

第12章 MATLAB程序设计

MATLAB有一些命令可以来控制MATLAB语句的执行，如条件语句、循环语句和支持用户交互的命令，本章将介绍这些命令。MATLAB是一种高级的程序设计语言，能帮助用户解决矩阵问题或其他问题。那些熟悉其他编程语言的用户，如熟悉Pascal、C++、FORTRAN等，对理解本章内容有一定的优势。但是确信这部分内容能够让所有的读者理解和掌握。

12.1 条件控制语句

MATLAB中由if语句做出判断。if语句的基本格式如下：

```
if logical expression
    statements
end
```

注意，在if和logical expression(逻辑表达式)之间要有一个空格。statement(程序语句)可以是一个命令，也可以是由逗号、分号隔开的若干命令或者是‘returns’。只有当逻辑表达式为true(真)时，才能执行这些命令。逻辑表达式可以是一个标量、一个向量或者一个矩阵。如果逻辑表达式的所有元素为非零值，它才为true(真)。

if语句也可以写成一行。

```
if logical expression, statements, end
```

当然，通常前一种形式使得MATLAB程序更加结构化和易读。

例12.1

假设定义 $m \times n$ 的矩阵A。下面的语句是判断矩阵A的第1列元素是否为0，若全为0，则从矩阵A中删除第1列：

```
if A(:,1) == 0
    A = A(1:m,2:n)
end
```

或者写成一行：

```
if A(:,1) == 0, A = A(1:m,2:n), end
```

if语句可以与elseif或else组合起来用于更复杂的上下文语句中。可能有如下的结构存在：

```
if logical expression
    statements 1
else
    statements 2
end
```

如果逻辑表达式为true，则执行statements 1中的命令语句；如果为false则执行statements 2中的语句。

考虑下面的if语句：

```
if logical expression 1
    statements 1
elseif logical expression 2
    statements 2
end
```

当 *logical expression 1* 为 true 时，执行 *statements 1* 中的命令；如果 *logical expression 1* 为 false 并且 *logical expression 2* 为 true 时，执行 *statements 2*。

注意，elseif 必须写成一个单词，如果分开写成 else if，将会被解释成不同的意思。命令 elseif 不像 else if 一样需要一个额外的 end。

另外 if 语句可以被嵌套成下面的形式：

```
if logical expression 1
    statements 1
elseif logical expression 2
    statements 2
else
    statements 3
end
```

更复杂的情况如下：

```
if logical expression 1
    statements 1
    if logical expression 2
        statements 2
    else
        statements 3
    end
else
    statements 4
end
```

例12.2

(a) 如果 A 为非奇异矩阵，就能解方程 $Ax=b$ ；否则要取决于扩展矩阵 $(A \ b)$ 的梯形形式行的个数。提示：如果一个矩阵是方阵或为满秩的，则它为非奇异矩阵。

% 给出矩阵 A 和方程右边 b 。

```
s = size(A)

if (s(1) == s(2)) & (rank(A) == s(1))
    x = A\b
else
    rref([A b])
end
```

(b) 如果矩阵 A 的行列式为 0，则计算特征值为 0 的个数：

```
if det(A) == 0
    length(find(eig(A) == 0))
end
```

另一种条件语句是switch-case语句，如下：

```
switch logical expression (scalar or string)
case value 1
    expression 1
case value 2
    expression 2
...
otherwise
    expression
end
```

*logical expression*经过计算给出一个标量或字符串作为结果。将这个结果与*value 1, value 2, ...*进行比较，如果它们匹配，则执行相应的*case*下的语句*expression*。如果没有匹配的，则执行*otherwise*下的语句。

如果*expression*的结果是一个标量，将通过检查：*expression == value*来决定执行的表达式。如果表达式的结果是一个字符串，那么用 `strcmp(expression, value)`来检查。测试结果为真，则执行相应的表达式，而其他*case*语句中的表达式将不会被执行。

通过将不同的值放入细胞矩阵，就能用*case*语句与不同的值进行比较；见例 12.3。

例12.3

检测掷一次骰子所得的点数是单数还是双数：

```
function dicetest(result)

switch result
case {1,3,5}
    disp('odd number of eyes')
case {2,4,6}
    disp('even number of eyes')
otherwise
    disp('What kind of dice do you have?')
```

运行这个函数可以得到如下结果：

```
dicetest(1)

odd number of eyes

dicetest(4)

even number of eyes

dicetest(7)

What kind of dice do you have?
```

如果表达式出错，可以使用try/catch组合，其形式如下：

```
try
    expression 1
catch
    expression 2
end
```

MATLAB开始执行 *expression 1*，但如果有错误，错误信息将被存储在 `lasterr` 中，并且执行 *expression 2*。

12.2 循环语句

MATLAB有两个命令 `for` 和 `while` 能反复执行语句。在逻辑控制下，这些命令能灵活地一次或多次执行语句。

命令 `for` 与大多数的程序设计语言中的 `do` 或 `for` 命令一样。这个命令就是反复执行一条语句或一组语句，而执行的次数已预先定义好。以 `end` 结束这组语句。

`for` 循环通常的语法为：

```
for variable = expression
    statements
end
```

象 `if` 语句一样，`for` 语句也能写在一行上：

```
for variable = expression, statements, end
```

在 `for` 和 `variable` 之间需要有一个空格。这里的 `variable` 是循环变量名。在表达式中给出循环的初始值、步长和终值。这个步长可为负数或单位值。如果为单位值，循环变量每次迭代将增加1。通常我们用冒号来定义 *expression*，例如 `i:j:k` 或 `i:j`，参见4.3节。

表达式中的列值被一个一个地存放在循环变量中。因此，可以用一个矩阵来代替表达式。例如下面的语句：

```
for v = A, ..., end
```

就等价于：

```
for j = 1:n, v = A(:,j); ..., end
```

当表达式用冒号来表示时，那么列值都是标量，例如MATLAB中的语句：`for v=i:j: k`。循环是可以嵌套的：

```
for variable I = expression A
    statements 1
    for variable II = expression B
        statements 2
    end
    statements 3
end
```

例12.4

(a) 下列矩阵有三个非零对角值 (这是一个三对角阵):

$$\mathbf{A} = \begin{pmatrix} 5 & 1 & 0 & 0 & 0 \\ 1 & 5 & 1 & 0 & 0 \\ 0 & 1 & 5 & 1 & 0 \\ 0 & 0 & 1 & 5 & 1 \\ 0 & 0 & 0 & 1 & 5 \end{pmatrix}$$

这个矩阵可循环使用命令 `for` 来创建。这种方法在任何标准的程序设计语言中都是一样的。

```
A = [];

for k = 1:5
    for j = 1:5

        if k == j
            A(k,k) = 5;
        elseif abs(k-j) == 1
            A(k,j) = 1;
        else
            A(k,j) = 0;
        end

    end
end
```

这里的分号';'是非常重要的。如果这些赋值语句没有分号,矩阵 A 将在屏幕上输出 25 次,每一次 A 中的元素将被赋值一次。

同样也会遇见由于不注意使用 for 循环而导致无效操作的例子。通过定时器时钟就能清楚地计算出花费的时间。例如, for 循环,见例 12.21。下面的命令能完成上面同样的事情,并且更加有效:

```
A = zeros(5);

for k = 1:4
    A(k,k) = 5;
    A(k,k+1) = 1;
    A(k+1,k) = 1;
end
```

```
A(5,5) = 5;
```

这个矩阵能通过更加快速有效的方法得到,但是使用命令 diag 更加清楚。

```
A = [];

A = diag(5*ones(5,1)) + diag(ones(4,1),1) + ...
    diag(ones(4,1),-1);
```

这种结构的大矩阵应该创建成稀疏矩阵;参见第 9 章。

(b) 在区间 $[-2, -0.75]$ 内,步长为 0.25,对函数 $y=f(x)=1+1/x$ 求值,并列表。将所得 x 值和 y 值分别存入向量 r 和 s 中,并列表显示:

```
r = []; s = [];

for x = -2.0:0.25:-0.75
    y = 1 + 1/x;
    r = [r x];
    s = [s y];
end

[r; s]'
```

此表也能够不用for循环语句创建，其结果为：

```
ans =
-2.0000    0.5000
-1.7500    0.4286
-1.5000    0.3333
-1.2500    0.2000
-1.0000     0
-0.7500   -0.3333
```

