Prentice-Hall

精通Matlab 6

Mastering Matlab® 6

A Comprehensive Tutorial and Reference

[美] D. Hanselman, B.Littlefield 著

张 航 黄 攀 译

清 华 大 学 出 版 社

Pearson Education培生教育出版集团

第 26 章　三 维 图 形

Matlab提供了多个函数来显示三维数据。有的函数用三维来显示曲线，而有的函数则负责绘制表面和构建框架。另外，可以用颜色来表示第四维。当以这种方式使用颜色的时候，它被称为伪色(pseudocolor)，因为颜色不再像在一张照片里边那样是图形的一个自然属性，而在这种用法中颜色已经不是底层数据的内在或者说是自然属性。为了简化对三维图形的讨论，颜色的用法被推迟到了下一章进行讲述。本章重点讨论了生成三维图形的基本概念。

26.1　曲　线　图

二维中的plot函数在三维中被扩展成了plot3。这个函数的用法和二维plot的用法是一样的，只是需要给plot3提供三个数据参数而不是两个。这个函数的常用调用格式为plot3(x1,y1,z1,s1,x2,y2,z2,s2,…)，其中xn，yn，zn为向量或者矩阵，而sn则是可选的用来声明颜色、标记符号以及/或者线型的字符串。plot3通常用来绘制一个单一变量的三维函数，例如：

```
>> t = linspace(0,10*pi);
>> plot3 (sin(t),cos(t),t)
>> xlabel('sin(t) '),ylabel('cos(t) '),zlabel('t')
>> text(0,0,0, 'Origin')
>> grid on
>> title('Figure 26.1: Helix')
>> v = axis
v =
    -1      1       -1      1       0       35
```

从这个简单的例子可以很清楚地看出，二维图形的所有基本特性在三维图形中仍然存在。命令axis通过返回z轴的轴限（0和35）作为轴向量的两个附加元素而扩展到了三维。命令grid在图形底部生成了一个三维的格栅，而命令box生成了一个包围图形的三维边框。plot3的默认设置为grid off和box off。函数text(x,y,z,'string')将一个字符串放置在由三个坐标值x，y，z标志的位置。另外，子图和多个图形窗口也可以直接应用到三维图形函数上。

在上一章中，通过给plot函数声明多个参数，或者通过使用hold命令，可以在一个图形上绘制多条直线或者曲线。plot3和其他三维图形函数提供了相同的功能。例如，plot3新增的维使得多个二维图形可以沿着某一维叠加在另一个二维图形上，而不是仅仅直接位于另一个图形的顶部。请看下边这个例子：

Figure 26.1　Heilix

```
>> x = linspace(0,3*pi);    % x-axis data
>> z1 = sin(x);     % plot in x-z plane
>> z2 = sin(2*x);
>> z3 = sin(3*x);
>> y1 = zeros(size(x)); % spread out along y-axes by giving
>> y3 = ones(size(x));  % each curve different y-axis values
>> y2 = y3/2;
>> plot3(x,y1,z1,x,y2,z2,x,y3,z3)
>> grid on
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 26.2: sin(x),sin(2x),sin(3x) ')
>> pause(5)
>> plot3(x,z1,y1,x,z2,y2,x,z3,y3)
>> grid on
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 26.3: sin(x),sin(2x),sin(3x) ')
```

26.2　含有两个变量的标量函数

与用plot3**生成曲线图相对应，用户往往希望将一个有两个变量的标量函数进行可视化显示，这个标量函数也就是：**

$$z = f(x, y)$$

Figure26.2 sin(x),sin(2x),sin(3x)曲线

Figure26.3 sin(x),sin(2x),sin(3x)曲线

这里，每一对x和y的值都会得到一个z值。变量z以x和y为自变量的图形在三维中就是一个面。为了在Matlab中绘制这个面，z的值被保存在一个矩阵中。正如在二维插值中所描

述的那样，假设x和y为自变量，那么z就是一个矩阵，并且x，y，z的关系为：

```
z(i,:) = f(x,y(i))
z(:,j) = f(x(j),y)
```

也就是说，z的第i行和y的第i个元素相关，而z的第j列和x的第j个元素相关。

当z=f(x,y)可以进行简单表达的时候，在一条语句中用数组运算符来计算出z的所有值是很方便的。为了做到这一点，需要用户在适当的方向上生成所有的x值和y值的矩阵。Mathworks公司通常称之为格子。Matlab提供了函数meshgrid来完成这个步骤，例如：

```
>> x = -3:3;    % choose x-axis values
>> y = 1:5;     % y-axis values
>> [X,Y] = meshgrid(x,y)
X =
    -3    -2    -1     0     1     2     3
    -3    -2    -1     0     1     2     3
    -3    -2    -1     0     1     2     3
    -3    -2    -1     0     1     2     3
    -3    -2    -1     0     1     2     3
y =
     1     1     1     1     1     1     1
     2     2     2     2     2     2     2
     3     3     3     3     3     3     3
     4     4     4     4     4     4     4
     5     5     5     5     5     5     5
```

正如用户所看到的那样，meshgrid将行向量x复制5次（y的列数）。类似地，它将y当作列向量复制7次（x的列数）。

> 要想记住某个变量被meshgrid复制了多少次，一个简单的方法就是想想二维图形。x轴从左到右变化，正像meshgrid的输出X一样。类似地，y轴从下到上变化，正像meshgrid的输出Y一样。

给定X和Y之后，如果$z=f(x,y)=(x+y)^2$，那么定义了这个三维面的数据值的矩阵就用如下的式子给出：

```
>> Z = (X+Y).^2
Z =
    4     1     0     1     4     9    16
    1     0     1     4     9    16    25
    0     1     4     9    16    25    36
    1     4     9    16    25    36    49
    4     9    16    25    36    49    64
```

当一个函数不能像上边那样简单表示的时候，用户必须用For循环或者While循环来计算Z的元素值。在很多情况下，还可以整行或者整列地计算Z的元素值。例如，如果可以整行计算Z的值，那么下面这段脚本文件就对用户有所帮助。

```
x= ???  % statement defining vector of x-axis values
y= ???  % statement defining vector of y-axis values
```

```
nx = length(x);    % length of x is no. of rows in Z
ny = length(y);    % length of y is no. of columns in Z
Z = zeros(nx,ny);  % initialize Z matrix for speed

for r=1:nx
   (preliminary commands)
   Z(r,:)= {a function of y and x(r) defining r-th row of Z}
end
```

另一方面，如果Z可以整列进行计算，下面这段脚本文件也会对用户有所帮助。

```
x= ???  % statement defining vector of x-axis values
y= ???  % statement defining vector of y-axis values

nx = length(x);     % length of x is no. of rows in Z
ny = length(y);     % length of y is no. of columns in Z
Z= zeros(nx,ny);    % initialize Z matrix for speed
for r =1:ny
   (preliminary commands)
   Z(:,c)= { a function of y(c) and x defining c-th column of Z}
end
```

只有在Z的元素值必须一个元素一个元素地进行计算的时候，才通常需要嵌套的For循环，比如下边这段脚本文件代码：

```
x= ???  % statement defining vector of x-axis values
y= ??? % statement defining vector of y-axis values
nx = length(x);    % length of x is no. of rows in Z
ny = length(y);     % length of y is no. of columns in Z
Z = zeros(nx,ny);  % initialize Z matrix for speed
for r=1:nx
      for c=1:ny
       (preliminary commands)
         Z(r,c)= {a function of y(c) and x(r) defining (r,c)-th element}
   end
end
```

26.3 网 眼 图

Matlab用在x-y平面上方形格栅上方的点的z坐标来定义一个网眼面。它通过将相邻的点用直线连接来构成了一个网眼图。这样形成的图形看上去有点像一个渔网，其节点就是数据点。请看下边这个例子：

```
>> [X,Y,Z] = peaks(30);
>> mesh(X,Y,Z)
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 26.4: Mesh Plot of Peaks')
```

Figure26.4 Mesh Plot of Peaks

请注意在用户的显示器上，线条的颜色和网眼的高度之间的关系。总的来说，mesh接受可选的参数来控制图中所使用的颜色。这个改变Matlab使用颜色方式的功能将在下一章中进行讨论。在任何情况下，对颜色的使用都被称作伪色，这是因为颜色被用来给图形添加了第四个有效维。还需要用户注意的是，绘制的图形是带了格栅的。除了plot3之外的大多数的三维图形以及一些其他图形的默认设置都是grid on。

除了上边的输入参数之外，mesh和大多数的三维图形函数还可以用不同的输入参数进行调用。这里所使用的调用语法是最特殊的一种调用语法，因为它提供了所有三个轴的信息。mesh(Z)绘制出矩阵Z相对于其行和列索引的图形。最常用的调用语法是使用传递给meshgrid作为x轴和y轴的向量，例如，mesh(x,y,Z)。

正如在上边的图形中所显示的那样，在网眼线之间的区域是不透明的。Matlab命令hidden负责控制网眼图在这方面的属性，例如：

```
>> <X,Y,Z]=sphere(12);
>> subplot(1,2,1)
>> mesh(X,Y,Z),title('Figure 26.5a: Opaque')
>> hidden on
>> axis square off
>> subplot(1,2,2)
>> mesh(X,Y,Z),title('Figure 26.5b: Transparent')
>> hidden off
>> axis square off
```

Figure 26.5a Opaque

Figure 26.5b Transparent

左边这个球体就是不透明的（后边的线条被隐藏了），而右边这个球体是透明的（后边的线条没有被隐藏）。

Matlab**的**mesh**函数有两个同类函数，一个是**meshc**，它是一个网眼图以及等值线函数，另一个是**meshz**，它是一个包含了**0**平面的网眼图函数。请看下边这个例子：**

```
>> [X,Y,Z] = peaks(30);
>> meshc(X,Y,Z)     % mesh plot with underlying contour plot
>> title('Figure 26.6: Mesh Plot with Contours')
>> pause(5)
>> meshz(X,Y,Z)     % mesh plot with zero plane
>> title('Figure 26.7: Mesh Plot with Zero Plane')
```

Figure 26.6 Mesh Plot with Contours

函数waterfall**和**mesh**是相同的，只是其网眼线只在**x**轴的方向上显示，例如：**

```
>> waterfall(X,Y,Z)
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 26.8: Waterfall Plot')
```

Figure 26.7　Mesh Plot with Zero Plane

Figure 26.8　Waterfall Plot

26.4　表 面 图

表面图看上去类似于网眼图，只是在网眼线之间的空间（称为碎片）是被填充了颜色

的。这种类型的图形是使用surf函数生成的，例如：

```
>> [X,Y,Z] = peaks(30);
>> surf(X,Y,Z)
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 26.09: Surface Plot of Peaks')
```

Figure 26.9 Surface Plot of Peaks

请注意，这个图形类型是一个复式的mesh图形。这里，线条是黑色的而碎片有颜色，但是在mesh中，碎片的颜色就是坐标轴的颜色，而线条是有颜色的。就像mesh图形一样，颜色是沿着z轴的各个碎片变化的，而线条有恒定不变的颜色。表面图形的默认设置也是grid on。

在一个表面图形中，用户不会像在一个mesh图形中那样考虑是否隐藏线条，而是要考虑遮蔽表面的不同方法。在上边的surf图形中，阴影就像一个彩色玻璃的窗户或者物体那样是分成块面的，其中黑线是在不变颜色的碎片之间的接缝。除了分成块面，Matlab还提供了平面投影以及插值投影。这些函数是通过使用函数shading实现的，例如：

```
>> [X,Y,Z] = peaks(30);
>> surf(X,Y,Z)  % same plot as above
>> shading flat
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 26.10: Surface Plot with Flat Shading')
>> pause(5)
>> shading interp
>> title('Figure 26.11: Surface Plot with Interpolated Shading')
```

Figure 26.10　Surface Plot with Flat Shading

Figure 26.11　Surface Plot with Interpolated Shading

在下一页所显示的平面投影中，黑线都被去掉了，并且每一个碎片都保持了它自己的单色，而在插值投影中，黑线也被去掉了，但是每个碎片都有插值投影。也就是说，每个碎片的颜色都根据分配给每个高度值的颜色在它的区域中进行了插值。很明显，插值投影需要比块面投影和平面投影多的多的计算量。投影对于surf图形有重要的视觉影响，它也被应用到了mesh图形中，尽管在这种情况下，因为只有线条有颜色，视觉效果相对来说要小

一些。投影还会影响到pcolor和fill图形。

在有的计算机系统中，插值投影会带来特别长的打印延迟，在最坏的情况下，还会导致打印错误。这些问题不是由PostScript数据文件引起的，而是由为了生成在图形表面上连续变化的阴影，而在打印机中所需要进行的大量计算引起的。通常这个问题最简单的解决办法就是用平面投影作为打印输出。

在有的情况下，用户可能需要将一个表面的某个部分删除，以便能够看见表面底层的部分。在Matlab中，这是通过将空洞想要出现的位置的数据值设定为NaN来实现的。因为NaN没有值，因此所有的Matlab绘图函数都会忽略NaN数据点，就在NaN出现的位置留出一个空洞。请看下面这个例子：

```
>> [X,Y,Z] = peaks(30);
>> x = X(1,:);              % vector of x-axis
>> y = Y(:,1);              % vector of y-axis
>> i = find(y>.8 & y<1.2);  % find y-axis indices of hole
>> j = find(x>-.6 & x<.5);  % find x-axis indices of hole
>> Z(i,j) = nan;            % set values at hole indices to NaNs
>> surf(X,Y,Z)
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 26.12: Surface Plot with a Hole');
```

Figure 26.12 Surface Plot with a Hole

Matlab的surf函数也有两个同类函数：一个是surfc，它是一个表面绘图和底层等高线绘图函数；另一个是surfl，它是一个有光照的表面绘图函数，例如：

```
>> [X,Y,Z]= peaks(30);
>> surfc(X,Y,Z) % surf plot with contour plot
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 26.13: Surface Plot with Contours')
>> pause(5)
```

```
>> surfl(X,Y,Z)     % surf plot with lighting
>> shading interp   % surfl plots look best with interp shading
>> colormap pink    % they also look better with shades of a single color
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 26.14: Surface Plot with Lighting')
```

Figure 26.13　Surface Plot with Contours

Figure 26.14　Surface Plot with Lighting

函数surfl**利用了与应用到表面的光照有关的一系列假设。它没有用到下一章中将讨论的**light**对象。相反，它仅仅是将表面的颜色进行了修改以便得到光照的效果。下一章提供**

了关于光照属性的更严格信息。另外，在上边这些被执行的命令中，colormap是一个用来将不同的颜色集应用到一个图形的Matlab函数。这个函数也将在下一章中进行讨论。

函数surfnorm(X,Y,Z)为用X，Y和Z定义的表面计算表面法线，绘制出表面，然后在数据点的位置绘制出表面的法线向量。表面法线是没有经过标准化的，并且在每个定点都有效。[Nx,Ny,Nz]=surfnorm(X,Y,Z)计算出三维表面的法线，然后将这些法线的元素返回，但是并不会绘制这个表面，例如：

```
>> [X,Y,Z] = peaks(15);
>> surfnorm(X,Y,Z)
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 26.15: Surface Plot with Normals')
```

Figure 26.15 Surface Plot with Normals

26.5 不规则数据的网眼图和表面图

不规则数据或者不均匀间隔的数据可以用函数trimesh，trisurf和voronoi来进行可视化，例如：

```
>> x = rand(1,50);
>> y = rand(1,50);
>> z = peaks(x,y*pi);
>> t = delaunay(x,y);
>> trimesh(t,x,y,z)
>> hidden off
>> title('Figure 26.16: Triangular Mesh Plot')
>> pause(5)
```

```
>> trisurf(t,x,y,z)
>> title('Figure 26.17: Triangular Surface Plot')
>> pause(5)
>> voronoi(x,y,t)
>> title('Figure 26.18: Voronoi Plot')
```

Figure 26.16　Triangular Mesh Plot

Figure 26.17　Triangular Surface Plot

Figure 26.18 Voronoi Plot

关于Delaunay三角形和Voronoi图形的更多信息请参见第18章。

26.6 改 变 视 角

请注意，三维图形默认的视角是向下看z=0的平面，角度为30度，并且同时向上看x=0的平面，角度为37.5度。与z=0平面相关的这个方向角度被称作仰角，而与x=0这个平面有关的角度被称作方位角。这样，默认的三维视角就是仰角为30度而方位角为-37.5度。默认的二维视角就是仰角为90度而方位角为0度。方位角和仰角的概念可以用下面的图形形象地描述。

在Matlab中，函数view改变所有二维图形和三维图形的视角。view(az,el)和view([az,el])将视角改变为指定的方位角az和仰角el。请看下边这个例子：

```
>> x = -7.5:.5:7.5; y = x;          % create a data set
>> [X,Y] = meshgrid(x,y);
>> R = sqrt(X.^2+Y.^2)+eps;
>> Z = sin(R)./R;
>> subplot(2,2,1)
>> surf(X,Y,Z)
>> view(-37.5,30)
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 26.19a: Az = -37.5,El = 30')

>> subplot(2,2,2)
>> surf(X,Y,Z)
>> view(-37.5+90,30)
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 26.19b: Az Rotated to 52.5')
```

```
>> subplot(2,2,3)
>> surf(X,Y,Z)
>> view(-37.5,60)
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 26.19c: El Increased to 60')

>> subplot(2,2,4)
>> surf(X,Y,Z)
>> view(0,90)
>> xlabel('X-axis'),ylabel('Y-axis')
>> title('Figure 26.19d: Az = 0,El = 90')
```

Figure26.19a　Az=-37.5　El=30　　Figure26.19b　Az 旋转为 52.5

Figure26.19c E1 增加到 60　　Figure 26.19d Az=0,E1=90

除了上边的形式之外，view还提供了一些额外的特性，在它的帮助文档中总结如下：

```
>> help view
 VIEW   3-D graph viewpoint specification.
     VIEW(AZ,EL) and VIEW([AZ,EL]) set the angle of the view from which an
     observer sees the current 3-D plot. AZ is the azimuth or horizontal
     rotation and EL is the vertical elevation (both in degrees). Azimuth
     revolves about the z-axis, with positive values indicating counter-
     clockwise rotation of the viewpoint. Positive values of elevation
     correspond to moving above the object; negative values move below.
     VIEW([X Y Z]) sets the view angle in cartesian coordinates. The
     magnitude of vector X,Y,Z is ignored.

     Here are some examples:

     AZ = -37.5, EL = 30 is the default 3-D view.
     AZ = 0, EL = 90 is directly overhead and the default 2-D view.
     AZ = EL = 0 looks directly up the first column of the matrix.
     AZ = 180 is behind the matrix.
     VIEW(2) sets the default 2-D view, AZ = 0, EL = 90.
     VIEW(3) sets the default 3-D view, AZ = -37.5, EL = 30.

     [AZ,EL] = VIEW returns the current azimuth and elevation.

     VIEW(T) accepts a 4-by-4 transformation matrix, such as
     the perspective transformations generated by VIEWMTX.

     T = VIEW returns the current general 4-by-4 transformation matrix.

     See also VIEWMTX, the AXES properties View, Xform.
```

除了view函数之外，还可以用函数rotate3d函数来利用鼠标交互地设置视角。rotate3d on将基于鼠标的视角旋转打开，rotate3d off将它关闭，而不带参数的rotate3d实现这两个状态之间的切换。这个功能还可以从图形窗口中的Tools菜单栏获得，也可以通过选择图形工具栏中的按钮来获得。实际上，Rotate 3D菜单项和工具栏按钮都调用了rotate3d来实现用户需要的操作。

26.7 摄像头控制

函数view所提供的视角控制是很方便的，但是在功能上却非常有限。为了提供对三维场景的完全控制，需要用到摄像头功能。也就是说，用户在用一个摄像头拍摄一部电影的时候，必须获得摄像头的全部功能。或者作为一种替代，用户必须获得三维计算机或者是控制台游戏环境的所有功能。在这样的环境中，有两个三维坐标系统需要进行管理——一

个在摄像头那里，另一个在摄像头所指向的位置，也就是摄像头目标。Matlab中的摄像头函数管理和处理这两个坐标系统之间的关系，并且提供对摄像头镜头的控制。

对一个新手而言，Matlab中摄像头函数的使用总的来说不是一件容易的事情。因此，为了简化这些函数的使用，大多数的函数都可以通过Tools菜单或者从图形窗口的View菜单交互地获得。利用这些交互的摄像头工具，用户就可以避免命令窗口函数所需要的输入和输出参数。虽然这些交互工具带来了极大的方便，但是考虑到使用和描述这些摄像头函数的复杂性，以及相对而言较少的潜在用户，因此本书没有对摄像头函数进行详细的描述。Matlab图形手册（graphg.pdf）中有对这些函数严格的讨论。本章的最后列出了Matlab中提供的摄像头函数。

26.8　等 高 线 图

等高线图显示了相同海拔或高度的曲线。如果用户曾经见过地形图，用户就应该了解等高线图。在Matlab中，二维等高线图和三维等高线图分别是由函数contour和countour3生成的。例如：

```
>> [X,Y,Z] = peaks;
>> contour(X,Y,Z,20)            % generate 20 2-D contour lines
>> xlabel('X-axis'),ylabel('Y-axis')
>> title ('Figure 26.20: 2-D Contour Plot')
>> pause(5)
>> contour3(X,Y,Z,20)           % the same contour plot in 3-D
>> xlable('X-axis'), ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 26.21: 3-D Contour Plot')
```

Figure 26.20　2-D Contour Plot

Figure 26.21 3-D Contour Plot

函数pcolor**按照高度进行绘图，它将不同的高度表示成不同的颜色，然后在相同的刻度上以等高线的形式提供相同的信息。例如：**

```
>> pcolor(X,Y,Z)
>> shading interp   % remove the grid lines
>> title('Figure 26.22: Pseudocolor Plot')
```

Figure 26.22 Pseudocolor Plot

将伪色图的思想和二维等高线组合起来，就生成了一幅填充了颜色的等高线图。在Matlab中，这种图形是用函数contourf生成的，例如：

```
>> contourf(X,Y,Z,12)           % filled contour plot with 12 contours
>> xlabel('X-axis'), ylabel('Y-axis')
>> title('Figure 26.23:Filled Contour Plot')
```

Figure 26.23　Filled Contour Plot

等高线可以用clabel函数进行标记。clabel需要一个线条矩阵和可选的文本字符串，这些字符串是contour，contourf和contour3返回的字符串，例如：

```
>> C = contour(X,Y,Z,12);
>> clabel(C)
>> title('Figure 26.24: Contour Plot With Labels')
```

另外，用两个参数可以生成嵌入在等高线内部的标签。这些标签只是在空间足够大的地方才显示，例如：

```
>> [C,h] = contour(X,Y,Z,8);
>> clabel(C,h)
>> title('Figure 26.25  Contour Plot With In-line Labels')
```

最后，用户可以通过将'manul'作为最后一个参数提供给clabel函数，来用鼠标选择给哪条等高线添加标签。嵌入或者水平标签的使用，要依赖于如上边显示的第二个参数h是否存在。

igure 26.24 Contour Plot With Labels

Figure 26.25 Contour Plot With In-line Labels

26.9 特殊的三维图形

除了上边讨论的函数之外，Matlab**还提供了一些专用的绘图函数。函数**ribbon(Y)**将**Y**的列画成一个一个的带状。**ribbon(x,Y)**绘制**x-Y**的列图。还可以用**ribbon(x,Y,width)**这样的调**

用语法来声明这些带的宽度，默认的宽度为0.75。请看下边这个例子：

```
>> Z = peaks;
>> ribbon(Z)
>> title('Figure 26.26: Ribbon Plot of Peaks')
```


Figure 26.26　Ribbon Plot of Peaks

函数quiver(x,y,dx,dy)在点（x,y）画出方向或速度向量（dx,dy），例如：

```
>> [X,Y,Z] = peaks(16);
>> [DX,DY] = gradient(Z,.5,.5):
>> contour(X,Y,Z,10)
>> hold on
>> quiver(X,Y,DX,DY)
>> hold off
>> title('Figure 26.27:2-D Quiver Plot')
```

形如quiver3(x,y,z,Nx,Ny,Nz)的三维箭头显示了在点（x,y,z）处的向量（Nx,Ny,Nz），例如：

```
>> [X,Y,Z] = peaks(20);
>> [Nx,Ny,Nz] = surfnorm(X,Y,Z);
>> surf(X,Y,Z)
>> hold on
>> quiver3(X,Y,Z,Nx,Ny,Nz)
>> hold off
>> title('Figure 26.28: 3-D Quiver Plot')
```

Figure 26.27 2-D Quiver Plot

Figure 26.28 3-D Quiver Plot

函数fill3**，也就是**fill**在三维中的等效函数，在三维空间中画出填充了颜色的多面体。**fill(X,Y,Z,C)**用数组**X**，**Y**和**Z**作为这个多面体的顶点，**C**声明了填充的颜色，例如：**

```
>> fill3(rand(3,5),rand(3,5),rand(3,5),rand(3,5))
>> grid on
>> title('Figure 26.29: Five Random Filled Triangles')
```

Figure 26.29 Five Random Filled Triangles

stem在三维中的等效函数绘制出三维空间中的离散数据序列。stem3(X,Y,Z,C, 'filled')用延伸到x-y平面的线条绘制出在（X,Y,Z）上的点。可选的参数C声明了标记类型以及/或者颜色，可选的'filled'参数使得标记被填充上颜色。stem3(Z)绘制出在Z中的点，并且自动地生成X和Y值。例如：

```
>> Z = rand(5);
>> stem3(Z, 'ro', 'filled');
>> grid on
>> title('Figure 26.30: Stem Plot of Random Data')
```

26.10 立体可视化

除了常用的网眼图、表面图和等高线图之外，Matlab还提供了几个更为复杂的立体和向量可视化函数。这些函数在三维空间中构建标量和向量图形。因为他们通常构建的是立体而不是表面，因此它们的输入参数是三维数组，每一维代表一个坐标轴x，y和z。在每个三维数组中的点定义了一个坐标轴格栅或者坐标轴的数据。对于标量函数而言，需要4个三维数组，其中的3个各代表了一个坐标轴，剩下的一个代表了在这些坐标处的标量数据。这些数组通常分别标记为X，Y，Z和V。对于向量函数而言，需要6个三维数组，其中的3个各表示一个坐标轴，剩下的3个各表示在坐标点处向量的一个坐标轴。这些数组通常被分别表示为X，Y，Z，U，V和W。

Figure 26.30 Stem Plot of Random Data

使用Matlab中的立体和向量可视化函数需要对立体和向量的术语有一定的了解。例如，divergence和curl描述了向量过程，而isosurfaces和isocaps则描述了立体的视觉外观。如果用户对这些术语不是很熟悉，那么使用Matlab立体和可视化函数的时候就会感到有些困惑。为了严格地使用这些立体和向量可视化函数而讲述相应术语已经超出了本书的范围。但是，下面我们还是展示了数据数组的结构和几个函数的用法。要想获得更进一步的信息，请参见Matlab附带的使用Matlab图形手册(graphg.pdf)。另外，输入命令volvec就能够获得Matlab中大多数立体可视化函数交互式显示的图形用户界面。

请考虑一下定义在一个立体上的标量函数的构建。首先，必须创建这个立体坐标轴，例如：

```
>> x = linspace(-3,3,13);       % x-coordinate points
>> y = 1:20;                    % y-coordinate points
>> z = -5:5;                    % Z-coordinate points
>> [X,Y,Z] = meshgrid(x,y,z);   % meshgrid works here too!
>> size(X)
ans =
    20      13      11
```

这里，X，Y，Z为定义了格栅的三维数组。X包含了被复制了length(y)这么多行和length(z)那么多页的x值。类似地，Y先被转置成一个列向量，然后被复制了length(x)那么多列和length(z)那么多页。Z被排列成一个1×1×length(z)的向量，并且被复制了length(y)那么多行和length(x)那么多列。正如这里所叙述的那样，这是将meshgrid直接扩展到了三维的情况。

下面，我们需要定义这些数据的一个函数，例如：

```
>> V = sqrt(X.^2 + cos(Y).^2 + Z.^2);
```

现在，三维数组X，Y，Z和V定义了一个定义在一个立体上的标量函数v=f(x,y,z)。为了形象地展示这个函数，我们可以沿着平面查看各个截面。例如：

```
>> slice(X,Y,Z,V,[0 3],[5 15],[-3 5])
>> xlabel('X-axis')
>> ylabel('Y-axis')
>> zlabel('Z-axis')
>> title('Figure 26.31: Slice Plot Through a Volume')
```

Figure 26.31　Slice Plot Through a Volume

这幅图形显示了x=0，x=3，y=5，y=15，z=-3和z=5所定义的平面上的截面，这些参数，如上所示，作为函数slice的最后的三个参数而使用。这幅图的颜色是根据截面上V的值来进行绘制的。

所显示的截面不一定非平面不可。它们可以是任意的表面，例如：

```
>> [xs,ys] = meshgrid(x,y);
>> zs = sin(-xs+ys/2);              % a surface to use
>> slice(X,Y,Z,V,xs,ys,zs)
>> xlabel('X-axis')
>> ylabel('Y-axis')
>> zlabel('Z-axis')
>> title('Figure 26.32: Slice Plot Using a Surface')
```

这里xs，ys和zs定义了一个贯穿这个立体的截面。

让我们再回到刚才的那个截面图，用户还可以用contourslice函数来给选定的平面添加等高线，例如：

Figure 26.32 Slice Plot Using a Surface

```
>> slice(X,Y,Z,V,[0 3],[5 15],[-3 5])
>> h = contourslice(X,Y,Z,V,3,[5 15],[]);
>> set(h,'EdgeColor', 'k', 'Linewidth',1.5)
>> xlabel('X-axis')
>> ylabel('Y-axis')
>> zlabel('Z-axis')
>> title('Figure 26.33: Slice Plot with Selected Contours')
```

Figure 26.33 Slice Plot with Selected Contours

这里，等高线被添加到x=3，y=5和y=15的平面上。利用句柄图形特性，等高线被设定为黑色的，并且它们的宽度被设定为1.5个像素点。

除了查看贯穿一个立体的截面之外，标量立体数据V等于某个特定值的表面也可以用isosurface函数绘制出来。这个函数以类似于Delaunay三角形的方式返回三角形的顶点，返回的结果正是patch函数所需要的格式，这个函数就可以绘制出这个三角形。请看下面这个例子：

```
>>[X,Y,Z,V] = flow(13);              % get flow data
>> fv = isosurface(X,Y,Z,V,-2);     % find surface of value –2
>> subplot(1,2,1)
>> p= patch(fv);                     % plot V = -2 surface
>> set (p, 'FaceColor',[.5 .5 .5], 'EdgeColor', 'Black');  % modify
                                     patches
>> view(3), axis equal tight, grid on   % pretty it up
>> title({'Figure 26.34a: ' 'Isosurface Plot, V = 2'})
>> subplot(1,2,2)
>> p = patch(shrinkfaces(fv,.3));       % shrink faces to 30% of original
>> set(p, 'Facecolor',[.5 .5 .5], 'EdgeColor', 'Black');
                                        % modify patches
>> view(3),axis equal tight,grid on     % pretty it up
>> title({'Figure 26.34b: ' ' Shrunken Face Isosurface Plot, V = 2'})
```

Figure 26.34a: ' 'Isosurface Plot, V=2　　Figure 26.34b: Shrunken Face Isosurface Plot, V = 2'

前面的图形还展示了函数shrinkfaces的用法，这个函数的功能从它的函数名得到了体现。

有的时候，立体数据包含了太多的点，因而无法进行高效地显示。函数reducevolume和reducepatch代表了改进isosurface显示效果的两种方法。函数reducevolume在isosurface形成之前就先删除一些数据，而reducepatch则试图删除一些碎片，而同时将底层表面的失真减少到最小。请看下边这个例子：

```
>> [X,Y,Z,V] = flow;
>> fv = isosurface(X,Y,Z,V,-2);
```

```
>> subplot(2,2,1)   % Original
>> p = patch(fv);
>> Np = size(get(p,'Faces'),1);
>> set(p, 'FaceColor',[.5 .5 .5], 'EdgeColor', 'Black');
>> view(3),axis equal tight,grid on     % pretty it up
>> zlabel(sprintf('%d Patches',Np))
>> title('Figure 26.35a: Original')

>> subplot(2,2,2)   % Reduce Volume
>> [Xr,Yr,Zr,Vr] = reducevolume(X,Y,Z,V,[3 2 2]);
>> fvr = isosurface(Xr,Yr,Zr,Vr,-2);
>> p = patch(fvr);
>> Np = size(get(p, 'Faces'),1);
>> set(p, 'FaceColor',[.5 .5 .5], 'EdgeColor', 'Black');
>> view(3),axis equal tight,grid on     % pretty it up
>> zlabel(sprintf('%d Patches',Np))
>> title('Figure 26.35b: Reduce Volume')

>> subplot(2,2,3)   % Reduce Patch
>> p = patch(fv);
>> set(p, 'FaceColor',[.5 .5 .5], 'EdgeColor', 'Black');
>> view(3),axis equal tight,grid on     % pretty it up
>> reducepatch(p,.15)   % keep 15 percent of the faces
>> Np = size(get(p, 'Faces'),1);
>> zlabel(sprintf('%d Patches',Np))
>> title('Figure 26.35c: Reduce Patches')

>> subplot(2,2,4)   % Reduce Volume and Patch
>> p = patch(fvr);
>> set(p, 'FaceColor',[.5 .5 .5], 'EdgeColor', 'Black');
>> view(3),axis equal tight,grid on     % pretty it up
>> reducepatch(p,.15)   % keep 15 percent of the faces
>> Np = size(get(p, 'Faces'),1);
>> zlabel(sprintf('%d Patches',Np))
>> title('Figure 26.35d: Reduce Both')
```

Figure26.35a: Original

Figure 26.35b Reduce Volume

Figure 26.35c Reduce Patches

Figure 26.35d Reduce Both

三维数据也可以通过用smooth3**函数来过滤而实现其平滑化，例如：**

```
>> data = rand(10,10,10);           % random data
>> datas = smooth3(data, 'box',3);      % smoothed data

>> subplot(1,2,1)   % random data
>> p = patch(isosurface(data,.5),...
    'FaceColor', 'Blue', 'EdgeColor', 'none');
>> patch(isocaps(data,.5),...
    'FaceColor', 'interp', 'EdgeColor', 'none');
>> isonormals(data,p)
>> view(3); axis vis3d tight off
>> camlight; lighting phong
>> title({'Figure 26.36a: ' ' Random Data'})

>> subplot(1,2,2)   % smoothed random data
>> p = patch(isosurface(datas,.5),...
    'FaceColor', 'Blue', 'EdgeColor', 'none');
>> patch(isocaps(datas,.5),...
    'FaceColor', 'interp', 'EdgeColor', 'none');
>> isonormals(datas,p)
>> view(3); axis vis3d tight off
>> camlight; lighting phong
>> title({'Figure 26.36b: ' ' Smoothed Data'})
```

Figure 26.36a Random Data

Figure 26.36b Smoothed Data

上边的例子展示了函数isocaps和isonormals的用法。函数isocaps生成块状图的外层表面。函数isonormals调整所画碎片的属性，使得所显示的图形有正确的光照效果。

关于Matlab中立体和向量可视化特性的更进一步信息，请参见使用Matlab图形手册，并尝试使用一下volvec函数。

26.11 轻 松 绘 图

当用户不想花费时间来显式地声明一个三维图形数据点的时候，Matlab提供了函数ezcontour，ezmesh，ezmeshc，ezplot3，ezsurf和ezsurfc。这些函数构建的图形类似于它们不带ez前缀的等价函数所构建的图形。但是，它们的输入参数是函数，这些函数是由字符串或符号数学对象，以及可选的图形所在的坐标轴限所定义的。在这些函数内部，它们对数据进行计算，然后生成想要的图形，例如：

```
>> fstr = ['3*(1-x).^2.*exp(-(x.^2) – (y+1).^2)' …
  ' – 10*(x/5 – x.^3 – y.^5).*exp(-x.^2-y.^2)' …
  ' – 1/3*exp(-(x+1).^2 – y.^2)'];
>> subplot(2,2,1)
>> ezmesh(fstr)
>> title('Figure 26.37a: Mesh of peaks(x,y)')
>> subplot(2,2,2)
>> ezsurf(fstr)
>> title('Figure 26.37b: Surf of peaks(x,y)')
>> subplot(2,2,3)
>> ezcontour(fstr)
>> title('Figure 26.37c: Contour of peaks(x,y)')
>> subplot(2,2,4)
>> ezcontourf(fstr)
>> title('Figure 26.37d: Contourf of peaks(x,y)')
```

Figure 26.37a Mesh of peaks(x,y)

Figure 26.37b Surf of peaks(x,y)

Figure 26.37c Contour of peaks(x,y)　　Figure 26.37d Contourf of peaks(x,y)

26.12 小　结

下面这个表格列出了Matlab中进行三维绘图的函数。

函数	描述
plot3	在三维空间中绘制曲线和点
mesh	绘制网眼表面
meshc	绘制带底层等高线的网眼图
meshz	绘制带零平面的网眼图
surf	绘制表面图
surfc	绘制带底层等高线图的表面图
surfl	绘制有基本光照的表面图
fill3	绘制一个填充了颜色的三维多面体
shading	颜色投影模式
hidden	网眼隐藏线删除
surfnorm	表面法线
axis	控制坐标轴刻度和外观
grid	格栅线是否可见
box	坐标轴边框是否可见
hold	保持当前的图形
subplot	在图形窗口中生成多个坐标轴
daspect	设定数据屏幕高宽比
pbaspect	设置图形框屏幕高宽比
xlim	X坐标轴限
ylim	Y坐标轴限
zlim	Z坐标轴限
view	三维视角声明
viewmtx	视角转换矩阵

续表

函数	描述
rotate3d	交互坐标轴旋转
campos	摄像头位置
camtarget	摄像头目标
camva	摄像头视角
camup	设置窗口相对于显示对象的位置向量
camproj	摄像头投影
camorbit	摄像头旋转
campan	固定窗口位置旋转对象
camdolly	移动式摄像头
camzoom	放大摄像头
camroll	滚动摄像头
camlookat	查看特定的对象
camlight	摄像头光线生成和放置
title	图形标题
xlabel	X轴标签
ylabel	Y轴标签
zlabel	Z轴标签
text	在图中放置文本
gtext	在鼠标点击的位置放置文本
contour	绘制等高线图
contourf	填充等高线图
contour3	三维等高线图
clabel	等高线标签
pcolor	伪色图形
voronoi	Voronoi图
trimesh	三角形网眼图
trisurf	三角形表面图
scatter3	三维散布图
stem3	三维主干图
waterfall	瀑布图
ezmesh	字符串表达式的简易网眼图
ezmeshc	字符串表达式带等高线的简易网眼图
ezplot3	字符串表达式的简易三维曲线图
ezsurf	字符串表达式的简易表面图
ezsurfc	字符串表达式的带等高线的简易表面图
ezcontour	字符串表达式的简易等高线图

续表

函数	描述
ezcontourf	字符串表达式填充了颜色的简易等高线图
vissuite	可视化组的帮助文档
isosurface	从体积数据中选出表面数据
isonormals	Isosurface法线
isocaps	Isosurface边缘属性
isocolors	Isosurface和碎片颜色
contourslice	截面平面中的等高线
slice	立方体的截面图
streamline	数据的流线
stream3	三维数据流线
stream2	二维数据流线
quiver3	三维箭形图
quiver	二维箭形图
divergence	一个向量域的散度
curl	一个向量域的曲度和角速度
coneplot	锥形图
streamtube	流形管
streamribbon	流形带
streamslice	截面中的流形线
streamparticles	显示流粒子
interpstreamspeed	对速度中的流线顶点进行插值
subvolume	提取立方数据集合中的子集
reducevolume	削减立方数据集
volumebounds	返回体积和颜色限
smooth3	将三维数据平滑化
reducepatch	减少碎片表面的数量
shrinkfaces	缩小碎片表面的大小

第 27 章　使用颜色和光照

Matlab提供了一些工具来在二维或者三维中可视地显示信息。例如，正弦曲线肯定比一组数据点提供了更多的信息。利用图形和图表来展示数据集的技术被称为数据可视化。除了它是一个功能强大的计算引擎之外，Matlab在有趣的和提供大量信息的方式可视地表示数据方面也是非常出色的。

但是，通常一个简单的二维或者三维图形无法显示出用户希望同时显示的所有信息。颜色可以作为另外的一维来提供信息。在前一章中所讨论的很多绘图函数就接受一个颜色参数，这个参数可以用来添加这个额外的维。

关于这个问题的讨论让我们从对颜色表的研究开始：怎样使用、显示、改变以及生成它们。下一步，我们将展示在一个图形窗口中仿真多个颜色图的技术，或者只使用一个颜色表中的一部分的技术。最后，我们讨论了光照模型，并展示了一些示例。与上一章中的图形一样，本章所显示的图形在书中也无法表现出颜色，虽然在计算机显示器上它们确实是有颜色的。因此，如果用户在学习本书的时候没有边学边用Matlab，那么要想理解本章中所提到的概念，也许就需要用户发挥自己的想象力了。

27.1　理解颜色表

Matlab使用了一个有三列的数值数组来表示颜色值。这个数组被称作颜色表，这个矩阵中的每一行用从0到1的数字代表了一个不同的颜色。每行的数字表示了构成一个特定颜色的红、绿和蓝色的强度。下边这个表格展示了一个颜色表的数值的值和颜色之间的对应关系。

红	绿	蓝	颜色
1	0	0	红
0	1	0	绿
0	0	1	蓝
1	1	0	黄
1	0	1	洋红
0	1	1	青
0	0	0	黑
1	1	1	白
0.5	0.5	0.5	中灰
0.67	0	1	紫色
1	0.4	0	橙色
0.5	0	0	暗红
0	0.5	0	暗绿

颜色表中的第一列是红色的强度，第二列是绿色的强度，第三列为蓝色的强度。颜色表的值被严格地限制在0到1之间。

颜色表是一个包含了红－绿－蓝（RGB）值的行序列，它从第一行到最后一行按照某种规定的方式进行变化。Matlab提供了一些预先定义好的颜色表，如下表所示：

颜色表函数	描述
hsv	色度－饱和度－亮度（HSV）颜色表，以红开始，以红结束
jet	HSV颜色表的变种，以蓝色开始，以红色结束
hot	黑色－红色－黄色－白色
cool	暗青色和暗洋红
summer	暗绿色和暗黄色
autumn	暗红色和暗黄色
winter	暗绿色和暗蓝色
spring	暗洋红和暗黄色
white	全白
gray	线性灰度比例
bone	轻微染了蓝色的灰色
pink	粉红色的柔和投影
copper	线性铜色调
prism	红色，橙色，黄色，绿色，蓝色和紫色的交替
flag	红色，白色，蓝色和黑色的交替
lines	交替绘制线性颜色
colorcube	增强的颜色立方体

默认地，上边这些颜色表中的每一个都会生成一个64×3的数组，这个数组声明了64种颜色的RGB描述。这些函数中的每一个都会接受声明了需要生成的行数的参数。例如，hot(m)就生成一个m*3的矩阵，这个矩阵包含了从黑色、暗红、橙色、黄色到白色的颜色的RGB值。

大多数计算机都可以至少同时显示一个8位硬件颜色表中的256种颜色，虽然很多计算机现在都有了可以处理24位或者更多位的颜色的显卡（这样的显卡支持的颜色种类超过了1600万种）。这意味着保守地估计，在不同的使用256色模式的图形中，可以同时最多有3个或者4个正在使用的64×3的颜色表。更多的64×3颜色表可以通过将视频卡设置到更大的颜色深度来支持。如果比硬件支持的数量更多的颜色表项目被使用，计算机就将其硬件查用表中的项目换出。在这种情况下，屏幕的背景颜色在绘制Matlab图形的时候就会发生变化。

