```

```
080        10    continue
081              write(6,*) '按回车键将关闭计算引擎.....'
082              pause
083        C     关闭图形窗体
084              if (engEvalString(ep, 'close;') .ne. 0) then
085                  write(6,*) '无法执行指令!'
086                  stop
087              endif
088        C
089        C     关闭计算引擎
090        C
091              status = engClose(ep)
092              if (status .ne. 0) then
093                  write(6,*) '无法关闭计算引擎!'
094                  stop
095              endif
096              write(6,*) 'Done!'
097        C     好习惯!
098              call mxDestroyArray(T)
099              call mxDestroyArray(D)
100              stop
101              end
```

从程序的角度上看例 6-2 的代码是相当完整的，不过这里详细介绍一下编译该程序时需要注意的一些问题。在进行 Fortran 语言的计算引擎程序编译之前，一定要在操作系统中正确地安装了 Fortran 语言的编译器。无论是在 Windows 操作系统还是在 UNIX 操作系统上，MATLAB 支持的 Fortran 语言编译器都要比其支持的 C 语言编译器类型要少，而且 Fortran 语言编译器的路径——搜索路径和库文件路径等环境变量一定要切实地在操作系统中进行了注册。

和编译生成 MAT 数据文件引用程序非常类似，编译计算引擎应用程序也必须指定相应的 MEX 编译选项文件才能够正确生成用户需要的产品。不过，对于同一种编译器，生成 MAT 数据文件应用程序和生成 MATLAB 计算引擎应用程序的 MEX 选项文件是同一个，也就是说，利用 MSVC++ 6 生成 MAT 数据文件应用程序使用选项文件 msvc60engmatopts.bat，则在生成计算引擎应用时也使用这个选项文件。

然后，在 MEX 指令中，需要使用-f 命令行参数定义必要的选项文件，进行 MATLAB 计算引擎应用程序的编译使用的选项文件位于%MATLABROOT%\bin\win32\mexopts 路径下，这些选项文件都具有共同的后缀名称和扩展名——engmatopts.bat，例如 Msicrosoft Visual C++ 6 使用的选项文件就是 msvc60engmatopts.bat，而在例 6-2 中使用的 Fortran 语言的选项文件则是 df60engmatopts.bat。当然，也可以将相应的选项文件拷贝到当前的工作路径下，并且重命名为 mexopts.bat，这样就不必在 MEX 命令行中使用-f 命令行参数了。为了编译并

执行例 6-2 的代码，需要在 MATLAB 命令行下键入下面的指令：

```
>> mex -f %MATLABROOT%\bin\win32\mexopts\df60engmatopts.bat fengdemo.f
```

其中，%MATLABROOT%是程序员使用的 MATLAB 的安装路径。如果 fengdemo.f 程序中没有任何错误，则 MEX 指令将成功地将 fengdemo.f 源代码编译生成为可执行的应用程序。如果存在任何错误或者警告提示，请读者对照例子的源代码，仔细修改直到编译通过为止。接着，就可以运行得到的可执行应用程序。

在 MATLAB 命令行窗口中键入如下命令：

```
>> dir *.exe
fengdemo.exe
>> !fengdemo &
```

在 Windows 的命令行提示符中会显示以下信息：

```
开始 MATLAB 计算引擎应用程序.....
```

当显示下面的信息时，例子将利用 MATLAB 进程绘制出计算的结果，如图 6-3 所示。

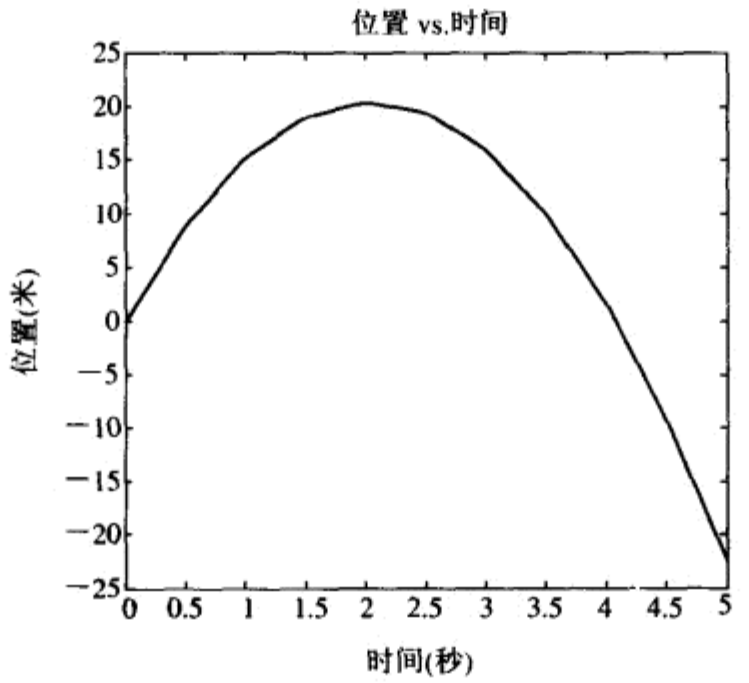


图 6-3 MATLAB 绘制出来的应用程序结果

```
按回车键将继续.....
Fortran Pause - Enter command<CR> or <CR> to continue.
```

在 Windows 的命令行提示符中按回车键继续程序的运行，则程序将通过 073~080 行的代码将计算的结果 D 读回到 Fortran 的应用程序中，于是利用 Fortran 语言将计算结果输出到了命令行提示符中：

```
MATLAB 的计算结果如下：
  time(s)   distance(m)
  0.000      0.000
  0.500      8.78
  1.00       15.1
  1.50       19.0
  2.00       20.4
  2.50       19.4
```

3.00	15.9
3.50	9.97
4.00	1.60
4.50	-9.23
5.00	-22.5

当再次出现提示时，按回车键将关闭 MATLAB 计算引擎，这时将退出整个程序的运行。

6.3.2　使用集成开发环境

如果读者使用集成开发环境来进行计算引擎应用程序的开发，则需要按照选择的编译器不同，根据不同的集成开发环境设置程序的开发过程。无论是 C 语言应用程序还是 Fortran 语言应用程序都具有自己的优势，所以对于程序员来说，需要根据要求选择合适的编译器以及开发环境。

本小节利用例 6-1 的代码——simpleengdemo.c 和 Microsoft Visual Studio 6 来讲解使用集成开发环境进行计算引擎应用程序编译的完整过程。

首先启动 Visual Studio，然后执行“File”菜单下的“New”命令，在弹出的对话框中选择“Win32 Console Application”作为即将创建的应用程序类型，在“Project name”文本输入框处设定应用程序的工程名称，也就是未来生成的可执行程序的名称，如图 6-4 所示。

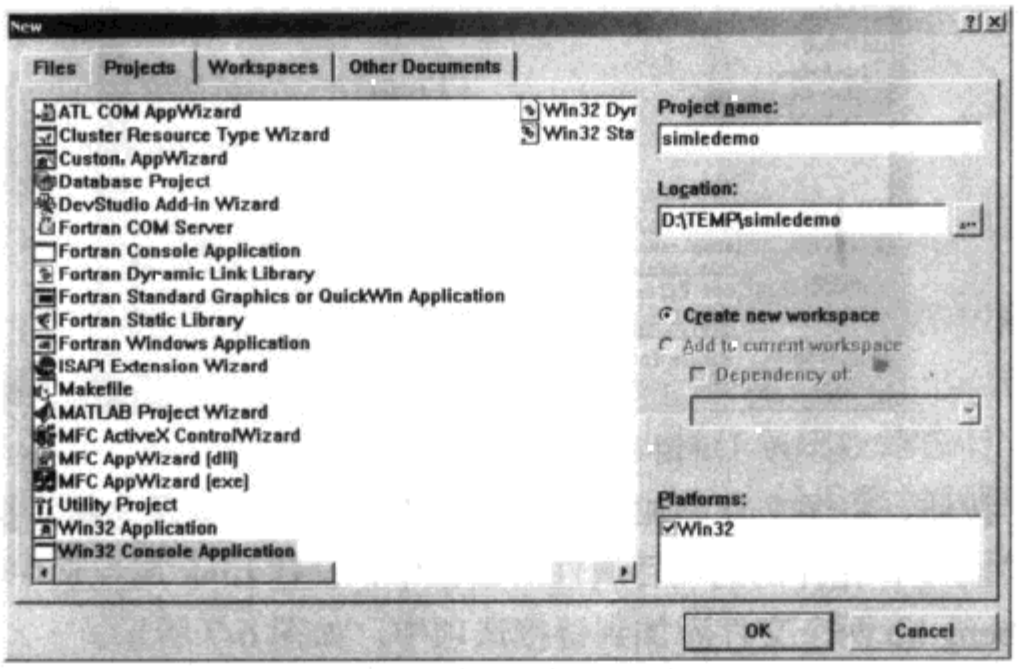


图 6-4　新建项目对话框

单击“OK”按钮之后，在弹出的对话框中选择创建空的项目文件选项，并且单击对话框的“Finish”按钮，这样 Visual C++将自动创建一个空白的项目，等待添加其它类型文件。

执行“Tools”菜单下的“Options”命令，在弹出的对话框中选择“Directories”属性页。在该属性页下设置 Include 路径属性和 LIB 路径属性，分别将 MATLAB 外部接口应用程序的 Include 路径和 LIB 路径添加到这里。MATLAB 外部接口应用程序头文件所在的 Include 路径为 E:\MATLAB6P5P1\EXTERN\INCLUDE，注意如果开发应用程序时使用了 C++语言，则还要将 E:\MATLAB6P5P1\EXTERN\INCLUDE\CPP 路径添加到 Include 路径下。在设置 LIB 路径时，需要根据不同的编译器选择 LIB 路径，MATLAB 外部接口应用程序针对 MSVC++ 6.0 的库文件路径是 E:\MATLAB6P5P1\EXTERN\LIB\WIN32\MICROSOFT\MSVC60。将上述的两个路径分别在如图 6-5 所示的对话框中设置完毕。

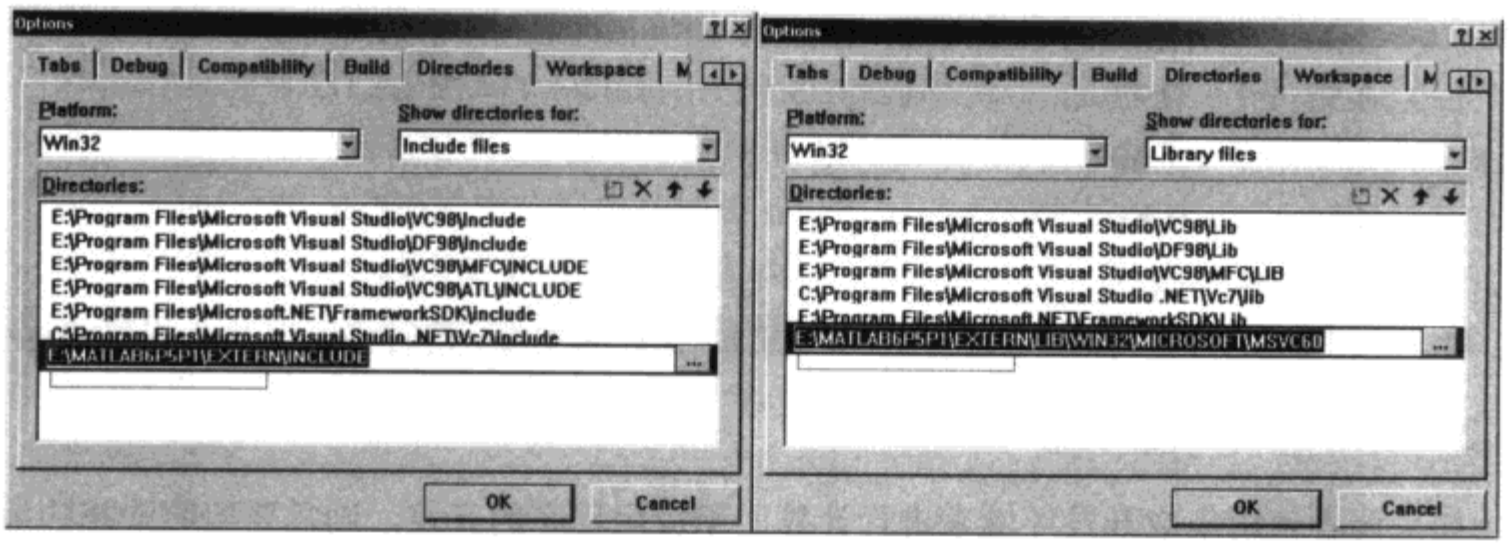


图 6-5 设置必要的 Include 路径和 LIB 路径

设置完毕后，单击“OK”按钮确认并退出 Options 对话框。

向项目工程中添加必要的源代码文件，如果用户的源代码文件不存在，则可以直接执行“File”菜单下的“New”命令，创建一个新的源代码文件并将其添加到项目工程中。如果需要添加的用户源代码文件已经存在，则执行“Project”菜单下的“Add to Project”子菜单下的“Files”命令，在弹出的对话框中选择图 6-6 的源代码文件 simpleengdemo.c。

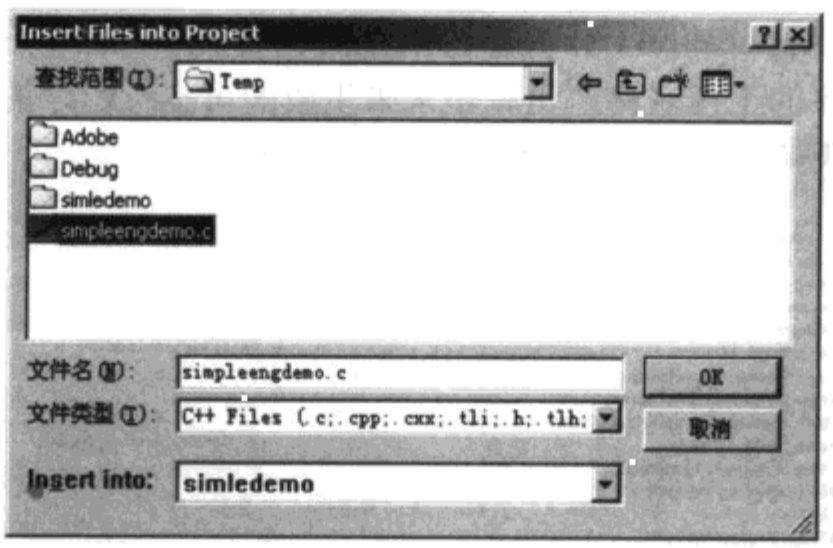


图 6-6 选择源代码文件

单击“OK”按钮，将源文件添加到项目工程中。在进行编译之前，执行“Project”菜单下的“Setting”命令，在弹出的项目属性对话框中，设置 Link 属性页中链接库文件选项，将 libmx.lib 和 libeng.lib 两个文件添加到链接选项中，如图 6-7 所示。

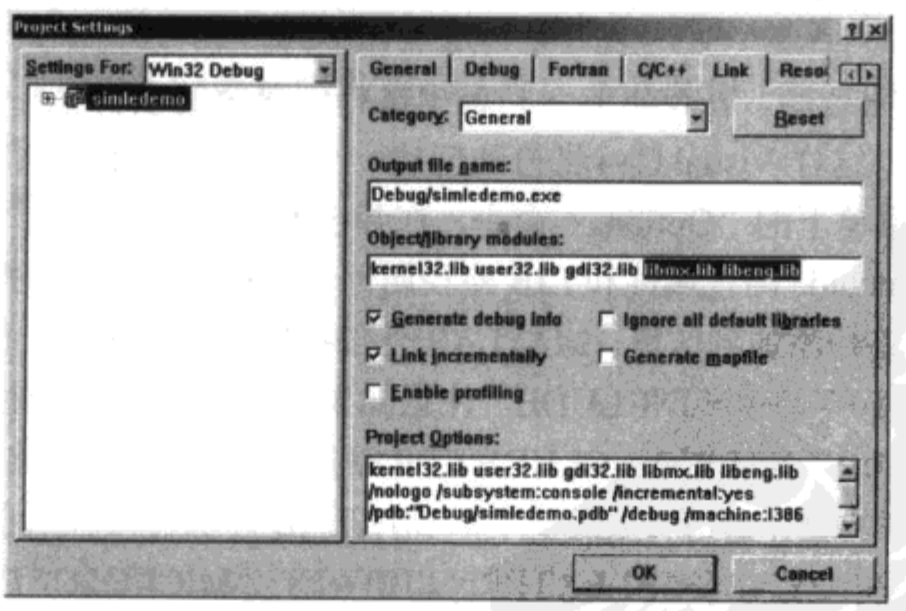


图 6-7 添加必要的库文件

注意：

如果在 MATLAB 计算引擎应用程序中，使用了 MAT 数据文件应用接口函数，则需要在设置链接选项时将 libmat.lib 库文件添加到项目中。

单击“OK”按钮之后确认，这时就可以直接生成可执行应用程序了。执行“Build”菜单下的“Build simledemo.exe”命令，如果应用程序没有任何错误，则 Visual C++将生成相应的可执行文件，也可以在 Visual C++中直接完成程序的运行。

如果读者使用其它的C语言编译器集成开发环境进行MATLAB 计算引擎应用程序的开发，则其过程和使用 Visual C++的过程是类似的，只要注意在相应的链接选项增加必要的库文件，设置好编译器的系统环境变量应该就没有别的问题了。

6.4　计算引擎应用示例

通过前面章节的介绍，读者应该能够基本掌握MATLAB 计算引擎应用程序的开发方法，其实MATLAB 计算引擎应用程序就是eng 函数的使用方法。在本小节，将通过一些MATLAB 计算引擎应用示例来进一步演示 MATLAB 计算引擎应用。

例 6-3　多项式拟合——cfengdemo.c。

在 curvedata.mat 文件中包含两个数据 x 和 y，下面通过例子演示利用 MATLAB 的多项式拟合能力进行数据拟合计算的过程。这里利用了在本书第 5 章例 5-6 mutil.h 文件包含的 ReadArray 和 WriteArray 函数。以下是例 6-3 的源代码：

