

第3章 Matlab程序设计

Matlab程序设计既有传统高级语言的特征，又有自己独特的优点，在Matlab程序设计时，充分利用Matlab数据结构的特点，可以使程序结构简单、编程效率高。本章介绍有关Matlab程序控制结构以及程序设计的基本方法。

- Matlab在执行命令时，一种交互的命令执行方式，在命令窗口逐条输入指令，执行时Matlab逐条解释执行。这种方式虽然简单、直观，但速度慢，执行过程不保留，当某些操作需要反复进行时，更使人感到不便。
- 另一种M文件的程序执行方式，是将有关命令变成程序存储在一个文件中（M文件）当需要运行时，直接调用运行，运行时Matlab就自动依次执行该文件中的命令。

3.1 M文件

通常，M文件可分为2大类

$$\begin{cases} (1) \text{ M脚本文件} \\ (2) \text{ M函数文件} \end{cases}$$

这两种Matlab程序代码所编写的文件通常都是以“**.m**”为扩展名，因此都统称为M文件。M函数文件是Matlab的主体。

Matlab本身的一系列工具箱的内部函数就是Matlab的开发者设计的一些M函数，提供给我们使用。

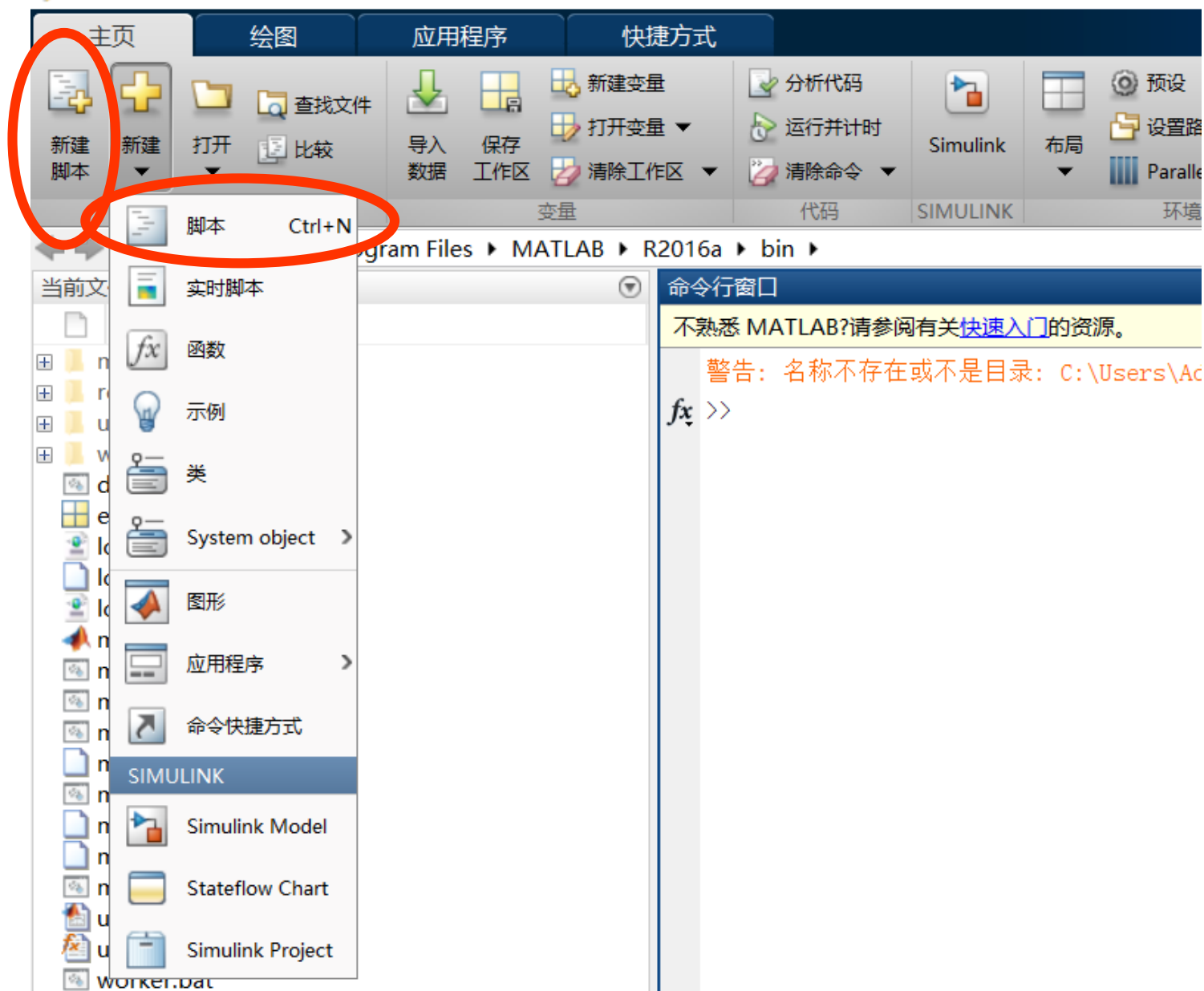
我们也可以根据我们的需要，开发设计我们自己需要M函数

3.1.1 M文件的建立与打开

1) 建立新的M文件

为建立新的M文件，启动MATLAB文本编辑器有3种方法：

- (1) 菜单操作。从MATLAB主窗口的File菜单中选择**New**菜单项，再选择M-file命令，屏幕上将出现MATLAB 文本编辑器窗口。
- (2) 命令操作。在MATLAB命令窗口输入命令**edit**，启动MATLAB文本编辑器后，输入M文件的内容并存盘。
- (3) 命令按钮操作。单击MATLAB主窗口工具栏上的New M-File命令按钮，启动MATLAB文本编辑器后，输入M文件的内容并存盘。



2) 打开已有的M文件

打开已有的M文件，也有3种方法：

- (1) 菜单操作。从MATLAB主窗口的File菜单中选择Open命令，则屏幕出现Open对话框，在Open对话框中选中所需打开的M文件。在文档窗口可以对打开的M文件进行编辑修改，编辑完成后，将M文件存盘。
- (2) 命令操作。在MATLAB命令窗口输入命令：
`edit 文件名`，则打开指定的M文件。
- (3) 命令按钮操作。单击MATLAB主窗口工具栏上的Open File命令按钮，再从弹出的对话框中选择所需打开的M文件。



例：用matlab编写一M文件和M函数绘制
以下函数图形

$$y(x) = \begin{cases} \sin x & x \leq 0 \\ x & 0 < x \leq 3 \\ -x + 6 & x > 3 \end{cases}$$

在区间 $[-6, 6]$ 中的图形

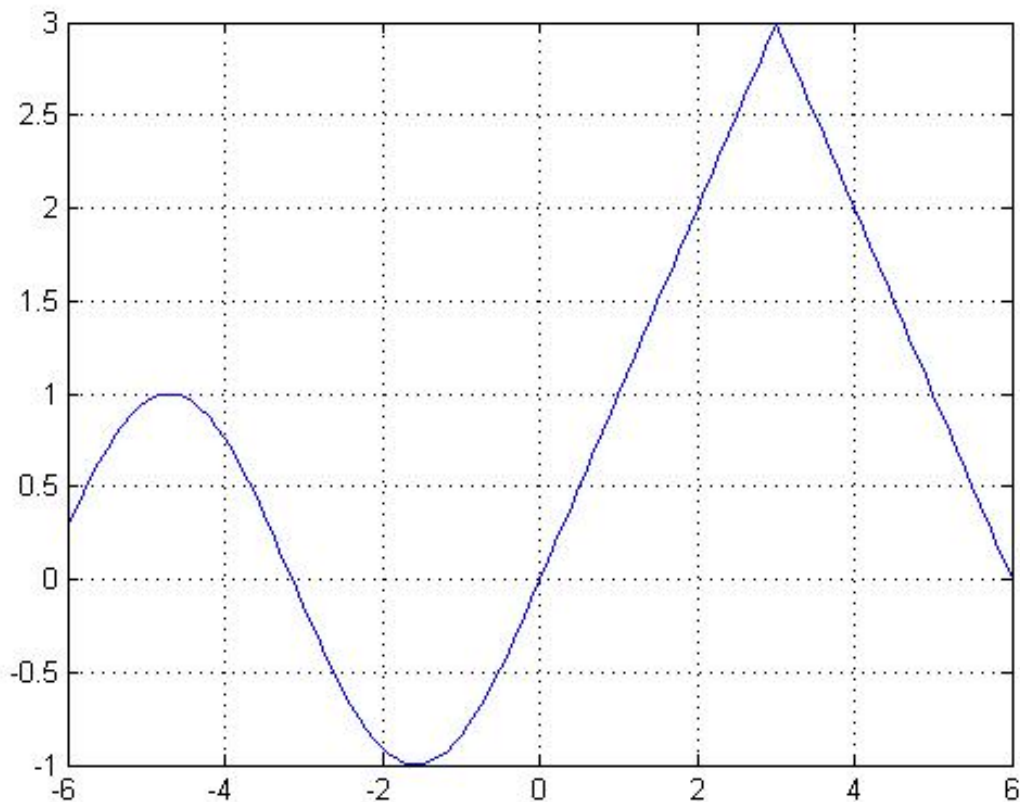
3.1.2 用M脚本文件形式 编写程序。

$$y(x) = \begin{cases} \sin x & x \leq 0 \\ x & 0 < x \leq 3 \\ -x + 6 & x > 3 \end{cases}$$

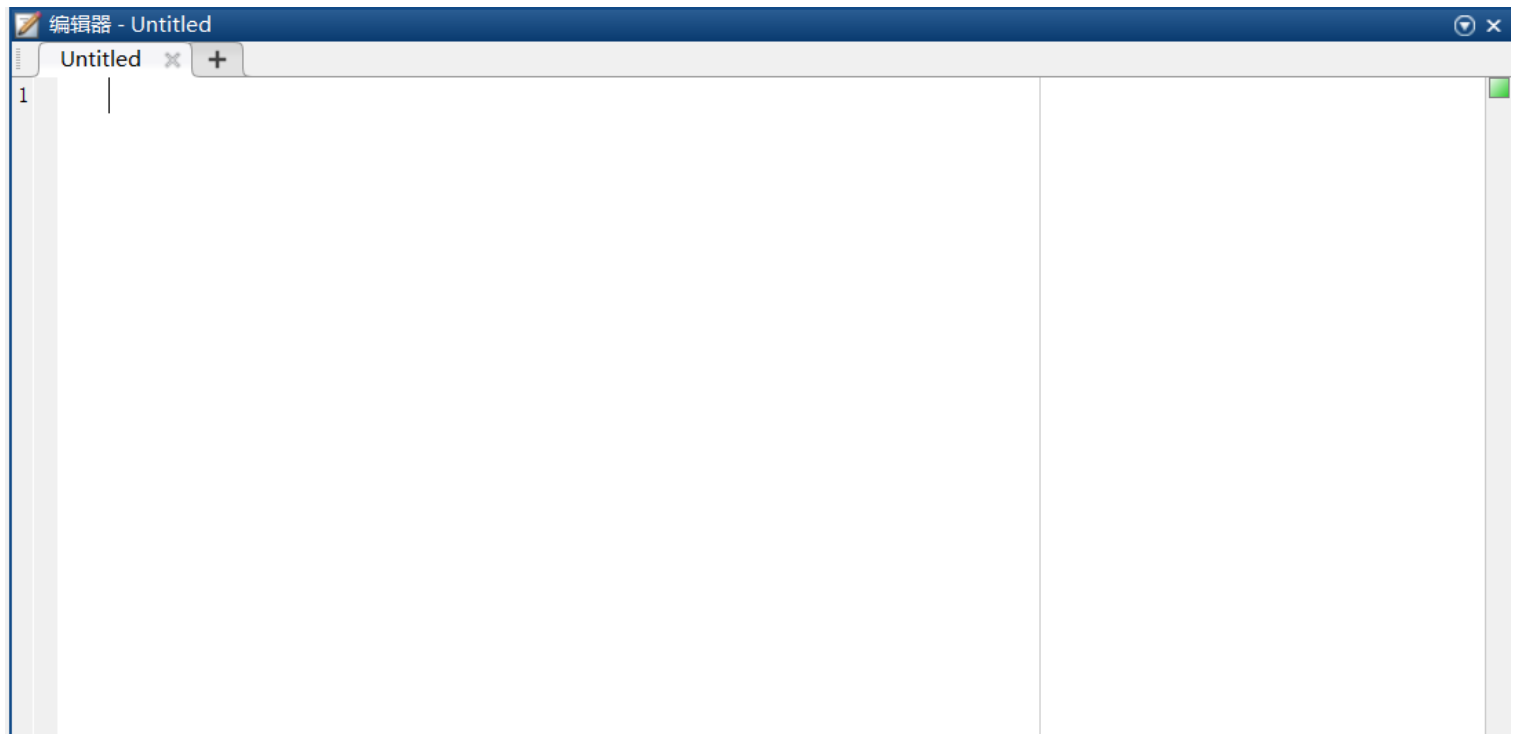
```
x=[-6 : 0.1 : 6]; n=length(x);  
 for m=1:n;           %循环结构  
     if x(m)<=0           %选择结构  
        y(m)=sin(x(m));  
    elseif x(m)<=3  
        y(m)=x(m);  
    else y(m)=-x(m)+6;  
    end  
end  
plot(x,y); grid on
```