27.2　使用颜色表

语句colormap(M)将矩阵M安装成在当前“图形”窗口中要使用的颜色表。例如，

colormap(cool)就安装cool颜色表的有64个项目的版本。colormap default安装默认的颜色表，通常是hsv或者jet，究竟是哪一种这要依赖于用户用命令colordef所选择的默认颜色机制。

诸如plot和plot3这样的线性绘图函数没有用到颜色表；它们用到了在plot颜色和线型列表中列出的颜色。这些函数所用到的颜色序列随着用户选择的绘图类型的变化而变化。大多数绘图函数，比如mesh、surf、contour、fill、pcolor以及它们的变种，都使用当前的颜色表来决定颜色序列。

接受color参数的绘图函数通常以下列的三种方式之一接受这个参数：

- 一个表示了plot颜色和线型表中的一种颜色的字符串，比如用'r'或者'red'来表示红色。
- 一个表示了单个RGB值的有三个条目的行向量，例如，[.25 .50 .75]。
- 一个数组。如果颜色参数是一个数组，那么其元素就被标定，并用作为指向当前颜色表的索引。

27.3 显示颜色表

可以用多种方式来浏览颜色表。其中一种方式就是直接查看一个颜色表矩阵中的元素，例如：

```
>> hot(8)
ans =
        0.33333        0         0
        0.66667        0         0
        1              0         0
        1              0.33333   0
        1              0.66667   0
        1              1         0
        1              1         0.5
        1              1         1
>> gray(5)
ans =
       0              0         0
       0.25           0.25      0.25
       0.5            0.5       0.5
       0.75           0.75      0.75
       1              1         1
```

上面这个例子显示了两个标准的颜色表。第一个颜色表是一个8个元素的hot颜色表，第二个是一个5个元素的gray颜色表。gray颜色表等值地增加了所有三个颜色组元的值，这样就得到了灰色的不同阴影。

一个颜色表最好还是通过图形来进行可视化浏览。函数pcolor和rgbplot在这种情况下就很有用处，如下例所示：

```
>> n = 21;
>> map = copper(n);
>> colormap(map)
>> subplot(2,1,1)
```

```
>> [xx,yy] = meshgrid(0:n,[0 1]);
>> c = [1:n+1; 1:n+1];
>> pcolor(xx,yy,c)
>> set(gca, 'Yticklabel', '')
>> title('Figure 27.1a: Pcolor of Copper')
>> subplot(2,1,2)
>> rgbplot(map)
>> xlim([0,n])
>> title('Figure 27.1b: RGBplot of Copper')
```

Figure 27.1a　Pcolor of Copper

Figure 27.1b　RGBplot of Copper

这些图形显示了一个铜颜色表——从它的第一行，也就是图上的最左边的一列，到最后一行，也就是图中最右边的一列。函数rgbplot**仅仅用红色，绿色和蓝色画出了颜色表的三列，因此就将各种颜色的三原色形象地剖析开了。**

在生成一个三维图形时候，可以用函数colorbar**将图中的颜色信息以辅助信息的形式在一个颜色条中显示，例如：**

```
>> mesh(peaks)
>> axis tight
>> colorbar
>> title('Figure 27.2: Colorbar Added')
```

Figure 27.2 Colorbar Added

在这个图形中，颜色是和z坐标轴相关的，并且颜色条的颜色表中的颜色与z坐标轴的值形成对应关系。

27.4 颜色表的生成和改变

颜色表是数组的事实意味着用户可以像处理其他数组那样处理它们。函数brighten利用这个特性来调整一个给定的颜色表，以便增加或者减小各个颜色的强度。brighten(beta)将当前的颜色表进行亮化（0≤beta≤1）或者暗化（-1≤beta≤0）处理。brighten(beta)命令执行之后，再执行一个brighten(-beta)就将颜色表恢复到原来的状态。命令newmap=brighten(beta)生成当前的颜色表一个偏亮或者偏暗的版本，而不会改变当前的颜色表。命令mymap=brighten(cmap,beta)生成命令中声明的经过修正之后的颜色表版本，而不会影响到当前的颜色表或者命令中声明的这个颜色表cmap。

用户可以通过生成一个m×3的数组mymap并用colormap(mymap)以安装的方式生成一个颜色表。这个颜色表矩阵的每一个值必须在0和1之间。如果用户试图使用的矩阵其列数不等于3，或者矩阵的任何一个值小于0或大于1，colormap都会报告出一个错误。

颜色表可以用函数rgb2hsv和hsv2rgb实现红－绿－蓝（RGB）标准和色度－饱和度－亮度（HSV）标准之间的转换。Matlab总是将颜色表理解成RGB值。颜色表也可以进行组合，只要所得到的结果满足对矩阵维数和元素值的限制。例如，被调用的颜色表pink可以是：

```
pinkmap = sqrt(2/3*gray + 1/3*hot);
```

在这里，我们再次重申，组合的结果只有在这个m×3矩阵的所有元素都大于等于0，小于等于1的情况下才是一个合法的颜色表。

在正常的情况下，一个颜色表被换算成从用户数据的最小值一直延伸到最大值，也就是说，将用到整个的颜色表来绘制用户的图形。用户可能偶尔会希望改变颜色使用的方式。函数caxis（这个函数名表示“颜色坐标轴”的意思）允许用户用整个颜色表来表示用户数据范围中的一个子集，或者用整个当前颜色表中的一个部分来表示整个数据集。

[cmin,cmax]=caxis返回分别对应于颜色表的第一个和最后一个条目的最小和最大数据值。这些数值通常被设置为用户数据的最小值和最大值。例如，mesh(peaks)生成一个peaks函数的网眼图，并将caxis设置为[-6.5466,8.0752]，分别为z坐标轴的最小和最大值。在这些值之间的数据点所使用的颜色都来自于对颜色表进行插值计算所得到的颜色。

caxis([cmin,cmas])利用整个颜色表来表示位于cmin和cmax之间的数据。大于cmax的数据被绘制成与cmax对应的颜色，而小于cmin的数据点被绘制成与cmin对应的数据。如果cmin小于min(data)或者cmax大于max(data)，那么与cmin或者cmax对应的颜色将永远不会被用到。只有与data对应的颜色表部分才会被用到。caxis('auto')或者其命令形式caxis auto保存cmin和cmax的默认值。下边这个例子显示了颜色坐标轴的设置。

```
>> N = 17;
>> data = [1:N+1;1:N+1]';

>> subplot(1,3,1)
>> colormap(hsv(N))
>> pcolor(data)
>> set(gca,'XtickLabel','')
>> title('Figure 27.3: Auto Limits')
>> caxis auto       % automatic limits (default)

>> subplot(1,3,2)
>> pcolor(data)
>> axis off
>> title('Extended Limits')
>> caxis([-5,N+5]) % extend the color limits

>> subplot(1,3,3)
>> pcolor(data)
>> axis off
>> title('Restricted Limits')
>> caxis([5,N-5]) % restrict the color limits
```

左边的第一个图形是默认的颜色表。这幅图涵盖了整张颜色表。中间这幅图中，颜色坐标轴被扩展了，使得所有被绘制的值都使用的是颜色表的中间子集。右边的这个图形中，颜色坐标被限制了，迫使在图形的中部就涵盖了整个颜色表。这幅图剩下的部分就简单地用颜色表两端的颜色代替了。

a. **自动限制** b. **扩展限制** c. **受限限制**

图 27.3

27.5 用颜色来描述第四维

诸如mesh**和**surf**这样的表面图形根据**z**轴的值来改变颜色，除非给定了一个颜色参数——例如，**surf(X,Y,Z)**等价于**surf(X,Y,Z,Z)**。将颜色应用到**z**轴会得到一幅色彩斑斓的图形，但是无法提供额外的信息，因为**z**轴是已经存在的了。为了更好地利用颜色，我们建议将颜色用来描述数据的某项没有被三个坐标轴所反映的属性。为了做到这一点，需要给三维绘图函数的颜色参数声明不同的数据。**

如果一个绘图函数的颜色参数是一个向量或者矩阵，那么它将会被换算并用来作为指向颜色表的索引。这个参数可以是任何实值的向量或者与其他参数维数相同的矩阵。请看下边这个例子：

```
>> x = -7.5:.5:7.5;              % data
>> [X Y] = meshgrid(x);          % create plaid data
>> R = sqrt(X.^2 + Y.^2)+eps;    % create sombrero
>> Z = sin(R)./R;
>> subplot(2,2,1)
>> surf(X,Y,Z,Z)         % default color order
>> colormap(gray)
>> shading interp
>> axis tight off
>> title('Figure 27.4a: Default,Z')
```

```
>> subplot(2,2,2)
>> surf(X,Y,Z,Y)        % Y axis color order
>> shading interp
>> axis tight off
>> title('Figure 27.4b: Y axis')

>> subplot(2,2,3)
>> surf(X,Y,Z,X-Y)  % diagonal color order
>> shading interp
>> axis tight off
>> title('Figure 27.4c: X - Y')

>> subplot(2,2,4)
>> surf(X,Y,Z,R)        % radius color order
>> shading interp
>> axis tight off
>> title('Radius')
```

Figure274a　Default,Z　　Figure 27.4b　Y axis

Figure 27.4c　X-Y　　Figure 27.4d　Radius

这几张图显示了将颜色作为第四维的4个简单的方法。被作为第4个参数提供给surf的任何数据都被用来对颜色表进行插值。前3个参数的任何函数都可以作为第4个参数，或者是某个完全独立的变量也行。利用函数del2和gradient，用户可以用颜色来分别表示曲率和斜率，例如：

```
>> subplot(2,2,1)
>> surf(X,Y,Z,abs(del2(Z)))         % absolute Laplacian
>> colormap(gray)
>> shading interp
>> axis tight off
>> title('Figure 27.5a: |Curvature|')

>> subplot(2,2,2)
>> [dZdx,dZdy] = gradient(Z);   % compute gradient of surface
```

```
>> Surf(X,Y,Z,abs(dZdx))        % absolute slope in x-direction
>> shading interp
>> axis tight off
>> title('Figure 27.5b: |dZ/dx|')

>> subplot(2,2,3)
>> surf(X,Y,Z,abs(dZdy))        % absolute slope in y-direction
>> shading interp
>> axis tight off
>> title('Figure 27.5c: |dZ/dy|')

>>subplot(2,2,4)
>> dR = sqrt(dZdx.^2 + dZdy.^2);
>> surf(X,Y,Z,abs(dR))          % absolute slope in radius
>> shading interp
>> axis tight off
>> title('Figure 27.5d: |dR|')
```

Figure27.5a |Curvature| Figure27.5b |dZ/dx|

Figure27.5c |dX/dy| Figure27.5d |dR|

请注意在这些图中的颜色是怎样给绘制好的表面提供额外的一维的。函数del2**是离散拉普拉斯函数，它根据表面的曲率来应用色彩。函数**gradient**对表面关于两个坐标轴方向的梯度或者斜率进行近似。**

27.6 光 照 模 式

前一章中讨论了图形函数pcolor**，**fill**，**fill3**，**mesh**和**surf**，它们所绘制的图形对象都从各个角度被非常发散的光线进行了良好照明。这种处理方法侧重于在“图形”窗口中展示对象的特点，并且增强了用户可视地展示所分析数据的能力。尽管通过这种方式可以非常清晰地展示数据，但是我们还可以通过生成不同的光照效果来增强或者削弱显示的实际效果。**

函数shading**选择小平面（**faceted**），平面或者插值投影。每一个这样的投影示例都可以从关于三维图形的那一章找到。尽管需要更强大的计算功能和多得多的时间来显示，在一幅图像中对对象进行插值投影确实是可以提高所显示图像的视觉效果。**

可以添加一个或者多个光源来模仿光照和与直接光照相对应的阴影。函数light**生成一**

个沿着射线[1 0 1]到无穷远的光源。一旦生成了光源，函数lighting就允许用户从四个不同的光照模式中选择一种：none（忽略任何光源），flat（光源生成以后的默认模式），phong以及gouraud。这些模式中的每一个都用到了不同的算法来改变物体的外观。flat光照给物体的每一个面都使用了统一的颜色；Gouraud光照根据顶点的颜色对表面颜色进行插值；Phong光照对每个表面的顶点的法线进行插值，并且计算在每个坐标轴的反光。正如颜色表是“图形”窗口的属性一样，光线也是坐标轴的属性或者说是产物。因此，“图形”窗口中的每个坐标轴都可以单独地设置光照。请考察下边这个例子：

```
>> subplot(2,2,1)
>> sphere
>> light
>> shading interp
>> axis square off
>> lighting none
>> title('Figure 27.6a: No Lighting')

>> subplot(2,2,2)
>> sphere
>> light
>> shading interp
>> axis square off
>> lighting flat
>> title('Figure 27.6b: Flat Lighting')

>> subplot(2,2,3)
>> sphere
>> light
>> shading interp
>> axis square off
>> lighting gouraud
>> title('Figure 27.6c: Gouraud Lighting')

>> subplot(2,2,4)
>> sphere
>> light
>> shading interp
>> axis square off
>> lighting phong
>> title('Figure 27.6d: Phong Lighting')
```

Figure 27.6a No Lighting

Figure 27.6b Flat Lighting

Figure 27.6c Gouraud Lighting

Figure 27.6d Phong Lighting

除了光照之外，在轴上的物体的外观可以通过调整其外观上的反射特性或者表面“反射系数”来改变。反射系数是由一些组元构成的：

- 环境光线——图中统一漫反射光线的强度
- 散射反射——弱漫反射光线强度
- 镜面反射——强漫反射光线强度
- 镜面指数——控制镜面“热点”大小或者扩展
- 镜面颜色反射－确定表面颜色对反射的贡献率

利用函数material，可以获得一些预先定义的表面反射属性。选项包括shiny，dull，metal和default，default用来存储默认的表面反射属性。如material([ka kd ks n sc])这样的函数调用格式（其中n和sc都是可选的），它设置坐标轴上物体的环境光线强度，散射反射，镜面反射，镜面指数，以及镜面颜色反射，例如：

```
>> subplot(2,2,1)z
>> sphere
>> colormap(gray)
>> light
>> shading interp
>> axis square off
>> material default
>> title('Figure 27.7a: Default Material')

>> subplot(2,2,2)
>> sphere
>> light
>> shading interp
>> axis square off
>> material shiny
>> title('Figure 27.7b: Shiny Material')

>> subplot(2,2,3)
>> sphere
>> light
>> shading interp
>> axis square off
>> material dull
>> title('Figure 27.7c: Dull Material')
```

```
>> subplot(2,2,4)
>> sphere
>> light
>> shading interp
>> axis square off
>> material metal
>> title('Figure 27.7d: Metal Material')
```

Figure 27.7a Default Material　　Figure 27.7b Shiny Material

Figure 27.7c Dull Material　　Figure 27.7d metal Material

正如上边所显示的那样，函数light的功能是相当有限的。它生成从一个无穷远的距离沿着一个指定的方向发出的白光。实际上，light是一个句柄图形对象生成函数。句柄图形函数将在第30章中讨论。函数light提供了用户可以设置的不同属性。例如，光线的颜色、位置以及类型都可以设置，其中，类型意味着光线可以是一个指定位置的点光源，也可以是沿着某条射线的无穷远处的光源：

```
>> Hl = light('Position',[x,y,z], 'Color',[r,g,b], 'Style', 'local');
```

这条命令生成了一个位于（x,y,z）的光源，其光的颜色为[r,g,b]，并且声明了光源的位置是一个点（'local'）而不是一个指向无穷远的射线（'infinite'）。它还将光源对象（H1）保存起来，这使得用户可以在以后的使用中用它来改变光源的属性。例如：

```
>> set (Hl, 'Position',[1 0 1], 'Color',[1 1 1], 'Style', 'infinite');
```

这条命令将用句柄H1定义的光源恢复成它原来的默认设置。关于句柄图形的更多的信息，请参见第30章。

27.7 小　结

下面这个表格列出了颜色和光照的Matlab函数。

函数	描述
light	光照对象生成函数
lighting	设置光照模式（flat,gouraud,phong或者none）
lightangle	球坐标中的位置光照对象
material	设置反射的物质类型（default,shiny,dull或者metal）
camlight	设置与摄像头相关的光照对象
brighten	亮化或者暗化颜色表
caxis	设置或获取颜色轴的限制
diffuse	找出表面散射反射
specular	找出表面镜面反射
surfnorm	计算表面法线
colorbar	生成颜色条
colordef	定义默认颜色属性
colormap	设置或获取图形窗口颜色表
hsv2rgb	将色度－饱和度－亮度颜色值转换为红－绿－蓝模式值
rgb2hsv	将红－绿－蓝颜色值转换为色度－饱和度－亮度模式值
rgbplot	绘制颜色表
shading	阴影模式（flat,faceted或者interp）
spinmap	使颜色旋转
whitebg	将图形窗口设置成白色背景
graymon	将图形窗口设置成灰度默认值，以便在单色显示器中显示
autumn	带红色和黄色阴影的颜色表
bone	带有蓝色的灰度颜色表
cool	天蓝粉色基本颜色表
copper	线性铜色调颜色表
flag	红，白，蓝，黑基色颜色表
gray	线性灰度颜色表
hot	黑，红，黄，白基色颜色表
hsv	色度、饱和度、亮度（HSV）颜色表
jet	HSV颜色表的变种（开始是蓝色，结尾是红色）
lines	基于线颜色的颜色表
prism	带交替的红、桔、黄、绿、蓝和紫色的颜色表
spring	带洋红和黄色阴影的颜色表
summer	带绿色和黄色阴影的颜色表
winter	带蓝色和绿色阴影的颜色表

第 28 章　图像、影片和声音

Matlab提供了显示几种类型图像的命令。用户可以将图像创建或者保存成标准的双精度浮点数（double），也可以创建或者保存为8位（unit8）或者16位（unit16）无符号整型。Matlab可以读写多种标准图形文件格式的图形文件，也可以用load和save来将图像数据保存在MAT文件中。Matlab还提供了以影片（帧序列）的方式创建和播放动画的命令。Matlab也提供了声音函数，只要您的计算机具有声音处理功能，就能够使用这些函数。

28.1　图　像

Matlab中的图像是由一个数据矩阵（并且通常还有一个相关的颜色表矩阵）构成的。有三种类型的图像数据矩阵，它们的Matlab解释的方式各不相同：被索引的图像、亮度图像和真彩色即RGB图像。

索引图像需要一个颜色表，并且将图像数据解释成指向颜色表矩阵的索引号。这个颜色表矩阵是一个标准的颜色表：任何包含了有效RGB数据的m×3矩阵。给定一个图像数据数组X(i,j)和一个颜色表数组cmap，每个图像像素$P_{i,j}$的颜色就是cmap(X(i,j),:)。这意味着X中的数据值是在[1 length(cmap)]范围之内的整数。可以用下边这个命令显示这幅图像：

```
>> image(X); colormap(cmap)
```

亮度图像在一个亮度范围内对图像数据进行处理。这种形式通常用于显示在灰度或者其他某种单色颜色表中的图像，但是如果需要，也可以使用其他颜色表。图像数据没有要求一定要像在被索引的图像中那样在[1 length(cmap)]范围之内。数据是在一个指定的范围内进行标定的，并且结果被用来作为指向颜色表的索引。例如：

```
>> imagesc(X,[0 1]); colormap(gray)
```

这个命令将值0与颜色表的第一个条目关联起来，并且将值1和颜色表的最后一个条目关联起来。X在0和1之间的值就被标定在0和1的范围之内，并且被用来作为指向颜色表的索引。如果这个刻度[0 1]被省略了，那么它的默认值就是[min(min(X))　max(max(X))]。

真彩色即RGB图像是利用一个包含了有效RGB值的m×n×3数组创建的。行维和列维声明了像素的位置，页维或者说是第三维声明了每一个颜色组元。例如，像素$P_{i,j}$被画成X(i,j,:)所声明的颜色。这种图像不需要颜色表，因为颜色数据已经被保存在图像数据数组自身里边了。如果计算机硬件不支持真彩色图像（例如，它只有一块8位显卡），那么Matlab就利用颜色近似和抖动来显示图像。例如：

```
>> image(X)
```

其中，X是一个m×n×3的真彩色或者RGB图像。这条命令显示了图像。X可以包含

unit8，unit16或者double数据。

如果图像显示在默认的坐标轴上，那么屏幕高宽比通常就会是不正确的，这样就会造成图像的扭曲变形。输入如下的命令：

```
>> axis image off
```

就对坐标轴属性进行了设置，使得屏幕高宽比与图像相匹配，并且将坐标轴和记号隐藏。为了强制使得图像中的每个像素占据显示器中的一个像素点，需要对figure和axes属性进行设置，如下例所示：

```
>> load clown               % sample image
>> [r,c] = size(X);         % pixel dimensions
>> figure('Units', 'Pixels', 'Position',[100 100 c r])
>> image(X)
>> set(gca, 'Position',[0 0 1 1])
>> colormap(map)
```

这里，通过将figure的宽度和高度设置成等于图像的宽和高，figure被设置显示出和图像完全相同的像素数量。然后，axes位置被设定为以标准化单位占据整个的figure。

除了上边使用的clown.mat之外，Matlab安装程序中还有一些示例图像。Matlab的demos子目录下有cape.mat，clown.mat，detail.mat，durer.mat，flujet.mat，gatlin.mat，mandrill.mat和spine.mat。这些图像中的每一个都可以通过输入下面的指令来显示。

```
>> load filename
>> image(X), colormap(map)
>> title(caption)
>> axis image off
```

28.2 图 像 格 式

在Matlab中，默认的数值型数据类型为double。这指的是双精度，64位浮点数。Matlab对其他数据格式，比如图像的16位字符数据类型（unit16）和8位无符号整型（unit8），提供了有限的支持。

命令image和imagesc可以显示8位和16位图像，而不用预先把它们转换成double型。但是，unit8数据值的范围是[0 255]，这是在标准图形文件格式中支持的数据格式，unit16的数据值的范围是[0 65535]。

对于被索引的索引图像，image通过自动地提供相应的偏移量将值0映射到有256个条目的颜色表的第一个条目，而将值255映射到最后的一个条目。因为被索引图像的double数据的正常范围为[1 length(cmap)]，因此在unit8和double6或者unit16和double之间的转换需要将数据值平移1。另外，对unit8数组的数学运算还没有定义。因此，为了对无符号整型数执行数学运算，它们必须被转换成double格式，例如：

```
>> Xdouble = double(Xuint8) + 1;
>> Xuint8 = uint8(Xdouble - 1);
```

这些命令将Xunit8中的unit8数据转换成double，然后再转换回来。对于8位亮度和RGB图像而言，数据值的范围通常是[0 255]，而不是[0 1]。为了显示8位强度和RGB图像，需要使用下面的命令。

```
>> imagesc(Xuint8,[0 255]); colormap(cmap)
>> image(Xuint8)
```

向double的转换也可以进行标准化，例如：

```
>> Xdouble = double(Xuint8)/255;
>> Xuint8 = uint8(round(Xdouble*255));
```

RGB图像中的8位颜色数据在显示的时候被自动地标定。例如，在使用双精度数的时候，白色通常是[1 1 1]。如果相同的颜色储存在8位数据中，白色就被表示为[255 255 255]。

尽管unit8数组的数学运算没有定义，但是对这些数据的文件读写操作却是有定义的，因此imread，imwrite，save和load都支持unit8数据类型。标准的Matlab索引和下标，以及reshape，cat，permute，max，min和find函数也支持对这种数据类型的操作，[ ]和“ ' ”操作符也是如此。其次为了执行其他任何数学运算，首先考虑用上边建议的转换函数将数据转换成双精度格式。其次用户也可以生成对unit8数据类型操作的自定义方法。Matlab中提供的可选图像处理工具箱包含了很多图像处理函数。如果用户需要经常处理图像，这个工具箱是很有用处的。

28.3　图 像 文 件

可以用多种不同的文件格式来将图像数据储存在文件中和重新载入到Matlab中。通常的Matlab函数save和load支持double，unit8或者unit16格式的图像数据，就像这些函数支持其他Matlab变量和数据类型一样。当保存非标准颜色表的被索引图像或者强度图像的时候，一定要保存颜色表及所显示的图像数据，例如：

```
>> save myimage.mat X map
```

Matlab还利用函数imread和imwrite支持多种工业标准的图像文件格式。可以用函数imfinfo获得关于图形文件内容的信息。Imread的帮助文档，给出了关于图像读取格式和特性的广泛信息，下面显示了其中的一部分。

```
>> help imread
 IMREAD Read image from graphics file.
     A = IMREAD(FILENAME,FMT) reads the image in FILENAME into
     A. If the file contains a grayscale intensity image, A is
     a two-dimensional array. If the file contains a truecolor
     (RGB) image, A is a three-dimensional (M-by-N-by-3) array.
     FILENAME is a string that specifies the name of the
     graphics file, and FMT is a string that specifies the
     format of the file. The file must be in the current
     directory or in a directory on the MATLAB path. if IMREAD
```

```
    cannot find a file named FILENAME, it looks for a file
    named FILENAME.FMT.

    The possible values for FMT include:
    'jpg' or 'jpeg' Joint Photographic Experts Group (JPEG)
    'tif' or 'tiff' Tagged Image File Format (TIFF)
    'gif'           Graphics Interchange Format (GIF)
    'bmp'           Windows Bitmap (BMP)
    'png'           Portable Network Graphics (PNG)
    'hdf'           Hierarchical Data Format (HDF)
    'pcx'           Windows Paintbrush (PCX)
    'xwd'           X Window Dump (XWD)
    'cur'           Windows Cursor resources (CUR)
    'ico'           Windows Icon resources (ICO)

[X,MAP] = IMREAD(FILENAME,FMT) reads the indexed image in
FILENAME into X and its associated colormap into MAP.
Colormap values in the image file are automatically
rescaled into the range [0,1].

[...] = IMREAD(FILENAME) attempts to infer the format of the
file from its content.

Data types
--------------------
In most of the image file formats supported by IMREAD,
pixels are stored using 8 or fewer bits per color plane.
When reading such a file, the class of the output (A or X)
is uint8. IMREAD also supports reading 16-bit-per-pixel
data from TIFF and PNG files; for such image files, the
class of the output (A or X) is uint16.

TIFF-specific syntaxes
---------------------
[...] = IMREAD(  ...,  IDX) reads in one image from a
multi-image TIFF file. IDX is an integer value that
specifies the order that the image appears in the file.
For example, if IDX is 3, IMREAD reads the third image in
the file. If you omit this argument, IMREAD reads the
first image in the file.
```

调用imwrite**的语法根据图像的类型和文件格式的变化而变化。**imwrite**的帮助文档给出了关于图像保存格式和特性的广泛信息，下面显示了其中的一部分。**

```
>> help imwrite
 IMWRITE Write image to graphics file.
     IMWRITE(A,FILENAME,FMT) writes the image A to FILENAME.
     FILENAME is a string that specifies the name of the output
     file, and FMT is a string the specifies the format of the
     file. A can be either a grayscale image (M-by-N) or a
     truecolor image (M-by-N-by-3).

     The possible values for FMT include:
    'tif' or 'tiff' Tagged Image File Format (TIFF)
```

```
     'jpg' or 'jpeg' Joint Photographic Experts Group (JPEG)
     'bmp'           Windows Bitmap (BMP)
     'png'           Portable Network Graphics (PNG)
     'hdf'           Hierarchical Data Format (HDF)
     'pcx'           Windows Paintbrush (PCX)
     'xwd'           X Window Dump (XWD)

IMWRITE(X,MAP,FILENAME,FMT) writes the indexed image in X,
and its associated colormap MAP, to FILENAME. If X is of
class uint8 or uint16, IMWRITE writes the actual values in
the array to the file. If X is of class double, IMWRITE
offsets the values in the array before writing, using
uint8(X-1). MAP must be a valid MATLAB colormap. Note
that most image file formats do not support colormaps with
more than 256 entries.

IMWRITE(  ...,  FILENAME) writes the image to FILENAME,
inferring the format to use from the filename's extension.
The extension must be one of the legal values for FMT.

IMWRITE(  ...,  PARAM1,VAL1,PARAM2,VAL2,...  ) specifies
parameters that control various characteristics of the
output file. Parameters are currently supported for HDF,
JPEG, TIFF, and PNG files.

Data types
---------------
Most of the supported image file formats store uint8
data. PNG and TIFF additionally support uint16 data. For
grayscale and RGB images, if the data array is double, the
assumed dynamic range is [0,1]. The data array is
automatically scaled by 255 before being written out as
uint8. If the data array is uint8 or uintl6, then it is
written out without scaling as uint8 or uint16,
respectively. NOTE: If a logical double or uint8 is
written to a PNG or TIFF file, it is assumed to be a
binary image and will be written with a bitdepth of 1.

For indexed images, if the index array is double, then the
indices are first converted to zero-based indices by
subtracting 1 from each element, and then they are written
out as uint8. If the index array is uint8 or uintl6,
then it is written out without modification as uint8 or
uint16, respectively.

JPEG-specific parameters
   ---------------------
 'Quality'      A number between 0 and 100; higher numbers
                mean quality is better (less image degradation
                due to compression), but the resulting file
                size is larger

TIFF-specific parameters
```

```
 ------------------
 Compression'  One of these strings:  'none'   'packbits'
               (default for nonbinary images),  'ccitt'
               (default for binary images), 'fax3',  'fax4';
               'ccitt', 'fax3', and 'fax4' are valid for
               binary images only
'Description'  Any string; fills in the ImageDescription
               field returned by IMFINFO
'Resolution'   A two-element vector containing the
               XResolution and YResolution, or a scalar
               indicating both resolutions; the default value
               is 72
'WriteMode'    One of these strings:  'overwrite' (the
               default) or 'append'
```

28.4 影 片

Matlab中的动画采用了两种形式。一种形式是，如果生成一个图像序列所需要的计算足够快，那么就可以设置figure和axes属性，使得屏幕绘制以足够快的速度进行，这样动画从视觉上看起来就是平稳的。另一种形式是，如果计算需要大量的时间，或者结果得到的图像过于复杂，用户就必须生成一个影片。

在Matlab中，函数getframe和movie提供了捕获和演示影片所需要的工具。函数getframe对当前的图像进行一次快照，movie在这些快照都被捕获之后，回头重新播放这些帧序列。函数getframe的输出是一个结构，它包含了movie所需要的所有信息。捕获多个帧的过程仅仅就是向这个结构里边添加元素的过程。请看下边这个例子：

```
% movie-making example: rotate a 3-D surface plot
[X,Y,Z]=peaks(50);          % create data
surfl(X,Y,Z)                % plot surface with lighting
axis([-3 3 -3 3 -10 10]     % fix axes so that scaling does not change
axis vis3d off              % fix axes for 3D and turn off axes ticks etc.
shading interp              % make it pretty with interpolated shading
colormap(copper)            % choose a good colormap for lighting
for i=1:15                  % rotate and capture each frame
   view(-37.5+15*(i-1),30)  % change the viewpoint for this frame
   m(i)=getframe;           % add this figure to the frame structure
end
cla         % clear axis for movie
movie(m)    % play the movie
```

通过不断地旋转顶峰的表面并且每旋转一次就捕获一个帧，上边这个脚本文件生成了一个影片。最后，在清除axes之后，播放了这个影片。变量m包含了一个结构数组，其每个数组元素包含了一个帧，例如：

```
>> m
m =
1×15 struct array with fields:
```

```
    cdata
    colormap

>> size(m(1).cdata)
ans =
    412     369     3
```

保存了图像cdata的颜色数据是一个真彩色或者RGB位图图像数据。因此，axes内容的复杂性并没有影响存储一个影片所需要的字节数。用像素表示的axes的大小决定了图像的大小，因此也就决定了储存一个影片所需要的字节数量。

28.5　图 像 工 具

可以用函数im2frame和frame2im实现被索引图像和影片之间的转换。例如：

```
>> [X,cmap] = frame2im(M(n))
```

这条命令将影片矩阵M的第n帧转换成一个被索引图像X和相关的颜色表cmap。类似地：

```
>> M(n) = im2frame(X,cmap)
```

这条命令将被索引图像X和颜色表cmap转换成影片矩阵M的第n帧。请注意，im2frame可以用来将一系列图像转换成一个影片，其方式和getframe将一系列figure或者axes转换成一个影片的方式。

28.6　声　音

Matlab用/dev/audio设备支持PC平台和任何UNIX平台上的声音。sound(y,f,b)将向量y中的信号以采样频率f发送到计算机的扬声器中。变量y中超出了[-1 1]范围之外的值被省略。如果f被省略，就使用默认的采样频率8192Hz。如果有可能，Matlab用b位/秒播放这个声音。大多数的平台都支持b=8或者b=16。如果b被省略，就使用b=16。

函数soundsc和sound基本相同，只是其向量y中的值都被标定在范围[-1 1]之内，而不是把超出这个范围的值省略。这使得声音可以尽可能的大，而不用将过大的声音省略。还可以用一个额外的参数来使用户将y值的某个范围和整个声音范围对应起来。其格式为soundsc(y,…,[smin smax])。如果这个参数被省略了，默认的范围就是[min(y) max(y)]。

在Matlab中，还提供了两个工业标准的声音文件格式。Matlab可以对NeXT/Sun音频格式（file.au）文件和Microsoft WAVE格式(file.wav)文件进行读写操作。

NeXT/Sun音频声音存储格式支持8位μ法则、8位线性和16位线性格式的多通道数据。auwrite的最常用调用格式是auwrite(y,f,n, 'method', 'filename')，其中y是采样数据，f是用赫兹表示的采样频率，b表示在编码器中的位数，'method'是一个表示编码方法的字符串，'filename'是一个表示输出文件名的字符串。y的每一列代表了一个不同的通道。y中任何超

出了范围[-1 1]的值在写入文件之前就被忽略了。参数f，n和'method'都是可选的。如果这些参数被省略了，那么就默认地将这些参数设置为f=8000，n=8，'method'='mu'。参数'method'必须是'linear'或者'mu'。如果文件名字符串没有包含后缀，Matlab就自动地给它加上'.au'的后缀。

μ法则和线性格式之间的转换可以利用函数mu2lin和lin2mu来进行。关于这两个函数所涉及到的确切的转换过程的信息请参见联机帮助文档。

多通道8位或者16位WAVE声音存储格式声音文件可以用wavwrite函数来生成。其最常用的调用格式为wavwrite(y,f,n, 'filename')，其中y是采样数据，f是以赫兹为单位的采样频率，b声明了在编码器中的位数，'filename'是一个声明了输出文件的字符串。y的每一列都代表了一个单独的通道。y中任何超出了范围[-1 1]的值在写入文件之前都被忽略了。参数f和n是可选的。如果这些参数被省略了，Matlab就使用其默认值f=8000和n=16。如果文件名字符串没有带后缀名，那么Matlab就会自动给它加上'.wav'的后缀。

auread 和 wavread 都有相同的调用语法和选项。最常用的调用格式为[y,f,b]=auread('filename',n)，这条语句载入由字符串'filename'声示的声音文件，并将采样数据返回给y。如果这个'filename'字符串没有给出后缀（.au或者.wav），那么Matlab就将相应的后缀名添加到文件名后边。y中的数据值都在范围[-1 1]之内。如果需要输出如上所示的三个输出参数，那么就在f和b中分别返回以赫兹为单位的采样频率和每个采样的位数。如果给出了参数n，那么就只返回文件中每个通道的前n个采样。如果n=[n1 n2]，那么只返回每个通道从第n1个到第n2个之间的采样。形如[samples,channels]=wavread('filename', 'size')的调用格式将返回文件中音频数据的数量大小，而不是数据本身。这种调用格式对于预先分配存储空间或者估计资源使用量来说是很有用处的。

28.7 小 结

下表对Matlab中所提供的图像、影片和声音功能进行了总结。

函数	描述
image	生成被索引的或者真彩色(RGB)图像对象
imagesc	生成亮度图像对象
colormap	将颜色表应用到图像
axis image	调整图像的坐标轴刻度
unit8	转换为无符号8位整型
uint16	转换为无符号16位整型
double	转换为双精度数
imread	读取图像文件
imwrite	写图像文件
imfinfo	图像文件信息
getframe	将影片帧放在结构中
movie	从影片结构中播放影片

续表

函数	描述
frame2im	将影片帧转换成图像
im2frame	将图像转换成影片帧
avifile	生成avi影片文件
addframe	将帧添加到avi影片文件中
close	关闭avi影片文件
aviread	读取avi影片文件
aviinfo	关于一个avi影片文件的信息
movie2avi	将Matlab格式的影片转换为avi格式
sound	将向量以声音的形式播放
soundsc	将向量进行自动标定并以声音的形式播放
wavplay	播放WAVE格式的声音文件
wavrecord	利用Windows的音频输入设备记录声音
wavread	读取WAVE格式的声音文件
wavwrite	写WAVE格式的声音文件
auread	读取NeXT/SUN声音文件
auwrite	写NeXT/SUN声音文件
lin2mu	将线性音频转换为 μ 法则音频
mu2lin	将 μ 法则音频转换为线性音频

第 29 章　打印和导出图形

Matlab图形是进行数据可视化和数据分析的非常有效的工具。因此，用户经常会希望生成硬拷贝输出或者将这些图形应用在其他应用程序中。Matlab提供了一个非常灵活的系统来打印图形和生成多种不同图形格式的输出，这些图形格式包括EPS和TIFF。大多数应用程序都可以导入一种或者多种被Matlab支持的图形文件格式。

也许在打印或者导出图形的时候需要认识的最重要的事情就是：图形是在绘制或导出的过程中被重新绘制的。也就是说，用户在屏幕上看见的与用户在打印在纸上的或者在导出文件中所得到的不同。默认地，Matlab可以选择不同的绘制器（painter，zbuffer或者OpenGL），可以改变axes的标记符号的分布，并且可以改变被打印和导出的“图形”的大小。Matlab还提供了所见即所得（WYSIWYG）的功能，但是这并不是默认的功能。

总的来说，打印和导出并不是一件很简单的事情，因为涉及到打印机驱动程序，打印机协议，图形文件类型，绘图器，位图－向量图形描述，dpi选择，颜色－黑和白，压缩，平台限制等等，几乎有不计其数的组合。在默认输出时复杂的程序有很大简化。但是在某些情况下，就必须理解打印和导出的复杂性。在这样的情况下，花费大量的时间来调整输出就是一件很常见的事情了。

本章介绍了Matlab中的打印和导出功能。就像在其他领域中一样，Matlab提供了菜单栏选项和“命令”窗口函数来进行打印和导出。菜单方法提供了很大的方便，但是在灵活性上有所欠缺，而函数方法灵活性很强，但是需要用户掌握很多的知识。菜单方法允许用户设置当前图形或者是当前对话期的特性，但是函数方法提供了更多的功能，使得用户可以设置在各个Matlab对话期中都可以用的默认特性。

> 输入命令>>doc print可以访问函数print的在线文档, 它提供了关于打印和导出的大量信息

29.1　用菜单打印和导出

当图形在一个“图形”窗口中显示的时候，这个窗口的顶部包含了一个菜单栏，还可能有一个或者多个工具栏。菜单栏上有一些菜单，包括File，Edit，View，Insert，Tools，Window和Help。在这些菜单中，File菜单列出了打印和导出当前“图形”的菜单项。另外，PC系统上的Edit菜单包含了将当前“图形”导出到系统剪贴板的菜单项。File菜单包括了Export，Page Setup，Print Setup，Print Preview和Print这些菜单项。这些菜单项中的每一项都可用来设置打印或导出。

Export菜单项可以将当前的“图形”保存为多种图形格式中的一种。Export对话框显示了一个Save对话框，在这个对话框中，用户需要为当前这个图形选择目录，文件名和文件类型。被保存“图形”的大小和特性是由在Page Setup对话框中所选择的设置决定的。所提

供的文件类型选项使得用户可以为不同的文件类型属性设置默认的导出文件类型。“命令”窗口的导出函数提供了对导出“图形”的更加全面的控制。

选择Page Setup菜单项（或者pagesetupdlg命令）打开了一个带制表符的对话框，这个对话框可以设置当“图形”被打印或者导出的时候将要用到的一些特性。这个对话框是指定大小，方向，布局以及这个被导出的“图形”很多特性的主要场所。

在PC系统中选择Print Setup菜单项（或者print –dsetup命令）打开了一个标准的对话框，这个对话框用来设定一个打印机以及诸如纸张大小，纸张来源和打印方向这样的选项。在Unix系统中，Print Setup菜单项（或者printdlg –setup）打开一个和Print对话框类似的对话框，这两个对话框的差别在于两者的对话框标题不同，并且在Unix中，点击这个对话框中的OK按钮关闭这个对话框，但是并不把“图形”发送到打印机或者文件。UNIX的Print Setup对话框包括了选择打印机、文件名、打印机驱动、图形大小、坐标轴和标记等众多选项，还有一个Option按钮，按下这个按钮，就可以弹出一个设置很多其他更多选项的对话框，这些选项包括选择一个绘图器和设置输出分辨率。在这两个平台中，这个对话框中选择的选项在这个Matlab对话期中保持有效，并且将Page Setup对话框中设置的相关选项覆盖。

Print Preview菜单项（或者printpreview命令）可打开一个窗口，这个窗口显示了图形打印的效果。这个窗口提供了一些按钮来调用Page Setup对话框进行任何必要的改动，或调用Print对话框来打印“图形”。

Print菜单项（或者printdlg命令）在PC系统中打开一个标准的Print对话框。从这个对话框中，用户可以选择打印机和打印的拷贝数。在这个对话框中也可以选择一个打印文件选项。在Unix系统中，可以选择打印机、文件名、打印机驱动程序，以及冻结坐标轴和刻度线选项。还提供了一个Options按钮，按下这个按钮就弹出一个对话框，在这个对话框中可以设置一些附加选项，所有这些附加选项也会出现在Page Setup对话框中。

> Page Setup选项应用于当前的图形，并且和当前图形一起保存。Print Setup选项在当前这个Matlab对话期中保持不变，并且将当前图形的Page Setup中的任何相关的选项覆盖。Print选项只应用于当前图形并且将当前图形的Page Setup或者Print Setup中的相关选项设置覆盖。

这些File菜单项对于大多数打印和导出任务而言已经足够了。如果想要进一步控制输出或者进行默认设置，就需要用到“命令”窗口函数。

在PC系统中，Edit菜单提供了利用系统的剪贴板进行导出的功能。Copy Figure菜单项根据由选择的Copy Option菜单项所打开的参考对话框中的选项设置将当前的图形放到剪贴板上。当前的图形可以被拷贝成位图（BMP）格式或者增强型图文文件（EMF）格式。可以用一个模板来改变线条的宽度和字体大小，也可以用模板来设置拷贝操作的其他默认选项。这些选项只在用Copy Figure菜单项将一个“图形”拷贝到剪贴板的时候生效。它们在将一个“图形”打印或者导出到一个图形文件的时候无效。

29.2 命令行打印和导出

函数print在“命令”窗口中处理所有的打印和导出任务。这个函数提供了多个选项，用来指定附加的输入参数。print的命令形式的调用语法为：

```
>> print –device –option –option filename
```

其中，所有的参数都是可选的，-device指定要使用的设备驱动程序，-option指定一个或者多个选项，filename如果输出不是直接送到打印机的情况下指定输出文件名。因为命令-函数的双重性，print也可以当作函数进行调用。例如：

```
>> print('-device', '-option', '-option', 'filename')
```