```
001    /*
002     * file: cfengdemo.c
003     *
004     *   本文件演示了从 curvedata.mat 文件中读取变量 x 和 y，并且利用
005     *   函数 CalculateFcn 完成多项式拟合计算的过程
006     *   读取变量的函数和前面介绍的 MAT 文件例子中的函数完全一致
007     *
008     *   编译该文件时：
009     *
010     *       >> mex -f optsfile cfengdemo.c mutil.c
011     *
012     */
013    #include "engine.h"
014    #include "mutil.h"
015    #define BUFFERLEN 1024
016
017    /* CalculateFcn 函数完成下列的工作：
```

```
018         1. 接受 x 和 y 数据作为输入参数
019         2. 进行 5 阶多项式拟合计算
020         3. 在不同的子图内绘制原始数据和拟合运算数据
021         4. 返回拟合运算结果 */
022     mxArray * CalculateFcn( mxArray *x, mxArray *y )
023     {
024         Engine   *ep;
025         mxArray *equation;
026         char       flag[1],buffer[BUFFERLEN];
027         int        status;
028
029         /* 打开计算引擎 */
030         ep = engOpen( NULL );
031         if( ep == (Engine *)NULL )
032             fprintf( stderr, "错误：无法打开 MATLAB！\n" );
033         fprintf(stdout,"计算引擎 MATLAB 是可见的.....\n");
034         fprintf(stdout,"隐藏它!\n 按任意键继续.....\n");
035         getch();
036         /* 隐藏计算引擎的窗口 */
037         status = engSetVisible(ep,false);
038         if(status != 0)
039         {
040             fprintf( stderr, "错误：无法隐藏计算窗口.\n" );
041             return( NULL );
042         }
043         fprintf(stdout,"进行数据处理计算.....");
044         /* 将数据传输到 MATLAB 工作空间*/
045         status = engPutVariable( ep, "x", x );
046         if( status != 0 )
047         {
048             fprintf( stderr, "错误：无法将数据传输到 MATLAB 工作空间.\n" );
049             return( NULL );
050         }
051         status = engPutVariable( ep, "y", y );
052         if( status != 0 )
053         {
054             fprintf( stderr, "错误：无法将数据传输到 MATLAB 工作空间.\n" );
055             return( NULL );
```

```
056         }
057         /* 设置输出缓存区 */
058         engOutputBuffer( ep, buffer, BUFFERLEN );
059         /* 执行多项式拟合 */
060         status = engEvalString( ep, "k5=polyfit(x,y,5);y5=polyval(k5,x);");
061         if( status != 0 )
062         {
063             /* 进行错误检测.....*/
064             fprintf( stderr, "错误：无法运行 MATLAB 命令.\n" );
065             fprintf( stderr, "%s\n", buffer );
066             return( NULL );
067         }
068         engEvalString(ep,"subplot(2,1,1);plot(x,y,'b',x,y5,'r');");
069         engEvalString(ep,"xlabel('x'); ylabel('y');title('x Vs y');axis tight;");
070         engEvalString(ep,"legend('Original','Fitted');");
071         engEvalString(ep,"subplot(2,1,2);bar(x,y-y5,'r');xlabel('x');ylabel('y-y5');");
072         engEvalString(ep,"title('Residuals');axis tight;");
073         /* 获取计算的结果 */
074         equation = engGetVariable( ep, "k5" );
075         fprintf(stdout,"Done!\n");
076         fprintf(stdout,"按任意键继续.....\n");
077         getch();
078         /* 关闭计算引擎 */
079         status = engClose( ep );
080         if( status != 0 )
081         {
082             fprintf( stderr, "错误：无法关闭 MATLAB 计算引擎.\n" );
083         }
084         return( equation );
085     }
086     /* 主函数 */
087     void main()
088     {
089         mxArray *xdata, *ydata, *eqn;
090         char        matfile[] = "curvedata.mat";
091         /* 从数据文件 curvedata.mat 中获取变量 x & y   */
092         xdata = ReadArray( matfile, "x");
093         ydata = ReadArray( matfile, "y");
```

```
094
095         /* 计算多项式拟合计算：
096         函数同时绘制原始数据和拟合计算数据   */
097         eqn = CalculateFcn( xdata,ydata);
098         /* 将拟合计算结果保存到 curvedata.mat */
099         WriteArray( matfile, "k5",eqn );
100     }
```

例 6-3 的代码中使用了两个比较特别的 eng 函数：在第 037 行的代码中，使用了 engSetVisible 函数，该函数只能够用于 C 语言计算引擎应用程序中，它的功能是使当前的 MATLAB 计算引擎窗口成为不可见，也就是隐藏起来。该函数的定义如下：

```
int engSetVisible(Engine *ep, bool value);
```

该函数的第一个参数是 MATLAB 计算引擎的指针 ep，第二个参数是一个布尔类型的数据 value，如果该数据为逻辑真时，则 MATLAB 的窗口为可见的状态，也就是显示在计算机平台上。如果该数据为逻辑假时，则 MATLAB 的窗口为不可见的状态，也就是隐藏起来。所以当例 6-3 运行到 037 行时，MATLAB 计算引擎窗口就隐藏起来了。如果函数成功运行，则函数返回值为整数 0。

如果需要判断当前的 MATLAB 计算引擎窗口是否可见，可以使用 engGetVisible 函数，该函数的定义如下：

```
int engGetVisible(Engine *ep, bool *value);
```

该函数的参数和 engSetVisible 的参数类似，不同的是第二个输入参数为布尔类型变量指针，当 MATLAB 计算引擎窗口不可见时，该数值为逻辑假，否则为逻辑真。

在例 6-3 代码的 058 行调用了函数 engOutputBuffer ，该函数的作用是定义一个字符串缓冲区，该缓冲区将获取 engEvalString 函数在运行 MATLAB 指令过程中，MATLAB 指令本应该输出到 MATLAB 窗口中的文本信息。该函数在 C 语言中的定义如下：

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

该函数中有三个参数，分别是计算引擎的指针 ep、缓冲区字符串指针 p 和缓冲区的长度 n。该函数在 Fortran 语言中的定义如下：

```
integer*4 function engOutputBuffer(ep, p)
integer*4 ep
character*n p
```

该函数在 Fortran 语言中的定义只有两个参数，分别为计算引擎的指针 ep 和作为缓冲区的字符串指针 p，不过 p 的长度被预先定义为 n。

上面的两个函数是在 MATLAB 计算引擎应用程序中比较常用的两个。其余的 eng 函数说明请参阅 MATLAB 的帮助文档。

编译运行例 6-3 的代码：

```
>> mex cfengdemo.c mutil.c
>> !cfengdemo &
```

在运行程序的过程中，MATLAB 计算引擎被打开，然后隐藏起来，并且完成了多项式

拟合的计算，得到计算的图形如图 6-8 所示。

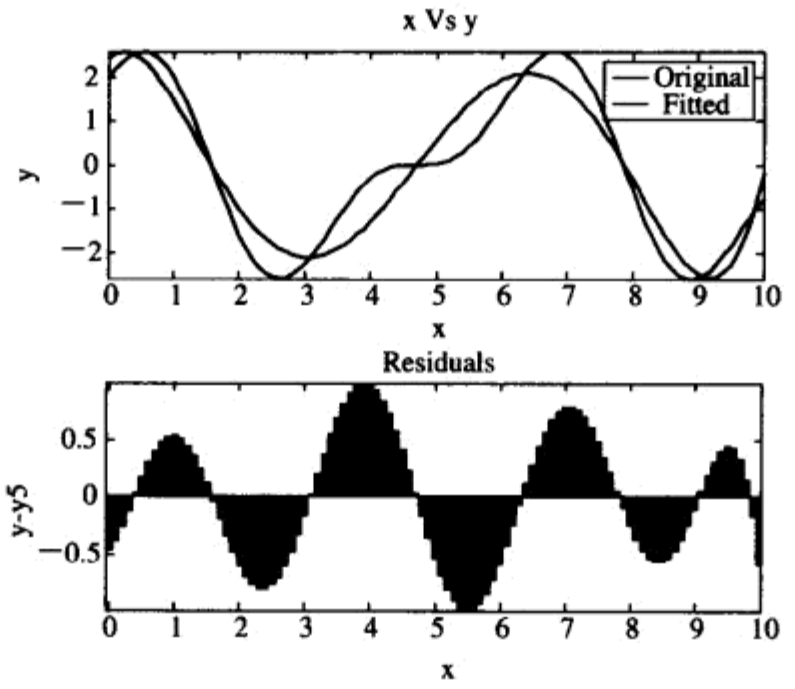


图 6-8　曲线拟合得到的图形

如果重新加载 MATLAB 的数据文件可以看到数据文件中增加了相应的变量 k5：

```
>> load curvedata
>> whos
  Name        Size                    Bytes  Class
  k5          1x6                        48  double array
  x           1x101                     808  double array
  y           1x101                     808  double array
Grand total is 208 elements using 1664 bytes
>> k5
k5 =
  Columns 1 through 4
    0.0069   -0.1636    1.2773   -3.5214
  Columns 5 through 6
    1.4046    2.4537
```

例 6-4　常系数微分方程组的初值问题——fodeinit.f。

常系数微分方程组的初值问题是科学计算领域中经常要解决的一类问题，解决此类问题就是利用类似龙格库塔法之类的数值算法求解常系数微分方程的数值解。MATLAB 提供了常用的数值求解算法，这些算法将极大提高解决工程问题的能力和效率。

假设有这样的常系数微分方程：

$$dy(dy_1)/dt - \mu(1-y_1^2)dy_1/dt + y_1 = 0$$

这是一个二阶常系数微分方程，可以写作是两个一阶的微分方程构成的方程组：

$dy_1/dt = y_2$

$dy_2=dt=\mu(1-y_1^2)y_2-y_1$

其中，$\mu=0.9$。而为了求得微分方程的数值解，必须具备相应的初值，这里定义初值为 $y_1=2$，$y_2=0$。下面将利用 Fortran 语言计算引擎应用程序解决上述的问题，源代码如下：

```
001     C
002     C       fodeinit.f
003     C   计算常系数微分方程的初值问题
004     C
005         program main
006     C
007     C   调用的函数
008     C
009         integer engOpen, engOutputBuffer,engEvalString
010         integer engClose
011     C   变量的声明
012         integer ep
013         character*128 buffer
014         integer status
015     C
016     C   打开计算引擎
017     C
018         ep = engOpen('\0')
019         if (ep .eq. 0) then
020             write(6,*) '无法打开计算引擎!'
021             stop
022         endif
023     C
024     C   设置输出缓冲
025     C
026         status = engOutputBuffer(ep , buffer)
027         if(status .ne. 0) then
028             write(6,*) '无法设置计算引擎的输出缓冲!'
029             stop
030         endif
031
032         status = engEvalString(ep,
033        *'vdp = inline("[y(2);0.9*(1-y(1)^2)*y(2)-y(1)];"
```

```
034          *,"t","y")')
035          if(status .ne. 0) then
036              write(6,*) '无法创建微分方程组!'
037              stop
038          endif
039      C   看看输出了什么.....
040          write(6,*) buffer
041      C   求解微分方程
042          status = engEvalString(ep,'[t,y] = ode45(vdp,[0 10],[2;0]);');
043          if(status .ne. 0) then
044              write(6,*) '无法求解微分方程组!'
045              stop
046          endif
047          call engEvalString(ep,'plot(t,y);legend("Y1","Y2")')
048      C   获取计算结果数据
049      C   .....
050          write(6,*) '按回车键将关闭计算引擎.....'
051          pause
052      C
053      C   关闭计算引擎
054      C
055          status = engClose(ep)
056          if (status .ne. 0) then
057              write(6,*) '无法关闭计算引擎!'
058              stop
059          endif
060          write(6,*) 'Done!'
061          stop
062          end
```

在上面的程序中，主要利用了计算引擎完成常系数微分方程组的数值解求解，在这里并没有使用任何 Fortran 语言的计算能力，所以读者可以在程序中相应的位置进行适度的修改，或者将求解微分方程的能力应用到别的代码中。

编译运行例 6-4 的代码：

```
>> mex -f E:\MATLAB6p5p1\bin\win32\mexopts\df60engmatopts.bat fodeinit.f
>> !fodeinit &
```

计算得到的曲线输出如图 6-9 所示。

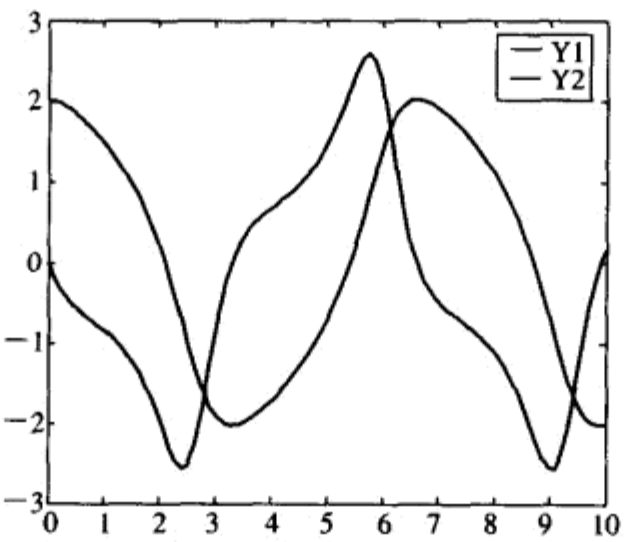


图 6-9 计算得到的曲线

在运行程序的时候，程序首先需要在命令行提示符中输出下面的信息：

```
vdp =
    Inline function:
    vdp(t,y) = [y(2);0.9*(1-y(1)^2)*y(2)-y(1)]
```

这些信息就是通过 engOutputBuffer 函数设置的缓冲区的内容，它是创建内建函数对象时得到的输出。

6.5 本章小结

本章详细讨论了 MATLAB 计算引擎应用程序的开发过程以及 eng 函数的使用方法。MATLAB 计算引擎应用在所有集成 MATLAB 算法到 C 语言或者 Fortran 语言的各种办法中是最简单和最容易实现的一种，它能够快速地将 MATLAB 算法集成到 C 语言或者 Fortran 语言的应用程序中。用户可以通过将 C 语言或者 Fortran 开发的前端程序作为客户端，而将 MATLAB 作为后台的服务程序提供计算能力支持。虽然 MATLAB 计算引擎应用程序无法脱离 MATLAB 环境使用，它不是一个真正的独立的可执行应用程序，但是掌握此类外部接口应用也是非常重要的内容。本章通过具体示例说明了函数的使用方法，而具体的函数应用说明请读者参阅 MATLAB 的帮助文档。

其它将 MATLAB 算法集成到 C 语言或者其它开发环境的方法将在《MATLAB 应用程序集成与开发》一书中作详细讲解。

练 习

1. 求解系统表达式。

编写 C 语言或者 Fortran 语言计算引擎应用程序求解系统表达式，原始数据都保存在数据文件中，要求完成如下功能：

- 打开 datafile.mat 数据文件。
- 在引擎应用程序中从数据文件获取 x 和 y 变量。
- 创建 MATLAB 计算引擎。

- 将变量 x 和 y 传递到 MATLAB 中。
- 执行 MATLAB 表达式 result = mldivide(x,y)，求解系统方程。
- 将计算结果保存到同一个数据文件中。

2. 集成开发环境的编译。

请尝试使用自己熟悉的集成开发环境完成计算引擎应用程序的编译，如例 6-3 或者例 6-4 的代码。

第 7 章　在 MATLAB 中调用 Java

Java 语言是一种流行的面向对象的高级编程语言，能够完成各种类型的应用程序开发。MATLAB 与 Java 语言是密不可分的，在 MATLAB 软件发布时就包含了一个 Java 虚拟机，MATLAB 工具箱中很多图形化用户界面工具都利用了 Java 语言的能力,也就是说,MATLAB 本身可以作为 Java 的接口存在。本章将详细讲述在 MATLAB 的 M 语言编程过程中，如何加载 Java 的类，创建相应的对象，并且执行对象的方法。

本章重点内容

- MATLAB 的 Java 接口;
- 引入 Java 类;
- 创建对象;
- 执行方法。

7.1　MATLAB 的 Java 接口概述

MATLAB 软件和 Java 之间的关系是非常密切的，从 MATLAB 5.3 开始，每一个客户安装的 MATLAB 中都包含了 Java 虚拟机，而且 MATLAB 的图形用户界面就是利用 Java 语言进行开发的。在 MATLAB 中可以直接调用 Java 语言开发功能丰富的应用程序，新版本的 MATLAB 工具箱中就包含了很多使用Java语言开发的图形用户界面工具,例如COM Builder 的用户界面 comtool 等。利用 MATLAB 的 Java 接口可以完成下列工作：

- 调用 Java API 类(class)和包(package)，完成核心功能；
- 调用第三方 Java 类(class)；
- 在 MATLAB 环境下创建 Java 对象；
- 通过 Java 语法或者 MATLAB 语法使用 Java 对象的方法；
- 在 Java 对象和 MATLAB 之间交互数据。

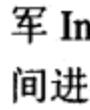
Java 语言可以用来填补 MATLAB 的一些功能上的空白，例如 MATLAB 本身在文件的 I/O 能力上有所欠缺，可以通过 Java 来完成适当的补充，MATLAB 创建的图形用户界面功能单一，也可以利用 Java 来弥补。特别是由于 Java 语言自身的优势，完全可以通过 Java 语言获取大量的来自于互联网或者数据库的数据,而MATLAB本身的优势是进行数据分析、科学计算。于是，将两者有机地结合使用，充分发挥各自的优势，将极大提高工作效率，节约开发时间和成本。

目前 Java 已经发展到了 Java 2 版本，从 Sun 公司的官方网站上可以下载最新版本的 JDK(Java 开发包)供用户开发程序。不过，MATLAB 推荐使用其中相对稳定的 JDK 版本进行 MATLAB 的 Java 程序开发。在 MATLAB 中键入下面的指令可以查看当前 MATLAB 使

用的 Java 虚拟机的版本：