(c) MATLAB 命令sum(A)给出一行向量，向量的元素是矩阵A每一列元素的和。用下面的程序能得到相近的结果：

```
A = [1 2 3;4 5 6]

sum_v = [];

for v = A
    sum_v = [sum_v sum(v)]
end

disp('Compare w/ sum(A):');
disp(sum(A));
```

其结果是：

```
A =
    1    2    3
    4    5    6

sum_v =
    5

sum_v =
    5    7

sum_v =
    5    7    9

Compare w/ sum(A):
    5    7    9
```

(d) 将下列MATLAB命令保存在文件qrmethod.m中：

```
% 矩阵A及整数m和n应在调用该文件之前定义好。
% 在变换成上海森伯形式之后使用QR方法。
% n步以后结束。
% 每隔m步输出结果。
```

```
A = hess(A);

for i = 1:n
    [Q,R] = qr(A);
    A = R*Q;
    nd = norm(diag(A,-1));
```

```

if rem(i,m) == 0
    A, i, nd
end

```

```
end
```

下面的程序是用来完成非移位的 QR方法(见8.2节)30次迭代，每隔15次输出结果：

```

A0 = [-9 -3 -16;13 7 16;3 3 10 ];
m = 15; n = 30;
format long;
A = A0;
qrmethode;

A =
    9.98997467074377    22.62301237506363   -15.53274662438004
    0.00708686385759   -5.98568512552925    5.77401643542405
                     0    0.00741470005235    3.99571045478546

i =
    15

nd =
    0.01025677416162

A =
    10.00000471624660    22.62743993744967    15.51339551121122
   -0.00000333488655   -6.00001449452640   -5.77348898879412
                     0    0.00001693654612    4.00000977827978

i =
    30

nd =
    1.726175143943722e-05

```

(e) 在7.5节中定义了命令planerot。这个算法是使用该命令来返回一个矩阵，该矩阵的0元素都在任何作为参数输入的 $m \times n$ 矩阵的主对角线的下方。

```

function B = Givens(A)
%
% 如果将A矩阵乘函数B返回的矩阵，就能将大小为 m×n的A矩阵变成上三角阵。也就是说，根据 Q = 和
R=B*A，其中满足Q*R=A，该函数可以对A进行QR分解。
[m,n] = size(A);
B      = eye(m);

for j = 1:n
    for i = j:m
        for k = (i+1):m
            G      = eye(m);
            Plan    = planerot([A(j,j) A(k,j)]');
                                %找出2×2的矩阵

            G(j,j) = Plan(1,1); G(k,j) = Plan(2,1);
            G(j,k) = Plan(1,2); G(k,k) = Plan(2,2);

```

% 在 $m \times m$ 矩阵中正确定位

```
B = G*B;
A = G*A ;      % <- To see the step-by-step reduction
                % of AA.remove this semicolon.
end
end
end
```

在这个算法中，每一步的循环将 A 中的两个元素取出构成 2×2 的矩阵，然后将矩阵主对角线下的一个元素赋值为零。这个结果矩阵可用来创建 QR 因式分解。

定义一个检测矩阵 Atest：

$$\text{Atest} = \begin{pmatrix} 1 & 2 & 3 & 1 & 2 & 3 \\ 4 & 4 & 1 & 2 & 2 & 1 \\ 7 & 6 & 3 & 2 & 1 & 1 \\ 1 & 2 & 1 & 0 & 0 & 2 \end{pmatrix}$$

下面的命令为：

```
Atest = [1 2 3 1 2 3; 4 4 1 2 2 1;
          7 6 3 2 1 1; 1 2 1 0 0 2];

Giv = Givens(Atest);
Q = Giv', R = Giv*Atest
QR = Q*R      % 仅仅是检测
```

给出 MATLAB 输出：

```
Q =
    0.1222    0.6630    0.6674    0.3162
    0.4887    0.1842   -0.5721    0.6325
    0.8552   -0.2947    0.2860   -0.3162
    0.1222    0.6630   -0.3814   -0.6325

R =
    8.1854    7.5745    3.5429    2.8099    2.0769    1.9547
    0.0000    1.6208    1.9523    0.4420    1.3997    3.2047
   -0.0000   -0.0000    1.9069    0.0953    0.4767    0.9535
    0.0000    0.0000    0.0000    0.9487    1.5811    0.0000

QR =
    1.0000    2.0000    3.0000    1.0000    2.0000    3.0000
    4.0000    4.0000    1.0000    2.0000    2.0000    1.0000
    7.0000    6.0000    3.0000    2.0000    1.0000    1.0000
    1.0000    2.0000    1.0000         0         0    2.0000
```

注意，通过乘 Q 和 R 我们又能得到初始矩阵 Atest，因此 $QR = \text{Atest}$ 。

(f) 以下程序通过使用两个 for 循环和平面组合来画出雪花图形。这个算法生成 Helge von Koch 曲线，这是一个不规则碎片例子。程序中使用的图形命令定义在第 13 章中，这里的注释主要说明其功能。该算法将当前几何图形每一面分成了相同的三个部分。第一部分和最后部

分是新几何学的两个方面。中间的部分用等边的三角形的两个边替代，如图 12-1所示。

如果将迭代进行下去，几乎平面的每一部分都将被覆盖到。事实上，不规则碎片的尺寸为1.2619，比1大一点而比2小。

```
% 文件：Koch.m
% 该程序画出Helge von Koch 雪花，一个不规则碎片图形

clear;                                %删除旧变量

%向量在平面中新定义一个三角形。这是开始的几何状态。

new = [0.5+(sqrt(3)/2)*i,-0.5+(sqrt(3)/2)*i,...
       0,0.5+(sqrt(3)/2)*i];

plot(new);                             %画出三角形并等待0.5秒
pause(0.5);

% 迭代5次：                            向量old是前一次迭代

for k = 1:5;
    old = new;
    [m,n] = size(old);
    n = n - 1;

    % old定义了图中的n-1条边。
    % 对每条边：定义4个新点(其中一个是'old')。

    for j = 0:n-1;
        diff = (old(j+2)-old(j+1))/3;
        new(4*j+1) = old(j+1);
        new(4*j+2) = old(j+1) + diff;
        new(4*j+3) = new(4*j+2) + diff*((1-sqrt(3)*i)/2);
        new(4*j+4) = old(j+1) + 2*diff;
    end;

    % 向量new的最后一个元素与向量old的最后一个元素相同。

    new(4*n+1) = old(n+1);

    plot(new);                          %画出新图并等待0.5秒。
    pause(0.5);

end;

% 移开坐标轴，并使其等长度，图形会更匀称。
```

```
axis off; axis square;
```

执行程序，就可得到一个逐渐复杂的图形。

图12-2给出了最后的图形。

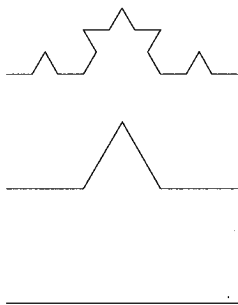


图12-1 von Koch算法对直线的两次迭代的结果图

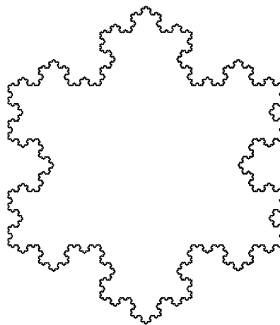


图12-2 5次迭代后的Helge von Koch不规则碎片图形，其原始几何图形为三角形

只要逻辑表达式为真，while命令将反复执行程序语句。像for语句一样程序体由一个end来结束。使用while循环来表示整个while语句，具体形式如下：

```
while, statements, end
```

通常，while循环有如下形式：

```
while logical expression
    statements
end
```

将其写在一行的形式为：

```
while logical expression, statements, end
```

while循环能够像for循环一样嵌套：

```
while logical expression A
    statements 1
    while logical expression B
        statements 2
    end
    statements 3
end
```

例12.5

(a) 构造一个特征值在1和-1之间的 2×2 的随机矩阵，可以用下面的迭代来实现：

```
A=rand(2); % 构造一个特征值在1和-1之间的矩阵。

while max(abs(eig(A))) >= 1
    A = rand(2);
end

e = eig(A);
TheText = ['lambda_1 = ', num2str(e(1)), ...
           ', lambda_2 = ', num2str(e(2))];
```

