

宁波大学信息科学与工程学院

《MATLAB 与数值计算》

MATLAB 实验手册

2020-08

备注：

建议课时安排：四章 MATLAB 基础实验 12-14 课时、八章数值计算程序设计实验 16 课时、二章综合练习实验 4-6 课时。总计 32-34 课时

MATLAB 版本：MATLAB 7.0（32 位）

目录

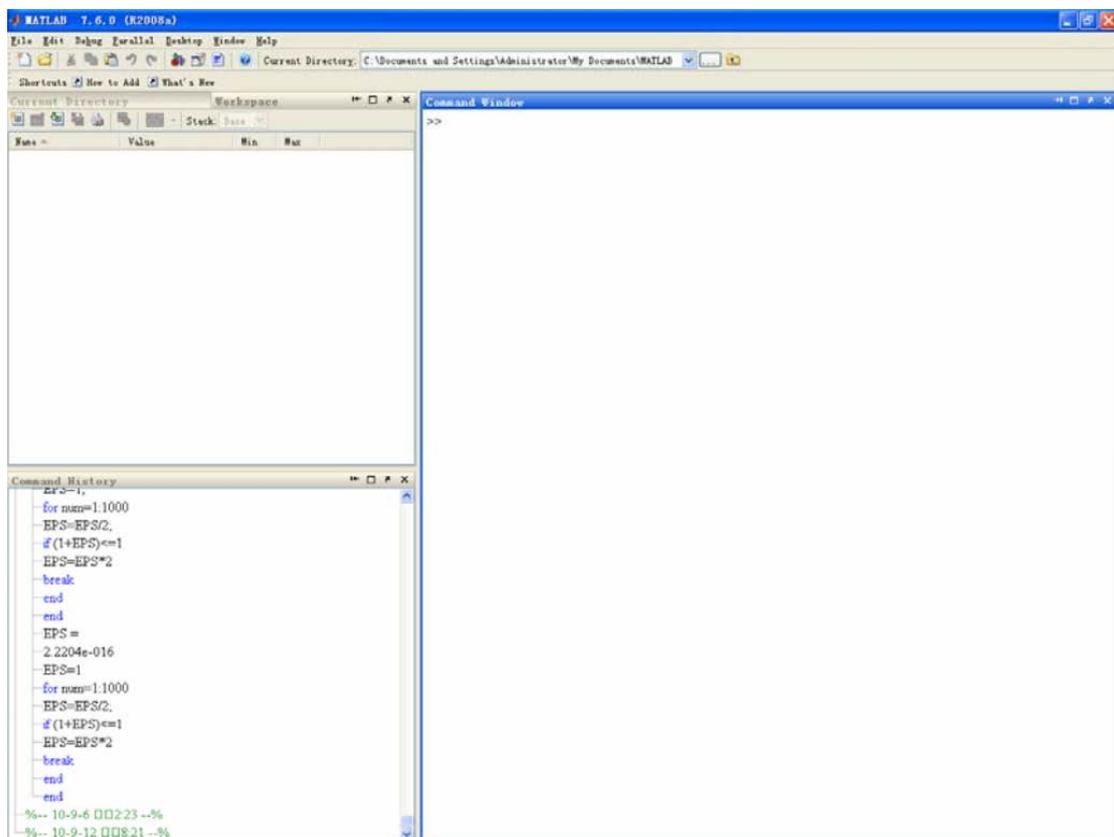
一、MATLAB 基础.....	3
1.1 运行环境.....	3
实验题 Q1.1 : 1.4.....	6
1.2 帮助.....	6
实验题 Q1.5 : 1.6.....	7
1.3 简单运算.....	8
实验题 Q1.7.....	9
1.4 数组与矩阵运算.....	9
实验题 Q1.8 : 1.10.....	14
1.5 内置函数.....	15
实验题 Q1.11.....	15
二、MATLAB 程序设计.....	16
2.1 脚本编辑器.....	16
实验题 Q2.1 : 2.3.....	17
2.2 脚本语法语句.....	19
实验题 Q2.4 : 2.5.....	21
2.3 多函数脚本程序开发样例.....	22
实验题 Q2.6.....	23
三、MATLAB 简单数据分析.....	24
3.1 简单统计.....	24
实验题 Q3.1.....	25
3.2 条形分布图.....	25
实验题 Q3.2.....	26
3.3 随机数.....	27
实验题 Q3.3.....	28
3.4 多项式.....	29
实验题 Q3.4 : 3.8.....	32
四、MATLAB 图形工具.....	33
4.1 二维图形曲线.....	33
4.2 图形窗口操作.....	36
4.3 三维图形绘制.....	37
4.4 实验题 Q4.1 : 4.4.....	40
五、非线性方程求根.....	41
Q5.1 二分法与迭代法.....	41
Q5.2 牛顿法及混合算法.....	42
六、线性方程组求根--消元法.....	44
Q6.1 消元法.....	44
七、线性方程组求根—迭代法.....	46
Q7.1 迭代法求解线性方程组.....	46
八、矩阵特征值求解.....	50

Q8.1 幂法、原点平移法、反幂法.....	50
九、多项式插值.....	52
Q9.1 朗格朗日插值、牛顿插值.....	52
十、最小二乘拟合与样条插值.....	54
Q10.1 基础练习.....	54
Q10.2 增强练习.....	54
十一、数值积分.....	55
Q11.1 牛顿科特斯积分.....	55
Q11.2 复化积分.....	55
Q11.3 龙贝格积分.....	57
十二、常微分方程求解.....	59
Q12.1 欧拉法.....	59
Q12.2 龙格库塔法.....	61
十三、综合练习（一）.....	62
十四、综合练习（二）.....	64
Q14.1 迭代思维编程.....	70
附：实验报告样例.....	71

一、MATLAB 基础

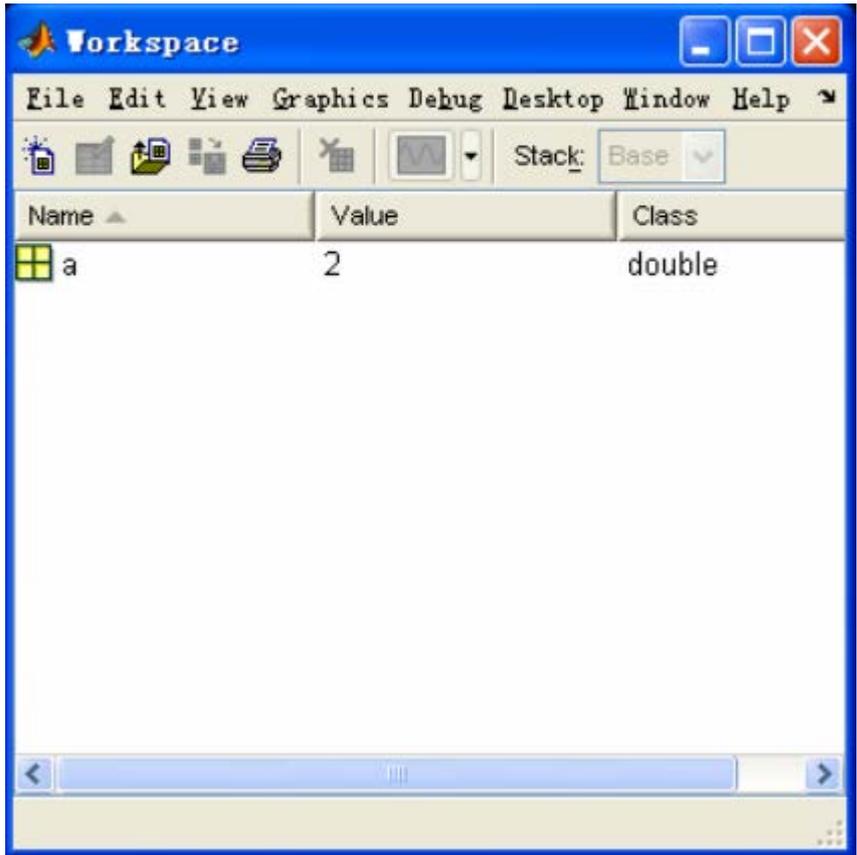
1.1 运行环境

安装好 MATLAB 之后，一般在桌面上或开始菜单里面可以找到 MATLAB 图标。 双击启动后，会出现 MATLAB 主界面。



其中从上往下、从左往右分别是：菜单栏、工具栏、工作空间窗口（Workspace）、历史指令窗口（Command History）、指令窗口（Command Window）。

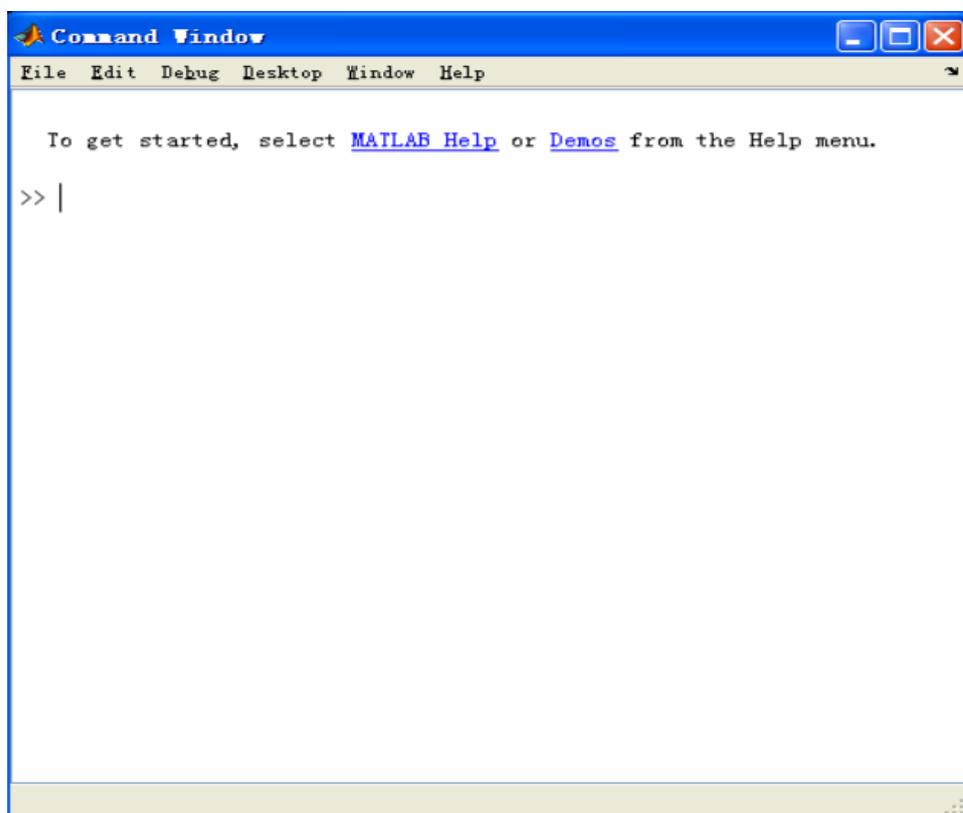
工作空间窗口（Workspace）列出当前 MATLAB 平台中有效的变量名称、维度、大小等信息，可以在程序运行跟踪时（或指令窗口 Command Window 调试时），通过该窗口观察每个变量的取值。双击每个变量就会弹出另一个小窗口显示该变量的具体数值。可以对变量的取值进行编辑修改、保存、绘制曲线等操作。



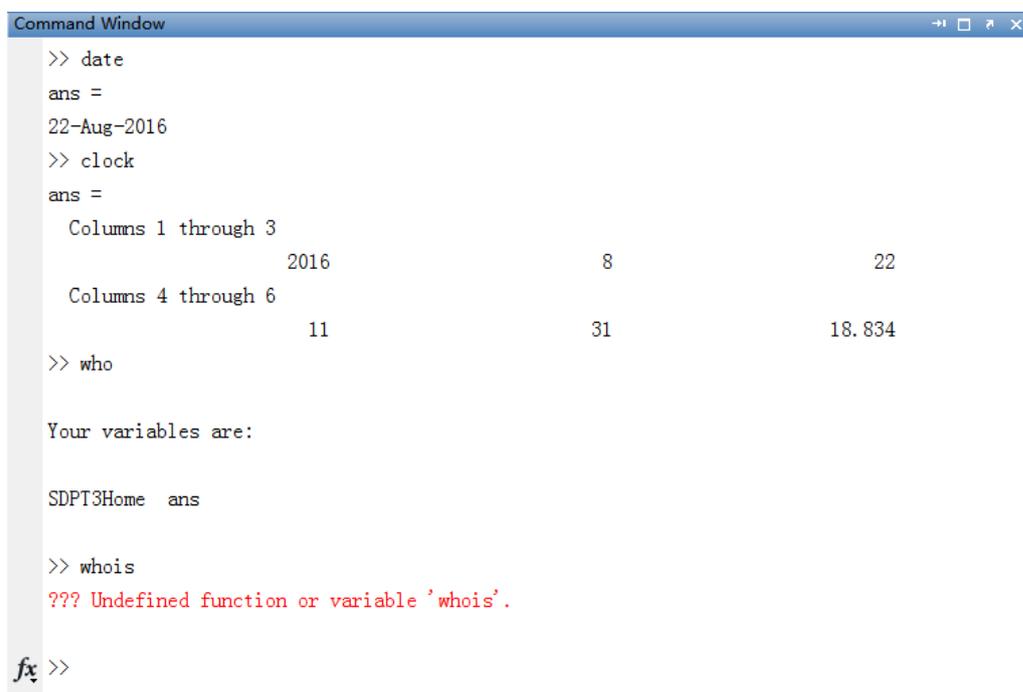
历史指令窗口（Command History）会自动记录用户在 Command Window 中写下的任何一条指令（或脚本），供用户回忆和重新使用。在该窗口中选中一条指令，鼠标双击后，将会自动复制到 Command Window 中进行执行。也可以把指令用鼠标拖拉到 Command Window 中进行修改后再执行。



指令窗口（Command Window）是 MATLAB 最重要的窗口。所有的调试性、临时性的工作都在这里完成。用户通过输入脚本语句或指令，MATLAB 把执行后的反馈信息显示在语句下方。这个窗口中的编辑方式是“**行编辑**”，语句一旦被执行过，将不能被更改。只要复制后再次修改执行。



现在输入若干条语句试试看。输入“date”，按回车，会返回当前日期。



输入“who”，会返回当前有哪些活跃变量。输入“whois”，MATLAB 用**红色字体**提示没有找到有函数或变量的名字叫做“whois”，表明输入了错误的指令或语句。

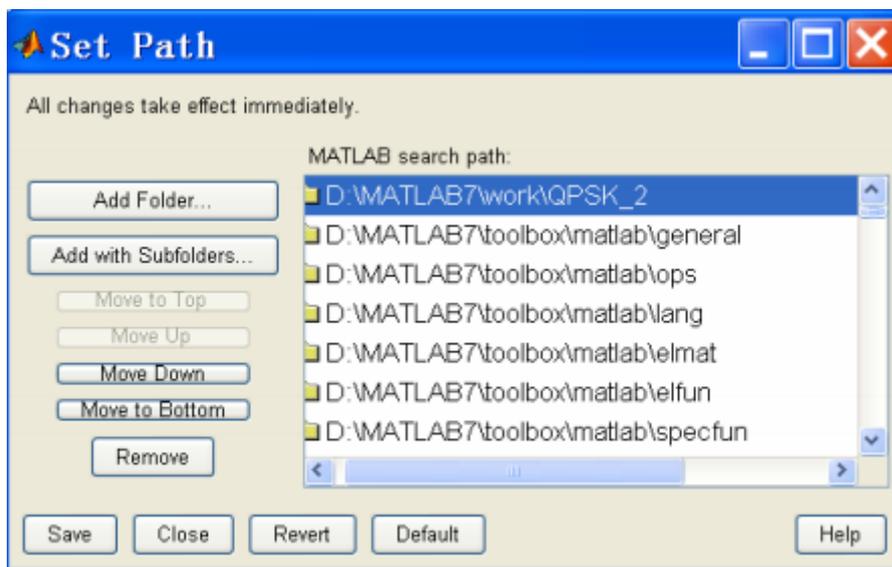
实验题 Q1.1 : 1.4

Q1.1 启动 MATLAB，找到工具栏上的“Current Directory”输入框，把当前目录（Current Directory）修改成“C:\Program Files”，然后找到“Current Directory”窗口，观察并记录里面有哪些文件夹和子文件夹。

Q1.2 在“Command Window”窗口中输入“version”，按回车，记录返回的信息。接着输入如下指令和语句，分别记录下来。

```
format long、exp(1)、pi、format short、eps、Inf、tic、toc、clc  
cd、dir、cd c:\、cd windows、clear、quit
```