用数组命令实现

```
x=-6 : 0.1 : 6;  
y=(x<=0).*sin(x)+(x>0&x<=3).*x+(x>3).*(-x+6);  
plot(x,y), grid on
```



Matlab的脚本文件（**script File**）比较简单，当需要在命令窗口运行大量的命令时，直接从命令窗口输入比较麻烦，可以打开**M**文件编辑器，将这组命令存放在脚本文件中，运行时只要输入脚本文件名，**Matlab**就会自动执行该文件。



3.1.3 M函数的形式编写程序

```
function y=demofile(x)
leng=length(x);
for m=1 : leng
    if x(m)<=0
        y(m)=sin(x(m));
    elseif x(m)<=3
        y(m)=x(m);
    else y(m)=-x(m)+6
    end
end
```

编写好该程序，保存时自动以demofile为函数名存入内存。当需要运行该程序时在命令窗口键入

```
function y=demofile1(x)
y=(x<=0).*sin(x)+(x>=0&x<=3).*x+(x>3).*(-x+6);
plot(x,y),grid on
```

```
x=-6 : 0.1 : 6;  
y=demofile(x);      %自动调用demofile.m  
plot(x,y);grid on
```

从例题中我们看到了，M函数和M脚本的不同，这里主要介绍M函数的基本格式

M函数文件基本格式：

函数声明行

H1行（用%开头的注释行）

在线帮助文本（用%开头）

编写和修改记录（用%开头）

函数体

例： 用M脚本文件和M函数文件，编写二阶系统时间响应曲线

$$y = 1 - \frac{1}{\sqrt{1 - \xi^2}} e^{-\xi t} \sin(\sqrt{1 - \xi^2} t + \arccos \xi)$$

① M脚本文件

%M脚本文件程序

%Ex0301 二阶系统时域曲线

%画阻尼系数为0.3的曲线

t=0:0.1:20;

y1=1-1/sqrt(1-0.3^2)*exp(-0.3*t).*sin(sqrt(1-.3^2)*t+acos(0.3));

plot(t,y1,'r')

②M函数文件

```
function y=Ex0302(zeta)
```

```
% 二阶系统时域曲线
```

```
%Ex0302
```

```
%zeta 阻尼系数
```

```
%y 时域响应
```

```
%
```

```
%copyright 2015-3-16
```

```
t=0:0.1:20;
```

```
y1=1-1/sqrt(1-zeta^2)*exp...
```

```
(-zeta*t).*sin(sqrt(1-zeta^2)*t+acos(zeta));
```

```
plot(t,y1,'r')
```

第一行一定为
函数声明行

H1行（用%开
头的注释行）

帮助文本

编写和修改记录
（用%开头）

函数体

M文件的一般结构

- 从结构上看，脚本文件只是比函数文件少一个“函数申明行”。
- 典型M函数文件的结构：
 - 函数申明（定义）行 (**Function declaration line**):
 - H1行 (**The first help text line**):
 - 在线帮助文本区 (**Help text**):
 - 编写和修改记录 (**Revised recorder**):
 - 函数体 (**Function body**):

函数申明（定义）行 (Function declaration line)

位于函数文件的首行，以MATLAB关键字**function**开头，函数名以及函数的输入输出变量都在这一行定义。

H1行(The first help text line):

紧随函数申明行之后以%开头的第一注释行。
按MATLAB自身文件的规则，H1行包含：大写的函数文件名；运用关键词简要描述的函数功能。该H1行供lookfor关键词查询和help在线帮助使用。

在线帮助文本(Help text)区：

H1行及其之后的连续以%开头的**所有注释行**构成整个在线帮助文本。它通常包括：函数输入输出变量的含义，调用格式说明。

编写和修改记录

与在线帮助文本区相隔一个空行，也以%开头，标志编写及修改该M文件的作者和日期、版本记录。它用于软件档案管理。

函数体(Function body):

这部分内容由实现该M函数文件功能的指令组成：接收输入变量，进行程序流控制，得到输出变量。

为清晰起见，它与前面的注释行以空行相隔。

为阅读、理解方便，也在函数体中配置适当的空行和注释。

可在命令窗口输入help和lookfor命令查看帮助信息

```
>> help Ex0302
```

二阶系统时域曲线

Ex0302

zeta 阻尼系数

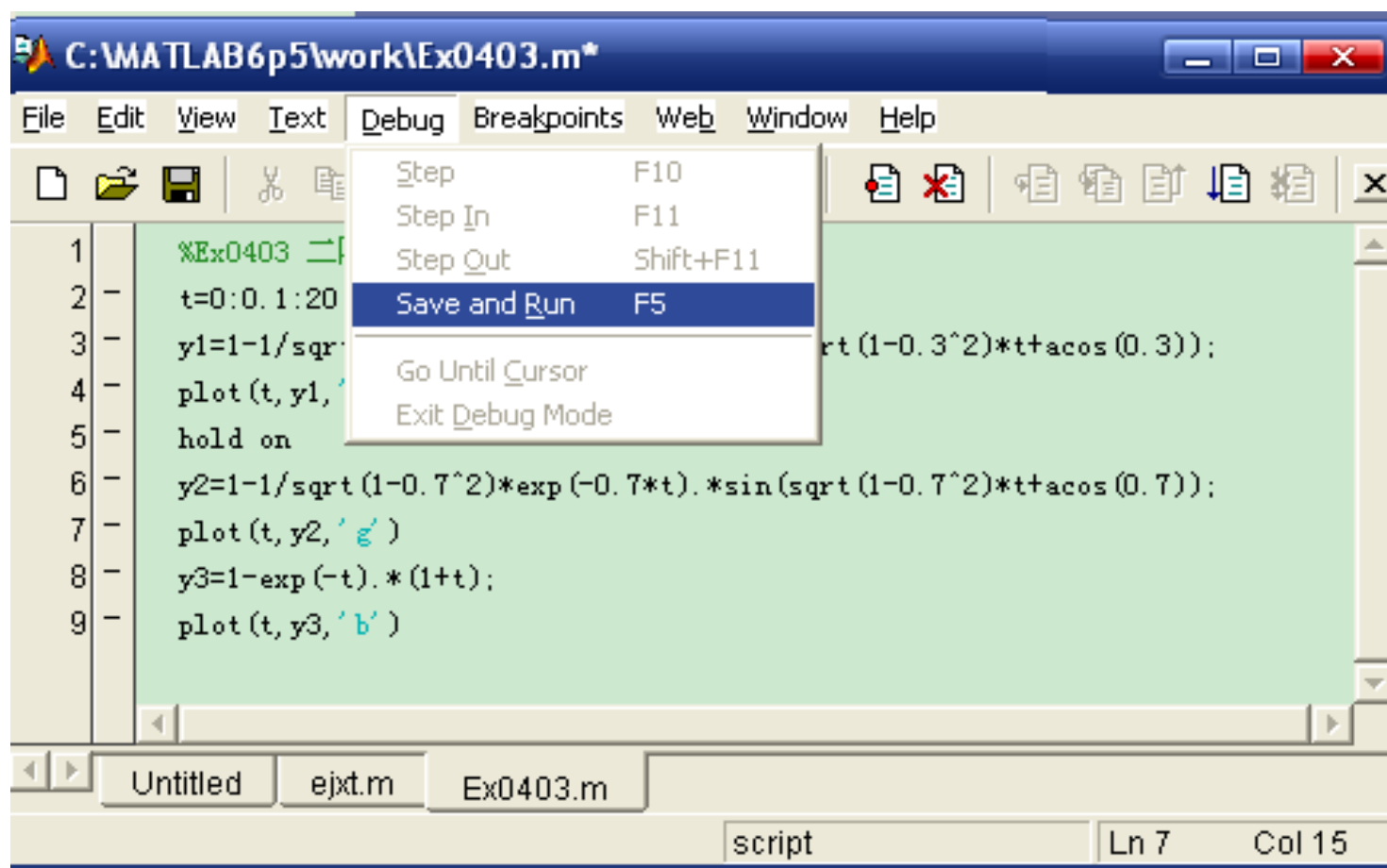
y 时域响应

copyright 2015-3-16

help命令显示M文件的第一个连续注释块

```
>> lookfor '阶系统时域曲线'  
Ex0302.m: % 二阶系统时域曲线
```

lookfor命令显示第一行注释，**lookfor**命令的查找必须是Matlab搜索路径上的文件



选择M文件编辑器选单“Debug/Save and Run”，
就可以观察运行结果

3.1.4 两种文件的区别

1) M脚本（命令）文件（Script）

- 是若干命令和函数的集合，执行特定的功能
- 不存在参数的输入和输出
- 在MATLAB环境下，脚本可以直接访问整个工作空间的变量；而且由脚本建立的变量在脚本文件执行完毕后仍保持在工作空间中，直到clear命令清除为止。

2) M函数文件 (M-Function)

- 函数不但可以接受输入参数，还可以输出参数
- MATLAB允许使用比“标称数目”较少的输入输出变量，实现对函数的调用。
- 存储函数的M文件的文件名必须与函数名一致
- 函数只能访问本身空间的变量
- 每当函数文件运行，MATLAB就会专门为其开辟一个临时函数工作空间(Function workspace)。所有中间变量都存放在函数工作空间中。当执行完文件最后一条指令或遇到return时，就结束该函数文件的运行，同时该临时函数空间及其所有的中间变量就立即被清除。

- 函数空间随具体M函数文件的被调用而产生，随调用结束而删除。函数空间是相对独立的、临时的。在MATLAB整个运行期间，可以产生任意多个临时函数空间。
- 假如在函数文件中，发生对某脚本文件的调用，那么该脚本文件运行产生的所有变量都存放于该函数空间之中，而不是存放在基本空间。

3.2 程序控制结构

程序 = 算法 + 数据

算法 = 逻辑 + 控制

3.2 程序控制结构

Matlab的程序控制结构有3种，任何复杂的程序都可以有着3种基本结构构成：

① 顺序结构

② 选择结构

③ 循环结构

3.2.1 顺序结构

顺序结构是指按照程序中语句的排列顺序依次执行，直到程序的最后一个语句。这也是最简单的一种程序结构。一般涉及数据的输入、数据的计算或处理、数据的输出等内容。

1) 数据的输入

从键盘输入数据，可使用input函数来实现，其格式：

```
A=input(提示信息, 选项);
```

其中：提示信息---为字符串，用于提示用户
输入什么样的数据

```
A=input('输入A矩阵:')           %提示输入A矩阵
```

2) 数据的输出

Matlab提供的命令窗口输出函数主要有disp函数，其调用格式为：

disp(输出项)

其中：输出项——既可以是字符串、也可以是矩阵

例如： `A='Hello,Tom';`
`disp(A)`

输出结果： Hello,Tom

例如： `A=[1 2 3;4 5 6;7 8 9];`
`disp(A)`

输出结果：

1	2	3
4	5	6
7	8	9

例： 求解一元二次方程 $ax^2 + bx + c = 0$

```
a=input('a=?');  
b=input('b=?');  
c=input('c=?');  
d=b^2-a*4*c;  
x=[(-b+sqrt(d))/(2*a),(-b-sqrt(d))/(2*a)];  
disp(['x1=',num2str(x(1)), 'x2=',num2str(x(2))])
```

输出结果：

```
a=?4  
b=?78  
c=?54  
x1=-0.7188  
x2=-18.7812
```

```
a=?23  
b=?-6  
c=?51  
x1=0.13043+1.4834i  
x2=0.13043-1.4834i
```


练习题1：

输入x, y的值，并将它们的值互换后输出。

```
function f=change(x, y)
```

```
x=input('Input x:');
```

```
y=input('Input y:');
```

```
a=x; x=y; y=a;
```

```
x, y
```

3.2.2 选择结构

选择结构是根据给定条件成立或不成立，分别执行不同的语句。Matlab中用于选择结构的语句有

1) if...else...end

2) switch...case

3) try...catch

1) if...else...end 语句

Matlab中，if语句有3种格式

① 单分支if语句

```
格式： if 条件
           语句组
        end
```

② 双分支if语句

```
格式： if 条件
           语句组1
        else
           语句组2
        end
```

③ 多分支if语句

格式：if 条件1

语句组1

elseif 条件2

语句组2

.....