A % 输出显示矩阵及其特征值。

```
disp(TheText)
```

运行程序，得到：

```
A =
    0.1517    0.6628
    0.2098    0.5295

lambda_1 = -0.077395, lambda_2 = 0.7586
```

其中变量lambda_1 和lambda_2是矩阵A的特征值。

加入一个变量用来计算迭代的次数。注意：如果程序再运行一次，会得到不同的结果。

```
A = rand(2); niter = 1;

while max(abs(eig(A))) >= 1
    disp(['Step ' num2str(niter)]);
    disp(['Eigenvalues: ' num2str(eig(A)', 5)]);
    A = rand(2);
    niter = niter + 1;
end

disp(['Final result - step ' num2str(niter)]);
disp(['Eigenvalues: ' num2str(eig(A)', 5)]);
```

结果是：

```
Step 1
Eigenvalues: 1.3871    -0.41031
Step 2
Eigenvalues: -0.18924    1.2166
Step 3
Eigenvalues: 1.0151    -0.11415
Final result - step 4
Eigenvalues: 0.054835    0.95675
```

(b) 函数 $\ln(1+x)$ 的Maclaurin序列为：

$$\ln(1+x) = \sum_{k=1}^{\infty} \frac{(-1)^{k+1} x^k}{k}$$

用 $x=0.5$ 带入，并把Maclaurin序列的各项相加，直至要加的下一项系数小于内建变量 ϵ 。计算出所加项的个数。用下面的方法来实现：

```
lnsum = 0; x = 0.5; k = 1;

while abs((x^k)/k) >= eps
    lnsum = lnsum + ((-1)^(k+1))*((x^k)/k);
    k = k + 1;
end

disp(['The sum = ', num2str(lnsum), ...
    ', number of iter = ', num2str(k) ]);
```

给出的结果为：

```
The sum = 0.40547, number of iter = 47
```

检验这个结果：

```
ln = log(1.5)

ln =
    0.4055
```

有时在循环正常结束前终止循环是有用的，这可以用命令 `break` 来实现。如果 `break` 命令用于嵌套循环的内部循环，那么只能终止内部循环，外部循环仍然继续。

应该尽量避免使用 `break`，因为使用命令 `break` 的程序通常不易理解和维护。这样的程序通常被改写成没有 `break` 的程序。

例12.6

通过迭代求机器最小正数。

(a) 使用 `break` 的 `for` 循环：

```
macheps = 1;

for i = 1:1000
    macheps = macheps/2;

    if macheps + 1 <= 1
        break
    end

end

macheps = macheps*2
```

在 Sun SPARC 工作站上运行的结果为：

```
macheps =
    2.2204e-16
```

(b) 未使用 `break` 的 `while` 循环：

```
macheps = 1;

while macheps + 1 > 1
    macheps = macheps/2;
end

macheps = macheps*2
```

12.3 M文件的其他相关内容

在2.9节中介绍了M文件。在本节中将涉及到与M文件相关的其他方面的内容。

`inline` 命令可以不用M文件就能创建函数；参见5.1.4节。

MATLAB能处理递归函数。这样的函数能调用本身，但要通过改变一些判断条件以防止该程序进入死循环。

例12.7

将下面的M文件命名为sqpulse.m：

```
function f = sqpulse(n,x)

% 递归函数用于求和：
% 1/2 + 2/pi cos(x pi) + ... + 2sin(n pi/2)cos(n x pi).
% For n --> inf 这将等于阶跃的平方

if (n == 1)
    f = 1/2 + 2/pi*cos(x*pi);          % 递归终止准则
else
    f = 2*sin(n*pi/2)/n/pi*cos(n*x*pi) + ...
        sqpulse(n-1,x);
end
```

如果 n 足够大，对 $x \in [-0.5, 0.5]$ ，函数将返回1。对于通过加上一个偶数也可变成属于这个集合的其他的 x 值，也就是如果 $x = -1.75$ ，函数将给出 $\text{sqpulse}(n, x) = 1$ ，因为 $-1.75 + 2$ 等于0.25。对于其他所有的数值，函数 sqpulse 都是0；如图12-3所示。

如果 n 选得太小，阶跃的平方将会是正弦的，因为阶跃的平方是由余弦函数得到的。

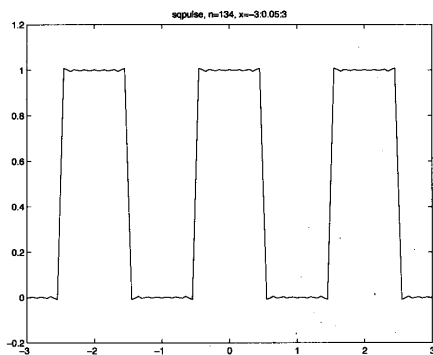


图12-3 用标题中的 x 和 n 值，函数 $\text{plot}(x, \text{sqpulse}(n, x))$ 的图形

已经在M文件中加入很多注释。所有注释是以百分号 %开头的：

% 注释。

在编写程序时加入注释是很好的习惯，这有助于以后对程序功能和程序运行的理解。

lookfor命令(见2.7节)能浏览所有规定的M文件中第一行注释行。因此，将关键词放入第一行注释行是比较好的。

例12.8

```
% Course:      Applied Linear Algebra; Uppsala Univ.
% Assignment:  homework #7 - LU-decomposition
% Date:        980505
% Author:      Tomas P.
% File name:   assignment7.m
```

```

%-----
% diary assignment          % Stores output in file "assignment".

txt1 = ...
    sprintf('\nAssignment #7, Syst. of Eq., T.P. 980505.\n');
txt2 = ...
    sprintf('Ax=b <==> (LU)x=b <==> (i) Ly=b + (ii) Ux=y.\n');

disp([txt1 txt2]);

A = [4 3;1 2];              % 创建系统矩阵A
b = [5;10];                 % 右边向量b

[L,U] = lu(A);              % 矩阵A的LU分解

%L, U                        % 显示L和U

y = L\b;                    % (i) 前除
x = U\y;                    % (ii) 后置换
disp('Solution of Ax=b:');  % 写出结果向量
disp(x);

%x2 = A\b                   % 检查解

%程序结束

显示结果为：
Assignment #7, Syst. of Eq., T.P. 980505.
Ax=b <==> (LU)x=b <==> (i) Ly=b + (ii) Ux=y.
Solution of Ax=b:
    -4
     7

```

在调试过程中可使用命令符号 ‘ % ’。用关键字命令 ‘ commenting away ’ 能跟踪发现错误；参见 12.7 节。

当 MATLAB 第一次执行一个函数时，系统将创建一个该函数编译后的形式以便下一次调用该函数。可以将编译后的变量存入一个文件，也叫做 P 文件。P 文件的使用与 M 文件使用相同，但不能列出 P 文件。如果想要隐藏代码，这种方法是非常好的。

命令集 109 P 文件

<code>pcode fun1 fun2...</code>	编译函数 <code>fun1, fun2, ...</code> , 并以相同的名字 <code>fun1, fun2, ...</code> 将其存成文件，但后缀名为 ‘ .p ’。
<code>- inplace</code>	如果给出 <code>- inplace</code> ，则象 M 文件一样将所有的 P 文件保存到相同的路径下。
<code>[M,MEX]=inmem</code>	返回一个细胞矩阵 <code>M</code> ，包含的是内存中编译过的 M 文件的文件名字符串。如果给定 <code>MEX</code> ，则返回已加载的 MEX 文件列表；参见 15.2.1 节和 15.3.1 节。

命令 `echo` 用来切换正在执行的命令的显示和不显示。当回显为打开状态时，所有的命令和注

释将被显示在屏幕上，这种方法对于调试程序非常有用。命令`echo`可写成`echo on`和`echo off`。

函数文件不会被以上操作所影响。下面是函数文件的保留字：

命令集110 从函数文件中回显

<code>echo fname on</code>	打开函数 fname.m 的显示模式。
<code>echo fname off</code>	关闭函数 fname.m 的显示模式。
<code>echo fname</code>	切换函数 fname.m 开或关的显示模式。
<code>echo on all</code>	打开所有函数的显示模式。
<code>echo off all</code>	关闭所有函数的显示模式。