Q1.3 重新启动 MATLAB，在菜单栏找到“File | Set Path”，弹出如下界面：



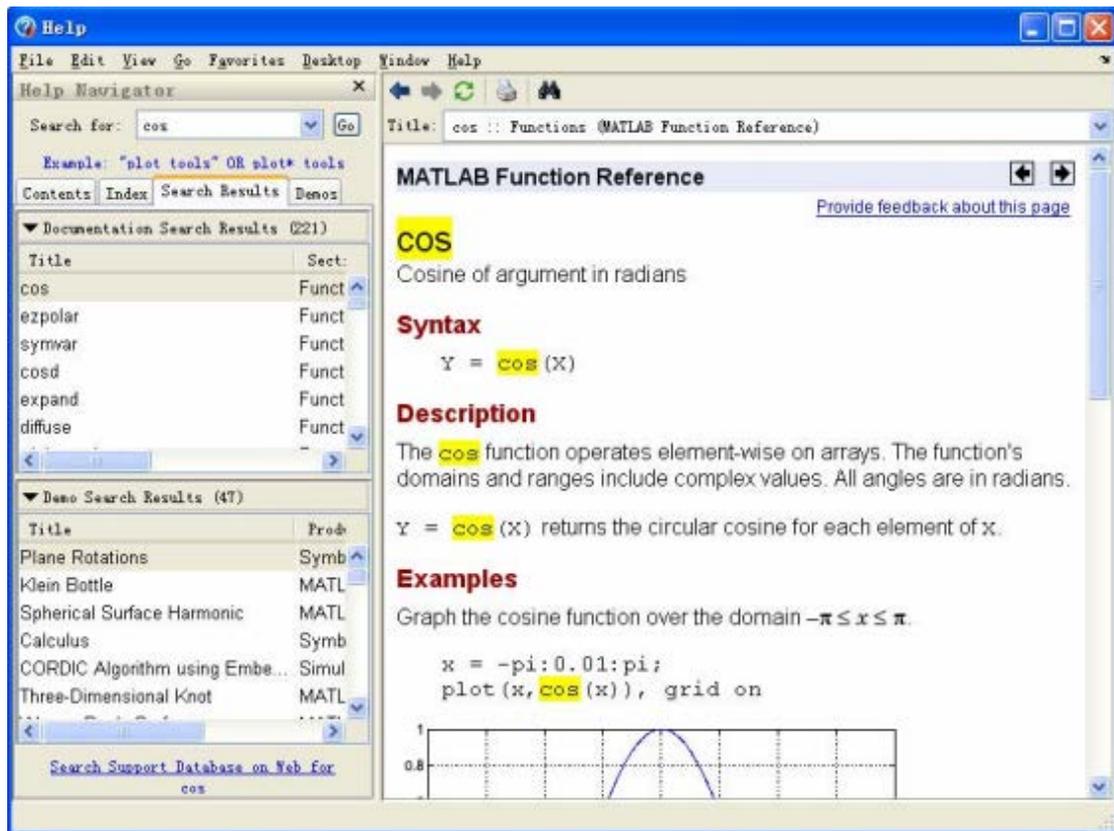
随便找一个文件夹添加到 MATLAB 搜索路径中，测试“Add Folder”、“Add with Subfolders”、“Remove”等功能。退出该界面。

Q1.4 在 MATLAB 菜单栏中找到“File | Preferences”，弹出一个选项设置界面。在该界面中可以设置 MATLAB 的许多选项。要求把字体大小设成 18、把 Command Window 的 Numeric format 设成“long e”。

1.2 帮助

在 command window 中输入“**help**”指令可以看到帮助信息。help 指令格式：
help TopicName 列出某一关键词或函数组的帮助信息
help FunctionName 列出某一函数的帮助信息

还可以在 command window 中输入 **doc** 指令（例如 doc max），会弹出 MATLAB 帮助与演示界面。该窗口以图文并茂的详细形式显示每一个函数、变量、关键词等的内容、样例等，供用户编写脚本时参考。也提供所有工具箱的样例与演示程序，用户可以看演示代码轻松学会工具箱函数的使用。



实验题 Q1.5 : 1.6

Q1.5 在 Command Window 中输入“help sin”按回车，看懂里面的意思；接着在 Command Window 中输入“doc sin”，在弹出的帮助界面窗口中找到“Examples”，可以看到有两条语句。把这两条语句输入到 Command Window，看看是否会出来 sin 函数的曲线图？

Q1.6 查看求最大值函数（max）的帮助信息，并在 doc 界面找到它的“Examples”，输入样例语句到 Command Window 中，记录反馈的信息。

1.3 简单运算

常用预定义的常量如下表：

预定义变量	含 义	预定义变量	含 义
ans	计算结果的默认变量名	NaN 或 nan	不是一个数 (Not a Number), 如 0/0, ∞/∞
eps	机器零阈值	nargin	函数输入宗量数目
Inf 或 inf	无穷大, 如 1/0	nargout	函数输出宗量数目
i 或 j	虚 单 元 $i = j = \sqrt{-1}$	realmax	最大正实数
pi	圆周率 π	realmin	最小正实数

MATLAB 中所有的变量只有一种类型：**矩阵**。在矩阵中元素的类型有 double、char、cell、object、unit8 等。一般情况下，MATLAB 的变量在使用前不需要申明数据类型，系统会自动强制转换。

```

Command Window
>> 1+2

ans =

    3

>> 2*sin(10*pi)+exp(-1)^2

ans =

    0.1353
    
```

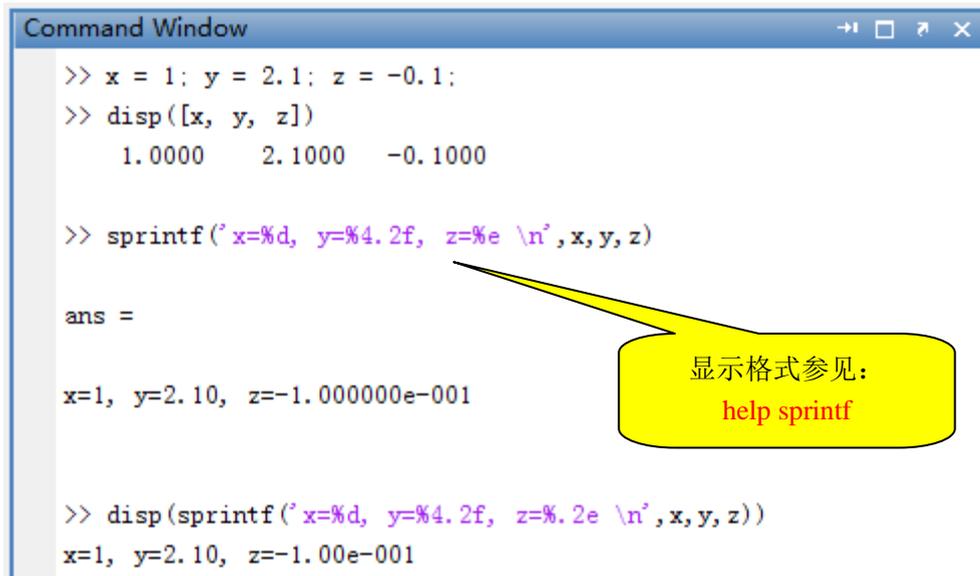
```

Command Window
>> x = 1;
>> if (x==1), y = 0;
else y =1;
end;
>> y

y =

    0
    
```

关系运算符有：
==, >, <, >=, <=, ~=, ~



```
Command Window
>> x = 1; y = 2.1; z = -0.1;
>> disp([x, y, z])
    1.0000    2.1000   -0.1000

>> sprintf('x=%d, y=%4.2f, z=%e \n', x, y, z)

ans =

x=1, y=2.10, z=-1.000000e-001

>> disp(sprintf('x=%d, y=%4.2f, z=%0.2e \n', x, y, z))
x=1, y=2.10, z=-1.00e-001
```

显示格式参见：
[help sprintf](#)

运行上述的语句，可以看到 MATLAB 脚本非常简单。

实验题 Q1.7

Q1.7 有一笔钱存入余额宝，假定余额宝的年利率是 2.8%，那么问 2 万元存入，10 年后可以拿到多少利息？在 MATLAB 的 Command Window 中简单计算一下。

提示：利息计算公式 $P = X * ((1+R)^Y - 1)$

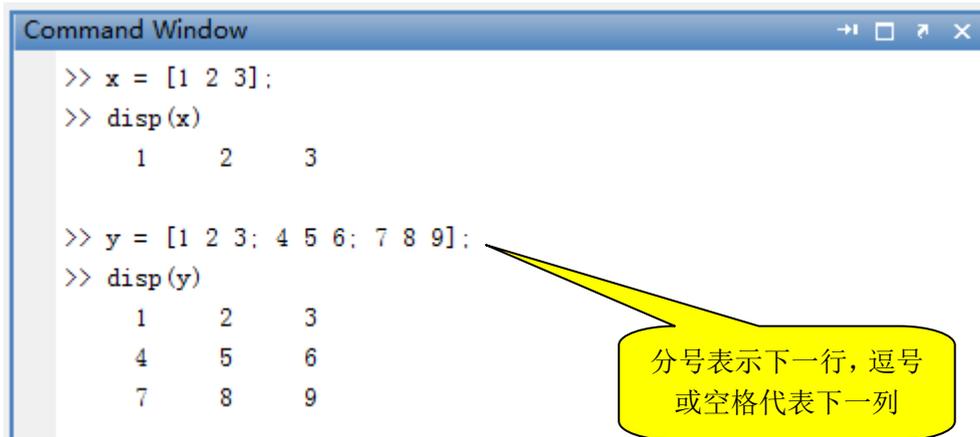
1.4 数组与矩阵运算

MATLAB 中数组看成是单行或单列的矩阵，因此两者的运算是相同的。MATLAB 是并行语言，其大部分函数、语句（赋值语句、算术运算语句、关系判断语句等）都是并行运算的，也就是说，只需要一条语句就可以对矩阵中所有的元素做批量的相同运算，而不需要象 C 语言一样用 For 循环来运算每个元素。在多核 CPU 或向量计算机中，MATLAB 的运行效率会很高，但前提要求是编程者遵循并行语言的开发规范、以及算法符合并行性。

1) 数组和矩阵的创建

创建一个数组或矩阵有三种以上的方法。分别是用 “[]”、“:” 和 “**linspace** 函数” 等。这三种方法也可以混合在一起共同完成数组或矩阵的创建。例子如下：

方法 1: 采用符号 “[]”

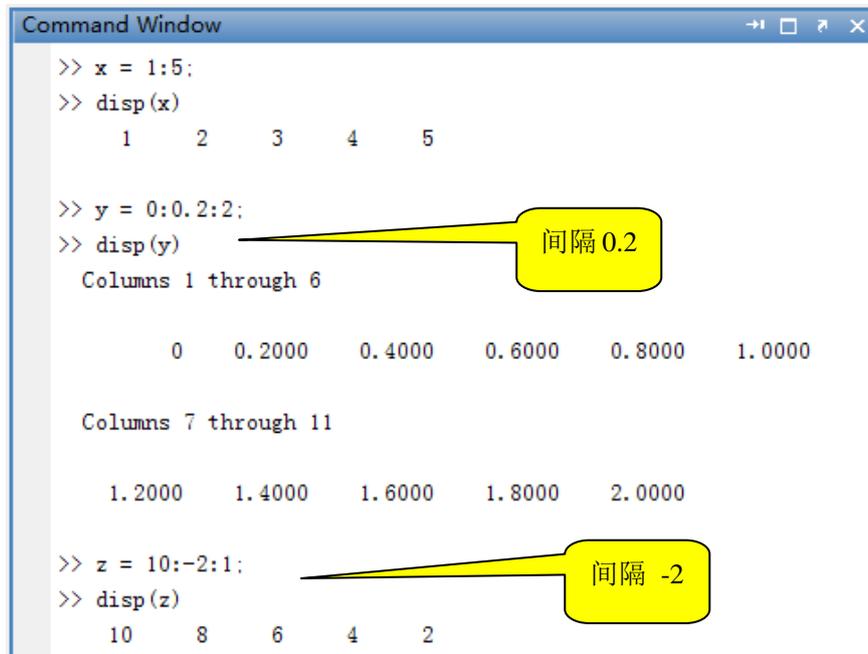


```
Command Window
>> x = [1 2 3];
>> disp(x)
    1    2    3

>> y = [1 2 3; 4 5 6; 7 8 9];
>> disp(y)
    1    2    3
    4    5    6
    7    8    9
```

分号表示下一行，逗号或空格代表下一列

方法 2: 采用符号 “:”



```
Command Window
>> x = 1:5;
>> disp(x)
    1    2    3    4    5

>> y = 0:0.2:2;
>> disp(y)
Columns 1 through 6
    0    0.2000    0.4000    0.6000    0.8000    1.0000

Columns 7 through 11
    1.2000    1.4000    1.6000    1.8000    2.0000

>> z = 10:-2:1;
>> disp(z)
    10    8    6    4    2
```

间隔 0.2

间隔 -2

用冒号创建数组或矩阵的基本格式为:

$$V = V_0 : Step : V_n$$

表示从 V_0 开始 (包含 V_0), 每间隔 $Step$ 的值创建一个数组元素, 直到 V_n 为止 (不一定包含 V_n)。例如: $x = 0:0.3:1$, 等价于 $x = [0, 0.3, 0.6, 0.9]$, 里面没有 1。(为什么?)

注: MATLAB 语句的每行最后用 “;” 代表语句结束, 这样的语句的运行结果将不会显示在 Command Window 中, 可以在 Workspace 中双击变量名看到变量的值。

如果某条语句的最后没有 “;”, 也是正确的 (跟 C 语言不一样)。但是该语句的运行结果会 **自动显示** 到 Command Window 中。

因此, 语句最后的 “;” 控制着运行结果是否要显示在 Command Window 中。切记!

方法 3: 采用特殊函数 (linspace, zeros, ones, eye)

```
Command Window
>> x = linspace(0, 1, 4); disp(x)
    0    0.3333    0.6667    1.0000

>> y = zeros(2); disp(y)
    0    0
    0    0

>> z = zeros(2, 3); disp(z)
    0    0    0
    0    0    0

>> u = ones(2, 1); disp(u)
    1
    1

>> v = eye(3); disp(v)
    1    0    0
    0    1    0
    0    0    1
```

在区间[0,1]中均匀取 4 个点

方法 4: 多方法混合

```
Command Window
>> x = [1:3, [8, 7]; linspace(2, 6, 5); [ones(1, 4), 10]];
>> disp(x)
    1    2    3    8    7
    2    3    4    5    6
    1    1    1    1   10

>> y = 0;
>> for k=[2, 5, 3, 10:-1:3], y = y + k; end;
>> disp(y)
    62

>> z = [x, ones(3, 2)]; disp(z)
```

多方法混合要保持矩阵行列数一致的原则

1	2	3	8	7	1	1
2	3	4	5	6	1	1
1	1	1	1	10	1	1

2) 数组与矩阵的运算

取出数组（矩阵）中指定的元素

```
Command Window
>> x = 2:9; disp(x)
     2     3     4     5     6     7     8     9

>> y = x(3:5); %取出x中第3到5号的成员组成新数组赋值给y
>> disp(y)
     4     5     6

>> z = [1 2 3; 4 5 6; 7 8 9]; disp(z)
     1     2     3
     4     5     6
     7     8     9

>> u = z(1:2, 2:3); disp(u)
     2     3
     5     6

>> v = [u; z(3, 1:2)]; disp(v)
     2     3
     5     6
     7     8
```

通过符号“(”和“)”取出 z 矩阵中的第 1-2 行、第 2-3 列的元素组成新矩阵 u

两个矩阵的合并
注意行列数保持一致!

数组（矩阵）合并、拆分

```
Command Window
>> x = [1 2 3; 4 5 6]; disp(x)
     1     2     3
     4     5     6

>> y = ones(1,3); disp(y)
     1     1     1

>> z = [x; y]; disp(z); % x 和 y 合并成3行3列的矩阵
     1     2     3
     4     5     6
     1     1     1

>> u = [x, zeros(2,1)]; disp(u); % 合成成2行4列的矩阵
     1     2     3     0
     4     5     6     0

>> w1 = u(:, 1:2); w2 = u(:, 3:4);
>> disp(w1)
     1     2
     4     5

>> disp(w2)
     3     0
     6     0
```

单个“:”代表所有的标号。

数组（矩阵）算术运算（加、减、点乘、点除、点乘方）

```
Command Window
>> x = 1:5; y = 6:-1:2; z = x + y; disp(z)
     7     7     7     7     7

>> u = x - y; disp(u)
    -5    -3    -1     1     3

>> v = x .* y; disp(v)
     6    10    12    12    10

>> w = x .^2; disp(w)
     1     4     9    16    25

>> b = x ./ y; disp(b)
    0.1667    0.4000    0.7500    1.3333    2.5000

>> c = x .\ y; disp(c)
    6.0000    2.5000    1.3333    0.7500    0.4000
```

数组（矩阵）逻辑、关系运算

```
Command Window
>> x = 1:5; y = 6:-1:2; z = (x == y); disp(z)
     0     0     0     0     0

>> u = (x > y); disp(u)
     0     0     0     1     1

>> w = (x ~= y); disp(w)
     1     1     1     1     1

>> disp(x & y)
     1     1     1     1     1

>> disp(x | y)
     1     1     1     1     1

>> if (sum(x==y)), a=1; else a=0; end; disp(a)
     0
```

MATLAB 的关系与逻辑运算符包含：

大于 (>)、小于 (<)、大于等于 (>=)、小于等于 (<=)、等于 (==)、不等于 (~=)、逻辑与 (&)、逻辑或 (|)、逻辑非 (~)、关系与 (&&)、关系或 (||)、关系非 (~)。

矩阵乘法、除法、乘方运算

```
Command Window
>> A = [1 2 3;4 5 6; 7 3 1];
>> B = ones(3);
>> disp(A*B)
     6     6     6
    15    15    15
    11    11    11

>> disp(A/B)
    NaN    NaN    NaN
    NaN    NaN    NaN
    NaN    NaN    NaN

>> disp(A\B)
     1.5000     1.5000     1.5000
    -4.0000    -4.0000    -4.0000
     2.5000     2.5000     2.5000

>> disp(A^2)
     30     21     18
     66     51     48
     26     32     40
```

分母为奇异矩阵，除法结果为“NaN”（不定值）

注：MATLAB 的数组与矩阵运算中增加了点乘（.*）、点除（./ 或 .\）、点乘方（.^）运算。两个行列数相同的数组或矩阵可以用点乘等运算仅对数组或矩阵中的单个对应元素进行运算，而不遵循矩阵运算规则。