这条语句等价于前面的print命令语句。

通常，包含了-device的打印选项可以按任何顺序进行指定。下表描述了识别print的-option字符串。

-option	描述
-adobecset	选择PostScript默认字符集编码（只适用于早期的PostScript打印驱动程序和EPS文件格式）
-append	将图形附加在现存的文件后边（只适用于PostScript打印驱动程序）
-cmyk	用CMYK颜色而不是用RGB颜色打印（只适用于PostScript打印驱动程序和EPS文件格式）
-device	指定使用的打印驱动程序
-dsetup	显示Print Setup对话框（只适用于Windows系统）
-fhandle	指定要打印或者导出的图形的数值句柄
-loose	使用loose的PostScript边框（只适用于PostScript，EPS和GhostScript）
-noui	抑制uicontrol对象的打印
-opengl	用OpenGL算法绘图（位图格式）
-painters	用Painter的算法绘图（向量格式）
-Pprinter	指定要使用的打印机的名字（只适用于对Unix）
-rnumber	以dpi指定分辨率。对于大多数的打印设备均可设置；对于除了EMF和ILL之外的内置Matlab函数均可设置；对于很多GhostScript导出格式不可设置。Z缓冲和OpenGL绘图算法的默认导出分辨率为150dpi，Painter的绘图算法的分辨率为864dpi。
-swindowtitle	指定用来打印和导出的SIMULINK系统窗口的名字
-v	冗余。显示Print对话框（只适用于Windows）
-zbuffer	用Z缓冲算法绘图（位图格式）

29.3　打印机和导出文件格式

函数print支持多种输出设备（打印机和多种文件类型）。下表列出了支持的打印机型号。这些打印机中的很多都通过使用GhostScript打印机驱动程序得到了Matlab的支持，这个打印机驱动程序将PostScript打印机代码转换成本地打印机代码。这个转换过程对用户而言是透明的，但是将打印的字体限定在了PostScript所支持的字体范围之内。

-device	描述
-dps	PostScript第一层，黑白打印机，包括灰度打印机
-dpsc	PostScript第一层，彩色打印机
-dps2	PostScript第二层，黑白打印机，包括灰度打印机
-dpsc2	PostScript第二层，彩色打印机
-dwin	黑白打印，包括灰度打印（只适用于Windows）
-dwinc	彩色打印（只适用于Windows）
-dbj10e	佳能Bubblejet BJ10e（使用GhostScript）
-dbj200	佳能Bubblejet BJ200（使用GhostScript）
-dbjc600	佳能Bubblejet 600/4000/70（使用GhostScript）
-dln03	DEC LN03（使用GhostScript）
-depson	Epson和兼容型9或24针点阵打印机（使用GhostScript）
-deps9high	Epson和兼容型9针高清晰度打印机（使用GhostScript）
-depsonc	Epson LQ-2550和兼容型打印机，包括富士通3400/2400/1200（使用GhostScript）
-ddnj650c	HP DesignJet 650C（使用GhostScript）
-ddjet500	HP Deskjet 500（使用GhostScript）
-dcdjmono	HP Deskjet 500C单色打印（使用GhostScript）
-dcdjcolor	HP Deskjet 500C彩色打印（使用GhostScript）
-dcdj500	HP Deskjet 500C/540C（使用GhostScript）
-dcdj550	HP Deskjet 550C（使用GhostScript）
-ddeskjet	HP Deskjet和Deskjet Plus（使用GhostScript）
-dlaserjet	HP Laserjet
-dljet3	HP Laserjet III
-dljet2p	HP Laserjet IIP
-dljetplus	HP Laserjet+（使用GhostScript）
-dpaintjet	HP Paintjet（使用GhostScript）
-dpjxl	HP Paintjet XL（使用GhostScript）
-dpjxl300	HP Paintjet XK300（使用GhostScript）
-dhpgl	HP 7475A和兼容的绘图器
-dibmpro	IBM 9针Proprinter（使用GhostScript）

函数print还支持将图形导出到文件。导出文件的类型也是通过print的-device选项来进行指定的。下表列出了支持的文件格式。

-device	描述
-dbmp16m	24位位图（只适用于Windows。使用GhostScript）
-dbmp256	有固定颜色表的8位位图（只适用于Windows。使用GhostScript）
-dbmp	24位位图（只适用于Windows）
-dmeta	EMF（只适用于Windows）
-deps	EPS第一层，黑白图形，包括灰度图形
-depsc	EPS第一层，彩色图形
-deps2	EPS第二层，黑白图形，包括灰度图形
-depsc2	EPS第二层，彩色图形
-hdf	HDF，24位（只适用于Windows）
-dill	Adode图
-djpeg	JPEG，24位，质量设置为75（用Z缓冲复制图）
-djpegNN	JPEG，24位，质量设置为NN
-dpbm	PBM普通格式（只适用于UNIX。使用GhostScript）
-dpbmraw	PBM原始格式（只适用于Unix。使用GhostScript）
-dpcxmono	PCX，1位（只适用于Windows。使用GhostScript）
-dpcx24b	PCX，24位彩色（只适用于Windows。使用GhostScript）
-dpcx256	PCX，8位彩色（只适用于Windows。使用GhostScript）
-dpcx16	PCX，16位彩色（只适用于Windows。使用GhostScript）
-dpcx	PCX，8位彩色（只适用于Windows）
-dpgm	PGM可移植灰度映射，普通（只适用于Unix。使用GhostScript）
-dpgmraw	PGM可移植灰度映射，原始（只适用于Unix。使用GhostScript）
-dpng	PNG24位。
-dppm	PPM可移植像素映射，普通（只适用于Unix。使用GhostScript）
-dppmraw	PPM可移植像素映射，原始（只适用于Unix。使用GhostScript）
-dtiff	TIFF（用Z缓冲绘制图）
-tiff	将TIFF预览只添加到EPS格式中。还有必须使用EPS设备规范

Matlab还提供了将图像导出到图形文件的方法。函数getframe，inwrite，avifile和addframe提供了将图形生成和保存为图像文件的功能。关于这个方面更多的信息，请参见第28章。

29.4 PostScript支持

所有的PostScript设备和使用了GhostScript的设备都只能提供有限的字体支持。这些设备包括了打印设备和保存图像的设备。下面的表格显示了Matlab中支持的字体以及Windows的常用字体在标准PostScript字体中的映射关系。在下表中没有列出的字体都被映

射为Courier。Windows设备-dwin和-dwinc都使用了标准的Windows打印驱动程序，因此能够支持所有的安装字体。

PostScript字体	Windows中的等价字体
AvantGarde	
Bookman	
Courier	Courier New
Helvetica	Arial
Helvetica-Narrow	
NewCenturySchlBk	New Century Schoolbook
Palatino	
Symbol	
Times-Roman	Times New Roman
ZapfChancery	
zapfDingbats	

如果用户的打印机支持PostScript，那么就需要用到一个内置的PostScript驱动程序。第一层的PostScript是一个老版本的规范，有的打印机需要这个版本的规范。第二层的PostScript生成比第一层规范更精简和更快的代码，因此要尽可能地使用这一层的规范。

如果用户正在使用一台彩色打印机，那么就需要使用一个彩色打印机的驱动程序。黑白或者灰度打印机可以使用单色或者彩色打印机的驱动程序。但是，当一个彩色打印驱动程序被用在一个黑白打印机上的时候，文件要大一些，并且颜色出现了抖动，这使得在有的时候线条和文本不是那么清晰。当彩色线条用一个黑白打印驱动程序进行打印的时候，它们被转换成黑色的。当彩色线条用彩色打印驱动程序在黑白打印机上打印的时候，线条则被打印成灰色的。在这种情况下，除非线条有足够的宽度，否则它们通常和被打印的页面没有足够的视觉差异。

在Matlab中应用的时候，PostScript支持具有插值阴影的“表面”和“碎片”。在打印的时候，相应的PostScript文件包含了在表面或者碎片顶点处的颜色信息，这些顶点信息要求打印机去进行阴影插值。根据各个打印机特性的不同，这个插值的过程可能会占用大量的时间，也许还会导致打印机错误。解决这个问题的一个方法就是使用平面投影以及更细的网孔表面。另一种能够确保打印输出与屏幕显示图像相匹配的方法就是使用由足够高的分辨率的Z缓冲或者OpenGL算法绘图来打印输出。在这种情况下，输出格式是位图的格式，因此可能会导致一个很大的输出文件，但是这样做就不需要打印机进行插值计算了。

29.5　选择绘图器

绘图器对诸如顶点的数组和颜色数据这样的图形数据进行处理，将它们转换成适合于显示、打印或者导出的格式。有两种主要的图形格式类型：位图（或者是光栅）图形和向量图形。

位图图形格式包含了诸如格栅中各点的颜色这样的信息。随着点间距离的缩小和点的总数增加，图形的分辨率就增加，文件的大小也增大。增加用来指定格栅中每个点的颜色的位数也会增加在结果图形中可能的颜色的总数，因此也会增加结果文件占用字节数的大小。增加图形的复杂度不会对最后的输出文件的大小产生任何影响。

向量图形格式包含了关于用点，线和其他几何对象来重构这个图形的指令。向量图形可以很容易地改变大小，并且通常会得到比位图更高的分辨率。但是，随着在图形中对象数量的增加，重构这个图形所需要的指令的数量也增加，导致最后的文件的大小也增大。在有的点，这些指令的复杂程度会相当的大，使得输出设备无法处理，这样就使得图形无法在指定的输出设备上输出。

Matlab支持三种绘图方法：OpenGL，Z缓冲和Painter's。Painter's使用向量格式，而OpenGL方法和Z缓冲方法生成位图。默认时，Matlab根据figure和所使用的打印驱动程序或者文件格式的特点自动地选择一个绘图器。

> Matlab选择来进行打印或者导出的绘图算法不一定非要和用来在屏幕上显示“图形”的绘图算法相同。

Matlab的默认选项可以被覆盖，通过覆盖默认选项可使被打印出来的或者导出的“图形”看上去和屏幕上显示的一样，也可以避免将一个位图嵌入到一个向量格式的输出文件（比如PostScript或者EPS）中。

在有的情况下需要专门的绘图器。例如：

- 如果图形使用了真彩色而不是单色来表示表面或者碎片对象，那么图形必须用一个位图方法来绘制以便适当地捕获颜色。
- HPGL（-dhpgl）和Adode图形（-dill）输出格式必须使用Painter's绘图器。
- JPEG（-djpeg）和TIFF（-dtiff）输出格式总是使用Z缓冲绘图器。
- 光照效果不能用向量格式Painter's绘图器进行复制，因此在这种情况下必须使用位图方法。
- OpenGL绘图器是支持透明的惟一方法。

用来打印和导出的绘图器可以用对话框或者print命令的选项进行选择，或者通过设置句柄图形属性来进行选择。通过在Unix平台中Print Setup对话框和Print对话框中的Options按钮打开的对话框中进行选择，以及通过在所有其他平台上的Page Setup对话框中的Axes and Figures制表符所打开的对话框中进行选择，可以用来选择一个绘图器。如果用命令print的-zbuffer，-opengl和-painters选项在打印或者导出的时候选择一个指定的绘图器，这个选择将覆盖在任何其他地方所进行的选择。

29.6 句柄图形属性

某些句柄图形属性会影响图形打印或者导出的方式。打印和导出对话框中所进行的一些设置会改变当前图形的这些属性。下面这个表格列出了影响打印和导出的图形属性。

'PropertyName'	'PropertyValue'选项，[默认]	描述
Color	[RGB vector]	设置图形的背景颜色
InvertHardcopy	[{on} | off]	确定是否将图形的背景颜色打印或者导出。如果设置为on，不论Color属性是什么，都强制输出白色的图形背景
PaperUnits	[{inches} | centimeters| normalized | points]	设置用来度量一个打印或者导出图形大小的单位
PaperOrientation	[{portrait} | landscape | rotated]	设置图形在纸张中打印的方向
PaperPosition	[left bottom…width height] vector	设置图形在纸张或导出文件中的位置。width和height决定了打印或者导出的图形大小
PaperPositionMode	[auto | {manual}]	确定是用PaperPosition width还是height。当设置为auto的时候，就用“图形”窗口中所显示的图形的width和height来显示或者导出图形，即输出为所见即所得（WYSIWYG）
PaperSize	[width height]vector	用PaperUnits为单位度量的纸张大小
PaperType	[{usletter} | uslegal | A0 | A1 | A2 | A3 | A4 | A5 | B0 | B1 | B2 | B3 | B4 | B5 | arch-A | arch-B | arch-C | arch-D | arch-E | A | B | C | D | E | tabloid | <custom>]	所使用的纸张类型。选择一个PaperType就设置了相应的PaperSize
Renderer	[{painters}] | zbuffer | OpenGL]	显示绘图器
RendererMode	[{auto} manual]	确定如何选择绘图器。当设置为auto的时候，Matlab就自动独立地选择显示、打印和导出的绘图器。当设置为manual的时候，Renderer属性所设置的绘图器用来显示、打印和导出图形

有的axes属性也会影响打印和导出图形。尤为重要的是axes刻度线模式属性：

```
>> XTickMode [{auto} | manual]
>> YTickMode [{auto} | manual]
>> ZtickMode [{auto} | manual]
```

在一个图形被打印或者导出的时候，所绘制的结果图形通常和显示器上图形的大小不同。因为宽和高与显示器上显示的不同，因此Matlab可以重新对每个坐标轴上的刻度线的数字和位置进行重新标定。将上面的属性设置为'manual'将防止Matlab在图形被打印或者导出的时候改变坐标轴上的刻度线。

有的line属性还可以用来优化输出，例如：

```
Color:[ 3-element RGB vector ]
LineStyle: [ {-} | -- | : | -. | none ]
LineWidth: [ scalar ]
Marker: [ + | o | * | . | x | square | diamond | v | ^ | > | <
```

```
      | pentagram | hexagram | { none} ]
MarkerSize: [ scalar ]
MarkerEdgeColor: [ none | {auto} ] – or – a ColorSpec.
MarkerFaceColor: [ {none} | auto ] –or- a ColorSpec.
```

在打印彩色线条的时候，结果图形依赖于输出设备和打印驱动程序的功能。如果使用的是彩色打印机以及彩色打印驱动程序，那么所打印出来的线条颜色就是用户所希望的颜色。如果使用了一个黑白打印驱动程序，那么结果就是黑白的线条。如果使用了彩色打印驱动程序，但是所使用的打印机是一个黑白打印机，那么所得到的线条就是灰度线条。因为灰度是通过色彩调谐打印出来的，而色彩调谐可能会导致线条无法与背景区分开来，因此在这种情况下就会出现问题。当用黑白打印机打印这些线条的时候，也会出现不同的相关的问题。当多条实线都被打印成黑色的时候，他们就无法像屏幕中显示的那样进行区分了。在这种情况下，line属性'color'，'LineStyle'，'LineWidth'和'Marker'就可以用来为绘制的图形添加一些其他可以加以区分的特征。

最后，在打印或者导出图形的时候修改文本通常也是很有用处的。可供使用的text的属性包括下边几个：

- Color: [3-element RGB vector]
- FontAngle: [{normal} | italic | oblique]
- FontName: [font name]
- FontSize: [scalar]
- FontUnits: [inches | centimeters | normalized | {points} | pixels]
- FontWeight: [light | {normal} | demi | bold]

在将一个图形打印或者导出到一个较小尺寸的图形的时候，增加诸如标题和坐标轴标签等文本字符串的字体大小通常会使得文本更加易读。如果在图形中使用的字体不是上面所列出的Matlab支持的PostScript输出的11种字体，那么将字体变成这11种字体中的一种就能够避免出现意想不到的字体置换。有时候，在打印输出中，黑体字的效果要比普通字体要好。这样的改动通常能够使得打印或者导出的图形的外观更加美观。当导出图形用演示软件进行显示的时候，在这样的软件中字体必须足够的大并且能够和其他东西区分开，使得它从很远的距离都能够看到，因而字体特性是很重要了。

29.7 设置默认值

Matlab将打印和导出的厂家默认选项进行如下设置。

- 图形大小为8×6英寸
- 方向为纵向
- 图形和坐标轴的背景色是反白色
- 如果可能，就用US信纸(8.5×11英寸)
- 图形位于打印纸中央

- 图形被修剪
- 输出是RGB图形（不是CYMK）。
- 刻度线被重新计算
- Matlab选择绘图器
- Uicontrol打印
- 在Unix中打印设备为-dps2，在PC平台中为-dwin。

默认的打印设备是在$TOOLBOX/local/printopt.m文件中设置的。用户可以编辑这个文件来改变各个Matlab对话期中默认的打印设备。如果用户没有对这个文件的写权限，用户可以用命令edit printopt.m来编辑这个文件，作出用户自己的修改，将printopt.m文件保存到一个本地目录中，并且确保这个本地目录在MATLABPATH的$TOOLBOX/local之前。例如，如果用户在UNIX平台上使用彩色PostScript打印机来打印图形输出，可编辑printopt.m文件，并将行dev='-dpsc2'；添加到这个文件中指定的位置。

其他默认选项可以通过设置startup.m文件中的默认属性来改变。例如，set(0, 'DefaultfigurePaperType', 'A4')；将默认的纸张类型变为A4。句柄0寻址“根”对象。属性'DefaultFigurePaperType'设置默认的'PaperType'图形属性值。给任何句柄图形属性名加上前缀'Default'都表明后面伴随的属性值必须当作默认值使用。任何给定对象的默认属性必须在句柄图形层次中的更高一级进行设置。例如，默认的图形属性必须在“根”对象这一级进行设置，而默认的坐标轴属性必须在“图形”对象或者“根”对象这一级进行设置。因此，默认设置通常是在“根”这一级进行的。例如：

```
set(0,'DefaultFigurePaperOrientation','landscape');
set(0,'DefaultFigurePaperPosition',[0.25 0.25 10.5 8]);
set(0,'DefaultAxesXGrid','on','DefaultAxesYGrid','on');
set(0,'DefaultAxesLineWidth','1');
```

这些语句告诉Matlab使用风景模式，将整个页面用图形填满，打印（并显示）x轴和y轴格栅，以一个像素点宽度显示和打印线条，并将它们作为所有图形和坐标轴中的默认设置。

还有很多选项可以用句柄图形属性进行设置。大多数的打印和导出属性都是图形和坐标轴属性。关于句柄图形属性的更多信息，请参见下一章。

29.8　小　结

Matlab提供了一个非常灵活的系统来以多种格式打印图形和生成输出。大多数应用程序可以导入Matlab支持的图形文件格式，但是很多这样的应用程序对结果图形的编辑能力有限。如果图形在打印或者导出之前得到了编辑，并且设置了适当的选项，那么就能够获得最佳的输出效果。使用最广泛，最灵活的输出格式为PostScript，EPS，EMF和TIFF。Matlab自身支持所有这些格式，并且可用GhostScript将它们转换成其他格式。

第30章 句柄图形

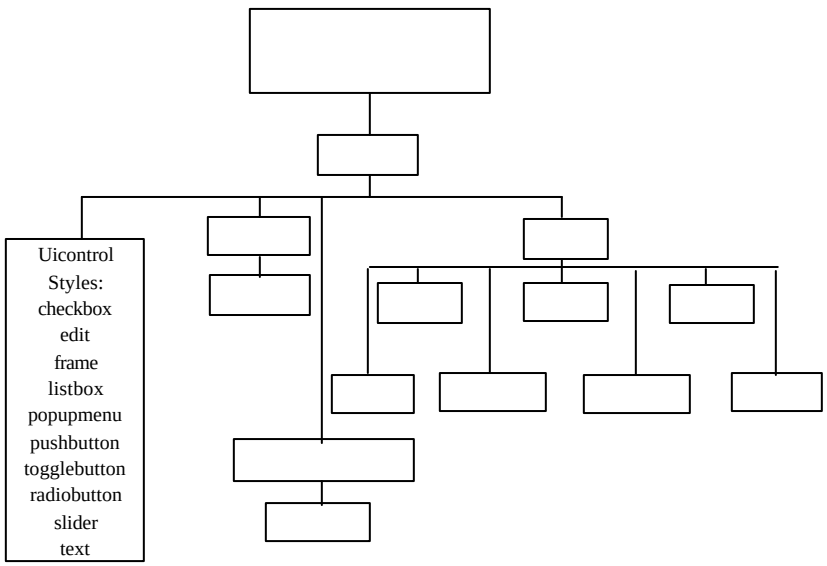
什么是句柄图形？句柄图形就是一组底层图形函数的名称，这些函数用来在Matlab中生成图形。句柄图形提供了对图形的高级控制。这些函数和选项通常隐藏在高层图形的M文件中。句柄图形使得用户可以定制图形中不能用前面描述的高层命令和函数寻址的部分。图形窗口工具栏和菜单为很多句柄图形特性提供了图形用户界面接口，使得那些不是经常使用Matlab的用户没有必要去理解句柄图形。

本章没有对句柄图形进行详尽的讨论，因为这样做将会牵涉到过多的细节。本章旨在使用户对句柄图形概念有基本的理解，并且提供了足够的实际应用信息，这样，很少用Matlab的用户也能理解句柄图形的特性。

30.1 对 象

句柄图形的基本思想就是：Matlab的每一个可视部分就是一个对象，每个对象都有一个相应的惟一标识符或者句柄，并且每个对象都有可以根据需要进行修改的属性。在现代计算机科学中，很多领域都出现了对象这个词。面向对象的编程语言，数据库对象，操作系统和应用程序接口都使用了对象的概念。对象不太严格的定义就是：对象是一组密切相关的数据和函数的集合，这些数据和函数构成了一个统一的整体。在Matlab中，一个图形对象就是一个可以被单独处理的单位。

图形命令所创建的都是图形对象，包括有图形窗口或者仅包括图形，以及坐标轴、线条、表面、文本等等。这些对象的继承层次图如下所示。



计算机屏幕本身就是根对象，它是所有其他对象的父对象。图形对象是根对象的子对象；坐标轴对象和用户接口对象（将会在下一章中讨论）是图形对象的子对象。线条、方形、文本、表面、光照、碎片和图像对象都是坐标轴对象的子对象。根对象可以包含一个或者多个图形，而每个图形可以包含一套或者多套坐标轴。其他所有对象（除了下一章讨论的uicontrol，uicontextmenu和uimenu对象之外）都是坐标轴的子对象，并且都显示在某个坐标轴系之内。如果要生成的图形对象的父对象不存在，则函数在生成图像之前先生成父对象。例如，如果图形对象不存在，函数plot(rand(1,10))就会先生成一个有预先定义的属性值的图形对象和一组坐标轴对象，然后才在这个坐标轴内部绘制曲线。

30.2 对 象 句 柄

假设用户有三个打开的图形窗口，其中的两个有子图，并且用户想要改变其中一个子图的坐标轴上一个线条的颜色。用户该怎样来确定想要改变颜色的线条呢？在Matlab中，每个对象都有一个相关的标识符，称为句柄，它是一个双精度数。每当一个对象被创建的时候，就为这个对象生成一个惟一的句柄。根对象，也就是计算机屏幕的句柄，总是0。命令Hf_fig=figure生成了一个新的图形，并将其句柄返回给变量Hf_fig。图形句柄在正常的情况下是整数，并且通常显示在图形窗口的标题栏中。其他对象句柄都是双精度浮点数。所有的对象生成函数都返回它们创建的对象的句柄。下边这个表格列出了Matlab中的句柄图形对象。

对象	描述
root	运行Matlab的计算机平台
figure	计算机屏幕上显示的窗口
uicontrol	一个图形中的用户接口控件
uimenu	一个图形中的用户接口菜单
uicontextmenu	一个图形中的用户接口上下文菜单
axes	绘制在一个图形中的直角坐标系
image	一个坐标系中的二维图形
light	在一个坐标系中发光的直接光源
line	在一个坐标系中连接数据点的线条
patch	在一个坐标系中由顶点定义的多边形区域
rectangle	在一个坐标系中方形和椭圆形图形对象
surface	在一个坐标系中一个表面的三维表示
text	一个坐标系中的字符串

Matlab提供了获得图形、坐标轴以及其他对象句柄的命令。例如，Hf_fig=gcf就返回当前图形的句柄，而 Ha_ax=gca就返回当前图形中当前坐标轴系的句柄。这些函数和其他对象处理工具将会在本章的后边陆续进行讨论。

为了提高可读性，本书中包含有对象句柄变量的变量名都以大写的H开头，后边跟着

一个标识对象类型的字母，然后是一个下划线，最后是一个或者多个描述性的字符。因此，Hf_fig是一个图形的句柄，Ha_axl是一个坐标轴系对象的句柄。而 Ht_title是一个文本对象的句柄。当一个对象类型未知的时候，就使用字母x，比如Hx_obj。虽然用户可以给句柄赋以任何名称，但是按照这个规则进行命名，将会使得在M文件中找出句柄变量变得非常方便。

所有创建对象的Matlab函数都会为所生成的对象返回一个句柄或句柄列向量。有的图形是由多个对象构成的。例如，一个mesh图形是由一个表面对象构成的，因此只有一个句柄。而一个polar图形是由多个线条对象构成的，每个线条对象都有相关的且相互独立的。

30.3 对 象 属 性

所有的对象都有一组定义其特征的属性。通过设置这些属性，用户可以调整图形显示的方式。尽管有的属性名在所有的对象中都能见到，但与每个对象类型（例如，坐标轴系线条，表面）相关的属性都是惟一的。对象属性可以包含很多属性，比如对象的位置，颜色，对象类型，父对象句柄，子对象句柄以及很多其他属性。每个不同的对象都有其自身独立的属性，用户可以改变这些属性，而不会改变相同类型的其他对象的属性。

对象属性是由属性名以及相应的属性值构成的。属性名是大小写混合的字符串，通常，字符串的第一个字母大写，例如'LineStyle'。但是，Matlab在识别属性的时候不区分大小写。另外，用户只需要用足够多的字符来惟一地标识属性名。例如，一个坐标轴对象的位置属性可以被命名为'Position'，'position'，或者甚至是'pos'。

在生成一个对象的时候，其初始化属性值就是其默认属性值，而这些默认属性值可以用两种方法的任何一种来改变。对象生成函数的调用可以用（'PropertyName',PropertyValue）对的方式，也可以在对象生成之后再改变它的属性值。例如：

```
>> Hf_1 = figure('Color', 'yellow')
```

这条命令生成一个属性值为默认设置的新的图形对象，但是它的背景颜色被设置成黄色，而不是其默认的颜色。

除了Matlab的图形窗口菜单和工具条特性之外，函数inspect还提供了查看和修改对象属性的图形用户界面。为了使用这个函数，只需要输入inspect(H)，其中H是想要查看和修改的对象的句柄。

30.4 get和set

get和set这两个函数被用来获得或改变句柄图形对象的属性。函数get返回对象的一个或者多个属性的当前值。最常用的调用语法为get(handle, 'PropertyName')。例如：

```
>> p = get(Hf_1, 'Position')
```

这条命令返回句柄为Hf_1的图形的位置向量。类似地：

```
>> c = get(Hl_a, 'Color')
```

这条命令返回句柄为Hl_a的对象的颜色属性。

函数set改变句柄图形对象的属性值，其使用语法为set(handle, 'PropertyName'，PropertyValue)。例如：

```
>> set(Hf_1, 'Position',p_vect)
```

这条命令将句柄为Hf_1的图形对象的位置设置为由向量p_vect所设定的位置。类似地，

```
>> set(Hl_a, 'Color', 'r')
```

这条命令将句柄名为H1_a的对象的颜色设置为红色。通常，函数set可以有任何数量的('PropertyName'，PropertyValue)对。例如：

```
>> set(Hl_a, 'Color',[1 0 0], 'LineWidth',2, 'lineStyle', '--')
```

这条命令将句柄名为H1_a的线条对象的颜色变为红色，将它的线条宽度变为两个像素点，并且将其线型变成虚线。

除了这些主要的用途之外，函数get和set还提供了对象属性的反馈信息。例如，set(handle, 'PropertyName')返回一列值，这列值就是句柄名为handle的对象的PropertyName可以被赋予的值。例如：

```
>> set(Hf_1, 'Units')
[ inches | centimeters | normalized | points | {pixels} | characters]
```

这条语句显示了句柄名为Hf_1的图形对象的'Units'属性的6个可能的字符串值，并且表明'pixels'是默认值。

如果用户将'propertyName'设置成为一个没有固定值系列的属性，Matlab就会告诉用户这个事实，例如：

```
>> set(Hf_1, 'Position')
A figure's 'Position' property does not have a fixed set of property values.
```

除了set命令之外，句柄图形对象生成函数也接受多个属性名和属性值对。例如：

```
>> figure('Color', 'blue', 'NumberTitle', 'off', 'Name', 'My Figure')
```

这条命令生成了一个新的图形对象，其背景为蓝色，标题为'My Figure'，而不是默认的窗口标题'Figure No.1'。

为了更形象地阐述上边的概念，请看下边这个例子。

```
>> Hf_fig = figure  % create a figure
Hf_fig =
       1
>> Hl_light = light % add default light to an axes in the figure
Hl_light =
         107
>> set(Hl_light)    % find settable properties of light
          Position
          Color
```

```
            Style: [ {infinite} | local ]

            ButtonDownFcn: string -or- function handle -or- cell array
            Children
            Clipping: [ {on} | off ]
            CreateFcn: string -or- function handle -or- cell array
            DeleteFcn: string -or- function handle -or- cell array
            BusyAction: [ {queue} | cancel ]
            HandleVisibility: [ {on} | callback | off ]
            HitTest: [ {on} | off ]
            Interruptible: [ {on} | off ]
            Parent
            Selected: [ on | off ]
            SelectionHighlight: [ {on} | off ]
            Tag
            UIContextMenu
            UserData
            Visible: [ {on} | off ]
>> get Hl_light)   % get all properties and names for light
            Position = [1 0 1]
            Color = [1 1 1]
            Style = infinite

            BeingDeleted = off
            ButtonDownFcn =
            Children = []
            Clipping = on
            CreateFcn =
            DeleteFcn =
            BusyAction = queue
            HandleVisibility = on
            HitTest = on
            Interruptible = on
            Parent = [108]
            Selected = off
            SelectionHighlight = on
            Tag =
            Type = light
            UIContextMenu = []
            UserData = []
            Visible = on
```

在上边的例子中用到了光照对象，这是因为它几乎不包含任何属性。上边的例子还生成了一个图形对象并且返回其句柄。然后，就生成了一个返回其自身句柄的光照对象，生成了一个坐标轴对象，因为光照对象是坐标轴对象的子对象；坐标轴句柄可以从光照对象的'Parent'**属性中获得。**

请注意，每个对象的属性列表被分成了两组。第一组列出了对该对象类型而言是惟一的属性，第二组列出了对所有的对象类型通用的属性。还需要用户注意的是，函数set**和**get**返回的属性列表有些许差异。函数**set**只列出可以用**set**命令改变的属性，而**get**列出所有可见**

的对象属性。在上边的例子中，get列出了'Type'属性而set没有将其列出。这个属性只能读而不能改变；也就是说，它是一个只读属性。

在Matlab的每一个发布版本中，各个对象类型的属性数量是固定的，但是不同的对象类型的属性数量是不同的。如上边的例子所显示的那样，一个光照对象列出了3个惟一的属性和18个通用的属性，或者说总共21个属性。而一个坐标轴对象列出了93个属性。很明显，彻底地描述和显示所有13种对象类型的所有属性已经超出了本书的范畴。但是，在附录中，本书还是列出了这些属性。

作为对象句柄的用法的一个示例，考虑一下用非标准颜色绘制一条曲线的问题。在这个例子中，线条颜色被声明为使用一个RGB值为[1 .5 0]的颜色，也就是一种中橙色。

```
>> x = -2*pi:pi/40:2*pi;           % create data
>> y = sin(x);                     % find sine of x
>> Hl_sin = plot(x,y)              % plot sine and save handle
Hl_sin =
    59.0002
>> set(Hl_sin, 'Color',[1 .5 0], 'LineWidth',3)% Change color and width
```

现在，添加一条颜色为淡蓝色的余弦曲线。

```
>> z = cos(x);                     % find cosine of x
>> hold on                         % keep sine curve
>> Hl_cos = plot(x,z);             % plot cosine and save handle
>> set(Hl_cos, 'Color',[.75 .75 1])     % color it light blue
>> hold off
```

也可以用更少的步骤来实现上述相同的功能。

```
>> Hl_lines = plot(x,y,x,z);   % plot both curves and save handles
>> set(Hl_line(1), 'Color',[1 .5 0], 'LineWidth',3)
>> set(Hl_line(2), 'Color',[.75 .75 1])
```

应该怎样给图形添加一个标题，并且使得字体大小比正常字体要大呢？

```
>> title('Handle Graphics Example')         % add a title
>> Ht_text = get (gca, 'Title')             % get handle to title
>> set(Ht_text, 'FontSize',16)              % customize font size
```

最后这个例子展示了坐标轴对象非常有趣的一个特点。每个对象都有一个'Parent'属性以及一个'Children'属性，这些属性包含了与这个对象有继承关系的对象的句柄。绘制在一个坐标系中的线条对象，则它的'Parent'属性中包含了坐标轴对象的句柄，它的'Children'属性中包含了一个空数组。同时，坐标轴对象的'Parent'属性中包含了它所在的图形的句柄，而其'Children'属性中包含了线条对象的句柄。尽管用text和gtext命令创建的文本对象是坐标轴对象的子对象，并且它们的句柄都包含在坐标轴对象的'Children'属性中，但是和标题字符串和坐标轴标签相关的句柄却不在其中。这些句柄位于坐标轴对象的'Title'，'XLabel'，'Ylable'和'ZLable'属性中。这些文本对象总是在一个坐标轴对象创建时生成。命令title只是设置当前的坐标轴中的标题文本对象的'String'属性。最后，标准的Matlab函数title，xlable，ylabel和zlabel返回句柄并接受属性和属性值参数。例如，下面这个命令就给当前的图形添

加了一个大小为24像素点的绿色标题，并且返回这个标题文本对象的句柄。

```
>> Ht_title = title('This is a title. ', 'FontSize',24, 'Color', 'green')
```

除了set和get之外，Matlab还提供了几个其他函数用来处理对象以及对象的属性。可以用函数copyobj将对象从一个父对象拷贝到另一个父对象。例如：

```
>> Ha_new = copyobj(Ha_ax1,Hf_fig2)
```

这条语句拷贝句柄名为Ha_ax1的坐标轴对象及其所有子对象，然后给它分配一个新的句柄，然后再将这个对象放在句柄名为Hf_fig2的图形中。这个新的坐标轴对象的句柄被返回到Ha_new中。根据刚才我们讲述的继承关系，任何对象都可以被拷贝到任何合法的父对象中。函数copyobj的两个参数中的任何一个或者两个参数都可以是句柄向量。

请注意，可以通过将任何对象的'Parent'属性变成另一个合法的父对象句柄而将这个对象从它的一个父对象转移到另一个父对象。例如：

```
>> figure(1)
>> set(gca, 'Parent',2)
```

这条命令将当前的坐标轴对象及其所有子对象从句柄为1的图形移动到句柄为2的图形。图形2中的任何现存的对象都不会收到影响，除了它们可能被这个改变了位置的对象所遮掩之外。

可以用函数delete(handle)来删除任何对象及其所有的子对象。类似地，reset(handle)将句柄名为handle的除了'Position'之外的所有属性重置为这个对象类型的默认属性。如果handle是对象句柄的一个列向量，set，reset，copyobj和delete能够影响所有被列出的对象。

在将函数get和set的输出赋值给其他变量的时候，将返回一个结构，请看下边这个例子：

```
>>lprop = get(Hl_light)
lprop =
               BeingDeleted : 'off'
                 BusyAction : 'queue'
              ButtonDownFcn : ''
                   Children : [0×1 double]
                   Clipping : 'on'
                      Color : [1 1 1]
                  CreateFcn : ''
                  DeleteFcn : ''
          HandleVisibility : 'on'
                    HitTest : 'on'
              Interruptible : 'on'
                     Parent : 108
                   Position : [1 0 1]
                   Selected : 'off'
        SelectionHighlight : 'on'
                      Style : 'infinite'
                        Tag : ''
                       Type : 'light'
              UIContextMenu : []
```

```
                    UserData: []
                     Visible: 'on'
>> class(lprop) % class of get (Hl_light)
ans =
struct
>> lopt = set(Hl_light)
lopt =
                       BusyAction: {2×1 cell}
                    ButtonDownFcn: {}
                         Children: {}
                         Clipping: {2×1 cell}
                            Color: {}
                        CreateFcn: {}
                        DeleteFcn: {}
                  HandleVisibility: {3×1 cell}
                          HitTest: {2×1 cell}
                     Interruptible: {2×1 cell}
                           Parent: {}
                         Position: {}
                         Selected: {2×1 cell}
               SelectionHighlight: {2×1 cell}
                            Style: {2×1 cell}
                              Tag: {}
                     UIContextMenu: {}
                         UserData: {}
                          Visible: {2×1 cell}
>> class(lopt) % class of set(Hl_light)
ans =
struct
```

由此产生的结构的各个字段名就是对象各属性名字符串，并且根据字母顺序进行了排列。请注意，尽管属性名是不区分大小写的，但是这些字段名却是区分大小写的，例如：

```
>> lopt.BusyAction
ans =
    'queue'
    'cancel'
>> lopt.busyaction
??? Reference to non-existent field 'busyaction'.
```

也可以利用结构来设置属性值的组合。例如：

```
>> newprop.Color = [1 0 0];
>> newprop.Position = [-10 0 10];
>> newprop.Style = 'local';
>> set(Hl_light,newprop)
```

这些语句改变了属性'Color'，'Position'**和**'Style'**的值，但是对光照对象的其他属性没有任何影响。请注意，用户不能在获得一个属性值结构后直接用相同的结构来重置这些值，例如：**

```
>>light_prop = get(Hl_light);
```

```
>>light_prop.Color = [1 0 0];       % change the light color to red
>>set(Hl_light,light_prop);         % reapply the property values
??? Error using ==> set
Attempt to modify read-only light property: 'BeingDeleted'.
```

这是因为'BeingDeleted'和'Type'是光照对象仅有的只读属性，用户可以通过将'BeingDeleted'和'Type'字段从结构中删除的方法来解决这个问题，例如：

```
>>light_prop = rmfield(light_prop, 'Type', 'BeingDeleted');
>> set(Hl_light,light_prop)
```

对于有更多只读属性的对象，所有的只读属性必须先从结构中删除，然后才能用这个结构来设置属性值。

一个单元数组也可以用来查询一组属性值。为此，先创建一个单元数组，该数组包含需要查询的属性名，各属性名需要按一定顺序排列，然后将这个单元数组传递给get。那么返回的结果也是一个单元数组，例如：

```
>> plist = {'Color', 'Position', 'Style'}
plist =
    'Color'  'Position'  'Style'
>> get(Hl_light,plist)
ans =
    [double]    [1×3 double]    'local'
>> class(ans)
ans =
cell
```

关于get函数，还有一点需要引起用户的注意。如果H是一个句柄向量，get(H, 'PropertyName')就返回一个单元数组而不是一个向量。请看下面这个例子，给定一个有四个子图的图形窗口：

```
>> Ha = get(gcf, 'Children')               % get axes handles
Ha =
    15.0002
    13.0002
    11.0002
     9.0002
>> Ha_kids = get(Ha, 'Children')           % get handles of axes children
Ha_kids =
    [   16.0002]
    [4×1 double]
    [   12.0002]
    [2×1 double]
>> class(Ha_kids)
ans =
    cell
>> Hx = cat(1,Ha_kids{:})              % convert to column vector
Hx =
    16.0002
```

```
    26.0002
    24.0002
    18.0002
    22.0002
    12.0002
    14.0002
    10.0002
>> class(Hx)
ans =
    double
```

现在，Hx可以作为需要以一个对象句柄向量为参数的句柄图形函数的一个参数。

30.5　查 找 对 象

如上所示，句柄图形提供了一个访问图形窗口中对象的方法，并且允许用户用get和set命令来定制图形。这些函数的使用需要用户知道需要进行处理的对象的句柄。在句柄未知的情况下，Matlab提供了一些函数用来寻找对象句柄。在这些函数中，有两个函数，gcf和gca，是引入较早的两个函数。例如：

```
>> Hf_fig = gcf
```

将返回当前图形窗口的句柄，而

```
>> Ha_ax = gca
```

将返回当前图形窗口中当前坐标轴的句柄。

除了上边这两个函数之外，Matlab还提供了一个函数gco用来获得当前对象的句柄。例如：

```
>> Hx_obj = gco
```

将返回当前图形中当前对象的句柄，或者用另外一种方式：

```
>> Hx_obj = gco(Hf_fig)
```

将返回句柄名为Hf_fig的图形中当前对象的句柄。

当前对象被定义为在一个图形中鼠标所点击的最后一个对象。这个对象可以是除了根对象之外的任何对象。当一个图形被刚刚生成的时候，不存在当前对象，因此gco将返回一个空数组。在gco可以返回一个对象句柄之前，鼠标必须先要点击在一个图形之内。

一旦获得了对象句柄，就可以通过查询一个对象的'Type'属性来找出这个对象的类型，这个'Type'属性是一个诸如'figure'，'axes'或者'text'这样的字符串对象名。'Type'属性对所有的对象都是通用的。例如：

```
>> x_type = get(Hx_obj, 'Type')
```

这条语句对所有的对象而言，肯定会返回一个有效的对象字符串。

当用户需要知道除了'CurrentFigure'，'CurrentAxes'或者'CurrentObject'之外的对象的句柄的时候，函数get可以用来获取一个对象的子对象的句柄向量。例如：

```
>> Hx_kids = get(gcf, 'Children');
```

这条语句返回一个包含了当前图形的子对象的句柄的向量。

这种获取'Children'句柄的方法可以用来在整个句柄图形层次结构中搜索以找到指定的对象。例如，请考虑一下在绘制了一些数据之后，寻找一个绿色线条对象的句柄的问题。

```
>> x = -pi:pi/20:pi;                    % create some data
>> y = sin(x); z = cos(x);
>> plot(x,y, 'r',x,z, 'g');             % plot lines in red and green

>> Hl_lines = get(gca, 'Children');     % get the line handles
>> for k=1: size(Hl_lines)              % find the green line
>>  if get(Hl_lines(k), 'Color') == [ 0 1 0]
>>       Hl_green = Hl_lines(k)
>>      end
>> end
Hl_green =
    58.0001
```

尽管这种方法是有效的，但是在对象数量较多的情况下，就会变得很复杂。除非对这些对象进行单独的检测，否则这项技术将会遗漏标题中的文本对象和坐标轴标签。

为了简化寻找对象句柄的过程，Matlab包含了内置的函数findobj，它返回给定属性值的对象的句柄。Hx=findobj(Handles, 'flat', 'PropertyName',PropertyValue)返回在Handles中所有'PropertyName'的属性值为PropertyValue的对象的句柄。也允许出现多个('PropertyName',PropertyValue)对，并且要求结果对象的所有的这些属性都必须满足给出的相应的属性值。当Handles被省略的时候，就将这个参数假定为根对象。当没有给出('PropertyName',PropertyValue)对的时候，所有的对象都符合要求，因此将返回所有对象的句柄。当没有找到符合给定条件的对象，那么findobj将会返回一个空矩阵。利用函数findobj，上边那个例子所列出的问题可以用一行语句来解决：

```
>> Hl_green = findobj(0, 'Type', 'line', 'Color',[0 1 0]);
```