通常使用`clear`命令能将内存中的所有M文件清除。这种清除方法可以用下面的命令来控制：

命令集111 M锁定文件

<code>mlock</code>	锁住正在运行的M文件以便其不会被 <code>clear</code> 命令作用。
<code>mlock filename</code>	锁住M文件 filename 。
<code>munlock</code>	解锁正在运行的M文件，以便调用 <code>clear</code> 命令将其清除。
<code>munlock filename</code>	解锁M文件 filename 。
<code>mislockedfilename</code>	如果正在运行的M文件对于给出的 filename ，或者 filename ，处于锁定状态，则都返回1；否则返回0。

一个函数可以有0个、1个或者多个参量(参数)，调用同一个函数可以带有不同个数的参量。例如，函数`triu(A)`返回一个上三角矩阵，而`triu(A,1)`则返回一个严格的上三角矩阵。

命令集112 参数个数

<code>nargin</code>	这是一个变量，指定调用函数所带参数的个数。
<code>nargout</code>	这是一个变量，指定调用函数所返回的参数的个数。
<code>inputname(x)</code>	返回输入表上数字x所在位置的输入参数变量的名字。如果用一个表达式代替已命名的参数，则返回一个空字符串。
<code>errorstr=</code> <code>nargchk(min,</code> <code>max,number)</code>	用来控制函数的输入参数的个数。参数 number 是由 <code>nargin</code> 指定的输入参数的个数。如果 number 的值超过 min 到 max 的区间范围，系统将返回一个错误字符串 errorstr ，否则将返回一个空矩阵。
<code>varargin</code>	是函数可带有任意多个输入参数的细胞矩阵。
<code>varargout</code>	是函数可带有任意多个输出参数的细胞矩阵。

例12.9

可以将参数的可变个数定义成默认值，如果没有特别指定，就使用这个值。存放在**random.m**中的函数**random**可用来创建服从正态分布的 $m \times n$ 随机矩阵。如果期望值 v 没有给定，

则令 $v=0$ 。

```
function A = Random(m,n,v)
% 返回一个方差为0、期望值为 v的m×n矩阵。
% 如果v没有给定，则使用v=0。
if nargin == 2, v = 0; end
A = randn(m,n) + v;
```

调用

```
A = Random(2,2,4); B = Random(2,2);
```

给出A元素的期望值为4，B元素的期望值为0。

因为细胞矩阵中的细胞可以是不同的数据类型，所以就可以将不同类型的输入、输出参数放入到细胞矩阵 **varargin** 和 **varargout** 中。示例如下：

例12.10

用一个函数来对任意多个向量计算每个向量的平均值、中位值和标准方差。可以使用下面的方法来计算：

```
function [varargout] = stat(varargin)

for i = 1:length(varargin)
    x = varargin{i};    % 取出输入参数
    y.medel = mean(x);  % 将结果保存到结构中
    y.median = median(x);
    y.std = std(x);
    varargout{i} = y;   % 将结果放入输出变量中
end
```

上面的程序中首先从细胞矩阵 **varargin** 中挑选出一个输入的参量，然后计算出给这个参量的平均值、中位值和标准方差并将其结果保存到结构 **y** 中。现在将这个结构放入到细胞矩阵 **varargout** 的一个细胞中。运行上面的程序能得到下面的结果：

```
a = [1 6 8 9];
b = [42 12 56 72 5 34];
[ares,bres] = stat(a,b)
```

```
ares =
    medel: 6
    median: 7
    std: 3.5590
```

```
bres =
    medel: 36.8333
    median: 38
    std: 25.5689
```


也许会将普通的输入、输出参数和 `varargin` 和 `varargout` 混淆，但 `varargin` 和 `varargout` 在每一个参数列表的最后。下面是具有不同函数名的几个例子。

例12.11

函数 `test1` 接收参数 x ，然后接收任意多个附加的参数，并只返回一个输出变量：

```
function y=test1(x, varargin)
```

函数 `test2` 只接收一个输入参量。但是可以返回所需的输出参量和任意多个附加的输出参量：

```
function[y, varargout]=test2(x)
```

一个函数也可以有一个可选的返回参量。例如 `bar(x,y)` 在向量 x 处画出向量 y 元素的图形，而 `[xx,yy]=bar(x,y)` 不画出图形，而只返回向量 xx 和 yy 。命令 `plot(xx,yy)` 也能画出与 `bar(x,y)` 相同的图形。有关 `bar` 更多的内容参见 6.5 节。

例12.12

下面的 M 文件 `ngon.m` 能计算出 $c^n=1$ 的根作为缺省值，但也可定义一个复数 z 作为可选的输入参量，那么就计算 $c^n=z$ 的根。

这些根在复平面上定义了一个 n 边形。如果不给出任何输出参量，就画出多边形。如果给出输出参数，就得到一个在复平面上定义多边形边角的复数向量。如果给出两个输出参数，就得到两个在平面上定义多边形的实数向量。

```
function [aa,bb] = ngon(n,z)
```

```
% 文件: ngon.m
```

```
%  $c^n=z$  的  $n$  个根是一个复平面上  $n$  边形的边角
```

```
% 对每种情况: 0, 1, 2, 检查返回值的个数。
```

```
% 如果只有一个输入变量，则将  $z$  设为默认值 1。
```

```
if nargin == 1
```

```
    z = 1;
```

```
end
```

```
% 根为  $c=re+i*im$ ,  $k=1:n$ 
```

```
k = 1:n;
```

```
re = abs(z)*cos((angle(z)+(k-1)*2*pi)/n);
```

```
im = abs(z)*sin((angle(z)+(k-1)*2*pi)/n);
```

```
xx = [re re(1)]; yy = [im im(1)];
```

```
% 检查要求的输出参数的个数
```

```
% nargin == 0:
```

```
% 根据  $z$  的相角在灰色底面上画出  $n$  边形。
```

```
% nargin == 1:
```

%返回复数向量，以便plot(xx, yy)可在复平面上画出多边形的轮廓。

% nargout == 2:

%返回实数xx和yy向量，以便plot(xx, yy)画出多边形轮廓。

```
if nargout == 0
    patch(xx,yy,[abs(angle(z)/pi) abs(angle(z)/pi)...
              abs(angle(z)/pi)])
    axis('equal')
elseif nargout == 1
    aa = xx + yy*i;
else
    aa = xx;
    bb = yy;
end
```

在2.4节中介绍了命令angle，在14.2.11中介绍命令patch。为了说明ngon的功能，给出下面的命令：

```
subplot(2, 2, 1); ngon(5);
```

ngon将画出左上角图形，这个方程的解是 $c^n=1$ ， $n=5$ ：

```
subplot(2, 2, 2); cv=ngon(5, i); plot(cv); 'axis(')
```

右上角图形是在复平面上用方程式为 $c_n=i$ 、 $n=5$ 的解画出的：

```
subplot(2,2,3); [rv1,rv2] = ngon(5,-1);
plot(rv1,rv2); axis('equal')
```

左下角图形是用方程 $c^n=-1$ 、 $n=5$ 的解画出的多边形，和 $rv1+i*rv2$ 相等：

```
subplot(2, 2, 4); ngon(5,i);
```

右下角图形的多边形的角是方程 $c^n=-i$ 、 $n=5$ 的解。

这些多边形如图12-4所示。subplot和plot的定义见第13章。

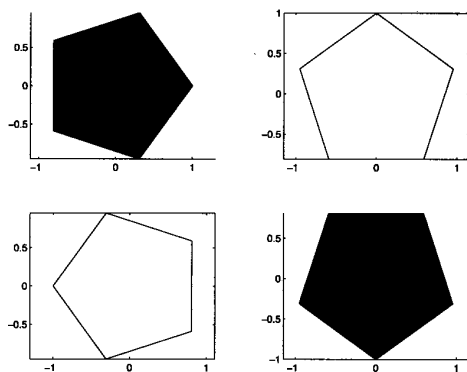


图12-4 用户定义的函数ngon生成的五边形

函数文件中的所有变量都是局部变量。因此，在一个函数文件中的变量与MATLAB工作区中的同名变量是完全不同的变量，它们存在内存的不同位置。象所有的规则一样，这个规则有一个例外：全局变量可在MATLAB中使用。

一个全局变量在所有声明它为 `global` 的函数文件中都是可访问的。使用命令 `who` 和 `whos` 可以知道哪些变量声明为全局变量。要清除全局变量，参见 2.3 节。