除法有两种，左除和右除。其中分母和分子要搞清楚。切记！

实验题 Q1.8 : 1.10

Q1.8 用函数 linspace 创建与如下一样的数组：

```
T = 4:6:35;
```

```
X = -4:2;
```

Q1.9 用冒号创建与如下一样的数组：

```
V = linspace(-2,1.5,8);
```

```
R = linspace(8,4,5,8);
```

Q1.10 写出如下语句的矩阵 A

```
A = [3 2 1; 0:0.5:1; linspace(6,8,3)]
```

并把 A 的第 1 行与第 3 列进行点乘，点乘后的结果取代 A 中的第 3 行，形成新的矩阵 C，写出 C。

注：单行数组与单列数组不能直接点乘，需要先进行转置（'）确保行列数一致。

1.5 内置函数

MATLAB 内置函数非常多，每个工具箱都有成百上千个函数。系统提供常用的基础函数如下：

length: 返回数组的元素个数或矩阵的大小中大的那个数值
isempty: 判断是否为空矩阵
isnan、**isinf**、**is***系列: 判断特殊值
size: 返回矩阵行数、列数（多维维度）
rand: 均匀分布随机数（0~1 之间）
randi: 均匀分布随机整数
randn: 高斯分布的随机数
randperm: 自然数的随机序列
awgn: 数组添加高斯噪声
sin、**cos**、**tan**、**atan**、**asin**、**acos**、**sinh**、**cosh**、**tanh** 等
exp、**log**、**log10**、**log2** 等
sqrt: 平方根
sum、**mean**、**median**、**cumsum**、**std** 等
max、**min** 等
floor: 取整数部分
inv、**rank**、**det**: 求矩阵的逆、秩、行列式
roots: 求多项式根
eig: 求矩阵的特征值与特征向量
norm、**cond**: 范数、条件数
zeros、**ones**、**eye**: 元素为 0、1 的矩阵，单位矩阵
tril、**triu**: 按主对角线的间距斜向取矩阵元素组成新矩阵
diag: 取矩阵的主对角线元素
rot90、**fliplr**、**flipud**: 矩阵旋转 90 度、左右翻转、上下翻转
reshape: 更改矩阵行列数

还有一些图形图像、第三方工具箱等函数。

实验题 Q1.11

Q1.11 绘制 \cos 函数曲线，分别用两种方法，一是直接 \cos 函数，二是用如下展开式近似：

$$\cos(x) \approx 1 - \frac{x^2}{2!} + \frac{x^4}{4!}$$

$$x \in [0, 4\pi]$$

提示：先创建 x 数组，求出两种不同的 \cos 函数值数组，然后用 **plot** 函数绘制曲线。

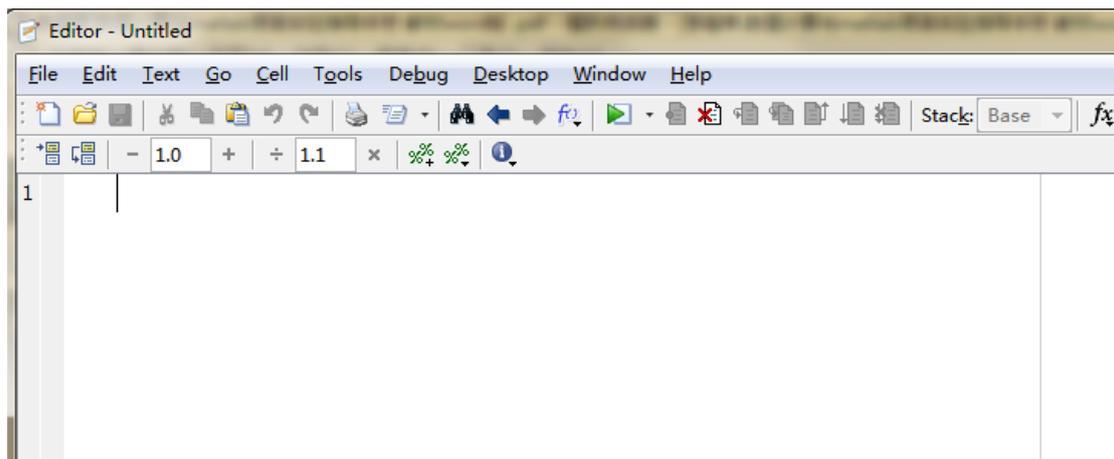
二、MATLAB 程序设计

MATLAB 提供了 M-file 的方式, 可让使用者自行将指令及算式写成脚本程序, 然后储存成程序文件, 其后缀名 “.m”, 例如 test.m, 其中的 test 就是文件名称, 也等同于文件中主函数名称。文件名与主函数(文件中的第一个函数)名称两者一定要一致, 不然 MATLAB 很难找到指定的函数。切记!

2.1 脚本编辑器

可以用任何一种文本编辑软体(如 Window 系统提供的记事本软件均可)或是文书处理软体(如 Word, AmiPro), 但是储存格式必须是 Ascii 文本格式。

在 MATLAB 中, 提供了一个程序开发的集成软件 IDE, 即脚本编辑器(Editor)。该软件可以很方便地编写脚本程序(关键词会自动高亮、函数参数会自动提醒等)、可以实时提醒语句输入错误、可以跟踪测试代码等等。

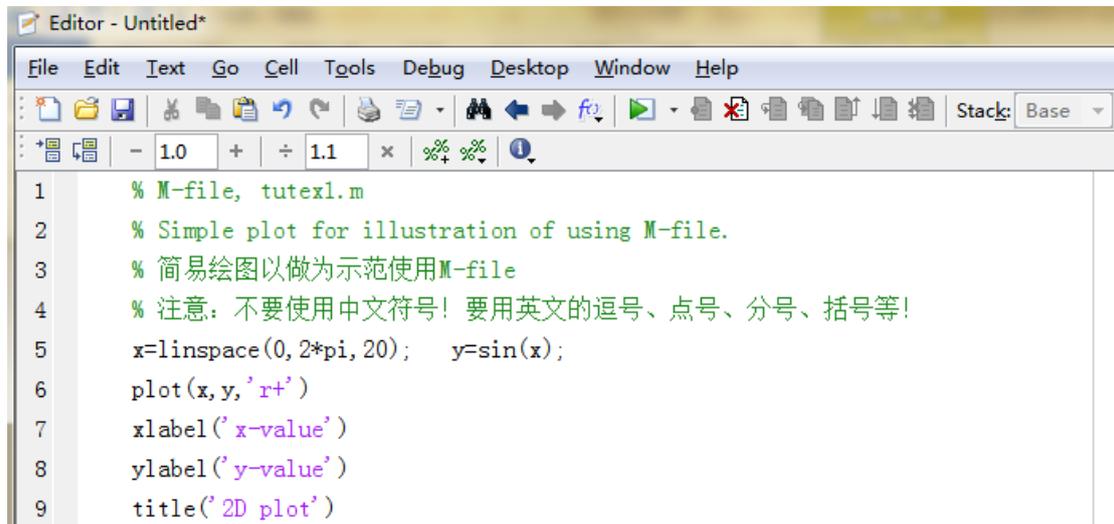


要进入脚本编辑器, 在 MATLAB 界面中选择菜单 “File | New | Script”, 即可弹出脚本编辑器(Editor)。当程序写完后要保存时, 必须以.m 后缀名称保存到 MATLAB 指定的搜索目录(见前面的实验题 Q1.3 Set Path)。

程序编写完成后, 要执行 M-file 可以在 Command Window 窗口里直接输入该文件名称(同等于函数名称)如 test; 或者直接在脚本编辑器 Editor 中按快捷键 “F5”。要跟踪诊断测试程序, 可以先设好断点(用鼠标在 Editor 的行号上点击即可), 然后按快捷键 “F5” 运行程序、遇到断点程序会自动停下来, 然后按快捷键 “F10” 逐句运行。所有运行的快捷键在 Editor 界面的 “Debug” 菜单下拉项中找得到。

实验题 Q2.1 : 2.3

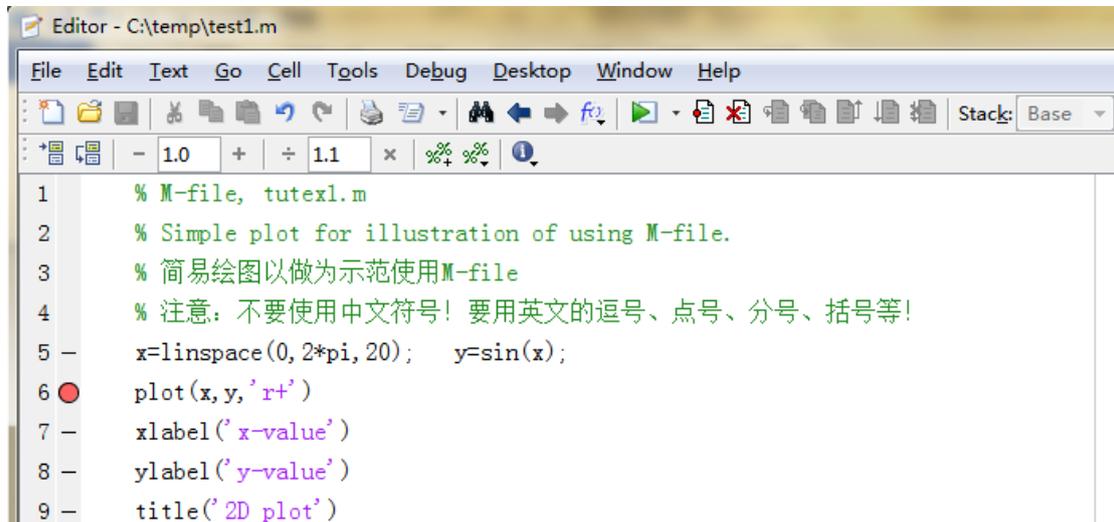
Q2.1 打开 Editor，输入如下程序代码。（%号开头的是注释语句）



```
Editor - Untitled*
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: Base
- 1.0 + ÷ 1.1 × % %
1 % M-file, tutex1.m
2 % Simple plot for illustration of using M-file.
3 % 简易绘图以做为示范使用M-file
4 % 注意：不要使用中文符号！要用英文的逗号、点号、分号、括号等！
5 x=linspace(0, 2*pi, 20); y=sin(x);
6 plot(x,y,'r+')
7 xlabel('x-value')
8 ylabel('y-value')
9 title('2D plot')
```

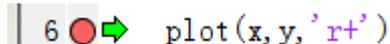
然后把程序保存成文件：test1.m，注意直接保存到 MATLAB 的 Work 文件夹下即可，不要自行更改文件夹！如果文件夹改动了，那么必须要按照前面的实验题 Q1.3 Set Path 来添加新的文件夹！保存成功后，Editor 的工具栏上的**磁盘图标会变灰色**，不然就是没有保存。按“F5”执行程序，记录下程序运行的结果。读懂程序，看看运行结果是否期待的结果。

Q2.2 继续上一题。鼠标点击行号 6，即在第 6 条语句上设置断点。如下图所示的红点。



```
Editor - C:\temp\test1.m
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: Base
- 1.0 + ÷ 1.1 × % %
1 % M-file, tutex1.m
2 % Simple plot for illustration of using M-file.
3 % 简易绘图以做为示范使用M-file
4 % 注意：不要使用中文符号！要用英文的逗号、点号、分号、括号等！
5 - x=linspace(0, 2*pi, 20); y=sin(x);
6 ● plot(x,y,'r+')
7 - xlabel('x-value')
8 - ylabel('y-value')
9 - title('2D plot')
```

按快捷键“F5”运行程序，你会发现程序运行并停留在第 6 行。



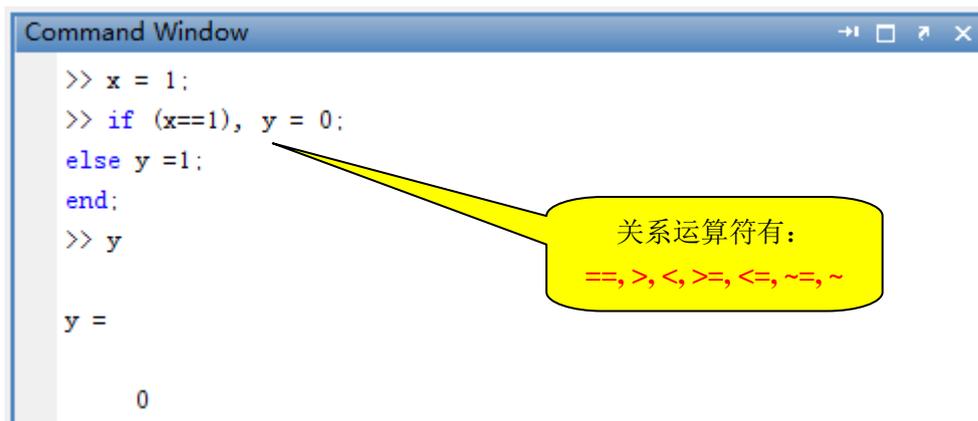
```
6 ● → plot(x,y,'r+')
```

回到 Command Window 界面，在 Workspace 里可以看到目前的变量情况。

2.2 脚本语法语句

MATLAB 脚本语句非常简单明了，基础语句分别如下几种类型。MATLAB 语句是并行语言，可以批量对数组或矩阵中的所有元素进行操作。因此在 C 语言中的好多 For 循环均可以取消。显示语句指结果显示到 Command Window 中，sprintf 用来对显示结果进行格式化。所有的语句均可以编写到 M-File 中，作为程序的一部分。

赋值语句、判断语句



```
Command Window
>> x = 1;
>> if (x==1), y = 0;
else y = 1;
end;
>> y

y =

    0
```

关系运算符有：
==, >, <, >=, <=, ~=, ~

判断语句有如下几种典型形式：

<pre>if (逻辑运算式) else end</pre>	<pre>if (逻辑运算式) elseif (逻辑运算式) else end</pre>	<pre>if (逻辑运算式) else if (逻辑运算式) else end end</pre>
--	---	--

需要注意的是 **elseif** 与 **else if** 之间的区别。中间有空格的其实是第 2 个 if 语句嵌套在第 1 个 if 语句的 else 分支中，所以需要额外的 end 来结束。

在 Editor 中会自动判断 if 和 end 是否对应，如果发现有 if 语句没有对应的 end，则会在 if 底下出现红色波浪线。程序也会拒绝运行。

与 if 类似还有 switch 语句，switch 语句比较复杂，且关系表达式用途较窄，一般不建议使用。可以用 doc switch 查看该语句的定义和 Examples。

显示语句

```
Command Window
>> x = 1; y = 2.1; z = -0.1;
>> disp([x, y, z])
    1.0000    2.1000   -0.1000

>> sprintf('x=%d, y=%4.2f, z=%e \n', x, y, z)

ans =

x=1, y=2.10, z=-1.000000e-001

>> disp(sprintf('x=%d, y=%4.2f, z=%.2e \n', x, y, z))
x=1, y=2.10, z=-1.00e-001
```

显示格式参见：
help sprintf

循环语句

```
Command Window
>> x = 1;
>> for k=2:100, x = x + k; end; %计算 1+2+...+100 的值
>> disp(sprintf('x=%d', x))
x=5050
>> y = 1; k = 2;
>> while (k<=100), y = y + k; k = k + 1; end;
>> disp(sprintf('y=%d', y))
y=5050
```

特殊脚本语句

clc: 命令窗口清屏
clear all: 清除所有内存变量
figure: 新建一个图形子窗口
figure(n): 新建一个指定编号的图形子窗口
format long: 用长格式显示变量值
break, continue: 循环体内跳转

特殊变量

ans: 上一次运算结果
Inf: 无穷大 (被 0 除)
NAN: 不定值 (0/0)
eps: 大于 0 的最小值 (最小计数单位)
pi: 3.14159.....
nargin: 函数的输入参数个数

输入、输出语句



```
Command Window
>> x = input('请输入x的值: ');
请输入x的值: 10
>> disp(x)
    10

>> save test1.txt x -ascii; %把x变量值保存到test1.txt文件
>> y = load('test1.txt'); %从文件test1.txt文件读取
>> disp(y)
    10
```

输入语句是在 Command Window 界面中从键盘输入到变量中。其中空格、逗号、分号代表数组或矩阵元素的分割，规则与创建矩阵一样。

从文件输入到变量（如读取文本文件的内容到一个变量）可以用 **load 函数**。在 doc load 的帮助信息中有详细的 Examples。

输出语句 **save** 可以把变量值写入指定文件中，也可以用函数形式来调用。具体可以参见 doc save 的帮助信息。

实验题 Q2.4 : 2.5

Q2.4 编写一个自定义函数，并在 Command Window 中调试通过。

编写一个成绩转换等级的自定义函数，用户输入成绩的分数值，函数自动判断成绩分数属于哪个等级，输出等级对应的字母。转换规则按下表。

成绩	等级
Score < 60	F
60 <= Score < 70	D
70 <= Score < 80	C
80 <= Score < 90	B
90 <= Score <= 100	A

要求函数的调用形式为：

Grade = letter_grade(Score);

在 Command Window 中用 Score = 61、52、73、88、91、100、101、-1、-20 等测试，记录测试信息。

Q2.5 把下列表格中的数据输入到一个变量 X，然后用 sprintf、disp、for 循环配合，把 X 在 Command Window（或文本文件）中按照规定的格式显示出来。