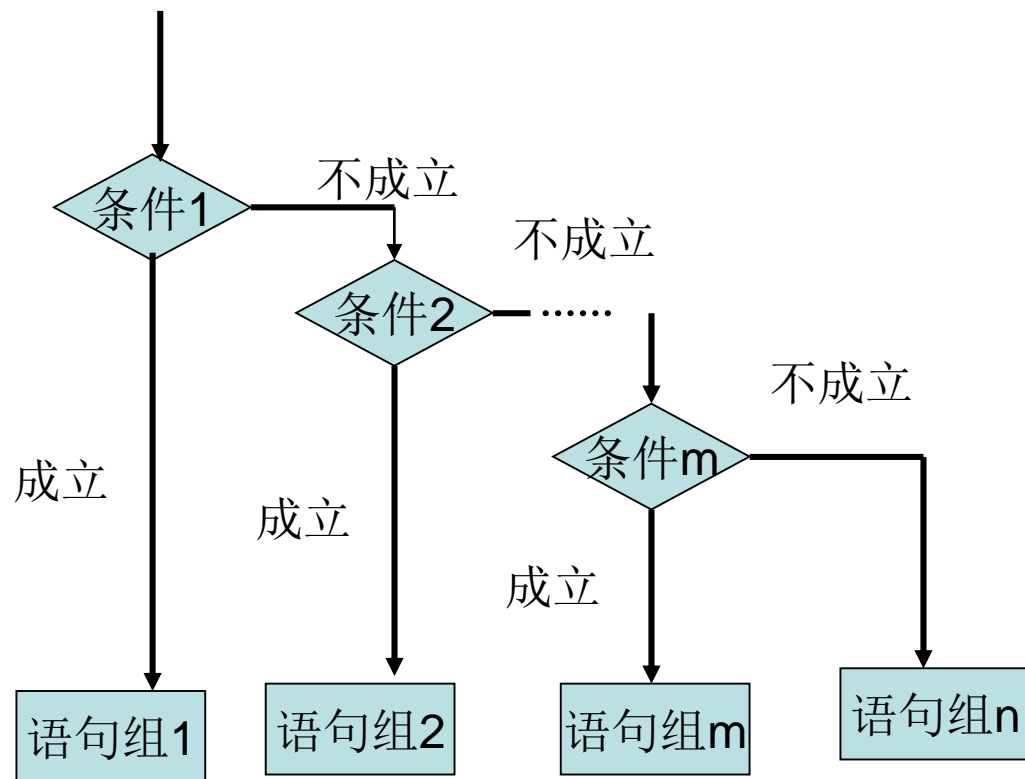
elseif 条件m

语句组m

else

语句组n

end



多分支if语句的执行过程结构框图

例：计算分段函数

$$y = \begin{cases} \cos(x+1) + \sqrt{x^2 + 1} & x = 10 \\ x\sqrt{x + \sqrt{x}} & x \neq 10 \end{cases}$$

运用单分支if语句编程

```
x=input('请输入的值');  
y=cos(x+1)+sqrt(x*x+1);  
{ if x~=10  
    y=x*sqrt(x+sqrt(x));  
  end  
y
```

运用双分支if语句编程

```
x=input('请输入的值');  
if x~=10  
    y=x*sqrt(x+sqrt(x));  
else  
    y=cos(x+1)+sqrt(x*x+1);  
end  
y
```

```
x=input('请输入的值');  
if x==10  
    y=cos(x+1)+sqrt(x*x+1);  
else  
    y=x*sqrt(x+sqrt(x));  
end  
y
```

例：已知符号函数 $y = \operatorname{sgn} x = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases}$

使用if语句判断当给定变量x的值时，对应的函数值y

```
x=input('enter x:');  
if x>0  
    y=1;  
elseif x==0  
    y=0;  
else  
    y=-1;  
end  
disp(y)
```

例：输入一个字符，若为大写字母，则输出其对应的小写字母；若为小写字母，则输出其对应的大写字母；若为数字字符则输出其对应的数值，若为其他字符则原样输出。

```
c=input('请输入一个字符','s');
```

```
if c>='A' & c<='Z'
```

```
    disp(setstr(abs(c)+abs('a')-abs('A')));
```

% abs为字符到ASCII转换函数

```
elseif c>='a' & c<='z'
```

```
    disp(setstr(abs(c)-abs('a')+abs('A')));
```

```
elseif c>='0' & c<='9'
```

```
    disp(abs(c)-abs('0'));
```

```
else
```

```
    disp(c);
```

```
end
```

提示：字母、数字都是按照顺序排列的

练习题2:

编写一个函数，对于输入的2个数a、b，使他们从小到大排序输出

```
function f=mm3(a,b)
```

```
%This file is devoted to demonstrate the use of 'if'
```

```
%The function of this file is to convert the value of a and b
```

```
if a>b
```

```
    t=a;
```

```
    a=b;
```

```
    b=t;
```

```
end
```

```
a
```

```
b
```

练习题3:

判断学生成绩的等级，90分以上为优，80～90为良，70～80为中，60～70为及格，60分以下为不及格。

```
score=98;  
s1=fix(score/10);    %取十位数  
if s1>=9  
    s='优'  
elseif s1>=8  
    s='良'  
elseif s1>= 7  
    s='中'  
elseif s1>= 6  
    s='及格'  
else  
    s='不及格'  
end
```

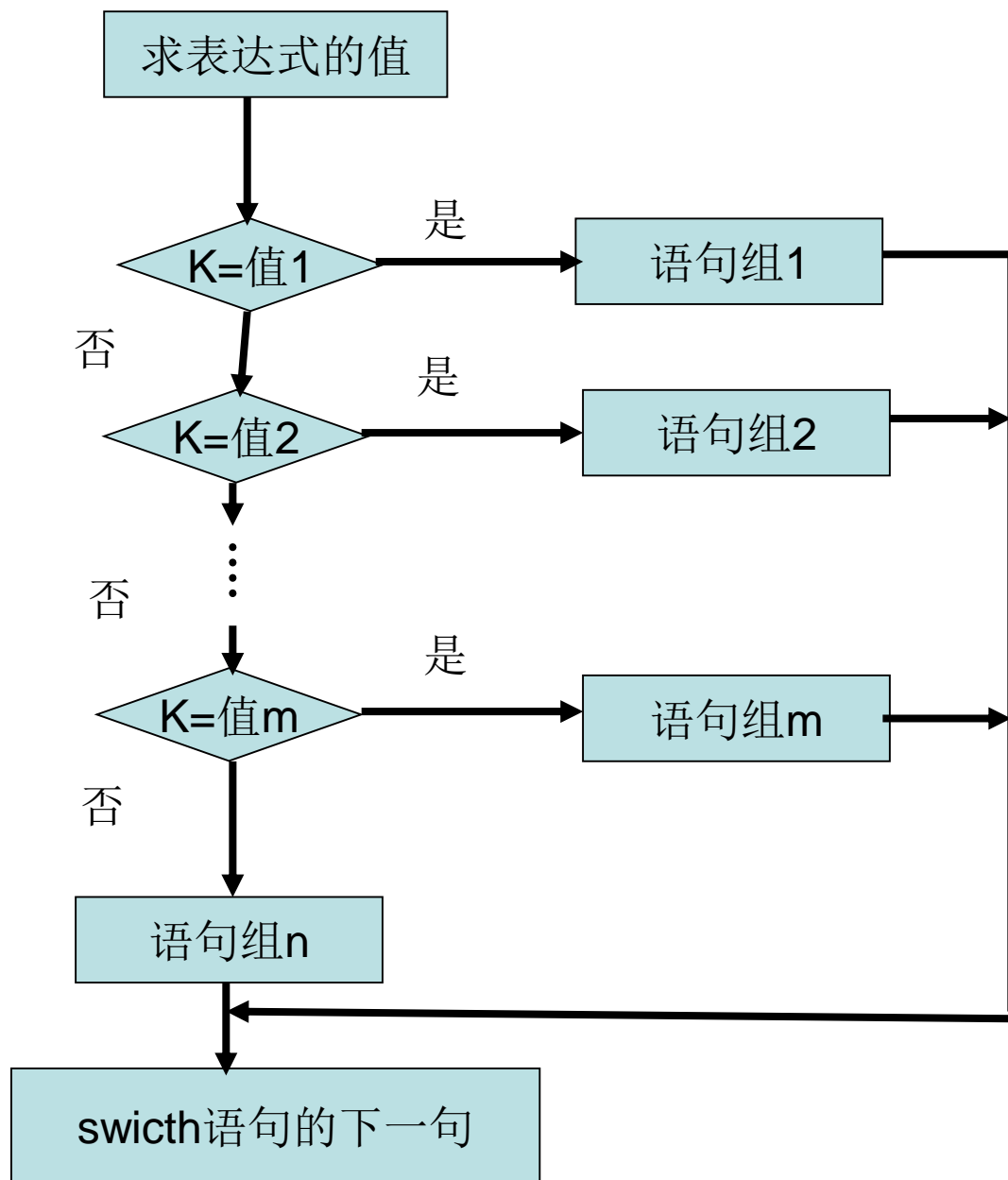
2) switch...case语句

switch语句是根据表达式的取值不同，
分别执行不同的语句，其语句格式：

格式：

```
switch 表达式
case 检测值1
    语句组1
case 检测值2
    语句组2
.....
case 检测值m
    语句组m
otherwise
    语句组n
end
```

Switch 语句的执行过程



- **switch-case**结构的句法格式保证了至少有一组指令将会被执行
- **switch**指令之后的表达式**value**应为一个标量或字符串：
 - 当表达式为标量时，比较命令为表达式==检测值
 - 当表达式为字符串时，调用字符串比较函数来进行**strcmp(表达式, 检测值)**
- **case**指令之后的检测值可以是标量、字符串、元胞数组
- 检测值是元胞数组时，会将**switch**后面的表达式值与元胞数组的所有元素进行比较，只要其中有某个元素值与表达式值相等，就执行与该检测值对应的语句命令组

例：使用switch结构判断学生成绩的等级，90分以上为优，80~90为良，70~80为中，60~70为及格，60分以下为不及格。

```
score=98;
```

```
s1=fix(score/10);    %取十位数
```

```
switch s1
```

```
    case {9,10}
```

```
        s='优'
```

```
    case 8
```

```
        s='良'
```

```
    case 7
```

```
        s='中'
```

```
    case 6
```

```
        s='及格'
```

```
    otherwise
```

```
        s='不及格'
```

```
end
```

S = 优

case后：单元数组格式

例： 某商场对顾客所购买的商品实行打折销售，标准如下（商品价格用price）

$price < 200$	没有折扣
$200 \leq price < 500$	3% 折扣
$500 \leq price < 1000$	5% 折扣
$1000 \leq price < 2500$	8% 折扣
$2500 \leq price < 5000$	10% 折扣
$5000 \leq price$	14% 折扣

price=input('请输入商品价格')

switch fix(price/100)

% 向零取整

case {0,1}

%商品价格小于200

rate=0;

case {2,3,4}

%商品价格大于200小于500

rate=3/100;

case num2cell(5:9)

%商品价格大于500小于1000

rate=5/100;

case num2cell(10:24)

%商品价格大于1000小于2500

rate=8/100;

case num2cell(25:49)

%商品价格大于2500小于5000

rate=10/100;

otherwise

%商品价格大于等于5000

rate=14/100;

end

price=price*(1-rate)

%输出商品实际价格

if语句和switch语句的比较

if语句	switch语句
相对复杂，尤其是嵌套使用时	结构简单，可读性强，容易理解
要调用strcmp函数比较不同长度的字符串	可直接比较不同长度的字符串
可检测相等和不相等	只能检测相等

3) try...catch语句

try语句是一种试探性执行语句，其语句格式：

```
try
    语句组1
catch
    语句组2
end
```

try语句先试探执行语句组1，如果语句组1在执行过程中出现错误，则将保留的最后出错信息lasterr变量，并转去执行语句2。

例如 矩阵乘法要求两矩阵的位数相等，否则出错

Matlab程序

```
A=[1 2 3;4 5 6];  
B=[7 8 9;10 11 12];
```

```
try  
    C=A*B;  
catch  
    C=A.*B;  
end  
C  
lasterr
```

输出结果

C =

7	16	27
40	55	72

ans =

Error using ==> *
Inner matrix dimensions
must agree.