通常变量的值在函数调用时会发生改变。如果要想变量的值不变，应该将它声明为 `persistent`。如何声明可参见例 12.13。

例12.13

创建函数 `persdemo`。

```
function persdemo(x)
```

```
if x
    persistent TIMESUSED;
    TIMESUSED
end;
if (exist('TIMESUSED','var') & ~isempty(TIMESUSED))
    TIMESUSED = TIMESUSED + 1;
else
    TIMESUSED = 1;
end;

TIMESUSED

if TIMESUSED < 3
    disp('Keep on calling');
else
    disp('Type clear persdemo to clear persistent value');
end;
```

运行程序，可得：

```
persdemo(1)

TIMESUSED =
     []
TIMESUSED =
     1
Keep on calling
persdemo(1)

TIMESUSED =
     1
TIMESUSED =
     2
Keep on calling

persdemo(1)

TIMESUSED =
     2
TIMESUSED =
     3
Type clear persdemo to clear persistent value
```

以 0 作为参量调用可使 `TIMESUSED` 被解释成不能保持其值不变。

```
persdemo(0)
```

```
TIMESUSED =  
1  
Keep on calling
```

```
persdemo(0)
```

```
TIMESUSED =  
1  
Keep on calling
```

如果persdemo再次以TIMESUSED作为不变值来调用,就能得到:

```
persdemo(1)
```

```
TIMESUSED =  
3  
TIMESUSED =  
4  
Type clear persdemo to clear persistent value
```

要零化变量必须将其从内存中清除。

```
clear persdemo,persdemo(1)
```

```
TIMESUSED =  
[]  
TIMESUSED =  
1  
Keep on calling
```

下面是控制M文件的命令。

命令集113 M文件的控制

run filename	运行命令文件filename, filename包括文件的全部路径和文件名。
pause	暂停M文件的运行,按下任意键后继续运行(见例13.19(c))。
pause(n)	暂停运行秒后继续执行。这个暂停命令在显示大量图形时非常有用。
pause off	指示MATLAB跳过后面的暂停。
pause on	指示MATLAB遇到暂停时执行暂停命令。
break	终止for和while循环。如果在一个嵌套循环中使用该命令,只有内部循环被终止;参见12.2节。
return	结束M文件运行, MATLAB立即返回到函数被调用的地方。
error(str)	终止M文件的运行,并在屏幕上显示错误信息和字符串 str。
errortrap state	决定当有错误发生时是否停止运行。 state的值可为on(此时捕获错误信息并继续执行)或off(发生错误时停止运行)。
global	声明变量为全局变量。全局变量能在函数文件中被访问,而不必包括在参数列表中。命令global后面是以空格分开的变量列表。

	声明为全局变量将保持其全局性直至工作区完全被清除或使用了clear global命令。
isglobal(name)	如果变量name是全局变量，则返回1；否则返回0。
keyboard	将键盘当成一个命令文件来调用。当给出一个内部的M文件，运行将被暂停，这样就可在MATLAB的命令窗口中给出命令。提示符k>>表示这种特殊状态。当执行一个M文件时，这是检查或改变参数变量的一个很好的方法，所有命令都可以在命令窗口中输入。当输入关键字return时，M文件将继续运行。如果在一个函数文件中调用keyboard，那么该函数的工作区和它的全局变量都可访问。命令keyboard在调试过程中很有用。
mfilename	返回正在运行的M文件名字字符串，一个函数能用这个函数获得它自己的名字。
warning(message)	在字符串message中显示一条警告信息但不终止程序运行。
warning val	控制警告信息。val合法的值有：
	off 终止后面的警告信息。
	on 将警告信息再次打开。
	backtrace 显示造成警告的所在命令行。
	debug 当发生警告时激活调试器。
	once 每部分只显示一次与警告向下兼容的图形句柄。
	always 显示所有的警告信息。
[vt,f]=warning	将当前警告状态vt和警告频率f作为字符串返回。

对于用户自定义的函数可以有子函数。这些子函数只能被其与M文件同名的主函数或在M文件中的其他函数所调用。在一个M文件中只能有一个主函数。在文件main.m中有一个函数结构及其子函数的示例：

```
function y=main(x)           % 主函数。
...                           % 程序语句。
z1=under1(x);                 % 调用第1个子函数。
...                           % 程序语句。
y=under2(a);                  % 调用第2个子函数。
...                           % 程序语句。
function y=under1(x)          % 第1个子函数。
...                           % 程序语句。
function y=under2(x)          % 第2个子函数。
...                           % 程序语句。
[b1,b2]=under3(a1,a2);        % 调用第3个子函数。
...                           % 程序语句。
function y=under3(x1,x2)      % 第3个子函数。
...                           % 程序语句。
```

还有一类函数称为私有函数，这一类函数是放入一个叫private子目录中的M文件，私有函数只能被private直接上层目录中的函数调用。

当MATLAB在调用在M文件中的函数时，它首先查找子函数，再查找私有函数，最后在MATLAB的搜索路径中查找函数；见命令集 22。这就表明用户可以创建与MATLAB函数同名的私有函数，并将其放入 **private** 子目录中，这样程序就能对它们进行调用。同时，其他路径下的程序能调用和私有函数的同名的M文件，但此时执行的是MATLAB的函数。

12.4 将函数作为参数传递给其他函数

在大部分高级语言中，如Pascal或FORTRAN，能够创建一个通用函数F，该函数是以另一个函数f作为参数的。在MATLAB中同样也能这样做，将函数的表达式或函数名包含在字符串f中。在函数F中，函数f能用eval或feval来求值。

命令eval(见5.1.4节)对包含MATLAB表达式的字符串求值，比如，这个字符串可包含数学表达式：

```
a = eval('sin(2*pi)') or  
x = 2*pi; b = eval('sin(x)')
```

如果要用命令val，则字符串f中使用的变量必须要和F中的变量名字相同。

命令feval既可以对内建函数如sin、也可以对保存在M文件中的函数求值。调用feval的形式如下：

```
a = feval('sin',2*pi)
```

命令集114 求值函数

<code>feval(fcn,x1,...,xn)</code>	对字符串fcn给出的函数求值。参数x1, ..., xn是按出现的顺序传递给函数。feval命令通常在以其他函数为参数的函数内使用。
<code>[y1,y2,...]= feval(fcn,x1,...,xn)</code>	同上，但返回多个变量。

假设函数fcn带有元素操作运算符，这些运算符也就是+，-，.*，./，.\，.^。如果x是一个向量，则命令feval(fcn,x)也返回一个向量。如果在F中使用feval，则将一个被赋予向量值的函数作为参数传给F是没有问题的。如果在F中使用了命令eval，并且eval直接作用于向量，那么，算术操作运算符必须使用在函数f的参数字符串中。

例12.14

现在要编写MATLAB函数来得到 $f(x)$ 的数值表， x 在区间 $[a, b]$ 内，步长为 k 。假定函数f是使用算术操作符定义的。

(a) 用feval：

输入参量：包含函数名的字符串A，上下限a和b及步长k。

输出参量：有两列向量的矩阵A，矩阵值为x值和 $f(x)$ 。

下面的函数保存在文件Func tabl.m中：

```
function Y=Func tabl(f, a, b, k)  
% 在区间[a,b]中对一个标量函数求值，x(j)=a+j*k。
```

% 结果在表中，其中包含 x 值和 $f(x)$ 。

```
x = a:k:b;
z = feval(f,x);
Y = [x;z]';
```

(b) 用`eval`。将命令字符串作为函数的输入参量就能得到相同的结果。使用 `eval` 命令对这个命令字符串求值；参见 5.1.4 节。

输入参量：包含定义函数的命令字符串 A ，上下限 a 和 b 及步长 k 。

输出参量：有两个列向量的矩阵 A ，矩阵值为 x 值和 $f(x)$ 。

下面的函数保存在文件 **Funcstab2.m** 中：

```
function Y=Funcstab2(f, a, b, k)
% 在区间[a, b]中对于一个标量函数求值， $x(j)=a+j*k$ 。
% 结果在表中，其中包含 $x$ 值和 $f(x)$ 。
x=a:k:b;
z=eval(f);           % f必须是x的函数。
Y=[x; z]';
```

假设想要得到函数 $\text{oneplusx}(x)=1+x$ 的函数值表， x 在区间 $[-1, 1]$ 中。函数 **oneplusx** 保存在文件 **oneplusx.m** 中。下面的例子即为如何调用函数 **Funcstab1** 和函数 **Funcstab2** 来创建这样的表。它们都给出相同的结果。

```
Tab = Funcstab1('oneplusx',-1,1,0.25)
```

```
Tab =
-1.0000      0
-0.7500    0.2500
-0.5000    0.5000
-0.2500    0.7500
      0    1.0000
 0.2500    1.2500
 0.5000    1.5000
 0.7500    1.7500
 1.0000    2.0000
```

