

## 第3讲 MATLAB程序设计

2019年10月9日

### 内容提要

- MATLAB控制语句
- M文件
- 程序的调试与优化

### 内容提要

- MATLAB控制语句
  - 顺序结构语句
  - 循环结构语句
  - 条件结构语句
  - 选择结构语句
  - try-catch语句
- M文件
- 程序的调试与优化

## 顺序结构

### • 顺序结构

- 顺序结构是指按照程序中语句从上到下、从左到右的排序顺序依次执行，直到程序的最后一个语句
- 这是最简单的一种程序结构，结构单一，容易编写；
- 一般涉及数据的输入、计算或处理、输出等内容

### • 数据的输入

- 通过键盘输入数据，可以使用input函数来实现
- 调用格式为：A=input(提示信息，选项)  
如采用's'选项，则允许用户输入一个字符串  
`xm=input('What's your name?','s');` %提示输入一个人的姓名

## 顺序结构

### • 数据的输出；主要有disp函数和fprintf函数

- disp函数：调用格式为disp(输出项)  
`>>disp('What's your name?')`  
What's your name?
- fprintf函数：调用格式为fprintf('输出格式'，输出项)  
`>>fprintf('圆周率pi=%10.9f\n', pi)`  
圆周率pi=3.141592654  
`>>n=23;`  
`>>fprintf('n=%d',n)`  
n = 23

## 顺序结构

### • 程序的暂停

- 当程序运行时，为了查看程序的中间结果或输出的图形，有时需要暂停程序的执行,这时可以使用pause函数
- 调用格式为：`pause (延迟秒数)`
- 如果省略延迟时间，直接使用`pause`，则将暂停程序，直到用户按任一键后程序继续执行
- 若要强行中止程序的运行可使用`Ctrl+C`命令

## 顺序结构

### • 程序的顺序执行

- 按照程序语句逐条顺序执行语句命令  
`>>a=1;`  
`>>b=2;`  
`>>c=a+b;`  
`>>d=c^2;`  
`>> disp('Done!')`

## 顺序结构-示例

求一元二次方程  $ax^2 + bx + c = 0$  的根

```
a=input('a=?');
b=input('b=?');
c=input('c=?');
d=b*b-4*a*c;
x=[(-b+sqrt(d))/(2*a),(-b-sqrt(d))/(2*a)];
disp(['x1=',num2str(x(1)),',x2=',num2str(x(2))]);
a=?1
b=?9
c=?6
x1=-0.72508,x2=-8.2749
```