0.36	0.001	2
0.2	0.002	1
0.15	0.12	1.5
0.3	0.21	2.6

把表格数据输入到 MATLAB 变量：


```

10 function R1=Do_Game
11     disp(' ');
12     R1 = input(' 请输入您的选择[1-石头, 2-剪刀, 3-布, 0-不玩了]: ');
13     if (R1==0), return; end;
14     R0 = floor(3*rand + 1);
15     disp(sprintf(' 您选择了%d, 计算机选择了%d', R1, R0));
16     if (R1==R0), disp(' 两者一样, 和!');
17     elseif (R1==1)
18         if (R0==2), disp(' 您赢了!'); else disp(' 您输了!'); end
19     elseif (R1==2)
20         if (R0==1), disp(' 您输了!'); else disp(' 您赢了!'); end
21     elseif (R1==3)
22         if (R0==1), disp(' 您赢了!'); else disp(' 您输了!'); end
23     end;
24

```

第 3 步：点击保存按钮，把代码保存成“shi_tou_jian_dao_bu_text.m”文件（默认文件夹）；

第 4 步：回到“Command Window”，输入：shi_tou_jian_dao_bu_text，按回车，显示：

第 5 步：输入 0、1、2、或 3，测试程序。

实验题 Q2.6

Q2.6 把上述的小游戏按步骤实现，测试通过，并读懂。把程序的两个函数改成一个函数实现。

三、MATLAB 简单数据分析

3.1 简单统计

MATLAB 中提供了对数据的基础统计分析函数。

`max` 函数，求数组或矩阵中的最大值。样例如下：

Examples

Return the maximum of a 2-by-3 matrix from each column:

```
X = [2 8 4; 7 3 9];
max(X, [], 1)
ans =

     7     8     9
```

Return the maximum from each row:

```
max(X, [], 2)
ans =

     8
     9
```

Compare each element of X to a scalar:

```
max(X, 5)
ans =

     5     8     5
     7     5     9
```

其中后面两个输入参数可以省略，默认是**按列**求最大值。第 3 个参数指明求最大值的方向。具体参看例子。第 3 个例子中是求保底最大，即每个元素与 5 比较，如果 <5 那么就取 5，不然取原来的值。

最小值函数 `min`、均值函数 `mean`、方差（标准差）函数 `std`、中位数函数 `median`、求和函数 `sum`、求连乘 `prod`、求累积和 `cumsum`、求累计连乘 `cumprod` 等这些函数与 `max` 函数一样，均有求值方向选择参数，具体参见每个函数的帮助信息。

排序函数有两个：**`sort`**、**`sortrows`**，函数 `sort` 仅按第 1 列数据进行排序，`sortrows` 可以按照矩阵的多列同时来排序。下例是按照第 1、7 列同时进行排序。

```
D = sortrows(A, [1 7])
D =

    76    79    91     0    19    41     1
    76    61    93    81    27    46    83
    95     7    73     5    19    44    20
    95     7    73    89    20    74    52
    95     7    40    35    60    93    67
    95    45    92    41    13     1    84
```

实验题 Q3.1

Q3.1 对下列矩阵分别求每列、每行的最大值、最小值、平均值、中位数、标准差。

76	79	91	0	19	41	1
76	61	93	81	27	46	83
95	7	73	5	19	44	20
95	7	73	89	20	74	52
95	7	40	35	60	93	67
95	45	92	41	13	1	84

并按第 1、2、3 列同时排序。

求上述矩阵所有元素中的最大值、最小值，并求最大值、最小值所对应的行号、列号。

求上述矩阵每行的和、每列的和、所有元素的和。

3.2 条形分布图

hist 函数用来对数据进行条形概率统计，并画出条形的概率分布图。也就是说，一组数据拿来后，可以按照指定的条块区间进行分类归并，统计每个区间的数据点数，归一化成概率，最后把这些区间概率用条形图画出来。

条形分布图在揭示物理现象、统计分析数据规律等过程中具有非常大的作用。可以从不同角度来对待分析具体现象。

hist 函数的调用格式如下：

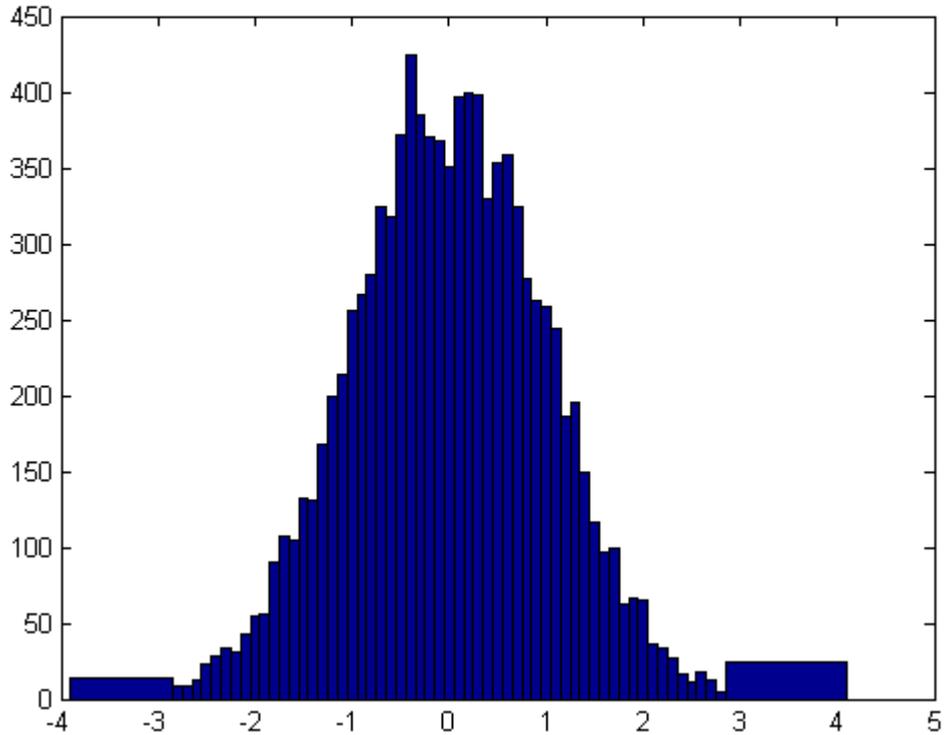
Syntax

```
n = hist(Y)
n = hist(Y,x)
n = hist(Y,nbins)
[n,xout] = hist(...)
hist(...)
hist(axes_handle,...)
```

第 1 种形式是按默认的 20 个区间对 Y 数据进行归并统计；第 2 种形式是按指定的区间 x 来进行归并统计；第 3 种是按指定的区间个数来归并统计；第 4 种表示除了显示条形图之外，还输出 xout 区间的取值情况；第 5 种形式是仅看看条形分布图，不需要图形窗口句柄 n；第 6 种是把条形分布图画在指定的画布（窗口坐标轴）上。

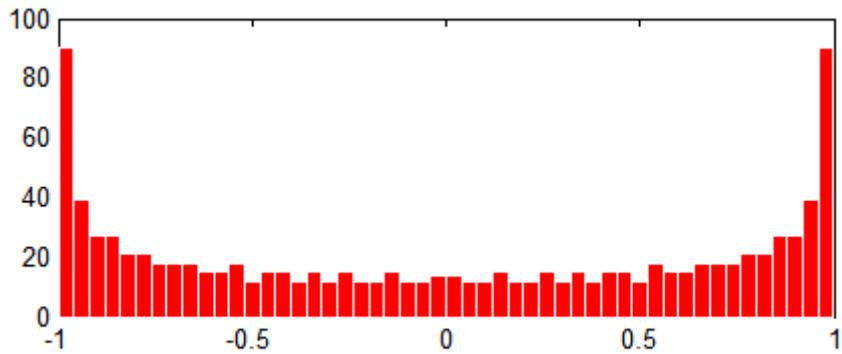
一个随机数的条形分布图样例如下：

```
x = -4:0.1:4;  
y = randn(10000,1);  
hist(y,x)
```



可以看到概率分布基本符合高斯正态分布，但有误差。

下面看看 \sin 函数的条形分布图，这是从统计的角度看 \sin 函数。



可以看到， \sin 函数值为 0 附近的个数最少，为 1 和 -1 附近的个数最多，呈对称形态。

实验题 Q3.2

Q3.2 编程画出直线、 \cos 曲线、 \exp 曲线、 x^2 曲线（或其它多项式）等的条形分布图，总结这些曲线的统计规律。

3.3 随机数

1) 均匀随机数

用 MATLAB 函数 **rand** 产生在区间 **[0, 1]** 的均匀随机数，它是平均分布在 **[0, 1]** 之间。随机数种子 **seed** 用来控制产生随机数的初始值，一般尽可能让 **seed** 的值随机。每次产生随机数的值都不会一样，这些值代表的是随机且不可预期的，这正是用随机数的目的。可利用这些随机数代入算式中，来表示某段讯号的不规则振幅或是某个事件出现的机率。均匀随机数其值平均地分布于一区间的特性可以从其统计密度函数(probability density function, PDF) 说明。从其 PDF 分布类似长条图的分布可以看出其每一个随机数值出现的机率皆相同，所以它被称为均匀随机数。

rand 函数的格式如下：

Syntax

```
r = rand(n)
r = rand(m,n)
r = rand([m,n])
r = rand(m,n,p,...)
r = rand([m,n,p,...])
r = rand
r = rand(size(A))
r = rand(..., 'double')
r = rand(..., 'single')
```

第 1 种格式是产生 **n** 行 **n** 列的随机数矩阵；第 2 种格式是产生 **m** 行 **n** 列的随机数矩阵；第 3 种格式同等于第 2 种。。。。

rand 函数的使用样例：

[1] 产生在区间[a, b]之内的随机数：

```
r = a + (b-a).*rand(100,1);
```

[2] 在 1-100 之间产生 1 行 5 列的随机整数矩阵，用 **randi** 函数，也可以用 **rand** 函数。

```
r = randi(100,1,5);
```

[3] 用 **rng** 函数重置随机数发生器，然后生成 1 行 5 列的随机数矩阵。

```
rng('default')
rand(1,5)
ans =
    0.8147    0.9058    0.1270    0.9134    0.6324
```

[4] 以当前时刻为随机数种子来初始化随机数发生器，这样可以确保每次程序启动得到的随机数是肯定不一样的。

```
rng('shuffle');
rand(1,5);
```

注：老版本中的随机数种子 **seed** 也可以兼容，具体参见 **rand** 和 **seed** 的帮助信息。

2) 常态随机数

用 MATLAB 函数 **randn** 产生概率分布随机数（也称**常态随机数**），它是以高斯分布在随机数出现的上下限区间。常态随机数其值分布于一区间的特性可以从其统计密度函数(PDF) 看出。从其 PDF 分布可以看出其每一个随机数值出现的机率皆不相同，靠近中间的数值出现的机率比起两端的值要高，这是一般不规则现象较可能出现的情形，所以它被称为常态随机数。

由于常态随机数并非以上下限来定义，它是用数据的平均值和方差定义。因此在产生一常态随机数时，需设定平均值和方差的大小。`randn(n)`和`randn(n,m)`是分别产生一矩阵含 $n \times n$ 个随机数和一矩阵含 $m \times n$ 行列的常态随机数，其平均值为 0、方差为 1。`randn` 函数的调用格式与 `rand` 函数一致。

Syntax

```
r = randn(n)
r = randn(m, n)
r = randn([m, n])
r = randn(m, n, p, ...)
r = randn([m, n, p, ...])
r = randn
r = randn(size(A))
r = randn(..., 'double')
r = randn(..., 'single')
```

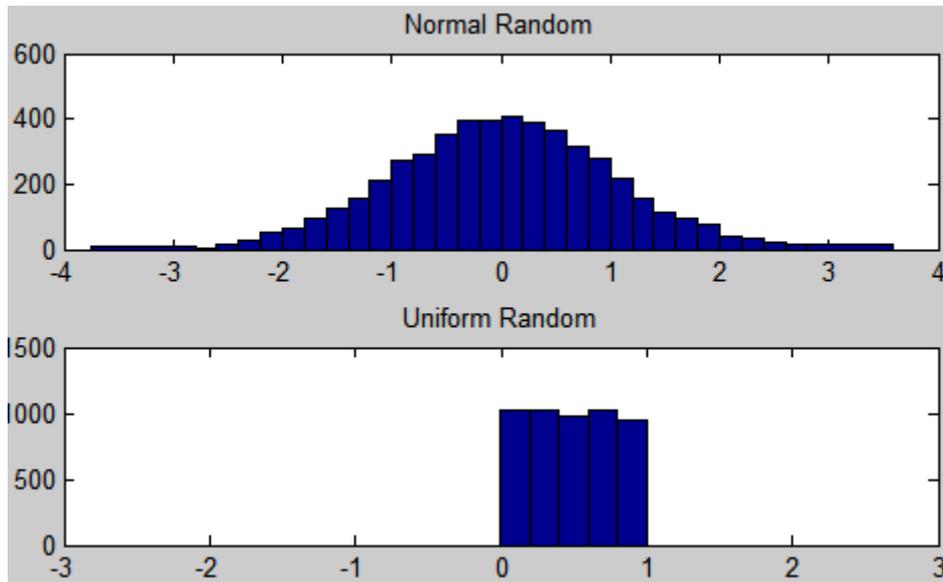
产生一个正态分布随机数矩阵，要求按照指定的均值和方差产生：

```
mu = [1 2];
Sigma = [1 .5; .5 2]; R = chol(Sigma);
z = repmat(mu, 100, 1) + randn(100, 2)*R;
```

以下的例子是分别产生均匀随机数和常态随机数，并画出它们的条形分布图。

```
Command Window
>> x = -2.9:0.2:2.9;
>> yn = randn(1, 5000);
>> y0 = rand(1, 5000);
>> figure, subplot(211), hist(yn, x), title('Normal Random');
>> subplot(212), hist(y0, x), title('Uniform Random');
```

运行结果：



实验题 Q3.3

Q3.3 产生一个 1 行 1000 列的正态分布随机数矩阵，要求正态分布的均值为 2、方差为 0.1。用 `hist` 查看其条形分布图，与均值为 0、方差为 1 的正态分布随机数的条形分布图有何不同？

3.4 多项式

当系数确定后一个多项式也就确定了，因此在 MATLAB 中按照其系数数组来代表一个多项式。也就是说，数组就是多项式。数组的加减，也代表了多项式之间的运算。另外 MATLAB 还提供了多项式求根等函数。

在工程及科学分析上，多项式常被用来模拟一个物理现象的解析函数。之所以采用多项式，是因为它很容易计算。例如一个多项式如下：

$$p(x) = x^3 + 4x^2 - 7x - 10$$

MATLAB 以一个数组的方式代表上述的多项式。

$$p = [1, 4, -7, -10];$$

其中的数值是多项式的各阶项（从高到低）的各个系数。

有了多项式的表示式后，可以来计算其函数值。假设要计算一组数据 x 对应的多项式值，依照一般的函数计算须以下列式子计算：

$$p = x.^3 + 4 * x.^2 - 7 * x - 10$$

为了能直接运用多项式，可以用函数 **polyval** 直接做运算。语法为 **polyval(p,x)**，其中 p 即是代表多项式各阶系数的数组。如下例：

```
Command Window
>> x = linspace(-1, 3);
>> p = [1 4 -7 -10];
>> V = polyval(p, x);
fx >>
```

当二个多项式间要做加减乘除时，加减运算可以直接进行。假设有二个多项式 $a(x)$ 和 $b(x)$ 定义如下。

$$a(x) = x^3 + 2x^2 + 3x + 4, \quad b(x) = x^3 + 4x^2 + 9x + 16$$

如果多项式 $c(x)$ 为上述二多项式相加，则： $p_c = p_a + p_b = [2, 6, 12, 20]$ 。相减则有：

$p_c = p_a - p_b = [-2, -6, -12]$ ，可以发现只有 3 个系数了。

而将两个多项式相乘，则需要用到 **conv** 卷积函数。自己考虑一下为什么？可以得到一个新的多项式： $p_c = p_a \otimes p_b = [1, 6, 20, 50, 75, 84, 64]$ ，可以发现多项式次数增加了。

conv 函数对多项式系数数组做卷积，效果同等于两个多项式相乘。其语法格式如下：

Syntax

$$\begin{aligned} w &= \text{conv}(u, v) \\ w &= \text{conv}(\dots, 'shape') \end{aligned}$$

卷积的数学定义如下:

Definitions

Let $m = \text{length}(u)$ and $n = \text{length}(v)$. Then w is the vector of length $m+n-1$ whose k th element is

$$w(k) = \sum_j u(j)v(k-j+1)$$

实际计算过程如下:

$$\begin{aligned} w(1) &= u(1)*v(1) \\ w(2) &= u(1)*v(2)+u(2)*v(1) \\ w(3) &= u(1)*v(3)+u(2)*v(2)+u(3)*v(1) \\ &\dots \\ w(n) &= u(1)*v(n)+u(2)*v(n-1)+ \dots +u(n)*v(1) \\ &\dots \\ w(2*n-1) &= u(n)*v(n) \end{aligned}$$

如果是两个多项式相除,则需要用到 **deconv** 解卷积函数。deconv 函数的调用格式如下,其中 q 为商多项式、 r 为余数多项式。

Syntax

$$[q, r] = \text{deconv}(v, u)$$

解卷积函数 deconv 使用简单的滤波技术,在有误差或噪声影响下,会产生较大的偏差,甚至不可信。deconv 函数的使用样例如下:

Examples

If

$$\begin{aligned} u &= [1 \quad 2 \quad 3 \quad 4] \\ v &= [10 \quad 20 \quad 30] \end{aligned}$$

the convolution is

$$\begin{aligned} c &= \text{conv}(u, v) \\ c &= \\ & \quad 10 \quad 40 \quad 100 \quad 160 \quad 170 \quad 120 \end{aligned}$$

Use deconvolution to recover u :

$$\begin{aligned} [q, r] &= \text{deconv}(c, u) \\ q &= \\ & \quad 10 \quad 20 \quad 30 \\ r &= \\ & \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \end{aligned}$$

This gives a quotient equal to v and a zero remainder.