另一种创建表格的方法为：

```
Tab = Funcstab2('oneplusx(x)',-1,1,0.25)
```

or

```
Tab = Funcstab2('1+x',-1,1,0.25)
```

12.5 结构

MATLAB 能创建有多个域的结构 **structs**。在许多现代程序语言中都用到了结构，如 C/C++ 中的结构和 Pascal 中的记录，但它们之间也有一些差别。可以直接分配或使用 `struct` 命令来创建一个结构。结构类型是可变的，所以结构的向量不必有相同的数据类型。但对于某些域要求其具有相同的数据类型；见例 12.15。对于整个结构来说，唯一的限制就是域只能包含标量或者固定维数的细胞矩阵。

为了从这些域中获取数据，应该在结构名和域名之间使用句点表达式，‘.’；见例12.15。
下面对结构操作的函数都是可用的：

命令集115 有关结构的函数

<code>struct(f1,V1,f2, V2,...)</code>	返回带有域 $f1, f2, \dots$ 及其相应域值 $V1, V2, \dots$ 的结构。参数 $V1, V2, \dots$ 可以是相同大小的细胞矩阵或标量。
<code>fieldname(S)</code>	返回结构 S 中域名的列向量。
<code>getfield(S,f)</code>	返回结构 S 中域 f 的值。也可以写成 $S.f$ 。
<code>isstruct(S)</code>	如果 S 是结构，则返回1；否则返回0。
<code>isfield(x)</code>	如果 S 是结构中的一个域，则返回1；否则返回0。
<code>setfield(S,f,v)</code>	设定结构 S 中域 f 的值为 v ，也可以写成 $S.f=v$ 。
<code>rmfield(S,fvect)</code>	在向量 $fvect$ 中返回无域的结构 S 。
<code>struct2cell(S)</code>	返回带有结构 S 中值的细胞矩阵。
<code>handle2struct</code>	将图形层次句柄转换成带有 type 域的结构(对象类型，如 line 、 handle 、 properties 、 children 或 special)。高级图像和句柄将在第14章中讨论。
<code>struct2handle</code>	将一个结构转换成图形层次句柄。其域与 <code>handle2struct</code> 中的相同。
<code>[out1,out2,...]=</code>	将输入拷贝到输出，即 $out1=in1; out2=in2$ ；可参见 <code>helpdesk</code> 中的示例。
<code>deal(in1,in2,...)</code>	

例12.15

一个存储方程的结构，其名字和描述如下：

```
curve(1).name = 'Circle';
curve(1).function = '(x-a)^2 + y(-b)^2 = r^2';
curve(1).description =
'Circle with radius r centered in x = a, y=b';
```

使用`struct`在向量`curve`中创建其他元素：

```
curve(2) =
struct('name','line','function',2,'description','A two?')
```

尽管在第1种情况下域`function`中的值是字符串，而在第2种情况下为整数，这种方法还是有效的。注意：`curve`是两个结构的向量。

12.6 对象

在MATLAB中，对象不同于结构，因为存在着将函数和对象联系起来的可能性，并且对象是根据类来创建的。在面向对象的上下文中，这些函数通常被叫做 **methods**(方法)。这样可以保护类中的成员变量，而只有类方法才能对它们进行访问。

命令集116 面向对象的函数

<code>class(object)</code>	返回对象 <code>object</code> 的类名。
<code>class(object,class, parent1,parent2,...)</code>	返回 <code>object</code> 作为 <code>class</code> 的变量。如果返回的对象要有继承属性,则应给定参数 <code>parent1, parent2,...</code> 。
<code>isa(object,class)</code>	如果 <code>object</code> 是 <code>class</code> 类型,则返回1;否则返回0。
<code>isobject(x)</code>	如果 <code>x</code> 是一个对象,则返回1;否则返回0。
<code>superiorto(class1, class2,...)</code>	当调用方法时,控制优先权的次序。如果要将一个类定义成 <code>superiorto</code> ,首先就用这种方法。
<code>inferiorto(class1, class2,...)</code>	当调用方法时,控制优先权的次序。如果要将一个类定义成 <code>inferiorto</code> ,最后用这种方法。
<code>methods class</code>	返回类 <code>class</code> 定义的方法名字。

为了创建一个类对象,就必须定义类的属性 **properties**, 如创建一个类对象的模板。首先要创建一个目录。该目录必须与类同名但开头加上 @ 的记号(在VAX/VMS系统中使用\$符号), 然后根据类模板需要一个能创建对象的构造函数 **constructor**。这就是根据类模板返回变量的函数(在一个M文件中)。

例12.16

要创建一个名为 **curve** 的对象, 首先要创建一个目录 `@curve`, 然后在文件 `@ curve / curve.m`中创建构造程序。具体如下:

```
function l = curve(a)

% curve类的构造函数
% l = curve 创建并初始化一个curve对象
% 参数a可以是一个细胞数组, 其中一个细胞是数学函数, 另一个是说明或
% 另一个curve对象

% 数学公式必须和FPLOT要求的形式相同, 参见FPLOT

% 如果没有传递参数, 则返回包含x轴的一个对象

if nargin == 0 % 在此情况下为缺省的构造函数
    l.fcn = '0';
    l.descr = 'x axis';
    l = class(l, 'curve');

% 如果传递的参数是一个curve对象, 则返回该对象的副本
elseif isa(a, 'curve')
    l = a;
elseif (ischar(a{1}) & ischar(a{2}))
    l.fcn = a{1};
    l.descr = a{2};
    l = class(l, 'curve');
```

% 如果传递的参数是错误类型，则将给出错误信息

```
else
    disp('Curve class error #1: Invalid argument.')
end
```

例12.16中的`line l=class(lcurve')`给出了与类有关的变量。如果不考虑这个，则只返回一个不能对类方法访问的结构。利用方法才能使用对象的值，即，将与类连接并且分别在M文件中创建的函数放入类目录中。注意，对象是那些不同于其他面向对象语言参量中的一个，这里它的语法是`object.method(参数)`。

例12.17

根据这些描述，可以画出例 12.16的曲线：

```
function p = plot(l,area)

% curve.plot 在area中画出函数curve1的图形
% area必须是一个1×4的矩阵，元素为XMIN XMAX YMIN YMAX
% 或一个1×2的向量，元素为XMIN XMAX

% 产生步长和一个带x值的向量

step = (limits(2)-limits(1))/40;
x = limits(1):step:limits(2);

% 画出函数图形

fplot(l.function, limits);
title(description)
```

运行类curve，如下：

```
parabola = curve({'x*x' 'A parabola'})

parabola=
curve=object: 1-by-1

plot(parabola,[-2 2])
```

结果如图12-5所示。

对象的值只对方法是有用的。这给程序员提供了一种检查输入的变量的途径，而结构无法提供这种途径。因此，需要一个对于类特定的方法，以使用户能够改变对象的域。

在MATLAB中，不同的运算符对于不同的类是不同的。为了重载运算符，就必须用运算符的名字创建一个方法，方法的代码指明运算符的功能。

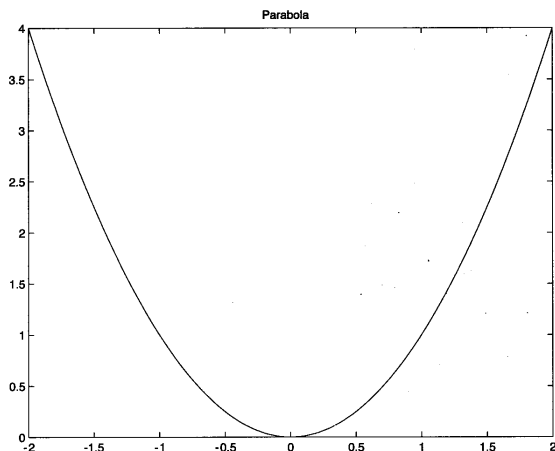


图12-5 对象parabola的图形

例12.18

为了能使两个`curve`类的对象相加，可以在目录`@curve`下创建一个M文件来重载加法运算符。注意，该方法对于对象的数据是完全可访问的。

```
function ltot = plus(l1,l2)
```

```
% 将曲线L1和L2相加
```

```
function = strcat(l1.function,' + ',l2.function);
description = strcat(l1.description,' plus ',l2.description);
ltot = curve({function description});
```