```
>> version -java
ans =
Java 1.3.1_01 with Sun Microsystems Inc. Java HotSpot(TM) Client VM
    (mixed mode)
```

请读者按照自己的计算机上显示的 Java 虚拟机的版本选择 JDK 的版本，避免利用新版的 JDK 开发的程序而出现无法在本地 MATLAB 上正确运行的错误。

 注意：

在 Microsoft Windows XP 操作系统中进行 MATLAB 的安装时，MATLAB 会自动检测当前操作系统下使用的 Java 虚拟机的版本，如果操作系统下 Java 虚拟机的版本不能满足 MATLAB 的需要，则安装程序会自动使用 MATLAB 安装光盘上的 Java 虚拟机安装包来安装相应的 Java 虚拟机。

7.2　Java 语言概述

在详细介绍 MATLAB 的 Java 接口前首先简要回顾一下 Java 语言。如果读者对 Java 语言非常熟悉，则可以直接略过本小节的内容，如果读者对 Java 语言不甚了解，则需要仔细阅读本小节内容，熟悉 Java 语言程序的开发过程。

 提示：

Java 语言本身是一个复杂而且庞大的体系，不可能通过短短几页文字就能说清楚 Java 语言的使用方法。本小节仅讲述 Java 的程序运行过程，如果读者希望了解 Java 语言开发的具体技术细节，请参阅市面上的其它介绍 Java 语言的书籍。

Java 来自于 Sun 公司的一个叫 Green 的项目，其原先的目的是为家用消费电子产品开发一个分布式代码系统，这样可以把 E-mail 发给电冰箱、电视机等家用电器，对它们进行控制，和它们进行信息交流。起初准备采用 C++编程语言，但 C++太复杂，安全性差，最后基于 C++开发了一种新的语言 Oak(Java 的前身)，Oak 是一种用于网络的精巧而安全的语言，Sun 公司曾依此投标一个交互式电视项目，但结果被 SGI 打败。可怜的 Oak 几乎无家可归，恰巧这时 Mark Ardreesen 开发的 Mosaic 和 Netscape 启发了 Oak 项目组成员，他们又编制了 HotJava 浏览器，得到了 Sun 公司首席执行官 Scott McNealy 的支持，触发该语言进军 Internet，于是就有了 Java。目前 Java 语言正在日益成为客户机、服务器、数据库应用之间进行通信的“中间件”，而且 Java 也正在慢慢向嵌入式系统进军，使其在手持设备、移动通信、机顶盒系统等领域都能够看到 Java 的身影。

Java 本身并不简简单单是一种编程语言，它还是一个程序的运行环境(JVM)，一种开发平台和手段，例如 J2EE 和 J2ME 等。作为一种编程语言，Java 是一个优秀的完全面向对象的高级编程语言，它具有如下基本特征：

- 简单；
- 可移植；

- 面向对象；
- 解释型；
- 分布式；
- 高性能；
- 健壮；
- 多线程处理能力；
- 安全；
- 动态；
- 中性结构。

其中，最关键的特征就是面向对象、可移植、多线程处理能力，丰富的类库使其具备强大的联网能力。

例 7-1　简单的 Java 语言示例。

这里用一个简单的 Java 语言示例来演示开发 Java 程序的基本过程，该程序的作用是在控制台上输出一段文本。

```
001     //
002     // 第一个 Java 语言的例子,HelloWorld.java
003     //
004     public class HelloWorld{
005         public static void main(String[] args){
006             System.out.println("Hello Java World!");
007         }
008     }
```

编译运行该程序，在控制台方式下(命令行提示符)键入下面的指令：

```
javac HelloWorld.java
```

该指令完成了程序的编译。接着，在控制台方式下键入下面的指令：

```
java HelloWorld
```

则控制台中将显示运行的结果：

```
Hello Java World!
```

例 7-1 演示了一个完整的 Java 程序的编辑、编译和运行过程。首先 Java 语言的源程序是纯文本格式的文件，可以使用任何一种能够编辑标准纯文本的编辑器编辑该文件，注意，Java 语言源文件的扩展名为 java。

然后通过 javac 指令将源文件(扩展名为.java)编译成为二进制字节码文件，得到的文件为 HelloWorld.class。字节码文件必须通过 Java 虚拟机解释执行，如果在不同的计算机平台上具有能够解释执行字节码文件的虚拟机，就可以完成程序的跨平台应用。事实上，Java 语言就是这样做的。最后使用 Java 指令解释执行前面生成的字节码文件——HelloWorld.class。Java 指令就是调用 Java 虚拟机解释执行前面生成的字节码文件。

提示：

成功编译运行Java程序需要在用户的计算机平台中至少安装J2SDK的标准版，即J2SE，

并且在操作系统的环境变量中设置路径、classpath 等环境变量。J2SDK 可以直接从 Sun 公司的网站上下载，而关于如何设置路径、classpath 等环境变量请参阅 Java 的手册或者其它介绍 Java 语言的书籍。

Java 程序的运行必须具有 Java 虚拟机，Java 虚拟机负责解释编译生成的字节码文件，几乎每一种操作系统都有自己的 Java 虚拟机，这样就可以达到“一处编译，随处运行”的目的。在图 7-1 中揭示了 Java 应用程序编译和运行的过程。

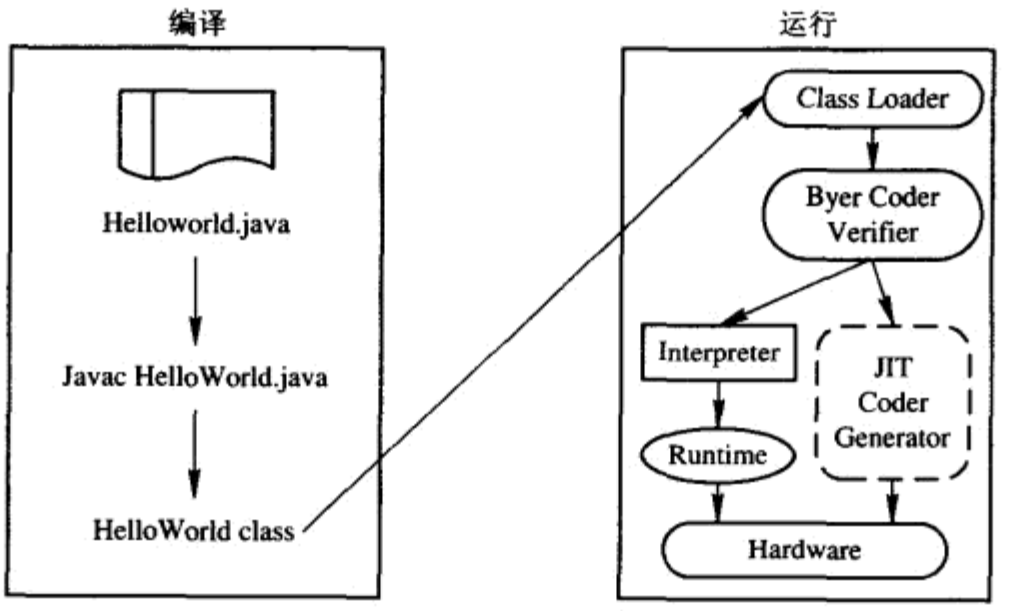


图 7-1　Java 程序的编译运行过程

例 7-1 中第 006 行的代码：

```
System.out.println('Hello Java World!");
```

使用了 J2SDK 提供的标准类库 System.out 当中的 println 方法。实际上进行 Java 语言的开发多数情况下就是利用已有的类库开发自己的类库以实现特殊的要求。而这些丰富的类库就可以极大地丰富 MATLAB 的功能。

7.3　Java 接口应用

通过前面章节的介绍用户已经大概了解了使用 Java 语言进行开发的必要性，也适当了解或者回顾了 Java 语言的有关知识。不过毕竟 Java 语言是一种独立的高级编程语言，而 M 语言才是MATLAB 的开发语言，两种编程语言之间进行混合编程必须按照一定的规则进行。在本小节将结合一些简单的命令和实例介绍 Java 接口的应用。

7.3.1　引入 Java 类

Java 是一种面向对象的高级编程语言，在面向对象的编程语言中，类和对象是其最基本的概念。如果需要创建对象，则必须有相应的类存在。Java 语言除了最基本的关键字以外，还提供了大量的预先编制好的类，应用 Java 就是通过 Java 语言的类完成各种各样的功能。一般可以将 Java 的类分为以下三种。

1. Java 内建类

由 Java 语言本身提供的类和类包，例如 Java.awt 类包，这些类包构成了 Java 语言的基

本应用，关于这些类包请参阅 Java 语言的文档说明。

2. 第三方定义类

除了 Java 内建类，有很多商业化或者免费的 Java 类包可以供程序开发人员使用，这些类包多数用来完成一些专门领域的特殊应用，例如 Mathworks 公司提供的工具箱类包。

3. 用户自定义类

通过面向对象语言的继承机制从已有的 Java 类派生或者使用 Java 语言直接开发的新类，这些新类一般可以完成用户需要的特殊功能，比如在例 7-1 中创建的 HelloWorld 类。

众所周知，MATLAB 依靠文件的目录结构来管理不同的 M 函数和工具箱函数，而 Java 则是通过 classpath 环境变量和类包完成 Java 类的管理。

一般地，由不同的类组合在一起就构成了类包，这些类包一般都安装在系统路径下面，通过系统变量 classpath 定义给操作系统。那么，在 MATLAB 中是如何管理和加载相应的 Java 类的呢？

在 MATLAB 的系统路径下存在一个文本文件——classpath.txt，该文件定义了 MATLAB 环境可以直接引入的 MATLAB 包。一般该文件位于%MATLABROOT%\toolbox\local\路径下，其内容为：

```
##
## FILE: classpath.txt
##
## Entries:
##      o path_to_jarfile
##      o [alpha,glnx86,sol2,UNIX,win32,mac]=path_to_jarfile
##      o $matlabroot/path_to_jarfile
##      o $jre_home/path_to_jarfile
##

$jre_home/lib/rt.jar
$jre_home/lib/i18n.jar
$jre_home/lib/swingall.jar
$matlabroot/java/patch
mac=$matlabroot/java/jarext/aquaDecorations.jar
$matlabroot/java/jar/util.jar
$matlabroot/java/jar/widgets.jar
$matlabroot/java/jar/beans.jar
$matlabroot/java/jar/hg.jar
$matlabroot/java/jar/icebrowserbean.jar
$matlabroot/java/jar/ide.jar
$matlabroot/java/jar/jmi.jar
$matlabroot/java/jar/mde.jar
```

$matlabroot/java/jar/mlwidgets.jar
$matlabroot/java/jar/mwswing.jar
$matlabroot/java/jar/mwt.jar
$matlabroot/java/jar/page.jar
$matlabroot/java/jar/services.jar
$matlabroot/java/jar/test.jar
$matlabroot/java/jar/timer.jar
$matlabroot/java/jar/toolbox/bdd.jar
$matlabroot/java/jar/toolbox/curvefit.jar
$matlabroot/java/jar/toolbox/dastudio.jar
$matlabroot/java/jar/toolbox/database.jar
$matlabroot/java/jar/toolbox/dials.jar
$matlabroot/java/jar/toolbox/dspblks.jar
$matlabroot/java/jar/toolbox/ecoder.jar
$matlabroot/java/jar/toolbox/filterdesign.jar
$matlabroot/java/jar/toolbox/images.jar
$matlabroot/java/jar/toolbox/instrument.jar
$matlabroot/java/jar/toolbox/matlab.jar
$matlabroot/java/jar/toolbox/mbc.jar
$matlabroot/java/jar/toolbox/mdldisc.jar
$matlabroot/java/jar/toolbox/nnet.jar
$matlabroot/java/jar/toolbox/physmod.jar
$matlabroot/java/jar/toolbox/reqmgt.jar
$matlabroot/java/jar/toolbox/rptgen.jar
$matlabroot/java/jar/toolbox/rptgencore.jar
$matlabroot/java/jar/toolbox/simulink.jar
$matlabroot/java/jar/toolbox/softinstruments.jar
$matlabroot/java/jar/toolbox/vr.jar
$matlabroot/java/jar/verctrl.jar
$matlabroot/java/jar/xml.jar
$matlabroot/java/jarext/dcxjp.zip
$matlabroot/java/jarext/mwjava2specific.jar
$matlabroot/java/jarext/SilkTest_Java1.jar
$matlabroot/java/jarext/audio.jar
$matlabroot/java/jarext/classes111.jar
$matlabroot/java/jarext/collections.jar
$matlabroot/java/jarext/commapi/comm.jar
$matlabroot/java/jarext/FTPProtocol.jar
$matlabroot/java/jarext/gnu_regexp.jar

$matlabroot/java/jarext/ice/ib5core.jar
$matlabroot/java/jarext/ice/ib5crypto.jar
$matlabroot/java/jarext/ice/ib5extra.jar
$matlabroot/java/jarext/ice/ib5http.jar
$matlabroot/java/jarext/ice/ib5https.jar
$matlabroot/java/jarext/ice/ib5js.jar
$matlabroot/java/jarext/ice/ib5ref.jar
$matlabroot/java/jarext/ice/ib5ri.jar
$matlabroot/java/jarext/ice/ib5swing.jar
$matlabroot/java/jarext/ice/ib5util.jar
$matlabroot/java/jarext/ice/ib5xalan.jar
$matlabroot/java/jarext/mousewheel.jar
$matlabroot/java/jarext/mwucarunits.jar
$matlabroot/java/jarext/saxon.jar
$matlabroot/java/jarext/vb20.jar
$matlabroot/java/jarext/windows.jar
$matlabroot/java/jarext/xalan.jar
$matlabroot/java/jarext/xercesImpl.jar
$matlabroot/java/jarext/xJava2NotShipping.jar
$matlabroot/java/jarext/xml-apis.jar

提示：

在 MATLAB 命令行中键入下面的指令可以快速定位 classpath.txt 文件所在的位置：

which classpath.txt

在默认情况下，classpath 文件中只是定义了 Java 内建类包和由 Mathworks 公司提供的工具箱类包，如果需要在 MATLAB 中使用第三方或者用户自定义的类包，则需要在该文件中具体定义，添加的规则如下：

■ 用户自定义类库：添加直到父目录的完整路径。例如对 d:\work\javaclasses\invoice.class 而言，在文件中添加 d:\work\javaclasses。

■ 包：添加直到最高级父目录的完整路径。例如对 d:\work\com\mw\transactions 而言，在文件中添加 d:\work。

■ JAR 文件中的类库：添加完整的路径、完整的文件名。例如对 d:\work\classes\utilpkg.jar 而言，在文件中添加 d:\work\classes\utilpkg.jar。

引入 Java 类包之后就可以直接创建 Java 对象。

注意：

在 classpath.txt 文件中定义的类路径中必须包含“.”，它代表当前的工作路径。另外，可以通过编写 classpath.txt 文件中的内容，为 MATLAB 指定其它版本的 Java 虚拟机，但是，Mathworks 不推荐用户去做这种事情，以免出现不必要的错误。

7.3.2　创建 Java 对象

熟悉 Java 语言的读者对创建 Java 对象丝毫不陌生，在本小节将介绍在 MATLAB 中创建 Java 对象以及相关函数的方法。

创建 Java 对象的方法非常简单，直接用 Java 类(完整的名称)就可以定义 Java 对象了，不过 MATLAB 还提供了 javaObject 函数用于创建 Java 类对象，参见下面的例子。

例 7-2　在 MATLAB 中创建 Java 对象。

在 MATLAB 命令行中键入下面的指令：

```
>> %直接创建
>> fa = java.awt.Frame('Frame A');
>> %用 javaObject 方法
>> fb = javaObject('java.awt.Frame','Frame B');
>> whos
  Name        Size                    Bytes  Class
  fa          1x1                            java.awt.Frame
  fb          1x1                            java.awt.Frame
Grand total is 2 elements using 0 bytes
```

通过上面的简单指令，创建了两个 Java 对象：fa 和 fb，它们的类型都是 java.awt.Frame 类。使用 Java 类直接创建对象时，必须给出完整的 Java 类名称，同样使用 javaObject 函数也是如此。其中 javaObject 函数的定义如下：