上例可以看出,解卷积正确地解出了商和余数多项式。

MATLAB 提供了 **roots** 函数对多项式方程进行**求根**。函数 roots 能够求解任意阶数多项式方程的根,包括复数根。函数 roots 的格式如下。

Syntax

$$r = \text{roots}(c)$$

一个求根的例子如下:

Examples

The polynomial $s^3 - 6s^2 - 72s - 27$ is represented in MATLAB software as

```
p = [1 -6 -72 -27]
```

The **roots** of this polynomial are returned in a column vector by

```
r = roots(p)

r =
    12.1229
    -5.7345
    -0.3884
```

再看一个例子：

```
Command Window

>> p = [1 2 3 4 5];
>> roots(p)

ans =

    0.287815479557648 +    1.41609308017191i
    0.287815479557648 -    1.41609308017191i
   -1.28781547955765 +    0.85789675832849i
   -1.28781547955765 -    0.85789675832849i

fx >>
```

可以看到，函数 **roots** 可以求解复数根。当 roots 函数求解出的根为复数时，如果仅仅想取复数的实部，可以用 **real** 函数；如果仅仅想取复数的虚部，可以用 **imag** 函数；用 **abs** 可以取复数的模。

函数 **poly** 刚好与 roots 相反，函数 poly 是已知多项式的根反过来求多项式系数数组。函数 poly 的格式如下。

Syntax

```
p = poly(A)
p = poly(r)
```

下面这个例子先假定已知多项式的根为 [1; 2; 3]三个，然后用 poly 函数求解出多项式系数，最好在用 roots 来求多项式的根。可以看到两者完全相同。

```
Command Window

>> p = poly([1; 2; 3])

p =

     1     -6     11     -6

>> roots(p)

ans =

         3
         2
         1

fx >>
```

实验题 Q3.4 : 3.8

Q3.4 已知两个多项式 y_1 和 y_2 如下。

$$y_1 = x^4 - 3x^3 + 2x^2 + x + 2$$

$$y_2 = 3x^5 + 2x^3 + x^2 + 7$$

求 $y_1 + y_2$, $y_1 - y_2$, $y_1 \otimes y_2$, $[S, R] = y_1 / y_2$ 。

其中 S 为商多项式, R 为余数多项式。

Q3.5 用前面学到的函数分别求出 Q3.4 中两个多项式的根。

Q3.6 已知一个多项式 y 有 5 个根: $-2, -1, 1, 3, 5$, 并且满足 $y(0) = 1$, 请用前面学到的函数求出该多项式 y 。

Q3.7 函数 $\cos(x)$ 展开成幂级数为:

$$\cos(x) \approx 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \dots$$

定义, P_4 为上述幂级数保留最高次数为 4 次的多项式 (即前 3 项), P_6 为保留最高次数为 6 次的多项式 (即前 4 项), P_8 为保留最高次数为 8 次的多项式.....

令 $x = \frac{\pi}{3}$, 求出 $\cos(x)$ 、 $P_4(x)$ 、 $P_6(x)$ 、 $P_8(x)$ 、 $P_{100}(x)$ 。

提示: 用函数 **polyval** 计算多项式, 用函数 **factorial** 计算阶乘。

Q3.8 在控制系统中一般用传递函数来表达输入信号与输出信号之间的系统特征。下面是一个机器人定位系统的传递函数, 其中 $G(s)$ 为系统传递函数, $C(s)$ 为系统输出信号, $N(s)$ 为系统输入信号, s 为拉普拉斯变换复变量。

$$G(s) = \frac{C(s)}{N(s)} = \frac{s^3 + 9s^2 + 26s + 24}{s^4 + 15s^3 + 77s^2 + 153s + 90}$$

用 MATLAB 把 $G(s)$ 变换成如下形式。

$$G(s) = \frac{C(s)}{N(s)} = \frac{(s + a_1)(s + a_2)(s + a_3)}{(s + b_1)(s + b_2)(s + b_3)(s + b_4)}$$

四、MATLAB 图形工具

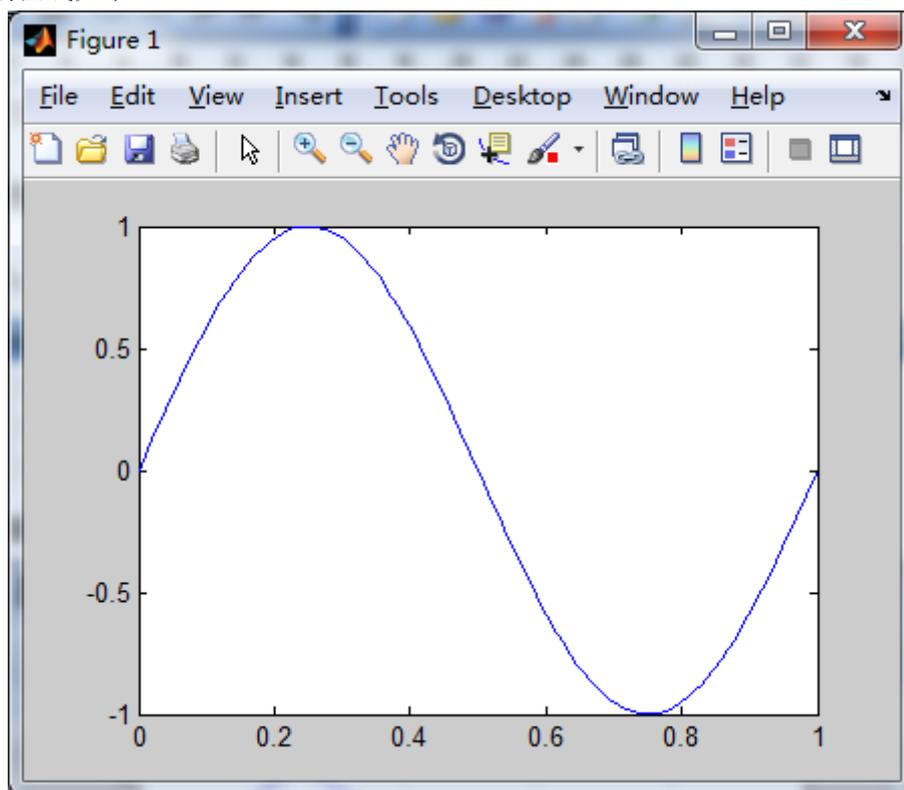
MATLAB 提供了丰富的图形图像工具，包含多个图形图像、视频、计算机视觉等第三方工具箱。这里仅介绍图形最基本的功能。

4.1 二维图形曲线

输入脚本：

```
Command Window
>> x = linspace(0,1,100);
>> y = sin(2*pi*x);
>> plot(x,y)
```

则绘制的曲线如下：



在“Command Window”中输入“`doc plot`”，可以观看“plot”指令的所有功能和相关的样例代码。常见的二维图形绘制函数有：

- figure**: 创建一个空白的图形窗口
- plot**: 绘制多条二维曲线（直角坐标系）
- grid on**: 显示网格线，**grid off**: 不显示网格线
- xlabel**: 显示 x 轴单位，**ylabel**: 显示 y 轴单位
- title**: 显示标题

legend: 显示图例
axis on: 显示坐标轴及其刻度, **axis off:** 关闭坐标轴及其刻度
ginput: 用鼠标点击的方式在图形窗口中输入坐标
gtext: 在图形指定位置显示文本字符串或数值
drawnow: 立刻刷新图形窗口
subplot: 子图
semilogx, semilogy, loglog: 按对数轴方式显示图形曲线
polar: 极坐标系作图
bar, stem, stairs, area, pie, pie3: 直方图、圆点图、阶梯图、圆饼图...

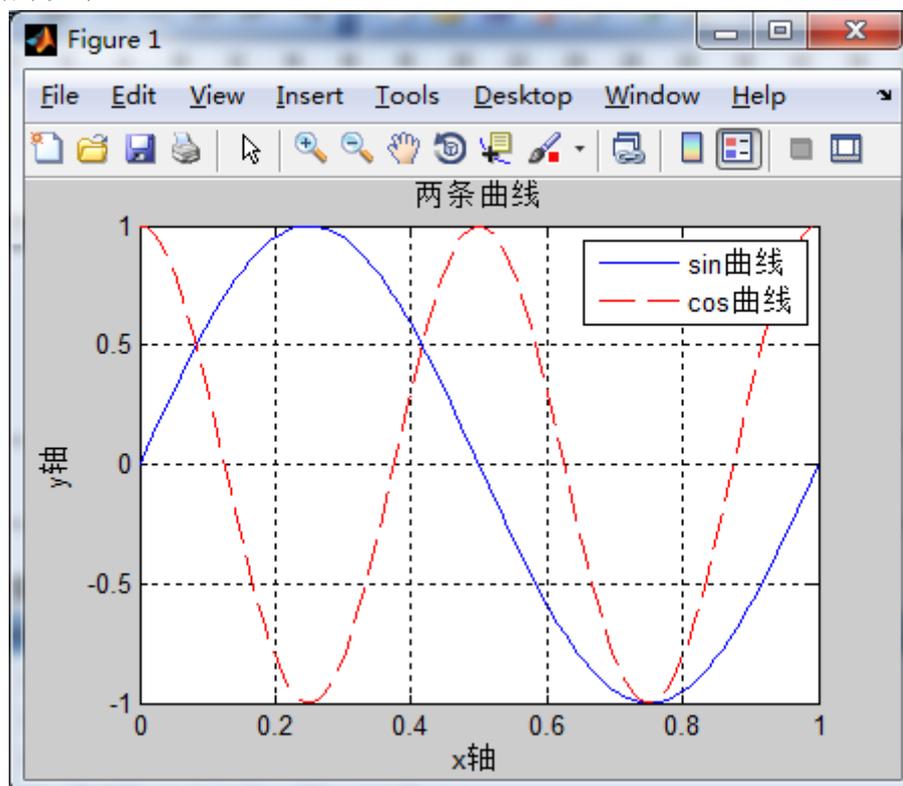
一个稍微复杂一点的样例:

```

Command Window
>> x = linspace(0,1,100);
>> y = sin(2*pi*x);
>> z = cos(2*pi*2*x);
>> plot(x,y,x,z,'r--'), grid on;
>> title('两条曲线');
>> xlabel('x轴'); ylabel('y轴');
>> legend('sin曲线','cos曲线');

```

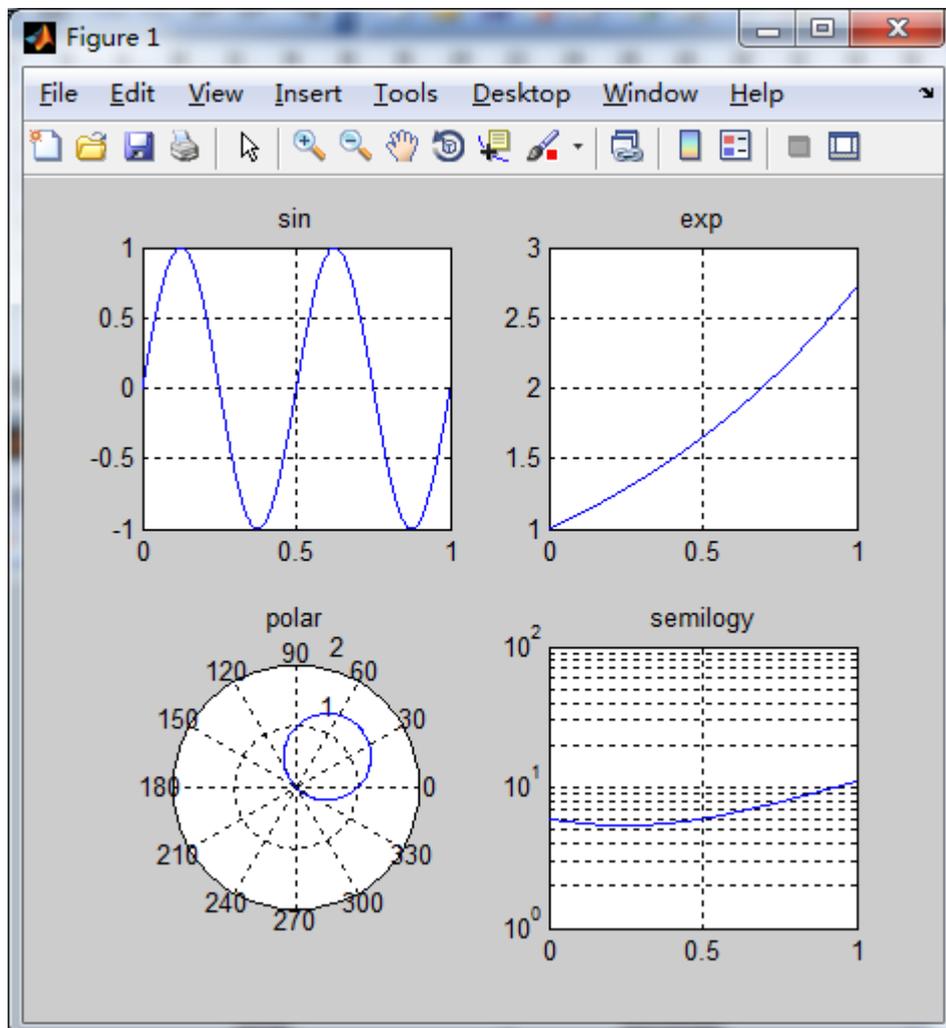
显示的结果如下:



子图样例：

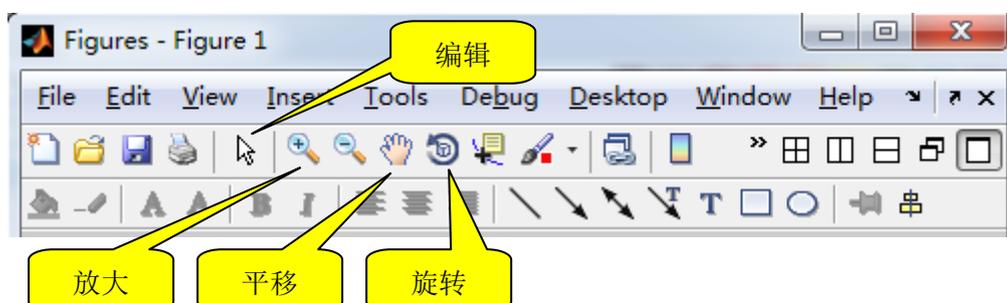
```
Command Window
>> x = linspace(0,1,1000);
>> y1 = sin(2*pi*2*x);
>> y2 = exp(x);
>> h = linspace(0,2*pi,1000);
>> y3 = sin(h) + cos(h);
>> y4 = 10*x.^2 - 5*x + 6;
>> subplot(221), plot(x,y1), grid on, title('sin');
>> subplot(222), plot(x,y2), grid on, title('exp');
>> subplot(223), polar(h,y3), grid on, title('polar');
>> subplot(224), semilogy(x,y4), grid on, title('semilogy');
```

显示结果如下：



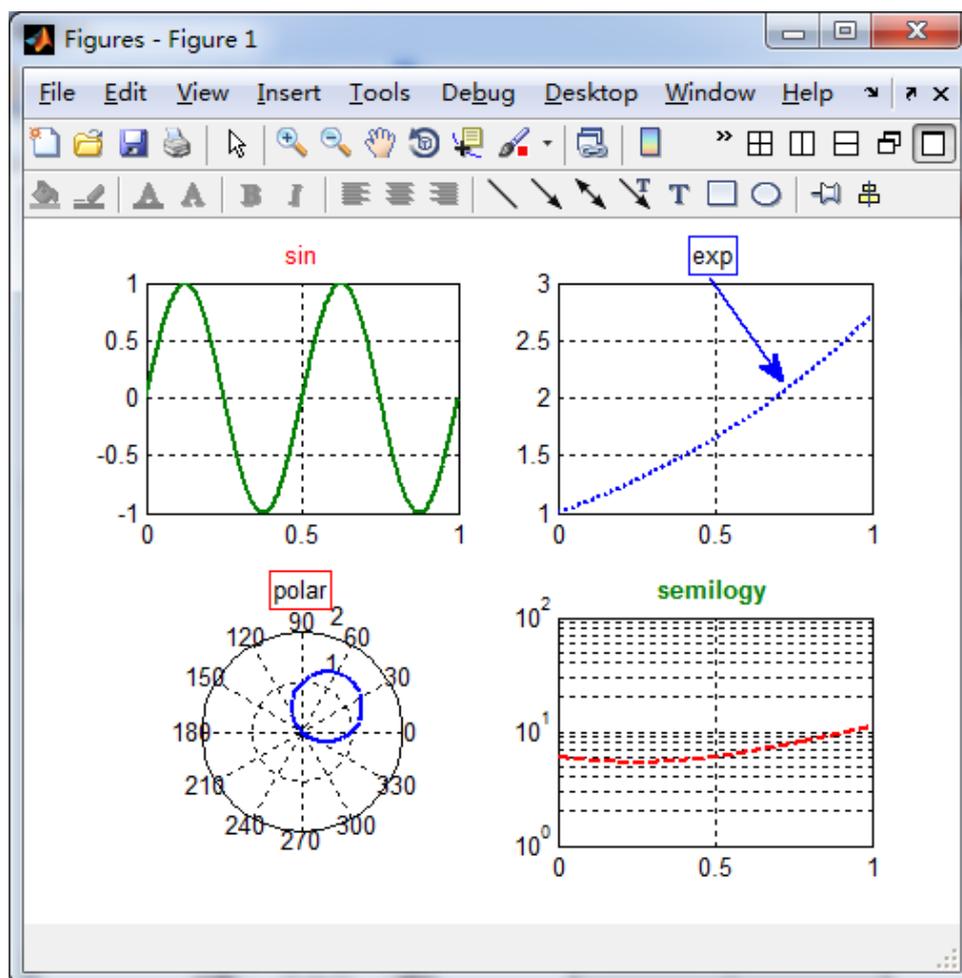
4.2 图形窗口操作

MATLAB 图形（图像）窗口中提供了丰富的手动操作，可以用手工的方式对绘制完成的图形（图像）做很多方面的修改。



放大/缩小：点击放大按钮后，可以在图形窗口中按鼠标左键选取放大区域，按鼠标右键进行缩小等操作；旋转：点击旋转按钮后，可以在图形窗口中按住鼠标左键不放，移动鼠标即可观看不同旋转角度后的图形；**手工编辑**：点击编辑按钮后，编辑工具栏的按钮就会亮起来，可以修改曲线粗细、线型、颜色，可以删除某一条曲线，可以添加文本、添加直线、箭头线、方框，可以更改坐标刻度、坐标颜色、坐标比例、图例、背景等；