这样就能输入`l1+l2`或`plus(l1,l2)`，将两条曲线`l1`和`l2`相加。

输入`help operatorname`就能知道运算符到底是如何进行工作的，以及哪些对象有运算符重载。运算符定义在命令集117中。

命令集117 对象的运算符重载

<code>plus(a,b)</code>	表示 <code>a+b</code> 的函数。
<code>minus(a,b)</code>	表示 <code>a - b</code> 的函数。
<code>uplus(a)</code>	表示 <code>+a</code> 的函数。
<code>uminus(a)</code>	表示 <code>- a</code> 的函数。
<code>times(a,b)</code>	表示 <code>a.*b</code> 的函数。
<code>mtimes(a,b)</code>	表示 <code>a*b</code> 的函数。
<code>rdivide(a,b)</code>	表示 <code>a./b</code> 的函数。
<code>ldivide(a,b)</code>	表示 <code>a.\b</code> 的函数。
<code>mrdivide(a,b)</code>	表示 <code>a/b</code> 的函数。
<code>mldivide(a,b)</code>	表示 <code>a\b</code> 的函数。
<code>power(a,b)</code>	表示 <code>a.^b</code> 的函数。

<code>mpower(a,b)</code>	表示 a^b 的函数。
<code>lt(a,b)</code>	表示 $a < b$ 的函数。
<code>gt(a,b)</code>	表示 $a > b$ 的函数。
<code>le(a,b)</code>	表示 $a \leq b$ 的函数。
<code>ge(a,b)</code>	表示 $a \geq b$ 的函数。
<code>ne(a,b)</code>	表示 $a = b$ 的函数。
<code>eq(a,b)</code>	表示 $a = b$ 的函数。
<code>and(a,b)</code>	表示 $a \& b$ 的函数。
<code>or(a,b)</code>	表示 $a b$ 的函数。
<code>not(a)</code>	表示 a 的函数。
<code>colon(a,b)</code>	表示 $a:b$ 的函数。
<code>colon(a,s,b)</code>	表示 $a:s:b$ 的函数。
<code>transpose(a)</code>	表示 a' 的函数。
<code>ctranspose(a)</code>	表示 $a.$ 的函数。
<code>display(a)</code>	表示 a 的函数。
<code>horzcat(a,b,...)</code>	表示 $[a \ b \dots]$ 的函数。
<code>vertcat(a,b,...)</code>	表示 $[a; b; \dots]$ 的函数。
<code>subsref(a,i)</code>	表示 $a(i_1, i_2, \dots, i_n)$ 的函数，当重新定义该函数时，命令 <code>substruct</code> 非常有用。
<code>subsasgn(a,i,b)</code>	表示 $a(i_1, i_2, \dots, i_n) = b$ 的函数。
<code>subsindex(a,b)</code>	表示 $b(a)$ 的函数。

当函数有一个重载运算符时，而用户又希望能在 `builtin` 命令中使用普通运算符。

命令集 118 跳过重载运算符

`builtin(fcn,x1,x2,...)` 用参数 x_1, x_2, \dots 对内建函数 `fcn` 求值，而不是使用重载运算符。

在 MATLAB 中的类能从其他类中继承属性，这方面的使用已超出本书范围，推荐从 C++ 的书可以找到这方面的有关信息。要创建能继承属性的对象，可以在 `class` 命令中对其父对象进行设置。

12.7 调试和计时

MATLAB 中有些命令对调试 M 文件很有用。可以是在调试过程中寻找错误，设置和清除断点，逐行运行 M 文件，或在不同的工作区检查变量。所有的调试命令都是以字母 `db` 开头，已用过的命令 `dbtype` (见 2.9 节)，就能生成带行数的程序列表。

所有设置、清除和列出断点的命令在下面的命令集 119 中给出。

命令集119 断点

<code>dbstop in fname</code>	在M文件 fname 的第一可执行程序上设置断点。
<code>dbstop at r in fname</code>	在M文件 fname 的第 r 行程序上设置断点。如果第 r 行程序是不可执行的，则程序会在运行行可执行程序后停止。
<code>dbstop if v</code>	当遇到条件时，停止运行程序。当发生错误时，条件可以是 error ，当发生 NaN 或 inf 时，也可以是 naninf/infnan 。
<code>dstop if warning</code>	如果有警告，则停止运行程序。
<code>dbclear at r in fname</code>	清除文件 fname 的第 r 行处断点。
<code>dbclear all in fname</code>	清除文件 fname 中的所有断点。
<code>dbclear all</code>	清除所有M文件中的所有断点。
<code>dbclear in fname</code>	清除文件 fname 第一可执行程序上的所有断点。
<code>dbclear if v</code>	清除第 v 行由 dbstop if 设置的断点。
<code>dbstatus fname</code>	在文件 fname 中列出所有的断点。
<code>mbdstatus</code>	显示存放在 dbstatus 中用分号隔开的行数信息。

这些命令与下面列出的命令用来跟踪和控制 M 文件运行是非常有用的。在调试中使用 `try/catch` 结构也是非常有用的；参见 12.1 节。例如，如果使用 `try/catch` 来解例 5.8(a) 的问题，可以得到比用 `eval` 来解更多的通解。

命令集120 运行控制命令

<code>dbstep</code>	运行M文件的下一行程序。
<code>dbstep n</code>	执行下 n 行程序，然后停止。
<code>dbstep in</code>	在下一个调用函数的第一可执行程序处停止运行。
<code>dbcont</code>	执行所有行程序直至遇到下一个断点或到达文件尾。
<code>dbmex</code>	调试MEX文件的命令，见 15.2.1 节和 15.3.1 节。在 Windows 中或 Macintosh 系统中，这个命令是无效的。
<code>dbquit</code>	退出调试模式。

进行程序调试，要调用带有一个断点的函数。当 MATLAB 进入调试模式时，以 **K** 作为该状态的提示符：**K>>**。最重要的区别在于现在能访问函数的局部变量，但不能访问 MATLAB 工作区中的变量。以函数 **Factab.m** 来举例说明，该函数能产生 $1!, \dots, n!$ 的阶乘表。

例12.19

首先列出函数 **Factab.m** 的行数：

`dbtype Factab`

```

1  function Tab = Factab(n)
2  %
3  %生成一个1!, ..., n!的阶乘表
4
```

```

5  numbers = 1:n; facts = [];
6
7  for i = numbers
8      facts = [facts factorial(i)]
9  end
10
11  Tab = [numbers' facts'];

```

这个函数调用了函数 factorial，程序如下：

dbtype factorial

```

1  function p = factorial(nn)
2  %
3  %计算nn的阶乘
4
5  if ( nn == 0 )
6      p = 1;
7  else
8      p = nn*factorial(nn-1);
9  end

```

开始调试程序，在函数第一行可执行程序处设置一个断点，然后调用该函数。注意提示符中字母 K。

```

dbstop in Factab           % 在Factab中设置断点
Table = Factab(5);         % 调用Factab并调试
5  numbers = 1:n;
K>> dbstep                 % 执行一行程序
6  facts = [];
K>> numbers                 % 返回程序行数

```

numbers =

```

      1      2      3      4      5

```

```

K>> numbers = [numbers 6]  % 用数字6扩充向量

```

numbers =

```

      1      2      3      4      5      6

```

```

K>> dbstop 12              % 在第12行设置断点
K>> dbcont                 % 继续执行到下一个断点
12  Tab = [numbers' facts'];
K>> dbquit                 % 退出调试

```

```

dbstatus Factab            % 列出所有用过的断点
Breakpoints for Factab are on lines 5, 12.