3.2.3 循环结构

循环是指按照给定的条件，重复执行指定的语句，这是十分重要的一种程序结构。

Matlab提供了两种实现循环的结构语句

①for语句； ②while语句

1) For语句

格式： for 循环变量=循环初值：步长：终值
循环体语句
end

分别求循环的初值、步长、终值

将循环的初值赋给循环变量 i

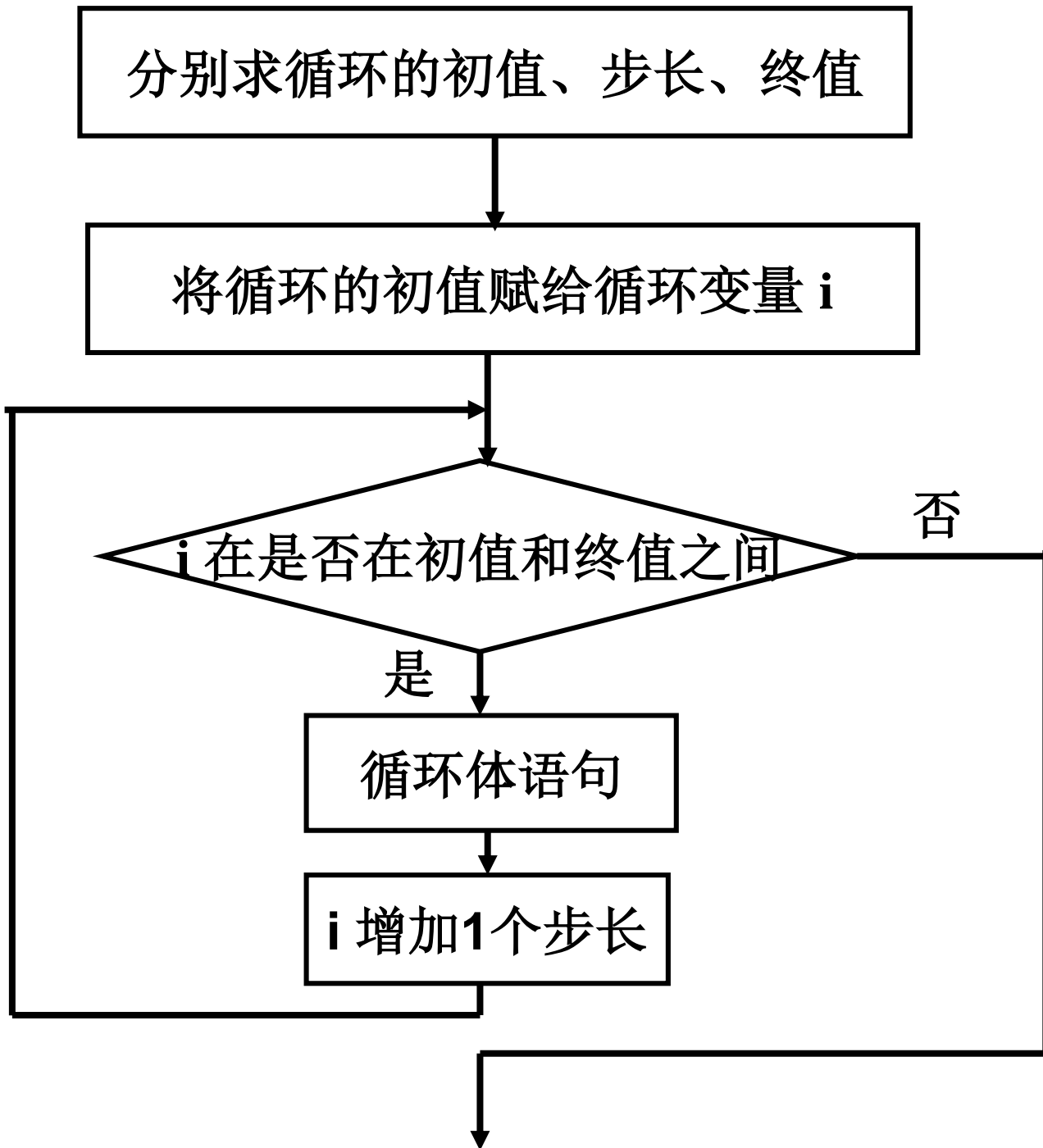
i 是否在初值和终值之间

否

是

循环体语句

i 增加1个步长



例：求 $1+3+5+\dots+99$ 的值

```
% 数组求和  
sm=0;  
for n=1:2:99;  
    sm=sm+n;  
end  
sm
```

```
%用Matlab函数求和  
x=1:2:99  
s=sum(x)
```

程序分析： 循环变量为 n ， n 对应为向量 $1:2:99$ ，
循环次数为向量 $1:2:99$ 的列数，每次循环 n 取
一个元素

例： 一个3位整数，其各位数字的立方和等于该数本身则称为水仙花数，求出100~999之间的全部水仙花数。

```
for m=100:999;
```

```
    m1=fix(m/100); %求m的百位数字，fix向零方向取整
```

```
    m2=rem(fix(m/10),10); %求m的十位数字, rem求余数
```

```
    m3=rem(m,10); %求m的个位数字
```

```
    if m==m1*m1*m1+m2*m2*m2+m3*m3*m3
```

```
        disp(m)
```

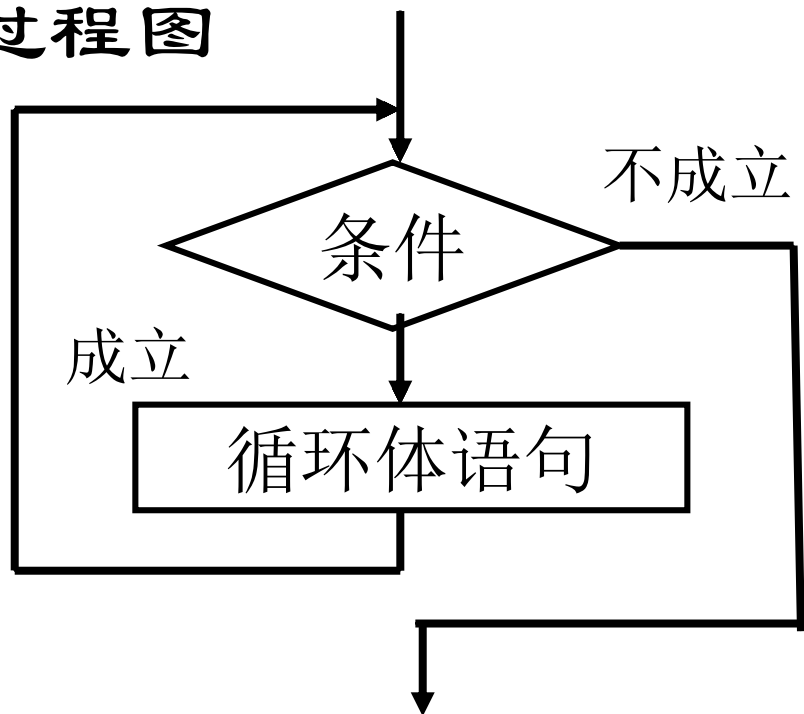
```
    end
```

```
end
```

2) while语句

格式： while （条件）
 循环体语句
 end

循环过程图



例： 利用while循环，求 $1+3+5+\dots+99$ 的值

% 利用while循环求 $1+3+5+\dots+99$ 的值

```
sm=0;
```

```
n=1;
```

```
while n<=99
```

```
    sm=sm+n
```

```
    n=n+2
```

```
end
```

```
sm
```

程序分析： 可见while...end循环的循环次数由表达式来决定，当 $n=101$ 时停止循环

例：从键盘输入若干个数，当输入0时结束运算，求这些数的平均值和它们的和

```
sm=0;
```

```
n=0;
```

```
x=input('Enter a number (end in 0):');
```

```
while(x~=0)
```

```
    sm=sm+x;
```

```
    n=n+1;
```

```
    x=input('Enter a number (end in 0):');
```

```
end
```

```
if (n>0)
```

```
    sm
```

```
    mean=sm/n
```

```
end
```

◆ 为了得到最大的速度，在For循环(While循环)被执行之前，应预先分配数组。

```
for n=1:10  
    x(n)=sin(n*pi/10);  
end
```

MATLAB每通过一次循环
要花费时间对x分配更多的
内存

```
x=zeros(1,10);  
for n=1:10  
    x(n)=sin(n*pi/10);  
end
```

只有x(n)的值需要改变

练习题4:

找出1~n之间，既能被5整除，又能被11整除的数

```
n=input('enter n:');
```

```
for x=1:n
```

```
    if rem(x, 5)==0 & rem(x, 11)==0
```

```
        x
```

```
    end
```

```
end
```

```
n=input('enter n:');
```

```
for x=11:11:n
```

```
    if rem(x, 5)==0
```

```
        x
```

```
    end
```

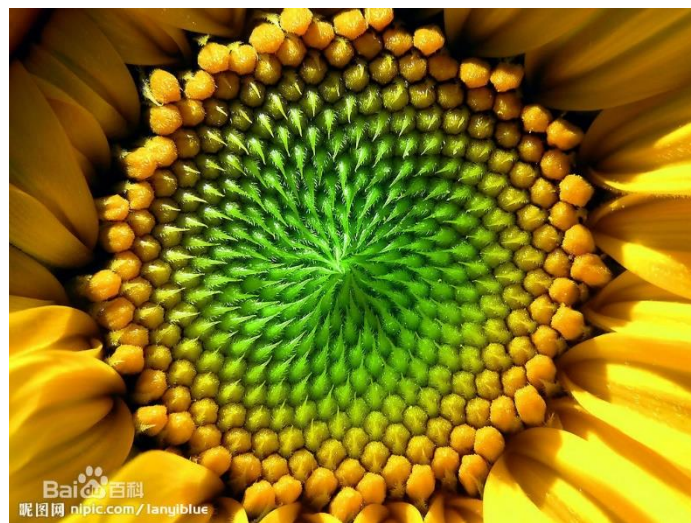
```
end
```

练习题5:

计算Fibonacci数列 $(1, 1, 2, 3, 5, 8, 13, 21, \dots)$,
即后1项为前2项的和, 当某一项大于1000时, 输出该数

斐波那契数列 (Fibonacci sequence) : 又称黄金分割数列、因数学家列昂纳多·斐波那契 (Leonardoda Fibonacci) 以兔子繁殖为例子而引入, 故又称为“兔子数列”, 以递推的方法定义:
 $F(1)=1, F(2)=1, F(n)=F(n-1)+F(n-2) \ (n \geq 3, n \in \mathbb{N}^*)$

兔子在出生两个月后, 就有繁殖能力, 一对兔子每个月能生出一对小兔子来。如果所有兔子都不死, 那么一年以后可以繁殖多少对兔子?



练习题5:

计算Fibonacci数列（1, 1, 2, 3, 5, 8, 13, 21, ...），
即后1项为前2项的和，当某一项大于1000时，输出该数

```
f1=1; f2=1; f=0      %数值初始化
while (f<=1000)      %设定循环停止条件
    f=f1+f2;
    f1=f2;            %变量赋值
    f2=f;
end
f
```

3.2.4 流程控制语句

在程序执行中，有一些可以控制程序流程的命令，下面包括：

- 1) break
- 2) continue
- 3) return
- 4) pause
- 5) keyboard

1) break语句

break语句用于终止**for**循环和**while**循环的执行。当遇到**break**时，则退出循环体，继续执行循环体外的下一个语句。在嵌套循环中，**break**往往存在于内层的循环中

例：用**if**与**break**命令结合，停止**while**循环，计算 $1+3+5+\cdots+99$ 的值，当和大于1000时终止计算

% 用break终止while循环

sm=0;

n=1;

while n<=99

if sm<1000

sm=sm+n

n=n+2

else

break

end

end

程序分析: while··end循环结构嵌套

if··else··end分支结构，当sm为1024时跳出while循环，终止运算

2) continue语句

continue命令用于结束本次**for**或**while**循环，与**break**命令不同的是**continue**指结束本次循环而继续进行下次循环

例：将**if**命令与**continue**命令结合，计算1~100中所有素数的和，判断是否为素数是将100以内的每个数都被 $2\sim\sqrt{n}$ 整除，不能被整除的就是素数

```
% 用continue终止for循环
sm=2;ss=0;y=2;
for n=3:100
    for m=2:fix(sqrt(n))
        if mod(n,m)==0
            ss=1; %能被整除就用ss=1表示
            break; %能被整除就跳出内循环
        else
            ss=0; %不能被整除就用ss=0表示
        end
    end
    if ss==1
        continue; %能被整除就跳出本次循环
    end
    sm=sm+n; %100以内的素数之和
end
sm
```

程序分析：

`fix(sqrt(n))` 是将 \sqrt{n} 取整；本程序为双重循环，两个for循环嵌套还嵌套一个if结构；当`mod(n, m)=0`时就用break跳出判断是否是素数的内循环，并继续用continue跳出求素数和的循环而继续下依次外循环

例：把100到150之间的能被7整除的整数输出

```
for n=100:150
    if rem(n,7)~=0 %rem为求余数函数，判断能否被7整除
        continue %把不能被7整除的数去掉，判断下一个数
    end
    n %输出能被7整除的数
end
```

程序分析：在for循环和while循环中，当出现continue语句时，则跳出循环体所有剩余的语句，继续下一个循环。在嵌套循环中，continue控制执行本循环中的下一次循环。

rem和mod的区别

不仔细区分的话，可把rem和mod都当作是求余数的命令。

```
>> mod(3,2)      ans = 1
```

```
>> rem(3,2)      ans = 1
```

通过帮助文件可知，这两个数的符号一致时的结果是一样的，但是当两个数的符号不一样时，就会出现不同了。

```
>> mod(3,-2)     ans = -1
```

```
>> rem(3,-2)     ans = 1
```

主要区别在：

`rem(x, y)`返回的是 $x - n * y$ ，当 $y \neq 0$ 时， $n = \text{fix}(x./y)$ ，

`mod(x, y)`返回的是 $x - n * y$ ，当 $y \neq 0$ 时， $n = \text{floor}(x./y)$

因此他们之间的区别主要在与fix与floor的区别：

fix：向最近的整数取整

floor：向负无穷取整。

3) **return**语句

用来终止被调用函数的运行，后面的程序代码将不再执行，直接返回到上一级调用函数。

```
function d = det(A)
```

```
    if isempty(A)
```

```
        d = 1;
```

```
        return
```

```
    else
```

```
        ...
```

```
    end
```

4) **pause语句**

其调用格式有：

- (1) **pause**: 暂停程序运行，按任意键继续；
- (2) **pause(n)**: 程序暂停运行n秒后继续；
- (3) **pause on/off**: 允许/禁止其后的程序暂停。

```
clear all;  
t=0:pi/20:2*pi;  
x=sin(t);  
figure;  
plot(t,x)  
xlabel('t');  
ylabel('x');  
hold on;  
for i=1:7  
    pause;  
    plot(t,sin(t+i/5));  
    hold on;  
end
```