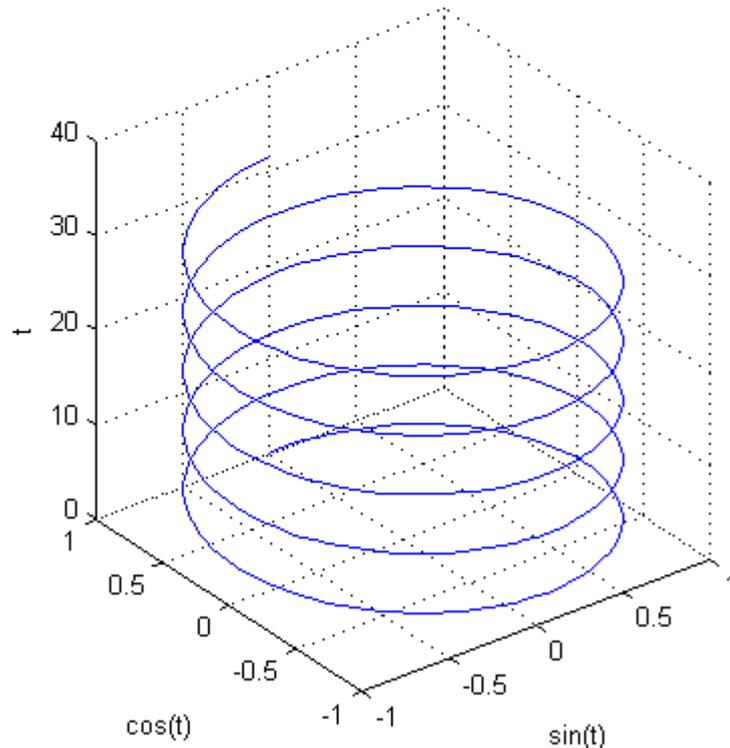
这是前面的样例图经过手工修改后的结果：



4.3 三维图形绘制

三维曲线绘制样例：

```
t = 0:pi/50:10*pi;
plot3(sin(t), cos(t), t)
xlabel(' sin(t)')
ylabel(' cos(t)')
zlabel(' t')
grid on
axis square
```



plot3 的使用可以参考 MATLAB 帮助，大部分图形语句可以适用于三维坐标系。

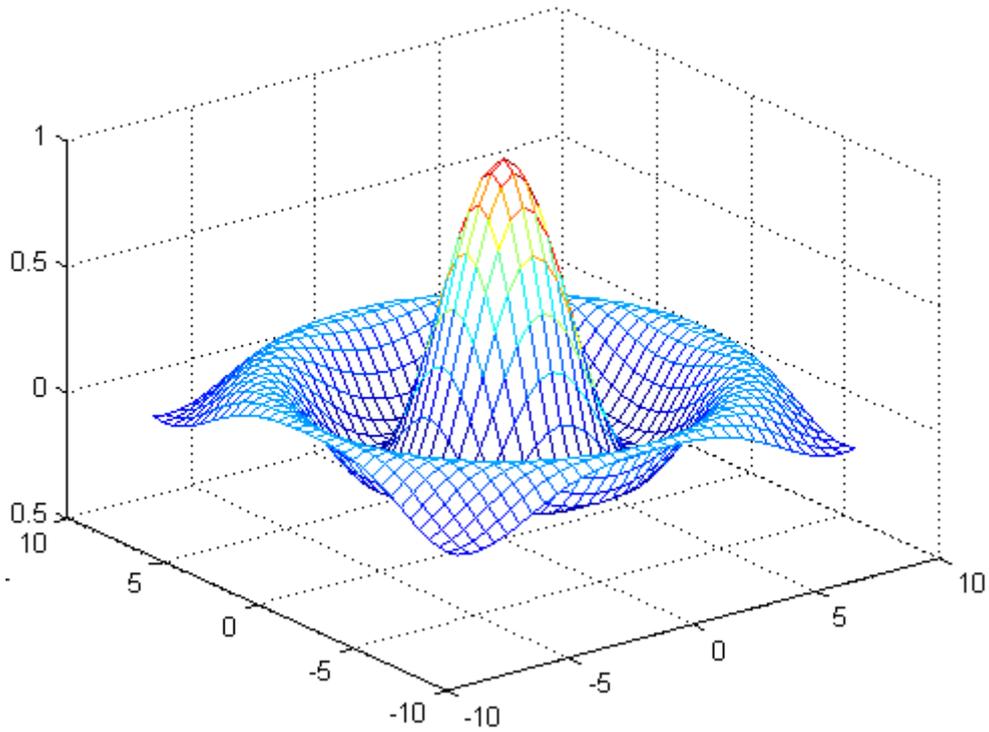
三维曲面的绘制主要有两类：**mesh** 和 **surface**。其中 **mesh** 仅绘制曲面骨架，**surface** 对曲面进行渲染。

mesh 函数的样例：

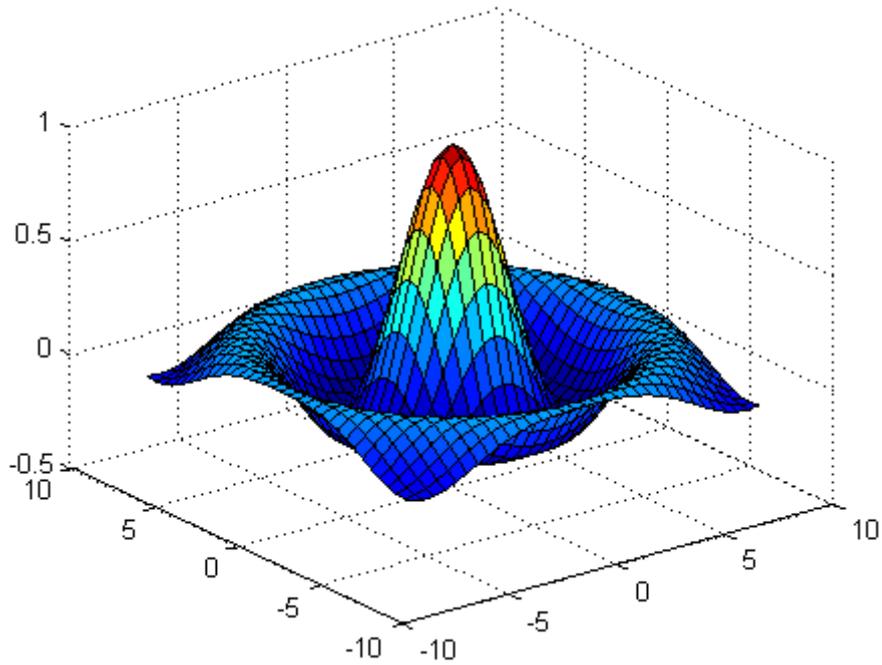
```
[X, Y] = meshgrid(-8:.5:8);
R = sqrt(X.^2 + Y.^2) + eps;
Z = sin(R) ./ R;
mesh(X, Y, Z)
```

meshgrid 函数生成 X-Y 平面上的网格

运行结果为：



把代码中的 `mesh` 函数换成 `surface`, 则结果显示为:



三维曲面、等高线函数还有: `meshc`、`meshz`、`surface`、`surfc`、`contour`、`contour3` 等等。

函数 `contour` 和 `quiver` 组合起来可以绘制等高线及其偏导数方向。函数 `contour`、`quiver` 的调用格式如下:

Syntax

```
contour(Z)
contour(Z, n)
contour(Z, v)
contour(X, Y, Z)
contour(X, Y, Z, n)
contour(X, Y, Z, v)
contour(..., LineSpec)
contour(axes_handle, ...)
[C, h] = contour(...)
```

Syntax

```
quiver(x, y, u, v)
quiver(u, v)
quiver(..., scale)
quiver(..., LineSpec)
quiver(..., LineSpec, 'filled')
quiver(..., 'PropertyName', PropertyValue, ...)
quiver(axes_handle, ...)
h = quiver(...)
```

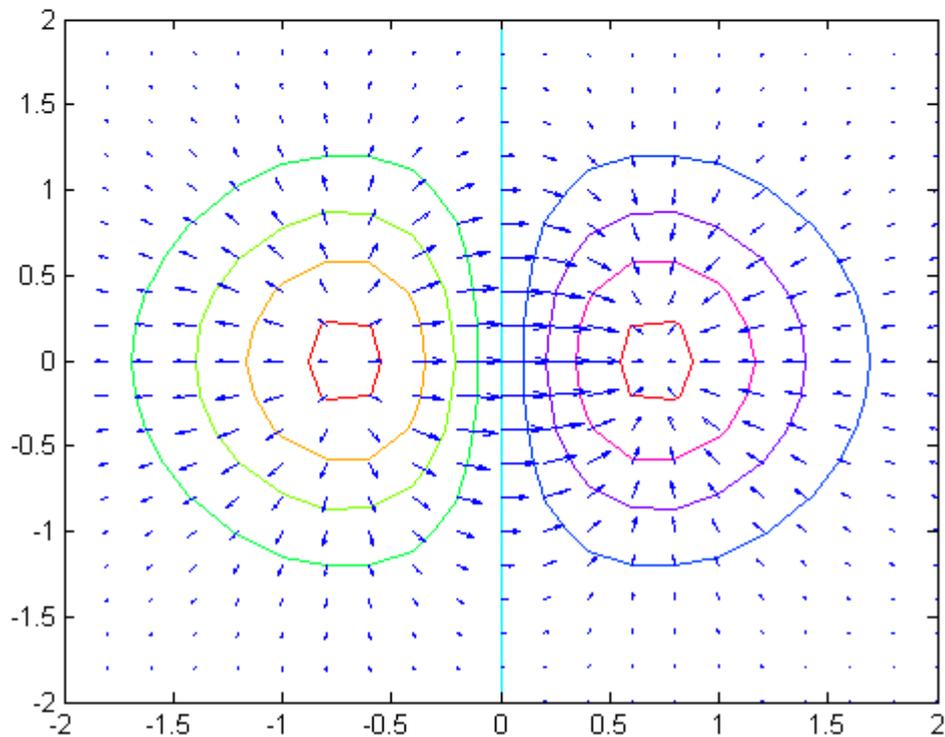
其中， Z 为二元函数值， X 和 Y 为二元函数自变量， u, v 为二元函数的两个偏导数变量。例子如下：

Examples

Showing the Gradient with Quiver Plots

Plot the gradient field of the function $z = xe^{-x^2 - y^2}$:

```
figure
[X, Y] = meshgrid(-2:.2:2);
Z = X.*exp(-X.^2 - Y.^2);
[DX, DY] = gradient(Z, .2, .2);
contour(X, Y, Z)
hold on
quiver(X, Y, DX, DY)
colormap hsv
hold off
```

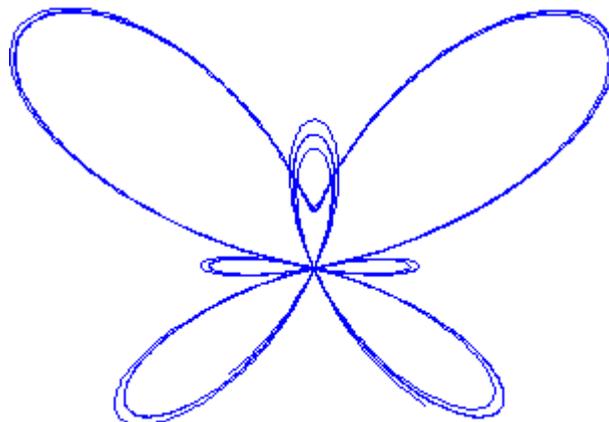


4.4 实验题 Q4.1 : 4.4

Q4.1 重复上述所有的样例代码，要求手工调整图形显示效果，记录每次显示出来的图形。

Q4.2 下面是蝴蝶曲线的参数方程组，右图是显示效果。

$$\begin{cases} x = \sin(t) \left(e^{\cos t} - 2 \cos 4t - \sin^5 \frac{t}{12} \right) \\ y = \cos(t) \left(e^{\cos t} - 2 \cos 4t - \sin^5 \frac{t}{12} \right) \end{cases}$$



要求用 MATLAB 的 subplot 画出 x, y 与 t 之间的关系曲线、 x 与 y 之间的关系曲线。
参数 t 的参考取值范围是 $0:1/16:100$ ，改变 t 的取值，观看不同的蝴蝶形状。

Q4.3 续上题，蝴蝶曲线也可以用如下的极坐标表达式。

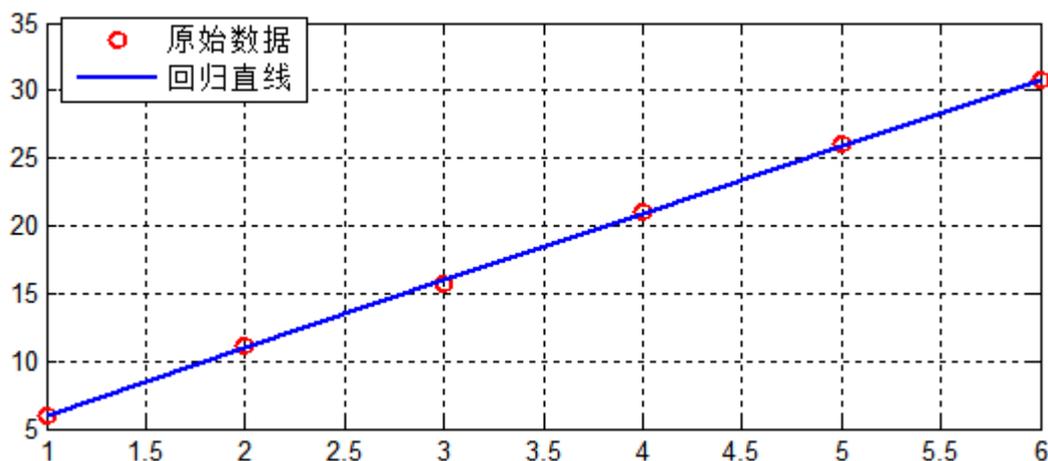
$$r = e^{\sin \theta} - 2 \cos 4\theta - \sin^5 \left(\frac{2\theta - \pi}{24} \right)$$

用 polar 函数绘制出该曲线，并且设置图的标题、XY 轴标签、图例等。

Q4.4 下表列出了一组原始数据和建模回归后的数据。

X	1	2	3	4	5	6
Y0	6	11.1	15.7	21	26	30.8
Ym	6	10.97	15.94	20.91	25.89	30.86

请在 MATLAB 下绘制这两组数据，其中 Y0 用红色圈、Ym 用蓝色粗线条，配上图例，效果如下。（提示：可以用手工调整，也可以用属性编程设置）



五、非线性方程求根

Q5.1 二分法与迭代法

实验目标：学会二分法和迭代法编程，能用两种方法求解任一非线性方程。

详细步骤：

- 1) 创建源代码文件：func0.m，输入上述代码，存盘到work目录；
- 2) 创建源代码文件：ddf_fxxfc.m，输入代码，存盘到work目录；
- 3) 在command window 输入调试脚本来测试程序，例如：
`>>[x,hasroot]=ddf_fxxfc(1,0.001)`
- 4) 若有错误，则返回源代码编辑窗口修改代码，然后重新回到command window 进行测试；
- 5) 测试成功后，修改func0.m文件中的 `y=sin(x).*cos(x)`，换一个方程试试看，看看是否可以正确求解。
- 6) 创建源代码文件：eff_fxxfc.m，把改写好的二分法程序代码输入这个文件，存盘到work目录；
- 7) 回到 command window，测试eff_fxxfc程序是否正确求解；
- 8) 修改func0.m中的方程，更换方程，看看是否正确求解
- 9) 文件：func0.m、ddf_fxxfc.m、eff_fxxfc.m三个文件 + command window中的测试语句复制粘贴到记事本保存。这四个文件压缩成一个rar压缩包，发送到 作业邮箱

具体要求：

- 1 调试通过上述迭代法程序，求解方程的根
- 2 参照上述迭代法程序，改写成二分法程序，求解同一方程
- 3 上交两个程序的源代码文件+运行结果的文本文件

附送参考源代码：

```
%文件 func0.m
function y=func0(x)
y=sin(x).*cos(x);      %假定非线性方程为： x=sin(x)cos(x)
```

```

%文件 ddf_fxxfc.m
function [x,hasroot]=ddf_fxxfc(x0,e)
hasroot=0;
k=0;    %迭代的次数
while k<10000
    x=func0(x0);
    if abs(x-x0)<e
        hasroot=1; break;
    end
    k=k+1;
    x0=x;
end