```
J = javaObject('class_name',x1,...,xn)
```

其中，x1～xn 为 Java 类的构造函数需要的输入参数。

使用 javaObject 函数的好处是，在创建对象时允许用户输入的类名称超过系统定义的最大命名长度。例如在 Windows 平台上，最大的命令字符串长度为 63，则 javaObject 函数可以创建名称超过 63 个字符的类的对象。而且 javaObject 可以通过输入、输出参数与用户的应用交互。例如，在 MATLAB 命令行窗口中键入下面的指令：

```
>> string = 'Java Class';
>> classname = 'java.lang.String';
>> strObj = javaObject(classname,string)
 strObj =
Java Class
```

提示：

关于不同系统平台上 MATLAB 最大命名字符串的长度，可以通过执行 namelengthmax 函数来查看。例如在 Windows 平台上键入命令：

```
>> namelengthmax
ans =
     63
```

如果每次创建 Java 类对象的时候都需要给出完整的类名称，那么创建类对象的过程就太繁琐了。因此，可以在 Java 语言中使用 import 命令导入必要的类包，这样声明对象的时候就不必给出完整的类名称了。同样，在 MATLAB 的 M 语言中也是如此。在 M 语言中也存在 import 指令，该指令可以将 Java 类包加载到 MATLAB 的工作空间中，这样在 M 语言中声明类的对象时，就不用给出完整的类名称了。例如，在 MATLAB 中声明 Frame 类的对象，该类属于 java.awt 类包，于是就可以这样做：

```
>> import java.awt.Frame
>> fa = Frame('Frame A');
```

可以使用 import 指令查看已经加载的类：

```
>> import
ans =

    'java.awt.Frame'
```

前面的操作既可以导入一个类，也可以导入整个类包：

```
>> import java.awt.*
>> fb = Frame('Frame B');
>> import
ans =
    'java.awt.Frame'
    'java.awt.*'
```

创建了对象之后，需要设置对象的属性。在 Java 语言中，设置类对象的属性必须通过相应的方法来完成，然而在 MATLAB 中创建的类对象应该如何设置属性呢？参见例 7-3。

例 7-3　设置 Java 对象的属性。

接例 7-2，在 MATLAB 命令行中键入下面的指令：

```
>> %设置 fa 的属性
>> setTitle(fa,'New A')
>> %获取 fa 的属性
>> title = getTitle(fa)
 title =
 New A
 >> whos
  Name          Size                    Bytes  Class
  fa            1x1                            java.awt.Frame
  fb            1x1                            java.awt.Frame
  title         1x1                            java.lang.String
Grand total is 3 elements using 0 bytes
>> %设置 fb 的属性
>> fb.setTitle('New B')
```

```
>> %获取 fa 的属性
>> title = getTitle(fb)
 title =
 New B
```

在例 7-3 中，使用两种不同的方法设置了不同对象的同一个属性，首先使用了 MATLAB 语句格式设置了 fa 的属性，然后利用 MATLAB 的语法结构获取了修改之后的属性值。在设置 fb 对象的属性时，使用了 Java 的语法结构，即调用类的 public 方法时使用“.”运算符。两种语法结构完成的功能是一样的。

从例 7-3 可以看出，Java 与 MATLAB 之间结合得非常紧密，在进行 Java 与 M 语言混合编程时，程序员完全可以根据自己的喜好选择不同的语法结构——M 语言的语法结构或者 Java 语言的语法结构。不过这里推荐大家在开发程序时，至少在一个函数文件中的语法结构是统一的，这样便于阅读和修改程序。

修改 Java 对象的属性还可以像操作句柄图形对象一样使用 get 或者 set 函数，例如在 MATLAB 命令行中键入下面的指令：

```
>> get(fb)
        Owner = []
        Foreground = none
        IconImage = []
        PreferredSize = [ (1 by 1) java.awt.Dimension array]
        MinimumSize = [ (1 by 1) java.awt.Dimension array]
        AlignmentY = [0.5]
        Toolkit = [ (1 by 1) sun.awt.windows.WToolkit array]
        AlignmentX = [0.5]
        MenuBar = []
        State = [0]
    ... ...
```

在 MATLAB 命令行窗口中将显示长长的对象属性列表，注意在列表中有若干有关回调函数的属性设置，利用这些对象属性，就可以在 M 语言中通过 Java 对象完成用户界面应用程序的开发了。

set 指令可以设置对象的属性，例如：

```
>> set(fb,'Title','Frame B')
>> fb.getTitle
 ans =
 Frame B
```

在 M 语言中创建的 Java 对象可以通过 save 命令保存到 MAT 数据文件中。例如利用下面的指令可以将例 7-2 和 7-3 创建的所有 Java 对象保存到数据文件中：

```
>> save javaobj fa fb title
>> what
MAT-files in the current directory D:\Temp
```

javaobj

同样，保存到数据文件中的 Java 对象也能够通过 load 命令加载到当前的工作空间中：

```
>> clear all
>> whos
>> load javaobj title
>> whos
  Name          Size                    Bytes  Class
  title         1x1                            java.lang.String
Grand total is 1 element using 0 bytes
```

不过在保存和加载 Java 对象时需要注意如下几点：

- 创建 Java 对象的 Java 类必须实现了 Serializable 接口。关于 Serializable 接口请参阅 Java 语言的文档。
- 保存在数据文件的 Java 对象在加载之前，用于创建对象的 Java 类没有被修改。
- 保存的 Java 对象没有 transient 字段。关于 transient 关键字和 transient 字段请参阅 Java 语言的文档。

7.3.3 应用 Java 对象

在 M 语言中使用 Java 对象能够为 MATLAB 带来更加丰富的功能，但是 M 语言和 Java 语言是两种截然不同的语言，所以在 M 语言中处理 Java 对象和数据就需要格外注意。在本小节将仔细讲解应用 Java 对象、操作 Java 数据的方法和注意事项。

熟悉 Java 语言的读者应该知道，在 Java 语言中所有 Java 的对象都是引用类型的变量，而且 Java 的方法也都是用引用来传递参数，而不是传递数值的，也就是说 Java 的方法内部可以修改传递到方法中的参数数值，这一点和 MATLAB 的函数不同。参见下面的例 7-4。

例 7-4　引用 Java 对象。

使用例 7-2 创建的 Java 对象，在 MATLAB 命令中键入下面的指令：

```
>> %是简单的赋值吗?
>> fc = fb;
>> %修改 fc 的属性
>> fc.setTitle('New C')
>> %fb 对象的属性也被修改了
>> fb.getTitle
ans =
 New C
```

由于 Java 对象使用引用，所以当执行了 fc = fb 的操作之后，fc 对象和 fb 对象都指向同一个对象(内存地址)，所以当修改 fc 对象的属性时，fb 的属性值就会跟着修改，所以得到了例 7-4 的结果——fc 和 fb 的对象属性同时被修改。

细心的读者可能已经发现，在 MATLAB 中，有些执行 Java 对象的方法其返回值的类型是相应的 Java 类。Java 的对象在 MATLAB 中属于一类特殊的数据类型，几乎所有的

MATLAB 数据都可以向 Java 数据对象转变，但是 Java 的对象一般不能直接参与 MATLAB 的数据运算，需要进行相应的类型转换才可以，参见例 7-5。

例 7-5　Java 的数据。

在 MATLAB 命令行中键入下面的指令：

```
>> %创建一个双精度类型的 Java 对象
>> jD = java.lang.Double(33)
jD =
33.0
>> %这种运算可以么？
>> jD+3
??? Error using ==> +
Function '+' is not defined for values of class 'java.lang.Double'.
>> %  当然不可以，因为 jD 是一种特殊的数据类型
>> whos
  Name        Size                    Bytes  Class
  jD          1x1                            java.lang.Double
Grand total is 1 element using 0 bytes
>> %强制类型转换
>> double(jD)
ans =
     33
>> whos
  Name        Size                    Bytes  Class
  ans         1x1                         8  double array
  jD          1x1                            java.lang.Double
Grand total is 2 elements using 8 bytes
```

一般地，Java 对象的运算在 MATLAB 中是无效的，需要进行特殊的数据类型转换，就像例 7-5 中 jD+3 这种运算是无法直接进行的。

然而在 java.lang.Double 类中包含方法 floatValue，该方法的返回值就是 double(Java 类型)，如果在 MATLAB 中执行该方法，则返回的数值会自动转变为 MATLAB 的双精度类型数据：

```
>> class(jD.floatValue)
ans =
double
>> class(jD)
ans =
java.lang.Double
```

可以看出对于那些返回值是普通数据类型的方法，这些方法返回值类型是什么，则返

回值就直接转换成相应的 MATLAB 类型，如表 7-1 所示。

表 7-1 Java 数据类型向 MATLAB 数据类型的转换

数据类型(Java)	MATLAB 数据类型(标量)	MATLAB 数据类型(数组)
Boolean	logical	logical
byte	double	int8
short	double	int16
int	double	int32
long	double	double
float	double	single
double	double	double
char	char	Char

在使用 Java 方法之前如何确认该方法返回的是什么类型的数据呢？可以使用 MATLAB 的 methods 或者 methodsview 函数，这两个函数的作用都是将 Java 的类所包含的方法显示出来，不同的是前者将 Java 类的方法显示在 MATLAB 命令行中，而后者是将方法显示在一个图形窗口中。例如在 MATLAB 中分别键入下面的指令：

```
>> methods java.lang.Double -full
Methods for class java.lang.Double:
Double(double)
Double(java.lang.String) throws java.lang.NumberFormatException
static java.lang.String toString(double)
static java.lang.Double valueOf(java.lang.String) throws java.lang.NumberFormatException
static boolean isNaN(double)
static boolean isInfinite(double)
static double parseDouble(java.lang.String) throws java.lang.NumberFormatException
static long doubleToLongBits(double)
static long doubleToRawLongBits(double)
static double longBitsToDouble(long)
void wait() throws java.lang.InterruptedException    % Inherited from java.lang.Object
void wait(long,int) throws java.lang.InterruptedException    % Inherited from java.lang.Object
void wait(long) throws java.lang.InterruptedException    % Inherited from java.lang.Object
java.lang.Class getClass()    % Inherited from java.lang.Object
void notify()    % Inherited from java.lang.Object
void notifyAll()    % Inherited from java.lang.Object
int hashCode()
int compareTo(java.lang.Double)
int compareTo(java.lang.Object)
boolean equals(java.lang.Object)
```

java.lang.String toString()

boolean isNaN()

float floatValue()

boolean isInfinite()

byte byteValue()

short shortValue()

int intValue()

long longValue()

double doubleValue()

如果在使用 methods 函数时没有指定-full 参数，则仅显示函数的名称。

如果使用图形化的界面，则显示如图 7-2 所示的对话框界面，该对话框中包含 Java 类中定义的所有方法的详细细节。

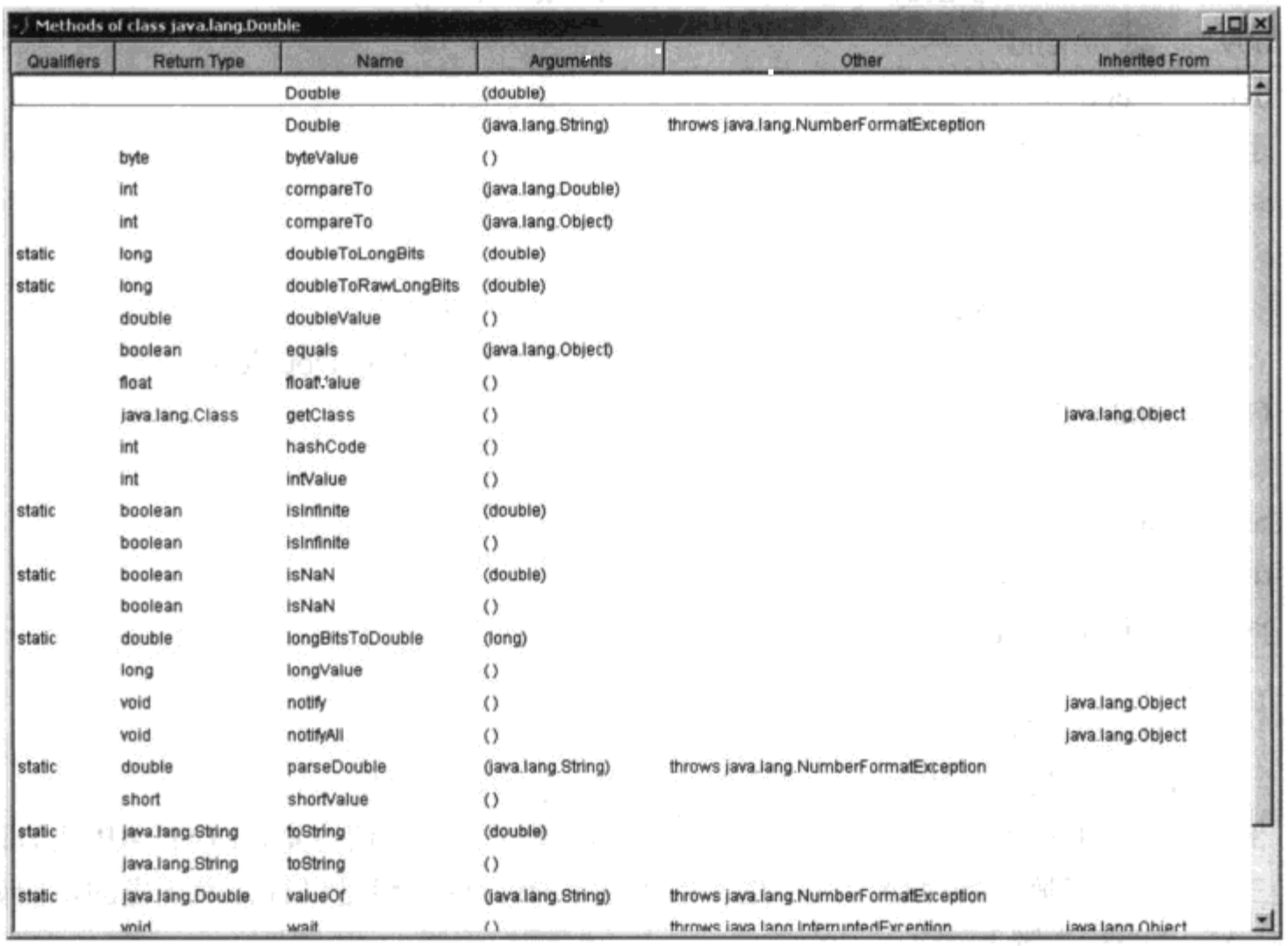


图 7-2　显示完整的方法列表

>> methodsview java.lang.Double

通过 methodsview 对话框就可以查看相应方法的输入、输出参数类型了。

7.3.4　Java 数组

Java 的数组组织比较灵活，一般地来说，Java 数组是一维的，如果具有两个下标，则可认为是数组的数组；如果具有三个下标，则数组的元素是数组的数组。同时，二维的 Java 数组不一定是完整的矩阵，在数组中的每一行或者每一列的元素个数都可以不尽相同。

如果需要在 MATLAB 中创建 Java 的数据，则需要使用 java_array 函数，它的定义如下：

javaArray('package_name.class_name',x1,...,xn)

其中 x1～xn 等为数组每一维的尺寸。

例 7-6　创建 Java 数组。

参见下面的脚本代码：

```
001     % 创建 Java 数组
002     % Java 数组通常是一维向量
003     % 2-D Java 数组通常是 1-D Java 数组的数组
004     % 3-D Java 数组通常也是 1-D 数组，其元素是数组的数组
005     dArray = java_array('java.lang.Double',3,4);
006     for i = 1:3
007         for j = 1:4
008             dArray(i,j) =   java.lang.Double(i*10+j);
009         end
010     end
```

运行例子 7-6 的代码：

```
>> jarray
>> whos
  Name          Size                    Bytes  Class
  dArray        3x1                             java.lang.Double[][]
Grand total is 5 elements using 16 bytes

>> dArray
dArray =
java.lang.Double[][]:
    [11]    [12]    [13]    [14]
    [21]    [22]    [23]    [24]
    [31]    [32]    [33]    [34]
```

例 7-6 的脚本代码执行之后得到了二维的 Java 数组，不过请注意，MATLAB 认为该数组是三行一列的矩阵。和 MATLAB 的矩阵不同，Java 的数组是以行元素优先的，例如在 MATLAB 命令行中键入：

```
>> dArray(2)
 ans =
 java.lang.Double[]:
    [21]
    [22]
    [23]
    [24]
```

从运行结果可以看出，Java 数组在 MATLAB 中是行向量，可以直接向数组中不存在的

元素直接赋值，例如：

```
>> dArray(2,5) = java.lang.Double(100)
 dArray =
 java.lang.Double[][]:
     [4 element array]
     [5 element array]
     [4 element array]

>> dArray(4,1) = java.lang.Double(100)
 dArray =
 java.lang.Double[][]:
     [4 element array]
     [5 element array]
     [4 element array]
     [1 element array]
```

可见 Java 数组不是标准的矩阵，请读者在使用 Java 数组进行编程时注意这一特点。

7.4　应 用 示 例

前面几个小节分别介绍了在 MATLAB 中创建 Java 对象，执行对象的方法等内容。在本小节将通过若干示例演示在 MATLAB 的应用程序中使用 Java 对象的过程，同时也进一步介绍在 M 语言文件中创建、应用 Java 对象的方法。

例 7-7　读取 URL 信息。

在 M 语言中应用 Java 类的主要目的之一就是充分利用 Java 语言的网络功能，例如读取网络上的信息等。本例子就是利用 Java 的 java.util.URL 类，以及相应的 Java 语言数据 I/O 流的功能从互联网上读取数据。下面是该例子的源代码：