## 条件结构

- if语句称为条件执行语句，其关键字包括if、else、elseif和end

- 调用格式：

- 1、单分支结构

if 表达式

语句组A

end

当条件成立时，则执行语句组，执行完后继续执行if语句的后继语句，若条件不成立，则直接执行if语句的后继语句

## 条件结构

- 调用格式：

- 2、双分支结构

if 表达式

语句组A

else

语句组B

end

当条件成立时，执行语句组1，否则执行语句组2，语句组1或语句组2执行后，再执行if语句的后继语句

## 条件结构

- 调用格式：

- 3、多分支if结构

if 表达式1

语句组A

elseif 表达式2

语句组B

else


语句组C

end

# 条件结构

- 多分支if 结构-示例
  - 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')));  
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
```



北京航空航天大学  
BEIHANG UNIVERSITY

13

| ASCII 字符代码表 一 |     |              |       |      |     |       |      |    |     |      |     |        |     |            |     |      |     |      |     |      |     |    |      |     |              |
|---------------|-----|--------------|-------|------|-----|-------|------|----|-----|------|-----|--------|-----|------------|-----|------|-----|------|-----|------|-----|----|------|-----|--------------|
| 高四位           |     | ASCII非打印控制字符 |       |      |     |       |      |    |     |      |     |        |     | ASCII 打印字符 |     |      |     |      |     |      |     |    |      |     |              |
|               |     | 0000         |       |      |     | 0001  |      |    |     | 0010 |     | 0011   |     | 0100       |     | 0101 |     | 0110 |     | 0111 |     |    |      |     |              |
|               |     | 0            |       |      |     | 1     |      |    |     | 2    |     | 3      |     | 4          |     | 5    |     | 6    |     | 7    |     |    |      |     |              |
| 低四位           | 十进制 | 字符           | ctrl  | 代码   | 十进制 | 字符    | ctrl | 代码 | 十进制 | 字符   | 十进制 | 字符     | 十进制 | 字符         | 十进制 | 字符   | 十进制 | 字符   | 十进制 | 字符   | 十进制 | 字符 | ctrl |     |              |
| 0000          | 0   | 0            | BLANK | FULL | 0   | ␣     | NULL | 空  | 16  | 16   | ␣   | P      | DLR | 数据分隔符      | 32  | 48   | 0   | 64   | @   | 80   | P   | 96 | ?    | 112 | p            |
| 0000          | 1   | 1            | ☺     | A    | SOH | 标题开始  | 17   | 17 | ^   | Q    | DC1 | 设备控制 1 | 33  | !          | 49  | 1    | 65  | A    | 81  | Q    | 97  | a  | 113  | q   |              |
| 0010          | 2   | 2            | ☹     | B    | STX | 正文开始  | 18   | 18 | ↑   | R    | DC2 | 设备控制 2 | 34  | "          | 50  | 2    | 66  | B    | 82  | R    | 98  | b  | 114  | r   |              |
| 0011          | 3   | 3            | ♥     | C    | ETX | 正文结束  | 19   | 19 | !!  | S    | DC3 | 设备控制 3 | 35  | #          | 51  | 3    | 67  | C    | 83  | S    | 99  | c  | 115  | s   |              |
| 0100          | 4   | 4            | ☹     | D    | EOF | 传输结束  | 20   | 20 | ↑   | T    | DC4 | 设备控制 4 | 36  | \$         | 52  | 4    | 68  | D    | 84  | T    | 100 | d  | 116  | t   |              |
| 0101          | 5   | 5            | ♣     | E    | ENQ | 查询    | 21   | 21 | ↑   | U    | NAK | 反确认    | 37  | %          | 53  | 5    | 69  | E    | 85  | U    | 101 | e  | 117  | u   |              |
| 0110          | 6   | 6            | ♠     | F    | ACK | 确认    | 22   | 22 | ↑   | V    | SYN | 同步空闲   | 38  | &          | 54  | 6    | 70  | F    | 86  | V    | 102 | f  | 118  | v   |              |
| 0111          | 7   | 7            | ☹     | G    | BEL | 晨铃    | 23   | 23 | ↑   | W    | ETB | 传输结束   | 39  | '          | 55  | 7    | 71  | G    | 87  | W    | 103 | g  | 119  | w   |              |
| 1000          | 8   | 8            | ☹     | H    | BS  | 退格    | 24   | 24 | ↑   | X    | CAN | 取消     | 40  | (          | 56  | 8    | 72  | H    | 88  | X    | 104 | h  | 120  | x   |              |
| 1001          | 9   | 9            | ☹     | I    | TAB | 水平制表符 | 25   | 25 | ↓   | Y    | EM  | 媒体结束   | 41  | )          | 57  | 9    | 73  | I    | 89  | Y    | 105 | i  | 121  | y   |              |
| 1010          | A   | 10           | ☹     | J    | LF  | 换行/换行 | 26   | 26 | →   | Z    | SUB | 替换     | 42  | *          | 58  | 10   | 74  | J    | 90  | Z    | 106 | j  | 122  | z   |              |
| 1011          | B   | 11           | ☹     | K    | VT  | 垂直制表符 | 27   | 27 | ←   | [    | ESC | 转义     | 43  | +          | 59  | 11   | 75  | K    | 91  | [    | 107 | k  | 123  | z   |              |
| 1100          | C   | 12           | ♀     | L    | FF  | 换页/换页 | 28   | 28 | ↔   | ^    | FS  | 文件分隔符  | 44  | 0          | 60  | <    | 76  | L    | 92  | ^    | 108 | l  | 124  | l   |              |
| 1101          | D   | 13           | ♠     | M    | CR  | 回车    | 29   | 29 | ↔   | ^    | GS  | 组分隔符   | 45  | -          | 61  | =    | 77  | M    | 93  | ]    | 109 | m  | 125  | }   |              |
| 1110          | E   | 14           | ♫     | N    | SO  | 移出    | 30   | 30 | ▲   | ^    | RS  | 记录分隔符  | 46  | .          | 62  | >    | 78  | N    | 94  | ^    | 110 | n  | 126  | ~   |              |
| 1111          | F   | 15           | ☹     | O    | SI  | 移入    | 31   | 31 | ▼   | ^    | US  | 单元分隔符  | 47  | /          | 63  | ?    | 79  | O    | 95  | ^    | 111 | o  | 127  | A   | End of ASCII |

注：表中的ASCII字符可以用:ALT + “小键盘上的数字键” 输入


## 选择结构

- 选择结构用于分支判断选择，包括switch-case语句
- 调用格式：