```

函数调用另一函数可认为是嵌套的函数调用。MATLAB使用栈来跟踪工作区和函数中的变量，下面命令集中的命令就可用来在嵌套函数的工作区中进行切换。

命令集121 切换工作区

dbstep in	如果下一可执行程序行是函数调用，则跟踪函数。
dbup	切换到调用函数的工作区以检查变量。
dbdown	切换回被调函数的工作区。
dbstack	显示嵌套函数调用的栈。

例12.20

再次使用函数Factab；见例12.19。都将在Factab和factorial中设置断点开始：

```
dbstop Factab           % 在Factab中设置断点
dbstop factorial        % Factroial中设置断点
Factab(3);              % 调用Factab
5  numbers = 1:n;
K>> dbcont              % 跟踪到下一个断点
5  if ( nn == 0 )
K>> dbstack             % 跟踪进入factorial函数
In /home/aw/BOOK/factorial.m at line 5
In /home/aw/BOOK/Factab.m at line 8
K>> who                 % 显示当前值
```

Your variables are:

```
nn      p
```

```
K>> dbup                % 切换到调用工作区
In workspace belonging to /home/aw/BOOK/Factab.m.
K>> who                 % 显示当前变量
```

Your variables are:

```
Tab      facts      i      n      numbers
```

```
K>> dbdown              % 返回到factorial工作区
In workspace belonging to /home/aw/BOOK/factorial.m.
K>> dbquit              % 退出调试
```

要调试命令文件，必须使用命令keyboard。MATLAB的特定调试命令只能用在调试函数文件中。

为了编写有效的程序，需要一种工具来计算哪一部分程序所需时间最多。profile命令就是这样的工具：

命令集122 M文件的计时

profile choice M文件的计时命令。参数choice可以为下面情况之一：

filename	给M文件filename计时；在同一时间里只能对一个文件计时。
----------	---------------------------------

	<code>on, off</code>	打开或关闭指定M文件的计时器。
	<code>reset</code>	清除程序评述器的计时数据。
	<code>report</code>	显示计时报告。
	<code>report n</code>	显示所花时间最多的 n 行程序。
	<code>report</code>	显示至少使用部分时间的程序行。
	<code>frac</code>	时间的 $frac$, $frac$ 的值必须在区间 $[0, 1]$ 中。
	<code>done</code>	结束定时。
<code>info=profile</code>		返回结构 <code>info</code> , 用于计时器数据的图形显示。它包括下面字段:
	<code>info.file</code>	存放文件名, 包含被计时的 M文件的完全路径。
	<code>info.function</code>	包含函数名。
	<code>info.interval</code>	包含计时区间。
	<code>info.count</code>	包含带有计时数据的向量。
	<code>info.state</code>	包括计时工具的状态: on 或 off。
<code>profsumm choice</code>		创建M文件的评述摘要。choice的有效值为:
	<code>n</code>	显示程序中使用时间最多的 n 行程序。
	<code>frac</code>	显示程序中至少使用时间 $frac$ 的程序行, $frac$ 的值必须在0和1之间。
	<code>str</code>	如果程序行中有字符串 <code>str</code> , 则报告。
<code>profsumm</code>		命令也可以使用保存在结构 <code>info</code> 中的信息。

例12.21

同时对两个不同的程序进行计时, 而这两个程序都完成相同的任务, 通过计时来判断哪个程序将效率更高。第一个程序在 `particle.m` 中, 如下:

```
% 随机路径。一个质点从原点开始, 每一步在8个方向任意地走半个单位

%disp('Give the number of steps') % 步数
%n = input('>>>');
n = 500;

x = cumsum(rand(n,1)-0.5); % 任取x值
y = cumsum(rand(n,1)-0.5); % 任取y值

clf; % 清除图形窗口
plot(x,y); % 画出路径
hold on; % 保持当前图形
plot(x(1),y(1),'o',x(n),y(n),'o'); % 标出start/finsh
axis = axis; % 取min和max
scale = axis(2) - axis(1); % 计算刻度值

text(x(1)+scale/30,y(1),'Start'); % 在Start和Finish
```



```
text(x(n)+scale/30,y(n),'Finish'); % 右边写出文本
```

```
hold off; % 删除图形
```

```
xlabel('x'); ylabel('y');
title('Random walk')
```

另一个程序在文件particleBad.m中：

% 随机路径。一个质点从原点开始，每一步在8个方向任意地走半个单位

```
%disp('Give the number of steps') % 步数
```

```
%n = input('>>>');
```

```
n = 500;
```

```
%x = cumsum(rand(n,1)-0.5); % 任取x值
```

```
%y = cumsum(rand(n,1)-0.5); % 任取y值
```

```
x(1) = rand(1,1)-0.5;
```

```
y(1) = rand(1,1)-0.5;
```

```
for i = 2:n
```

```
    x(i) = rand(1,1)-0.5 + x(i-1);
```

```
    y(i) = rand(1,1)-0.5 + y(i-1);
```

```
end
```

```
clf; % 清除图形窗口
```

```
plot(x,y); % 画出路经
```

```
hold on; % 保持当前图形
```

```
plot(x(1),y(1),'o',x(n),y(n),'o'); % 标出start/finish
```

```
axis = axis; % 取min和max
```

```
scale = axis(2) - axis(1); % 计算刻度值
```

```
text(x(1)+scale/30,y(1),'Start'); % 在Start和Finish
```

```
text(x(n)+scale/30,y(n),'Finish'); % 右边写出文本
```

```
hold off; % 删除图形
```

```
xlabel('x'); ylabel('y');
```

```
title('Random walk')
```

给出下面的命令：

```
profile particle; particle; profile report; profile off;
```

```
profile particleBad; particleBad; profile report; profile off;
```

结果为：

```
Total time in "particle.m": 0.22 seconds
```

```
100% of the total time was spent on lines:
```

```
[13 25 12 26 17 15]
```

```
11:
```

```
0.03s, 14% 12: clf;
```

```

% 清除图形窗口
0.11s, 50% 13: plot(x,y);
% 画出路径
14: hold on;
% 保持当前图形
0.01s, 5% 15: plot(x(1),y(1),'o',x(n),y(n),'o');
% 标出start/finish
16:
0.02s, 9% 17: axis = axis;
% 取min和max
18: scale = axis(2) - axis(1);
% 计算刻度值
24:
0.03s, 14% 25: xlabel('x'); ylabel('y');
0.02s, 9% 26: title('Random walk')

```

Total time in "particleBad.m": 0.57 seconds

98% of the total time was spent on lines:

[16 15 21 20 33 34 31 25 23 22]

```

14: for i = 2:n
0.15s, 26% 15: x(i) = rand(1,1)-0.5 + x(i-1);
0.18s, 32% 16: y(i) = rand(1,1)-0.5 + y(i-1);
17: end

19:
0.04s, 7% 20: clf;
% 清除图形窗口
0.10s, 18% 21: plot(x,y);
% 画出路径
0.01s, 2% 22: hold on;
% 保持当前图形
0.01s, 2% 23: plot(x(1),y(1),'o',x(n),y(n),'o');
% 标出start/finish
24:
0.01s, 2% 25: axis = axis;
% 取min和max
26: scale = axis(2) - axis(1);
% 计算刻度值
30:
0.01s, 2% 31: hold off;
% 删除图形
32:
0.03s, 5% 33: xlabel('x'); ylabel('y');
0.02s, 4% 34: title('Random walk')

```

这样就能知道程序所使用的时间，并且知道哪一行程序所花时间最多。如果要立即画出最新的profile运行结果，可以输入命令：

```
t = profile
```

```
t =  
    file: '/home/matlab/VER5/kapitel12/particleBad.m'  
interval: 0.0100  
count: [33x1 double]  
state: 'off'
```

```
pareto(t.count)
```

在6.5节和13.1节中介绍了命令pareto。图12-6中的x轴上的棒形图为行程序运行所需的时间。y轴的左边是百分之一秒为单位的运行时间，右边是其所占全部执行时间的百分比。图上的线条表示总运行时间。

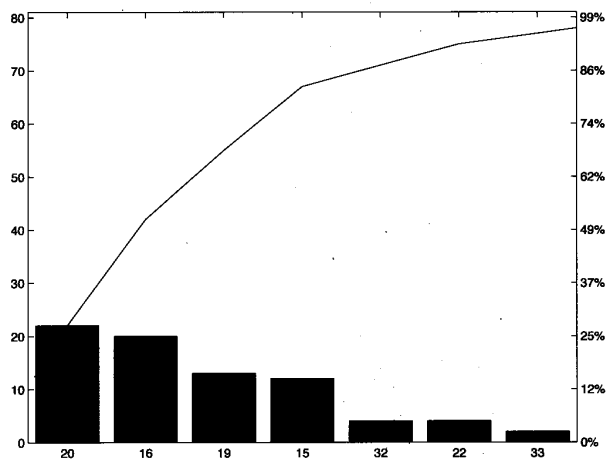


图12-6 particleBad.m计时的图形表示