```
001    function readURL(website)
002    %READURL MATLAB 中调用 Java 类示例
003    % 读取 website 指定的网页数据，如果没有输入参数
004    % 读取默认参数设定的主页
005    if nargin == 0
006        website = 'http://www.hirain.com';
007    end
008    % 创建 URL 对象
009    url = java.net.URL(website);
010    % 创建输入流
011    is = openStream(url);
012    isr = java.io.InputStreamReader(is);
013    br = java.io.BufferedReader(isr);
```

```
014     % 读取网页的数据
015     for i = 0:44
016         s = readLine(br);
017     end
018     readLine(br)
019     readLine(br)
020     readLine(br)
```

例 7-7 的代码中基本上没有 M 语言的成分，只有 015 行的 for 循环使用了 M 语言的语法，其余的都是 Java 语言的语法结构。首行利用 009 行的代码创建了 Java 的 URL 类对象，然后利用 Java 语言强大的 I/O 流的功能，从互联网的网页上读取了信息。这里将互联网的网页看作纯文本的文件读取，例如 015~020 行的代码，每一句代码都一次读取文本中的一行。运行例 7-7 的代码：

```
>> readURL
 ans =
                    <table width="565" border="1" cellspacing="0" cellpadding="2"
bordercolor="#000000" bordercolordark="#ffffff" bgcolor="#0099cc">
 ans =
                      <tr valign="bottom">
 ans =
                        <td height="23" width="80" align="middle" bgcolor="#cc6600"><A
class=menu href="index03-03-14.htm" >首页</A></td>
```

注意，这里读取回来的信息是网页中的超文本代码。读者在执行本例子代码时，请确认已经能够正常的浏览网页，也可以用别的超链接测试本例子代码。

例 7-8　读取 zip 文件信息。

zip 文件是一种常用的数据压缩文件格式，Java 语言提供了相应的 I/O 流可以从 zip 文件中直接读取数据。本例子就是通过该功能读取 zip 文件的数据信息，并且将它显示在 MATLAB 的图形窗口中。下面是该例子的源代码：

```
001     function readzipdir
002     %READZIPDIR Java 和 MATLAB 混合编程示例
003     % readzipdir 读取 zip 文件内部包含数据信息
004
005     % 导入 Java 类包
006     import java.io.*
007     import java.util.*;
008     import java.util.zip.*;
009     % 通过通用文件对话框选取 ip 文件
010     [filename,pathname] = uigetfile('*.zip','选择 Zip 文件');
011     file = fullfile(pathname,filename);
012     count = 0;
```

```
013     % 创建数据流
014     zin = ZipInputStream(FileInputStream(file));
015     % 创建 zip 数据项
016     entry = zin.getNextEntry;
017     % 依次读取数据信息
018     try
019         while(~isempty(char(entry.getName)))
020             count = count + 1;
021             String{count} = char(entry.getName);
022             zin.closeEntry;
023             entry = zin.getNextEntry;
024         end
025     end
026     % 关闭数据流
027     zin.close;
028     % 创建图形界面——句柄图形
029     fig = figure('Position',[232,258,320,240]);
030     list = uicontrol(fig,'Style','ListBox');
031     set(list,'Position',[20,20,280,200],...
032             'String',String);
```

可以将程序简单分为两个部分：从第 001 行代码开始到第 027 行是第一部分，这部分代码利用了 MATLAB 的数据文件选择对话框(uigetdfile)以获取 zip 文件，然后利用 Java 的 I/O 流读取 zip 文件中包含的数据信息，读取数据信息时，将读取的信息写入 MATLAB 元胞数组中(代码的 021 行)；代码的 028 行到最后是程序的第二部分，这部分代码将前面获取的元胞数组中的内容显示在 MATLAB 的图形窗体中，这里使用了句柄图形的编程方式，对于熟悉 MATLAB 的读者都知道，使用句柄图形开发简单的图形界面是比较容易的。这里创建了一个包含列表框控件的图形窗体，运行该例子的代码：

```
>> readzipdir
```

该程序首先弹出一个对话框，要求用户选择一个 zip 文件，如图 7-3 所示。

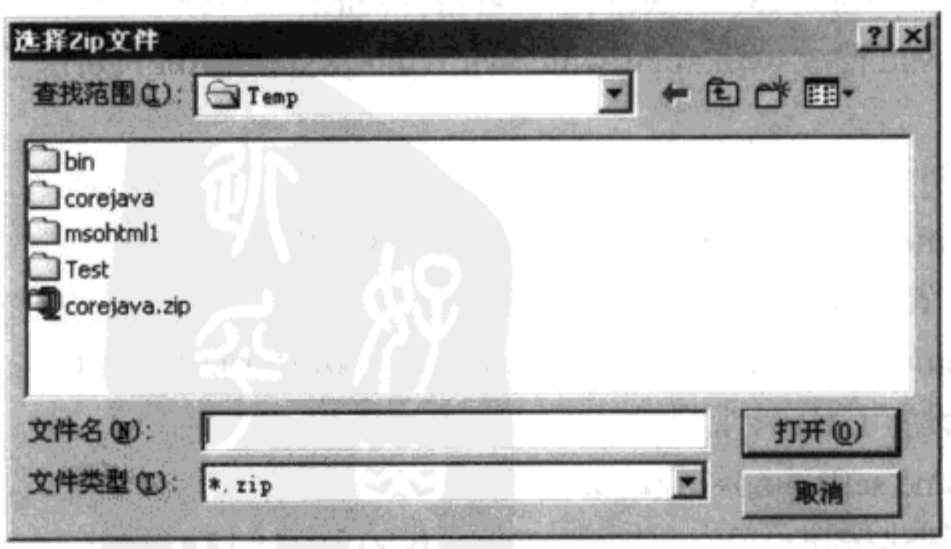


图 7-3　选择文件

本书使用的 zip 文件是《Java 核心编程》一书提供的实例 zip 文件——corejava.zip，选择该文件并且单击打开，之后 MATLAB 将在图形窗体中显示该文件内部的数据信息，如图 7-4 所示。

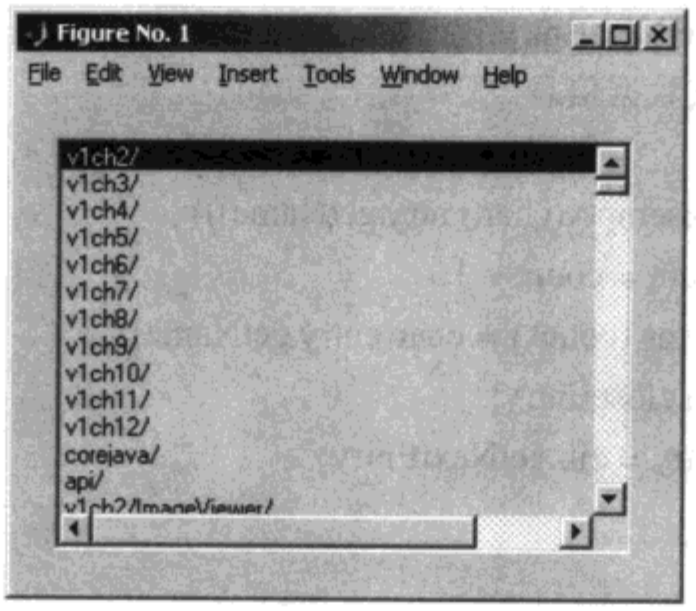


图 7-4 显示 zip 文件内部信息的图形窗体

MATLAB 图形窗体中包含的信息就是 zip 文件内部的内容，有兴趣的读者可以进一步修改例 7-8 的代码，可以利用 Java 的 zip 文件输入流或者输出流读写 zip 文件中的内容。

例 7-9 使用 Java 对象创建界面。

Java 类库中相当一部分内容是用来创建用户图形界面应用程序的，分别有 java.awt 包和 javax.swing 包，将这两种类包提供的功能组合起来就可以完成各种图形用户界面程序的开发。同样，这些开发包开发出来的用户图形界面也可以用在 MATLAB 的环境下，和 M 语言混合编程，实现图形界面，特别是 MATLAB 本身的控件数量类型有限，可以使用 Java 来实现特殊界面功能要求。本小节演示了一个简单界面，复杂的界面编程将在后面的例子中讲述。下面是本例子的源代码：

```
001    function jsimplegui(varargin)
002    %JSIMPLEGUI 演示在 MATLAB 中使用 Java 类创建 GUI 的过程
003    %    MATLAB 结合 Javax. swing 组件完成图形用户界面程序开发
004
005    % 设置窗体的外观
006    javax.swing.UIManager.setLookAndFeel(...
007    com.sun.java.swing.plaf.windows.WindowsLookAndFeel);
008
009    % 创建 Java frame 类对象
010    fFrame = javax.swing.JFrame('简单的 GUI');
011    %设置 fFrame 对象的外观属性
012    fFrame.getContentPane.setLayout(java.awt.BorderLayout);
013    % 设置 GUI 外观尺寸
014    fFrame.setBounds(100,100,400,250);
015    % 创建按扭
```

```
016     DisplayButton = javax.swing.JButton('显示信息');
017     % 设置按扭的回调函数
018     set(DisplayButton,'ActionPerformedCallback','disp("喂！ 你好吗?")');
019     % 再次获取相应的属性
020     get(DisplayButton,'ActionPerformedCallback')
021     % 添加一个新的按扭
022     fFrame.getContentPane.add(DisplayButton,java.awt.BorderLayout.SOUTH);
023     % 创建表格
024     fTable = javax.swing.JTable(num2cell(rand(15,2)),{'日期','价格'});
025     % 创建滚动条窗口
026     fTableScrollPane = javax.swing.JScrollPane(fTable);
027     % 添加表格
028     fFrame.getContentPane.add(fTableScrollPane,java.awt.BorderLayout.CENTER);
029     % 显示窗体
030     fFrame.show;
```

熟悉 Java 语言 SWING 类包和 AWT 类包的读者读上面的代码应该是没有什么障碍的。不过需要说明一点，MATLAB 的图形用户界面无论用何种手段来实现，相应用户输入的动作都只能够在回调函数中完成，也就是说，不能够用 Java 语言程序员熟悉的事件监听机制来编写 Java 程序界面的用户响应回调函数。所以在例 7-9 的 018 行中，简单的设置回调函数可以是一段字符串的指令。执行例 7-9 的代码：

```
>> jsimplegui
```

将显示如图 7-5 所示的对话框。

图 7-5　M 语言编写的 Java 对话框

并且在 MATLAB 的命令行窗体中显示如下信息：

```
ans =
disp('喂！ 你好吗?')
```

注意，在对话框中的表格控件被 MATLAB 的数据填充了起来，如果单击几次“显示信息”按钮，则在 MATLAB 的命令行窗口中将显示如下内容：

```
喂！ 你好吗?
```

喂！你好吗?
喂！你好吗?
喂！你好吗?
喂！你好吗?
喂！你好吗?
……

注意，例 7-9 运行得到的对话框不是 MATLAB 的，而是 Java 的，所以任何绘图的功能都无法添加到该对话框中。如果需要替换对话框上的 Java 图标，可以使用相应的 JFrame 类包含的方法，代码如下：

```
imic = javax.swing.ImageIcon(fullfile(matlabroot,'toolbox/matlab/icons/matlabicon.gif'));
im = imic.getImage;
frame.setIconImage(im);
```

将这几行代码添加到适当的位置上，则创建的用户界面标题栏处将显示 MATLAB 的商标。

例 7-10　复杂界面编程。

其实复杂的界面也是由若干简单的控件和界面组合起来的，无非为了响应用户的输入需要设置很多回调函数的参数而已。总的来说用 Java 语言直接开发界面还是比较困难的，就好像用 C++语言直接开发 Windows 用户界面程序而不使用 MFC 一样，这就需要程序开发人员针对任务需要选择合适的编程方法，本例子的代码如下：

```
001    function payment_calc
002    %payment_calc 使用 Swing and MATLAB 计算数据示例
003    % 注意：本例子需要使用 financial toolbox
004
005    %导入类包
006    import javax.swing.*;
007    import java.awt.*;
008    % 创建窗体
009    frame = JFrame('还贷计算器');
010    frame.setSize( 300,200)
011    %设置窗体位置
012    defXSize = 300;
013    defYSize = 200;
014    % 获取屏幕尺寸
015    ss = get(0,'ScreenSize');
016    % 计算位置
017    xloc = ss(3) / 2 - defXSize / 2;
018    yloc = ss(4) / 2 - defYSize / 2;
019    % 创建界面上其它的控件
020    LabelLoan     = JLabel('贷款数额');
```

```
021     LabelYears   = JLabel('贷款年限');
022     LabelRate    = JLabel('              年利率');
023     TextLoan      = JTextField(9);
024     TextYears    = JTextField(3);
025     BoxRate      = JComboBox({' ','9.25','9.0','8.75',...
026                 '8.5','8.25','8.0','7.75',...
027                 '7.5','7.25','7.0','6.75',...
028                 '6.5','6.25'});
029     ButtonCalc   = JButton('计算');
030     ButtonClear = JButton('清除');
031     ButtonQuit = JButton('退出');
032     LabelPayment = JLabel('每月需月供');
033     TextPayment   = JTextField(8);
034     LabelIntPaid = JLabel('总计利息总额');
035     TextIntPaid   = JTextField(9);
036     % 创建外观
037     MainPanel = JPanel( GridLayout(1,2) );
038     InPanel     = JPanel( BoxLayout.Y_AXIS );
039     OutPanel    = JPanel( BoxLayout.Y_AXIS );
040     InPanel.add( LabelLoan );InPanel.add( TextLoan );
041     InPanel.add( LabelYears );InPanel.add( TextYears );
042     InPanel.add( LabelRate );InPanel.add( BoxRate );
043     InPanel.add( ButtonCalc );InPanel.add( ButtonClear );
044     OutPanel.add( LabelPayment );OutPanel.add( TextPayment );
045     OutPanel.add( LabelIntPaid );OutPanel.add( TextIntPaid );
046     OutPanel.add( ButtonQuit );
047     MainPanel.add( InPanel );
048     MainPanel.add( OutPanel );
049     frame.getContentPane.add( MainPanel );
050     %设置对话框位置并显示
051     frame.setLocation(Point(xloc,yloc));
052     frame.show;
053     % 以下设置必要的数据和回调函数
054     set( InPanel, 'UserData', [0 0 0]);
055     set( frame,'WindowClosingCallback', ...
056          {@Callback, InPanel},...
057          'Name','quit');
058     set( BoxRate, 'ActionPerformedCallback',...
059          {@Callback, InPanel}, ...
```

```
060         'Name','rate');
061     set( ButtonCalc, 'ActionPerformedCallback',...
062         {@Callback, InPanel}, ...
063         'Name','calc');
064     set( ButtonClear, 'ActionPerformedCallback',...
065         {@Callback, InPanel}, ...
066         'Name','clear');
067     set( ButtonQuit, 'ActionPerformedCallback',...
068         {@Callback, InPanel}, ...
069         'Name','quit');
070     set( TextLoan, 'ActionPerformedCallback',{@Callback, InPanel}, ...
071         'FocusLostCallback',{@Callback, InPanel},...
072         'HorizontalAlignment',[4], ... %right justified
073         'Name','loan');
074     set( TextYears, 'ActionPerformedCallback',{@Callback, InPanel}, ...
075         'FocusLostCallback',{@Callback, InPanel},...
076         'HorizontalAlignment',[4], ... %right justified
077         'Name','year');
078     set( TextPayment, 'HorizontalAlignment',[4], … %right justified
079         'Editable','off',...
080         'Background',[1 1 1]);
081     set( TextIntPaid, 'HorizontalAlignment',[4], … %right justified
082         'Editable','off',...
083         'Background',[1 1 1]);
084
085     while ~(exist('STOP'))
086         waitfor(InPanel,'ApplicationData')
087         result = getappdata(InPanel,'QuestionAppData');
088         if ~isnumeric(result)
089             if (strcmp(result,'calc'))          %向界面控件写数据
090             UData = get( ButtonCalc, 'UserData');
091             set( TextPayment , 'Text', cur2str( UData(1) ));
092             set( TextIntPaid , 'Text', cur2str( UData(2) ));
093             elseif (strcmp(result,'clear')) %  清除界面数据
094             set( TextPayment , 'Text', '');
095             set( TextIntPaid , 'Text', '');
096             set( TextYears, 'Text', '' );
097             set( TextLoan, 'Text', '' );
098             set( BoxRate, 'SelectedItem',' ' );
```