```
switch 表达式
case 常量表达式1
    语句1
case 常量表达式2
    语句2
...
otherwise
    语句组n+1
end
```
- switch后表达式可为任意类型，如字符串、矩阵、数组等

特别注意：

- 1、多个case语句的常量表达式的值可相同，但只执行第一个而忽略其他
- 2、执行完一个case后的语句后，不会再判断其他case语句：不需要break语句

 北京航空航天大学  
BEIHANG UNIVERSITY

15

## 选择结构-示例

- 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来表示):
  - $\text{price} < 200$  没有折扣
  - $200 \leq \text{price} < 500$  3%折扣
  - $500 \leq \text{price} < 1000$  5%折扣
  - $1000 \leq \text{price} < 2500$  8%折扣
  - $2500 \leq \text{price} < 5000$  10%折扣
  - $5000 \leq \text{price}$  14%折扣
- 输入所售商品的价格，求其实际销售价格。



北京航空航天大学  
BEIHANG UNIVERSITY

16

## 选择结构-示例

```
price=input('请输入商品价格');
switch fix(price/100)
    case {0,1}           %价格小于200
        rate=0;
    case {2,3,4}         %价格大于等于200但小于500
        rate=3/100;
    case num2cell(5:9)   %价格大于等于500但小于1000
        rate=5/100;
    case num2cell(10:24) %价格大于等于1000但小于2500
        rate=8/100;
    case num2cell(25:49) %价格大于等于2500但小于5000
        rate=10/100;
    otherwise            %价格大于等于5000
        rate=14/100;
end
price=price*(1-rate)      %输出商品实际销售价格
```

## 循环结构

### • for循环

- for循环将循环体中的语句重复执行预定的次数，其循环次数通常是已知的

- 调用格式:

for 循环变量=表达式1:表达式2:表达式3

循环体语句

end

其中表达式1的值为循环变量的初值，表达式2的值为步长，表达式3的值为循环变量的终值。步长为1时，表达式2可以省略

## 循环结构-示例

- 求解水仙花数：一个三位整数各位数字的立方和等于该数本身则称该数为水仙花数。输出全部水仙花数。

```
for m=100:999
    m1=fix(m/100);      %求m的百位数字
    m2=rem(fix(m/10),10); %求m的十位数字
    m3=rem(m,10);       %求m的个位数字
    if m==m1*m1*m1+m2^3+power(m3,3)
        disp(m)
    end
end
End
153 370 371 407
```

## 循环结构-示例

- 已知  $y = \sum_{i=1}^n \frac{1}{2i-1}$ ，当n=100时，求y的值。

```
y=0; n=100;
for i=1:n
    y=y+1/(2*i-1);
end
y
```

- 在实际MATLAB编程中，采用循环语句会降低其执行速度，所以前面的程序通常由下面的程序来代替：

```
n=100; i=1:2:2*n-1;
y=sum(1./i);
y
```

## 循环结构

### • while语句

- while循环将循环体中的语句重复执行不定次数，其循环次数通常是未知的，这是与for循环的根本区别

- 调用格式:

```
while 表达式
    循环体语句组
end
```

循环判断语句为由逻辑、关系及一般运算组成的表达式

执行过程为：若条件成立，则执行循环体语句，执行后再判断条件是否成立，如果不成立则跳出循环

## 循环结构-示例

从键盘输入若干个数据，输入0时结束，求这些数的平均值。

```
sum=0;
cnt=0;
val=input('Enter a number (end in 0):');
while (val~=0)
    sum=sum+val;
    cnt=cnt+1;
    val=input('Enter a number (end in 0):');
end
if (cnt > 0)
    mean=sum/cnt
end
```

## 循环结构

### • break语句和continue语句

- 与循环结构相关的语句还有break语句和continue语句。它们一般与if语句配合使用
- break语句用于终止循环的执行。当在循环体内执行到该语句时，程序将跳出循环，继续执行循环语句的下一语句
- continue语句控制跳过循环体中的某些语句。当在循环体内执行到该语句时，程序将跳过循环体中所有剩下的语句，继续下一次循环

## 循环结构-示例

### • 求[100, 200]之间第一个能被21整除的整数