```

输入参数:

x0 迭代初值

e 解的最大误差

输出参数:

x 方程的解

hasroot 解的合法性。=1 合法; =0 非法

Q5.2 牛顿法及混合算法

上机目标: 完成二分法和牛顿法联合算法, 用于一般性的一维非线性方程求根。

步骤:

1) 创建文件: first_derivative.m, 输入如下代码:

```

1      % 对指定的函数在自变量x0上求一阶导数值
2      % 输入参数:
3      %   Fname, 指针变量, 指向待求导数的函数文件名称
4      %   x0, 自变量值, 求在x0上的一阶导数值
5      %   err0, 迭代的最大许可误差
6      % 输出参数: dF, x0上的一阶导数值
7      % 调用格式:
8      %   dF = first_derivative(Fname, x0, err0);
9      % 使用样例: (command window 中输入如下语句)
10     % >> f1 = inline('sin(x)');
11     % >> d1 = first_derivative(f1, 0, 0.0001)    %求sin函数在0上的导数值
12     % >> d2 = first_derivative(f1, pi/2, 0.0001) %求sin函数在pi/2上的导数值
13     % 另一个样例:
14     % 1) 首先在Editor中编写一个复杂函数, 假定函数代码如下, 保存成 test_func.m
15     %   function y = test_func(x)
16     %   y = x.^3 + 2*tan(x.^2) - 5*log(x) + 7;
17     % 2) 然后在command window 输入如下调试语句:
18     % >> d1 = first_derivative(@test_func, 1, 0.0001);

```

```

19 % -----
20 function dF = first_derivative(Fname, x0, err0)
21 if (x0==0), dx = 1.0e-2; else dx = 0.1*x0; end; %初始的差商区间间隔
22 dF0 = inf; %初始导数值设为无穷大
23 K = 1; %迭代次数
24 while (K<10000)
25     f1 = Fname(x0-dx);     f2 = Fname(x0+dx);
26     dF1 = (f2-f1)/2/dx; %采用中心差商法近似导数
27     if (abs(dF0-dF1) < err0), break; end; %达到误差要求则退出
28     dF0 = dF1;
29     dx = dx/2; %缩小差商区间间隔
30     K = K+1; %迭代次数累加
31 end;
32 dF = dF1;

```

- 2) 参考实验题 Q5.1 的代码和上述求导代码，自行编写用于求解一维非线性方程的牛顿迭代法，保存到 fxxfc_newton.m 文件；
- 3) 自行编写一个待求根的非线性函数，保存到 test_func.m 文件；
- 4) 在 command window 中用 fplot 函数观看 test_func 的函数曲线，寻找二个有根的区域，记为(a1,b1), (a2,b2)；
- 5) 把 test_func 和 (a1,b1) 代入二分法函数 Q5.1，求解得到一个粗糙的根 x1，误差自行决定，可以稍大；然后把 test_func 和 x1 代入 fxxfc_newton 中求解精确的根 x10；
- 6) 同理对 (a2,b2) 进行二分法和牛顿法联合求解，得到粗糙的根 x2 和精确的根 x20；
- 7) 要求上交的文件：test_func.m、二分法的源代码、first_derivative.m、fxxfc_newton.m，以及 command window 中的调试语句的复制文本文件。压缩成一个压缩包，命名为：学号+姓名+非线性方程 2.zip 或 rar。发送到作业邮箱。

六、线性方程组求根--消元法

Q6.1 消元法

上机目标：在高斯消元法基础上，理解并掌握消元法编程和应用。

具体要求：

- 1) 输入附送的源代码，把所有函数保存到同一个文件中，**注意：文件名要与主函数保持一致！**
- 2) 读懂并理解消元、回代过程的代码，调试测试这些代码；
- 3) 更改不同的线性方程组，测试算法的结果，并记录到文本文件；

附送的源代码：

```
% 下面几个函数统一保存在一个文件 Gauss_Elimination_test.m 中
function Gauss_Elimination_test
    clc;
    A = [ 1, 2, 2;      %样例线性方程组的系数矩阵
        -2, -2, -1;
        2, -3, -2];
    b = [3; -3; -1];
    disp('消元前的增广矩阵'); disp([A, b]); %显示到 command window 中
    [U, bb] = Gauss_Elimination(A, b);
    disp('消元后的增广矩阵'); disp([U, bb]);
    x = Back_Substitution(U, bb);
    disp('方程组的解'); disp(x); %显示方程组的解
```

```

% 高斯消元法子函数
% A为系数矩阵, b为非齐次列向量
% U 为消元后的上三角系数矩阵, bb 为消元后的非齐次列向量
function [U,bb] = Gauss_Elimination(A, b)
C = [A, b]; %消元用的增广矩阵
n = length(b); %取方程的个数 = 未知数个数 = 矩阵行列数
%开始 n-1步 消元循环
for k = 1:(n-1)
    % 求解计算因子  $m_{ik} = a_{ik} / a_{kk}$ ;  $i = k+1, \dots, n$ , 并放入L矩阵
    L = eye(n); %初始化 L 矩阵 (行初等变换矩阵  $L_k$ )
    for i = (k+1):n
        mik = C(i,k)/C(k,k); %求第i个计算因子, 其中  $C(k,k)$  是主元
        L(i,k) = -1 * mik; %把计算因子放入L矩阵
    end;
    C = L * C; %进行第k步消元
end;
U = C(:, 1:n); %取出消元后的上三角系数矩阵
bb = C(:, n+1); %取出消元后的非齐次列向量

```

```

%回代法, x 为求解出来的未知数列向量
function x = Back_Substitution(U,bb)
n = length(bb); %取方程个数
x = zeros(n,1); %初始化 x
x(n) = bb(n) / U(n,n); %先求解最后一个未知数  $x(n)$ 
for i = (n-1):-1:1
    x(i) = (bb(i) - U(i, (i+1):n) * x((i+1):n)) / U(i,i); %按回代法公式计算  $x(i)$ 
end;

```

七、线性方程组求根—迭代法

Q7.1 迭代法求解线性方程组

上机目标：掌握迭代法求解线性方程组的编程，学会显示迭代过程的收敛速度。

作业要求：

1. 输入样例代码，调试通过
2. 读懂代码的每个块内容，尤其是两个迭代法看懂曲线图代表的含义
3. 更改不同A、b的值，看看结果
4. 把 Show_Convergence 中的代码换成高斯-赛代尔迭代法
5. （较难）参考 Gauss_Elimination_Time_Test 中的代码，把这个代码改造成多次测试的时间平均，并对比四个不同算法
6. 用相同的线性方程组对比测试高斯消元法、雅克比迭代法、高斯赛代尔迭代法三者的结果和算法运行时间
7. 改变未知数的个数（从2、3、4...10），重复第6步，看看随着未知数的增多，三种不同算法的运算时间增长情况，用曲线图显示出来（较难）
8. 作业递交文件：
 - [1] Linear_Equations_Iteration_Test.m
 - [2] 每次调试的 Command window 文本存入同一个文本文件
 - [3] 改写后的源代码文件，及其调试文本
 - [4] 时间测试的曲线图

附送的源代码：

```
% 线性方程组的迭代法测试
function Linear_Equations_Iteration_Test
    clc
    %单次测试方程组的迭代法求解（雅克比、高斯赛代尔）
    A = [ 5.0, 0.2, 2;          %样例线性方程组的系数矩阵
        -0.2, -1.0, -0.1;
        1, -0.1, 2.0];
    b = [3; -3; -1];
    disp('**** 线性方程组迭代法测试，线性方程组为：'); disp([A, b]);
    L = tril(A, -1);    U = triu(A, 1);    D = A - L - U; % DLU分解
    B1 = -D\ (L+U);    B2 = -(L+D)\U;    %雅克比、高斯赛代尔迭代法的B矩阵
    p1 = Spectral_Radius(B1);            p2 = Spectral_Radius(B2);    %求谱半径，判断收敛性
    disp(sprintf('雅克比迭代矩阵B谱半径=%8.6f，高斯赛代尔迭代矩阵B谱半径=%8.6f', p1, p2));
```

```

tic; x1 = Jacobi_Iteration1(A,b); t1=toc;
disp(' 雅克比迭代法一的解为: '); disp(x1);
disp(sprintf(' 耗费的时间为T1=%8.6f(s)', t1));

tic; x2 = Jacobi_Iteration2(A,b); t2=toc;
disp(' 雅克比迭代法二的解为: '); disp(x2);
disp(sprintf(' 耗费的时间为T2=%8.6f(s)', t2));

tic; x3 = Gauss_Seidel_Iteration1(A,b); t3=toc;
disp(' 高斯赛代尔迭代法一的解为: '); disp(x3);
disp(sprintf(' 耗费的时间为T3=%8.6f(s)', t3));

tic; x4 = Gauss_Seidel_Iteration2(A,b); t4=toc;
disp(' 高斯赛代尔迭代法二的解为: '); disp(x4);
disp(sprintf(' 耗费的时间为T4=%8.6f(s)', t4));

% 雅克比迭代过程解的2-范数曲线显示, 揭示收敛性
Show_Convergence(A, b);

% 求矩阵的谱半径 (特征值的绝对值的最大值)
function p = Spectral_Radius(A)
d = eig(A); %求A的所有特征值
p = max(abs(d)); %谱半径

% 雅克比迭代1: 采用C语言思路
function x1 = Jacobi_Iteration1(A,b)
n = size(A,1); %取未知数个数
x0 = zeros(n,1); x1 = zeros(n,1); %初始化 x0、x1 数组
M = 10; %最大迭代次数
for k=1:M
    for j=1:n
        x1(j) = 0;
        for s=1:n, if (s~=j), x1(j) = x1(j) - A(j,s)*x0(s); end; end;
        x1(j) = (x1(j) + b(j))/A(j,j);
    end;
    for j=1:n, x0(j) = x1(j); end;
end;

```

% 雅克比迭代2: 采用矩阵的迭代表达式

```
function x1 = Jacobi_Iteration2(A,b)
n = size(A,1);    %取未知数个数
L = tril(A,-1);  U = triu(A,1);  D = A - L - U; % DLU分解
M = 10;          %最大迭代次数
x0 = ones(n,1); %初始 x0, 可以随机指定
for k=1:M
    x1 = D \ (-(L+U)*x0 + b);
    x0 = x1;
end;
```

% 高斯赛代尔迭代1: 采用C语言思路

```
function x1 = Gauss_Seidel_Iteration1(A,b)
n = size(A,1);    %取未知数个数
x1 = zeros(n,1); %初始化 x 数组
M = 10;          %最大迭代次数
for k=1:M
    for j=1:n
        x1(j) = 0;
        for s=1:n, if (s~=j), x1(j) = x1(j) - A(j,s)*x1(s); end; end;
        x1(j) = (x1(j) + b(j))/A(j,j);
    end;
end;
```

% 高斯赛代尔迭代2: 采用矩阵的迭代表达式

```
function x1 = Gauss_Seidel_Iteration2(A,b)
n = size(A,1);    %取未知数个数
L = tril(A,-1);  U = triu(A,1);  D = A - L - U; % DLU分解
M = 10;          %最大迭代次数
x0 = ones(n,1); %初始 x0, 可以随机指定
for k=1:M
    x1 = (D+L) \ (-U*x0 + b);
    x0 = x1;
end;
```

% 雅克比迭代过程解的2-范数曲线显示, 揭示收敛性

```
function Show_Convergence(A, b)
n = size(A, 1);      %取未知数个数
L = tril(A, -1);    U = triu(A, 1);    D = A - L - U; % DLU分解
M = 10;             %最大迭代次数
x0 = ones(n, 1);    %初始 x0, 可以随机指定
xx = zeros(1, M);   %存放每次迭代的解的2-范数
xx(1) = norm(x0);   %求x0的2-范数
figure(1024)
for k=2:M
    x1 = D \ (-(L+U)*x0 + b);
    xx(k) = norm(x1);
    x0 = x1;
    plot(1:k, xx(1:k), 'o-r'), grid on; drawnow;
end;
xlabel('迭代次数'); ylabel('解的2-范数');
```

八、矩阵特征值求解

Q8.1 幂法、原点平移法、反幂法

上机目标：学会幂法等算法的编程，改编程序实现平行迭代法。

上机要求：

1. 输入代码，调试通过
2. 读懂代码含义，改变A的元素，看看不同结果
3. 编写 `Origin_Translation` 函数
`Origin_Translation_Inverse` 函数
并在主函数编写调试语句
调节 `p` 的变化范围，看不同结果。
回答下列问题：
[1] 原点平移法能否求出所有特征值？
[2] `p` 的范围很大时，有什么影响？
[3] 原点平移法求出很多的特征值，
有什么规律？怎么处理？
4. *思考如果用两列的矩阵代替 `v0` 进行迭代
(即平行迭代法)，是否可以求出两个特征值？动手改写代码试试看。

附送的源代码：（其中原点平移法只有思路，需要自己改写）

```
% 幂法测试
function mifa_test
    clc;
    A = [2 1 3; 2 5 1; 1 2 1]; %测试矩阵
    err1 = 0.01; %最大允许误差
    [Lamda1, Vector1] = mifa(A, err1);
    disp('矩阵'); disp(A);
    disp('A矩阵精确的特征值、特征向量');
    [Vectors, Lamdas] = eig(A);
    disp([Lamdas, Vectors]);
    disp('*****幂法求解结果*****');
    disp(sprintf('绝对值最大的特征值为:%8.6f', Lamda1));
    disp('特征向量'); disp(Vector1);
    [Lamda2, Vector2] = Inverse_mifa(A, err1);
    disp(sprintf('绝对值最小的特征值为:%8.6f', Lamda2));
    disp('特征向量'); disp(Vector2);
```

```

% 原点平移法（调用幂法函数）的测试语句
% for p = 0:1:10, 调用 Origin_Translation 求解特征值, 显示, end;

% 原点平移法（调用反幂法函数）的测试语句
% for p = 0:1:10, 调用 Origin_Translation_Inverse 求解特征值, 显示, end;

% 原始幂法函数, Lamdal 绝对值最大的特征值, Vector1 特征向量
function [Lamdal, Vector1] = mifa(A, err1)
v0 = ones(size(A,1),1); %假定v0是元素值均为1的列向量
% 或者 v0 = rand(size(A,1),1);
maxD = 1000; %最大迭代次数
K= 0;
while (1)
    v1 = A*v0;
    Lamdal = mean(v1./v0); %用 max, min 等均可
    err0 = std(v1./v0); %标准差衡量特征值变化幅度大小
    if (err0 <= err1), break; end; %若特征值标准差很小则退出
    v0 = v1;
    K = K+1; if (K>maxD), disp('超出最大迭代次数, 幂法失败!'); break; end;
end;
Vector1 = v1/sqrt(sum(v1.^2));

% 反幂法函数, Lamdal 绝对值最小的特征值
function [Lamdal, Vector1] = Inverse_mifa(A, err1)
B = pinv(A); %求广义逆矩阵
[Lamdal, Vector1] = mifa(B, err1);
Lamdal = 1/Lamdal; %特征值 = 1/逆矩阵的特征值

% 原点平移法, 调用幂法函数
% function [Lamdal, Vector1] = Origin_Translation(A, err1,p)
% 参考反幂法函数的编程思路

% 原点平移法, 调用反幂法函数
%function [Lamdal, Vector1] = Origin_Translation_Inverse(A, err1,p)
% 参考反幂法函数的编程思路

```

九、多项式插值

Q9.1 朗格朗日插值、牛顿插值

上机目标：掌握插值算法的编程，学会利用插值算法分析建模

上机要求：

- 1) 把附送的源代码按要求保存到3个不同的M文件；
- 2) 读懂代码含义，调试通过，并观看运行结果；
- 3) 更改插值节点，观看运行结果；
- 4) 更改Runge函数的插值节点个数，观看不同的误差效果；
- 5) *编写一个求解差商的函数，用它来编写牛顿插值函数（较难）

附送的源代码：

- 1) 主程序，要求2个函数保存到同一个文件“Lagrange_Interp_Test.m”

```
function Lagrange_Interp_Test
xi = 0:0.1:pi;   yi = sin(xi); %生成插值节点
%xx = [pi/2, pi/3, pi/4, pi/5, pi/6, pi/7, pi/8]; %待插值点
xx = 0.15:0.2:pi; %另一组待插值点
yy = Lagrange_Interp_M(xi,yi,xx);
%显示插值效果
figure(1), plot(xi,yi,xx,yy,'*r'), grid on;
legend('插值曲线(sin)','插值结果'); xlabel('x');
%测试两个算法的平均运行时间
D = 1000; %重复运行次数
[T1,T2] = Time_Test1(xi,yi,xx,D);
title(sprintf('平均时间：C程序=%8.8fs, M程序=%8.8fs',T1,T2));
% Runge 现象演示
xx = (-10:0.1:10)';   yy0 = 1./(1+xx.^2); % Runge 函数
xi1 = linspace(-10,10,3);   yi1 = 1./(1+xi1.^2); %第1组插值节点, 3个插值节点
xi2 = linspace(-10,10,6);   yi2 = 1./(1+xi2.^2); %第2组插值节点, 6个插值节点
xi3 = linspace(-10,10,11);  yi3 = 1./(1+xi3.^2); %第3组插值节点, 11个插值节点
yy1 = Lagrange_Interp_M(xi1,yi1,xx); %用第1组插值节点对xx进行插值
yy2 = Lagrange_Interp_M(xi2,yi2,xx); %用第2组插值节点对xx进行插值
yy3 = Lagrange_Interp_M(xi3,yi3,xx); %用第3组插值节点对xx进行插值
err(1)=sum((yy1-yy0).^2); err(2)=sum((yy2-yy0).^2); err(3)=sum((yy3-yy0).^2);
figure(2), p1=plot(xx,[yy0,yy1,yy2,yy3]); set(p1,'LineWidth',2); %设置为粗线显示
legend('Runge曲线','3个节点插值','6个节点插值','11个节点插值'); xlabel('x');
title(sprintf('插值误差=[%4.2f, %4.2f, %4.2f]',err));
```

```

function [T1,T2] = Time_Test1(xi,yi,xx,D)
    T1 = 0;    T2 = 0;
    for k=1:D
        tic; Lagrange_Interp_C(xi,yi,xx); T1 = T1+toc;
        tic; Lagrange_Interp_M(xi,yi,xx); T2 = T2+toc;
    end;
    T1 = T1/D;    T2 = T2/D;