```
099                 else                                    % 结束应用程序
100                     STOP = 1;
101                 end
102             end
103             rmappdata(InPanel,'QuestionAppData'); % 清除应用程序数据
104         end
105         % 关闭窗口
106         frame.dispose;
107
108         %回调函数
109         function Callback( eventSrc, eventData, panel )
110         UData = get( panel, 'UserData');
111         name = get(eventSrc,'Name');
112         switch name
113         case 'rate'       % get rate selected and save in panel
114             Srate = get( eventSrc, 'SelectedItem' );
115             result = str2num(Srate);
116             UData = [UData(1) UData(2) result];
117         case 'year'       % get number of years and save in panel
118             Syear = get( eventSrc, 'Text' );
119             if (strcmp(Syear,''))
120                 result = 1;
121             else
122                 result = str2num(Syear);
123             end
124             UData = [ UData(1) result UData(3)];
125         case 'loan'       % get amount of loan and save in panel
126             Sloan = get( eventSrc, 'Text' );
127             if (strcmp(Sloan,''))
128                 result = 0;
129             else
130                 result = str2num(Sloan);
131             end
132             UData = [ result UData(2) UData(3)];
133         case 'calc'       % calculate payments and interest
134             [PRINP,INTP,BAL,P] = amortize(UData(3)/1200,UData(2)*12,UData(1));
135             set( eventSrc, 'UserData', [ P sum(INTP) ]);
136             result = 'calc';
137         case 'clear'      % send command to clear text fields
```

```
138             result = 'clear';
139         case 'quit'      % send command to stop waiting and close application
140             result = 'quit';
141         end
142         set( panel, 'UserData', UData );
143         setappdata( panel, 'QuestionAppData', result );
```

例 7-10 的代码很长，主要是这里面使用了较多的 Java 对象，并且依次设置这些不同控件的回调函数。M 语言的回调函数和 Java 里面的事件监听机制有一定的对应关系，比如 Java 的 ActionListener 接口中的 actionPerformed 方法，对应在 M 语言中就是 ActionPerformedCallback。

提示：

有关 Java 语言事件监听和事件响应等话题，感兴趣的读者请参阅 Java 语言的书籍。

如果要设置控件的回调函数可以通过直接设置需要响应的回调函数属性值来实现，例如在例子 7-10 中，很多控件需要响应 ActionPerformed 事件，于是分别设置相应的属性值就可以了，设置回调函数属性的时候，使用函数的句柄和监听事件的容器作为参数，请参阅代码的 055～069 行。

如果在 M 语言中实现回调函数则需要遵循下面的语法规则：

```
function CallbackFunction(eventSrc,eventData,Component)
```

其中，CallbackFunction 为回调函数的名称，可以任意定义，也可以定义在不同的 M 文件中；eventSrc 定义了发生或者接收事件的目标控件，例如单击按钮时，这里就是按钮的 Java 对象；eventData 为保留的参数，目前还不使用，也许未来的版本将使用该参数传递数据；Component 一般为容纳事件响应控件或者事件发出控件的容器，比如 Panel 或者 Frame。

在回调函数中编写需要响应用户动作的代码，就像例 7-10 的代码一样。

在不同控件之间或者回调函数与主函数之间传递数据可以使用 MATLAB 提供的 setappdata 函数和 getappdata 函数将数据和控件绑定在一起，就像例 7-10 中将计算的数据和控制程序运行的数据与 Panel 控件绑定在一起一样，见例子的 087 行、110 行和 143 行。

运行例 7-10 的代码，在弹出的对话框中设置必要的数据，例如贷款数额设置为 100 000，贷款的年限设置为 10，年利率为 9.25，单击“计算”按钮，则会计算出相应的数额，如图 7-6 所示。不过这里直接使用了 MATLAB 的财经工具箱中的函数，所有计算的单位都是美元，因此会显示美元符号。

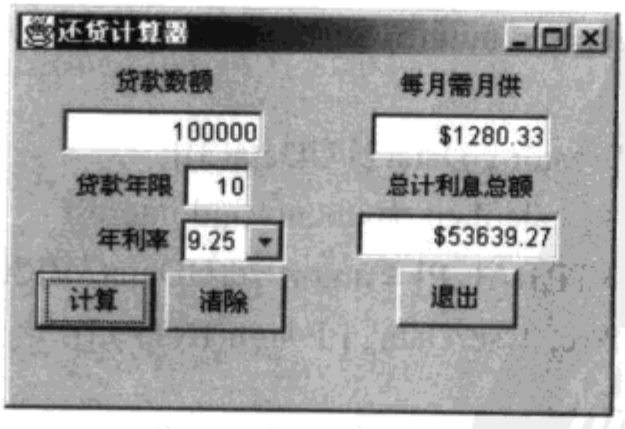


图 7-6 计算器的界面

7.5　本 章 小 结

本章详细讨论了在 M 语言中集成 Java 语言特性的必要性和方法。MATLAB 与 Java 是密不可分的两种开发工具，其中 MATLAB 软件中很多功能都是利用 Java 开发的。而且，MATLAB 和 Java 在各自的领域内都有无可比拟的优势，集成两种开发工具的优势，将两种工具综合起来使用以解决复杂的问题就是本章需要讨论的内容。在本章，首先简要回顾了 Java 语言的特性，了解必要的 Java 语言知识是学习本章的基础。然后讨论了在 M 语言中使用 Java 对象的方法，特别针对 Java 的数据类型与 MATLAB 的数据类型相互转换的话题进行了讨论。最后，通过若干简单的示例，演示了在 M 语言中使用 Java 类的方法，并且简要介绍了在 M 语言中实现事件监听的方法。

通过本章的学习，读者应该能够基本了解 Java 与 MATLAB 之间的关系，能够将一些简单的 Java 功能集成到 M 语言中，从而以小见大，逐步完成复杂的功能。Java 语言在互联网、数据库等企业级应用方面有强大的优势，结合 MATLAB 提供的科学计算能力，强强联合必然能够解决更多的复杂问题。

最后，笔者向希望学习 Java 语言的读者推荐两本学习 Java 的好书：

- 《Java 2 核心技术》I、II 两卷，由机械工业出版社出版，它是初学者的入门教材。
- 《Java 编程思想》Think in Java，它经典地介绍了面向对象和 Java 语言的实现，如果读者英文过关的话，则可以直接阅读该书的英文版，其中文版已经能够在市面上购买到。

附录 A MATLAB 产品支持的编译器

表 A-1 总结了 MATLAB R13 不同产品需要的 C 语言或者 Fortran 语言编译器，其它版本的 MATLAB 支持的编译器请参阅相应版本的 MATLAB 帮助文档。

表 A-1 Windows 平台上 MATLAB R13 对编译器的需求

Microsoft Windows		MATLAB	MATLAB Compiler	MATLAB COM Builder	MATLAB Excel Builder	Simulink	Stateflow	Real-Time Workshop	Real-Time Windows Target	xPC Target
编译器	版本									
Borland C++ Builder	6	*	*	*	*	*		*		
	5	*	*	*	*	*	*	*		
	4	*	*	*	*	*	*	*		
	3	*	*	*	*	*	*	*		
Borland C/C++ Compiler	5.5[1]	*	*			*	*	*		
	5.2[2]	*	*			*	*	*		
	5	*	*			*	*	*		
Compaq Visual Fortran	6.6	*				*				
	6.1	*				*				
Digital Visual Fortran	6.0	*				*				
	5.0	*				*				
Lcc-win32	2.4.1	*	*			*	*	*	*	
Microsoft Visual C/C++[4]	.NET[3]	*	*	*	*	*	*	*	*	*
	6	*	*	*	*	*	*	*	*	*
	5	*	*	*	*	*	*	*	*	*
Watcom C/C++	11	*				*	*	*	*	*
	10.6	*				*	*	*	*	*

(1) Borland C/C++ Compiler 5.5 是 Borland 提供的免费命令行(Free Command Line) C/C++编译器。

(2) Borland C/C++ Compiler 5.2 在 mex –setup 或者 mbuild –setup 时显示为 Borland C/C++ Compiler 5.02。

(3) Microsoft Visual C/C++ .NET 在 mex –setup 或者 mbuild –setup 时显示为 Microsoft Visual C/C++ version 7.0。

(4) 目前须选择通过 Mathworks 测试验证的专业版(Professional)Visual C/C++，如果使用其它版本的 Visual C/C++可能会出现代码错误，所以推荐使用专业版的 Visual C/C++。

UNIX 平台上的 MATLAB 对 C 语言编译器的要求比较复杂，主要原因是不同的 Unix 平台具有不同的 C/C++编译器，总的来说，MATLAB 支持所有系统下的内建(native)ANSI C 编译器。此外

■ MATLAB 不支持 HPUX 系统内建的 C 编译器，请选用 GNU C 3.0 或者以上版本的编译器。

■ 在所有平台下使用 GNU C 编译器可以创建 MEX 文件、MATLAB 计算引擎应用和 MAT 应用程序。

■ 在使用 MATLAB Compiler 时不能使用 GNU C 编译器，但在 Linux 和 MAC 平台下例外。

■ 对于 MATLAB R13，在 Solaris 系统下可以使用 Workshop Compiler 6.0。

■ 仅在 Linux 平台上可以使用 GNU C++编译器。

UNIX 平台上 MATLAB R13 对 C++编译器的支持见表 A-2。

表 A-2　UNIX 平台上 MATLAB R13 对 C++编译器的支持

系统	编译器
Dec/Compaq Alpha	Compaq C++ V6.2-024 for Digital UNIX V4.0F (Rev. 1229)
HP700	HP aC++ B3910B A.01.27
HPUX	HP aC++ B3910B A.03.30
IBM RS6000	VisualAge C++ Compiler 5.0.0.0 COMMITTED
Linux	gcc version 2.95.2 19991024
MAC	Apple Computer, Inc. version gcc-932.1, based on gcc version 2.95.2 19991024
SGI	MIPSpro Compilers:Version 7.3.1.2m
Solaris	WorkShop Compilers 5.0 98/12/15 C++ 5.0
UNIX 平台	MATLAB R13 对 Fortran 编译器的支持见表 A-3。

UNIX 平台上 MATLAB R13 对 Fortran 编译器的支持见表 A-3。

表 A-3　UNIX 平台上 MATLAB R13 对 Fortran 编译器的支持

系统	Fortran 77	Fortran 90
Compaq Digital	Compaq Fortran 77 V5.3-189-449BB	Compaq Fortran 90 V5.3-189-449BB
HP-UX		HP F90 v2.4.10
HP 700		HP F90 v2.5.1
IBM_RS	7.1.0.0 Committed I XL Fortran Compiler	7.1.0.0 Committed I XL Fortran Compiler
Linux	g77 version 2.95.2 19991024	
MAC	Absof Fortran 7.0	Absoft 7.0
SGI	MIPSpro Compilers: Version 7.3.1.2m	MIPSpro Compilers: Version 7.3.1.2m
Solaris	WorkShop Compilers 5.0 99/09/16 Fortran 77 5.0 patch 107596-0	WorkShop Compilers 5.0 99/09/16 Fortran 90 2.0

附录 B　加载和应用动态链接库函数

所有的操作系统都预先提供了一部分函数用于程序的开发，操作系统将这些函数组织起来编译成为共享库，在 Windows 平台上共享库以动态链接库的形式提供，文件的扩展名为.dll。一般地，共享库(动态链接库)由操作系统的开发商或者其它软件的开发商提供，可以被其它的运行在该操作系统上的应用软件调用以实现相应的功能。在 MATLAB 环境下访问这些共享库函数是扩充 MATLAB 功能的强大手段。

在 MATLAB 6.5 (R13)中，已经具备了直接在命令行加载动态链接库(.dll)、执行动态链接库内部函数的能力，这种特性只能够在 Windows 98/NT/2000/XP 系统下实现，并且需要下载并安装相应的补丁才可以，下面是该补丁程序的下载超链接地址：

ftp://ftp.mathworks.com/pub/tech-support/solutions/s33513/GenericDll_1p1.exe

用户可以将该补丁下载之后，解压缩出必要的安装文件。安装时请首先关闭 MATLAB，然后就可以直接将文件包中的文件解压缩安装到用户的 MATLAB 安装路径下，例如 D:\MATLAB6p5。安装完毕后重新启动 MATLAB，就可以在 MATLAB 中正常使用加载的动态链接库的功能了。如果用户使用的是 MATLAB 6.5.1(R13SP1)则不需要安装该补丁，因为加载动态链接库的能力已经集成在 MATLAB 中了。

使用动态链接库外部接口函数能够完成以下功能：

- 加载动态链接库到 MATLAB 空间；
- 执行动态链接库的函数，使 MATLAB 获得操作系统底层的能力。

B.1 基本使用方法

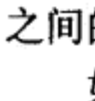
本小节简要介绍加载和应用动态链接库函数的基本使用方法。MATLAB 加载和使用动态链接库函数仅通过七八个函数就可以完成，其实只要掌握了这些函数的使用方法，就掌握了加载和应用动态链接库的基本方法。注意，此部分外部接口函数的应用不需要具有特殊的工具箱，但是需要使用 MATLAB 以及 MATLAB 自带的标准 C 语言编译器 Lcc。

B.1.1 加载/卸载动态链接库

和在 MATLAB 中使用 Java 类类似，在使用动态链接库函数之前，必须正确加载动态链接库到 MATLAB 的工作空间中，此项工作需要使用外部接口函数 loadlibrary。例如在 MATLAB 的命令行窗口下键入以下指令：

```
>> hfile = [matlabroot '\extern\include\mex.h']
hfile =
E:\MATLAB6p5p1\extern\include\mex.h
>> loadlibrary('libmex', hfile)
```

上面的两条指令将 MATLAB 提供的动态链接库 libmex.dll 加载到了 MATLAB 的工作空间。loadlibrary 函数的输入参数有两个，第一个输入参数是库文件的名称，第二个输入参数是说明该库文件的 C 语言头文件，C 语言的头文件向 MATLAB 提供了函数的基本说明信息。

 **注意：**

在执行上述指令加载库文件时可能会在 MATLAB 命令行中接受到部分警告信息，目前可以忽略这些警告信息，动态链接库已经成功的加载了。

可以使用 libisloaded 函数来判断库文件是否被成功加载，例如接着上面两行指令继续键入：

```
>> libisloaded('libmex')
ans =
     1
>> libisloaded('libmx')
ans =
     0
```

如果动态链接库已经成功加载，则 libisloaded 函数返回值为逻辑真 1，否则为逻辑假 0。

当动态链接库使用完毕后，最好将加载的动态连接库卸载，卸载动态链接库需要使用函数 unloadlibrary，例如：

```
>> unloadlibrary('libmex')
>> libisloaded('libmex')
ans =
     0
```

随时将不需要使用的动态链接库卸载是非常好的编程习惯。

B.1.2　浏览并执行库函数

加载动态链接库的目的是执行库函数丰富的 MATLAB 功能，MATLAB 提供了浏览并执行库函数的方法。不过，在调用动态链接库函数时需要留意 MATLAB 数据与 C 语言数据之间的交换问题，特别是当调用函数需要传递参数的时候。

如果需要浏览动态链接库包含的函数说明，可以使用 libfunctions 或者 libfunctionsview 函数。例如在 MATLAB 命令行中键入下面的指令：

```
>> loadlibrary('shrlibsample.dll','shrlibsample.h')
```

这里加载的库函数是由 MATLAB 提供的加载动态链接库应用的示例，正确安装了补丁程序后，此文件可以在%MATLABROOT%\extern\examples\shrlib 路径下找到。

使用 libfuncitions 命令可以查看函数信息：

```
>> libfunctions('shrlibsample')
Functions in library shrlibsample:
addDoubleRef          multDoubleArray
addMixedTypes         multDoubleRef
addStructByRef        multiplyShort
```

```
addStructFields     readEnum
allocateStruct      stringToUpper
deallocateStruct
ans =
     []
```

上面使用的 libfunctions 指令输出了动态链接库包含的函数列表，可以在调用函数的时候使用-full 命令行参数，输出详细的函数说明。

```
>> libfunctions('shrlibsample','-full')
Functions in library shrlibsample:
doublePtr multDoubleArray(doublePtr, int32)
double addMixedTypes(int16, int32, double)
[double, doublePtr] addDoubleRef(double, doublePtr, double)
[string, string] stringToUpper(string)
string readEnum(Enum1)
double addStructFields(c_struct)
[lib.pointer, doublePtr] multDoubleRef(doublePtr)
[double, c_structPtr] addStructByRef(c_structPtr)
c_structPtrPtr allocateStruct(c_structPtrPtr)
voidPtr deallocateStruct(voidPtr)
int16Ptr multiplyShort(int16Ptr, int32)
ans =
Too many input arguments.
```

这里将动态链接库函数的详细说明罗列了出来，可以使用 libfunctionsview 函数在图形窗体中显示上面的函数清单。