```
for n=100:200
    if rem(n,21)~=0
        continue
    end
    break
end
n
```

## 循环结构-示例

### • 西拉古斯(Syracuse)猜想/角谷猜想

- 从1到n的任一个自然数, 只要反复进行下列运算, 最后的结果总是1
- 1) 如果n是偶数, 就除以2;
- 2) 如果n是奇数, 就乘以3加1;
- 这个问题大约是在二十世纪五十年代被提出来的。在西方它常被称为西拉古斯(Syracuse)猜想, 因为据说这个问题首先是在美国的西拉古斯大学被研究的; 而在东方, 这个问题由将它带到日本的日本数学家角谷静夫的名字命名, 被称作角谷猜想

## 循环结构-示例

### • 西拉古斯(Syracuse)猜想/角谷猜想

```
n=input('请输入一个大于1的正整数 n=');
if n<=0
    disp('输入的数为负数或零, 程序中断! '); break;
end
while n>1
    if rem(n,2)==0
        n=n/2
    else
        n=n*3+1
    end
end
```

## 循环结构-示例

完数: 若一个数等于它的各个真因子之和, 则称该数为完数, 如 $6=1+2+3$ , 所以6是完数。求[1,500]之间的全部完数

```
for m=1:500
    s=0;
    for k=1:m/2
        if rem(m, k)==0
            s=s+k;
        end
    end
    if m==s
        disp(m)
    end
end
```

6 28 496

## 循环结构-示例

### • 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
  - 如猜测的数大于产生的数, 则显示“您的数字较大!”, 小于则显示“您的数字较小!”;
  - 等于则显示“YOU WIN!”, 同时退出游戏
  - 用户最多可以猜6次



## 循环结构-示例

### • 猜数游戏

```
y=round(10+89*rand());    %产生一个两位随机数
for k=1:6;
    x=input(['第',num2str(k),'次输入一个两位数(输完请按回车):']);
    if(x<y)
        '您的数字较小!'
    else if(x==y)
        msgbox('YOU WIN!');    return;
    else
        '您的数字较大!'
    end
end
if(k==6)    msgbox('YOU LOSE!? GAME OVER!')
end
end
```

## try-catch语句

### • 作用:

- 用于捕捉程序运行过程中的错误（不希望发生的结果），并对错误做适当处理，确保程序可靠运行

### • 调用格式1:

- try-catch语句块

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

- try语句执行语句组1，如执行过程中出现错误，则将错误信息赋给保留的lasterr变量，并转执行语句组2

## try-catch语句-示例

矩阵乘法运算要求两矩阵的维数相容，否则会出错。  
先求两矩阵的乘积，若出错，则自动转去求两矩阵的点乘。

```
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    %显示出错原因
```

```
C =    7    16    27
     40    55    72
```

```
ans = Error using *
       Inner matrix dimensions must agree.
```

## try-catch语句

### • 调用格式2:

- 嵌套try-catch块

- try-catch块也是能够嵌套的。下面的嵌套模式，是想从第一个try块产生的错误中恢复程序的正常运行:

```
try
    程序语句1
catch
    try
        程序语句2
    catch
        disp '操作失败'
    end
end
```



## 内容提要

- MATLAB控制语句
- M文件
  - 脚本文件
  - 函数文件
- 程序的调试与优化

## M文件

- M文件因其扩展名为.m而得名，是一个标准的文本文件，可在任何文本编辑器中进行编辑、存储、修改和读取,常用且最为方便的是使用默认的meditor编辑器
- M文件的语法类似于一般的高级语言，是一种程序化的编程语言，但又比一般的高级语言简单、直观，且程序易调试、交互性强
- MATLAB在初始运行M文件时会将其代码装入内存，再次运行该文件时会直接从内存中取出代码运行，因此会大大加快程序的运行速度
- 主要包括函数（Functions）和脚本（Scripts）两种

## M文件

- 建立新的M文件-三种方法（不同版本略有不同）
  - 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
  - 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
  - 命令按钮操作：单击主窗口工具栏上New Script命令按钮，启动Matlab文本编辑器后，输入M文件的内容并存盘

## M文件

- 打开已有M文件-三种方法（不同版本略有不同）
  - 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
  - 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
  - 命令按钮操作：单击主窗口工具栏上的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

## 脚本/命令文件

- 定义：Matlab中**既不接受输入参数也不返回输出参数的M文件**称为脚本。这种M文件是在Matlab的工作空间内对数据进行操作的
- 运行/执行：当用户在MATLAB中调用一个脚本文件时，Matlab将执行该脚本文件中所有可识别的命令。脚本文件不仅能够对工作空间内已经存在的变量进行操作，**还能够使用这些变量创建新的数据**
- 注意：尽管脚本文件不能返回输出参数，但其建立的新的变量却能够保存在Matlab的工作空间中，并且能够在之后的计算中被使用。除此之外，脚本文件还能够使用Matlab的绘图函数来产生图形输出结果

## 脚本文件

- 注意：
  - 脚本文件中的变量都为**全局变量**，程序运行后，这些变量保存在Matlab的基本工作空间内，一般采用函数clear清除这些变量
  - 为了避免因为变量名相同引起冲突，一般在脚本文件的开始，都采用函数clear all，清除所有基本空间中的变量

## 函数文件

- 为实现特定功能而定义的供其他程序调用的M文件
- 第一行一般都以functions开始
- 各种Matlab函数或命令实际上都是函数文件
- 函数M文件在执行的过程中，所产生的变量一般都是局部变量，存放在函数自身的函数工作空间中，不会和基本工作空间中的变量产生冲突
- 对用户来说，函数M文件就像一个黑匣子，只有输入和输出
- 采用函数M文件，非常易于实现程序的模块化，可以实现程序的分工合作、共同开发，适合大型程序开发

## 函数文件

- 函数文件组成结构：
  - function 返回变量=函数名(输入变量) %**
  - %帮助文档语句段**
  - %注释说明语句段**
  - 程序语句段**
- 注：
  - 帮助文档语句段就是**help 函数名**时对函数的说明内容
  - 具体指：**第一个非空行前的所有注释**

## 函数文件

- 与命令文件的主要区别
  - **参数**: 命令文件没有输入参数, 也不返回输出参数, 而函数文件可以带输入参数, 也可以返回输出参数。
  - **变量**: 命令文件对Matlab工作空间中的变量进行操作文件中所有命令的执行结果也完全返回工作空间中, 而函数文件中定义的变量为局部变量, 当函数文件执行完毕时, 这些变量被清除
  - **运行**: 命令文件可以直接运行, 在Matlab命令窗口输入命令文件的名字, 就会顺序执行命令文件中的命令, 而函数文件不能直接运行, 要以函数调用的方式来调用

## M文件-示例

- 分别建立命令文件和函数文件, 将华氏温度f转换为摄氏温度c
- **命令文件**:
  - 首先建立命令文件并以文件名f2c.m存盘
  - `clear all;`      %清除工作空间中的变量
  - `f=input('Input Fahrenheit temperature: ');`
  - `c=5*(f-32)/9`
  - 在命令窗口中输入f2c, 将会执行该命令文件, 执行情况为:  
Input Fahrenheit temperature: 70  
c = 21.1111

## M文件-示例

- 分别建立命令文件和函数文件, 将华氏温度f转换为摄氏温度c
- **函数文件**:
  - 首先建立函数文件f2c.m
  - `function c=f2c(f)`
  - `c=5*(f-32)/9;`
  - 然后命令窗口调用该函数文件
  - `clear;`
  - `y=input('Input Fahrenheit temperature: ');`
  - `x=f2c(y)`
  - 输出情况为:  
Input Fahrenheit temperature: 70  
x = 21.1111

## 变量的检测传递

- 输入/输出宗量检测命令
  - **nargin**: 在函数体内, 用于获取实际输入变量数
  - **nargout**: 在函数体内, 用于获取实际输出变量数
  - **nargin('fun')**: 获取'fun'指定函数的标称输入变量数
  - **nargout('fun')**: 获取'fun'指定函数的标称输出变量数
  - **inputname(n)**: 在函数体内使用, 给出第n个输入变量数的实际调用变量名

## 变量的检测传递-示例

- 需求：当调用过程小于或等于一个变量时，系统提示错误的输入，当有两个变量时，程序将两个数相加，当有3个变量时，将前两个数相加并减去第3个

```
function d=nargintest(a,b,c)
if nargin<=1
    error('Not enough input arguments')
elseif nargin==2
    d=a+b;
elseif nargin==3
    d=a+b-c;
end
```

## 变量的检测传递-示例

- **inputname(n)**: 在函数体内使用，给出第n个输入变量的实际调用变量名

- 示例:

M函数定义如下:

```
function y = myfun(a,b)
    disp(sprintf('My first input is "%s".',inputname(1)))
    disp(sprintf('My second input is "%s".',inputname(2)))
y = a+b;
```

赋值运行: `x = 5; myfun(x,5)`

My first input is "x".

My second input is "".

## 变量的检测传递

- “变长度”输入/输出变量

- MATLAB中有相当一部分函数都具有接受任意多输入、返回任意多输出的能力
- 为了使用户的自定义函数也可以具备这种能力，Matlab提供了如下两个内置函数
- **varargin**: 变长度输入变量列表
- **varargout**: 变长度输出变量列表

## 变量的检测传递-示例

- 先编写varargTest函数文件

```
function varargout = varargTest(varargin)
fprintf('How many output arguments? %d\nAnd they are: \n',
    nargout);
for k=1:nargout
    varargout(k) = varargin(k); % the same as {varargin{k}};
    fprintf('%s ', num2str(varargout{k}));
end
disp(' ');
end
```

## 变量的检测传递-示例

- 调用

```
>>[y1,y2]=varargTest(1,2,3)
```

How many output arguments? 2

And they are:

1 2

y1 = 1

y2 = 2

```
>> [y1]=varargTest(1,2)
```

How many output arguments? 1

And they are:

1

y1 = 1

## 变量类型

- 局部变量 (Local)：局部变量是在函数中定义的变量

。每个函数都有自己的局部变量，只能被定义它的函数访问。当函数运行时，其变量保存在自己的工作区中，一旦函数退出运行，内存中变量将不复存在

- 全局变量 (Global)：多和函数可共享的变量；使用global函数定义

```
>> clear all; %清除原先用户自定义的变量
```

```
>> global MAXLEN; MAXLEN=45;
```

- 永久变量 (Persistent)：类似于static变量；在M文件中定义和使用，永久保存，只有在clear时被清除；使用persistent函数定义

```
>> persistent MAXLEN; MAXLEN=45;
```

## 变量类型

- 注意：全局变量需使用global函数定义，且在任何使用该全局变量的函数中都应加以定义，包括命令窗口

- 示例：建立函数文件wadd.m，实现输入参数的加权相加

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

```
global ALPHA BETA
```

```
f=ALPHA*x+BETA*y;
```

– 在命令窗口中输入：

```
global ALPHA BETA
```

```
ALPHA=1; BETA=2;
```

```
s=wadd(1,2)
```

## 内容提要

- MATLAB控制语句

- M文件

- 程序的调试与优化

- 错误种类
- 断点调试
- 程序优化

## 程序的错误种类

- 语法错误

- 语法错误发生在M文件程序代码的生成过程中，一般是由函数参数输入类型有误或是矩阵运算阶数不符等引起