用户可以用对所有对象都通用的'HandleVisibility'属性来隐藏指定句柄。这个属性是非常方便的，因为它避免了用户无意间对对象属性的删除或者修改。当一个对象的'HandleVisibility'属性被设置为'off'或者'callback'的时候，在Command窗口对findobj的调用将不会返回这些对象的句柄。被隐藏的句柄不会出现在子对象列表中，也不会作为gcf，gca或者gco的输出。但是，当这个属性被设置为'callback'的时候，这些句柄在执行一个回调字符串的时候还是能够被发现。关于回调字符串的信息请参见第30.12节。

30.6 用鼠标选择对象

命令gco返回当前对象的句柄，而当前对象就是鼠标最后点击的这个对象。当鼠标点击在多个对象的交叉点附近的时候，Matlab使用一些规则来确定哪个对象变成当前对象。每个对象都有一个自己的选择区域。鼠标点击在这个区域内，就选中了这个区域所对应的对象。对于线条对象而言，其选择区域包括线条本身，以及距离这个线条5个像素点范围内的区域（如果这个区域存在的话）。一个表面，碎片或者文本对象的选择区域是包含这个对象的最小的长方形区域。一个坐标轴对象的的选择区域是坐标轴框本身加上标签和标题所确定的区域。在坐标轴内部的对象，比如线条和表面，是重叠顺序中是处于更高一层的对象，因此鼠标点中它们的时候，选中的是相应的对象而不是坐标轴。点中在坐标轴的选择区域以外的区域，就选中了图形本身。

当鼠标点击的是在两个或者多个对象选择区域的边界之内的时候，重叠顺序就决定了哪个对象成为当前的对象。重叠顺序决定了哪个对象是重叠在其他对象的上面。初始时，重叠顺序在对象生成时即被决定，最新生成的对象位于重叠的最上面。例如，在用户输入两个figure命令之后，就生成了两个图形。第二个图形被画在第一个图形的上面。结果的重叠顺序就是图形2在图形1上面，并且gcf所返回的句柄就是2。如果输入了figure(1)命令，或者鼠标点击了图形1，那么重叠顺序就发生了变化，图形1移动到了重叠顺序的最上面，并且变成了当前的图形。

在上边这个例子中，重叠顺序对于重叠在计算机屏幕上的窗口而言，是显而易见的的。但是，并非总是这种情况。在绘制两条线条的时候，所绘制的第二条线条和第一条线条在两线相交的地方可以看出是第二条位于第一条的上边。如果用鼠标在某个其他地方点击了第一条线条，第一条线条就变成了当前的对象，但是重叠顺序并不会改变。直到重叠顺序被明确地改变之前，用鼠标点击两线的交点将会始终选择到第二条线条。

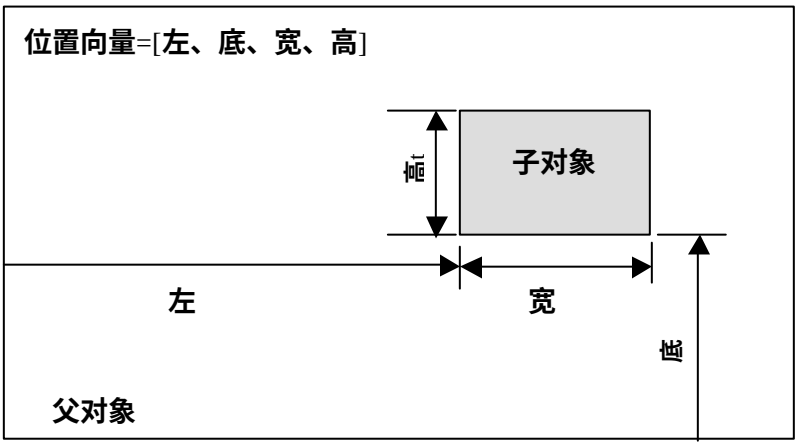
重叠顺序是由一个给定对象的'Children'句柄出现的顺序给定的。也就是说，Hx_kids=get(handle, 'Children')按照重叠顺序返回子对象的句柄。向量Hx_kids的第一个元素位于重叠的最上层，而最后一个元素位于重叠的最底层。可以通过改变父对'Children'属性值的顺序来改变重叠顺序，例如：

```
>> Hf = get(0, 'Children');
>> if length(Hf)>1
    set(0, 'Children',Hf([end 1:end-1]);
end
```

这条命令将底层的图形移动到重叠的最上面，使得这个图形成为新的当前图形。

30.7 位置和单位

图形对象和大多数其他的句柄图形对象的'Position'属性是一个四元素的行向量。如下图所示，这个向量中的值是[left,bottom,width,height]，其中[left,bottom]表示该对象的左下角相对于其父对象的位置，而[width,height]是这个对象的宽和高。



这些位置向量的单位是由对象的'Unit'属性中声明的单位决定的。例如

```
>> get(gcf, 'Position')
ans =
    920     620     672     504
>> get(gcf, 'units')
ans =
pixels
```

这些语句表明当前图形对象的左下角的位置位于距离屏幕的左下角右边920个像素点和上边620个像素点的位置，并且图形对象的宽度为672像素，高为504像素。请注意，一个图形的'Position'向量给出的是图形对象自身内部可以绘图的区域,并且不包括图形窗口的窗口边框，滚动条，菜单栏或者标题栏。

另外，图形还有一个没有在文档中出现的'OuterPosition'属性，这个属性给出的向量值包含了窗口的边框，例如：

```
>> get(gcf, 'Position')         % drawable position
ans =
    920     620     672     504
>> get(gcf, 'OuterPosition')    % outside position
ans =
    916     616     680     531
```

这里，外层位置包括了图形窗口的外层边框的左侧位置，底部位置，宽度和高度。当通过选择图形窗口中的View菜单，将图形或者摄像头工具栏设置为显示或者隐藏时，用户就可以通过设置图形的'ActivePositionProperty'属性来保持可绘图区域位置或者外层位置参数。将这个属性设置为'Position'就赋予可绘图位置优先权，因此在工具栏被显示或者隐藏的时候就保持'Positon'的值不变。而将这个属性设置为'OuterPosition'时，则优先权赋给'OuterPosition'，并且使得这些值在工具栏显示或者隐藏的时候保持不变。

图形的'Unit'属性的默认设置为像素，但是也可以是英寸，厘米，点，字符或者标准化的坐标。像素表示的是屏幕像素，是可以在计算机屏幕上显示的最小的矩形对象。例如，

一台计算机的显示被设置为800×600，意思是800像素宽，600像素高。点是一个排版标准，其中一点等于1/72英寸。字符单位是和默认的系统字体中一个字符的宽度相关的单位。值1等于默认系统字体中字母x的宽度。标准化的坐标是在0和1之间的值。在标准化的坐标中，父对象的左下角的坐标为[0,0]，右上角的坐标为[1,1]。英寸和厘米这两种单位保持原义，就不再解释了。

为了展示不同的'Units'属性值，请重新考虑上例：

```
>> set(gcf,'units','inches')    % INCHES
>> get(gcf,'position')
ans =
         7.9224       5.3362       5.7931        4.3448
>> set(gcf,'units','cent')      % CENTIMETERS
>> get(gcf,'position')
ans=
         20.108       13.544       14.703       11.027
>> set(gcf,'units','normalized')% NORMALIZED
>> get(gcf,'position')
ans=
       0.57438      0.51583          0.42         0.42
>> set(gcf,'units','points')    % POINTS
>> get(gcf,'position')
ans=
         570.41       384.21        417.1       312.83
>> set(gcf,'units','char')      % CHARACTERS
>> get(gcf,'position')
ans=
         153.17       38.688          112         31.5
```

当显示器及显示器分辨率都相同时，所有这些值均表示相对于计算机屏幕相同的图形位置。

坐标轴的位置也是一个四元素的向量，其形式和上边的形式相同，也是[left,bottom,width,height]，但是它们表示对象相对于父图形的左下角的位置。通常，一个子对象的'Position'属性是相对于其父对象的位置而言的。

为了描述性更强，计算机屏幕或者根对象的位置属性不叫'Position'，而是'ScreenSize'。在这种情况下，[left,bottom]总是为[0,0]，而[width,height]就是整个计算机屏幕的大小，它们的单位由根对象的'Units'声明的单位决定。

30.8 默认属性

在每个对象生成的时候，Matlab将对它的默认属性赋值。内嵌的默认属性指的就是工厂级默认属性。为了覆盖这些默认属性，用户必须用set和get函数来设置或者获得这些属性值。如果用户每次生成对象的时候都想要改变相同的属性，Matlab允许用户设置自己的默认属性。它使得用户可以改变单个对象的默认属性，也可以改变对象层次结构中某对象类型的默认属性。在创建一个对象的时候，Matlab在父对象这一层寻找默认属性值。如果没有找到默认属性值，它就沿着对象层次结构向上查找，直到找到一个默认值或者直到搜索

到内嵌的工厂级默认值为止。

用户通过使用一种特殊的属性名字符串，来设置对象层次结构中任何一层的用户自己的默认值。这样的字符串由'Default'，后边紧跟对象类型和属性名构成，用户在set命令中所使用的句柄决定了这个默认值所应用在父－子层次结构中的位置。例如：

```
>> set(0, 'DefaultFigureColor',[.5 .5 .5])
```

这条命令将所有新创建的图形对象的默认背景颜色设置为中灰，而不是Matlab的默认设置。这个属性也适用于根对象（其句柄总是为0），因此所有的新图形都有一个灰色的背景。下面给出了其他一些例子。

```
>> set(0, 'DefaultAxesFontSize',14)     % larger axes fonts - all figures
>> set(gcf, 'DefaultAxesLineWidth',2)   % thick axis lines – this figure only
>> set(gcf, 'DefaultAxesXColor', 'y')   % yellow X axis lines and labels
>> set(gcf, 'DefaultAxesYGrid', 'on')   % Y axis grid lines - this figure
>> set(0, 'DefaultAxesBox', 'on')       % enclose axes – all figures
>> set(gca, 'DefaultLineLineStyle', ': ') % dotted linestyle – these axes only
```

当一个默认属性发生改变时，只有在这个改变以后生成的对象才会使用新的默认属性。而已经存在的对象仍然保持原来的默认属性。

在对已经存在的对象进行操作的时候，人们总是希望在使用了这些对象之后将它们恢复到原来的状态。如果用户在一个M文件中改变了对象的默认属性，请保存原来的设置并且在退出程序的时候恢复这些设置，例如：

```
oldunits = get(0, 'DefaultFigureUnits');
set(0, 'DefaultFigureUnits', 'normalized');
    <MATLAB statements>
set(0, 'DefaultFigureUnits',oldunits);
```

为了使得Matlab在任何时候都使用用户定义的默认值，只需要在startup.m文件中包含所希望的set命令。例如：

```
>> set(0, 'DefaultAxesXGrid', 'on')
>> set(0, 'DefaultAxesYGrid', 'on')
>> set(0, 'DefaultAxesZGrid', 'on')
>> set(0, 'DefaultAxesBox', 'on')
>> set(0, 'DefaultFigurePaperType', 'A4')
```

这些命令使得所有的坐标轴都带刻度，并且将坐标轴边框打开，将默认的纸张大小设置为A4。在根一级所设置的默认值将会影响到所有图形窗口中的每一个对象。

有三个特殊的属性值字符串，它们取消，覆盖或者查询用户定义的默认属性。这些字符串为'remove'，'factory'和'default'。如果用户已经改变了一个默认属性，用户可以用'remove'来取消这次改动，因此就将这个属性重新设置为它原来的默认属性，例如：

```
>> set(0, 'DefaultFigureColor',[.5 .5 .5])  % set a new default
>> set(0, 'DefaultFigureColor', 'remove')   % return to MATLAB defaults
```

为了临时覆盖一个默认属性，并且对一个特定的对象使用原来的Matlab默认值，可以

使用特定的属性值'factory'，例如：

```
>> set(0, 'DefaultFigureColor',[.5 .5 .5])% set a new user default
>> figure('Color', 'factory')       % figure using default color
```

第三个特殊的属性值字符串是'default'。这个值强迫Matlab沿着对象层次结构向上搜索，直到它找到了所需属性的默认值为止。如果找到了这个默认值，它就使用这个默认值。如果已经到达了根对象，但是没有找到用户定义的默认值，就使用Matlab工厂级默认值。当用户使用非默认属性生成对象，并希望将该属性重新设为默认属性时，此特性非常有效。例如：

```
>> set(0, 'DefaultLineColor', 'r')      % set default at the root level
>> set(gcf, 'DefaultLineColor', 'g')    % current figure level default
>> Hl_rand = plot(rand(1,10));      % plot a line using'ColorOrder' color
>> set(Hl_rand, 'Color', 'default')     % the line becomes green
>> close(gcf)                           % close the window
>> Hl_rand = plot(rand(1,10));% plot a line using 'ColorOrder' color again
>> set(Hl_rand, 'Color', 'default')     % the line becomes red
```

请注意，命令plot没有用线条对象的默认值作为线条颜色。如果没有声明颜色参数，那么plot命令就用坐标轴的'ColorOrder'属性来设定它生成的每一个线条的颜色。

通过输入下面的指令可以获得工厂级默认值的列表：

```
>> get(0, 'factory')
```

可以通过输入下面的指令来获得对象层次结构中任何一层所设置的默认属性：

```
>> get(handle, 'default')
```

根对象包含了大量颜色属性的默认值以及图形在开启时候的位置，例如：

```
>> get(0,'default')
ans =
                  defaultTextColor : [0 0 0]
                 defaultAxesXColor : [0 0 0]
                 defaultAxesYColor : [0 0 0]
                 defaultAxesZColor : [0 0 0]
             defaultPatchFaceColor : [0 0 0]
             defaultPatchEdgeColor : [0 0 0]
                  defaultLineColor : [0 0 0]
       defaultFigureInvertHardcopy : 'on'
                defaultFigureColor : [0.8 0.8 0.8]
                  defaultAxesColor : [1 1 1]
             defaultAxesColorOrder : [7×3 double]
             defaultFigureColormap : [64×3 double]
          defaultSurfaceEdgeColor  : [0 0 0]
             defaultFigurePosition : [920 620 672 504]
```

其他默认属性只有在被用户创建之后才会显示在列表中，例如：

```
>> get(gcf, 'default')
```

```
ans =
0×0 struct array with fields:
>> set(gcf, 'DefaultLineMarkerSize',10)
>> get(gcf, 'default')
ans =
    defaultLineMarkerSize:10
```

30.9 通 用 属 性

所有的句柄图形对象都共享一组对象属性，如下表所示。也可以参见附录A。

属性	描述
BeingDeleted	在想把对象删除的时候，将其设置为'on'。在Matlab 6中没有文档记录
ButtonDownFcn	当鼠标在对象上按下的时候，在Command窗口中用eval求出的字符串回调。回调通常是一个函数调用
Children	可见的子对象句柄
Clipping	激活或者禁用坐标轴子对象的删减
CreateFcn	在一个对象被创建之后立即在Command窗口中用eval求出的字符串回调。回调通常是一个函数调用。
DeleteFcn	在一个对象被删除之前在Command窗口中用eval求出的字符串回调。回调通常是一个函数调用。
BusyAction	决定此对象的回调被其他回调中断的方式
HandleVisibility	决定此对象句柄是否在命令窗口中或者是执行回调的过程中可见
HitTest	决定对象是否能够用鼠标选定并成为当前对象
Interruptible	决定对这个对象的回调是否可以被中断
Parent	父对象的句柄
Selected	确定某个对象是否已经被选为当前对象
SelectionHighlight	确定一个选定的对象是否显示可见的选择句柄
Tag	用户定义的字符串，用来标识对象或者给对象加标签。通常与findobj一起使用。例如，findobj(0, 'tag', 'mytagstring')。
Type	标识对象类型的字符串
UIContextMenu	与一个对象有关的上下文菜单的句柄
UserData	储存与一个对象有关的所有用户定义的变量
Visible	对象是否可见

这些属性中，有三个属性包含了回调字符串：'ButtonDownFcn'，'CreateFcn'和'DeleteFcn'。回调字符串，或者简称为回调，是传递给函数eval并在Command窗口工作区中执行的字符串。属性'Parent'和'Children'包含了层次结构中其他对象的句柄。如果'Clipping'为'on'（这是除了文本对象之外所有对象的默认值），那么绘制在一个坐标轴系中的对象就在坐标轴限的地方被剪切掉了。如果后来又发生了一次回调，那么'Interruptible'和'BusyAction'控制原来

进行的回调。'Type'是一个设定了对象类型的字符串。如果这个对象是图形的'CurrentObject'，那么对象的'Selected'属性值就为'on'，并且'SelectionHightlight'决定了在对象被选中的时候，是否改变其外观。'HandleVisibility'属性设定了对象句柄是可见，不可见或者只是对回调可见。如果有必要，根对象的'ShowHiddenHandles'属性将覆盖所有对象的'HandleVisibility'属性。如果'Visible'属性被设置为'off'，那么这个对象将会从显示中消失，其实它仍旧在原处，并且它的对象句柄仍旧有效，但是不会被重新绘制。将'Visible'设置为'on'将会在显示上重新绘制这个对象。属性'Tag'和'UserData'是为用户保留的。'Tag'属性通常被用来用一个用户定义的字符串来给一个对象加上标签。例如：

```
>> set(gca, 'Tag', 'My Axes')
```

这条命令给当前图形中的当前坐标轴加上标签'My Axes'。这个字符串并不显示在坐标轴系或者当前的图形中，但是用户可以通过查询'Tag'属性来标识出这个对象。例如，在有很多个坐标轴的时候，用户可以通过输入如下的语句来找出上边这个坐标轴对象的句柄。

```
>> Ha_myaxes = findobj(0, 'Tag', 'My Axes');
```

属性'UserData'可以包含用户想要放置的任何变量。字符串，数字，结构，甚至多维单元数组都可以保存在任何对象的'UserData'属性中。没有Matlab函数改变或者预设本属性中所包含的值。

用命令get和set所列出的各个对象的属性都是在文档中出现的属性。也有一些Matlab的开发人员使用的没有在文档中出现或者是隐藏的属性。其中的一部分是可以进行修改的，而另一些是只读的。没有在文档中出现的属性仅仅是隐藏了看不见而已。这些属性仍旧存在的并且可以进行修改。没有在文档中出现的根对象属性'HideUndocumented'控制get是返回所有的属性还是只返回在文档中出现的属性。例如：

```
>> set(0, 'HideUndocumented', 'off')
```

这条语句使得没有出现在文档中的属性在Matlab中也可见。因为没有在文档中出现的属性是有意不在文档中出现的，因此用户在使用它们的时候务必非常小心。它们有的时候不如在文档中出现的属性那样具有很强的鲁棒性，并且经常发生变化。在Matlab以后的版本中，没有在文档中出现的属性可能会显示，不显示，改变功能，或者甚至变成为在文档中出现的属性。

30.10 新的图形

当一个新的图形对象用一个诸如line或者text这样的低级命令创建的时候，这个对象默认地出现在当前图形的当前坐标轴上。但是，高级图形函数，比如mesh和plot，却在显示一个图形之前将当前的坐标轴清除，并且将大多数的坐标轴属性重新设置成它们的默认值。正像先前讨论的那样，命令hold可以用来改变这个默认动作。图形和坐标轴都有一个属性'NextPlot'，它被用来控制Matlab如何使用现存的图形和坐标轴。函数hold，newplot，reset，clf和cla都会影响图形和坐标轴的'NextPlot'属性。'NextPlot'有三个可能的值，它们是：

```
>> set(gcf, 'NextPlot')
    [ {add} | replace | replacechildren ]
>> set(gca, 'NextPlot')
    [ add | {replace} | replacechildren ]
```

figure的默认设置为'add'；axes的默认设置为'replace'。当'NextPlot'被设置为'add'的时候，不需要清除或者重新设置当前的图形或者坐标轴，就可以添加一个新的图形。当'NextPlot'被设置为'Replace'的时候，一个新的对象的生成就会导致在绘制这个新的图形之前，图形或者坐标轴将它们所有的子对象清除，并且将除了'Position'和'Units'之外的所有属性重新设置为它们的默认值。这个默认设置清除并重置当前坐标轴，然后重新使用当前图形。

'NextPlot'的第三个可能的设置为'replacechildren'。这个设置将所有的子对象清除，但是不会改变当前的图形或者坐标轴属性。命令hold on将图形和坐标轴的'NextPlot'属性都设置为'add'。命令hold off将坐标轴的'NextPlot'属性设置为'replace'。

函数newplot根据上边'NextPlot'指定的方式生成一个新的坐标轴。调用此函数来生成一个坐标轴。用诸如line，patch等对象的生成函数所生成的子对象将包含了此坐标轴中。这和诸如plot和surf等高级图形函数形成了对比。函数newplot执行的代码类似于下面这段代码。

```
function Ha = newplot
Hf = gcf;                           % get current figure or create one
next = lower(get(Hf,'NextPlot'));
switch next
    case 'replacechildren',clf;     % delete figure children
    case 'replace',clf('reset');    % delete children and reset properties
end
Ha = gca;                           % get current axes or create one
next = lower(get(Ha, 'NextPlot'));
switch next
    case 'replacechildren',cla;     % delete axes children
    case 'replace',cla('reset');    % delete children and reset properties
end
```

30.11 绘 图 速 度

在有很多应用程序中，用户可能希望当一个图形解的一个或更多的属性发生改变时，能够显示变化情况；或者希望根据鼠标的反馈动态地修改一个图形图像。在这两种情况下，将屏幕绘图的速度最大化并且消除图像抖动都是用户希望的。Matlab的图形，坐标轴，线条，碎片，矩形和表面对象都提供了一些影响绘图速度和图像抖动的属性。

Matlab的图形有三个影响绘图速度的属性：'Renderer'，'DoubleBuffer'和'BackingStore'。'Render'指的是用来在屏幕上生成或者绘制一个图形图像所用到的底层算法。这个属性有三个可能的值，'painters'，'zbuffer'和'openGL'。在通常的情况下，Matlab根据要绘制的图形对象的复杂度和所使用的计算机的性能来自动地决定使用哪一种绘图算法。在大多数情况下，

Matlab会为每个图形选择最优的绘图算法。但是，在特定的情况下，绘图算法可以通过将这个属性设定为希望的值来进行设定。

'BackingStore'通常被设置为'on'，它控制图形窗口数据的副本的存在与否。当一个计算机屏幕上显示多个重叠的窗口的时候，将'BackingStore'设置为'on'允许当隐藏的或者部分隐藏的图形窗口被选中成为当前显示的窗口的时候，这些窗口能够得到立刻的重绘。当'BackingStore'被设置为'off'的时候，不会生成图形的后台副本，这样增加了绘图的速度，但是在一个图形被选中成为屏幕显示的当前图形的时候，就会强制计算机对图形数据进行重新计算。

图形的'DoubleBuffer'属性通常设定为'off'。将这个属性设置为'on'就是告诉Matlab将图形对象绘制在一个后台缓冲区中，并且当缓冲区被完全更新的时候才将数据绘制在屏幕上。通过将图形绘制在一个后台缓冲区中而不是直接更新屏幕显示，就将绘图过程对用户屏蔽起来了。Matlab将这个属性设置为'off'，因为它只适用于用Painter's算法绘制的图形。另外，它只对简单的包含线条的图形产生不抖动的图形绘制，比如那些用函数plot生成的图形。另外，所绘制的线条必须将其'EraseMode'属性设置为默认的'normal'。包含了表面或者很多碎片的图形往往无法从双缓冲中受益。

当一个坐标轴使用Painter's算法绘制的时候，可以用坐标轴的'DrawMode'属性来控制绘制诸如线条这样的对象所需的幕后工作量。默认地，这个属性被设置为'normal'。但是，将它设置为'fast'将会由于得到更快的绘制速度而牺牲绘制的准确性。对于Z缓冲和OpenGL绘图而言，这个属性没有任何影响。

图形绘制对象线条、碎片、矩形和表面都有一个'EraseMode'属性，这个属性允许用户设定它们应该怎样被擦除和重绘。这个属性的默认值为'normal'，其他候选选项为'none'，'background'和'xor'。当对象由于自身属性的改变或者由于它们的上层又置入了其他的对象而被重画的时候，该属性决定了它的重画方式。例如，当'EraseMode'被设置为'none'的时候，当对象的属性被改变的时候，最初的对象并不会被擦除；只是再绘制一个修改后的对象。在大多数的情况下，这使得坐标轴留下了不希望出现的影子。将'EraseMode'设置为'background'，则通过将其所占用的像素设置为坐标轴系的背景颜色而将初始的对象删除。这会破坏此初始对象下层的对象的外观。将'EraseMode'设置为'xor'将会通过对这个对象下层颜色应用一个异或运算来擦除和重画曲线。这不会破坏被改变对象下层的对象，但是将会影响到这个被改变的对象的颜色。根据这些属性值的描述，将'EraseMode'设置为'none'可以最大限度地加快绘图速度，但是这会导致在大多数情况下不希望出现的阴影。将'EraseMode'设置为'xor'或者'background'提供了在绘图速度和精度之间的一种妥协。在大多数情况下，'xor'是生成动画最好的选择。

30.12 回 调

所有的句柄图形对象都有属性'ButtonDownFcn'，'CreatFcn'和'DeleteFcn'。另外，图形还有属性'CloseRequestFcn'，'KeyPressFcn'，'WindowButtonDownFcn'，'WindowButtonMotionFcn'以及'WindowButtonUpFcn'，而用户界面函数有属性'CallBack'。与这些属性相关的属性值是被

称作回调的字符串。当特定的用户操作发生的时候，这些回调都在Command窗口的工作区中被传递给eval函数并由它执行。因为这个字符串是由eval执行的，因此它们可以包含任何合法的Matlab语句序列。在大多数的情况下，它们是函数调用，通常是对与回调定义相同的函数的调用。这些回调构成了Matlab的图形用户界面特性的基础，我们将在下一章讨论图形用户界面的特性。这些回调字符串的设置告诉Matlab执行某项任务，以对用户进行的操作作出响应。

最简单的回调是关闭请求回调，默认情况下，它是不为空的，例如，

```
>> get(gcf, 'CloseRequestFcn')
ans =
closereq
>> class(ans)
ans =
char
```

默认地，当点击图形工具栏的关闭按钮关闭一个图形窗口的时候，字符串closereq就被传递给eval。这个字符串是Matlab中的一个函数，这个函数仅仅负责删除当前的图形窗口。因此默认地，点击关闭按钮就删除了相关的图形窗口。此行为可以通过在关闭请求中将上面的字符串用其他字符代替来改变。例如：

```
>> set(gcf, 'CloseRequestFcn', '')
```

这个代替的字符串禁止了通过关闭按钮来实现关闭窗口的功能。这个关闭请求函数是一个空字符串，因此不执行任何操作。这个回调字符串可以是任何合法的Matlab语句序列。因此，这个字符串可以在真正关闭窗口之前，提示用户确认这个关闭请求。

30.13 M文件示例

Matlab本身就有很多很多的句柄图形应用的例子。在specgraph目录（>>helpwin specgraph）下几乎所有专用的绘图函数都是由句柄图形函数构成的。即便是M文件函数axis也是通过使用句柄图形函数调用实现的。本节提供了句柄图形用法的更进一步的阐释。

如果一个坐标轴是一个x-y平面的二维视图，函数mmis2d的显示就返回逻辑真。

```
function [tf,xa,ya]=mmis2d(H)
%MMIS2D True for Axes that are 2D.
% MMIS2D(H) returns True if the axes having handle H displays
% a 2D viewpoint of the X-Y plane where the X- and Y-axes are
% parallel to the sides of the associated figure window.
%
% [TF,Xa,Ya]=MMIS2D(H) in addition returns the angles of x- and y-axes
%
% e.g., if the x-axis increases from right-to-left Xa=180
% e.g., if the y-axis increases from left-to-right Ya=0
% e.g., if the x-axis increases from bottom-to-top Xa=90

if ~ishandle(H)
```

```
        error('H Must be a Handle.')
end
if ~strcmp(get(H,'Type'),'axes')
       error('H Must be a Handle to an Axes Object.')
end
v=get(H,'view');
az=v('1'); el=v(2);
tf=rem(az,90)==0 & abs(el)==90;

if nargout==3
       xdir=strcmp(get(H,'Xdir'),'reverse');
       ydir=strcmp(get(H,'Ydir'),'reverse');
       s=sign(el);

       xa=mod(-s*az - xdir*180,360);
       ya=mod(s*(90-az) - ydir*180,360);
end
```

本函数使用了函数ishandle**，该函数对于合法的对象句柄参数返回逻辑真。它通过获得属性**'Type'**的值来查看所提供的句柄是不是一个坐标轴的句柄。如果是，它再获得**'View'**属性来决定所要求的输出。**

下面所显示的函数mmgetpos**找出以特定单位表示的某个对象的位置。这个函数只要能获得当前的**'Units'**属性，就能够正常地运行，将单位设置为期望输出的**'Units'**属性值，获取期望单位的**'Position'**属性，然后重置**'Units'**属性的值。**

```
function p=mmgetpos(H,u,cp)
%MMGETPOS Get Object Position Vector in Specified Units.
% MMGETPOS(H,'Units') returns the position vector associated with the
% graphics object having handle H in the units specified by 'Units'.
% 'Units' is one of: 'pixels', 'normalized', 'points', 'inches', 'cent',
% or 'character'.
% 'Units' equal to 'data' is valid for text objects only.
%
% MMGETPOS does the "right thing", i.e., it: (1) saves the current units,
% (2) sets the units to those requested, (3) gets the position, then
% (4) restores the original units.
%
% MMGETPOS(H,'Units','CurrentPoint') returns the 'CurrentPoint' position
% of the figure having handle H in the units specified by 'Units'.
%
% MMGETPOS(H,'Units','Extent') returns the 'Extent' rectangle of the text
% object having handle H.
%
% 'Uimenu', 'Uicontextmenu', 'image', 'line', 'patch',  'surface',
% 'rectangle' and 'light' objects do NOT have position properties.

if~ischar(u), error('Units Must be a Valid String.'), end
if~ishandle(H), error('H is Not a Valid Handle.'), end
Htype=get(H,'Type');

if nargin==3 & ~isempty(cp) & ischar(cp)
       if strcmp(Htype,'figure') & lower(cp(1))=='c'
              pname='CurrentPoint';
       elseif strcmp(Htype,'text') & lower(cp(1))=='e'
```

```
            pname='Extent';
        else
            error('Unknown Input Syntax.')
        end
elseif H~=0
        pname='Position';
elseif H==0  % root object
      pname='ScreenSize';
else
        error('Unknown Input Syntax.')
end
hu=get(H,'units');
set(H,'units',u)
p=get(H,pname);
set(H,'units',hu)
```

下面显示得mmzap**函数展示了一项技术，它再写句柄图形函数**M**文件的时候非常有用。它与**waitforbuttonpress**和**gco**联合使用来获取用鼠标选中的对象的句柄。**waitforbuttonpress**是一个内嵌的**Matlab**函数，它等待一个鼠标点击或者键盘按键按下。它的帮助文档如下：**

```
>> help waitforbuttonpress

WAITFORBUTTONPRESS Wait for key/buttonpress over figure.
    T = WAITFORBUTTONPRESS stops program execution until a key or mouse button
    is pressed over a figure window. Returns 0 When terminated by a mouse
    buttonpress, or 1 when terminated by a keypress.Additional information
    about the terminating event is available from the current figure.

    See also GINPUT,GCF.
```

在鼠标指针在一个图形上的时候，按下鼠标，gco**就返回被选中的对象的句柄。这个句柄然后就被用来操作被选中的对象。**

```
function mmzap(arg)
%MMZAP Delete Graphics Object Using Mouse.
% MMZAP waits for a mouse click on an object in
% a figure window and deletes the object.
% MMZAP or MMZAP text erases text objects.
% MMZAP axes erases axes objects.
% MMZAP line erases line objects.
% MMZAP surf erases surface objects.
% MMZAP patch erases patch objects.
%
% Clicking on an object other than the selected type,or striking
% a key on the keyboard aborts the command.

if margin<1,arg='text';end

Hf=get(O,'CurrentFigure');
if isempty(Hf)
    error('No Figure Window Exists.')
end
if length(findobj(O,'Type','figure'))==1
        figure(Hf) % bring only figure forward
end
```

```
key=waitforbuttonpress;
if key  % key on keyboard pressed
       return
else    % object selected
       object=gco;
       type=get(object,'Type');
       if strncmp(type,arg,4)
              delete(object)
       end
end
```

Matlab中的函数xlim，ylim和zlim允许用户分别来设置和获取图形的三个坐标轴的坐标轴限。刻度线没有等价函数。函数grid将所有三个坐标轴的刻度线打开和关闭。为了对此特性加以补充，下面显示的mmxgrid允许用户将x轴的刻度线打开和关闭。将mmxgrid函数进行很小的改动，也可以为y轴和z轴生成类似的函数。

```
function y=mmxgrid(arg)
% MMXGRID X-axes Grid Lines.
% MMXGRID ON adds grid lines along the X-axes of the
% current axes.
% MMXGRID OFF turns them off.
% MMXGRID by itself toggles the X-axes grid state.
%
% TF=MMXGRID returns logical True if the X-axes grid is ON.
% Otherwise it returns logical False.
%
% See also GRID, XLIM, XLABEL

if nargin~=0
    if ischar(arg)
        if length(arg)>1 & strncmpi(arg,'on',2)
            set(gca,'XGrid','on')
        elseif length(arg)>1 & strncmpi(arg,'off',3)
           set(gca,'XGrid','off')
        else
            error('Unknown Input Argument.')
        end
    else
        error('Character Input Argument Required.')
  end
elseif nargout~=0
   y=strcmp(get(gca,'XGrid'),'on');
else
    if strcmp(get(gca,'XGrid'),'on')
        set(gca,'XGrid','off')
   else
        set(gca,'XGrid','on')
   end
end
```

30.14 小 结

句柄图形函数提供了精调Matlab可视化部件外观的功能。每个图形对象都有一个与其相关的句柄，这个句柄可以用来处理这个对象。下面这个表格列出了Matlab中的句柄图形函数。

函数	描述
get	获取对象属性
set	设置对象属性
gcf	获取当前图形
gca	获取当前坐标轴
gco	获取当前对象
findobj	寻找含有指定属性的对象
findall	寻找含有指定属性的隐藏和非隐藏对象
allchild	获得一个对象的隐藏和非隐藏自对象的句柄
copyobj	将对象拷贝到一个新的父对象
root	计算机根对象，句柄=0
figure	图形对象创建
axes	创建坐标轴对象
line	创建线条对象
text	创建文本对象
patch	创建碎片对象
rectangle	创建矩形对象
surface	创建表面对象
image	创建图像对象
light	创建光照对象
uicontrol	创建用户界面控件对象
uimenu	创建用户界面菜单对象
uicontextmenu	创建用户界面上下文菜单对象
reset	将对象属性重置为默认值
clf	清除当前图形
cla	清除当前坐标轴
ishandle	参数如果是对象句柄，就返回真
delete	删除对象
close	用关闭请求函数关闭图形
refresh	图形刷新
gcbo	获得当前回调对象
gcbf	获得当前回调图形
closereq	默认的图形'CloseRequestFcn'回调
newplot	根据'NextPlot'属性的设置来生成新的坐标轴

第 31 章　图形用户界面

本章介绍了Matlab中提供的图形用户界面特性。这些特性包括菜单、上下文菜单、按钮、滚动条、单选按钮、拨动按钮，弹出式菜单和列表框。另外，还可以跟踪鼠标的位置和运动。这些特性非常广泛和详细，完全可以用一整本书来专门讲述这些特性。但对本书而言，这样做是不可能的，因此关于这个部分，本书能作的就是给出一个有价值的介绍和几个能说明问题的示例。

31.1　什么是图形用户界面（GUI）

用户界面就是用户和计算机之间或者用户和计算机程序之间进行通讯的场所和实现交互的方法。它是计算机和用户用来交换信息的方法。计算机在计算机屏幕上显示文本和图形，并且还有可能利用扬声器发出声响。用户用诸如键盘、鼠标、跟踪球、绘图板或者麦克风这样的输入设备和计算机进行通讯。用户界面定义了计算机、操作系统或者说应用程序给人的外观和感觉。通常，用户在选择计算机或程序的时候，是以设计是否让人觉得舒服以及它的用户界面功能是否高效来决定的。

图形用户界面，或者说GUI（发音为goo'ey），是一个整合了诸如窗口、图标、按钮、菜单和文本这样的图形对象的用户界面。以某种方式选中或者激活这些对象通常会导致某个动作或者变化的发生。最常用的激活方法就是用鼠标或者其他点击设备来控制屏幕上指针的移动，以及按下鼠标按键来发出一个或者选中对象其他动作的信号。

和上一章中讨论的Matlab的句柄图形功能使用户可以定制Matlab显示信息的方式一样，本章中讲述的句柄图形用户界面函数使得用户可以定制用户和计算机交互的方式。

本章展示了句柄图形uicontrol，uimenu和uicontextmenu为Matlab函数和M文件添加图形用户界面的方法。uimenu对象在图形窗口中生成下拉菜单和子菜单。Uicontrol对象生成诸如按钮、滑尺、弹出式菜单和文本框这样的对象。uicontextmenu对象生成在对象上打开的上下文菜单。

使用demo命令，就可以看到一个Matlab中GUI功能很好的例子。为了浏览这些演示，只需要输入下面的命令就可以了。

```
>> demo
```

31.2　GUI应由谁创建及为何创建

在运行了demo之后，用户可能会问自己：“为什么我要在Matlab中创建一个GUI？”这个问题问得很好！最简单的回答就是：用户也可以不创建GUI。很多用户主要是用Matlab作数据分析、问题求解和结果显示，他们可能会认为不值得为了GUI工具去花费那么大的

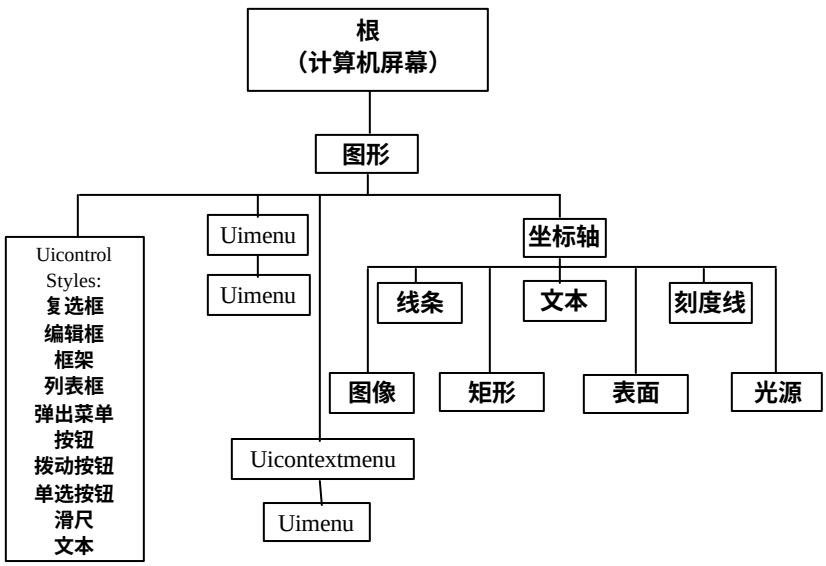
精力。但是，GUI可以用来在Matlab中生成非常高效的工具，或者用来将用户自己的工作交互式的展示出来。最常见的生成一个图形用户界面的例子如下：

- 用户正在编写一个函数，这个函数将被重复使用多次，而菜单、按钮或者文本框就是非常合适的输入方法。
- 用户正在编写一个函数或者一个应用程序供他人使用。
- 用户想要生成一个过程、技术或者分析方法的交互式展示程序。
- 用户认为GUI不错，因此想要试着使用一下。

在我们开始之前，请记住，在Matlab中设计和实施一个GUI的先决条件就是对句柄图形要有基本的了解。如果用户跳过了前一章，那么最好还是现在就回到前一章并仔细地阅读一遍。读一下Building GUIs with Matlab这本手册也是很有帮助的。这本相对而言较小的手册包含了关于良好的用户界面设计的有价值的基础信息，以及一组展示了Matlab中GUI编程的可以广泛应用的例子。

31.3 GUI对象层次结构

正如在前一章所展示的那样，用图形命令生成的都是图形对象。这些对象包括uimenu，uicontrol和uicontextmenu对象以及图形，坐标轴及其子对象。下面给出的这个对象层次结构图表明计算机屏幕本身就是根对象，并且图形是这个根对象的子对象；坐标轴，uiminu，uicontrol和uicontextmunu都是图形的子对象。



根对象可以包含一个或者多个图形，而每个图形又可以包含一套或者多套坐标轴及其子对象。每个图形也可以包含一个或者多个独立于坐标轴的uimenu，uicontrol和uicontextmenu。虽然uicontrol对象没有子对象，但却有多种类型。而且，uimemu和uicontextmenu对象通常都以uimenu对象为子对象。

在每台运行Matlab的计算机平台上所生成的图形显示效果是不同的。Unix工作站使用X-Window系统，它有多个窗口管理器（比如Motif和CDE）来控制显示的布局。PC依赖于Microsoft Windows或者Windows NT来进行窗口管理。尽管各个不同的平台上的显示看上去会有区别，但是在大多数情况下句柄图形的代码却是一样的。Matlab在内部处理平台和窗口系统之间的差异涉及到句柄图形代码的函数，包括那些使用了uimenus，uicontrols和uicontextmenus对象的函数，它们通常能够在所有的平台上运行。在有明显差异的地方，本章中的后边部分都会将其指出。

31.4　菜　单

每个视窗系统都用到了菜单来使得用户可以选择命令和选项。通常，在一个窗口的顶部有一个从左到右贯穿的菜单栏。当用户通过将鼠标指针移动到菜单标签上并用按下鼠标按键的方式选中顶层菜单中的某一个时，就从这个菜单标签上弹出一个菜单项列表。这种类型的菜单通常被称作下拉菜单。通过按下鼠标按键，将鼠标指针移动到一个菜单项上，然后释放这个鼠标按键，就选中了一个菜单项。MS Windows和有的X-Window系统平台提供了选择一个菜单项的另外一种方法。在一个顶层菜单上方按下并释放鼠标按键，或者说点击鼠标按键，就打开了这个下拉菜单。然后将鼠标指针移动到菜单项上方然后再次点击，就选中了一个菜单项。选中下拉菜单中的某一菜单项会导致某个事件的发生。

菜单也可以有子菜单。子菜单项将会在这个子菜单标签的右侧显示出一个小三角形或者一个向前的箭头，这表明这个选项有多个菜单项可供选择。当一个子菜单项被选中的时候，另一个有更多菜单项的菜单被显示在这个菜单项右侧的一个下拉菜单中。这个菜单通常被称作一个移动菜单（walking menu）。选中这个菜单中的某个菜单项也会导致某个事件发生。子菜单是可以嵌套的，嵌套的层次数量只受所使用的视窗系统和可提供的系统资源的限制。

菜单栏位于每个图形窗口的顶端，并且默认地包含一些默认菜单标题。用uimenu对象添加的菜单标题都位于最后一个菜单标题的右侧。但是，可以通过设置图形的'Menubar'属性来隐藏或者显示标准菜单和工具栏，例如：

```
>> set(gcf, 'menubar', 'none')      % hide standard menus
>> set(gcf, 'menubar', 'figure')    % display standard menus
```

31.5　菜单和子菜单生成

菜单项和子菜单项可以用uimenu函数生成，其方式和其他对象生成函数相同。例如：

```
>> Hm_1 = uimenu(Hx_parent, 'PropertyName',PropertyValue,…)
```