画正弦曲线，利用**pause**暂停程序的运行，然后每按一次键，画一条正弦曲线

5) keyboard语句

程序运行时如果遇到keyboard函数，将停止文件的执行并将控制权交给键盘。通过在提示符前显示K来表示一种特殊状态。在M文件中使用该函数，对程序的调试和在程序运行中修改变量都很方便。

```
clear all;  
  
a=[1 2 3;2 3 4]  
  
b=[4 5 3;7 8 9]  
  
keyboard  
  
c=a+b
```

a=

1	2	3
4	5	6

b=

4	5	3
7	8	9

K>> a(1,2)=100

a=

1	100	3
4	5	6

K>> b(1,1)=100

b=

100	5	3
7	8	9

K>> return (或dbcont)

c=

101	105	6
11	13	15

控制程序流指令汇总

指令及使用格式	使用说明
v=input('message') v=input('message','s')	该指令执行时，“控制权”交给键盘；待输入结束，按下 Enter 键，“控制权”交还 MATLAB 。 message 是提示用的字符串。第一种格式用于键入数值、字符串、元胞数组等数据；第二种格式，不管键入什么，总是以字符串形式赋给变量 v
keyboard	遇到 keyboard 时，将“控制权”交给键盘，用户可从键盘输入各种 MATLAB 指令。仅当用户输入 return/dbcont 指令后，“控制权”才交还给程序。它与 input 的区别是：它允许输入任意多个 MATLAB 指令，而 input 只能输入赋给变量的值
break	break 指令可导致包含该指令的 while 、 for 循环终止；也可在 if-end , switch-case , try-catch 中导致中断
continue	跳过位于其后的循环中的其他指令，执行循环的下一个迭代

指令及使用格式	使用说明
pause pause(n)	第一种格式使程序暂停执行，等待用户按任意键继续；第二种格式使程序暂停n秒后，再继续执行
return	结束return指令所在函数的执行，而把控制转至主调函数或指令窗。否则，只有待整个被调函数执行完后，才会转出
error('message')	显示出错信息message，终止程序
lasterr	显示最新出错原因，并终止程序
lastwarn	显示MATLAB自动给出的最新警告，程序继续运行
warning('message')	显示警告信息message，程序继续运行

3.3 函数文件和参数传递

在Matlab中，使用函数可以把一个比较大的任务分解为多个较小的任务，使程序模块化，每个函数完成特定的功能。

3.3.1 主函数和子函数

MATLAB允许一个M函数文件包含多个函数代码。保存时所用的函数名与主函数定义名相同。外部程序只能对主函数进行调用。

- 第一个出现的函数称为主函数(Primary function)，
- 文件中的其他函数称为子函数(Subfunction)。

子函数的性质：

- 每个子函数的第一行是其自己的函数申明行。
- 在M函数文件内，主函数的位置不可改变，但子函数的排列次序可任意改变。
- 子函数只能被处于同一文件的主函数或其他子函数调用。
- 在M函数文件中，任何指令通过“名字”对函数进行调用时，子函数的优先级仅次于内装函数。
- 同一文件的主函数、子函数的工作空间都是彼此独立的。各函数间的信息，或通过输入输出变量传递，或通过全局变量传递，或通过跨空间指令传递。
- 查看子函数的注释用help mainfun/subfun的方式

例: 将绘制正弦函数曲线的函数作为子函数，编写绘制多条正弦曲线的程序

```
function Exsinplot( )
```

```
% Exsinplot 使用子函数调用绘制正弦曲线
```

```
omega = 1; Exsin(omega); %调用Exsin
```

```
hold on;
```

```
omega = 2; Exsin(omega); %调用Exsin
```

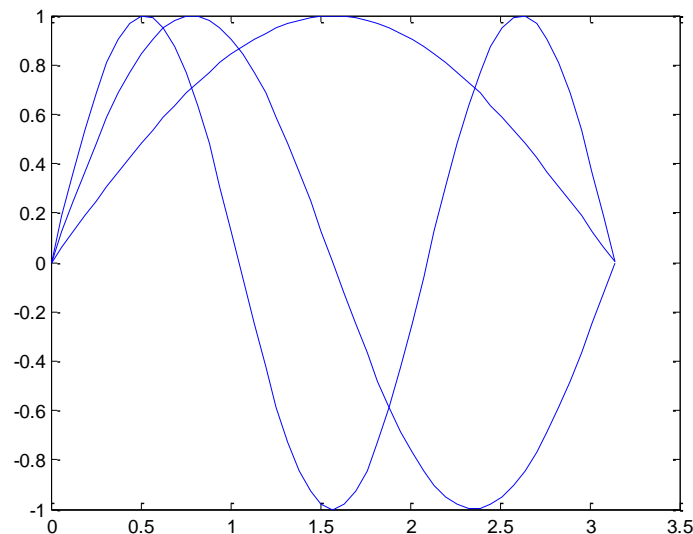
```
omega = 3; Exsin(omega); %调用Exsin
```

```
function y=Exsin(x) %子函数  $y=\sin(\omega t)$ 
```

```
t=0:pi/50:pi;
```

```
y=sin(x*t);
```

```
plot(t,y);
```



3.3.2 函数的输入输出参数

- Matlab函数调用过程实际上也是参数的传递过程。
- 函数A调用了函数B, 称A为“主调函数”, 而B为“被调用函数”。
- 函数通过输入参数接收数据, 经函数运算后由输出参数输出结果。

参数的传递

- 函数的参数传递是将主调函数中的变量值传给被调函数的输入参数。
 - (1) 函数参数传递的是数值；
 - (2) 参数的存储空间：被调函数的输入参数存放在函数的工作空间中，与matlab的工作空间是独立的，当调用结束时函数的工作空间被清除，输入参数就消失了。

在Matlab中有两个特殊的变量nargin和nargout, 函数的输入、输出个数可以通过nargin和nargout获得,

- nargin用于获得输入参数的个数,
- nargout用于获得输出参数的个数

格式 nargin %在函数体中获取实际输入变量的个数

nargout %在函数体中获取实际输出变量的个数

nargin('fun') %在函数体外获取定义的输入参数个数

nargout ('fun') %在函数体外获取定义的输出参数个数

例： 计算两个数的和，根据输入的参数个数使用不同的运算表达式

```
function[sm,n]=Ex_narg(x,y)
%Ex_narg 参数个数可变，计算x和y的和
if nargin==1
    sm=x; % 输入一个参数就计算与0的和
elseif nargin==0
    sm=0; %无输入参数就输出0
else
    sm=x+y; %输入的是两参数就计算和
end
n=nargin;
```

```
x=8;y=9;
[sm,n]=Ex_narg(x,y)
```

在命令窗口调用Ex_narg函数，分别使用2个、1个和无输入参数观察结果。此例计算输入参数不能大于2

3.3.3 局部变量、全局变量和静态变量

- 变量的作用域和生存期：作用域是变量的作用范围，生存期是变量的生存时间。
- 根据变量**作用域**的不同,可将其分为**局部变量**和**全局变量**。
- 根据变量**生存期**的不同，可将其分为**自动变量**和**静态变量**。

注意：函数名与同一作用域的变量名不能重复, 否则函数无法被调用。

1) 局部变量

- 局部变量（Local Variables）的作用范围是定义该变量的函数内部。
- 声明局部变量：不需要特别声明
- 变量的存储：在独立的函数工作空间中
- 变量结束：变量所在的函数执行完毕时。

2) 全局变量

全局变量具有全局作用域，可在不同函数工作空间中共享。

作用：减少参数传递过程，提高程序执行效率。

缺点：全局变量在任何定义过的函数中都可以修改，使用时应十分小心。

声明全局变量

全局变量在使用前必须用“**global**”声明，而且每个要共享全局变量的函数和工作空间，都必须逐个用“global”对该变量加以声明。

清除全局变量

使用clear命令，命令格式如下：

`clear global 变量名` %清除某个全局变量

`clear global` %清除所有的全局变量

3) 静态变量

- 格式: **persistent** x;

它只能在function里声明, 且只有这个function才能认识它, 声明后需初始化, 注意不能直接赋值; 而是先用isempty(x) 判断x是否已经赋值, 若没有, 则可以赋值给x。

如: function y=test()
 persistent a;
 if isempty(a)
 a=0;
 end
 a=a+1;
 y=a;

之后第一次调用y=test; 结果y=1, 第二次调用y=test; 结果y=2。就是说a记录了每次调用function后的结果。

如果a为自动变量, 则每次调用y=test后结果都是1。

3.3.4 嵌套函数、私有函数和重载函数

1) 嵌套函数

在MATLAB中一个函数的内部还可以定义一个或多个函数，这种定义在其他函数内部的函数就称为嵌套函数。此时，每个函数必须用end结束。

调用原则：

- (1) 外层函数可以调用内一层函数，但不能隔层调用；
- (2) 同层嵌套函数可以互相调用
- (3) 内层函数可以调用外一层函数及和外层函数同层的其他函数。

```
function A
    function B
        function C
        end
    end
    function D
        function E
        end
    end
end
```

这里A不是主函数，
如果A为主函数，则B
和D也不能调用A

A可以调用：B， D
B可以调用：C， D， A
C可以调用：B， D
D可以调用：E， B， A
E可以调用：D， B

2) 私有函数

私有函数是限制访问权限的函数，私有函数存放在“private”子目录中，只能被其直接父目录的M函数文件所调用。

3) 重载函数

重载函数是指两个函数使用相同的名称，处理的功能相似，但参数类型或个数不同，重载函数通常放在不同的文件夹下，文件夹名称以“@”开头后面跟一个数据类型名。

3.3.5 函数的工作过程和P码文件

1) 函数的搜索过程

当在MATLAB中输入一个标识符时，首先确认是不是变量名，若不是，则做如下搜索：

- ✓ 检查是否是本M函数文件内部的子函数；
- ✓ 检查是否是“private”目录下的私有函数；
- ✓ 检查是否在当前路径中；
- ✓ 检查是否在搜索路径中。

2) P码文件

- P码就是伪代码（Pseudocode），一个M文件第一次被调用时，MATLAB就将其进行编译并生成P码文件存放在内存中，生成的P码文件与原M文件名相同，其扩展名为“.p”，
- P码文件生成后可直接调用，但M文件修改后需重新生成
- P码文件的保密性好。
- **pcode File1.m,File2.m..... -inplace**
%生成File1.p,File2.p.....文件

3) 函数的工作空间

- 每一个M函数运行时都有一个内存区，称为函数的工作空间。
- 清除函数的工作空间：

clear functions

%清除所有编译过的M函数文件和MEX文件工作空间

clear function funname

%清除某个编译过的函数工作空间

3.4 程序调试

在编译和运行程序时出现错误（警告）无法避免，因此掌握程序调试的方法和技巧对提高工作效率很重要。

下面针对语法错误和逻辑错误推荐两种调试方法，即直接调试法和工具调试法。

出错信息

错误的程序大致分为以下三类：

- **拼写错误**：比如应该是`sum()`，写成了`smu()`。
拼写错误非常容易发现，在程序运行时，系统会提示错误。
- **语法错误**：比如`6/0`，输出结果变为无穷大`Inf`。程序在运行时不一定会报错，但是输出结果不正常。
- **逻辑错误**：这样的错误非常隐蔽，通常是对程序的算法考虑不周全。程序可以正常执行，但是输出结果不符合预期值。

3.4.1 直接调试法

对于不是很复杂的程序，可以使用直接调试法进行程序的调试：

- (1) 如果程序出错或给出警告信息，仔细分析出错或警告信息，会得到出错的原因。
- (2) 在程序中，利用函数`disp()`将函数M文件的中间结果显示出来。
- (3) 在单独调试一个函数M文件时，可以将该函数M文件修改为脚本M文件进行调试。

- (4) 将程序的某些行通过注释进行屏蔽，在程序行的前面输入注释符%。
- (5) 在程序中需要调试的地方添加keyboard函数。

复杂的程序，必须采用工具调试法，即借助MATLAB提供的工具调试器（Debugger）进行调试。

3.4.2 工具调试法

MATLAB提供了调试程序的工具，利用这些工具可以提高编程的效率，包括命令行的调试函数和图形界面的菜单命令。

- 1) 以命令行为主的程序调试
- 2) 以图形界面为主的程序调试

1) 以命令行为主的程序调试

以命令行为主的程序调试手段具有通用性，可以适用于各种平台，它主要是应用MATLAB提供的调试命令。

在命令窗口输入`help debug`可以看到对于这些命令的简单描述，下面分别进行介绍。

常用的调试命令

命令	描述
dbstop	设置断点
dbstatus	显示所有断点
dbclear	清除断点
dbstep	执行一行或多行语句
dbcont	恢复执行
dbtype	显示文件内容
dbdown	改变工作空间
dbup	改变工作空间
dbmex	允许MEX文件调试
dbquit	退出调试
dbstack	调用堆栈