```
>> min(x,y)
```

```
??? Undefined function or variable 'x'.
```

```
>> A=ones(3); B=magic(4); D=A*B
```

```
??? Error using ==> mtimes
```

```
Inner matrix dimensions must agree.
```

- 运行错误

- 指在程序运行过程中，出现溢出或是死循环等异常现象

## 简单错误识别和调试

- 将函数中输出关键值的行的分号(;)去掉，这样，这些运算的**中间结果**将在命令窗口中予以**显示**，用户可以据此来检查中间结果的正确性
- 在函数中添加一些语句，用来**显示用户认为重要变量值**
- 使用**keyboard**命令**中断**程序，实现函数工作区间和命令窗口工作区间的交互，从而获得用户所信息；使用该命令后，程序将处于调试状态，命令窗口提示符由“>>”变为“K>>”，用户可以进行相应的操作；使用**return**命令退出
- 在函数头前加“%”，这样**函数式M文件变为脚本式M文件**，而脚本式M文件运行时，其工作区间就是MATLAB的工作区间，在出现错误的时候就可以查询这个工作区间

## 设置断点

- 用途：

- 断点是指让运行的程序停在用户指定地方；设置断点是查找程序错误的一种方法。在可能出现问题的程序处设置断点，即可让运行程序停在断点处，以便检验变量值

- 标准断点

- 在M文件的指定行设置的断点，程序至此就停止

- 条件断点


- 在M文件的指定行设置的断点，但是为断点规定了一个条件，满足该条件，运行的程序才停止

- 错误断点

- 为指定类型的警告、错误、NaN或无限大值设置的断
- 任何M文件只要产生了这样的问题，运行的程序就停止

## 设置标准断点

- 在Editor/Debugger窗口设置断点

- MATLAB的Editor窗口中，行号的右边有专门放置断点的树形条，称为断点小径。未设置断点时，有些行号后被小横线占据，表示该行是可执行语句，设置标准断点后，变成小红点，即标准断点图标。设置断点的方法有
  - 单击工具栏中的图标 
  - 选择Editor窗口Debug菜单下Set/Clear Breakpoint命令
  - **在要设置断点的行的小横线上单击**

- 有效和无效断点

- 在某种情况下，断点变为灰色，就成了无效断点。无效断点不是用命令制造的。当修改了程序的内容而不保存时，有效断点（红色）就变成无效断点

## 设置标准断点

### • 在命令窗口设置断点

- 在Matlab中提供了dbstop函数设置断点，调用格式如下
  - **dbstop in mfile ...**: 在名为mfile的M文件上设置断点。运行文件前，先执行该函数。程序暂停在第一行可执行语句上，并被置于debug方式。其后可用dbcont或dbstep重新运行，也可用dbquit退出debug方式
  - **dbstop in mfile at lineno**: 在M文件mfile上设置断点，断点在lineno可执行语句上。程序暂停在lineno行，且被置于debug方式。其后可用dbcont或dbstep重新运行，也可用dbquit退出debug方式

## 标准断点

### • 清除断点

- 在设置的断点行的小红点上单击
- 在Debugger窗口清除断点:直接断点行，选择Debug菜单下的Set/Clear Breakpoint命令；或用鼠标右键单击断点，在弹出的快捷菜单中选择Clear Breakpoint命令。
- 利用命令函数清除断点：
  - **dbclear all**: 清除所有文件上的所有断点
  - **dbclear in mfile**: 清除指定文件mfile上的所有断点
  - **dbclear in mfile at lineno**: 清除指定文件mfile中指定行lineno上的断点

### • 显示断点状况:

- 在MATLAB中提供了dbstatus函数显示断点的情况

## 设置条件断点

- 如果给一个文件设置了条件断点(小黄点)，而运行时正好满足了该条件，那么文件将停在指定行。条件断点最好的用途是检验循环中特定迭代后的结果
  - 在Debugger窗口设置条件断点
  - 使用快捷菜单设置条件断点
  - 使用函数设置条件断点
- 在MATLAB中提供了dbstop函数来设置条件断点，其调用格式如下
  - **dbstop in FILESPEC if EXPRESSION**
  - **dbstop in FILESPEC at LINENO if EXPRESSION**
  - **dbstop if error**
  - ...

## 设置错误断点

### • 设置错误断点

- **dbstop if error**: 其后运行的任何M文件只要产生了运行错误，程序就停在生成错误的行（但不包括被try-catch语句检测到的错误），并被置于debug方式，且不能再运行，只能用dbquit退出debug方式
- **dbstop if warning**: 其后运行的任何M文件只要产生了运行时警告，程序就暂停在生成警告的行，并被置于debug方式，可以用dbcont或dbstep重新运行
- **dbstop if naninf/infnan**: 其后运行的M文件只要遇到无限大数（Inf）和非数值（NaN）时，程序就暂停在生成Inf和NaN的行，并给出警告信息，程序进入debug方式



## 清除错误断点

### • 清除错误断点

- **dbclear if error:** 清除用dbstop if error和dbstop if error identifier命令设置的断点。
- **dbclear if warning:** 清除用dbstop if warning和dbstop if warning identifier命令设置的断点
- **dbclear if naninf:** 清除用dbstop if naninf命令设置的断点
- **dbclear if infnan:** 清除dbstop if infnan命令设置的断点

## 报告错误

### • error函数

- 在MATLAB中提供了error函数用来报告错误，并停止程序执行。在程序代码中，error函数的输入是指定要报告的错误信息，其调用格式如下

– **error('not enough inputs!')**

### • lasterr函数:

- 能够帮助识别最近Matlab生成的错误，其调用格式如下
- **msgstr = lasterr:** msgstr为错误信息变量;lasterr函数返回最近被MATLAB生成的错误信息到变量msgstr中

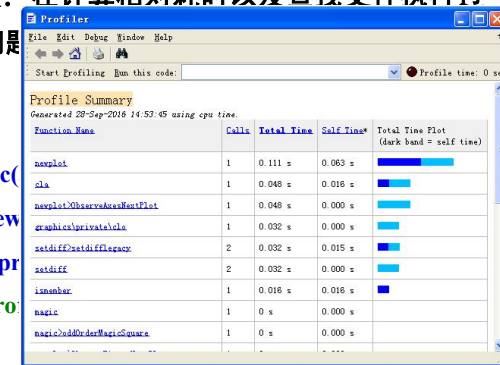
## 程序优劣的分析

- 在MATLAB中，使用profile函数以及计时函数tic和toc来分析程序中各个部分的耗时情况，从而帮助用户找出程序中需要改进的地方
- **Profile函数:** 在计算相对耗时以及查找文件执行过程中瓶颈问题时更为有效
- **tic和toc函数:** 在计算绝对耗时时更为有效

## 程序优劣的分析

- **Profile函数:** 在计算相对耗时以及查找文件执行过程中瓶颈问题时更为有效

>> profile on  
>> plot(magic)  
>> profile view  
>> profsave(profile)  
%save profile



## 程序优劣的分析

- tic和toc函数：在计算绝对耗时更为有效