这条语句生成一个句柄为Hm_1，由'PropertyName'－PropertyValue对定义的uimenu菜单项。Hx_parent是父对象的句柄，它必须是一个图形或者另一个uimenu的句柄。如果父对象是一个图形，那么这个uimenu就是在这个图形窗口中的一个新菜单。如果父对象是另一个

uimenu，那么这个新的uimenu就是这个父菜单的一个子菜单项。将'WindowStyle'属性设置为'modal'的图形窗口指的就是对话框。因此，uimenus和标准的菜单也存在于对话图形框中，但是它们被隐藏起来，并且被禁用了。

31.6 菜 单 属 性

下表显示了uimenus的有用且独特的句柄图形属性。

Uimenu属性	描述	值，{默认值}
Checked	uimenu菜单项左侧选中标记符号的可见性	'on'或者{'off'}
ForegroundColor	文本标签的颜色	颜色声明向量{[0 0 0]}
Label	菜单标签字符串	字符串
SelectionHighlight	将选中的对象置为高亮	{'on'}或者'off'
Separator	uimenu菜单项上方的分隔符的可见性	'on'或者{'off'}
Visible	uimenu菜单项的可见性。当设置为不可见的时候，这个菜单项就从菜单中消失，但是仍旧存在，并且保持'Position'属性不变	{'on'}或者'off'
Accelerator	uimenu的键盘快捷方式	单个字符
Children	如果子菜单存在，返回子菜单的句柄	向量
Enable	激活或者禁用uimenu。当uimenu被禁用的时候，菜单项可见，但是外观变成了灰色，并且不能被选中	{'on'}或者'off'
Parent	父对象。要么是图形，要么是uimenu或者uicontextmenu	双精度数值
Tag	用户定义的标识符	字符串
Type	对象类型（只读）	'uimenu'
Userdata	用户定义的变量存储	任何数据类型的变量
Position	菜单相对于相同菜单列表中的其他uimenu的相对位置。数1表示菜单顶端	标量
Callback	菜单被选中的时候所执行的字符串	字符串

uimenu对象最重要的属性就是'Label'和'CallBack'属性。'Label'属性值是在菜单条上放置的或者在下拉菜单中用来标识菜单项的可见字符串。'CallBack'属性包含了一个Matlab字符串，当菜单项被选中的时候，就将这个字符串传递给eval执行。有子菜单项的菜单项不执行回调。这些菜单项只显示它们所包含的子菜单。

'Accelerator'和'Label'属性提供了两种不用鼠标而选择菜单项的方法。'Accelerator'属性定义了一些等价于某个uimenu菜单项的控制键。也就是说，如果'x'是这个加速按钮属性的值，那么同时按下Control键和X键将使得相应的uimenu回调被执行，而不用显示菜单。并非所有的字符都可以用作加速按钮；每个操作系统都有为标准菜单保留的加速按钮。

另外，可以通过同时按下Alt键和一个快捷字符按钮来打开一个菜单并选中一个菜单项。这个快捷字符按钮是通过在uimenu的'Label'属性中将一个与（&）运算符加到希望的字符前面来标识的。例如，'&Grid'生成了菜单标签字符串Grid。类似地，'G&rid'生成了菜单标签Grid。当一个菜单被打开的时候，按下键盘上的快捷字符就会执行与那个菜单项相关的回调。

31.7　菜单的外观

很多uimenu属性都控制了菜单的外观。'Checked'控制是否显示一个菜单项左侧的选中符号。对于那些通过选择菜单项来实现切换的特性而言，将这个属性设置为回调结果的一个部分，就使得用户能够知道图形的当前状态。'SelectionHighlight'控制当一个菜单项被鼠标选中的时候是否显示与其他菜单项形成反差的颜色。'Separator'确定是否在菜单项的上方显示一条分隔线。这个属性对于将菜单项进行逻辑分组来说是非常有用的。'Visible'控制uimenu是否可见。当一个菜单或者子菜单不可见的时候，它不被显示出来，但是仍旧存在。'Enable'控制一个菜单项能否被选中。当一个菜单项或者子菜单被禁用的时候，它仍旧可见，但是颜色变成了灰色，并且不能被选中。'Position'控制一个uimenu相对于同一层的其他菜单的位置。uimenu的位置是连续编号的，位置1位于顶端或者最左侧的位置。所有这些外观属性都可以在运行的过程中进行设置，这样就使得菜单可以动态地改变。

31.8　上下文菜单

上下文菜单类似于菜单条，只是它们隶属于一个图形窗口中的图形对象。上下文菜单通常是在一个有隶属关系的上下文菜单的对象上点击鼠标右键打开的。在Matlab中，函数uicontextmenu生成了上下文菜单，然后将这个菜单隶属于对象。uicontextmenus是图形的子对象，尽管它们是隶属于图形中的其他对象的。一旦生成了一个uicontextmenu，它就成为一个或者多个uimenu菜单项的父对象。这些附加的uimenu对象拥有本章前面几节所讨论的所有属性。

下表显示了uicontextmenus的有用而独特的句柄图形属性。

Uicontextmenu属性	描述	值，{默认值}
Visible	uicontextmenu菜单项的可见性。如果要可见，将这个属性设置为'on'	'on'或者{'off'}
Children	其子对象uimenu对象的句柄	向量
Parent	父对象图形对象	双精度数值
Tag	用户定义的标识符	字符串
Type	对象类型（只读）	'uicontextmenu'
UserData	用户定义的变量存储	任何数据类型的变量

上下文菜单是用uicontextmenu函数创建的，其调用方式和其他对象创建函数一样。例

如：

```
>> Huix = uicontextmenu(Hf_parent, 'PropertyName',PropertyValue,…)
```

这条语句生成了一个父对象图形对象句柄为Hf_parent，由'PropertyName'－PropertyValue对定义的uicontextmenu对象。这个uicontextmenu的句柄返回到了变量Huix。这个句柄被用来作为出现在这个上下文菜单中的uimenu对象的父对象。这个句柄还被放在相关图形对象的'UIContextMenu'属性中，当鼠标指针移动到这个图形对象上方的时候，点击鼠标右键，就会弹出这个上下文菜单。

31.9 控 件

控件和菜单都被每个计算机平台的视窗系统应用，使得用户可以执行某项动作或者设置某个选项或属性。控件是诸如图标、文本框以及和菜单一起使用的滚动条这样的图形对象，它们被用来创建用户计算机上的视窗系统和窗口管理器所实施的图形用户界面。

Matlab控件非常类似于窗口管理器所使用的控件。它们是可以被放置在Matlab图形窗口中的任何地方的图形对象，并且可以用鼠标来激活。Matlab控件包括按钮，滑尺，文本框和弹出菜单。用Matlab创建的控件的外观在MS Windows和X-Window系统计算机平台上有轻微的差异，之所以出现这种差异是因为视窗系统在绘制图形的方式上存在差异。但是，控件的功能从本质上来说是一样的，因此相同的Matlab代码在不同的平台上会生成执行相同功能的相同对象。

Matlab中的所有控件都是用函数uicontrol创建的。常用的调用语法类似于前边讨论的uimenu和uicontextmenu的调用语法。例如：

```
>> HC_1 = uicontrol(Hf_fig, 'PropertyName',PropertyValue,...)
```

这条语句生成一个句柄为Hc_1、用'PropertyName'－PropertyValue对定义的uicontrol对象。Hf_fig是父对象图形对象的句柄。

31.10 控 件 属 性

下表显示了uicontrols的有用而独特的句柄图形属性。

uicontrol属性	描述	值，{默认值}
BackgroundColor	uicontrol的背景颜色	颜色声明向量{默认值和系统有关}
Cdata	显示在按钮或者拨动按钮上的真彩色图像	三维RGB数组
ForeGroundColor	文本颜色	颜色声明向量{[0 0 0]}
SelectionHighlight	选中时将对象置为高亮	{'on'}或者'off'
String	uicontrol的文本	字符串
Visible	uicontrol的可见与否	{'on'}或者'off'

续表

uicontrol属性	描述	值，{默认值}
Enable	激活或者禁用uicontrol	{'on'}，'off'或'inactive'
Parent	父对象图形对象	双精度数值
Selected	标识对象是否被选中	'on'或者{'off'}
SliderStep	滑尺或者滚动条的步长	一个2元素向量
Style	uicontrol的类型	'pushbutton', 'togglebutton','radiobutton', 'checkbox','edit', 'text', 'slider', 'frame', 'listbox', 'popupmenu'
Tag	用户定义的标识符	字符串
ToolTipString	显示成工具提示的字符串	字符串
Type	对象类型（只读）	'uicontrol'
UserData	用户定义的变量存储	任何数据类型的变量
Position	对象的大小和位置	[left,bottom,width,height]向量
Units	'Position'属性的单位	{'pixels'},'normalized','inches' 'centimeters','points', 'characters'
FontAngle	字符字体	{'normal'},'italic', 'oblique'
FontName	字体类型名	字符串{默认值和系统有关}
FontSize	字体大小	以FontUnits为单位的大小{默认值和系统有关}
FontUnits	字体大小单位	{'points'},'normalized','inches', 'centimeters','pixels'
FontWeight	文本颜色深度	{'normal'},'light', 'demi', 'bold'
HorizontalAlignment	文本对齐方式	'left', 'center', 'right'
Callback	当uicontrol被选中的时候所执行的字符串	字符串
UIContextmenu	与对象有关的uicontextmenu的句柄	双精度数值
ListBoxTop	显示在一个列表框里的最顶端字符串的索引	标量
Max	最大值	标量
Min	最小值	标量
Value	uicontrol的当前值。提供了当前状态的反馈	标量或者向量

最重要的uicontrol属性是'Style'。这个属性决定了生成的uicontrol的类型。uicontrol的类型决定了对很多属性的解释。

31.11 控 件 类 型

Matlab支持10种uicontrol类型。这些类型都生成一个通常意义上的GUI对象。本节介绍了这些uicontrol类型。

按钮(Pushbutton)。按钮有时候也被称作命令按钮，或者就称作按钮，它是很小的、方形的屏幕对象，它通常包含了一个文本标签。将鼠标指针移动到这个对象上边，然后点击鼠标按键就选中一个按钮，这个动作使得Matlab执行由对象的回调字符串定义的动作。在一个按钮被按下之后，它马上回到其默认的弹起状态。按钮通常被用来执行某个动作，而不是改变一个状态或者设置某个属性。

拨动按钮（Toggle button）。拨动按钮和按钮是一样的，只是当它被按下的时候，是在两个状态：弹起和按下之间切换之外。当拨动按钮被按下的时候，拨动按钮的'Value'属性就被设置为等于'Max'属性所声明的值。当按钮处于弹起状态的时候，它的'Value'就被设置为等于'Min'属性的值。

单选按钮。单选按钮是由包含了一个标签和这个标签文本左边的一个小圆圈或者菱形的按钮构成的。当单选按钮被选中的时候，这个圆圈或者菱形内部就填上了一个小黑点，并且'Value'属性被设置为'Max'属性所声明的值，这个值默认的情况下为1；在没有被选中的时候，这个填充的选中标志就被清除，并且将'Value'属性设置为'Min'属性声明的值，这个值在默认的情况下为0。单选按钮通常被用来选择一组互斥的选项中的某个选项。但是，为了实现这种互斥，每个单选按钮的回调字符串都必须通过将所有其他单选按钮的'Value'值设定为0或者设定为'Min'属性的值，来将所有其他单选按钮的选中状态清除。但是，这只是一个约定。如果需要，可以将单选按钮当作复选框使用。

复选框。复选框是由包含了一个标签和这个标签文本左侧的一个小正方形方框的按钮构成的。当复选框被激活的时候，这个控件在选中和不选中这两种状态之间切换。当复选框被选中的时候，这个小的正方形方框里边被填充以X号（具体填充什么符号是由平台决定的），并且'Value'属性被设置为'Max'属性声明的值，这个值默认的情况下为1；当选中状态被清除的时候，这个小的正方形方框里边的符号被清除，'Value'属性被设置为'Min'属性所声明的值，这个值默认的情况下为0。复选框通常被用来标识一个选项或者属性的状态。它们通常是独立的对象，但是如果有必要，可以把复选框当作单选按钮使用。

编辑框。可编辑的文本框在一个编辑框里显示文本，这样用户可以动态地修改或者替换文本字符串，就像用户使用一个文本编辑器和字处理程序一样。这样，这个新的文本字符串就成了这个uicontrol的'String'属性的值。可编辑的文本框通常允许用户输入文本或者一个数值。可编辑文本框可以包含一行或者多行文本。一个单行的可编辑文本框只接受用户输入的单行文本，而一个多行文本框接受多行文本。单行文本输入是通过输入回车键结束的。多行文本输入是通过Control+回车键结束的。多行文本框是通过将'Max'和'Min'属性设置成使得Max-Min>1的数字来创建的。'Max'属性的值并没有声明最大的行数。多行文本框可以有无限多行。多行字符串可以被声明为一个字符串单元数组或者一个字符数组。

文本框。静态文本框是只显示一个由'String'属性所确定的文本字符串的控件。静态文本框通常被用来显示标签、用户信息或者当前的值。静态文本框是静态的，也就是说用户

不能动态地改变所显示的文本。只能通过改变'String'属性的值来改变所显示的文本。文本字符串位于文本框顶部的中央。长于文本框宽度的文本字符串被自动换行；也就是说，在可以进行断行的地方自动地断行，这样就显示出多行文本。如果文本框的高度相对于文本字符串来说过小了，那么有的文本就不可见。多行字符串可以声明为一个字符串单元数组或一个字符数组。

滑尺。滑尺，或者称为滚动条，是由三个不同的部分构成的：滑槽，或者说是标识了对象的有效值范围的长方形区域；滑槽中代表了滑尺的当前值的指针；以及在这个滑槽两端的箭头符号。滑尺通常被用来选择一个范围里边的一个值。可以用三种方式来设置滑尺的值。第一种，可以将鼠标指针移动到滑尺指针的上方，按下鼠标按键不放，然后移动鼠标，当滑尺指针到达期望位置的时候，释放鼠标按键。第二种方法就是当鼠标指针位于滑槽内，并且位于滑尺指针的一侧时，单击鼠标按键。这个滑尺指针就在那个方向上移动一个默认的步长，这个步长等于这个滑尺的整个范围的大约10%。第三种方法，就是用鼠标点击滑尺某一端的箭头，这使得滑尺在这个箭头的方向上默认地移动整个滑尺范围的1%。滑尺通常是和单独的文本对象一起使用的，这些文本对象用来显示标签、当前的滑尺值以及范围限制。

uicontrol的'Position'属性包含了以'Units'属性所指定的单位为单位的常见[left bottom width height]向量。滑尺的方向取决于width对heigth的比值。如果width>heigth，就画出一个水平的滑尺，而如果width<heigth，就画出一个垂直的滑尺。在X-Window系统平台上，只有在某一维比另一维的四倍还大的时候才显示箭头。在其他平台上，所有的滑尺都有箭头。

'SliderStep'属性是一个两元素向量[arraow_step trough_step]，这个向量控制当鼠标点击一个滑尺箭头或者点击滑槽内部的时候滑尺值的变化。默认值为[0.01 0.10]，这分别表示变化量为最大滑尺值的1%和10%。请注意，Matlab强制要求滑尺的Max>Min，并且Max位于垂直滑尺的顶端和水平滑尺的右端。如果用户需要某些特殊的滑尺，例如，需要一个值0在上而值10在下的垂直滑尺，就需要使用负的'Min'和'Max'，还要使用负的'Value'，并且根据需要用abs函数将这些数据转化成绝对值。

框架。框架uicontrol对象仅仅是不透明的，有阴影的带边框的方形区域。框架类似于uimenu对象的'Separator'属性，其相似性在于它们都提供了视觉上的分离效果。框架通常被用来对单选框或者其他uicontrol对象进行逻辑分组。必须在其他对象放入框架之前对框架进行定义。否则，这个框架就有可能覆盖其他uicontrol控件。

列表框。列表框uicontrol对象看上去就像允许用户用鼠标点击来选择单个或者多个列表条目的多行文本框。各个列表条目是用一个字符串单元数组，一个在字符数不足的地方填补了空格的字符串数组，或者一个单一的用竖线'|'来分开列表条目的字符串来声明的。当一个列表条目被鼠标点击选中的时候，'Value'属性值就用这个被选中的条目的索引代替。

如果Max-Min>1，就可以选择多个列表条目。连续的列表条目可以通过在希望的条目上拖动鼠标来选中，或者通过用鼠标点击第一个条目，然后按住键盘上的Shift键，再用鼠标点击第二个条目来选中。非连续的条目可以通过按住Control键，然后用鼠标点击需要的各个条目的方法来选中。如果选中了多个条目，属性'Value'的值就用一个被选中条目的索引的向量来代替。

属性'Position'声明了列表框的大小和位置。如果列表框的'String'值超出了列表框

uicontrol的width或者height，Matlab就会根据需要给列表框添加水平或者垂直滚动条。

当一个鼠标键的按下－释放事件改变了'Value'属性的值的时候，列表框回调字符串就被执行。如果用户想要允许多项选择并且只在所有的选择都完成了之后才动作，就不要使用列表框'Callback'字符串。在这种情况下，就需要添加一个Done或者Apply按钮并且使用这个按钮的'Callback'字符串。

另外，鼠标双击可以被理解成回调的一部分。列表框uicontrols将图形的'SelectionType'属性设置成'normal'，表示鼠标单击，而设置成'Open'表示为鼠标双击。uicontrol回调可以被设计成只在一个鼠标双击之后才执行它所代表的函数，这个鼠标双击表明已经作出了最后选择。

弹出菜单。弹出菜单通常被用来向用户展示一列互相排斥的选项。这些菜单可以位于图形窗口的任何位置。在弹出菜单关闭的时候，这个弹出菜单就显示成一个方形或者一个按钮，它包含了当前选择的标签，在这个标签的旁边有一个很小的突起的方框或者一个指向下的箭头，表明对象是一个弹出菜单。当鼠标指针移动到一个弹出菜单控件上方并按下鼠标按键的时候，就出现了其他选项。将鼠标指针移动到不同的选项上然后释放这个鼠标按键，就关闭了这个弹出菜单，并显示出这个新的选择。MS Windows和有的X-Window系统平台允许用户点击一个弹出菜单来打开它，然后点击另一个选项来选择新的选项。

当弹出菜单项被选中的时候，'Value'属性就被设置为选择向量中这个被选中元素的索引。选项标签可以声明为一个字符串单元数组、一个字符数组或者一个用竖线'|'隔开的单一字符串。

弹出菜单的'Position'属性包含了我们所熟悉的[left bottom width height]向量，其中width和height值决定了这个弹出菜单对象的大小。在X-Window系统中，这两个数表示的是被关闭的弹出菜单的大小。当弹出菜单打开的时候，弹出菜单被扩展到尽可能地在屏幕的大小范围内显示所有的选项。在MS Windows系统中，height值基本上都被省略了。这些系统生成一个弹出菜单，这个弹出菜单足够的高，从而能够显示一行文本，而根本不考虑height的值。

31.12 控件大小和字体选择

uicontrol的'Position'和'Units'属性被用来定位图形窗口中的对象。给定某个有指定大小和期望字体uicontrol的图形窗口，GUI布局是一个二维几何学中的问题。在设计一个GUI的时候，用户必须使得uicontrol足够的大，以便可以显示所有希望显示的文本，用户还需要决定是否让这个GUI可以改变大小。

Matlab的默认uicontrol字体在每个平台上都能有最好的显示效果，因此我们建议用户使用默认的uicontrol字体。便携式的或者是平台无关的代码也在使用默认字体的情况下有最好的效果。

通常，给uicontrols指定大小和位置是一个不断的尝试和调整的过程。即便是用户已经对得到的大小和位置满意了，这个图形在另一个平台上的外观也许也会有很大的不同，因而还需要进行进一步的调整。通常，让uicontrols比必要的外观稍微大一些是比较合适的，

这样做的目的仅仅是为了使得uicontrols在所有的平台上都是可读的。

一个图形有默认的大小，但是并不能保证所有的图形在所有的平台上都是那个大小。如果用户向一个现存的图形中加入了uicontrols或者uimenus，那么这个图形就有可能比默认的大小要更小或者更大一些。另外，用户可以在任何时候改变任何图形的大小，除非用户通过将图形的'Resize'属性设置为'off'来禁止这样做。因此，总的来说，专门设置一个用作GUI的图形窗口的大小是一个不错的主意。

在给一个可改变大小的图形窗口添加uicontrols的时候有两件事情需要注意，这两件事情是'Units'属性和固定的字体大小字符串所带来的限制。当每个uicontrol的位置是用诸如像素、英寸、厘米或者点这样的绝对单位声明的时候，图形窗口在改变大小的时候就不会改变它们的大小或者位置。这些uicontrols将会保持这个相对于图形窗口左下角的绝对位置。如果图形窗口被变小了，那么有的uicontrol就会位于这个图形窗口之外，并且将不再可见。

当uicontrol的位置是用标准的单位声明的时候，如果图形窗口的大小发生了变化，uicontrols将保持它们彼此间的关系以及它们和图形窗口本身的相对大小和位置。但是，这种情况也有缺点。如果图形窗口变小了，因而就导致uicontrol也变小了，那么标签字符串就可能变成不可读的了，因为字符串的字体大小是固定的。标签的任何超出了uicontrol新尺寸的部分都被剪切掉了。

固定大小的可读字符串和GUI大小之间的矛盾是无法避免的。一个好的GUI布局通常都在uicontrol周围留出足够的空白区域以便当GUI从一个平台移动到另一个平台的时候不会发生标签字符的残缺。如果一个GUI是可以改变大小的，那么这个GUI图形的'ResizeFcn'回调属性就可以用来移动uicontrols并改变uicontrol的大小，来作为对图形窗口大小改变的响应。

31.13　捕获鼠标事件

GUI函数利用鼠标指针的位置和鼠标按键的状态来控制Matlab的动作。本节讨论了鼠标指针和对象位置以及鼠标按键动作之间的交互，以及Matlab如何对变化或者是事件（比如按下鼠标键，鼠标键释放或者鼠标指针移动）作出响应。

所有的句柄图形对象都有一个'ButtonDownFcn'属性。uimenus，uicontextmenus 和 uicontrols 都有一个 'CallBack' 属性，这是它们用法的中心内容。另外，图形还有 'WindowButtonDownFcn'，'WindowButtonUpFcn' 和 'WindowButtonMotionFcn' 属性以及 'KeyPressFcn'，'CloseRequestFcn' 和 'ResizeFcn' 属性。所有的图形对象还有 'CreateFcn' 和 'DeleteFcn' 属性。与这些属性相关的值都是一个回调字符串，当这些属性被激活的时候，这个字符串就被传递给eval执行。鼠标指针的位置决定了当事件发生的时候执行哪些回调，以及这些回调被激活的顺序。

前一章讨论了层叠顺序和与层叠顺序有关的对象选择区域的问题。Matlab根据一个图形中的三个区域来决定哪个回调被激活。当鼠标指针位于一个句柄图形对象由属性 'Position' 确定的范围之内的时候，这个鼠标指针就被认为是在这个对象上。如果鼠标指针不在对象上但是在对象的选择区域里，就说这个鼠标指针在这个对象的附近。最后，如果鼠标指针

在图形窗口内，但是既不在另一个对象上，也不在另一个对象附近，我们就说鼠标指针远离其他对象。当对象或者它们的选择区域重叠的时候，层叠顺序就决定了哪一个对象被选中。

句柄图形线条，表面，碎片，文本和坐标轴对象的选择区域都在前一章进行了讨论。Uimenu对象没有扩展的选择区域。鼠标指针要么在一个uimenu对象上，要么不在。Uicontrol有一个选择区域，这个区域从控件的位置向任何方向上扩展了5个像素。鼠标可以在一个控件上，也可以在这个控件附近。

鼠标单击被定义为当鼠标指针在一个对象上方的时候，按下鼠标键然后紧接着释放这个鼠标键。如果鼠标指针位于一个uimenu，uicontextmenu或者uicontrol对象上，只要对象的'Enable'属性被设置为'on'，那么点击鼠标键就会触发对象的'CallBack'属性字符串的执行。按下鼠标键使得uicontrol做好被触发的准备，并且通常会改变uicontrol或者uimenu的外观，而释放鼠标键就触发了回调的执行。如果鼠标指针不在一个uicontrol或者uimenu上，那么鼠标键按下和释放事件都按照如下的解释进行触发。

当鼠标键被按下的时候，就生成了一个鼠标键按下的事件。当这个事件发生并且鼠标指针位于一个图形窗口内部的时候，可以产生多个不同的动作，具体是什么动作这要取决于鼠标指针的位置和它与哪个句柄图形对象接近。如果某个对象被选中，它就成为当前对象。如果没有对象被选中，那么图形本身就成为当前对象。图形的'CurrentPoint'和'SelectionType'属性也被更新。然后就触发了相应的回调来执行。

下表列出了鼠标指针位置选项和作为对鼠标键按下事件的响应而执行的回调。

鼠标键按下时鼠标指针的位置	所采取的动作
如果'Enable'属性为'on'，且鼠标指针位于一个uimenu菜单项上	改变uimenu的外观并为释放鼠标键的动作做好准备
如果'Enable'属性为'on'，并且鼠标指针位于一个uicontrol上	改变uicontrol的外观并为释放鼠标键的动作做好准备
如果'Enable'属性为'off'，并且鼠标指针位于一个uimenu菜单项上	忽略鼠标键按下事件
如果'Enable'属性为'off'或者'inactive'并且鼠标指针位于一个uicontrol上	执行图形的'WindowButtonDownFcn'回调，然后执行uicontrol的'ButtonDownFcn'回调
在除了uimenu和uicontrol之外的任何句柄图形对象上或者附近	执行图形的'WindowButtonDownFcn'回调，然后执行对象的'ButtonDownFcn'回调
在一个图形内部，但是不在其他任何对象上或者附近	执行图形的'WindowButtonDownFcn'回调，然后执行图形的'ButtonDownFcn'回调

请注意，鼠标键按下事件总会在触发选中对象的'ButtonDownFcn'回调之前触发图形的'WindowButtonDownFcn'回调，只是鼠标指针位于一个uicontrol或者uimenu对象上时例外。当鼠标指针在一个uicontrol附近或者在一个'Enable'属性值为'off'或者'inactive'的uicontrol上的时候，uicontrol的'ButtonDownFcn'回调就在图形的'WindowButtonDownFcn'回调执行完成之后被触发执行，而不是触发执行'CallBack'属性。uimenu的'ButtonDownFcn'回调永远不会被触发。

当鼠标键被释放的时候，就产生了一个鼠标键释放事件。当这个事件发生的时候，图形的'CurrentPoint'属性被更新，并且图形的'WindowButtonUpFcn'回调被触发。如果'WindowButtonUpFcn'回调没有定义，那么在鼠标键释放的时候'CurrentPoint'属性就不被更新。

当鼠标指针被移动到一个图形内部之后，就产生了鼠标指针移动事件。当这个事件发生时，图形的'CurrentPoint'属性就被更新，并且图形的'WindowButtonMotionFcn'回调被触发。如果'WindowButtonMotionFcn'回调没有定义，那么当鼠标指针移动时，'CurrentPoint'属性就不被更新。

31.14　事 件 队 列

因为GUI依赖于用户的动作，因此GUI的执行是以事件队列的概念为基础的。也就是说，GUI展现在用户面前。然后就轮到用户和这个GUI在任何时间以几乎任意方式实现交互，而每一次交互都会生成一个事件被放入到事件队列中。另外，其他涉及到图形窗口输入或者输出的命令也会生成事件。事件包括鼠标指针移动和触发回调的鼠标键事件，waitfor和waitforbuttonpress函数，以及那些诸如drawnow，figure，getframe或者pause这样的重绘图形命令。对于所有这些事件，Matlab都按照它们出现在事件队列中的顺序进行操作。Matlab以一种默认的方式来管理它的事件队列，这种默认的方式在大多数的情况下都适用。而且，通常的句柄图形对象的属性'Interruptible'和'BusyAction'都可以用来声明特殊的动作。

在所有的情况下，一个回调会一直执行，直到它到达了一个waitfor，waitforbuttonpress，drawnow，getframe，pause或者figure命令为止。不包含任何一个这样的命令的回调不能被中断。当程序执行到这些特殊命令中的一个的时候，Matlab就暂停回调的执行，然后检验事件队列中的暂停事件。如果回调被暂停的对象的'Interruptible'属性被设置为'on'（这是这个属性的默认值），那么就将所有未完成的事件处理完毕，然后再恢复这个被暂停的回调的执行。如果'Interruptible'被设置为'off'，那么就只处理没有完成的屏幕刷新事件。同时，如果回调被暂停的对象的'BusyAction'属性被设置为'cancel'，那么中断回调事件就被忽略。如果'BusyAction'属性被设置为'queue'，那么中断回调事件就被保留在事件队列中，直到这个被打断的回调完成再来执行。即便是在一个执行的回调不能被打断的情况下，未完成的屏幕更新事件也会在当回调到达一个waitfor，waitforbuttonpress，drawnow，figure，getframe或者pause命令的时候进行处理。

31.15　回 调 编 程

句柄图形和GUI函数都特别广泛地用到了回调来执行用户的选择。所有这些回调字符串都是用函数eval在命令窗口工作区中执行的。当希望由一个回调执行的动作很简单的时候，比如close(gcbf)，把这个希望的动作本身放到这个回调字符串中是很方便的。但是，在大多数情况下，所期望的回调动作都需要多行Matlab代码，这些代码中的很多都会产生中间变量。将所有这些代码行都写成一个要执行的字符串往往是不明智的。原因如下。

首先，在这个回调内的引号必须成对地出现，这样才能被正确地解释。其次，命令窗口工作区充斥了所有生成的临时变量。再次，回调每执行一次，这个很长的回调字符串就被重新解释一次，而这个过程是比较慢的。最后，这个回调字符串中的语法错误直到回调被执行的时候才会被发现，而不是当这个GUI被生成的时候。

回调编程的一个更好的方法就是带输入参数的调用生成这个GUI自身的函数，这些输入参数惟一地决定了所要求的回调动作。这种方法通常被称作调度编程(switchyard programming)。例如，假设一个GUI函数M文件mygui.m生成了三个按钮Apply，Revert和Done，这些按钮的回调分别为mygui Apply，mygui Revert和mygui Done。请注意，根据命令－函数两重性，mygui Apply和mygui('Apply')是一样的，其它两个命令也是如此。这样，函数mygui就可以用如下的Switch-Case结构来实施这三个回调动作了。

```
function mygui(arg)
%MYGUI Sample Switchyard Programming Example.

if nargin==0
     arg= 'Initialize';
end
switch arg
case  'Initialize'
    % code that creates the GUI and sets the callbacks

case 'Apply'
    % code that performs Apply button callback actions

case 'Revert'
    % code that performs Revert putton callback actions

case  'Done'
    % code that performs Done button callback actions

otherwise
    % report error?
end
```

上边这个代码段是很容易读的，并且没有前边所讲的将回调动作显式地写在回调字符串里边所造成的诸多缺点。利用上边的方法，这个GUI用一个M文件就完成了创建和管理的功能。GUI创建就使得这个函数被编译并且被载入到内存中。这个GUI所构成的用户界面发布回调命令，这些回调命令反复地调用初始化函数。

回调编程最困难的地方是获取数据。回调通常需要来自GUI或者来自另一个图形的特定句柄或数据。利用调度编程方法，每个回调都生成它自己的函数工作区，这个工作区在回调完成了它的动作之后就消失。很幸运的是，Matlab提供了几个方法来获取数据或者将数据传递给回调。

使用函数gcf，gca，gco，gcbo和gcbf，可以很方便访问当前对象。特别地，gcbo和gcbf分别返回当前回调对象和图形的句柄。这些函数应用在一个回调中，提供了对生成这个回调对象和图形的访问。

当还需要其他句柄或者数据的时候，它们可以被储存在对象的'UserData'属性里边，这个对象的句柄被上述的某个函数返回。当保存的数据是一个结构，而这个结构包含了表示不同数据的不同域名时这种方法尤其有用。例如，如果变量x，y和z需要保存，它们可以被

放在一个结构中，像ud.x=x;，ud.y=y;和ud.z=z;。然后set(H, 'UserData',ud)就将结构ud放在了句柄为H的对象的属性'UserData'的存储位置。今后，在一个H已知的回调中，这些数据可以通过输入ud=get(H, 'UserData')得到。

只要用户能保证用户的GUI对'UserData'属性的访问是互斥的，这种利用'UserData'的存储区域的方法就能够使用。将'HandleVisibility'属性设置为'CallBack'有助于隐藏句柄，但不能绝对保证数据安全。当'UserData'存储不能被可靠地利用的时候，Matlab提供了应用数据存储。'ApplicationData'是所有句柄图形对象的一个隐含属性，它保存类似于'UserData'那样的数据。但是，'ApplicationData'通过函数getappdata，setappdata，rmappdata和isappdata来控制对其数据的访问。这样，数据就可以可靠地保存在图形对象中。这些函数在它们的帮助文档中描述如下：

```
>> help getappdata

GETAPPDATA Get value of application-defined data.
    VALUE = GETAPPDATA(H, NAME) gets the value of the
    application-defined data with name specified by NAME in the
    object with handle H. If the application-defined data does
    not exist, an empty matrix will be returned in VALUE.

    VALUES = GETAPPDATA(H) returns all application-defined data
    for the object with handle H.

>> help setappdata

 SETAPPDATA Set application-defined data.
    SETAPPDATA(H, NAME, VALUE) sets application-defined data for
    the object with handle H. The application-defined data,
    which is created if it does not already exist, is
    assigned a NAME and a VALUE. VALUE may be anything.

>> help rmappdata

RMAPPDATA Remove application-defined data.
    RMAPPDATA(H, NAME) removes the application-defined data NAME,
    from the object specified by handle H.

>> help isappdata

ISAPPDATA True if application-defined data exists.
    ISAPPDATA(H, NAME) returns 1 if application-defined data with
    the specified NAME exists on the object specified by handle H,
    and returns 0 otherwise.
```

但是，通过使用永久变量，Matlab提供了另一种数据存储方法。在一个函数M文件内部，可以用函数persistent来声明一个或多个永久变量，也就是说，当一个函数终止运行的时候它们不会消失，而是会保留在函数的工作区中，以便将来函数调用的时候使用。永久变量非常像全局变量，只是它们的范围被限制在声明它们的函数范围之内。persistent的帮助文档对它的用法描述如下：

```
>> help persistent
PERSISTENT Define persistent variable.
     PERSISTENT X Y Z defines X, Y, and Z as persistent in scope so that X,
     Y and Z maintain their values from one call to the next. PERSISTENT
```

```
 can only be used within a function.

Persistent variables are cleared when the M-file is cleared from
memory or when the M-file is changed. To keep an M-file in memory
until MATLAB quits, use MLOCK.

If the persistent variable does not exist the first time you issue
the PERSISTENT statement, it will be initialized to the empty matrix.
Also it is an error to declare a variable persistent if a variable
with the same name exists in the current workspace.

Stylistically, persistent variables often have long names with all
capital letters, but this is not required.

See also GLOBAL, CLEAR, MLOCK, MUNLOCK, MISLOCKED.
```

最后，为了获得对'UserData'**或者应用数据存储的访问，用户必须知道保存这些数据的对象的句柄。当句柄未知的时候，就可以用到**'Tag'**属性和函数**findobj**。通过给期望的对象提供一个惟一的**'Tag'**属性字符串，**findobj**可以用来查找带了期望的标识标签的句柄。例如：**

```
>> findobj(0, 'Tag', 'MyUniqueTagString')
```

这条语句返回包含了指定的字符串的对象的句柄。

31.16 M文件示例

用例子来展示GUI编程是非常困难的，这是因为GUI M文件通常是非常长的，并且它们的用户界面特性使得要想在一张打印纸上展示它们的结果也是很困难的。熟悉GUI编程的最好方法就是在Matlab中或在网上找到一个现成的GUI，用编辑器打开它，查看它的代码，然后运行它，看这些代码都生成了什么。将GUI运行的可视结果和生成这个结果的Matlab代码进行比较，就提供了大量的信息，而这些信息是无法通过其他方式得到的。

本节展示了两个GUI函数。

下面显示的函数mmtext展示了多个回调的用法。这个函数允许文本的图形化布局。mmtext中新加的特性就是当文本置入以后，可以用鼠标围绕着坐标轴拖动这个文本。

```
function h=mmtext(arg)
% MMTEXT Place and Drag Text with Mouse.
% MMTEXT waits for a mouse click on a text object in the current figure
% then allows it to be dragged while the mouse button remains down.
%
% MMTEXT('whatever') places the string 'whatever' on the current axes
% and allows it to be dragged with the mouse
%
% Ht=MMTEXT('whatever') returns the handle to the text object.
%
% MMTEXT becomes inactive after the move is complete or no text
% object is selected.

if nargin==0, arg=0; end
```

```
if ischar(arg)  % user entered text to be placed
    Ht=text('Units','normalized',...
            'Position',[.5 .5]  ,...
            'String',arg  ,...
            'HorizontalAlignment','center',...
            'VerticalAlignment','middle');
    if nargout>0, h=Ht; end
    mmtext(0)   % call mmtext again to drag it

 elseif arg==0  % initial call, select text for dragging
     Hf=get(0,'CurrentFigure');
     if isempty(Hf)
         error('No Figure Window Exists.')
     end
     set(Hf,'BackingStore', 'off',...
            'DoubleBuffer','on',...
            'WindowButtonDownFcn','mmtext(1)')
     figure(Hf)     % bring figure forward

 elseif arg==1 & strcmp(get(gco,'type'),'text')     % text object selected
     set(gco,'Units','data',...
             'HorizontalAlignment','left',...
             'VerticalAlignment','baseline');
 set(gcf,'Pointer','topr',...
         'WindowButtonMotionFcn','mmtext(2)',...
         'WindowButtonUpFcn','mmtext(99)')
elseif arg==2   % dragging text object
         cp=get(gca,'CurrentPoint');
         set(gco,'Position',cp(1,1:3))

else            % mouse button up or incorrect object selected, rese
   set(gcf,'WindowButtonDownFcn','',...
            'WindowButtonMotionFcn','',...
            'WindowButtonUpFcn','',...
            'Pointer','arrow' ,...
            'DoubleBuffer','on',...
            'BackingStore','on')
end
```

mmtext**中所用到的程序将图形的**'WindowButtonDownFcn'**属性设置为**'mmtext(1) '**，这是对**mmtext**的一个回调。然后**gco**被用来标识这个选中的文本的句柄。然后图形的**'WindowButtonMotionFcn'**属性被设置为**'mmtext(2)'**，而**'WindowButtonUpFcn'**被设置为**'mmtext(99) '**。**mmtext**的这三个不同的数值型参数使得这个函数执行用户希望的操作。最后，当文本移动完成了以后，这些回调都被重置为空字符串，因此就禁用了文本选择和拖动的功能。**

下一个例子，名为mmxy**，生成了一个文本对象，这个文本对象跟着鼠标指针绕着一个坐标轴移动。这个文本对象动态地显示鼠标键指针的**x-y**坐标，直到按下鼠标键后，程序结束运行为止。当鼠标键按下的时候，函数返回鼠标指针所在位置的坐标。**

```
function out=mmxy(arg)
```

```
%MMXY Show and Get x-y Coordinates Using Mouse.
% MMXY shows the coordinates of the pointer location over the
% the current axes. Clicking the mouse button returns the
% coordinates at the pointer location.
persistent xy

if nargin == 0 % initialize
    Hf = get (0,'CurrentFigure');
    if isempty(Hf)  % do nothing if there is no figure
       return
    end
    Ha=get(Hf,'CurrentAxes');
    if isempty(Ha) % do nothing if there is no axes
        return
    end
    v=get(gca,'view');
    if any(v~=[0 90])
        error('MMXY works only for 2-D axes.')
    end
    xlim=get(gca,'XLim'); % get axes limits
    ylim=get(gca,'YLim');
    xy=[sum(xlim) sum(ylim)]/2;
    text('Parent',gca,... % create text to display coordinates
         'Position',xy ,...
         'HorizontalAlignment','left',...
         'VerticalAlignment','bottom',...
         'Tag','MMXYtext')
    set(Hf,'Pointer','crossh',...       % change pointer
           'DoubleBuffer','on',...      % eliminate flickering
           'windowButtonMotionFcn','mmxy move',...  % motion callback
           'WindowButtonDownFcn','mmxy done')       % end callback
    figure(Hf)  % bring current figure forward

    tf=waitforbuttonpress; % sit and wait for button press
    if ~tf  % button press so return data
        out=xy;
    end

elseif strcmp(arg,'move') %'WindowButtonMotionFcn' callback
    cp=get(gca,'CurrentPoint'); % current mouse position
    xy=cp(1,1:2);
    Ht=findobj(gca,'Type','text','Tag','MMXYtext');
    set(Ht,'Position',xy ,...
          'String',sprintf('x=  %.2g\ny= %.2g',xy))

elseif strcmp(arg,'done') % mouse click occurred, clean things up
    Ht=findobj(gca,'Type','text','Tag','MMXYtext');
    delete(Ht)
    set(gcf,'Pointer','arrow',...
            'DoubleBuffer','off',...
           'WindowButtonMotionFcn','',...
```

```
            'WindowButtonDownFcn','')
end
```

当第一次调用的时候，mmx生成文本对象并将它放在当前坐标轴的中央，改变鼠标指针的形状，然后设置'WindowButtonMotionFcn'和'WindowButtonDownFcn'回调。如果鼠标键被点击，'WindowButtonDownFcn'回调负责清屏的事情。在等待的时候，图形中鼠标指针的移动触发了'WindowButtonMotionFcn'回调，它更新文本字符串和它的位置。这个函数使用了函数waitforbuttonpress，它暂停程序的运行，直到鼠标键被按下为止。使用这个函数可以避免在第一次执行时直到有一个鼠标按键被按下的时候才停止运行。为了做到这一点，persistent变量x y在鼠标移动回调中被连续地更新。当最终按下一个鼠标键的时候，程序第一次执行到结尾，它将鼠标光标的位置拷贝给输出变量，然后终止程序的运行。

31.17　GUIDE

GUIDE是Matlab提供的一个GUI工具，它可使创建用户自己的GUI更加容易和快速。函数guide提供了对对象进行生成、放置、布局和重新设置大小操作的工具，还提供了一个列出了对象属性、使得用户可以交互地修改这些属性的属性编辑查看器，以及一个交互地编辑和重新布置用户定义的下拉菜单和上下文菜单的菜单编辑器。GUIDE提供了一个开发GUI的交互式方法，这对于布置一个GUI的几何布局来说是非常好的工具。这个工具过去的版本仅仅是让用户在编辑器中编写用户自己的GUI M文件，其效率远远不及GUIDE。GUIDE的最大优势在于GUI对象的布局。当回调包含了多行代码的时候，将这些代码在编辑器内编写仍旧是需要的。本书的作者发现，用M文件编辑器编写GUI代码，并且遵从Matlab的手册Building GUIs with Matlab能够生成最高效和最容易维护的GUI。用户还会发现Matlab 6中新的GUIDE也非常有用。

31.18　小　结

图形用户界面设计并不是每个用户都需要的。但是，如果用户需要一个GUI，那么这个GUI就可以在Matlab中构建。利用Matlab提供的强大数值和图形功能，GUI可以成为展示和浏览多种技术数据的令人印象深刻的工具。

下表总结了Matlab中提供的GUI函数，包括一些在本章中没有讨论的函数。关于GUI构建的更进一步信息可以在在线文档中找到。在附录中，给出了GUI对象属性的完整列表。

函数	描述
uicontrol	生成用户界面控件
uimenu	生成用户界面菜单
uicontextmenu	生成用户界面上下文菜单
drawnow	处理所有未完成的图形事件，并且立即更新屏幕
gcbf	获得回调图形句柄