(1) 设置断点 **dbstop**

这是最重要的部分，可以利用它来指定程序代码的断点，使得程序在断点前停止执行，并进入调试模式，从而可以检查当前各个变量的值。

➤ **dbstop in mfile**

在文件名为mfile的M文件的第一个可执行语句前设置断点，忽略程序注释行。当程序执行到该断点处，暂时中止程序的执行，进入调试模式。

➤ **dbstop in mfile at lineno**

在文件名为mfile的M文件的第lineno行设置断点。如果第lineno行为非执行语句，则在其后的第一个可执行语句前设置断点。

➤ **dbstop in mfile at subfun**

在文件名为 mfile 的 M 文件的子程序 subfun 的第一个可执行语句前设置断点。

➤ **dbstop if error**

在程序运行遇到错误时，自动设置断点。
这里的错误不包括try...catch之间的错误。

➤ **dbstop if all error**

在程序运行遇到错误时，自动设置断点。
这里的错误包括try...catch之间的错误。

➤ **dbstop if warning**

在程序运行遇到警告时，自动设置断点。

➤ **dbstop if catch error**

在程序运行try...catch间代码遇到错误时，自动设置断点。

➤ **dbstop if naninf 或 dbstop if infnan**

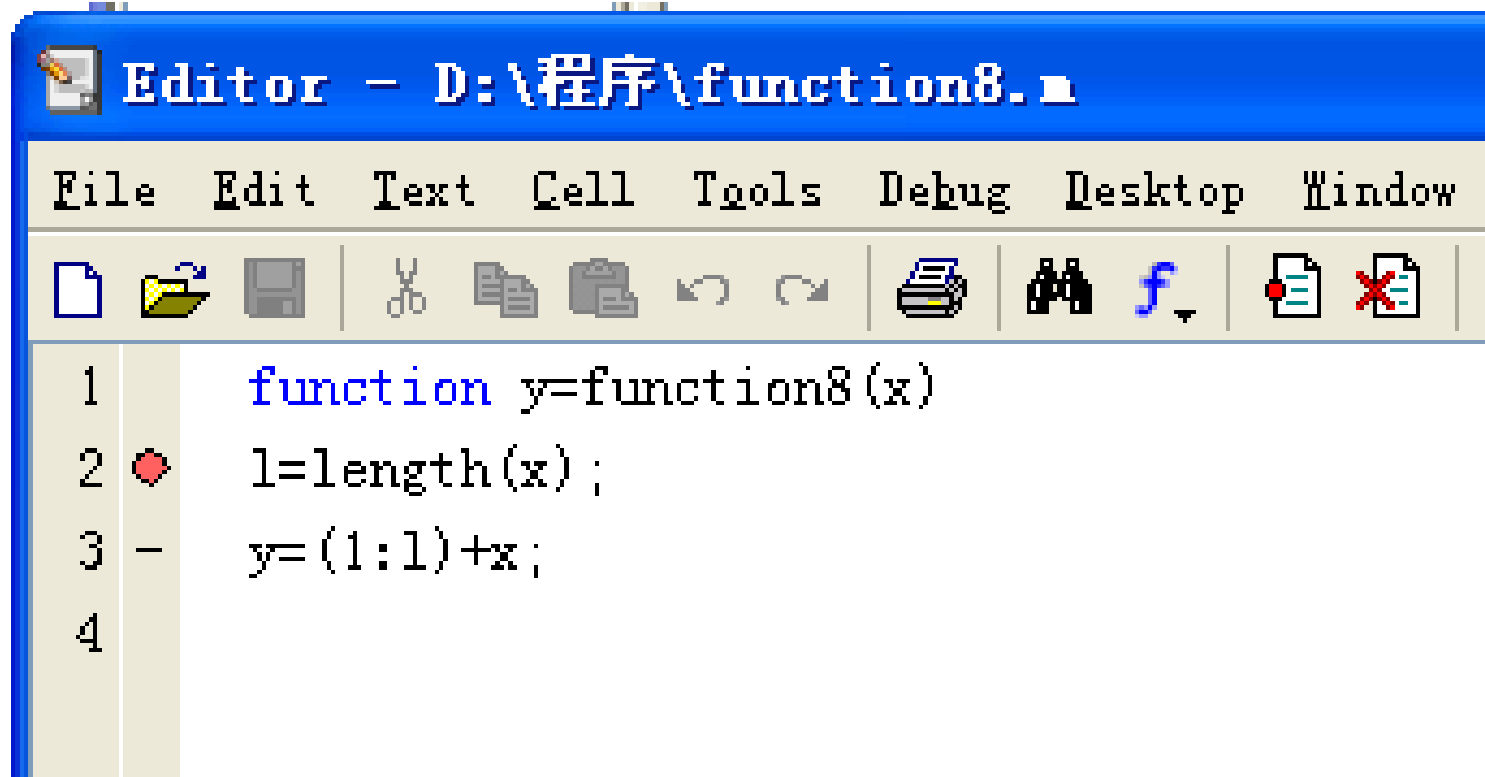
当程序运行遇到无穷值或者非数值时，自动设置断点。

例：以下列函数function8()为例说明如何使用命令调试程序。

```
function y=function8(x)  
l=length(x)  
y=(1:l)+x
```

由程序不难看出，函数function8()中的输入只能为向量，如果输入矩阵则会产生错误

在命令窗口输入**dbstop in function8**，并打开文件**function8.m**就可看到如下图所示的界面，它在第一个可执行语句前设置了断点。

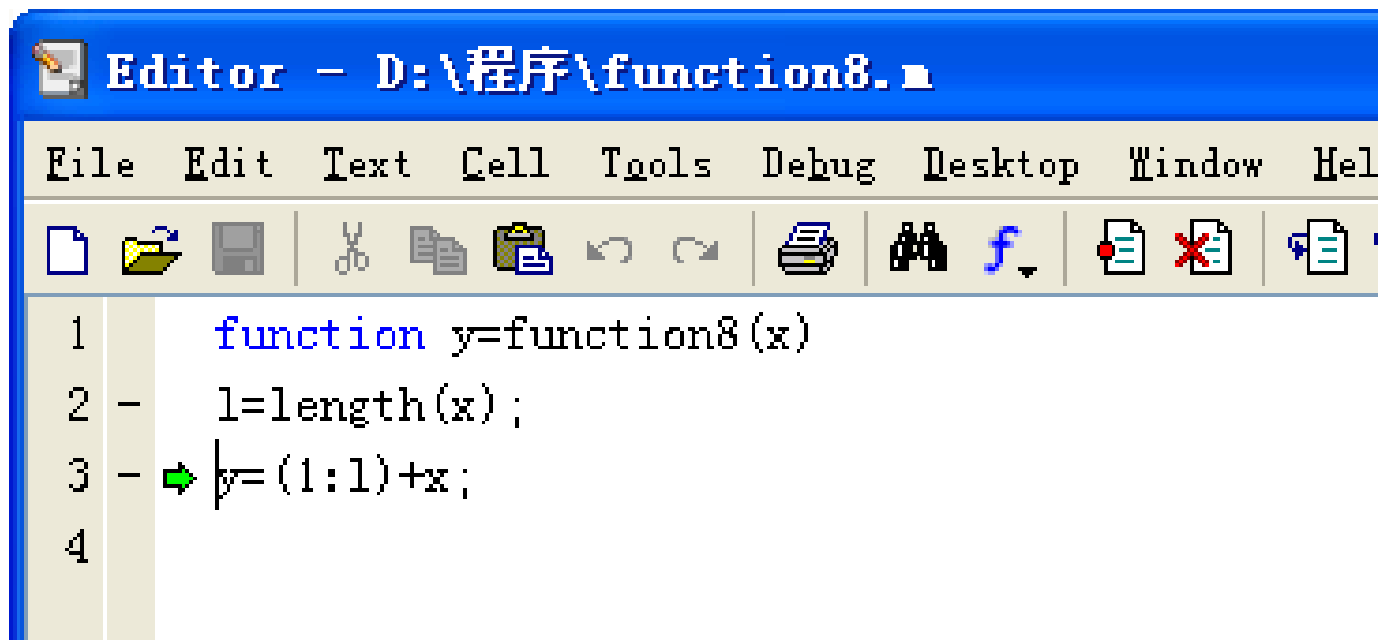


单击图中红点，会发现红点被取消，此时回复到初始状态。然后在命令窗口依次输入**dbstop if error**和**function8(magic(3))**，可得到如下的运行结果和如下图所示的界面。

??? Error using ==> unknown
Matrix dimensions must agree.

Error in ==> function8 at 3
y=(1:1)+x;

R>>



The image shows a screenshot of the MATLAB Editor window. The title bar reads "Editor - D:\程序\function8.m". The menu bar includes "File", "Edit", "Text", "Cell", "Tools", "Debug", "Desktop", "Window", and "Help". The toolbar contains icons for file operations (New, Open, Save, Cut, Copy, Paste, Undo, Redo), printing, and debugging. The code editor displays the following function definition:

```
1 function y=function8(x)
2 -     l=length(x);
3 -     ➡ y=(1:1)+x;
4
```

A green arrow points to line 3, indicating the location of the error. The error message above the editor states: "??? Error using ==> unknown Matrix dimensions must agree. Error in ==> function8 at 3 y=(1:1)+x;"