```
>> tic
>> plot(magic(35))
>> toc

Elapsed time is 6.023628 seconds.
```

## 程序优化的技巧

- 程序的向量化操作
  - 循环运算是Matlab中的最大弱点，在程序设计中，应当尽量避免使用循环运算
  - 用户可以通过将M文件向量化来优化M文件，所谓向量化就是使用向量和矩阵运算来代替for循环和while循环
- 数据的预定义
  - 使用for循环和while循环来增加数据结构的大小时，将影响系统和内存的使用
  - 提高效率的可行办法是进行预定义

## 程序优化的技巧

- 程序的向量化 – 示例：无穷级数求和  $S = \sum_{i=1}^{+\infty} \left( \frac{1}{2^i} + \frac{1}{3^i} \right)$
- 一般方法：

```
tic;
s=0;
for i=1:100000
    s=s+(1/2^i+1/3^i);
end
toc
s = 1.5000
elapsed_time = 1.9700
```
- 向量化方法

```
tic;
i=1:100000;
s=sum(1./2.^i+1./3.^i);
toc;
s = 1.5000
elapsed_time = 0.3800
```

## 程序优化的技巧

- 数据的预定义 – 示例：生成一个5x10000的Hilbert矩阵，其第i行第j列元素为  $h(i,j)=1/(i+j-1)$
- 一般方法：

```
tic
for i=1:5
    for j=1:10000
        H(i,j)=1/(i+j-1);
    end
end
toc
elapsed_time = 8.6800
```
- 预定义方法

```
tic
H=zeros(5,10000);
for i=1:5
    for j=1:10000
        H(i,j)=1/(i+j-1);
    end
end
toc
elapsed_time = 1.0400
```

## 程序优化的技巧

- 数据的预定义 – 示例：生成一个 5x10000 的 Hilbert 矩阵，其第 i 行第 j 列元素为  $h(i,j)=1/(i+j-1)$

- 预定义+向量化

```
tic
H=zeros(5,10000); %预定义
for i=1:5          %向量化
    H(i,:) = 1./[i:i+9999];
end
toc
elapsed_time = 0.060
```

## 课外实验二

- 实验名称:

– 用户注册与验证

- 实验目的:

– 通过实验使得学生巩固 MATLAB 程序设计知识，熟练运用输入输出、控制结构、错误调试等重点难点内容

- 实验内容:

– 设计开发一个用户注册和验证程序

## 课外实验二

- 要求-用户注册

- 提示用户（不少于10个）输入正确的注册信息（符合以下规则），并以结构体数组形式存储（RegInfo）
- 用户名：不少于8位，必须包含英文字母（区分大小写）和数字且以字母开头，不得包含空格和标点；不得与已有用户名重复
- 密码：同上，且不得与用户名重复，或包含或被包含于用户名
- 姓名：必须为2~4个中文汉字
- 性别：男或者女

## 课外实验二

- 要求-用户验证

- 提示用户输入用户名和密码信息
- 核实用户名和密码信息
- 最多输入和验证3次
- 验证成功后给出提示，并输出用户姓名和性别
- 验证错误后，提示（具体是用户名还是密码）错误和剩余次数