续表

函数	描述
gcbo	获得回调对象句柄
dragrect	用鼠标拖动方形
rbbox	捕获橡皮圈框的位置
selectmoveresize	交互地对坐标轴和uicontrol进行选择、移动和重置大小操作
waitforbuttonpress	等待在一个图形上的键盘键按下或者鼠标键按下
waitfor	阻塞执行，等待一个事件发生
uiwait	阻塞执行，等待恢复
uiresume	恢复一个被阻塞的M文件的执行
uistack	控制对象的层叠顺序
uisuspend	暂停一个图形的交互状态
uirestore	恢复一个图形的交互状态
uiclearmode	清除当前的交互模式
guide	图形用户界面设计环境
inspect	查看对象属性
align	uicontrol和坐标轴布局
propedit	属性编辑器
makemenu	生成uimenu结构
umtoggle	翻转一个uimenu菜单项的选中状态
getptr	获取图形指针
setptr	设置图形指针
hidegui	隐藏或者显示GUI
movegui	将GUI窗口移动到屏幕上的一个指定位置
overobj	获取鼠标指针所在的对象的句柄
popupstr	获得弹出菜单选择字符串
remapfig	转换图形中对象的位置

第32章 对 话 框

前一章讨论了Matlab用来生成GUI所提供的工具。这些工具可以生成几乎所有操作的用户界面。随着计算机的发展，一些GUI或者说对话框已经或多或少成为了标准功能。本章讨论了Matlab中提供的标准对话框。为了让用户有一个清楚的概念，当用户和Matlab进行交互的时候所使用的不是对话框，但是对话框可以在Matlab的编程和应用程序的开发中发挥作用。

32.1 文 件 选 择

基本上每一个计算机应用程序都提供了打开和保存数据文件的功能。为了浏览目录结构，操作系统或者视窗管理器提供了用来执行确定文件名和目录路径任务的对话框。在Matlab中，这些任务是由函数uigetfile和uiputfile来完成的。实际上，这两个函数都不打开或者保存一个文件。它们只是返回一个文件名和目录路径，这个文件名和目录路径可以和Matlab的低级文件I/O函数一起使用，共同完成打开和保存的操作（关于Matlab中低级文件I/O函数的信息，请参见第13章）。

uigetfile和uiputfile都使用了用户计算机平台上提供的标准对话框。这些对话框的布局是因平台而异的，但是它们的功能都是一样的。uigetfile的帮助文档描述了它的用法。

```
>> help uigetfile

UIGETFILE Standard open file dialog box.
     [FILENAME, PATHNAME] = UIGETFILE('filterSpec', 'dialogTitle')
     displays a dialog box for the user to fill in, and returns the
     filename and path strings. A successful return occurs only if
     the file exists. If the user selects a file that does not exist,
     an error message is displayed, and control returns to the dialog box.
     The user may then enter another filename, or press the Cancel button.

     The filterSpec parameter determines the initial display of files in
     the dialog box. For example '*.m' lists all the MATLAB M-files.

     Parameter 'dialogTitle' is a string containing the title of the dialog
     box.

     The output variable FILENAME is a string containing the name of the
     file selected in the dialog box. If the user presses the Cancel button
     or if any error occurs, it is set to 0.

     The output parameter PATHNAME is a string containing the path of
     the file selected in the dialog box. If the user presses the Cancel
     button or if any error occurs, it is set to 0.

     [FILENAME, PATHNAME] = UIGETFILE('filterSpec', 'dialogTitle', X, Y)
```

```
places the dialog box at screen position [X,Y] in pixel units.
Not all systems support this option.

See also UIPUTFILF.
```

利用uigetfile来查找本书作者计算机上的startup.m文件就得到了如下的结果：

```
>> [fname,dirpath] = uigetfile('*.m')
fname =
startup.m
dirpath =
c:\matlabR12\work\
```

将文件名附加在目录路径之后就给出了一个惟一的startup.m文件规范，例如：

```
>> myfile = [dirpath fname]
myfile =
c:\matlabR12\work\startup.m
```

有一点很重要，需要引起用户的注意，那就是uigetfile并不是只能对Matlab的搜索路径进行操作。这个所返回的目录路径可以是当对话框关闭的时候该对话框所指向的任何目录。另外，在Windows平台中，对话框位置参数不再有效。在这些参数适用的平台上，对话框的位置是用相对于计算机屏幕左上角的像素距离来指定的。

下面这个帮助文档描述了函数uiputfile。

```
>> help uiputfile

UIPUTFILE Standard save file dialog box.
    [FILENAME, PATHNAME] = UIPUTFILE('initFile', 'dialogTitle')
    displays a dialog box for the user to fill in and returns the
    filename and path strings.

    The 'initFile' parameter determines the initial display of files
    in the dialog box. Full file name specifications as well as wildcards
    are allowed. For example, 'newfile.m' initializes the display to
    that particular file and lists all other existing.m files. This may
    be used to provide a default file name. A wildcard specification such
    as '*.m' lists all the existing MATLAB M-files.
    Parameter 'dialogTitle' is a string containing the title of the dialog
    box.

    The output variable FILENAME is a string containing the name of the
    file selected in the dialog box. If the user presses the Cancel button
    or if any error occurs, it is set to 0.

    The output variable PATH is a string containing the name of the path
    selected in the dialog box. If the user presses the Cancel button
    or if any error occurs, it is set to 0.

    [FILENAME, PATHNAME] = UIPUTFILE('initFile', 'dialogTitle',X,Y)
    places the dialog box at screen position [X,Y] in pixel units.
    Not all systems support this option.

    Example:
```

```
                [newmatfile, newpath] = uiputfile('*.mat', 'Save As');
    See also UIGETFILE.
```

利用uiputfile来在用户计算机上的工作目录中保存一个Mat文件就会得到如下的结果：

```
>> [fname,dirpath] = uiputfile('*.mat')
fname =
mydata
dirpath =
c:\matlabR12\work\
```

这里，字符串mydata在对话框中输入为文件名。尽管'*.mat'指定了文件类型，但是'.mat'并没有自动地附加在返回的fname后面。需要由用户编程来添加这个文件后缀名。

32.2　颜 色 选 择

在Matlab中指定的基于RGB三色数据向量的颜色是很直观的，但是无法获得这个给定三色数据向量所指定的实际颜色的视觉效果。另外，大多数的图形对象都有一个用户可以设置的'Color'属性。正如下面的帮助文档所描述的那样，函数uisetcolor提供了一个图形用户界面，让用户来选择RGB三色数据向量，或者设定某个特定图形对象的颜色。

```
>> help uisetcolor

UISETCOLOR Color selection dialog box.
     C = UISETCOLOR displays a color selection dialog appropriate to
      the windowing system, and returns the color selected by the
      user. The dialog is initialized to white.
      C = UISFTCOLOR([R G B]) displays a dialog initialized to the
      specified color, and returns the color selected by the user.
      R, G, and B must be values between 0 and 1.
      C = UISETCOLOR(H) displays a dialog initialized to the color of
      the object specified by handle H, returns the color selected by
      the user, and applies it to the object. H must be the handle
      to an object containing a color property.
     C = UISETCOLOR( ..., 'dialogTitle') displays a dialog with the
     specified title.

      If the user presses Cancel from the dialog box, or if any error
     occurs, the output value is set to the input RGB triple, if provided;
      otherwise, it is set to 0.

     Example:
            hText = text(.5,.5,'Hello World');
            C = uisetcolor(hText,  'Set Text Color')
```

就像uigetfile和uiputfile一样，函数uisetcolor也使用用户计算机平台所提供的标准颜色对话框（如果这个对话框存在的话）。因此，uisetcolor的布局和特性是因平台而异，但是

功能都是一样的。

32.3 字 体 选 择

uisetfont内置函数提供了一个用来选择字体和字体属性的GUI，它的帮助文档描述如下：

```
>> help uisetfont

UISETFONT Font selection dialog box.
     S = UISETFONT(FIN, 'dialogTitle') displays a dialog box for the
     user to fill in, and applies the selected font to the input
     graphics object.
     All the parameters are optional.

     If parameter FIN is used, it must either specify a handle to a
     font-related text, uicontrol, axes object, or it must be a
     font structure.

     If FIN is a handle to an object, the font properties currently
     assigned to this object are used to initialize the font dialog box.

     If FIN is a structure, its fields must be some subset of
     FontName, FontUnits, FontSize, FontWeight, or FontAngle, and must
     have values appropriate for any object with font properties.

     If parameter 'dialogTitle' is used, it is a string containing the
     title of the dialog box.

     The output S is a structure. The structure S is returned with the
     font property names as fields. The fields are FontName,
     FontUnits, FontSize, FontWeight, and FontAngle.

     If the user presses Cancel from the dialog box, or if any error
     occurs, the output value is set to 0.

     Example:
          s = uisetfont(hText, 'Update Font');
          if isstnuct(s)          % Check for cancel
             set(hText_2, s);
          end

      See also LISTFONTS
```

利用这个函数来选择Palatino 12点大小的斜体字就会得到如下的结果。

```
>> fc = uisetfont
fc =
       FontName : 'Palatino'
       FontUnits: 'points'
```

```
        FontSize : 12
      FontWeight : 'normal'
       FontAngle : 'italic'
```

这里，fc是一个结构，它包含了在uisetfont GUI中所选择的特性。这个结果的结构很好，可以用来设置任何“文本”对象的字体属性。

32.4　M文件对话框

除了前面各小节所讨论的内置对话框函数之外，Matlab还提供了一些以M文件函数的方式执行的对话框。其中一些函数提示用户输入某种类型的数据。Matlab中提供的这些函数包括axlimdlg，dialog，inputdlg，menu和msgbox。

函数axlimdlg提供了一个设置图形坐标轴限的基本方法。Matlab 6中的图形窗口编辑特性要复杂得多。函数inputdlg生成了一组文本编辑框uicontrols，并且提示用户输入数据。这个函数对于提示用户输入一些值来说是很有用处的。Menu为用户提供了使用uicontrol按钮的另一种选择。

函数dialog和msgbox都是普通的对话框生成函数。在这两个函数中，dialog只是生成一个属性值为默认值的图形窗口，这个默认值与不包含任何坐标轴对象的图形窗口相对应。用dialog生成的图形窗口使用了最少的系统资源。例如，它们不为一个颜色表分配内存区域。函数msgbox被所有的消息对话框所使用，并且管理弹出一个对话框和接受输入所需要的所有底层动作。

Matlab中标准的消息对话框包括errordlg，helpdlg，questdlg和warndlg。这些对话框显示了相应图标，以及至少一个按钮（让用户确认并关闭对话框）。除了在应用程序开发的过程中有用之外，这些函数还是用户需要学习的很好的创建GUI的例子。下面显示的这些对话框的帮助文档描述了它们各自的用法：

```
>> help errordlg

ERRORDLG Error dialog box.
    HANDLE = ERRORDLG(ErrorString,DlgName,CREATEMODE) creates an
    error dialog box which displays ErrorString in a window
    named DlgName. A pushbutton labeled OK must be pressed
    to make the error box disappear.
    ErrorString will accept any valid string input but a cell
    array is preferred.


>> help helpdlg

HELPDLG Help dialog box.
   HANDLE = HELPDLG(HELPSTRING,DLGNAME) displays the
   message HelpString in a dialog box with title DLGNAME.
   If a Help dialog with that name is already on the screen,
   it is brought to the front. Otherwise a new one is created.
```

```
   HelpString will accept any valid string input but a cell
   array is preferred.

>> help warndlg

WARNDLG Warning dialog box.
   HANDLE = WARNDLG(WARNSTRING,DLGNAME) creates an warning dialog box
   which displays WARNSTRING in a window named DLGNAME. A pushbutton
   labeled OK must be pressed to make the warning box disappear.

   HANDLE = WARNDLG(WARNSTRING,DLGNAME,CREATEMODE) allows CREATEMODE
   options that are the same as those offered by MSGBOX. The default
   value for CREATEMODE is 'non-modal'.
   WARNSTRING may be any valid string format. Cell arrays are
   preferred.

>> help questdlg

QUESTDLG Question dialog box.
   ButtonName=QUESTDLG(Question) creates a modal dialog box that
   automatically wraps the cell array or string (vector or matriX)
   Question to fit an appropriately sized window. The name of the
   button that is pressed is returned in ButtonName.The title of
   the figure may be specified by adding a second string argument.
   Question will be interpreted as a normal string.

   QUESTDLG uses WAITFOR to suspend execu ton until the user responds.

   The default set of buttons names for QUESTDLG are 'Yes','No' and
   'Cancel'. The default answer for the above calling syntax is 'Yes'.
   This can be changed by adding a third argument which specifies the
   default Button. i.e. ButtonName=questdlg(Question,Title,'No').

   Up to 3 custom button names may be specified by entering
   the button string name(s) as additional arguments to the function
   call. If custom ButtonName's are entered, the default ButtonName
   must be specified by adding an extra argument DEFAULT, i.e.

      ButtonName=questdlg(Question,Title,Btn1,Btn2,DEFAULT);

   where DEFAULT=Btn1. This makes Btn1 the default answer.
   To use TeX interpretation for the Question string, a date
   structure must be used for the last argument, i.e.

      ButtonName=questdlg(Question,Title,Btn1,Btn2,0PTIONS);

   The OPTIONS structure must include the fields Default and Interpreter.
   Interpreter may be 'none' or 'tex' and Default is the default button
   name to be used.
```

32.5 小 结

我们将Matlab中所提供的对话框函数小结如下：

函数	描述
axlimdlg	坐标轴限对话框
dialog	生成对话框或者GUI的图形
errordlg	错误提示对话框
helpdlg	帮助对话框
inputdlg	输入对话框
listdlg	列表选择对话框
menu	菜单选项选择对话框
msgbox	普通消息对话框
pagedlg	页面位置对话框
pagesetupdlg	页面设置对话框
printdlg	打印对话框
printpreview	打印预览对话框
questdlg	提问对话框
uigetfile	标准打开文件对话框
uiputfile	标准保存文件对话框
uisetcolor	颜色选择对话框
uisetfont	字体和字体属性对话框
waitbar	显示等待条
warndlg	报警对话框

第 33 章　Matlab 类和面向对象编程

Matlab提供了多种基本的数据类型，它们就是类。例如，数字数组通常都是双精度的数组。一个包含这样的数组的变量就拥有了一个被称为double的类。同样，字符串也是一种数据类型，或者说是一种类。包含字符串的变量就拥有了一个char类。请参考下面的例子：

```
>> pi              % a simple double
ans =
        3.1416
>> class(pi)
ans =
double
>> s ='pi'         % a simple string
s =
pi
>> class(s)
ans =
char
```

Matlab的基本数据类型，即类，包括double、char、logical、cell、struct。这些数据类型是Matlab中最常用到的数据类型。另外，Matlab还包含了一些较少用到的类：sparse、function handle、inline、java、single和多种整数数据类型。

Matlab对每一个类都定义了相应的可执行的运算。例如，Matlab为double类的元素定义了加法运算符，但没有为char类和cell类的元素定义加法运算符。

```
>> x = pi+2
x =
            5.1416
>> y = 'hello' + 'there'
y =
    220     205     209     222     212
>> {'hello' 'there'}+{'sunny' 'day'}
??? Error using ==> +
Function '+' not defined for variables of class 'cell'.
```

上面的例子将两个字符串相加，结果生成了一个数字数组，而不是一个字符串。Matlab没有对该运算报错，而是将“hello”和“there”转换成相应的ASCII数字，然后依次将每两个数字相加。在上面的例子中，从表面上来看，Matlab对字符串进行了加法运算，但是实际上，Matlab是在将字符串转换成double类后才进行的加法运算。Matlab是按照惯例来进行这种隐含的数据类型（类）转换的，而不是因为对字符串定义了加法运算才进行转换的。但是如果试图强行将两个单元数组相加，那么将会立即产生一个错误。

从Matlab 5开始，Matlab中加入了两种新功能：为基本数据类型定义新运算，创建用户定义的数据类型（类）。创建和使用数据类型被称为面向对象的编程（OOP），每个数据类型或类中的变量被称为对象。对象的运算由封装了数据和过载运算符和函数的方法定义。OOP的词汇表中包含了运算符和函数过载、数据封装、方法、继承、集合等术语。Matlab中的这些术语和面向对象编程的基本规则将在本章中进行讨论。

33.1　重　载

在我们深入了解OOP的详细内容并创建新的变量类之前，先来看一看Matlab中过载标准类的过程。过载标准类和过载用户生成的类所用的技术是一样的。所以一旦理解了标准类的过载，就可以理解用户生成类的过载。

当Matlab的程序解释器遇到一个运算符（例如加法运算）或一个带有一个或多个输入参数的函数时，Matlab会判断运算符或函数的参数的数据类型（类），并按照内部定义的规则进行运算。例如，加法就意味着：如果参数是数字或可以被转换成数字的值（例如字符串），那么将计算参数的数字和。如果对运算符或函数的内部规则进行了重新定义，那么则称该运算符或函数被“过载”。

> 运算符和函数过载允许用户重新定义Matlab在遇到此运算符或函数时所执行的运算。用于重新定义运算符和函数的规则集或M文件称为**方法**。这些文件本身通常被称为方法函数。

Matlab中用于解释运算符和函数的重新定义的规则不过是一些函数M文件，这些文件就存储在Matlab搜索路径下的类目录中。这就是说：类目录本身不是，也不能成为Matlab的搜索路径，但是类目录必须是Matlab搜索路径上的某个目录的子目录。为了能够找到类子目录，Matlab要求类目录必须以“@class”命名，在这里“class”是“@class”中的M文件所应用的变量类。另外，Matlab还支持多个类目录。这意味着我们可以在Matlab路径下放置多个“@class”目录。Matlab在寻找函数和类目录时，会执行Matlab搜索路径所给出的命令，并且使用所找到的第一个符合要求的方法函数文件。

例如，如果目录“@char”在Matlab的搜索路径下，此目录中的M文件能够重新定义关于字符串的运算符和函数。为了更好地说明，请参阅下面给出的函数M文件——“plus.m”。

```
>> function s=plus(s1,s2)
% Horizontal Concatenation for char Objects.

if ischar(s1)&ischar(s2)
    s=cat(2,s1(:).',s2(:).');
elseif isnumeric(s2)
    s=double(s1)+s2;
else
    error('Operator+Not Defined. ')
end
```

如果此M文件正好存储在Matlab搜索路径下，那么字符串的加法将被重新定义为水平

串联。如果我们再次使用前面的声明y = 'hello' + 'there' ，那么：

```
>> y = 'hello'+'there'
y =
hellothere
```

Matlab将不会再将字符串转换成相应的ASCII代码并进行数值加法运算。Matlab将：(1) 对“ + ”两侧的字符串进行译码；（2）在Matlab的搜索路径下寻找@char子目录；（3）找到我们所创建的目录，然后继续寻找命名为plus.m的函数M文件；（4）找到上面所给出的plus.m函数，将两个参数传送到加法运算符相对应的函数中，然后由函数来决定所执行的运算；（5）最后将函数返回的输出作为加法运算的结果。

为了加快运算的速度，Matlab会在启动的时候将类子目录存储在缓存中。因此，对于以上的运算，必须要在创建子目录和M文件后，重新启动Matlab或使用rehash命令，使Matlab能够缓存最新创建的类子目录和相关联的M文件。

当对两种不同类型的数据，例如char和double，进行加法运算时，Matlab会考虑参数的优先级和顺序。因为所有的变量都具有相同的优先级，所以Matlab将给运算符或函数最左边的参数较高的优先级，例如：

```
>> z = 2 + 'hello'
z =
    106     103     110     110     113
```

double和char类具有相同的优先级。因此，Matlab会认为加法是数字加法，并应用内部规则将“ hello ”转换成相应的ASCII值，然后进行数字加法运算。但是如果将上面的两个操作数的顺序互换，

```
>> z ='hello' + 2
z =
    106     103     110     110     113
```

那么Matlab会认为加法是char类运算。在本例中，将使用plus('hello',2)调用plus.m函数。按照函数的代码，plus.m将判断出这是一个混合类，并调用isnumeric(s2)，然后返回与 z = 2 + 'hello' 相同的结果。

如上所述，我们在类子目录中定义了一个plus.m函数。为了能够支持其他运算符的过载，Matlab还为一些运算符指定了函数名，如下表所示。

运算符	函数名称	说明
a+b	plus(a,b)	数字加法
a-b	minus(a,b)	数字减法
-a	uminus(a)	一元减号
+a	uplus(a)	一元加号
a .* b	times(a,b)	逐个元素相乘
a * b	mtimes(a,b)	矩阵乘法
a ./ b	rdivide(a,b)	逐个元素相右除

续表

运算符	函数名称	说明
a .\ b	ldivide(a,b)	逐个元素相左除
a/b	mrdivide(a,b)	矩阵左除
a\b	mldivide(a,b)	矩阵右除
a .^ b	power(a,b)	逐个元素求幂
a ^ b	mpower(a,b)	矩阵求幂
a < b	lt(a,b)	小于
a > b	gt(a,b)	大于
a <= b	le(a,b)	小于或等于
a >= b	ge(a,b)	大于或等于
a ~= b	ne(a,b)	不等于
a = = b	eq(a,b)	等于
a & b	and(a,b)	逻辑与
a | b	or(a,b)	逻辑或
~a	not(a,b)	逻辑非
a:d:b	colon(a,d,b)	冒号运算符
a:b	colon(a,b)	冒号运算符
a'	ctranspose(a)	共轭转置
a.'	transpose(a)	转置
[a b]	horzcat(a,b)	水平串联
[a; b]	vertcat(a,b)	垂直串联
a(s1,s2,…)	subsref(a,s)	下标引用
a(s1,s2,…)=b	subsasgn(a,s)	下标分配
b(a)	subsindex(a)	下标索引
	display(a)	命令窗口输出
end	end(a,k,n)	end的下标说明

继续以上面的char类的例子为例，使用下面的函数将减法过载。

```
functon s=minus(s1,s2)
% Subtraction for char Objects
% Delete occurrences of s2 in s1.

if ischar(s1)&ischar(s2)
    s=strrep(s1,s2, '');
elseif is numeric(s2)
    s=double(s1)-s2;
else
    error('Operator – Not Defined. ')
end
```

按照上面的定义，减法将被解释成为从字符串中减去对应的字符串，例如：

```
>> z ='hello' -'e'
z =
hllo
>> a = 'hello' - 2
a =
    102     99      106     106     109
```

同样，在混合类的情况下，将返回Matlab的默认运算。

当在一个语句中出现多个运算符的时候，Matlab会按照它通常的处理规则来执行，即从表达式的左端执行到右端，例如：

```
>> a ='hello' + ' ' + 'there'
a =
hello there
>> a -'e'
ans =
hllo thr
>> a = 'hello' + ' ' + ('there' -'e')
a =
hello thr
```

当声明将一个字符串赋给一个不带有结束分号的输出时，Matlab会将字符串显示在命令窗口中。在命令窗口中显示的字符串可以使用函数display.m进行过载。虽然Matlab默认的字符显示模式比较方便，但是也可以使用如下所示的函数来过载。

```
function display(s)
% Display for char objects.

isloose=strcmp(get(0, 'FormatSpacing'), 'loose');
ssiz= size(s);

if isloose,disp(' '), end
disp(['A Character Array of Size: 'mat2str(ssiz)])
if isloose, disp(' '), end
```

上面给出的函数重新定义了命令窗口中显示字符串的模式，例如：

```
>> 'hello'
A character Array of Size: [1 5]
>> a = 'hello'+''+'there'
A Character Array of Size:[1 11]
>> format loose
>> a

A Character Array of Size:[1 11]
>> format compact
>> s=char('hello', 'there')
```

```
A Character Array of Size:[2 5]
```

上面我们讨论了Matlab对运算符的过载。过载函数也是按照同样的步骤进行的。在过载函数的时候，函数与标准的Matlab函数存储在同一个子目录下，例如：

```
function s=cat(varargin)
% CAT Concatenate Strings as a Row.

if length(varargin)>1&~ischar(varargin{2})
    error('CAT Not Defined for Mixed Classes. ')
else
    s =cat(2,varargin{:});
end
```

此函数过载了用于字符串的cat函数。只有在cat函数的第一个参数是字符串，即是char类的时候才会调用此函数。如果第一个参数是数字，那么将会调用标准的cat函数，例如：

```
>> cat('hello', 'there')    % call overloaded cat
ans =
hellothere
>> cat('hello',2)           % call overloaded cat
??? Error using ==> char/cat
CAT Not Defined for Mixed Classes.

>> cat(2, 'hello')          % call built-in cat
ans =
hello
```

除了本节列出的运算符的过载函数外，Matlab**还提供了几个**OOP**实用函数。它们包括**methods、isa、class、loadobj、saveobj。isa**和**class**函数用于帮助确定对象或变量的数据类型（类），例如：**

```
>> a = 'hello';
>> class(a)         % return class of argument
ans =
char
>> isa(a, 'double')     % logical class test
ans =
    0
>> isa(a, 'char')   % logical class test
ans =
    1
```

对于一个给定的类，methods**函数可以返回与该类相关联的方法或过载运算符和函数的列表，例如：**

```
>> methods cell

Methods for class cell:
```

```
deblank     ismember        setxor      strcat      union
intersect   setdiff         sort        strmatch    unique
```

所显示的结果表明Matlab自己已经在输入参数是单元数组的时候过载了被调用的函数。这些函数对Matlab的基本函数进行了扩展，以适合单元数组参数。这样做的好处是，无需对函数本身进行重写就可以使函数接受单元数组参数。

无论在何时使用用户定义的类来分别调用load和save函数，如果存在loadobj和saveobj函数，那么将调用loadobj和saveobj函数。可以在装载运算后或在保存运算前，通过向类子目录添加这些函数来修改用户定义的变量。

33.2 创 建 类

运算符和函数过载是OOP的关键内容。Matlab依据一个非常简单的方案进行操作，通过此方案，方法与存储在Matlab搜索路径下的类目录中的变量类相关联。方法本身存储在标准的M文件中（也可以是与M文件等效的P文件和MEX文件）。用户定义的类使用同一方案对方法进行创建和存储。本节将演示如何创建用户定义的类。

创建一个新的变量类需要创建一个@classname类目录，并至少提供两个函数M文件。第一个M文件是class.m，该文件用于定义在新类中变量的生成。因此该M文件被称为构造器。第二个M文件是display.m，该文件用于在命令窗口中显示新变量。虽然，如果没有附加的方法M文件，那么变量类将没有任何用处，但是实际上，仅提供构造器和display.m就可以创建一个变量类。

> 在OOP的词汇表中，构造器用于创建类的实例，该实例是一个拥有方法的对象，此方法规定了在对象前先过载何种运算符和函数。

构造器函数classname.m是一个带有输入参数的标准函数调用，输入参数中包含了为我们选定的类创建输出变量所需的数据。为了能够得到最大的灵活性，构造器应该可以处理三种不同的输入参数集合。对于空字符串、数组、单元等不同参数，构造器都应该可以在无输入参数的情况下输出一个空变量。另一方面，如果向构造器传递了一个属于由同一构造器创建的相同的类的变量，那么构造器应该直接将它作为输出参数传递出去。最后，如果提供了创建数据，那么应该创建一个新的属于选定的类的变量。在最后这种情况下，可以输入数据以检验变量是否正确。在构造器中，用于创建选定类的变量的数据存储在一个结构的字段中。一旦提供了该结构字段，则可以通过调用class函数来创建新变量。例如，下面给出了一个用于有理多项式的构造器函数：

```
function r=mmrp(varargin)
%MMRP Mastering MATLAB Rational Polynomial Object Constructor.
% MMRP(p) creates a polynomial object from the polynomial vector p
% with 'x' as the variable.
% MMRP(p,'s') creates the polynomial object using the letter 's' as
% the variable in the display of p.
% MMRP(n,d) creates a rational polynomial object from the numerator
% polynomial vector n and denominator polynomial d.
% MMRP(n,d,'s') creates the rational polynomial using the letter 's' as
```

```
% the variable in the display of p.
%
% All coefficients must be real.

[n,d,v,msg]=local_parse(varargin); % parse input arguments
if isempty(v) % input was mmrp so return it
    r=n;
else
  error(msg) % return error if it exists
  tol=l00*eps;
  if length(d)==1 & abs(d)>tol % enforce scalar d=1
           r.n=n/d;
           r.d=1;
   elseif abs(d(1))>tol % make d monic if possible
           r.n=n/d(1);
           r.d=d/d(1);
   else               % can't be made monic
           r.n=n;
           r.d=d;
   end
   r.v=v(1);

   r=class(r,'mmrp');   % create object from parts
   r=minreal(r);        % pole-zero cancellation
end
```

为了简化此函数，对输入参数的分解是由一个名为local_parse**的子函数处理的。该函数并不显示，但它返回四个输出：**n、d、v、msg**。变量**n**和**d**使数字行向量，分别包含有理多项式的分母和分子系数。变量**v**包含用于多项式显示的字符串变量。而变量**msg**包含了**local_parse**在遇到无效输入的情况时返回的错误信息。**

有理多项式的创建是由语句r = class (r,'mmrp') **来执行的。**class**函数的此种用法只在构造器本身中有效。在其他情况下，**class**将会返回它单个输入参数的类的字符串名。上面给出的构造器可以处理三种情况的输入参数。如果不存在输入参数，那么将会返回**n、d**和**v**来创建一个空有理多项式。如果输入参数是一个有理多项式对象，那么会直接将此对象作为输出参数返回。如果提供了数据，那么将会创建一个有理多项式对象。最简单的情况是分母等于**1**，显示变量为“ x ”的情况。**mmrp**中的最后一条语句将已创建的有理多项式传递给已被过载的**minreal**函数。**minreal**函数通过删除极点和零点来返回该对象的一个最小实现。**

在构造器中，让所有创建的结构字段在任何情况下都保持相同的顺序是非常重要的。如果两个结构包含有相同的字段，但是字段定义的顺序不同，那么这两个结构将不等。违反了这一规则将会导致对已创建对象运算的不确定性。

下面给出与前面的构造器相关联的display.m**文件的代码**

```
function display(r)
%DISPLAY Command Window Display of Rational Polynomial Objects.

loose=strcmp(get(0, 'FormatSpacing'),'loose');
if loose, disp(' '), end
var=inputname(1);
if isempty(var)
       disp('ans =')
else
```

```
        disp([var ' ='])
end
nstr=mmp2str(r.n,r.v); % convert polynomial to string
nlen=length(nstr);
if length(r.d)>1 | r.d~=1
        dstr=mmp2str(r.d,r,v);
else
        dstr=[];
end
dlen=length(dstr);
dash='-';
if loose, disp(' '), end
if dlen % denominator exists
        m=max(nlen,dlen);
        disp('MMRP Rational Polynomial Object:')
        disp([blanks(ceil((m-nlen)/2)) nstr]);
        disp(dash(ones(1,m)));
        disp([blanks(fix((m-dlen)/2)) dstr]);
else
         disp('MMRP Rational Polynomial Object:')
         disp(nstr);
 end
 if loose, disp(' '), end
```

上面给出的函数将调用mmp2str**函数来将数字多项式向量和所要求的变量转换成字符串表达式。此函数未存储在**Matlab**搜索路径下的**@mmrp**目录中，而是存储在搜索路径下的其他目录中。如果该函数存储在类目录下，那么**Matlab**将会找不到该函数，这是因为**mmp2str**的参数分别是**double**和**char**类的，而不是**mmrp**类的。正如我们在上一节中所描述的，只有在最左端或最高优先级的输入变量的类和方法相匹配时，才会调用方法函数。**

在一个方法函数中，可以像构造器中的最后一条语句 r = minreal (r) **一样对对象进行运算，在这里，右端的变量**r**是一个具有**mmrp**类的对象。此属性有两个例外。当在方法函数中出现带有下标的引用和带有下标的赋值时，不会调用过载函数**subsref**和**subsasgn**。这就允许用户可以更加自由地访问和操作方法函数中的类变量。**

还可以简单地通过寻址对象的结构字段（如display.m**中所示），将其应用在对象所包含的数据上。在这种情况下，数据的类决定了**Matlab**的操作。**

> **在方法函数之外，例如在命令窗口中，不可能访问对象的字段。同样也无法确定字段的数量和名称。此属性被称为数据封装（**data encapsulation**）。**

下面的例子将演示有理多项式对象的创建和显示。

```
>> p = mmrp([1 2 3])
p =
MMRP Rational Polynomial Object:
x^2 + 2x^1 + 3

>> q = mmrp([1 2 3],[4 5 6], 'z')
q =
MMRP Rational Polynomial Object:
```

```
0.25z^2 + 0.5z^1 + 0.75
-----------------------------------------------
z^2 + 1.25z^1 + 1.5

>> r = mmrp(conv([1 2],[1 4]),conv([1 2],[1 3]))
r =
MMRP Rational Polynomial Object:
x^1 + 4
------------------
x^1 + 3
```

如果运算符和函数没有过载，那么有理多项式对象仅仅拥有很小的值。特别是可以方便地在mmrp对象上定义算术操作符。下面的M文件中为mmrp对象定义了加法、减法、乘法和除法。因为乘法和除法有多种方法，所以使用了相关联的多项式操作对它们进行了过载。

```
function r=plus(a,b)
%PLUS Addition for Rational Polynomial Objects.

if isnumeric(a)
    rn=mmpadd(a*b.d,b.n) ; % see chapter 19 for mmpadd
    rd=b.d;
    rv=b.v;
e1seif isnumeric(b)
    rn=mmpadd(b*a . d , a . n );
    rd=a.d;
    rv=a.v;
else % both polynomial objects
   if ~1sequa1(a.d,b.d)
         rn=mmpadd(conv(a.n,b.d),conv(b.n,a.d));
         rd=conv(a.d,b.d);
   e1se
         rn=mmpadd(a.n,b.n);
         rd=b.d;
    end
    if ~strcmp(a.v,b.v )
           warning('Variables Not Identical')
    end
    rv=a.v;
end
r=mmrp(rn,rd,rv );

function r=minus(a.b)
%MINUS Subtraction for Rational Polynomial Objects.

r=a+(-b);  % use plus and uminus to implement minus
```

```
function r=uminus(a)
%UPLUS Unary Minus for Rational Polynomial Objects.

r=mmrp(-a.n,a.d,a.v);

function r=times(a,b)
%TIMES Dot Times for Rational Polynomial Objects.

a=mmrp(a); % convert inputs to mmrp if necessary
b=mmrp(b);
rn=conv(a.n,b.n);
rd=conv( a .d ,b .d );
if ~strcmp(a.v,b.v)
    warning('Variables Not Identical')
end
rv=a.v;
r=mmrp(rn,rd,rv );

function r=mtimes(a,b)
%MTIMES Times for Rational Polynomial Objects.

r=a.*b; % simply call times.m

function r=rdivide(a,b)
%RDIVIDE Right Dot Division for Rational Polynomial Objects,

a=mmrp(a); % convert inputs to mmrp if necessary
b=mmrp(b);
rn=conv(a.n,b.d) ;
rd=conv(a.d,b.n);
if ~strcmp(a.v,b.v)
    warning('Variab1es Not Identical')
end
rv=a.v;
r=mmrp(rn,rd,rv);

function r=mrdivide(a,b)
%MRDIVIDE Right Division for Rational Polynomial Objects.

r=a./b; % simply ca11 rdivide.m

function r=ldivide(a,b)
%LDIVIDE Left Dot Division for Rational Polynomial Objects.

r=b./a; % simply ca11 rdivide.m

function r=mldivide(a,b)
```

```
%MLDIVIDE Left Division for Rational Polynomial Objects.

r=b./a; % simply ca11 rdivide.m
```

上面给出的几个方法函数都带有注释，说明了它们各自的功能，它们可以执行简单的多项式运算。下面给出了使用这些方法函数的例子：

```
>> a = mmrp([1 2 3])
a =
MMRP Rational Polynomial Object:
x^2 + 2x^1 + 3
>> b = a + 2                  % addition
b=
MMRP Rational Polynomial Object:
x^2 + 2x^1 + 5
>> a - b                      % subtraction
ans =
MMRP Rational Polynomial Object:
-2
>> a + b                      % addition
ans =
MMRP Rational Polynomial Object:
2x^2 + 4x^1 + 8

>> 2*b                         % multiplication
ans=
MMRP Rational Polynomial Object:
2x^2 + 4x^1 + 10
>> a * b                       % multiplication
ans =
MMRP Rational Polynomial Object:
x^4 + 4x^3 + 12x^2 + 16x^1 + 15
>> b/2                         % division
ans =
MMRP Rational Polynomial Object:
0.5x^2 + x^l + 2.5

>> 2/b                         % division
ans =
MMRP Rational Polynomial Object:
       2
--------------
x^2 + 2x^1 + 5
>> c = a/b                     % division
 C =
MMRP Rational Polynomial Object:
x^2 + 2x^1 + 3
--------------
x^2 + 2x^1 + 5

>> d = c/(1+c)                 % mixed
```

```
d=
MMRP Rational Polynomial Object:
0.5x^2+x^1+1.5
--------------
x^2 + 2x^1 + 4

>> (a/b)*(b/a)                  % mixed
ans=
MMRP Rational Polynomial Object:
1
```

有了在Matlab中给定的多项式函数和对这些函数的简便操作，我们就可以对多种函数进行过载。例如：基本的Matlab函数roots和zeros可以使用下面的方法M文件过载。

```
function [z,p]=roots(r)
%ROOTS Find Roots of Rational Polynomial Objects.
% ROOTS(R) returns the roots of the numerator of R.
% [Z,P]=ROOTS(R) returns the zeros and poles of R in
% Z and P respectively.

z=roots(r.n);
if nargout==2
    p=roots(r.d);
end

function z =zeros(r)
%ZEROS Zeros of a Rational Polynomial Object.

z=roots(r.n);
```

方法函数roots会调用基本的Matlab函数roots，这是因为方法所带的参数是double类的。在创建zeros方法的同时，函数zeros因其所带参数的不同，具有两个完全不同的含义。OOP的优势在于：函数可以有多种含义或使用范围，而无需将它们都嵌入到同一个M文件中。输入参数的类规定了调用何种函数进行操作。

由于它们处理句柄图形对象的能力不强，所以通常要过载set和get函数。模仿句柄图形的用法，通常使用set和get函数来设置或获得独立的类结构的字段，例如：

```
function set(r,varargin)
%SET Set Rational Polynomial Object Parameters.
% SET(R,Name,Value,...) sets MMRP object parameters of R
% described by the Name/Value pairs:
%
%  Name          Value
% 'Numerator'    Numeric row vector of numerator coefficients
% 'Denominator'  Numeric row vector of denominator coefficients
% 'Variable'     Character Variable used to display polynomial
if rem(nargin,2)~=1
    error('Parameter Name/Values Must Appear in Pairs.')
end
```

```
for i=2:2:nargin-1
    name=varargin{i-1};
    if ~ischar(name), error('Parameter Names Must be Strings.'), end
    name=lower(name(isletter(name)));
     value=varargin{i};
     switch name(1)
     case  'n'
         if ~isnumeric(value) | size(value,1)>1
              error('Numerator Must be a Numeric Row Vector,')
          end
          r.n=value;
     case  'd'
          if ~isnumeric(value) | size(value,1)>1
               error('Denominator Must be a Numeric Row Vector.')
          end
          r.d=value;
     case  'v'
          if ~ischar(value) | length(value)>1
               error('Variable Must be a Single Character.')
          end
          r.v=value;
      otherwise
          warning('Unknown Parameter Name')
      end
 end
 vname=inputname(1);
 if isempty(vname)
     vname='ans';
 end
 r=mmrp(r.n,r.d,r.v);
 assignin('caller',vname,r);
```

```
function  varargout = get (r,varargin)
% GET Get Rational Polyomial Object Parameters.
% GET (R,Name) gets the MMRP object parameter of R described by
% one of the following names:
%
% name Description
% 'Numerator'   Numeric row vector of numerator coefficients
% 'Denominator' Numeric row vector of denominator coefficients
% 'Variable'    Character Variable used to display polynomial
%
% [A,B,...  1=get(R,NameA,NameB,...  ) returns multiple parameters
% in the corresponding output arguments.
```

```
if (nargout+(nargout==0))~=nargin-1
    error('No. of Outputs Must Equal No. of Names.')
end
for i=1:nargin-1
    name=varargin{i};
     if ~ischar(name), error('Parameter Names Must be Strings.'), end
    name=lower(name(isletter(name)));
       switch name(1)
       case  'n'
          varargout{i}=r.n;
       case  'd'
```

```
            varargout{i}=r.d;
         case  'v'
            varargout{i}=r.v;
         otherwise
            warning('Unknown Parameter Name')
         end
end
```

这些函数允许我们修改mmrp对象，或从mmrp对象中获得数据。请参考下面的例子：

```
>> c % recall data
c =
MMRP Rational Polynomial Object:
x^2 + 2x^1 + 3
---------------
x^2 + 2x^1 + 5

>> n = get(c, 'n') % get numerator vector
n =
        1       2       3
>> set (c, 'Numerator',[3 1]) % change numerator
>c
c =
MMRP Rational Polynomial Object:
3x^1 + 1
-----------------
x^2 + 2x^1 + 5

>> class(c) % class and isa know about mmrp objects
ans =
mmrp
>> isa(c, 'mmrp')
ans =
    1
```

33.3 下 标

由于Matlab数组的排列方向的关系，用户定义的类也可以使用下标。特别是还支持V(…)、V{…}和V.field三种格式。另外，这些结构可以出现在参数声明的任意一侧。如果它们出现在参数声明的右侧，那么函数将调用变量V。如果它们出现在参数声明的左侧，那么函数将对变量V的某些部分赋值。使用下标的引用和使用下标的赋值会分别调用这些索引过程。当应用在对象上时，用于控制如何解释下标的方法函数分别是subsref和subasgn。这些函数和其他过载的操作符和函数相比不是很容易直接理解。因此，本节将专门介绍它们。为了便于讨论，仍将在本节的例子中使用前面一节中创建的mmrp对象。

在处理有理多项式的时候，对下标引用有两种很明显的解释。对于一个有理多项式对象R，R(x)可以返回x中这些点上R的值（其中x是一个数据数组）。而另一种形式，R('v')则可将用于显示对象的变量修改成所提供的字母（其中'v'是一个字符）。

在subsref的帮助文档中介绍了如何编写subsref方法，如下所示：

```
B = SUBSREF(A,S) is called for the syntax A(I), A{I}, or A.I
```

```
when A is an object. S is a structure array with the fields:
     type -- string containing '()', '{}', or '.' specifying the
             subscript type.
     subs -- Cell array or string containing the actual subscripts.

For instance, the syntax A(1:2,:) invokes SUBSREF(A,S) where S is a
1-by-1 structure with S.type='()' and S.subs = {1:2,':'}. A colon
used as a subscript is passed as the string ':'
Similarly,the syntax A{1:2} invokes SUBSREF(A,S) where S.type ='{}'
and the syntax A.field invokes SUBSREF(A,S) where S.type ='. ' and
S.subs ='field'.

These simple calls are combined in a straightforward way for
more complicated subscripting expressions. In such cases
length(S) is the number of subscripting levels. For instance.
A(1,2).name(3:5) invokes SUBSREF(A,S) where S is 3-by-1 structure
array with the following values:
   S(1).type='()'      S(2).type='. '      S(3).type='()'
   S(1).subs={1,2}     S(2).subs = 'name'  S(3).subs={3:5}
```

基于此帮助文本，如果R**是一个**mmrp**对象，那么**R(x)**会创建**S.type=' ( ) '**和**S.subs=x**，其中**x**包含对**R**进行计算时所需的值，而不是数组的索引。同样，**R('v')**会创建**S.type=' ( ) '**和** S.subs='v'**。如果出现其他情况，则会产生错误。**