(2) 清除断点 **dbclear**

➤ **dbclear all**

清除所有M文件中的所有断点。

➤ **dbclear all in mfile**

清除文件名为mfile的M文件中的所有断点。

➤ **dbclear in mfile**

清除文件名为mfile的M文件中第一个可执行语句前的断点。

➤ **dbclear in mfile at lineno**

清除文件名为mfile的M文件中第lineno行语句前的断点。

➤ **dbclear in mfile at subfun**

清除文件名为mfile的M文件中子程序subfun的第一个可执行语句前的断点。

➤ **dbclear if error**

清除由dbstop if error设置的断点。

➤ **dbclear if warning**

清除由dbstop if warning设置的断点。

➤ **dbclear if naninf**

清除由dbstop if naninf设置的断点。

➤ **dbclear if infnan**

清除由dbstop if infnan设置的断点。

(3) 恢复执行

➤ dbcont

此命令可从断点处恢复程序的执行，直到遇到程序的另一个断点或错误。

(4) 调用堆栈

➤ **dbstack**

此命令显示M文件名和断点产生的行号、调用此M文件的文件名和行号等，直到最高层的M文件，即列出了函数调用的堆栈。

(5) 列出所有断点

➤ **dbstatus**

此命令可列出所有的断点，包括错误、警告、nan和inf等。

➤ **dbstatus mfile**

此命令可列出文件名为mfile的M文件中的所有断点。

(6) 执行1行或多行语句 **dbstep**

➤ **dbstep**

执行当前M文件下一个可执行语句。

➤ **dbstep nlines**

执行当前M文件下nlines行可执行语句。

➤ **dbstep in**

当下一条可执行语句是对另一个函数的调用，此命令将从被调用函数的第一个可执行语句执行。

➤ **dbstep out**

此命令将执行函数剩余的代码然后停止。

(7) 列出文件内容

➤ **dbtype mfile**

列出文件名为mfile的M文件中的内容。

➤ **dbtype mfile start:end**

列出文件名为mfile的M文件中指定行号范围的部分。

(8) 切换工作空间

➤ **dbdown**

遇到断点时，将当前工作空间切换到被调用M文件的工作空间。

➤ **dbup**

将当前工作空间（断点处）切换到调用文件的工作空间。

(9) 退出调试模式

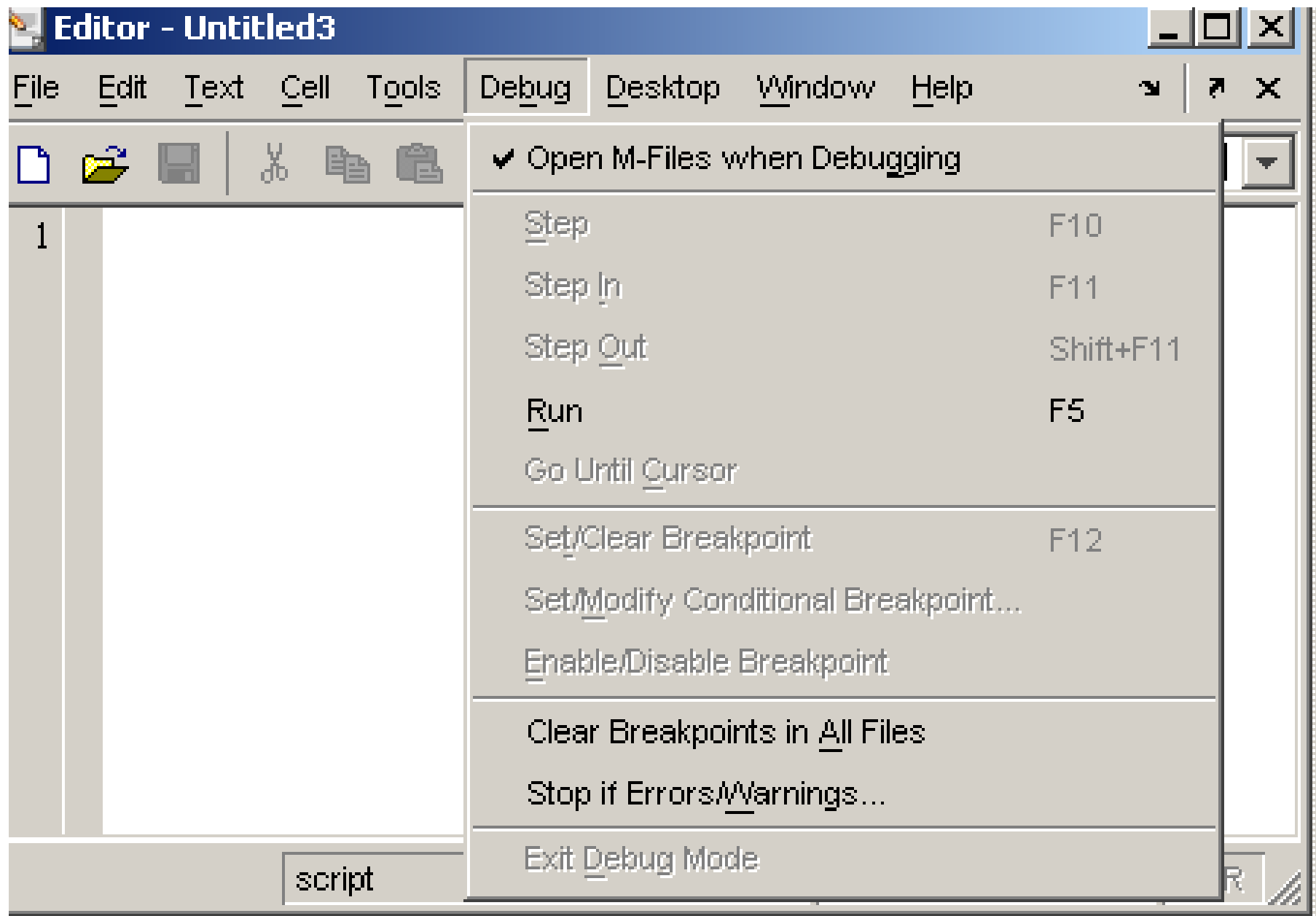
➤ **dbquit**

立即结束调试器并返回到基本工作空间，
但所有断点仍有效。

2) 以图形界面为主的程序调试

MATLAB自带的文本编辑器同时也是程序的编译器，用户可以在程序编辑后直接进行调试，更加方便和直观。

通过新建M文件打开文本编辑器和编译器，选择主菜单中【Debug】选项，其下拉菜单包括多种调试命令，如下图所示。





程序编辑工具区

程序调试工具区

下拉菜单中的命令有一部分在工具栏中有图标相对应，其功能与命令行调试程序是相同的，下面只对各命令做简单介绍。

➤ **step (F10)**

程序的单步执行，和命令dbstep相对应；

➤ **step in (F11)**

进入被调用的函数，和命令dbstep in对应；

➤ **step out (shift+F11)**

跳出被调用函数，和命令dbstep out对应；

➤ **run/continue (F5)**

执行程序或继续执行，和命令dbcont对应；

➤ **go until cursor**

运行到鼠标所在行；

➤ **set/clear breakpoint (F12)**

设置或清除断点，如果该行没有断点则设置断点，如果该行有断点，则清除断点。与命令dbstop和dbclear对应；

➤ **set/modify conditional breakpoint...**

设置或修改鼠标所在行的条件断点，运行后回提示用户输入断点的条件；

➤ **enable/disable breakpoint**

允许或禁止断点；

➤ **clear breakpoint in all files**

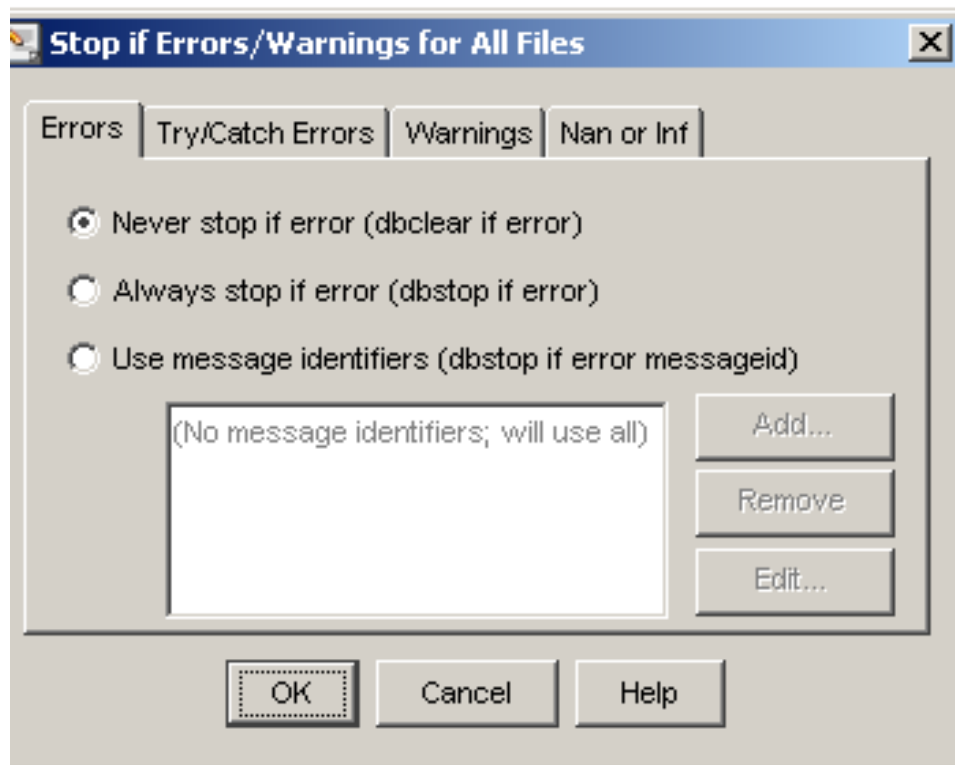
清除所有文件的断点，和命令**dbclear all**对应；

➤ **stop if errors/warnings...**

如果遇到错误或警告等则停止执行，进入调试模式；

➤ **exit debug mode (shift+F5)**

退出调试模式，与命令**dbquit**对应



See example

3.5 编程技巧

1) 程序执行时间

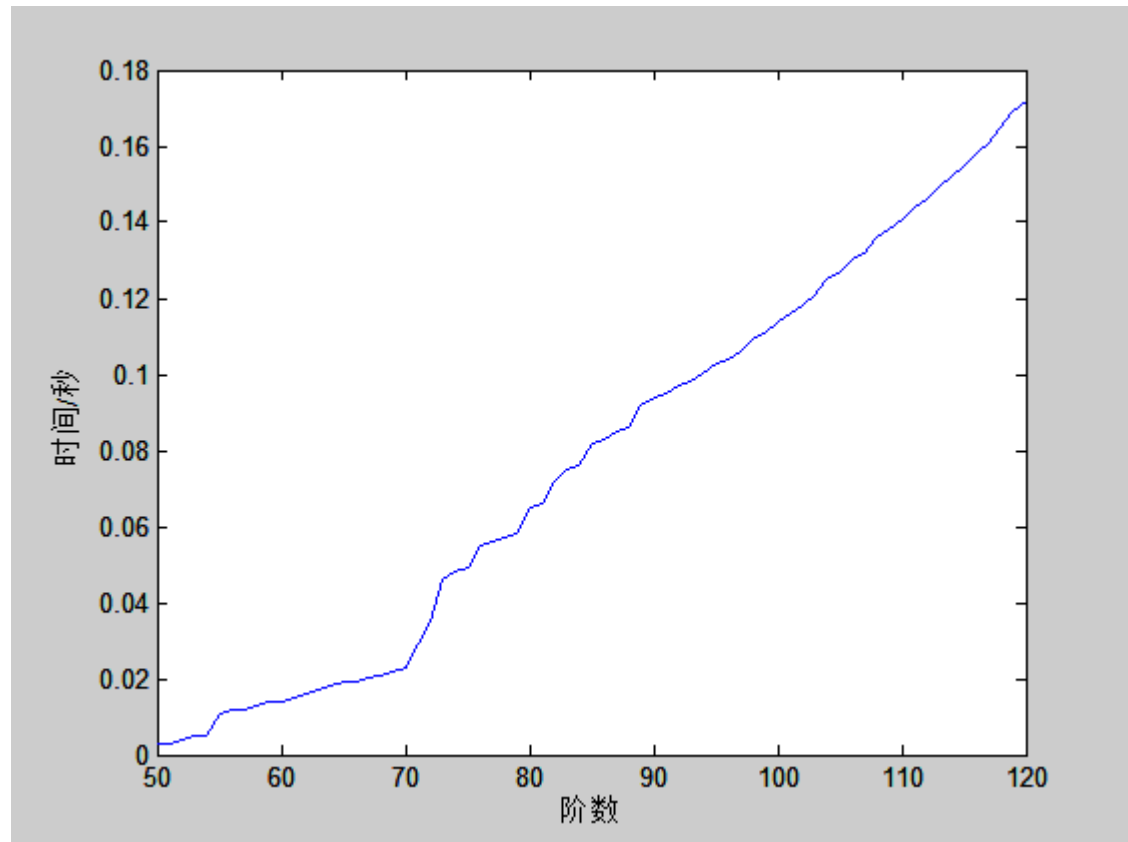
可以对程序进行计时，从而获得程序的执行时间，共有3种方法：

- 函数tic()和函数toc()
- 函数cputime()
- 函数etime()

➤ 函数tic()和函数toc()

函数tic启动一个秒表，函数toc停止这个秒表，并计算出所经历的时间，这两个函数配合使用，可以计算出程序或函数的运行时间。

```
tic
for i=50:1200
    a=inv(pascal(i));
    t(i-49)=toc;
end
figure;
plot(50:1200,t);
xlabel('阶数');
ylabel('时间/秒');
```



➤ 函数 `cputime(t1, t0)`

该函数返回用秒表示的从当前matlab启动后所用的CPU时间，单位为秒。

```
clear all;
```

```
t=cputime
```

```
i=1:1000000;
```

```
y(i)=exp(i);
```

```
t1=cputime-t    %计时
```

```
t =13.1977
```

```
t1 =0.0624
```

➤ 函数etime()

该函数返回参数t1和t0之间的时间，单位为秒，其中参数t1和t0为函数clock返回的包含6个元素的向量

```
t=clock
```

```
for i=1:1000000           %采用循环
```

```
    y(i)=exp(i);
```

```
end
```

```
t1=etime(clock, t)       %进行计时
```

```
clear all;
```

```
t=clock
```

```
i=1:1000000;             %采用矢量
```

```
y(i)=exp(i);
```

```
t2=etime(clock,t)        %进行计时
```

```
t =  
    1.0e+003 *  
    2.0180    0.0100    0.0150    0.0100    0.0540    0.0086  
t1 =  
    0.2500  
t =  
    1.0e+003 *  
    2.0180    0.0100    0.0150    0.0100    0.0540    0.0092  
t2 =  
    0.0470
```

采用for循环时，程序运行时间为0.2500秒，
采用向量进行计算时，程序运行时间为0.0470秒，
编程时尽量把循环向量化，对矩阵或向量编程，而不是对矩阵的元素编程。

2) 编程技巧

➤ 在利用matlab编程时，应尽量避免使用循环

```
x=rand(3,4,5);  
tic;  
m=x(1,1,1);  
for i=1:3  
    for j=1:4  
        for k=1:5  
            if x(i,j,k)>m  
                m=x(i,j,k);  
            end  
        end  
    end  
end  
m  
t=toc
```

求3维矩阵问题的最大值
利用for循环

```
m =  
    0.9706  
t =  
    4.7785e-004
```

```
x=rand(3,4,5);
```

```
tic;
```

```
m=max(x(:))
```

```
t=toc
```

求3维矩阵问题的最大值
利用max函数获得最大值

```
m =
```

```
0.9991
```

```
t =
```

```
7.9231e-005
```

在程序中，利用x(:)将多维矩阵转换为一维矩阵，直接采用函数max获得最大值。

➤ 如果必须使用多重循环，但是两个循环执行的次数不同，建议在外循环执行循环次数少的，内循环执行循环次数多的，可以显著提高速度

建立5x20000的矩阵，其i行，j列的元素分别为 $1/(i+j-1)$

```
tic;  
for i=1:20000  
    for j=1:5  
        H(j,i)=1/(i+j-1);  
    end  
end  
toc
```

外循环次数多
4.536170秒

```
tic;  
for i=1:5  
    for j=1:20000  
        H(i,j)=1/(i+j-1);  
    end  
end  
toc
```

内循环次数多
1.686976秒

➤ 对于大型矩阵，如果预先设定维数，会减少程序的运行时间，提高程序执行效率

```
tic;  
for i=1:5  
    for j=1:2000000  
        H(i,j)=1/(i+j-1);  
    end  
end  
toc
```

内循环次数多
1.101572秒

```
tic;  
H=zeros(5,2000000);  
for i=1:5  
    for j=1:2000000  
        H(i,j)=1/(i+j-1);  
    end  
end  
toc
```

预先设定维数
0.569074秒

不同的算法，运行时间可相差很多！！！！

➤ 在编写程序时，优先考虑matlab的内在函数
矩阵运算应该尽量采用matlab的内在函数，因为内在函数是由更底层的编程语言C编写的，其执行速度通常快于用户自己编写的程序。

matlab中由工具箱组成了一个非常庞大的函数库，用户不需要也不可能掌握所有的函数，但是对于常用的函数要非常熟悉。

➤ 如果想继续提高程序的效率，可以考虑采用更高效的算法

在实际问题中，解决同样的问题经常可以采用各种各样的算法。例如求解定积分的数值解法在matlab中就提供了两个函数quad()和quad8()，其中后一个算法在精度、速度上明显高于前一种。