```
>> libfunctionsview('shrlibsample')
```

这时将显示如图 B-1 所示的对话框，即显示库函数的清单。

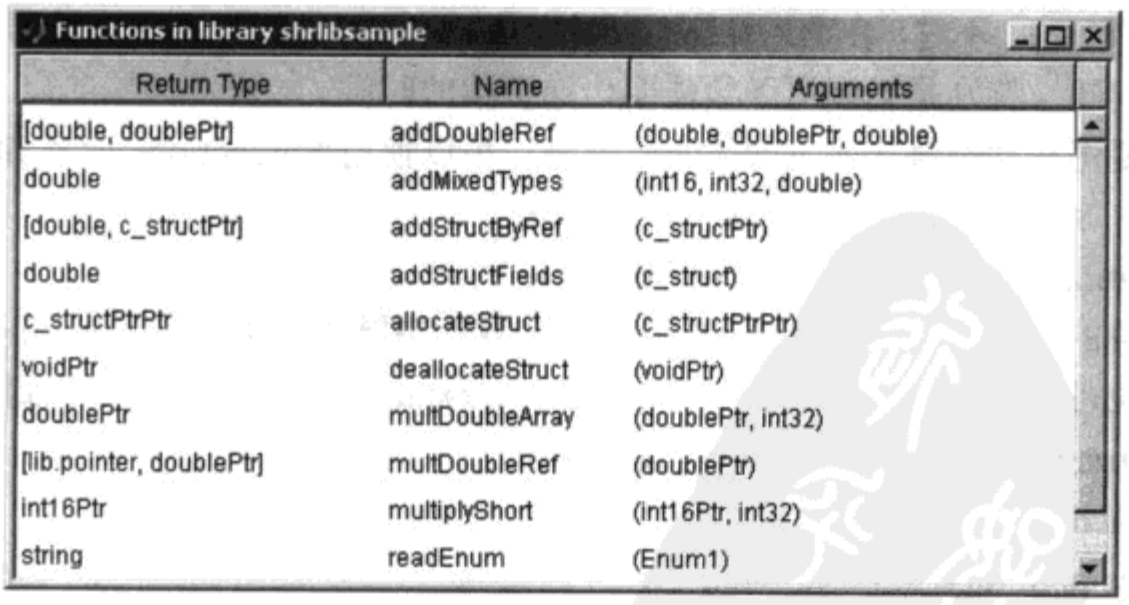


图 B-1　显示库函数的详细列表

在函数的列表中显示了函数的输入、输出参数，一般地，MATLAB 能够非常自然地将 MATLAB 的数据同 C 语言的数据进行转换。执行库函数时需要使用 callib 函数，例如执行

shrlibsample 中的 addMixedTypes 函数的指令如下：

```
>> y = calllib('shrlibsample','addMixedTypes',1,2,3)
y =
     6
```

addMixedTypes 函数的输入参数分别为 16 位整数、32 位整数和双精度数，而调用该函数时，可以直接使用 MATLAB 的数据，两者之间的数据转换由 MATLAB 自动完成。

calllib 函数是用来调用动态链接库的函数，它的输入参数一般为已经加载的动态链接库、需要调用的函数以及函数的输入参数，返回值就是函数运算的返回数值。

关于数据类型转换方面的内容请参阅 B.2 小节的介绍。

B.2　数据类型转换

一般情况下调用动态链接库时需要进行的数据类型转换都可以由 MATLAB 自动完成，不过还是需要强调一下 C 语言数据类型和 MATLAB 的数据类型之间的转换问题，两者之间的数据类型转换见表 B-1 和表 B-2。

表 B-1　一般数据类型之间的转换

C 语言数据类型	MATLAB 的数据类型
char，byte	int8
unsigned char，byte	uint8
short	int16
unsigned short	uint16
int ，long	int32
unsigned int，unsigned long	uint32
float	single
double	double
char *	1xN 的字符串数组

表 B-2　特殊数据类型转换

C 语言数据类型	MATLAB 的数据类型
整数类型指针 int *	(u)int(size)Ptr
单精度浮点类型指针 float *	singlePtr
双精度浮点类型指针 double *	doublePtr
mxArray *	MATLAB 的数组
void *	voidPtr
指针的指针 double **	相应的指针类型后面再加上 Ptr doublePtrPtr

熟悉 MATLAB 和 C 语言的读者对表 B-1 理解起来不会有任何困难，MATLAB 和 C 语言之间的数据转换往往自动完成，例如在 shrlibsample 里面的部分函数，这些函数都使用了

普通的数据类型，转换可以自动完成。

```
>> calllib('shrlibsample', 'addMixedTypes', 127, 33000, pi)
ans =
   3.3130e+004
>> str = 'This was a Mixed Case string';
>> calllib('shrlibsample', 'stringToUpper', str)
ans =
    THIS WAS A MIXED CASE STRING
```

不过表 B-2 总结的数据类型转换可能让读者感到难以理解，这里的数据类型转换是专门为转换那些引用类型变量而准备的。

在动态链接库外部接口函数中，MATLAB 提供了 libpointer 函数来创建引用类型的指针，例如：

```
>> x = pi;
>> xp = libpointer('doublePtr',x)
xp =
     lib.pointer
```

此时 xp 就是双精度类型的指针。

```
>> get(xp)
          Value: 3.1416
       DataType: 'doublePtr'
```

可以进行库函数中关于双精度类型数据的运算：

```
>> calllib('shrlibsample', 'multDoubleRef', xp);
>> get(xp,'Value')
ans =
    15.7080
>> whos
  Name        Size                    Bytes  Class
  ans         1x1                         8  double array
  x           1x1                         8  double array
  xp          1x1                            lib.pointer
Grand total is 3 elements using 16 bytes
```

关于其它常用引用类型数据的创建与应用这里就不一一列举了，请读者根据上述说明自行判断使用。

C 语言中还有一种特殊的数据类型——枚举类型，在 MATLAB 中，如果调用的函数使用了枚举类型参数，则可以通过枚举的名称或者序号完成枚举类型的转换，例如有如下枚举定义：

```
enum Enum1 {en1 = 1, en2, en4 = 4} TEnum1;
char* readEnum(TEnum1 val) {
    switch (val) {
```

```
        case 1 :return "You chose en1";
        case 2: return "You chose en2";
        case 4: return "You chose en4";
        default : return "enum not defined";
        }
    }
```

则可以进行如下操作：

```
>> calllib('shrlibsample', 'readEnum', 'en4')
ans =
You chose en4
```

同样也可以进行如下操作：

```
>> calllib('shrlibsample', 'readEnum', 4)
ans =
You chose en4
```

关于结构类型数据的转换请参阅 MATLAB 的帮助文档。

B.3　应用函数说明

■ calllib

调用通用动态链接库内部包含的函数。

语法：

```
[x1, ..., xN] = calllib('libname', 'funcname', arg1, ..., argN)
```

说明：

通过该函数将直接调用已经加载的动态链接库中的函数，其中 libname 为已经加载的动态链接库，funcname 为被执行的函数名，arg1～argN 是函数的输入参数，x1～xN 为函数计算的输出。

如果在加载动态链接库时使用了假名(alias)，则调用库函数的时候也需要使用库文件的假名，参见 loadlibrary 的说明。

■ libfunctions

显示动态链接库文件中包含的函数信息。

语法：

```
m = libfunctions('libname')
m = libfunctions('libname', '-full')
libfunctions libname –full
```

说明：

m = libfunctions('libname')显示库文件 libname 中函数的信息，其中 libname 是利用 loadlibrary 函数加载的动态链接库文件。

如果使用-full 命令行参数，例如 m = libfunctions('libname', '-full')或者 libfunctions libname

–full，则在 MATLAB 命令行窗口中显示函数的完整说明。

如果在加载动态链接库时使用了假名(alias)，则调用库函数的时候也需要使用库文件的假名，参见 loadlibrary 的说明。

■ libfunctionsview

通过图形化窗体显示动态链接库文件中的函数信息。

语法：

libfunctionsview('libname')

libfunctionsview libname

说明：

libfunctionsview libname 将在图形化窗体中显示库文件中定义的函数信息，其中 libname 是利用 loadlibrary 函数加载的动态链接库。

如果在加载动态链接库时使用了假名(alias)，则调用库函数的时候也需要使用库文件的假名，参见 loadlibrary 的说明。

MATLAB 将创建一个图形化的窗体显示函数的详细信息，该窗体中将显示如下信息：

- 函数的名称。
- 函数的返回值类型。
- 函数的输入参数。

■ libisloaded

判断指定的函数库是否被加载。

语法：

libisloaded('libname')

libisloaded libname

说明：

如果动态链接库文件 libname 已经加载，则 libisloaded('libname')函数返回逻辑真，否则返回逻辑假。

如果在加载动态链接库时使用了假名(alias)，则调用库函数的时候也需要使用库文件的假名，参见 loadlibrary 的说明。

■ libpointer

为应用函数创建对象指针。

语法：

p = libpointer

p = libpointer('type')

p = libpointer('type', value)

说明：

p = libpointer 返回空指针。

p = libpointer('type') 返回空指针，其类型为参数 type 指定的类型，type 可以是有效的数

值类型、结构或者枚举，这些类型应该在 loadlibrary 函数加载的库文件中被有效的定义。

p = libpointer('type', value) 返回 type 指定类型的指针，并且用初值 value 赋值。

■ libstruct

按照动态链接库中的定义构造结构。

语法：

s = libstruct('structtype')

s = libstruct('structtype', mlstruct)

说明：

s = libstruct('structtype') 按照加载的动态链接库中所定义的结构类型 structtype 创建 MATLAB 结构，结构的字段都赋初值为 0。

s = libstruct('structtype', mlstruct)按照加载的动态链接库中所定义的结构类型 structtype 创建 MATLAB 结构，结构的字段使用 mlstruct 结构的数值赋初值。

注意，这里使用的结构类型定义是在动态链接库中的定义，例如：

```
>> loadlibrary('shrlibsample.dll','shrlibsample.h')
>> s = libstruct('c_struct')
s =
    lib.c_struct
>> get(s)
    p1: 0
    p2: 0
    p3: 0
>> s.p1 = 123;    s.p2 = 456;    s.p3 = 789;
>> calllib('shrlibsample', 'addStructFields', s)
ans =
          1368
```

■ loadlibrary

加载动态链接库到 MATLAB 工作空间中。

语法：

loadlibrary('shrlib', 'hfile')

loadlibrary('shrlib', @protofile)

loadlibrary('shrlib', ..., 'options')

loadlibrary shrlib hfile options

说明：

loadlibrary('shrlib', 'hfile') 按照 shrlib 和 hfile 的说明加载动态接库到 MATLAB 工作空间中。

loadlibrary('shrlib', @protofile) 使用 M 文件——protofile 代替 C 语言的头文件，库函数的原型 M 文件详见下面关于原型文件的说明。

loadlibrary('shrlib', ..., 'options')在加载动态链接库的时候可以使用一些特殊的选项，这些选项可以分别设置如下：

addheader *hfileN* 附加头文件

当动态链接库内部包含的文件的声明不仅仅在一个头文件时，可以使用 addheader 选项将附加的头文件声明进来。

alias *name* 设置假名

当加载的头文件名称较长不容易记忆时，可以为加载的库文件声明一个代名 name，这样在后面使用库文件时直接使用这个代名就可以了。

includepath *path* 设置额外的 include 路径

一般地，可以指定头文件的 include 路径，避免每次都给出完整的路径名称。

mfilename *mfile*

生成 m 文件，以后可以使用该 M 文件替代 C 语言的头文件来加载库函数，例如

```
>> loadlibrary('shrlibsample.dll','shrlibsample.h','mfilename','msample')
>> what
M-files in the current directory D:\TEMP
msample
```

请读者自己编辑察看该 M 文件中的内容。

loadlibrary shrlib hfile options 加载动态链接库文件的命令行格式。

■ unloadlibrary

卸载动态链接库文件。

语法：

```
unloadlibrary('libname')
unloadlibrary libname
```

说明：

将指定的动态链接库从内存中卸载。注意，使用 clear 指令不能将动态链接库清除，必须使用此函数。

附录C　北京九州恒润科技有限公司简介

北京恒润科技有限公司成立于1998年4月，是一家民营高新技术企业。公司主要业务涉及工程系统设计仿真产品的代理销售、工程仿真分析项目咨询等。现有员工53人，其中专家3人，博士5人，硕士15人。

公司成立以来，承揽了众多的工程咨询项目，为用户提供了全方位的技术解决方案和技术支持，具备了很强的软件开发和技术咨询实力。目前公司项目咨询领域主要有导航、制导与控制、射频仿真和汽车电子。

公司注重建设具有特色的公司文化，培养员工认同共同的价值观，确保员工能伴随着公司的发展而发展，从而形成了激励上进型的工作氛围，建立了一支高素质的、有凝聚力的队伍。与此同时，公司管理层不断借鉴成功公司的组织体系和管理经验，经过消化、试行、改进、完善的过程，构建了公司特有的组织与管理体系、技术研发与服务体系、市场开拓与支持体系，目前公司已开始进入一个高速发展期。

公司在上海、成都两地设立了办事处，并于2003年3月被授予为北京市首批“纳税信用A级企业”。

公司的文化和价值取向：诚信，以人为本，追求卓越。

公司的发展方向：以为用户提供集成工程环境，提供技术解决方案为依托，进行自主产品的研制开发。

公司当前的产品信息：

公司随时追踪国内外技术发展的最新动态，把先进技术和一流产品介绍给国内用户，代理了若干处于国际领先的仿真分析和设计的工具产品，支持用户进行工程系统设计和仿真分析，这些产品包括：

美国MathWorks公司的MATLAB产品(独家代理)；

德国dSPACE公司实时半实物仿真系统dSAPCE(独家代理)；

德国Vector公司CAN网络系统的分析工具Vector(独家代理)；

英国Radioscape公司的通信仿真软件RadioScape(独家代理)；

美国Signalogic公司的DSP设计的辅助软件Signalogic(独家代理)；

德国TESIS公司的TESIS虚拟车辆实时仿真专家软件(独家代理)；

美国PSS公司的PSS卫星控制系统设计分析和仿真软件(独家代理)；

美国TAC公司的NEVADA热辐射分析软件(独家代理)；

美国Network Analysis公司的SINDA/G热分析(独家代理)；

美国Altia公司的Altia嵌入系统图形界面开发平台(独家代理)；

英国Ricardio公司的车辆发动机内燃机设计分析系统Ricardio(独家代理)。

……

详细产品信息请登录北京九州恒润科技有限公司网站 www.hirain.com。

培训服务

作为 Mathworks 公司中国大陆地区的独家代理，北京九州恒润科技有限公司拥有完善、权威的 MATLAB 培训服务，并且拥有丰富的培训经验和一流的培训讲师。欢迎使用 MATLAB 从事系统分析、仿真、设计的工程人员，大学教师和学生参加由北京九州恒润科技有限公司提供的 MATLAB 培训课程。请登录公司网站——www.hirain.com 查阅详细的培训信息。

基础培训：

- MATLAB 基础和编程入门。
- Simulink 建立动态系统模型。
- Real-Time Workshop 基础。
- Stateflow 建模技术基础。

中级培训：

- MATLAB 外部接口编程。
- MATLAB 高级编程技巧。
- MATLAB 图形用户界面。
- Simulink 高级建模技术。
- Stateflow 高级建模技术。

高级培训：

- 根据客户特定需要定制。

联系方式

► 北京九州恒润科技有限公司
www.hirain.com
北京公司总部
北京市西城区北三环中路 27 号商房大厦 430 室
电话：010-82011456
► 上海办事处
上海市徐汇区漕宝路 70 号光大会展中心 D 座 505 室
电话：021-64325416
► 成都办事处
成都市人民南路一段 86 号城市之心大厦 23 楼 N 座
电话：028-86203381/2/3

附录 D　部分习题提示与参考答案

第 2 章

习题 3

编写程序片段完成下面的功能：

创建无符号 16 位整数类型矩阵，矩阵的内容如下：

$$\begin{bmatrix} 1 & 0 & 3 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 2 & 0 & 1 & 0 & 2 \end{bmatrix}$$

通过 mxIsSparse、mxIsUint16 函数来判断矩阵，将判断的结果创建成逻辑类型向量。

解答：

参见下面的代码片段：

```
001    uint16_T data[]
002        = { 1,0,2,0,1,0,3,0,1,0,1,0,1,0,2 };
003    uint16_T   *pr;
004    char*   info;
005    mxLogical result[2];
006    /* 创建矩阵-> 3×5 矩阵*/
007    plhs[0] = mxCreateNumericMatrix( 3, 5, mxUINT16_CLASS, mxREAL );
008    pr = mxGetData( plhs[0] ); /*获取实际数据的指针 */
009    /*复制数据 */
010    memcpy( pr, data, 15*sizeof(uint16_T) );
011    /* 判断 */
012    result[0] = mxIsDouble(plhs[0]);
013    result[1] = mxIsUint16(plhs[0]);
014    /* 创建逻辑向量 */
015    plhs[1] = mxCreateLogicalMatrix(1,2);
016    pr = mxGetData(plhs[1]);/*获取实际数据的指针 */
017    /* 复制数据 */
018    memcpy(pr,result,2);
```

第 3 章

习题 3

编写程序完成下列功能：

编写函数，根据输入的变量输出不同的信息，需要输出的信息包括：

- 类型名称。
- ClassID。
- 行数。
- 列数。

该函数还需要完成下列功能：

- 返回操作是否成功的信息。
- 能够处理字符串数据和数值数组。

编译该函数时使用命令行编译和集成开发环境编译。

提示：

需要使用 mxGetClassName、mxGetClassID、mxGetM、mxGetN、mexErrMsgTxt 等函数。

解答：

```
001     /* 函数 Description 获取输入参数的类型
002        并输出在命令行窗口中 */
003
004     # include "mex.h"
005     void mexFunction(int nlhs, mxArray *plhs[],
006                              int nrhs, const mxArray *prhs[])
007     {
008          const char *ClassName;
009          int flag;
010          if (nlhs != 0 || nrhs != 1)
011          {
012               flag = 1;
013               mexErrMsgTxt("错误：输入输出参数不正确！ ");
014          }
015          else
016               flag = 0;
017
018          if (flag==0)
```