根据上面的描述，subsref**方法函数如下所示：**

```
function y=subsref(r,s)
%SUBSREF(R,S) Subscripted Reference for Rational Polynomial Objects.
% R('z') returns a new rational polynomial object having the same
% numerator and denominator, but using the variable 'z'.
%
% R(x) where x is a numerical array, evaluates the rational polynomial
% R at the points in x, returning an array the same size as x.

if length(s)>1
       error('MMRP Objects Support Single Arguments Only.')
end
if strcmp(s.type,'()') % R(x) or R('v')
    arg=s.subs{1};
    argc=class(arg);
    if strcmp(argc,'char')
        if strcmp(arg(1),':')
           error('MMRP Objects Do Not Support R(:).')
        else
            y=mmrp(r.n,r.d,arg(1)); % change variables
        end
    elseif strcmp(argc,'double')
if length(r.d)>1
            y=polyval(r.n,arg)./polyval r.d,arg);
```

```
        else
            y=polyval (r.n,arg);
        end
    else
        error( 'Unknown Subscripts.')
    end
else % R{ } or R.field
    error('Cell and Structure Addressing Not Supported.')
end
```

下面给出一些使用此方法的例子：

```
>> c % recall data
c =
MMRP Rational Polynomial Object:
3x^1 + 1
--------------
x^2 + 2x^1 +5

>> c = c('t') % change variable
c =
MMRP Rational Polynomial Object:
3t^1 + 1
--------------
t^2 + 2t^1 + 5

>> x = -2:2
x =
    -2      -1      0       12
>> c(x)     % evaluate at points in x
        -1      -0.5        0.2     0.5     0.53846
>> c{3}     % try cell addressing
??? Error using ==> mmrp/subsref
Cell and Structure Addressing Not Supported.

>> c.n      % Try field addressing
??? Error using ==> mmrp/subsref
Cell and Structure Addressing Not Supported.
```

同前面声明的一样，方法函数外的对象的字段结构是隐藏的，不可见。如果它不是隐藏的，则可以使用c.n**，这样就可以返回对象**c**的分子行向量。为了加入这一功能，必须要在** subsref **方法中明确加入相应的代码，代码如下所示：**

```
function y=subsref(r,s)
%SUBSREF(R,S) Subscripted Reference for Rational Polynomial Objects.
% R('z') returns a new rational polynomial object having the same numerator
% and denominator, but using the variable 'z'.
%
% R(x) where x is a numerical array, evaluates the rational polynomial R
```

```
% at the points in x, returning an array the same size as x.
%
% R.n returns the numerator row vector of R.
% R.d returns the denominator row vector of R.
% R.v returns the variable associated with R.

if length(s)>1
    error('MMRP Objects Support Single Arguments Only')
end
if strcmp(s.type,'()') % R( )
   arg=s.subs{1};
   argc=class(arg);
   if strcmp(argc,'char')
        if strcmp(arg(1),':')
            error('MMRP Objects Do Not Support R(:).')
        else
            y=mmrp (r.n,r.d,arg(1));
        end
    elseif strcmp(argc,'double')
        if length(r.d)>1
            y=polyval(r.n,arg)./polyval(r.d,arg);
        else
            y=polyval(r.n,arg);
        end
    else
        error('Unknown Subscripts.')
    end
elseif strcmp(s.type,'.') % R.field
    arg=lower(s.subs);
    switch arg(1)
    case  'n'
      y=r.n;
    case 'd'
       y=r.d;
     case  'v'
        y=r.v;
    otherwise
        error( 'Unknown Data Requested.')
    end
else % R{ }
    error('Cell Addressing Not Supported.')
end
```

下面给出一些使用此方法的例子：

```
>> c.n % return numerator
ans =
    3       1
>> c.v % return variable
```

```
ans =
x
>> c.nadfdf % only first letter is checked
ans =
    3       1
>> c.t      % not n,d,or v
??? Error using ==>mmrp/subsref
Unknown Data Requested.
>> c.d(1:2) % we didn't include subaddressing in subsref
??? Error using ==> mmrp/subsref
MMRP Objects Support Single Arguments Only.
```

同前面声明的一样，在方法函数中出现带有下标的引用和带有下标的赋值时，过载函数subsref和subsasgn不是被隐含调用的。对Matlab的多项式求值函数polyval的过载过程说明了这一点。如下面给出的方法所示：

```
function y = polyval(r,x)
%POLYVAL Evaluate Rational Polynomial Object.
% POLYVAL(R,X evaluates the rational polynomial R at the
% values in X.

if isnumeric(X)
    %y=r(x);        % what we'd like to do,but can't
    S.type='()';
    S.subs={x};
    y=subsref(r,S); % must call subsref explicitly
else
    error('Second Input Argument Must be Numeric. ')
end
```

因为subsref是为mmrp对象编写的，所以多项式求值也只仅仅是为了发布R(x)。其中R是mmrp对象，而x则包含对R进行计算时所需的值。因为这样符合我们希望polyval所执行的操作，所以在polyval中发布y=r(x)应该会引起Matlab调用subsref方法来计算有理多项式。但是因为Matlab无法在方法中调用subsref或subsasgn，所以不会发生以上情况。然而，如果要强制进行这种操作，那么可以像上面给出的代码一样明确地调用带有所希望的参数的subsref。

在处理有理多项式的时候，对带有下标的赋值过程有一个很明显的解释过程。对于一个有理多项式对象R，R(1,p)=v，其中p是一个数字向量，用于确定变量的幂数；v是一个和p一样长的数字向量。v的元素将成为p中已确定了幂数的分子多项式的系数。同样，R(2,q)=w将修改q中已确定了幂数的分母的系数。

有关subsasgn的帮助文本中描述了如何编写subsasgn方法，如下所示：

```
A = SUBSASGN(A,S,B) is called for the syntax A(I)=B, A{I}=B, or
A.I=B when A is an object. S is a structure array with the fields:
     type -- string containing '()', '{}', or '.' specifying the
             subscript type.
     subs --Cell array or string containing the actual subscripts.
```

```
For instance, the syntax A(1:2,:)=B calls A=SUBSASGN(A,S,B) where
S is a 1-by-1 structure with S.type='()' and S.subs = {1:2,':'}  A
colon used as a subscript is passed as the string ':'.
Similarly, the syntax A{1:2}=B invokes A=SUBSASGN(A,S,B) where
S.type='{}' and the syntax A.field=B invokes SUBSASGN(A,S,B) where
S.type='.' and S.subs='field'.

These simple calls are combined in a straightforward way for
more complicated subscripting expressions. In such cases
length(S) is the number of subscripting levels. For instance,
A(1,2).name(3:5)=B invokes A=SUBSASGN(A,S,B) where S is 3-by-1
structure array with the following values:
      S(1).type='()'       S(2).type='.'         S(3).type='()'
      S(1).subs={1,2}      S(2).subs='name'      S(3).subs={3:5}
```

基于此帮助文本和我们所要求的下标赋值，将创建如下的subsasgn**方法：**

```
function a=subsasgn(a,s,b)
%SUBSASGN Subscripted assignment for Rational Polynomial Objects.
%
% R(1,p)=C sets the coefficients of the Numerator of R identified
% by the powers in p to the values in the vector C.
%
% R(2,p)=C sets the coefficients of the Denominator of R identified
% by the powers in p to the values in the vector C.
%
% R(1,:) or R(2,:) simply replaces the corresponding polynomial
% data vector.
%
% For example, for the rational polynomial object
%            2x^2 + 3x + 4
% R(x)  = -------------------------
%          x^3 + 4x^2 + 5x + 6
%
% R(1,2)=5          changes the coefficient 2x^2 to 5x^2
% R(2,[3 2])=[7 8]  changes x^3 + 4x^2 to 7x^3 + 8x^2
% R(1,:)=[1 2 3]    changes the numerator to x^2 + 2x + 3

if length(s)>1
       error('MMRP Objects Support Single Arguments Only.')
 end
if strcmp(s.type,'()') % R(1,p) or R(2,p)
 if length(s.subs)~=2
     error('Two Subscripts Required.')
 end
 nd=s.subs{1}; % numerator or denominator
 p=s.subs{2};  % powers to modify
 if ndims(nd)~=2 | length(nd)~=1 | (nd~=1 & nd~=2)
     error('First Subscript Must be 1 or 2.')
 end
 if isnumeric(p) & ...
```

```
     (ndims(p)~=2 | any(p<0) | any(fix(p)~=p))
     error('Second Subscript Must Contain Nonnegative Integers.')
 end
 if ndims(b)~=2 | length(b)~=prod(size(b))
     error('Right Hand Side Must bea Vector.')
 end
  b=b(:).';     % make sure b is a row
  p=p(:)';      % make sure p is a row
  if ischar(p)  & length(p)==1 & strcmp(p,':') % R(1,:) or R(2,:)
     if nd==1   % replace numerator
         r.n=b;
         r.d=a.d;
     else     % replace denominator
         r.n=a.n;
         r.d=b;
     end
  elseif isnumeric(p) % R(1,p) or R(2,p)
     plen=length(p);
     blen=length(b);
     nlen=length(a.n);
     dlen=length(a.d);
      if plen~=blen
          error('Sizes Do Not Match.')
     end
      if nd==l  % modify numerator
          r.d=a.d;
          rlen=max(max(p)+1,nlen);
          r.n=zeros(1,rlen);
          r.n=mmpadd(r.n,a.n);
          r.n(rlen-p)=b;
      else     % modify denominator
          r.n=a.n;
          rlen=max(max(p)+1,dlen);
     r.d =zeros(1,rlen);
     r.d=mmpadd(r.d,a.d);
     r.d(rlen-p)=b;
    end
  else
    error('Unknown Subscripts. ')
  end
else % R { } or R.field
   error('Cell and Stucture Addressing Not Supported. ')
end
a=mmrp(r.n,r.d,a.v);
```

下面给出一些使用此方法的例子：

```
>> a = mmrp([3 1],[1 2 5 10]) % create test object
a =
```

```
MMRP Rational Polynomial Object:
      3x^1 + 1
--------------------
x^3 + 2x^2 + 5x^1 + 10
>> a(1,:) = [1 2 4]      % replace entire numerator
a =
MMRP Rational Polynomial Object:
   x^2 + 2x^1 + 4
--------------------
x^3 + 2x^2 + 5x^1 + 10
>> a(2,2) = 12          % replace x^2 coef in denominator
a =
MMRP Rational Polynomial Object:
    x^2 + 2x^1 + 4
-----------------------
x^3 + 12x^2 + 5x^1 + 10
>> a(1,0) = 0           % replace 4x^0 with 0x^0
a =
MMRP Rational Polynomial Object:
     x^2  +  2x^1
-----------------------
x^3 + 12x^2 + 5x^1 + 10
>> a(1,:)=a.d           % subsref and subsasn! (a.n and a.d cancel)
a =
MMRP Rational Polynomial Object:
1
```

33.4 转 换 函 数

正如本章前面所演示的，double、char、logical**函数都可以将它们的输入转换成和它们的函数名字相同的数据类型。例如：**double('hello')**可以将字符串“** hello **”转换成相应的**ASCII**代码。无论是否会用到这些函数，都应该将这些转换函数的方法包含到类库中。对于**mmrp**对象来说，很明显需要**double**和**char**函数。**double**方法应该用于提取多项式的分子和分母，**char**方法应该用于创建字符串表达式，例如**display.m**所显示的字符串。函数如下所示：**

```
function [n,d]=double(r)
%DOUBLE Convert Rational Polynomial Object to Double.
% DOUBLE(R) returns a matrix with the numerator of R in
% the first row and the denominator in the second row.
% [N,D]=DOUBLE(R) extracts the numerator N and denominator D
% from the rational polynomial object R.

if nargout<=1 & length(r,d)>1
    nlen=length(r.n);
    dlen=length(r.d);
    n=zeros(1,max(nlen,dlen));
    n=[mmpadd(n,r.n);mmpadd(n,r.d)];
elseif nargout<=1
    n=r.n;
```

```
else % nargout==2
    n=r.n;
    d=r.d;
end

function [n,d,v]=char(r)
%CHAR Convert Rational Polynomial Object to Char.
% CHAR(R) returns a 3-row string array containing R in the
% format used by DISPLAY.M
% (N,D)=CHAR(R) extracts the numerator N and denominator D
% as character strings from the rational polynomial object R.
% [N,D,V]=CHAR(R) in addition returns the variable V.

if nargout<=1
  nstr=mmp2str(r.n,r.v);
nlen=length(nstr);
 if length(r.d)>1
     dash=,'';
     dstr=mmp2str(r.d,r.v);
     dlen=length(dstr);
     m=max(nlen,dlen);
     n=char([blanks(ceil((m-nlen)/2)) nstr]  ....
         dash(ones(1,m))  ....
         [blanks(fix((m-dlen)/2)) dstr]);
 else
     n=nstr;
 end
elseif nargout>1
    n=mmp2str(r.n); % converts polynomial to string
   d=mmp2str(r.d);
end
if nargout>2
     v=r.v;
end
```

33.5 优先级、继承和集合

Matlab会自动赋予用户定义的类比Matlab内部的类更高的优先级。因此，同时带有内部的和用户定义的类的运算符和函数通常会调用用户定义的类的方法。对于简单的类而言，这种默认的优先级就足够了。但是如果存在多个用户定义的类，那么就需要有一种机制来允许用户控制各个类的优先级别。也有可能用户会希望强制用户定义的类的优先级低于内部的类。Matlab中的inferiorto和superiorto函数为用户提供了控制优先级的这种能力。这两个函数只能在类的构造器函数中使用。对这两个函数的声明中包含一个字符串列表，这些字符串标识了优先级高于或者低于由构造器创建的对象的各个类。例如，superiorto('double')

指定对象的优先级高于双精度变量；inferiorto('mmrp', 'char')指定对象的优先级低于mmrp和char对象。

对于大型的编程项目而言，为对象类型分层次会为编程带来很多方便。在这种情况下，可以便于让一种对象类型从另一种对象类型继承方法。如果这样做，那么就可以只编写几个方法，而着重方法的修改了。在OOP的单词表中，继承其他类的属性的对象被称为子类，而被继承的类被称为父类。最简单的情况就是子类从单个父类继承方法的情况。这被称为简单继承。如果一个子类从多个类继承了方法，则被称为多次继承。

在简单继承中，将父类的所有字段都赋予了子类，并赋予子类一个或多个它自己独有的字段。因此，与父类相关联的方法可以直接应用在子类的对象上。而父类的方法中自然也没有子类独有的字段的任何信息，更不可能使用这些字段。同样，子类独有的这些方法也无法访问父类的字段。子类必须使用从父类继承的方法来访问父类的字段。例如，Control Toolbox中的lti对象就拥有tf、zpk、ss、dss、frd 5个子类。

在多次继承中，子类将拥有所有父类的所有字段，再加上一个或多个它自己独有的字段。与简单继承相同，父类和子类都不能直接访问对方的字段。可能会有很多种情况出现，在这里很难一一介绍。在有多个父类的情况下，很难决定在何种情况下，该调用哪个父类以及该父类的何种方法的复杂性。可以在Matlab的文档中找到关于继承的详细信息和例子。

在前面用做例子的mmrp类中，对象字段包含作为Matlab的double和char类的元素的数据。实际上，没有理由阻止对象拥有包含用户定义的类的其他数据类型。在OOP的词汇表中，这被称为密封或集合。被过载的运算符和函数的规则并不发生改变。在一个类的方法函数中，如果需要对初始类的字段进行运算，那么将调用其他类的方法。

第 34 章　Matlab 编程接口

Matlab为外部程序提供了多种接口。在Matlab中可以使用MEX文件来调用C函数和FORTRAN子程序。此外，通过使用Matlab引擎，Matlab可以执行运算并向C或FORTRAN程序返回结果。Matlab为创建和访问标准的Matlab MAT文件提供了头文件和库。另外，Matlab可以和Java结合使用；可以通过使用动态数据交换（Dynamic Data Exchange， DDE）与PC应用程序进行数据交换；还可以作为ActiveX服务器与Visual Basic （VB）应用程序或使用Visual Basic for Application（VBA）的PC应用程序（例如Microsoft Excel、PowerPoint、Word）进行通信。有关Java、DDE、ActiveX的内容将在后面的章节中进行介绍。本章将着重介绍C和FORTRAN的接口。

Matlab提供了一组非常齐全的外部编程接口，这些接口远远超出了我们在这里所介绍的。对于Matlab的应用程序接口（API）的详细介绍足可以另写一本书了。本章和后面的两章只是采用实例对Matlab的API特性进行了一些建设性的介绍。读者可以在Matlab的文档中找到更加详尽的介绍。

为了避免产生混乱，本章中的讨论和例子都是基于使用UNIX平台下的C编译器的。本章中的例子都是使用具有2.2.14版内核的Linux i386平台上（glnx86体系结构）的GCC编译器进行开发和测试的。在Windows NT4.0 Pentium PC环境下（win32体系结构）使用Matlab提供的LCC编译器来测试例子，在Linux平台下使用G77 FORTRAN编译器来测试FORTRAN代码。

34.1　访问Matlab数组

任何编写Matlab接口程序的人都应该首先理解Matlab数组的结构。任何MEX、MAT或引擎程序都一定会访问Matlab的数组以执行有用的操作。在本节中将粗略解释Matlab是如何存储数组的，还将列出可以在编程中用于访问Matlab数组的所有函数。

Matlab 数组

Matlab只支持一种对象——Matlab数组。所有的Matlab变量都是Matlab数组。Matlab数组的元素是由6种基本数据类型（double、uint8、char、cell、struct、sparse）或其他Matlab数组组成的。所有的标量和组合元素，例如：字符串、向量、矩阵、单元数组、结构和对象，都是Matlab数组。

Matlab按列存储数组数据。当数据从一个字符串缓冲区中提取出来，并重组到单独的字符串中时，将会在mmcellstr的MEX文件中按列存储数组数据（MEX例4）。下面的例子解释了数据的组织形式。

```
>> x = [1 2 3; 4 5 6; 7 8 9]
```

```
x =
    1   2   3
    4   5   6
    7   8   9
>> size(x)
ans =
    3       3
>> x(:)'
ans =
    1   4   7   2   5   8   3   6   9
```

访问 Matlab 数组：mxArray

在C语言中，可以认为Matlab数组是一种新的C数据类型——由C结构建立的一个被称为mxArray的对象。mmArray结构存储了有关数组类型（double、sparse、cell等）的信息、数组的维数和数据本身。mxArray结构还存储了特定类型的信息，例如：数字数组的复杂性（实数或复数），结构或对象的字段的数量和名称，稀疏数组的下标和非零元素的最大数量。MX函数是用于访问mxArray的特征曲线和元素的方法。

数字矩阵是作为两个double类型（或其他数据类型）向量来存储的；一个向量存储实数部分，另一个向量存储虚数部分。使用两个指针来访问数据；一个指针指向实数向量（pr），另一个指针指向虚数向量（pi）。如果数组是实数数组，那么pi指针将为NULL。字符串将作为不带有虚数部分的16位的ASCII Unicode整数存储。C字符串是以空结束的，但Matlab字符串则不是。Matlab字符串的长度可以从数组的维数中获得。

单元数组是数组的集合，数据向量包含指向其他数组的指针，在这里没有虚数部分。指针向量中的每个指针都被称为一个单元。结构以折叠的单元数组的形式存储，在这些数组中，数据向量的每个元素都是一个字段。每个字段都和存储在mxArray的另一个结构元素中的名字相关联。对象是作为已命名的结构（类名）来存储的，在带有已注册的方法集。单元数组、结构和对象都不带有虚数部分。

mxArray中有一个元素存储了作为整数向量执行的数组大小。每个整数都对应相关维数的长度。如果向量中元素的数量大于2，那么mxArray就是一个多维数组。如果这些维数中的任意一维是零，那么数组将被认为是一个空数组。可以使用其他mxArray结构元素将非复数数组标记为逻辑数组。

稀疏矩阵包含前面所提到的那些元素：两个数字向量及指向它们的指针。这些向量包含有矩阵的非零元素。稀疏数组还包含有nzmax、ir和jc参数。在这里，ir是指向对应非零元素的行下标的数组的指针，jc是指向列下标的向量的指针。稀疏矩阵中非零元素的最大数量存储在nzmax中。

MX 函数

Matlab为从C或FORTRAN程序中访问和处理mxArray提供了100多种函数和子程序。需要注意的是，FORTRAN只支持Matlab的双精度和字符串数据类型，而C可以访问Matlab的所有数据类型。FORTRAN逻辑子程序（mxIs…）对真返回1，对假返回0。C的mxCreate…函数通常有两种版本：mxCreat…Matrix函数用于两维mxArray，mxCreat…Array用于n维

mxArray。FORTRAN的mxCopyPtrTo…函数用于从mxArray向FORTRAN数组拷贝数据，mxCopy…ToPtr子程序用于从FORTRAN数组向mxArray拷贝数据。这些子程序通常被用于向FORTRAN数组复制mxArray、向子程序发送数据、将结果从FORTRAN数组复制回mxArray。本章后面给出的例子将解释这些函数和子程序的用法。有关MX函数和子程序及其参数和返回值的完整列表可以在Matlab的文档中找到。

34.2 从Matlab中调用C或FORTRAN

MEX文件是已编译的C或FORTRAN函数，它们可以从Matlab环境中调用，如果这些函数完全像M文件的函数一样的话。可以通过加入几行代码来修改已有的C函数或FORTRAN子程序，使其可以访问Matlab的数据和函数。使用mex命令编译修改后的代码会生成可以在Matlab中调用的MEX文件。使用Matlab的M文件时，大多数计算函数会执行得更加迅速、效率更高。然而，某些像For循环这样的代码在C或FORTRAN中执行的效率会更高。如果无法通过向量化的方法来消除迭代，那么可以试着创建一个MEX文件来解决。

编译后的MEX文件带有特定平台的文件扩展名。例如，在Windows平台下，MEX文件的扩展名为dll，在Sun Solaris平台下，MEX文件的扩展名为mexsol。在任意平台下，mexext函数都将返回相应的MEX文件扩展名。

你所创建的每个MEX文件都应该有一个与之相关联的M文件（myfunc.m）来为MEX函数提供帮助文本。Matlab文档中提供了关于使用其他体系结构和其他编译器的附加信息。

准备 MEX 环境

MEX环境必须要经过初始化，才能访问已安装的C或FORTRAN编译器。Matlab可以支持多种编译器，甚至包括一些免费的编译器，例如GCC编译器（包括对FORTRAN的支持）。可以从因特网上的许多站点下载GCC编译器，例如从GCC的主页http://gcc.gnu.org/。还可以选择在PC平台上使用与Matlab同时提供的LCC编译器。同时还支持标准的第三方ANSI C编译器和FORTRAN编译器，例如PC平台上的Microsoft和Borland编译器，Linux和BSD平台上的GCC编译器，以及那些Compaq、Sun和HP服务器上由供应厂商提供的选装的ANSI C编译器。

某些编译器在默认情况下并不屏蔽浮点异常。比如Matlab数组允许无穷值，例如Inf和NaN，如果没有将它们屏蔽起来，那么他们将造成浮点异常。Alpha平台上的DEC FORTRAN编译器、Linux下的Absoft FORTRAN和Windows PC平台上的Borland C++都会受到影响。在Matlab和编译器的文档中都包含描述如何在编译MEX、引擎和MAT文件程序时屏蔽浮点异常的信息。

>>mex –setup命令将为编译器和计算机平台（操作系统和体系结构）选择适当的初始化文件或选项文件。选项文件将为平台和编译器设置某些环境变量，并指定相应的头文件和库的存储位置。一旦对MEX环境进行了初始化，则可以使用>>mex myprog.c命令来将MEX源文件myprog.c编译成编译后的MEX文件myprog.dll（或myprog.mexglx等等）。mex命令行选项-f可以用来在需要的情况下临时选择一个不同的初始化文件。我们将在后面编

译Matlab引擎和MAT程序时用到-f选项。-v选项可以用来列出编译器的设置，以及观察编译和连接的阶段。>>mex –help命令能够列出所有mex命令行选项。mex同时也支持其他标准的编译器选项，例如：-c、-g、-D。如果使用-g选项，那么可以使用调试器（例如dbx或gdb）来调试MEX程序。mex命令既可以作为一个函数由Matlab命令窗口调用执行，也可以作为批处理或脚本文件在Matlab环境外使用。有关mex命令、命令行选项、所支持的编译器和调试过程的详细内容可以参见Matlab的文档。

> MEX文件必须要设计为可以当作能够使用大量输入参数和输出参数的标准Matlab函数来操作。普通的Matlab函数通过数值而非参量来传递数据。MEX函数的输入是不能修改的。如果某个函数的目的是修改输入，那么被修改的值必须作为输出返回。例如：如果inc函数要将输入变量b的值增大，那么应该使用b=inc(b)语句来调用inc函数。

MEX 源文件的组织

MEX源代码文件可以分为头、子程序、接口三个部分。头部分中包含所有所需的#include和#define命令。接口部分（在Matlab的文档中也称为网关（gateway）部分包含从自己的程序访问Matlab的数据和函数所用的代码。子程序部分（在Matlab文档中也称为计算（computation）部分）实际执行数据的运算，这部分可以作为单独的函数或子程序来编写，或作为可以合并到接口部分的代码来编写。

除必需的包含文件外，头部分还必须包含一个#include "mex.h"命令，以提供对MEX函数的支持。mex前缀将指定那些只能在MEX文件中使用、在Matlab工作区中运算的函数。"mex.h"头文件也包含"matrix.h"，以提供MX函数来支持Matlab数据类型和标准的头文件<stdio.h>，<stdlib.h>和<stddef.h>。mx前缀指定了以Matlab数据类型进行运算的函数。

接口部分用于与Matlab环境进行通信。所需的函数定义是mexFunction声明，它是程序入口点，相当于独立C程序中的main函数。mexFunction函数的定义如下：

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

nlhs变量是一个整数，它代表左侧（输出）参数的数量，其与Matlab的nargout函数返回的数字相等。与之类似，nrhs代表右侧（输入）参数的数量。prhs[ ]是一个指向输入参数的指针数组，prhs[0]是指向第一个输入参数的指针，prhs[1]是指向第二个参数的指针，依此类推。所有的Matlab数据类型都是Matlab数组，因此所有MEX文件的输入和输出参数都是mxArray数据类型的。

在调用已编译的MEX函数的时候，prhs将包含指向输入参数的指针；plhs则包含空指针。程序员需要创建任意的输出数组，并向plhs指针数组赋予数组指针。即使在nlhs包含0的情况下，即不要求有任何输入参数的情况下，也可以创建输出数组。如果从命令窗口调用已编译的MEX函数，那么在MABLAB基本工作区中，所有的输出都被赋给ans变量。

工作区的问题

在调用Matlab的M文件函数的时候，函数将会在一个与Matlab基本工作区或发出调用的

函数的工作区完全不同的独立工作区中进行运算。在基本工作区或调用程序的工作区中的变量是通过数值而非参量进行传递的，它们不受被调用函数的影响。evalin和assignin函数除外，这两个函数可以影响当前工作区以外的变量。

MEX函数在它们自己的环境中进行运算。在这个环境中，局部变量的范围被限制在MEX函数之内。然而，此环境和前面的Matlab工作区并不完全相同。MX函数（带有mx前缀的函数）以Matlab数据类型进行运算。MEX函数（带有mex前缀的函数）在Matlab工作区内进行运算。例如：mexEvalString和mexCallMatlab函数可以在调用程序的工作区中计算它们的字符串参数。mexGetArray函数可以从指定的Matlab工作区（基本、调用者、全局）拷贝变量到mxArray中。mexGetArrayPtr可以获得一个指向Matlab变量的只读指针。mexPutArray可以将mxArray拷贝到指定的Matlab工作区中。所有其他的MEX函数在调用程序的工作区中进行运算。

最后一个特例是用于打印格式化字符串的mexPrintf函数。在MEX文件中最好不要使用标准C的printf函数，而应该使用mexPrintf。mexPrintf函数将调用基本工作区中的Matlab的printf函数，然后在“命令”窗口中打印（如果使用了日志文件，那么也将打印到日时志中）。

下面列出了C程序和FORTRAN程序中的MEX函数。

函数	C	F	功能
mexAtExit	C	F	注册一个在MEX文件被清除时调用的函数
mexCallMATLAB	C	F	调用一个Matlab函数、M文件或MEX文件
mexErrMsgTxt	C	F	发出一条错误信息，然后返回到Matlab中
mexEvalString	C	F	在调用程序的工作区中执行一条Matlab命令
mexFunction	C	F	MEX文件的入口点
mexFunctionName	C		当前的MEX函数的名字
mexGet	C		获得句柄图形属性的值
mexGetArray	C		从其他工作区获得一个变量的拷贝
mexGetMatrix		F	从调用程序的工作区拷贝一个mxArray
mexGetFull		F	获得双精度mxArray的组成部分
mexGetArrayPtr	C		从其他工作区获得指向变量的只读指针
mexGetMatrixPtr		F	获得调用程序工作区中指向mxArray的指针
mexGetGlobal		F	获得一个指向全局mxArray的指针
mexGetInf		F	获得无穷的值
mexGetNaN		F	获得NaN的值
mexGetEps		F	获得eps的值
mexIsGlobal	C		如果mxArray是全局范围的，则为真
mexIsFinite		F	判断一个值是否有限
mexIsInf		F	判断一个值是否为无穷
mexIsNaN		F	判断一个值是否为NaN
mexIsLocked	C		如果MEX文件是锁定的，则为真

续表

函数	C	F	功能
mexLock	C		锁定MEX文件，这样就不能从内存中清除MEX文件了
mexUnlock	C		为MEX文件解锁，这样就可以从内存中清除MEX文件了
mexWarnMsgTxt	C		发布一条警告信息
mexPrintf	C	F	ANSI C的printf风格的输出程序
mexPutArray	C		将一个mxArray拷贝到Matlab工作区中
mexPutMatrix		F	将一个mxArray写入到调用程序的工作区中
mexPutFull		F	在调用程序工作区中创建一个mxArray
mexSet	C		为句柄图形属性设置值
mexSetTrapFlag	C	F	控制mexCallMatlab对错误的响应
mexMakeArrayPersistent	C		令mxArray可以在MEX文件完成后持续存在
mexMakeMemoryPersistent	C		令由mxMalloc和mxCalloc分配的内存持续存在

MEX 例 1：fact

本例创建了一个计算阶乘函数的MEX文件，在本例中略去了注释和错误检查。

```
/*
 * fact.c - returns the factorial of a nonnegative integer.
 *
 *   MATLAB usage:  p=fact(n)
 *
 * Mastering MATLAB 6 C MEX Example 1
*/
#include  "mex.h"

void mexFunction( int nlhs,        mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )
{
    double n, j, *p;
    int i ;

    n=mxGetScalar(prhs[0]);
    plhs[0]=mxCreateDoubleMatrix(1.1,mxREAL);
    p=mxGetPr(plhs[0]) ;

    j=1.0;
    for (i=n ; i>1; i--)
        j=j*i;
    *p=j;
}
```

首先包含了mex.h头文件，接着声明了mexFunction函数和所需的参数。随后只声明了

几个变量，输入参数的值通过mxGetScalar函数获得。然后使用mxCreateDoubleMatrix函数创建了一个1×1的双精度实数输出矩阵，并使用mxGetPr函数获得了一个C指针。最后执行运算，运算结果被赋给输出矩阵。

>>mex fact.c将创建一个可以从Matlab中调用的MEX文件。所生成的MEX文件的文件名由计算机平台来决定：在Windows PC上为fact.dll；在Linux平台上为fact.mexglx。如果C源文件中有错误，那么编译器将显示错误信息，以帮助对源代码进行必要的修正。如果发现了错误，则不会创建MEX输出文件。

运行编译好的MEX文件将产生如下的输出：

```
>> x =5
x =
    5
>> y = fact(x)
y =
    120
>> which fact
/home/work/matlab/fact.mexglx
```

在这些例子中只使用了一些非常普通的MX和MEX函数，还有大量的函数没有用到。有关Matlab接口函数及其参数和返回值，即Matlab应用程序编程接口的完整内容可以参见HTML和PDF格式的文件。HTML版本的文件可以在Matlab Help Browser中浏览。PDF版本的《API指南》和《API参考》可以使用Adobe Acrobat Viewer来浏览，或直接在打印机上打印。

MEX 例 2：mycalc

本例介绍了更多的元素。在mycalc中包含错误检查，并使用了一个子程序来执行对数组元素的计算。通过自己有的C子程序原代码中添加接口部分代码，加入mex.h头文件并编译，这种用法可以将C子程序转化为MEX文件，包含mex.h命令并编译，从而转变成MEX文件。

MEX文件mycalc的输入是两维的双精度数组，返回的值是对输入元素进行了计算后的同样大小的数组。接口部分用于进行一些错误检查：决定输入数组的维数、创建指向输入和输出数组的C指针、调用子程序进行运算。

```
/*
 * mycalc.c - calculates x^2-x+1/x for each element of an array.
 *
 *   MATLAB usage: p=mycalc(n)
 *
 * Mastering MATLAB 6 C MEX Example 2
 */

#include "mex.h"

/* This is the original subroutine that performs the calculation. */
```

```
static void myca1c( double p[], double n[], int r, int c)
{
    int i ;
    for (i=0;i<r*c;i++)
        p[i]=n[i]*n[i]-n[i]+1.0/n[i];
}

/* This is the interface to MATLAB data types and arguments. */
void mexFunction( int nlhs.          mxArray *plhs[],
                  int nrhs. const  mxArray *prhs[] )
{
    double *p, *n;
    int r, c ;

    /* Do some error checking. */
    if (nrhs != 1)
        mexErrMsgTxt("One input argument required.");
    else if (nlhs > 1)
        mexErrMsgTxt("Too many output arguments.");
    else if (!mxIsNumeric(prhs[0]))
        mexErrMsgTxt("Input must be numeric.");
    else if (mxIsComplex(prhs[0]))
        mexErrMsgTxt("Input must be real.");
    else if (mxGetNumberOfDimensions(prhs[0]) > 2)
        mexErrMsgTxt("N-Dimensional arrays are not supported.");

/* Get the input array dimensions. */
r=mxGetM(prhs[0]);
c=mxGetN(prhs[0]);

/* Create a matrix for the return argument */
plhs[0]=mxCreateDoub1eMatrix(r,c,mxREAL) ;

/* Assign pointers to the parameters. */
p=mxGetPr(plhs[0]);
n=mxGetPr(prhs[0]);

/* Do the actual calculation in a subroutine. */
myca1c(p,n.r,c);
}
```

在MEX的C源文件中首先包含的是子程序，这样做可以减少许多麻烦，因为如果提前引用了子程序，就必须先对原型进行声明。mexErrMsgTxt**函数与**Matlab**的**error**函数的功能相同。错误信息文本将被打印在**Matlab**的命令窗口中，随后将退出**MEX**函数。请注意：要使用标准C语言中的双引号来分隔开字符串参数。可以使用**mexWarnMsgTxt**函数来仿效**Matlab**的**warning**函数，在不退出**MEX**文件的情况下在命令窗口中打印错误信息文本。**

mxGetM和mxGetN函数可以分别返回二维mxArray的行数和列数。语句：

```
plhs[0]=mxCreateDoubleMatrix(r,c,mxREAL);
```

可以创建一个r×c的mxArray，它的元素为Matlab的double类型实数。mxCOMPLEX标记，而非mxREAL标记指出需创建一个包含复数的数组。mxGetPr函数返回指向mxArray的C风格的指针。mxGetPr返回指向mxArray的实数元素的指针（使用mxGetPi函数获得一个指向复数mxArray的虚部的指针）。最后一步是以适当的参数调用mycalc子程序。

MEX 例 3：count

本例支持多维数组。在本例中使用了#define声明来简化源代码，并将Matlab帮助文本包含在MEX文件count中。

```
/*
 * count.c - count occurrences of values in an array.
 *
 * MATLAB usage:  c=count(a ,b,to1 )
 *
 * Mastering MATLAB 6 C MEX Example 3
 */

#include <math.h>
#include "mex.h"

/* Define some variables to make life easier. */
#define A    prhs[0] /* Pointer to first right-hand-side argument  */
#define B    prhs[l] /* Pointer to second right-hand-side argument */
#define TOL prhs[2] /* Pointer to third right-hand-side argument  */
#define C    plhs[0] /* Pointer to first 1eft-hand-side argument   */

void  mexFunction( int nlhs,         mxArray *plhs[],
                   int nrhs, const   mxArray *prhs[] )
{
    int i, j, sizea, sizeb;
    double tol, vcount;
    double *a; *b, *c;

    /* Do some error checking */
    if (nrhs < 2)
        mexErrMsgTxt("Missing input arguments.");
    else if (nrhs > 3)
        mexErrMsgTxt("Too many input arguments.");
    else if (nlhs > 1)
        mexErrMsgTxt("Too many output arguments.");

    /* Get tolerance value if supplied, otherwise use EPS. */
    if (nrhs == 3) {
```

```
        if (!mxIsNumeric(TOL) || mxIsComplex(TOL) ||
            mxGetNumberOfElements(TOL) != 1 )
        mexErrMsgTxt("TOL must be a real numeric scalar.");
    tol=mxGetScal ar(TOL);
    if (tol < mxGetEps())
        mexErrMsgTxt("TOL must be a positive value.");
    }
    else
        tol=mxGetEps();

    /* Make sure input arrays are non-complex numeric arrays. */
        if (!mxIsNumeric(A) ||  !mxIsNumeric(B) ||
            mxIsComplex(A) || mxIsComplex(B))
        mexErrMsgTxt("Input arguments must be real of type double.");

    /* Create the output mxArray the same size as A */
    C=mxCreateNumericArray(mxGetNumberOfDimensions(A),
        mxGetDimensions(A),mxDOUBLE_CLASS,mxREAL);

    /* Get the number of elements in A and B and create pointers */
    /* to the input and output arrays. */
    sizea=mxGetNumberOfElements(A);
    sizeb=mxGetNumberOfElements(B);
    a=(double *) mxGetPr(A);
    b=(double *) mxGetPr(B);
    c=(double *) mxGetPr(C);

    /* Cycle through the elements of the arrays and count values */
    for (i=0;i<sizea;i++) {
        vcount=0.0;
    for (j=0;j<sizeb;j++)
    if ((fabs(a[i]-b[j])) <= tol)
        vcount++ ;
    c[i]=vcount;
}
}
```

在本例中，除了包含mex.h**头文件外还包含了标准的**math.h**头文件，以支持在计算过程中使用的**fabs**函数。此外，还使用了**4**个**#define**声明来减少输入代码，并且提高了代码的稳定性。下面这行代码：**

```
C=mxCreateNumericArray(mxGetNumberOfDimensions(A),
        mxGetDimensions(A),mxDOUBLE_CLASS,mxREAL);
```

是创建n**维**mxArray**的**MX**函数。大多数**mxArray**创建函数都有两种不同的形式：**2**维形式和**n**维形式。其**2**维形式如下：**

```
C=mxCreateNumericMatrix(mxGetM(A),mxGetN(A),mxDOUBLE_CLASS,mxREAL);
```

2维形式（mxCreate…Matrix）中首先应该是独立的行数和列数参数。随后是其他参数（在本例中是数字类和实数/复数特征位）。n维形式还要求在其他参数后面给出维数和维数大小向量。例如：

```
Plhs[0]=mxCreateCellMatrix(3,4);
```

将创建一个空的3×4的mxArray，同时：

```
Plhs[0]=mxCreateCellArray(3,[2,3,4]);
```

将创建一个3维的2×3×4的mxArray。

mxGetNumberOfElements函数可以返回mxArray中所有元素的个数。因为子程序使用单个下标来标记要处理的数组元素，所以将向子程序传送元素总数，而非数组的维数。

最后，我们将创建一个可以为count函数提供相应的Matlab帮助文本的M文件。此文件将被命名为count.m，存储在和MEX文件count.mexglx相同的目录中。

```
function c=count(a,b,tol)
%COUNT Count Occurrences of Values in an Array.
% COUNT(A,B) returns an array the same size as A whose i-th element
% contains the number of times A(i) appears in the array B.
% A and B must be Real arrays.
%
% COUNT implements the following:
%           c = zeros(size(A)));
%           for i=1:prod(size(A))
%              c(i)=sum(A(i)==B);
%           end
```

当Matlab发现MEX文件和M文件在同一目录中的时候，Matlab会调用MEX文件来执行函数，同时使用M文件来提供帮助文本。

MEX 例 4：mmcellstr

本例离开了数字的范围，进入到了字符串和单元的世界。mmcellstr函数是Matlab的cellstr函数的一个C语言MEX文件实现。mmcellstr从一个字符数组中创建一个字符串单元数组。字符数组的每一行都被放置在单元数组的一个独立的单元中。

在本例中展示了许多新的技术，包括：在MEX文件中分配内存、处理空数组和复制数组。

```
/*
 * mmcellstr.c - Create a cell array of strings from a 2-D character array,
 *
 * MATLAB usage: c=mmcellstr(s)
 *
 * Mastering MATLAB 6 C MEX Example 4
 */
```

```
#include "mex.h"

void mexFunction( int nlhs,         mxArray *plhs[],
                  int nrhs, const  mxArray *prhs[] )
{
    int m, n, i, j;
    char *buf;
    char **line;

    /* Do some error checking */
    if (nrhs < 1)
        mexErrMsgTxt("Missing input argument.");
    else if (nrhs > 1)
        mexErrMsgTxt("Too many input arguments.");
    else if (mxGetNumberOfDimensions(prhs[0]) != 2)
        mexErrMsgTxt("Input must be 2-D.");
    else if (nlhs > 1)
        mexErrMsgTxt("Too many output arguments.");

/****** Start of Region 1 ******/
    /* If the input is already a cell array, duplicate it. */
    if (mxIsCell(prhs[0]))
        if (mxIsChar(mxGetCell (prhs[0],0))) {
            plhs[0]=mxCreateCellMatrix(mxGetM(prhs[0]),mxGetN(prhs[0]));
            plhs[0]=mxDuplicateArray(prhs[0]);
            return;
        }
/****** End of Region 1 ******/

/* Make sure the input is a character array. */
if (!mxIsChar(prhs[0]))
    mexErrMsgTxt("Input must be a character array.");

/****** Start of Region 2 ******/
    /* Handle the empty input case. */
    if (mxIsEmpty(prhs[0])) {
        plhs[0]=mxCreateCellMatrix(1, 1) ;
        mxSetCell(plhs[0],0,mxDuplicateArray(prhs[0]));
        return;
    }
/****** End of Region 2 ******/

    /* Determine the dimensions of the input structure array */
    m=mxGetM(prhs[0]);
    n=mxGetN(prhs[0]);

/****** Start of Region 3 ******/
```

```
    /* Stuff the input into a string buffer. */
    buf=mxCalloc(m*n.sizeof(char));
    buf=mxArrayToString(prhs[0]);

    /* Create line buffers for the individual strings. */
    for (i=0;i<m;i++)
        line[i]=mxCalloc(n+1 ,sizeof(char));
/****** End of Region 3 ******/

    /* Parse the buffer into individual lines. */
    for (j=0;j>n;j++)
        for (i=0;i<m;i++)
          line[i][j]=buf[i+m*j];

    /* Free the string buffer and create the output cell array. */
    mxFree(buf);
    plhs[0]=mxCreateCellMatrix(m, 1);

    for (i=0;i<m;i++) {
        /* For each line, remove trailing blanks... */
        j=n;
        while (--j >= 0)
            if (line[i][j] != ' ' )
                break;
        line[i][j+l]='\0';

        /* insert the line into the output array... */
        mxSetCell(plhs[0],i,mxCreateString(line[i]));

        /* and free the line buffer memory. */
        mxFree(line[i]);
    }
}
```