在利用matlab编程解决实际问题时，如果某个方法不能满足实际需要，可以考虑尝试其他方法。

3) 小技巧

- 对于M文件的命名，尽量不要用简单的英文单词，最好是由大小写英文字母、数字和下划线等组成。
- 在调试程序时，经常要屏蔽掉多行程序，可以在选定程序后，按快捷键Ctrl+r将这些程序行注释掉。
- 快捷键Ctrl+c可以中断正在执行的操作。
- 使用Tab键补全函数名或文件名。
- 在命令行窗口中，使用上下光标键↑和↓，将历史记录中的命令复制到输入位置。
- 使用文本编辑器的cell模式，能使程序更加清晰。

4) 良好的编程习惯

- ✓ 数据结构必须事先规划好，如果数据结构设计存在错误或不妥，那么程序修改的工作量将是巨大的。
- ✓ 尽量避免使用全局变量。
- ✓ 函数尽可能功能简明，使其可以重用，从而程序实现模块化。
- ✓ 良好的编写风格，使得别人或者自己能够容易读懂之前所写的代码。具体的方法包括：变量和函数名统一按规律命名，并具有较明确的意义；代码层次分明；注释清楚且充分等。
- ✓ 注重程序的充分测试，注意警告信息。
- ✓ 具有建立和求解数学模型的能力，能够简化程序的复杂性。

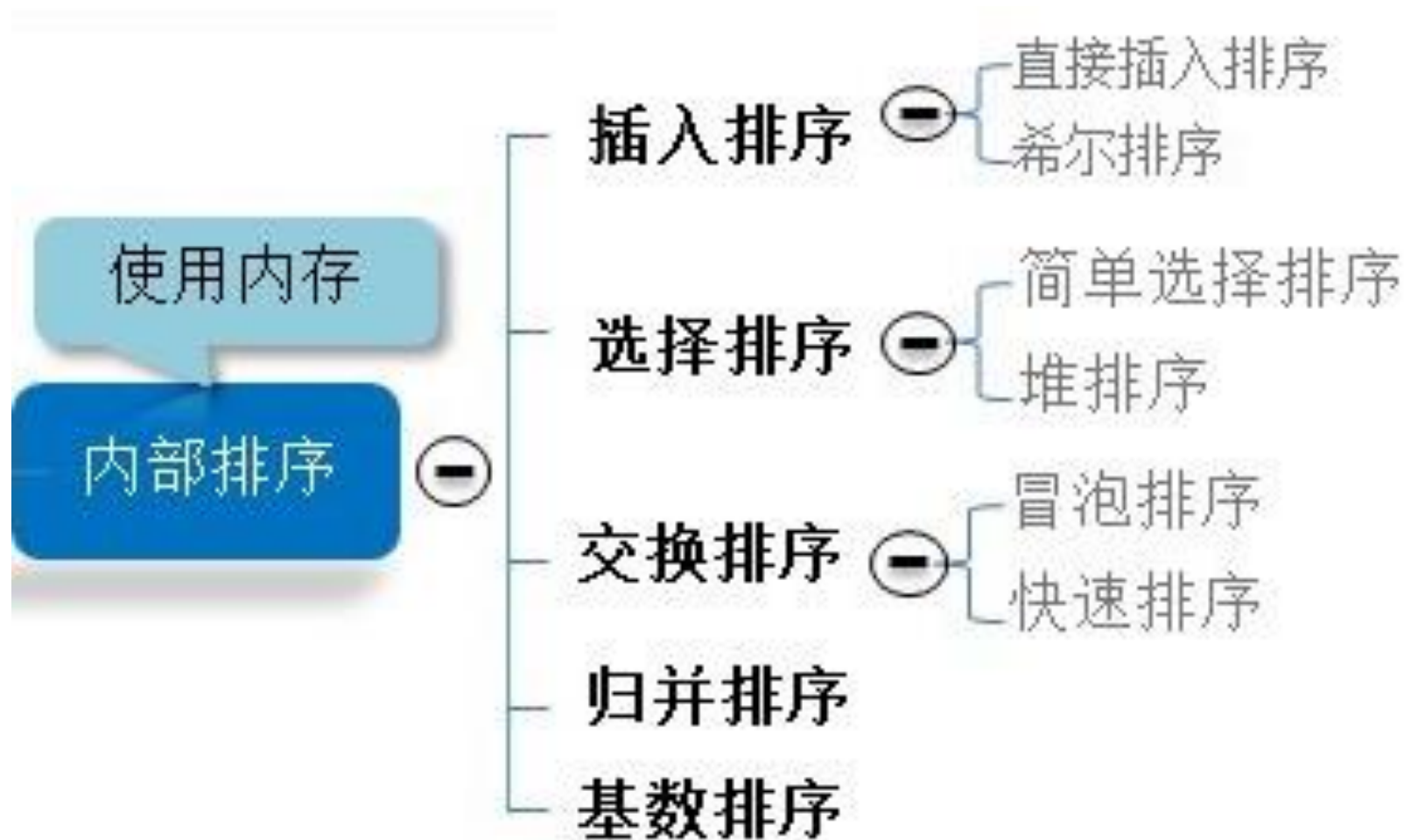
3.6 编程实例

3.6.1 排序算法：

所谓排序，就是使一串记录，按照其中的某个或某些关键字的大小，递增或递减的排列起来的操作。

排序算法，就是如何使得记录按照要求排列的方法。排序算法在很多领域得到相当重视，尤其是在大量数据的处理方面。

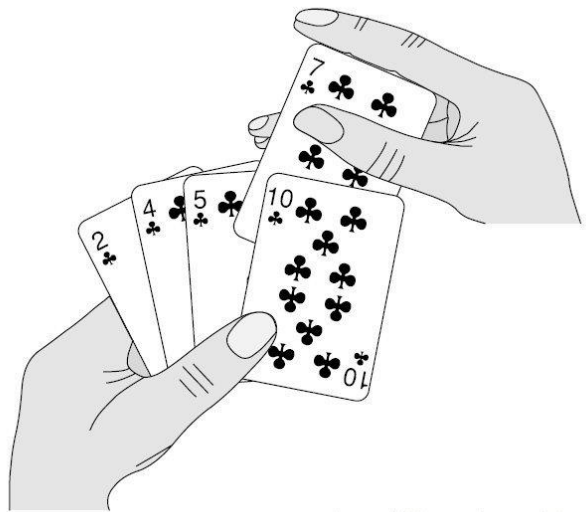
➤ 常用的排序算法



(1) 插入排序—直接插入排序

基本思想:

将初始序列中的第一个元素作为一个有序序列，然后将剩下的 $n-1$ 个元素按关键字大小依次插入该有序序列，每插入一个元素后依然保持该序列有序，经过 $n-1$ 趟排序后使初始序列有序。



有序序列L【1...i-1】

L(i)

无序序列L【i+1...n】

示例：

初始序列：92 77 67 8 6 84 55 85 43 67

第1趟： [77 92] 67 8 6 84 55 85 43 67
第2趟： [67 77 92] 8 6 84 55 85 43 67
第3趟： [8 67 77 92] 6 84 55 85 43 67
第4趟： [6 8 67 77 92] 84 55 85 43 67
第5趟： [6 8 67 77 84 92] 55 85 43 67
第6趟： [6 8 55 67 77 84 92] 85 43 67
第7趟： [6 8 55 67 77 84 85 92] 43 67
第8趟： [6 8 43 55 67 77 84 85 92] 67
第9趟： [6 8 43 55 67 67 77 84 85 92]
结 果： [6 8 43 55 67 67 77 84 85 92]

function y=insertsort(x)

%插入法排序

r=length(x)

for i=1:r

for j=i-1:-1:1

if x(j)>x(j+1)

 temp=x(j);

 x(j)=x(j+1);

 x(j+1)=temp;

elseif x(j)<=x(j+1)

break

end

end

 x

end

(2) 交换排序——冒泡排序

基本思想：

第一趟在序列($A[0] \sim A[n-1]$)中从前往后进行两个相邻元素的比较，若后者小，则交换，比较 $n-1$ 次；第一趟排序结束，最大元素被交换到 $A[n-1]$ 中，下一趟排序只需要在子序列($A[0] \sim A[n-2]$)中进行；冒泡排序最多进行 $n-1$ 趟。基本的冒泡排序可以利用旗标的方式稍微减少一些比较的时间，当寻访完序列后都没有发生任何的交换动作，表示排序已经完成，而无需再进行之后的比较与交换动作。

示例：

初始序列：95 27 90 49 80 58 6 9 18 50

第1趟： 27 90 49 80 58 6 9 18 50 [95]

第2趟： 27 49 80 58 6 9 18 50 [90 95]

第3趟： 27 49 58 6 9 18 50 [80 90 95]

第4趟： 27 49 6 9 18 50 [58 80 90 95]

第5趟： 27 6 9 18 49 [50 58 80 90 95]

第6趟： 6 9 18 27 [49 50 58 80 90 95]

第7趟： 6 9 18 [27 49 50 58 80 90 95]

由于之后不会再发生交换动作，排序提早结束

结 果： [6 9 18 27 49 50 58 80 90 95]

```
function y=bubblesort(x) %冒泡法排序.
```

```
r=length(x);
```

```
for i=1:r
```

```
    for j=1:r-1
```

```
        if x(j)>x(j+1)
```

```
            temp=x(j);
```

```
            x(j)=x(j+1);
```

```
            x(j+1)=temp;
```

```
        end
```

```
    end
```

```
x
```

```
end
```

```
y=x;
```

(3) 选择排序——简单选择排序

基本思想:

将初始序列($A[0] \sim A[n-1]$)作为待排序序列, 第一趟在待排序序列($A[0] \sim A[n-1]$)中找到最小值元素, 将其与第一个元素 $A[0]$ 交换, 这样子序列($A[0]$)已经有序, 下一趟在排序在待排序子序列($A[1] \sim A[n-1]$)中进行。第 i 趟排序在待排序子序列($A[i-1] \sim A[n-1]$)中找到最小值元素, 与该子序列中第一个元素 $A[i-1]$ 交换。经过 $n-1$ 趟排序后使得初始序列有序。

示例：

初始序列：70 80 31 37 10 1 48 60 33 80

第1趟： [1] 80 31 37 10 70 48 60 33 80
第2趟： [1 10] 31 37 80 70 48 60 33 80
第3趟： [1 10 31] 37 80 70 48 60 33 80
第4趟： [1 10 31 33] 80 70 48 60 37 80
第5趟： [1 10 31 33 37] 70 48 60 80 80
第6趟： [1 10 31 33 37 48] 70 60 80 80
第7趟： [1 10 31 33 37 48 60] 70 80 80
第8趟： [1 10 31 33 37 48 60 70] 80 80
第9趟： [1 10 31 33 37 48 60 70 80] 80
结 果： [1 10 31 33 37 48 60 70 80 80]

MATLAB程序由同学们自己课后设计！！！！

3.6.2 猜数游戏

首先由计算机产生 $[1,100]$ 之间的随机整数，然后由用户猜测所产生的随机数。根据用户猜测的情况给出不同提示：

- (1) 猜测的数大于产生的数，则显示 “High”
 - (2) 猜测的数小于产生的数，则显示 “Low”
 - (3) 等于产生的数，则显示 “You won”，同时退出游戏，
- 用户最多可以猜7次。


```
x=fix(100*rand(1));
n=7;
while n>0
    y=input ('please inpiut your guessing number:')
    if x>y
        disp('low')
    elseif x<y
        disp ('high')
    else
        disp('you win!!!')
        break
    end
    n=n-1;
end
```

3.6.3 柯雷茨 (collatz) 猜想

对于任意的正整数 n ：如果 n 是偶数则除以2，如果还是偶数再除以2，直至成为奇数，如果是奇数乘以3加1再除以2，如果是偶数继续除以2，直至成为一个新的奇数。然后再将这个奇数乘以3加1，按照同样的办法计算下去。经过若干次计算之后，最后的得数是1。

```
x=input ('please inpiut a integer number:')
```

```
n=0;
```

```
while x~=1
```

```
    if mod(x,2)==0
```

```
        x=x/2
```

```
    else
```

```
        x=x*3+1
```

```
    end
```

```
    n=n+1;
```

```
end
```

```
x
```

```
n
```

3.6.4 用筛选法求某自然数范围内的全部素数。

素数是大于1，且除了1和它本身以外，不能被其他任何整数所整除的整数。用筛选法求素数的基本思想是：要找出2~m之间的全部素数，首先在2~m中划去2的倍数(不包括2)，然后划去3的倍数(不包括3)，由于4已被划去，再找5的倍数(不包括5)，...，直到再划去不超过的数的倍数，剩下的数都是素数。

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Prime numbers

2 3 5 7

11 13 17 19

23 29 31 37

41 43 47 53

59 61 67 71

73 79 83 89

97 101 103 107

109 113

```
n=100000
a=1:n;
for j=2:fix(sqrt(n))
    for i=4:n
        if i==j
            continue
        end
        if a(i)==0
            continue
        end
        if mod(i,j)==0
            a(i)=0;
            continue
        end
    end
end
end
```

筛法计算

```
for k=2:n
    if a(k)~=0
        a(k)
    end
end
end
```

素数显示