```
019         {
020             ClassName = mxGetClassName(prhs[0]);
021             if (strcmp(ClassName,"cell")==0 || strcmp(ClassName,"struct")==0)
022                 mexWarnMsgTxt("该函数无法处理元胞数组数据或者结构数据");
023                 /*mexErrMsgTxt("该函数无法处理元胞数组数据或者结构数据");*/
024
025             mexPrintf("类型 : %s\n", mxGetClassName(prhs[0]));
026             mexPrintf("Class ID : %d\n", mxGetClassID(prhs[0]));
027             mexPrintf("行数 : %d\n", mxGetM(prhs[0]));
028             mexPrintf("列数 : %d\n", mxGetN(prhs[0]));
029             plhs[0]=mxCreateString("数据类型的描述已经成功输出！ ");
030
031         }
032     }
```

习题 4

编写程序完成下列功能：

编写函数，根据输入的参数输出不同的信息，要求在 Windows 信息对话框中输出类型的名称，如图 3-18 所示。

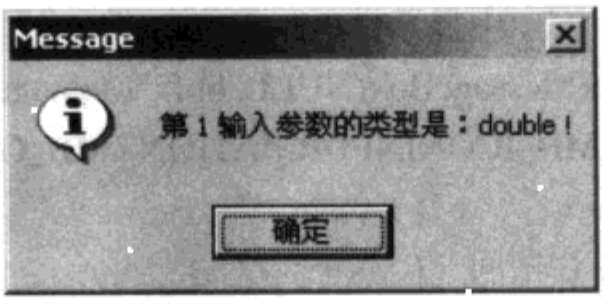


图 3-18　参数的信息

提示：

使用 Windows 函数 MessageBox 函数输出参数的信息，该函数的定义如下：

```
int MessageBox(HWND hWnd,
    LPCTSTR lpText,
    LPCTSTR lpCaption,
    UINT uType
);
```

具体的解释请参阅 Windows Platform SDK 的说明。

解答：

```
001     /* createinfo_ex2.c
002      * 第 3 章练习 4 的答案
003      * 编译时，使用 User32.lib
004      * mex createinfo_ex2.c */
```

```
005     #include "mex.h"
006     #include "windows.h"
007     /* 入口函数 */
008     void mexFunction( int nlhs, mxArray *plhs[],
009                       int nrhs, const mxArray *prhs[] )
010     {
011         /* 变量声明 */
012         char* buf;
013         int i;
014         /*判断输入参数 */
015         if(nrhs == 0){
016             mexErrMsgTxt("至少输入一个参数");
017         }
018         /* 分配内存空间 */
019         buf = mxMalloc(1024);
020         for( i = 0 ; i < nrhs ; i++){
021             /* 获取参数信息 */
022             sprintf(buf,"第 %d 输入参数的类型是：%s !",
023                         i+1,mxGetClassName(prhs[i]));
024             /* 在窗口中显示 */
025             while(MessageBox(NULL, buf, "Message",
026                 MB_ICONINFORMATION | MB_OK)!= IDOK){};
027         }
028         /* 释放内存空间 */
029         mxFree(buf);
030     }
```

注意：由于本练习中使用了 Windows 32 位 API，所以在编译该程序时，需要使用下面的命令行：

mex createinfo_ex4.c user32.lib

其中，user32.lib 是定义 Windows API 的函数库。关于该函数库的说明请参阅 Windows Platform SDK 文档说明。

第 4 章

习题 2

使用 Fortran 语言 MEX 文件，重新编写例 3-10 设置图像属性的代码，要求能够在 MEX 文件中直接创建曲线，并且设置曲线的属性，可以在集成开发环境中完成程序的编译过程。

解答：

```
001     C   modifyfig_ex2.f
002     C   定义曲线的样式
003     C   第 4 章练习 2 的示例
004     C   入口函数
005         subroutine mexFunction(nlhs, plhs, nrhs, prhs)
006     C-------------------------------------------------------------
007     C   参数声明
008         integer plhs(*), prhs(*)
009         integer nlhs, nrhs
010         integer flag;
011     C-------------------------------------------------------------
012     C   代码行
013     C   检测输入输出参数
014         if (nlhs .gt. 0 .OR. nrhs .ne. 1) then
015             call mexErrMsgTxt('错误：输入输出参数错误！')
016         endif
017     C   绘制曲线
018         call mexEvalString('x = 0:0.1:10;y = sin(x);')
019         call mexEvalString('h_line = plot(x,y,"r*")')
020     C   暂停
021         call mexEvalString('pause')
022     C   新的属性值
023         flag = mexPutVariable('base','newmarker',prhs(1))
024         if(flag .ne. 0) then
025             call mexErrMsgTxt('错误：设置变量错误！')
026         endif
027     C   设置新的属性
028         flag = mexEvalString('set(h_line,"marker",newmarker)')
029         if (flag .eq. 0) then
030             plhs(1) = mxCreateString('****  图形线条样式设置操作成功  ****')
031         else
032             call mexErrMsgTxt('错误：设置属性错误！')
033         endif
034     C   函数文件尾部
035         return
036         end
```

习题 3

使用 Fortran 语言 MEX 文件编写第 2 章练习 3 的代码，创建整数类型的矩阵即可。

解答：

```
001     C   creatematrix_ex3.f
002     C   创建整数类型的矩阵
003     C   第 4 章练习 3 的示例
004     C   入口函数
005         subroutine mexFunction(nlhs, plhs, nrhs, prhs)
006     C-------------------------------------------------------------
007     C   参数声明
008         integer plhs(*), prhs(*)
009         integer nlhs, nrhs
010         integer realdata(15)
011         data realdata / 1,0,2,0,1,0,3,0,1,0,1,0,1,0,2 /
012         integer flag , pr
013         integer classid
014     C-------------------------------------------------------------
015         integer mxClassIDFromClassName,mxCreateNumericMatrix
016         integer mxCopyInteger4ToPtr,mxGetData
017     C   代码行
018     C   创建数据类型 ID
019         classid = mxClassIDFromClassName('int32')
020         if(classid .eq. 0) then
021             call mexErrMsgTxt('无法创建指定的数据类型 ID')
022         endif
023     C   创建输出参数
024         plhs(1) = mxCreateNumericMatrix(3,5,classid,0)
025     C   获取数据的指针
026         pr = mxGetData(plhs(1))
027     C   拷贝数据
028         call mxCopyInteger4ToPtr(realdata,pr,15)
029     C   函数文件尾部
030         return
031         end
```

习题 4

编写程序完成下列功能：

调用 MATLAB 指令，根据输入的参数计算矩阵的特征值和特征向量(如果输入的参数

是双精度类型矩阵，则利用 MATLAB 的矩阵运算指令)。

解答：

```
001      C    callfunc_ex4.f
002      C    调用 MATLAB 函数
003      C    第 4 章练习 4 的示例
004      C    入口函数
006           subroutine mexFunction(nlhs, plhs, nrhs, prhs)
007      C-----------------------------------------------------------------
008      C    参数声明
009           integer plhs(*), prhs(*)
010           integer nlhs, nrhs
011           integer classidout,classidin
012           integer V , D
013           integer m , n
014           integer pr
015      C-----------------------------------------------------------------
016           integer mxGetM,mxGetN,mxGetClassID
017           integer mxClassIDFromClassName
018           integer mxCreateDoubleMatrix,mxDuplicateArray
019      C    代码行
020      C    判断输入参数
021           if (nrhs .ne. 1 .OR. nlhs .ne. 2) then
022                call mexErrMsgTxt('错误：输入参数错误!')
023           endif
024      C    判断输入参数是否为方阵
025           m = mxGetM(prhs(1))
026           n = mxGetN(prhs(1))
027           if( m .ne. n) then
028                call mexErrMsgTxt('错误：输入参数必须为方阵!')
029           endif
030      C    获取输入参数的类型
031           classidin = mxGetClassID(prhs(1))
032           classidout = mxClassIDFromClassName('double')
033           if(classidin .eq. classidout) then
034      C         调用 MATLAB 命令
035                call mexPutVariable('base','matrixin',prhs(1))
036                call mexEvalString('[V,D] = eig(matrixin)')
037      C         获取变量
038                V = mexGetVariable('base','V')
```

```
039             m = mxGetM(V)
040             n = mxGetN(V)
041     C       创建输出变量
042             plhs(1) = mxCreateDoubleMatrix(m,n,0)
043             plhs(1) = mxDuplicateArray(V)
044     C       获取变量
045             D = mexGetVariable('base','D')
046             m = mxGetM(D)
047             n = mxGetN(D)
048     C       创建输出变量
049             plhs(2) = mxCreateDoubleMatrix(m,n,0)
050             plhs(2) = mxDuplicateArray(D)
051     C       清除工作空间
052             call mexEvalString('clear V; clear D;clear matrixin;')
053         else
054             call mexErrMsgTxt('输入参数必须为双精度类型!')
055         endif
056     C   函数文件尾部
057         return
058         end
```

第 5 章

习题 3

读写数据。

有一纯文本格式的数据文件，该数据文件中包含的数据如下：

```
Broncos  14    2    0.8750  y
Falcons  14    2    0.8750  y
Lions    5     11   0.3125  n
Patriots 15    1    0.9375  y
Vikings  9     7    0.5625  y
```

编写程序实现下面的功能：将该纯文本格式的数据文件数据利用 C 语言或者 Fortran 语言的源程序读入；利用 MAT 的数据文件应用能力将这些数据写入一个 MAT 文件中，要求每一列数据保存在一个变量中，一共有五个变量，这些变量保存在一个数据文件中。

解答：

```
001     /*
002      * readfile_ex3.c
003      * 读取数据，写入 MAT 文件
```

```
004      * 第 5 章练习 3 的示例
005      */
006     /* 必要的头文件 */
007     #include "mex.h"
008     #include "mat.h"
009     /* MEX 函数文件入口 */
010     void mexFunction(int nlhs,mxArray *plhs[],
011                              int nrhs,const mxArray *prhs[]){
012         char* matfile = "season.mat";
013         int flag;
014         MATFile *mfp;
015         mxArray *team,*w,*l,*wp,*playoff;
016         int ndims[] = {5,1};
017         /* 执行 MATLAB 指令 */
018         flag = mexEvalString(
019         "[team,w,l,wp,playoff]=textread('season.txt','%s %d %d %f %c');");
020         if(flag != 0){
021             mexErrMsgTxt("错误：无法正确执行 MATLAB 指令");
022         }
023         /* 创建变量 */
024         team = mxCreateCellMatrix(5,1);
025         w = mxCreateDoubleMatrix(5,1,mxREAL);
026         l = mxCreateDoubleMatrix(5,1,mxREAL);
027         wp = mxCreateDoubleMatrix(5,1,mxREAL);
028         playoff = mxCreateCharArray(2,ndims);
029         /* 读取变量 */
030         team = mexGetVariable("base","team");
031         w = mexGetVariable("base","w");
032         l = mexGetVariable("base","l");
033         wp = mexGetVariable("base","wp");
034         playoff = mexGetVariable("base","playoff");
035         /* 打开数据文件 */
036         mfp = matOpen(matfile,"w");
037         if(mfp == NULL){
038             mexErrMsgTxt("错误，无法打开 MAT 数据文件!");
039         }
041         /* 写入变量 */
042         matPutVariable(mfp,"TEAM",team);
043         matPutVariable(mfp,"W",w);
```

```
044             matPutVariable(mfp,"L",l);
045             matPutVariable(mfp,"WP",wp);
046             matPutVariable(mfp,"PLAYOFF",playoff);
047             /* 输出信息 */
048             plhs[0] = mxCreateString("创建 MAT 文件成功!");
049             mexEvalString("clear;");
050             /* 清除内存空间 */
051             mxDestroyArray(team);
052             mxDestroyArray(w);
053             mxDestroyArray(l);
054             mxDestroyArray(wp);
055             mxDestroyArray(playoff);
056             /* 关闭数据文件 */
057             flag = matClose(mfp);
058             if(flag != 0){
059                 mexErrMsgTxt("无法关闭 MAT 文件!");
060             }
061     }
```

习题 4

读写数据。

例 5-2 生成的 MEX 文件执行的结果是生成包含三个变量的数据文件，其中一个变量为字符串类型的变量，它的内容是“MATLAB is Wonderful!”。请利用 C 语言或者 Fortran 编写程序实现下面的功能：将字符串从数据文件中读入，并且将该字符串内容完整地颠倒过来，也就是生成新的字符串“!lufrednoW si BALTAM”，并将新的字符串写入到 MAT 数据文件中。

解答：

```
001     /*
002      * revord.c
003      * 从数据文件中读入字符串，并将字符串翻转后写入数据文件
004      * 第 5 章练习 4 的示例
005      */
006     #include "stdio.h"
007     #include "mat.h"
008     /* 翻转字符串的函数 */
009     void revord(char *input_buf, int buflen, char *output_buf)
010     {
011         int     i;
012         /* 翻转字符串 */
```

```
013         for (i = 0; i < buflen-1; i++)
014             output_buf[i] = input_buf[buflen - i - 2];
015         output_buf[buflen-1]= '\0';
016     }
017     /* 主函数 */
018     void main(){
019         char inbuf[256],outbuf[256];
020         MATFile *mfp;
021         mxArray *string;
022         int flag , buflen;
023         /* 打开数据文件 */
024         printf("请输入文件名： ");
025         scanf("%s",inbuf);
026         printf("\n 打开数据文件： %s\n", inbuf);
027         mfp = matOpen( inbuf , "u");
028         if(mfp == NULL){
029             fprintf(stderr,"\n 无法打开指定的数据文件!\n");
030             return;
031         }
032         /* 读取数据 */
033         string = matGetVariable(mfp,"String");
034         if(string == NULL){
035             fprintf(stderr,"\n 无法读取指定的变量!\n");
036         }else{
037             /* 获取数据内容 */
038             buflen = mxGetM(string) * mxGetN(string) + 1;
039             flag = mxGetString(string,inbuf,buflen);
040             if(flag != 0){
041                 fprintf(stderr,"\n 无法获取变量数据!\n");
042             }else{
043                 printf("\n 成功读入数据： %s\n",inbuf);
044                 /* 翻转字符串 */
045                 mxDestroyArray(string);
046                 revord(inbuf,buflen,outbuf);
047                 /* 创建输出的数据 */
048                 string = mxCreateString(outbuf);
049                 if(string == NULL){
050                     fprintf(stderr,"\n 无法创建输出变量!\n");
051                 }else{
```

```
052                     /* 写入数据到数据文件 */
053                     flag = matPutVariable(mfp,"revod",string);
054                     if(flag != 0)
055                         fprintf(stderr,"\n 无法写入输出变量!\n");
056                     else{
057                         /* 释放占用的内存 */
058                         printf("\n 成功写入数据： %s\n",outbuf);
059                         mxDestroyArray(string);
060                     }
061                 }
062             }
063         }
064         /* 关闭数据文件 */
065         flag = matClose(mfp);
066         if(flag != 0){
067             fprintf(stderr,"\n 无法关闭指定的数据文件!\n");
068             return;
069         }else{
070             printf("\n 关闭数据文件!\n");
071         }
072     }
```

第 6 章

习题 1

求解系统表达式。

编写 C 语言或者 Fortran 语言计算引擎应用程序求解系统表达式，原始数据都保存在数据文件中，要求完成如下功能：

- 打开 datafile.mat 数据文件。
- 在引擎应用程序中从数据文件获取 x 和 y 变量。
- 创建 MATLAB 计算引擎。
- 将变量 x 和 y 传递到 MATLAB 中。
- 执行 MATLAB 表达式 result = mldivide(x,y)，求解系统方程。
- 将计算结果保存到同一个数据文件中。

解答：

```
001     /*
002      * file: engexercise.c
003      *
```

```
004      *   从 MAT 文件中获取变量,
005      *   计算结果保存到数据文件中
006      */
007     #include "engine.h"
008     #include "mat.h"
009     void main()
010     {
011        mxArray *Xdata, *Ydata, *xvalues;
012        Engine *ep;
013        MATFile *mfp;
014        int status;
015        /* 打开数据文件读取数据 */
016        mfp = matOpen( "datafile.mat", "r" );
017        /* 读取数据 */
018        Xdata = matGetVariable( mfp, "X"    );
019        Ydata = matGetVariable(mfp, "Y" );
020        /* 关闭 MAT*文件 */
021        matClose(mfp);
022        /* 打开计算引擎连接 */
023        ep = engOpen(NULL);
024        /* 向计算引擎传递数据 */
025        engPutVariable(ep,"X",Xdata);
026        engPutVariable(ep,"Y",Ydata);
027        /* 调用 MATLAB 进行计算 */
028        engEvalString(ep,"resuit = mldivide(X,Y);");
029        /* 获取计算结果 */
030        xvalues = engGetVariable(ep,"result");
031        /* 打开 MAT 文件写入数据 */
032        mfp = matOpen("datafile.mat","u");
033        /* 写入数据 */
034        status = matPutVariable(mfp,"Xval",xvalues);
035        /* 关闭 MAT 文件 */
036        matClose(mfp);
037        /* 关闭计算引擎连接 */
038        status = engClose(ep);
039     }
```

参 考 文 献

1 *MATLAB External Interface Version 6*. Mathworks Inc, September 2003

2 *MATLAB External Interface Reference Version 6*. Mathworks Inc, September 2003

3 *Using MATLAB Version 6*. Mathworks Inc, September 2003

4 [美]Tom Archer. Visual C++ 6 宝典. 张艳，王文学，张谦，尹岩清等译. 北京：电子工业出版社，1999

5 [美]Cay S. Horstmann Gary Cornel. Java 2 核心技术 卷 I：基础知识. 京京工作室译. 北京：机械工业出版社，2000