在头部分和函数定义部分定义了一些变量。buf**是一个指向字符数组（字符串缓冲区）的指针，**line**是一个指向字符指针数组的指针。在函数的后半部分为这些缓冲区分配了内存。随后还进行了一些错误检查。再往下的部分值得我们仔细研究一下。**

```
/****** Start of Region 1 ******/
    /* If the input is already a cell array,duplicate it.*/
    if (mxIsCell(prhs[0]))
        if (mxIsChar(mxGetCell(prhs[0],0))){
    plhs[0]=mxCreateCellMatrix(mxGetM(prhs[0]),mxGetN(prhs[0]));
    plhs[0]=mxDuplicateArray(prhs[0]);
    return;
    }
/****** End of Region 1 ******/
```

在这部分中要处理的是已经成为字符串单元数组的输入参数。在这种情况下，数组将被复制然后传送给输出参数。首先，我们要检查输入是否是一个单元数组（mxIsCell）。如果是，那么我们将检查单元内容的类。mxGetCell(prhs[0],0)可以返回输入数组的单元0的内容，然后使用mxIsChar判断元素是否包含字符数据。下面的代码：

```
plhs[0]=mxCreateCellMatrix(mxGetM(prhs[0]),mxGetN(prhs[0]));
```

将创建一个和输入数组具有相同维数的输出数组。然后mxDuplicateArray会将输入的mxArray深层拷贝(deep copy)到输出mxArray中。深层拷贝将拷贝数组中的所有层次；它在本质上是一种递归拷贝。在这种情况下，代码：

```
Plhs[0]=mxDuplicateArray(prhs[0]);
```

将输入数组中的所有内容都拷贝到输出数组中，然后函数将返回到正在调用的工作区中。

在这里，我们已经得到了参数的正确类型和数量。在下面所示的第三部分中，我们获得了输入字符串数组的维数，使用mxCalloc为字符缓冲区分配合适的大小（m×n），使用mxArraytoString将字符数组作为一个长字符串拷贝到新的缓冲区中。然后为输入数组的每一行分配一个行缓冲区。每个行缓冲区的大小等于字符数组的宽度减去一个C语言中的字符串结束符号“\0”。

```
/****** Start of Region 3 *******/
        /* Stuff the input into a string buffer. */
        buf=mxCalloc(m*n,sizeof(char));
        buf=mxArrayToString(prhs[0]);

        /* Create line buffers for the individual strings. */
        for (i=0;i<m;i++)
          line[i]=mxCalloc(n+1,sizeof(char));
/****** End of Region 3 *******/
```

MEX文件应该始终使用mxCalloc（或mxMalloc）来分配内存，并使用mcFree来将已分配的内存返回到堆栈中。这些函数将在Matlab内存管理器中注册内存的分配和解除分配，Matlab内存管理器将在函数退出的时候释放所有已分配的内存。

因为Matlab数组是以列存储的，所以缓冲区将按列顺序保存输入数组的元素。这与使用Matlab冒号运算符（buf=chararray(:)）的结果相类似。下面的代码将从buff中提取正确的元素存入到线性缓冲区中，释放字符串缓冲区内存，最后创建输出单元数组。然后将处理每一行。在缓冲区的最后一个非空字符前插入C语言的字符串结束符号（'\0'），然后复制该行，使用mxCreatString函数将它从C语言的字符串转换成为Matlab字符串，并使用mxSetCell函数将其插入到输出单元数组中。最后，使用mxFree函数释放该行的缓冲区内存，然后处理下一行。

MEX 例 5：mmv2struct

这个例子演示了一种将M文件转换成等效的MEX文件的方法。下面给出的

mmv2struct.m函数将独立的变量打包并解包到一个标量结构中。

```
function varargout=mmv2struct(varargin)
%MMV2STRUCT Pack/Unpack Variables to/from a Scalar Structure.
% MMV2STRUCT(X,Y,Z,...) returns a structure having fields X,Y,Z,...
% containing the corresponding data stored in X,Y,Z,...
% Inputs that are not variables are stored in fields named ansN
% where N is an integer identifying the Nth unnamed input.
%
% MMV2STRUCT(S)assigns the contents of the fields of the scalar structure
% S to variables in the calling workspace having names equal to the
% corresponding field names.
%
% [A,B,C,...]=MMV2STRUCT(S) assigns the contents of the fields of the
% scalar structure S to the variables A,B,C,... rather than overwriting
% variables in the caller. If there are fewer output variables than
% there are fields in S, the remaining fields are not extracted. Variables
% are assigned in the order given by fieldnames(S).

if nargin==0
    error('Input Arguments Required.')
elseif nargin==1            % Unpack Structure to Variables
    arg=varargin{1} ;
    if ~isstruct(arg)|length(arg)~=1
        error('Single Input Must be a Scalar Structure.')
    end
    names=fieldnames(arg);
    if nargout==0 % assign in caller
        for i=1:length(names)
            assignin('caller',names{i},getfield(arg,names{i}))
        end
    else % dump into variables in caller
        for i=1:nargout
            varargout{i}=getfield(arg,names{i});
        end
    end
else                        % Pack Variables into a Structure
    num=1;
    for i=1:nargin
        name=inputname(i) ;
        if isempty(name) % not a variable
            name=sprintf('ans%d',num);
            num=num+1;
        end
        eval(['y.' name '=varargin{i};'])
    end
    varargout{1}=y;
end
```

具有相同功能的C语言的MEX文件如下所示。比较M文件和等效的MEX文件可以让我

们更好地了解如何创建MEX文件。

```
/*
 * v2struct.c - Pack/Unpack Variables to/from a Scalar Structure.
 *
 * MATLAB usage: s=mv2struct(x,y,z,...) s.x=x, s.y=y, etc.
 * MATLAB usage: mv2struct(s) a=s.a, b=s.b, etc.
 * MATLAB usage: [x,y,z,...]=mv2struct(s) x=s.a, y=s.b,etc.
 *
 * MEX file to implement the function mmv2struct.m.
 *
 * Mastering MATLAB 6 C MEX Example 5
 */

 #include "mex.h"
 #include <string.h>

 void mexFunction( int nlhs,       mxArray *plhs[],
                   int nrhs,const  mxArray *prhs[] )
{
   int i , j, nfields ;
   char cmd[100], tmp[10];
   const char *ans = "ans";
   const char **fnames;
   char **name;

   /* Do some error checking */
   if (nrhs == 0)
     mexErrMsgTxt("Input Arguments Required.");

   else if (nrhs == 1) {   /* Unpack */
     if ( !mxIsStruct(prhs[0]))
       mexErrMsgTxt("Single Input Must be a Scalar Structure.");
     else if (mxGetNumberOfElements(prhs[0]) != 1)
       mexErrMsgTxt("Single Input Must be a Scalar Structure.");

   nfields = mxGetNumberOfFields(prhs[0]);
   if (nlhs == 0)    /* Assign fields in the caller workspace */
     for (i=0;i<nfields;i++) {
       strcpy(cmd,mxGetFieldNameByNumber(prhs[0],i));
       strcat(cmd,"=");
       strcat(cmd,mxGetName(prhs[0]));
       strcat(cmd,".");
       strcat(cmd.mxGetFieldNameByNumber(prhs[0],i));
       mexEvalString(cmd) ;
   }
else {           /* Assign fields to output variables */
```

```
        j=(nlhs<nfields)?nlhs:nfields;
        for (i=0;i<j;i++)
            p1hs[i]=mxDuplicateArray(mxGetFieldByNumber(prhs[0],0,i));
        }
    }
    else {            /* Pack */
        /* Create a list of fieldnames. */
        fnames=mxCalloc(nrhs,sizeof(*fnames));
        for (i=0;i<nrhs;i++) {
          name[i]=mxCalloc(BUFLEN,s1zeof(char));
          if (*mx.GetName(prhs[i]) != '\0')
            strcpy(name[i],mxGetName(prhs[i]));
          else {
            strcpy(name[i],ans);
            sprintf(tmp, "%d" ,i ) ;
            strcat(name[i],tmp);
          }
          fnames[i]=name[i];
        }

        /* Create the output structure and free the allocated memory. */
        p1 hs[0]=mxCreateStructMatrix(1,1,nrhs,fnames) ;
        mxFree(fnames);
        for (i=0;i<nrhs; i++)
          mxFree(name[i]);

        /* Stuff the inputs into the structure fields. */
        for (i=0;i<nrhs;i++)
          mxSetFieldByNumber(plhs[0],0,i,mxDuplicateArray(prhs[i]));
        }
    }
```

Windows PC **注意事项**

前面的例子也可以在Windows PC**上使用**Matlab**支持的**LCC**编译器进行编译。虽然**LCC**编译器提供的功能不多，但是它还是可以生成有效的**MEX**文件的。命令：**

```
>> mex myprog.c
```

将在当前目录下生成一个MEX**文件——**myprog.dll**。在**mex**命令行后加入**-v**参数，则可以打印编译器的设置，并显示编译器和连接状态。可以使用**mex –setup**命令为编译器选择选项文件。使用**-f**选项可以为编译指定不同的选项文件。在**$Matlab\bin\win32\mexopts**目录下保存着至少**8**个选项文件，以分别为不同版本的**Borland，Microsoft，Watcaom**编译器使用。在**Matlab**的文档中提供了有关配置编译器和调试**MEX**程序的详细内容。**

FORTRAN **注意事项**

FORTRAN**的**MEX**文件只能创建双精度数据和字符串，而**C**的**MEX**文件可以创建所有**

的 Matlab 数据类型。FORTRAN 的 MEX 文件只支持 mxCreateFull、mxCreateSparse 和 mxCreateString 这三种 mxArray 创建函数。因此，在 Matlab 文档中，FORTRAN 的 MEX 和 MX 函数是单独列出的。需要注意的是：FORTRAN 的源代码是不区分字母大小写的，因此 MXCREATEFULL、mxcreatfull 和 mxCreatFull 都将代表同一函数。在本章的 FORTRAN 的例子中，将继续使用与前面一致的大小写混用的写法，这样也使 FORTRAN 代码更易于阅读。

因为 FORTRAN 不支持新的数据类型，所以 Matlab 为每个输入和输出变量向 FORTRAN 程序传送一个特殊的 integer*4 类型的标识符，即一个指针。MEX 子程序可以使用这些指针从 Matlab 数组中获得那些使用 FORTRAN 的数据类型的数据。在所有平台上，都必须默认声明指针为 *interger4，但是在使用 64 位的 Alpha 处理器的平台上，必须要将指针声明为 interger*8。

某些 FORTRAN 编译器支持 %val 结构，该结构可以用于将值从函数（例如：p=mxGetPr()）获得的指针传送到子程序中。另外，可以使用 mxCopy…程序（例如 mxCopyPtrToReal8 和 mxCopyReal8ToPtr）从 mxArray 中提取数据，传送到子程序，然后将结果返回 mxArray。

在 FORTRAN 中，mexFunction 子程序的定义如下：

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

其中，nlhs 和 nrhs 中分别包含左侧和右侧参数的数量，plhs 和 prhs 指向参数本身的指针数组。

与 FORTRAN 等效的 C 语言的 MEX 文件 fact.c 如下：

```
C-----------------------------------------------------------
C       fact.f - returns the factorial of a nonnegative integer.
C
C       MATLAB usage:  p=fact(n)
C
C       Mastering MATLAB 6 FORTRAN MEX Example 1

        subroutine mexFunction(nlhs, plhs, nrhs, prhs)
C-----------------------------------------------------------
C       These are pointers: integer*4 (Integer*8 on Alphas)
C
        integer plhs(*),prhs(*)
        integer mxGetPr, mxCreateFull
        integer y_pr
C-----------------------------------------------------------
        integer nlhs.nrhs
        integer i
        real*8 x, y, mxGetScalar
C-----------------------------------------------------------
        x = mxGetScalar(prhs(1))
        plhs(1) = mxCreateFull(1, 1, 0)
        y_pr = mxGetPr(plhs(1))
```

```
C
        y = 1.0
        do 10 i=x,l,-l
            y = y * i
  10    continue
C
        call mxCopyReal8ToPtr(y, y_pr, 1)
        return
        end
```

请注意，和Matlab类似，在FORTRAN中也使用1基数组和循环索引，而不是像C程序一样使用0基索引。因此，在FORTRAN程序中，指向右侧（输入）第一个参数的指针是prhs(1)，而在C程序中则是prhs[0]。

34.3 从C或FORTRAN调用Matlab

我们既可以在Matlab中调用C或FORTRAN程序，也可以在一些大型的C或FORTRAN程序中调用Matlab来执行一些运算。在Matlab中，这被称为Matlab引擎。

什么是 Matlab 引擎

Matlab引擎包含一个通信库和少量连接例程集。通过连接例程集可以将Matlab作为服务器过程调用，而无需连接所有的Matlab。Matlab引擎允许你启动一个Matlab过程；向Matlab传送数据；执行Matlab命令；如果需要，可以获得普通“命令”窗口的输出；将数据传回你的程序；关闭Matlab过程。所有这些都可以在C或FORTRAN里执行。

Matlab 引擎如何工作

Matlab引擎过程所运行的背景与当前运行的所有交互的Matlab会话相独立。它不会干扰任何用户运行的Matlab。在引擎过程启动后，会创建一个新的Matlab实例。指定主机上的所有要求访问Matlab引擎过程的程序将共享此引擎过程。在C程序中保留有一个独占的引擎过程，而在FORTRAN程序中，引擎过程通常都是共享的。

带有eng前缀的函数都是Matlab引擎函数。C和FORTRAN程序中可用的Matlab引擎函数如下表所列：

函数	C	F	功能
engOpen	C	F	启动或共享Matlab引擎的实例
engOpenSingleUse	C		启动一个独占（非共享）的Matlab引擎会话
engPutArray	C		向Matlab引擎发送一个Matlab数组（mxArray）
engPutMatrix		F	向Matlab引擎发送一个Matlab数组（mxArray）
engOutputBuffer	C	F	创建一个存储Matlab文本输出的缓冲区
engEvalString	C	F	在Matlab引擎中执行一条Matlab命令
engGetArray	C		从Matlab引擎中获得一个Matlab数组（mxArray）

续表

函数	C	F	功能
engGetMatrix		F	从Matlab引擎中获得一个Matlab数组（mxArray）
engClose	C	F	关闭Matlab引擎

引擎程序结构

使用Matlab引擎的第一个步骤是使用engOpen或engOpenSingleUse打开一个引擎会话。然后使用适合的MX函数（例如mxCreatDoubleMatrix）创建所需的所有mxArray，并使用mxSetName将mxArray和Matlab变量名关联起来。使用engPutArray（在FORTRAN中使用engPutMatrix）将mxArray移植到Matlab环境中。如果需要，那么可以使用engOutputBuffer创建一个捕捉“命令”窗口输出的缓冲区，然后使用engEvalString来执行Matlab命令。使用engGetArray（或engGetMatrix）可以从Matlab环境中重新获得任何一个输出mxArray，并继续在C或FORTRAN程序中处理。在Matlab环境中完成这些操作后，使用engClose来结束程序，并使用mxDestroyArray来释放分配给mxArray的内存。

引擎程序描述

engOpen函数在启动一个Matlab引擎后可以返回一个惟一的“引擎ID”。引擎ID可以被用于对此引擎进行寻址。如果发生了错误，那么函数将返回NULL。如果engOpen的参数是NULL（'\0'），那么将使用matlab命令在本地计算机上启动引擎。在Unix或Linux计算机上，参数可以使用主机名。在这种情况下，将使用rsh命令在远程主机上启动引擎。还将设置DISPLAY环境变量，这样所有由引擎生成的图像（例如plot命令的结果）都会在本地计算机上显示，而不是在运行引擎程序的计算机上显示。还要设置适当的权限，以允许进行远程执行，并允许远程计算机在本地计算机上进行显示。如果engOpen的参数不是主机名，而是其他任何名字（例如包含空格、制表符或非数字非字母的字符的字符串），那么字符串将被逐字执行以启动引擎过程。可以利用这个功能来自定义引擎会话。例如：使用ssh将与远程主机的通信进行加密。因为在Windows PC平台上不支持远程执行，所以engOpen的参数必须是NULL。安装在本地PC上的Matlab应用程序将启动以响应任何引擎的请求。

必须要在引擎环境中创建Matlab变量。创建Matlab变量有两种方法。一种方法是使用engEvalString函数来得到Matlab命令的值，例如Matlab命令：y=-2*pi:pi/25:2*pi；然后在Matlab工作区中创建变量y并将值赋给y。另一种方法是在程序中创建一个Matlab数组，然后将所得到的mxArray传递给Matlab引擎。例如：在C变量dataset中给定一个1乘10的双精度数组。

```
Engine *mat
mxArray *mydata = NULL;

mat = engOpen("\0");
mydat = mxCreateDoubleMatrix(1, 10, mxREAL);
mxSetName(mydata, "newdata");
memcpy((void *)mxGetPr(mydata),(void *)dataset,sizeof(dataset));
engPutArray(mat,mydata);
```

```
engEvalString("sqdata=newdata.^2;");
```

上面的代码将会在本地计算机上打开一个到Matlab的连接，创建一个名为mydata的mxArray，将Matlab变量名newdata与mydata数组相关联，填充数组，将mydata数组发送给引擎，将数组的元素平方，将输出赋给Matlab变量sqdata。需要使用mxSetName函数来为mxArray分配一个Matlab变量名。这是识别Matlab工作区中的数组的惟一方法。在本例中，C语言和Matlab中所使用的名字是不同的，这样可以更清晰地演示它们各自的使用方法。而在一般情况下，我们应该采用相同的名字，以避免混淆。

一旦Matlab执行了所要做的运算，就必需要将结果返回到你的程序中。同样也有两种方法来传递结果。第一种方法是使用engCreateTextBuffer函数创建一个文本缓冲区来获得那些通常被丢弃的Matlab文本，然后使用engEvalString函数生成输出文本，最后对字符串缓冲区进行分析。例如：

```
char buf[256];
engOutputBuffer(mat,buf,256);
engEvalString("disp(sum(sqdata)) ");
```

将获得disp(sum(sqdata))命令在Matlab“命令”窗口所显示的内容，并将显示的内容输出到字符串缓冲区buf中。对缓冲区进行分析可以得到所需要的数据。

```
double x;
X=atof(buf+2);
```

每次调用engEvalString都会重新替换Matlab输出字符串缓冲区中的数据。请注意：输出字符串缓冲区中的前两个字符通常是Matlab的提示符“>>”，可以调用atof函数来删除这些字符。

如果得到的结果非常庞大，那么可以在Matlab工作区中检索一或多个数组。例如：

```
mxArray *sqdata = NULL;
double *sptr;

sqdata = engGetArray(mat, "sqdata");
sptr = mxGetPr(sqdata);
```

将使你的程序获得mxArray sqdata。在上面的例子中使用了mxGetPr函数来得到一个指向mxArray的实数部分的C指针。其他MX函数也可以访问sqdata数组。

在完成运算后，一定要释放mxArray所占用的内存，并且要在不再需要Matlab引擎时关闭Matlab引擎。

```
mxDestroyArray(mydata);
mxDestroyArray(sqdata);
engClose(mat);
```

编译和运行引擎程序

如果程序中要使用Matlab引擎的访问函数，那么除了那些必需的头文件外，还必须要包含engine.h头文件，以支持Matlab引擎函数。engine.h头文件中还包含matrix.h以支持MX

函数。编译时所使用的mex命令，除了使用常用的编译器选项外，还需要使用engopts.sh选项文件。例如：如果Matlab应用程序安装在/usr/local/matlab目录下，那么用于编译C引擎程序的mex命令应为：

```
mex –f /usr/local/matlab/bin/engopts.sh myprog.c
```

此命令会在当前目录下创建一个名为myprog的可执行程序。该程序可以通过在操作系统的命令提示符后输入命令或在Windows PC上双击来运行。如果标准的选项文件无法满足你的要求，那么可以制作一个engopts.sh文件的副本，然后对它进行修改。

在已编译的引擎程序运行的时候，会加载一个或多个Matlab共享库。必须要告诉操作系统区在何处寻找这些库。当操作系统在标准的系统目录下无法找到所有所需的共享库时，操作系统会在LD_LIBRARY_PATH环境变量（或你所使用的其他平台上等效的变量）中存储的附加目录下进行搜索。你需要在该环境变量中加入系统指定的Matlab共享库的目录。如果使用的是Solaris操作系统，Matlab安装在/opt/matlab目录下，并使用Bourne的命令解释程序（sh，bash，ksh），那么命令应写作：

```
LD_LIBRARY_PATH=/opt/matlab/extern/lib/sol2:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

如果是在Linux平台上使用C命令解释程序（csh，tcsh）和C的libc2库，Matlab安装在/usr/local/matlab/目录下，那么命令应写作：

```
setenv LD_LIBRARY_PATH /usr/local/matlab/extern/lib/glnx86:
$LD_LIBRARY_PATH
```

指定平台的库目录（sol2，glnx86，lnx86，sgi，sgi64，win32，hp700，alpha，ibm_rs）都存储在$Matlab/extern/lib目录下。有些平台使用不同的环境变量来存储附加共享库的位置。SGI64平台使用LD_LIBRARY64_PATH，而非LD_LIBRARY_PATH。HP700平台使用SHLIB_PATH，IBM的RS/600平台使用LIBPATH。设置环境变量的最简单的方法是向命令解释程序的启动脚本（.profile，.cshrc，.bashrc，.tcshrc等等）中加入适当的命令。

引擎示例

前面，我们讲解了Matlab引擎程序的各个独立部分。现在，我们该把这些部分合并在一起了。下面给出的例子建立在前面各章节的基础上，它将创建一个使用Matlab引擎作为后台的计算引擎的完整的C程序。该C程序将创建数据数组，将数组发送给Matlab，计算每个元素的平方和，然后将结果返回C程序并打印。为了使程序更加容易理解，我们还在程序中加入了注释。

```
/*
 * sos.c - Calculate the sum of the squares of the elements of a vector.
 *
 * Mastering MATLAB 6 C Engine Example 1
 *
 */
```

```
#include <stdio.h>
#include <string.h>
#include "engine.h"
#define  BUFSIZE 256

int main()
{
    Engine *mat;
    mxArray *mydata = NULL, *sqdata = NULL;
    int i,j;
    double x, *myptr, *sptr;
    char buf[BUFSIZE];
    double dataset[10] = { 0.0, 1.0, 2.0, 3.0, 4.0,
                           5.0, 6.0, 7.0, 8.0, 9.0 };

    /* Start the MATLAB Engine on the local computer. */
    if (!(mat = engOpen("\0"))) {
        fprintf(stderr, "\nCannot open connection to MATLAB!\n");
        return EXIT_FAILURE;
    }

    /* Create an mxArray and get a C pointer to the mxArray. */
    mydata = mxCreateDoubleMatrix( 1, 10, mxREAL);
    myptr = mxGetPr(mydata);
    /* Associate a MATLAB variable name with the mxArray. */
    mxSetName(mydata, "newdata");

    /* Copy the dataset array to the new mxArray. */
    memcpy((void *)myptr, (void *)dataset, sizeof(dataset));

    /* Pass the mxArray to the Engine and square the elements. */
    engPutArray(mat, mydata);
    engEvalString(mat, "sqdata = newdata.^2");

    /* Create an output buffer to capture MATLAB text output. */
    engOutputBuffer(mat, buf, BUFSIZE);

    /* Calculate the sum of the squares and save the result in x. */
    engEva1String(mat,"disp(sum(sqdata))");
    x=atof(buf+2);

    /* Retrieve the array of squares from the Engine, */
    if ((sqdata = engGetArray(mat,"sqdata")) == NULL) {
        fprintf(stderr, "Cannot retrieve sqdata!\n\n");
        return EXIT_FAILURE;
    }
```

```
    /* and get a C pointer to the mxArray. */

    sptr = mxGetPr(sqdata);

    /* Print the results to stdout. */
    printf("\nThe inputs are:\n");
    for (i=0;i<10;i++)
      printf("%6.1f ",myptr[i]);
    printf("\n\n The squares are:\n");
    for (i=0;i<10; i++)
      printf("%6.1f ",sptr[i]);
    printf("\n\nThe sum of the squares is %6.1f \n\n",x);

    /* Free the mxArray memory and quit MATLAB. */
    mxDestroyArray(mydata);
    mxDestroyArray(sqdata);
    engclose(mat);

    return EXIT_SUCCESS;
}
```

Windows PC注意事项

上面给出的例子还可以在Windows PC**平台上使用**LCC**编译器进行编译。使用命令：**

```
>> mex –f c:\matlab6\bin\win32\mexopts\lccengmatopts.bat sos.c
```

将在当前目录下生成可执行文件sos.exe**。如果不使用**LCC**编译器，那么需要在命令行中指定适当的选项文件。在**$Matlab\bin\win32\mexopts**目录下保存着至少**8**个选项文件，分别用于不同版本**Borland、Microsoft、Watcom**编译器。如果编译发生错误，可以使用**mex**命令的详细说明选项（**-v**）来查看编译器的设置和编译过程的各个步骤。**

Matlab**引擎程序要与**$Matlab\bin\win32**目录下的**DLL**库连接。在**Matlab**安装过程中，**Matlab**会将该目录添加到默认路径中以使**Windows**可以找到这些库文件。当在命令行中执行**sos.exe**文件，或双击鼠标左键执行**sos.exe**文件时，会（最小化）运行一个**Matlab**会话来执行计算。该会话不会干扰正在处于打开状态的其他交互的**Matlab**会话。**

有关如何配置编程软件来编译MEX**和引擎程序，和使用软件开发环境来调试程序的详细内容，请参考**Matlab**的文档。**

FORTRAN 注意事项

用FORTRAN**语言编写的**Matlab**引擎程序与用**C**语言编写的引擎程序相比只有很少的改动。大多数的引擎函数都只返回用于检测错误的状态值。平方和将作为字符串数组打印，这样做可以避免从文本缓冲区中提取数据进行打印。**

```
C
C   sos.c - Calculate the sum of the squares of the elements of a vector.
C
```

```
C   Mastering MATLAB 6 FORTRAN Engine Example 1
C
C=======================================================================
C   program main
C-----------------------------------------------------------------------
C   Pointers
C
    integer engOpen, engGetMatrix, mxCreateFull, mxGetPr
    integer mat, mydata, sqdata
C-----------------------------------------------------------------------
C   Other variable declarations
C
    double precision dataset(10), sqrs(10)
    integer engPutMatrix , engEvalString, engClose, engOutputBuffer
    integer temp, status
    character*256 buf
    data dataset / 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0 /
C-----------------------------------------------------------------------
C   Start the MATLAB Engine on the local computer.
C
    mat = eng0pen('matlab ')
    if (mat .eq. 0) then
        write(6,*) 'Cannot open connection to MATLAB!'
        stop
    endif
C
C   Create an mxArray, associate a MATLAB variable name with the
C    mxArray, and copy the data into the array.
C
    mydata = mxCreateFull(1, 10, 0)
    ca11 mxSetName(mydata, 'newdata')
    ca11 mxCopyRea18ToPtr(dataset, mxGetPr(mydata), 10)
C
C   Pass the variable mydata into the MATLAB workspace.
C
    status = engPutMatrix(mat, mydata)
    if (status .ne. 0) then
        write(6,*) 'Cannot pass mydata to the Engine!'
        stop
    endif
C
C   Square the elements of the array.
C
    if (engEvalString(mat, 'sqdata = newdata.^2;') .ne. 0) then
        write (6, *) 'engEvalString failed'
        stop
    endif
```

```
C
C   Create an output buffer to capture MATLAB text output.
C
    if (engOutputBuffer(mat, but) .ne. 0) then
        write(6,*) 'engEvalString failed'
        stop
    endif
C
C   Calculate the sum of the squares and capture the result.
C
    if (engEvalString(mat, 'disp(sum(sqdata))') .ne. 0) then
        write (6,*) 'engEvalString failed'
        stop
    endif
C
C   Retrieve the mxArray of squares from the Engine,
C   copy the data into an array of doubles, and print.
C
    sqdata = engGetMatrix(mat, 'sqdata')
    call mxCopyPtrToReal8(mxGetPr(sqdata), sqrs, 10)
C
 20 format(' ', G8.3, G8.3, G8.3, G8.3, G8.3, G8.3,
   & G8.3, G8.3, G8.3, G8.3)
     print *, 'The inputs are:'
     print 20, dataset
     print *, 'The squares are:'
     print 20, sqrs
     print *,  'The sum of the squares is ', buf(3:10)
C
C   Free the mxArray memory and quit MATLAB.
C
    call mxFreeMatrix(mydata)
    call mxFreeMatrix(sqdata)
    status = engClose(mat)
C
    if (status .ne. 0) then
        write (6,*) 'engClose failed'
        stop
    endif
C
    stop
    end
```

请注意：如果你的FORTRAN**编译器使用了共享库，那么你必须要在执行已编译的引擎程序前，先将**Matlab**共享库的目录添加到**LD_LIBRARY_PATH**（或等效的）环境变量中。如果使用**Windows PC**，那么**Matlab**会在安装过程中自动将**$Matlab\bin\win32**目录添加到默认路径中，以便**Windows**可以找到**Matlab**共享库（**DLL**）。**

34.4 与MAT文件交换数据

Matlab有几种方式与其他程序交换数据。大多数方法都要创建一个作为交换媒介的数据文件。选择何种方法取决于输入或输出的数据的数量和格式。在本节中，我们将介绍如何在C和FORTRAN中读写Matlab标准的MAT文件。

MAT 文件

MAT文件是独立于平台的，这是因为MAT文件可以识别平台差别（例如字节次序），从而Matlab在从MAT文件装载数据时会自动将数据格式转化为适当形式。通过提供在程序中读写MAT文件所使用的头文件和库，Matlab可以向C和FORTRAN程序提供平台独立的特性以及与Matlab方便的数据交换特性。

MAT 函数

带有mat前缀的Matlab函数，它是对MAT文件进行操作的。可以在C或FORTRAN中使用的Matlab的MAT函数如下表所列。

MAT函数	C	F	作用
matOpen	C	F	打开一个MAT文件
matClose	C	F	关闭一个MAT文件
matGetDir	C	F	从MAT文件中获得Matlab数组的列表
matGetFp	C		获得一个指向MAT文件的ANSI C文件指针
matGetArray	C		从MAT文件中读出一个Matlab数组
matGetMatrix		F	从MAT文件中读出一个Matlab数组
matGetNextArray	C		从MAT文件中读出下一个Matlab数组
matGetNextMatrix		F	从MAT文件中读出下一个Matlab数组
matGetArrayHeader	C		从MAT文件中装载Matlab数组头
matGetNextArrayHeader	C		从MAT文件中装载下一个Matlab数组头
matGetString		F	从MAT文件中读取Matlab字符串
matPutArray	C		向MAT文件写入Matlab数组
matPutMatrix		F	向MAT文件写入Matlab数组
matPutArrayAsGlobal	C		向MAT文件写入全局Matlab数组
matPutString		F	向MAT文件写入Matlab字符串
matDeletArray	C		从MAT文件中删除Matlab数组
matDeleteMatrix		F	从MAT文件中删除Matlab数组

matGetArrayHeader和matGetNextArrayHeader函数可以创建mxArray。所创建的mxArray可以包含除了实际数据外的mxArray结构的所有内容。matGetDir函数可以创建一个变量名称列表并返回MAT文件中Matlab变量的数量。matGetArray和matGetMatrix函数可以使用这些变量名来访问变量的内容，同时matGetNext…函数可以按顺序访问变量。

MAT 程序结构

读或写MAT文件的C或FORTRAN程序使用MX函数创建Matlab数据数组（mxArray）的方法和在MEX文件或引擎程序中使用MX函数的方法相同。在创建MAT文件时，首先使用matOpen函数打开MAT文件，然后为每个Matlab变量创建，命名并填充mxArray，然后使用适当的matPut…函数将MAT文件写入mxArray，最后使用matClose函数关闭MAT文件。在从MAT文件读取数据时，首先使用matOpen函数打开MAT文件，如果需要变量列表，则使用matGetDir函数获得一个列表，使用matGetArray或matGetNextArray（或FORTRAN语言中的matGetMatrix、matGetNextMatrix、matGetString）函数创建mxArray，最后使用matClose函数关闭MAT文件。

编译和运行 MAT 程序

带有Matlab的MAT文件访问函数的程序必须除包含任何其他必需的头文件外，还要包含mat.h头文件，以支持Matlab的MAT文件函数。mat.h头文件还包含支持MX函数的matrix.h。编译时使用mex命令，该命令需要使用matopts.sh选项文件，同时也可以使用其他需要的选项文件。例如，如果Matlab应用程序安装在/usr/local/matlab目录下，那么编译C的MAT程序的mex命令可以为：

```
>> mex –f /usr/local/matlab/bin/matopts.sh myprog.c
```

该命令将在当前目录下创建一个名为myprog的可执行程序。所生成的可执行程序可以在操作系统的提示符下运行。在引擎一节所讨论的有关UNIX和PC平台的共享库的内容在这里也同样适用。由mex命令创建的可执行程序所使用的共享库必须在程序运行时可以由操作系统找到。

Windows PC版本使用和引擎程序相同的选项文件。使用LCC编译器编译同一MAT文件——myprog.c，可以使用下面的命令：

```
>> mex –f c:\matlab6\bin\win32\mexopts\lccengmatopts.bat myprog.c
```

编译后将在当前目录下生成可执行文件myprog.exe，该文件可以在命令提示符下运行或者双击文件运行。如果不使用LCC编译器，那么需要在命令行中指定适当的选项文件。

MAT 程序示例 1：writemat

下面给出的C程序将创建一个包含一个字符串和一个双精度数组的MAT文件。

```
/*
 * writemat.c - Create a binary MAT fi1e.
 *
 * Mastering MATLAB 6 C MAT-file Example 1
 *
 */

#include "mat.h"
```

```
int makemat(const char *filename,
            double *data , int m, int n,
            char *mmstr)
{
    MATFile *mfile;
    mxArray *mdata, *mstr;

    /* Open the MAT file for writing. */
    mfile == matOpen(filename, "w");
    if (mfile == NULL) {
      printf("Cannot open %s for writing.\n", filename);

      return(EXIT_FAILURE);
  }

  /* Create the mxArray to hold the numeric data.    */
  /* Note that the array dimensions are reversed.    */
  /* C uses row order while MATLAB uses column order. */
  /* The data array will be transposed in MATLAB.    */
  mdata = mxCreateDoubleMatrix(n,m,mxREAL);
  mxSetName(mdata, "mydata");

  /* Copy the data to the mxArray. Note that mxGetData is */
  /* similar to mxGetPr in that it returns a void pointer */
  /* while mxGetPr returns a pointer to a double.          */
  memcpy((void *) (mxGetData (mdata )), (void *)data,
                    m*n*sizeof(double));

  /* Create the string array and set the variable name. */
  mstr = mxCreateString(mmstr);
  mxSetName(mstr, "mystr");

  /* Write the mxArrays to the MAT file. */
  matPutArray(mfile, mdata);
  matPutArray(mfile, mstr);

  /* Free the mxArray memory. */
  mxDestroyArray(mdata);
  mxDestroyArray(mstr);

  /* Close the MAT file. */
  if (matClose(mfile) != 0) {
      printf("Cannot close %s.\n",filename);
      return(EXIT_FAILURE);
  }

  return(EXIT_SUCCESS);
```

```
}

int main( )
{
    int status;
    char *mmstr = "Mastering MATLAB Rocks!";
    double data[3][4] = {{  1.0,   2.0,  3.0,   4.0 },
                         {  5.5,   6.6,  7.7,   8.8 },
                         { -4.0, -3.0, -2.0, -1.0 }};
    status = makemat("mmtest.mat", *data, 3, 4, mmstr);
    return(status);
}
```

该程序可以在Linux**和**Windows PC**平台上编译，它们的编译结果是一样的。所得到的**MAT**文件随后将被加载到**Matlab**会话中，过程如下所示：**

```
  >> clear all
  >> load mmtest
 >> whos
    Name            Size            Bytes  Class

    mydata          4×3                96  double array
    mystr           1×23               46  char array
Grand total is 35 elements using 142 bytes

 >> mydata
mydata =
        1.0000      5.5000      -4.0000
        2.0000      6.6000      -3.0000
        3.0000      7.7000      -2.0000
        4.0000      8.8000      -1.0000

>> mystr
mystr =
Mastering MATLAB Rocks!
```

请注意，在装载到 Matlab**中时，对数字数组进行了转置。这是因为**C**语言和**Matlab**存储数组的方式不同：它们分别是按行和按列存储的。因为**FORTRAN**和**Matlab**都是按列存储数组的，所以使用**FORTRAN**数组时不需要对矩阵进行转置。下面给出了同一程序的**FORTRAN**版本。**

```
C
C   writemat.f - Create a binary MAT file.
C
C   Mastering MATLAB 6 FORTRAN MAT-file Example 1
C
C
    program writemat
```

```
C-------------------------------------------------------------
C   Pointers.
C
    integer matOpen, mxCreateFull, mxCreateString
    integer matGetMatrix, mxGetPr
    integer mfile, mdata, mstr
C-------------------------------------------------------------
C   Other variables
C
    integer status, matClose
    double precision dat(12)
    data dat /  1.0, 5.5, -4.0,
   &            2.0, 6.6, -3.0,
   &            3.0, 7.7, -2.0,
   &            4.0, 8.8, -1.0 /
C
C   Open MAT-file for writing.
C
    mfile = matOpen('mmtest.mat', 'w')
    if (mfile .eq. 0) then
      write(6,*) 'Can''t open ''mmtest.mat'' for writing.'
      stop
    end if
C
C   Create the mxArray to hold the numeric data.
C
    mdata = mxCreateFull(4,3,0)
    call mxSetName(mdata, 'mydata')
C
C   Copy the data to the mxArray.
C
    call mxCopyReal8ToPtr(dat, mxGetPr(mdata), 12)
C
C   Create the string array and set the variable name.
C
    mstr = mxCreateString('Mastering MATLAB Rocks!')
    call mxSetName(mstr, 'mystr')
C
C   Write the mxArrays to the MAT-file.
C
    call matPutMatrix(mfile, mdata)
    call matPutMatrix(mfile, mstr)
C
C   Free the mxArray memory.
C
    call mxFreeMatrix(mdata)
    call mxFreeMatrix(mstr)
```

```
C
C   Close the MAT file.
C
    status = matClose(mp)
    if (status .ne. 0) then
      write(6,*) 'Cannot close mmtest.mat'
      stop
    end if
C
    stop
    end
```

MAT 程序示例 2：whomat

第二个例子将读入一个MAT文件，并检验文件的内容。然后以一个Matlab的who和whos命令相似的格式打印变量列表。

```
/*
 * whomat.c - Examine a binary MAT-file and print a list
 * of the contents(like "who" or "whos").
 *
 * Mastering MATLAB 6 C MAT-file Example 2
 *
 */

    #include "mat.h"
    #include <string.h>

    int whomat(const char *filename)
    {
        MATFile *mfile;
        mxArray *marray;
        char **dir;
        char siz[25], buf[10];
        int i, j, k, num, nel, elsize, ndim, eltot, btot;
        const int *dims;

        /* Open the MAT-file for reading. */
        mfile = matOpen(filename, "r");
        if (mfile == NULL) {
          printf("Cannot open %S for reading.\n", filename);
          return(EXIT_FAILURE);
        }

        /* Get the directory list and print in "who" format. */
        dir = matGetDir(mfile, &num);
        if (dir == NULL) {
          printf("Error reading the directory of %s.\n", filename);
```

```
            return(EXIT_FAILURE);
          } else {
            printf(" \n") ;
            printf("Variables in %S are:\n\n", filename);
            for (i=0; i<num; i++) {
              printf("%-10s",dir[i]);
              if (i>0 && i%4==0) printf("\n");
          }
      }

      /* Examine each variable and print a "whos" list. */
      eltot=btot=0;
      printf("\n\n  Name        Size        Bytes  Class\n\n");
      for (i=0; i<num; i++) {
        marray=matGetArray(mfile, dir[i]);
        if (marray == NULL) {
          printf("Cannot read file %s.\n\n", filename);
          return(EXIT_FAILURE);
        }

      /* If marray is a cell array or structure array, then  */
      /* mxGetElementSize returns the size of a pointer; not */
      /* the size of all the elements in each cell or field. */
      /* To get the correct number of bytes would require    */
      /* traversing the array and summing leaf element sizes.*/
      /* Java arrays return 0×0 array dimensions and 0 size. */
      elsize=mxGetElementSize(marray);
      btot=btot+(nel*elsize);
      nel=mxGetNumberOfElements(marray);
      eltot=eltot+nel;
      ndim=mxGetNumberOfDimensions(marray);
      dims=mxGetDimensions(marray);
      siz[0]='\0';
      for (j=0; j<ndim; j++) {
          sprintf(buf,"%d",dims[j]);
          strcat(siz,buf);
          if (j<(ndim-1))
              strcat(siz,"x");
      }
      printf(" %-12s %-12s %5d %s array\n", mxGetName(marray),
              siz,nel*elsize,mxGetClassName(marray));
      mxDestroyArray(marray);
  }
  printf("\nGrand total is %d elements using %d bytes\n\n",
              eltot,btot);

  /* Release the memory allocated for the directory. */
```

```
  mxFree(dir);

/* Close the MAT file. */
if (matClose(mfile) != 0) {
    printf("Cannot close %s.\n",filename);
    return(EXIT_FAILURE);
  }
  return(EXIT_SUCCESS);
}

int main(int argc, char **argv)
{
    int status;

    if (argc > 1)
        status = whomat(argv[1]);
    else{
        status = EXIT_FAILURE;
        printf("Usage: whomat <matfile>");
    }
    return(status);
}
```

此程序的输出与在工作区中对变量使用Matlab**的**who**和**whos**命令的输出非常相似。输出将报告单元数组、结构和**Java**数组的正确结果。如果**marray**是单元数组或结构数组，那么**mxGetElementSize(marray)**将返回指针的大小，而不是每个单元或字段的所有元素的大小。为了能够得到正确的字节数，需要遍历数组并计算端点元素的尺寸和。如果**marray**是一个**Java**数组，那么**mxGetElementSize(marray)**将返回**0**，**mxGetDimensions(marray)**将返回指向**mxGetNumberOfDimensions(marray)**长度的**0**向量的指针。添加对这些类型的数据的支持，将作为练习留给读者来完成。

34.5　小　结

本章介绍的Matlab**编程接口为扩大**Matlab**的使用范围提供了工具。所创建的**MEX**文件可以加速循环，这在以前通过添加多行接口代码编写的**C**或**FORTRAN**子例程无法向量化创建的**MEX**文件可以用来加速无法向量化的循环，或通过添加几行接口代码利用以前编写的**C**函数或**Fortran**子程序或得到的优点。编译好的**MEX**文件可以在**Matlab**中与调用**M**文件函数一样被调用。**Matlab**可以作为后端计算引擎使用，这样你就可以在程序中利用**Matlab**的计算速度、效率和图形化的函数。你可以在自己的程序中读写**Matlab**的**MAT**文件，并使用简单的**load**和**save**命令将变量和数据集传入或传出**Matlab**。所有这些异常分支指令（**hook**）可以在**Matlab**环境内添加工具集，你可以使用它们来解决难题或扩展已有的程序的功能。有关**Matlab API**的更多扩展文档，包括**MX**函数的完整列表和**MAT**文件的内部结构的详细信息，可以在**Matlab**文档中找到。