```

2) 子函数，按要求保存

```

% 拉格朗日插值法 - C语言思路, 保存文件名: Lagrange_Interp_C.m
% 已知n+1组数据 <xi,yi>, i=1... (n+1), 求 xx 对应的 yy 插值
function yy = Lagrange_Interp_C(xi,yi,xx)
    n = length(xi)-1;
    m = length(xx);    %待插值的数据个数
    yy = zeros(m,1);
    for j=1:m
        L = zeros(n+1,1);    %存放基函数值, 初始化
        for k=0:n
            L(k+1) = 1;
            for s=0:n
                if (k ~= s), L(k+1) = L(k+1)*(xx(j)-xi(s+1))/(xi(k+1)-xi(s+1)); end;
            end;
        end;
        for k=0:n, yy(j) = yy(j) + L(k+1)*yi(k+1); end;
    end;

```

3) 子函数，按要求保存

```

% 拉格朗日插值法 - 矩阵思路, 保存文件名: Lagrange_Interp_M.m
% 已知n+1组数据 <xi,yi>, i=1... (n+1), 求 xx 对应的 yy 插值
function yy = Lagrange_Interp_M(xi,yi,xx)
    n = length(xi)-1;
    m = length(xx);    %待插值的数据个数
    L = ones(m,n+1);    %基函数矩阵
    for j=1:m
        for k=0:n
            for s=0:n
                if (k ~= s), L(j,k+1) = L(j,k+1)*(xx(j)-xi(s+1))/(xi(k+1)-xi(s+1)); end;
            end;
        end;
    end;
    yy = L*yi(:);

```

十、最小二乘拟合与样条插值

Q10.1 基础练习

已知如下测量数据，分别用样条插值和拟合法来编程求解问题。

t	1	2	3	4	5	6	7	8
y	4.00	6.00	8.00	8.80	9.22	9.50	9.70	9.86

1. 用样条插值法 (**spline**) 来求解当 $t = [0.3, 2.5, 4.1, 5.3, 7.6, 8.2]$ 时对应的 y 值。
2. 用最小二乘法 (**矩阵形式**) 来拟合上述数据，设定的拟合模型为：

$$y = \frac{t}{a + bt}$$

用该模型来求解当 $t = [0.3, 2.5, 4.1, 5.3, 7.6, 8.2]$ 时对应的 y 值。

3. 熟悉多项式拟合函数 (观看 **polyfit** 和 **polyval** 的 doc)，用 **polyfit** 函数重做上述第 2 问题，并对比验证第 2 问题的程序正确性。

(矩阵形式的最小二乘法怎么编？把矛盾方程组用矩阵形式写出来， $\mathbf{Ax}=\mathbf{b}$ ，那么 $\mathbf{x} = (\mathbf{A}'\mathbf{A})\backslash\mathbf{A}'\mathbf{b}$ 就是最小二乘法的结果。)

Q10.2 增强练习

假定真实规律为： $y = \sin(10\pi \times x)$

1. 在 $x \in [0,1]$ 中均匀采集 100 个点，得到 100 组精确的数据 (x_0, y_0) ，其中： x_0 和 y_0 均为含有 100 个元素的列向量 (参见 **linspace** 函数)。另外在 $x \in [0,1]$ 区间均匀采集 1000 个点，得到 (x_2, y_2) ，供插值和拟合验证使用。
2. 对 y_0 加入高斯概率分布的噪声，噪声强度为 10dB (参见 **awgn** 函数的 doc)，加噪后的数据记为 y_{10} 。同样加 20dB 噪声后的数据记为 y_{20} ，在同一坐标轴下画出 y_0 、 y_{10} 、 y_{20} 三条曲线，对比观看。
3. 用样条插值对数据集 (x_0, y_{10}) 和 (x_0, y_{20}) 进行建模，并求出 x_2 所对应的 s_{10} 和 s_{20} ，把曲线 (x_2, y_2) 、插值曲线 (x_2, s_{10}) 和 (x_2, s_{20}) 画在同一坐标轴下，观看插值情况。
4. 用多项式 (20 次方) 最小二乘法对数据集 (x_0, y_{10}) 和 (x_0, y_{20}) 进行拟合建模，并求出 x_2 所对应的 p_{10} 和 p_{20} ，把曲线 (x_2, y_2) 、插值曲线 (x_2, p_{10}) 和 (x_2, p_{20}) 画在同一坐标轴下，观看拟合情况。

十一、数值积分

MATLAB 提供了很多积分函数，大致罗列以下：

- (1) **trapz**: 简单梯形积分，步长依赖给定的数据，不能自适应
- (2) **quad**: 采用自适应步长 **simpson** 方法
- (3) **quad8**: 使用 8 阶 Newton-Cotes 算法，精度和速度均优于 **quad**
- (4) **quadl**: 采用 lobatto 算法，精度和速度均较好
- (5) **quadg**: NIT(数值积分)工具箱函数，效率最高，但该工具箱需要另外下载
- (6) **quadv**: **quad** 的矢量化函数，可以同时计算多个积分
- (7) **quadgk**: 很有用的函数，功能在 Matlab 中最强大
- (8) **quad2dgggen**: 一般区域二重积分，效率很好，需要 NIT 支持
- (9) **dblquad**: 长方形区域二重积分
- (10) **triplequadL**: 长方体区域三重积分
- (11) **quadndg**: 超维长方体区域积分，需要 NIT 工具箱支持

Q11.1 牛顿科特斯积分

对于如下的定积分：

$$I = \int_0^1 \frac{1}{1+x^2} dx$$

按如下要求各自编程求解：

- 1) 单区间的梯形积分 T1、两个小区间的梯形积分 T2、三个小区间的梯形积分 T3、四个小区间梯形积分 T4
- 2) 两个小区间的辛普森积分 S1、四个小区间的辛普森积分 S2
- 3) 四个小区间的牛顿科特斯积分 K4

并对比各个积分值之间的误差。

Q11.2 复化积分

用复化梯形、复化辛普森积分来求解 Q11.1 中的定积分，要求：

- 1) 取小区间数为 $n = 64$ ，编程计算 T64、S32，并对比误差精度；
- 2) 输入附送的源代码，调试通过并运行，查看积分结果、迭代次数（区间个数），其中设定 $err = 0.0001$ 。

附送的源代码：

```
% 区间逐步对分的复化梯形求积
function I = Interval_Halving_Trapz(a, b, err)
    n = 1;
    D = 100; %最大迭代次数
    i = 0;   %初始迭代次数
    T1 = (b-a)*(g(a)+g(b))/2;
    while (1==1)
        n = 2*n;           %区间个数增加一倍 == 区间宽度减少一半
        x = linspace(a, b, n+1); %把区间分割成n等分
        h = x(2) - x(1);    %区间宽度
        y = g(x);          %对被积函数进行采样
        T2 = (h/2) * (y(1) + 2*sum(y(2:n)) + y(n+1)); %复化梯形公式
        e0 = abs(T2-T1);    %迭代前后误差
        if (e0<err), break; end;
        i = i + 1; if (i>D), break; end;
        T1 = T2;
    end;
    I = T2;

    %被积函数g(x), 假定为 y = sin(x)
    function y = g(x)
        y = sin(x);
    end;
end;
```

Q11.3 龙贝格积分

用附送的源代码求解 Q11.1 中的定积分，读懂其中的 `varagin` 变量的灵活应用。

```
% 龙贝格积分
% 调用形式:
%   [q, ea, iter] = romberg(func, a, b, es, maxit, p1, p2, ...);
% func 被积函数名称或指针; a, b 积分上下限;
% es 积分精度;   maxit 最大迭代次数
% varagin 其它输入参数 (func函数调用时的附加参数)
% q 积分值; ea 退出时实际积分精度; iter 实际迭代次数
function [q, ea, iter] = romberg(func, a, b, es, maxit, varagin)
n = 1;
I(1,1) = trap(func, a, b, n, varagin{:}); %计算 T1
iter = 0;
while iter < maxit
    iter = iter + 1;
    n = 2^iter;
    I(iter+1, 1) = trap(func, a, b, n, varagin{:});
    for k = 2:(iter + 1)
        j = 2 + iter - k;
        I(j, k) = (4^(k-1)*I(j+1, k-1) - I(j, k-1))/(4^(k-1)-1);
    end;
    ea = abs((I(1, iter+1) - I(2, iter))/I(1, iter+1))*100;
    if (ea<es), break; end;
end;
q = I(1, iter+1);
```

```

function I = trap(func, a, b, n, varargin)
% trap: composite trapezoidal rule quadrature
% I = trap(func, a, b, n, p1, p2, ...):
%             composite trapezoidal rule
% input:
% func = name of function to be integrated
% a, b = integration limits
% n = number of segments (default = 100)
% p1, p2, ... = additional parameters used by func
% output:
% I = integral estimate

if nargin<3,error('at least 3 input arguments required'),end
if ~(b>a),error('upper bound must be greater than lower'),end
if nargin<4|isempty(n),n=100;end
x = a; h = (b - a)/n;
s=func(a, varargin{:});
for i = 1 : n-1
    x = x + h;
    s = s + 2*func(x, varargin{:});
end
s = s + func(b, varargin{:});
I = (b - a) * s/(2*n);

```

十二、常微分方程求解

Q12.1 欧拉法

上机要求:

- 1) 读懂附送的源代码
- 2) 修改 Fxy 函数以及 x0,y0 等参数, 看看结果
- 3) 自行编写梯形法、改进欧拉法
- 4) 测试自己编写的算法是否正确, 对比五种欧拉法的误差

附送的源代码: 保存在同一个文件里 (Euler_Test.m)

```
% 一阶常微分方程求解的欧拉法
function Euler_Test
x0 = 0; h=0.1; y0 = 1; n = 10;
[~, yy1] = Euler_Diff(x0, y0, h, n, 1); % 前向欧拉法
[~, yy2] = Euler_Diff(x0, y0, h, n, 2); % 后向欧拉法
[xx, yy4] = Euler_Diff(x0, exp(-0.1), h, n, 4, y0); % 中点欧拉法
yy0 = exp(xx); %假定的微分方程精确解, 用于对比
figure, plot(xx, [yy0, yy1, yy2, yy4]), grid on;
title('欧拉法求解结果');
legend('精确', '前向', '后向', '中点');
```

```
%前向欧拉法迭代:  $y(k) = y(k-1) + h * F_{xy}(x(k-1), y(k-1))$ 
function yk = do_Forward_Euler(xk_1, yk_1, h)
yk = yk_1 + h * Fxy(xk_1, yk_1);

%后向欧拉法迭代:  $y(k) = y(k-1) + h * F_{xy}(x(k), y(k))$ 
function yk = do_Backward_Euler(xk, yk_1, h)
yk = yk_1;
for s = 1:100
    yk1 = yk_1 + h * Fxy(xk, yk);
    e1 = abs(yk1 - yk); %迭代误差
    if (e1 < 0.001), break; end; %达到误差精度, 则迭代退出
    yk = yk1;
end;
yk = yk1;
```

```

%中点欧拉法迭代:  $y(k) = y(k-2) + 2*h*F_{xy}(x(k-1), y(k-2), y(k-1))$ 
function yk = do_Midpoint_Euler(xk_1, yk_2, yk_1, h)
    yk = yk_2 + 2*h*Fxy(xk_1, yk_1);

% Fxy(x,y) 函数, 不同的微分方程可以修改这里
function z = Fxy(x, y)
    %z = 2*y + 5*x*y;
    z = exp(x); %微分方程的精确解为:  $y = \exp(x)$ , 用于测试求解精度

```

```

% 假定常微分方程为:  $y' = F_{xy}(x, y)$ ,  $y(x_0) = y_0$ ;
% 输入参数: x0, h, n: x的求解点。从x0开始, 每隔h求解一个对应的y值, 共n个
% OP: 欧拉法种类。OP=1, 前向; OP=2, 后向;
%     OP=3, 梯形; OP=4, 中点; OP=5, 改进欧拉法
function [xx,yy] = Euler_Diff(x0, y0, h, n, OP, y1)
    if ( nargin<5), OP = 1; end; %默认设为前向欧拉法
    xx = x0 + (1:n)'*h;
    yy = zeros(n,1);
    for k=1:n
        if (k==1), xk_1 = x0; yk_1 = y0;
            else xk_1 = xx(k-1); yk_1 = yy(k-1);
        end;
        xk = xx(k);
        switch(OP)
            case 1, yy(k) = do_Forward_Euler(xk_1, yk_1, h); %前向
            case 2, yy(k) = do_Backward_Euler(xk, yk_1, h); %后向
            %case 3, yy(k) = do_Trapezoid_Euler(xk_1, xk, yk_1, h); %梯形
            case 4
                if (k==1), yk_1 = y1; yk_2 = y0;
                    elseif (k==2), yk_1 = yy(1); yk_2 = y1;
                    else yk_1 = yy(k-1); yk_2 = yy(k-2);
                end;
                yy(k) = do_Midpoint_Euler(xk_1, yk_2, yk_1, h); %中点
            %case 5, yy(k) = do_Modified_Euler(xk_1, xk, yk_1, h); %改进
            otherwise, break; %非法OP
        end;
    end;
end;

```

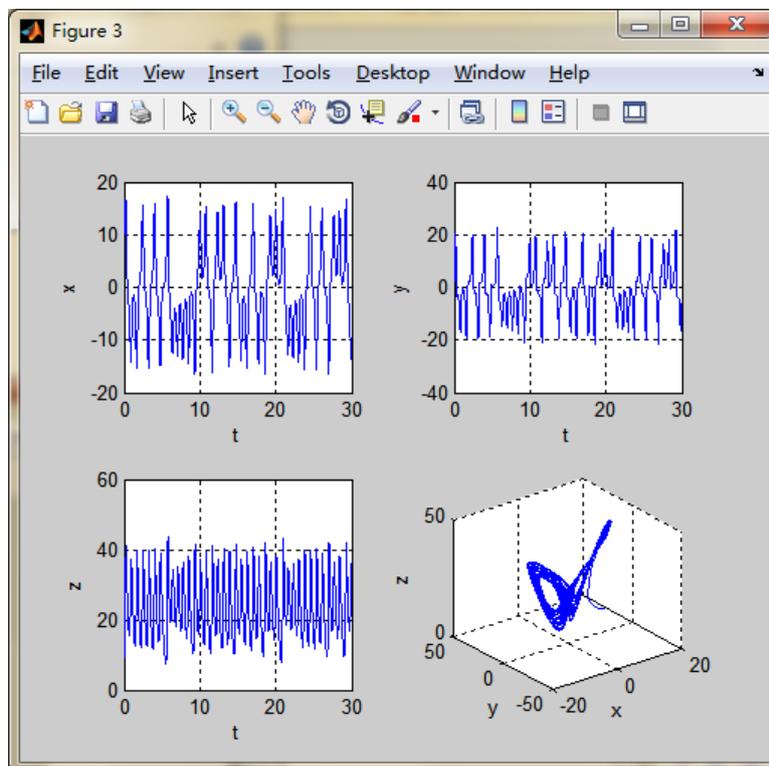
Q12.2 龙格库塔法

下面的代码实现了洛仑兹方程曲线的绘制。其中自定义函数 `do_Lorenz` 需要保存到 M 文件中。请读懂代码并绘制出曲线（函数 `ode45` 是龙格库塔法）。

```
% Lorenz曲线绘制
function do_Lorenz
[t,y] = ode45(@Lorenz,[0,30],[12,2,9]); %龙格库塔法求解洛仑兹方程
figure
subplot(221), plot(t,y(:,1)), grid on; xlabel('t'); ylabel('x');
subplot(222), plot(t,y(:,2)), grid on; xlabel('t'); ylabel('y');
subplot(223), plot(t,y(:,3)), grid on; xlabel('t'); ylabel('z');
subplot(224), plot3(y(:,1),y(:,2),y(:,3)), grid on;
xlabel('x'); ylabel('y'); zlabel('z');

function dy = Lorenz(~,y)
dy = zeros(3,1);
dy(1) = 10*(-y(1)+y(2)); dy(2) = 28*y(1)-y(2)-y(1)*y(3);
dy(3) = y(1)*y(2)-8*y(3)/3;
```

运行效果如下：



更改程序中的多个参数，观看不同的洛仑兹曲线。

十三、综合练习（一）

上机目标：综合回顾前面的实验题，学会应用 MATLAB 编程解决问题。

上机要求：

- 1) 按题目要求调用前面实验题的程序，或者自行编写简单语句
- 2) 九道题目至少完成五题以上，越多得分越多；
- 3) 作业提交源代码文件+Command Window 调试文本

综合性题目：

- 1、用二分法求方程 $x^2 - x - 1 = 0$ 的正根，要求准确到小数点后第 1 位。
- 2、你能用几种方法将 $x = \text{tg}(x)$ 化为适合于迭代的形式？并求 $x = 4.5$ （弧度）附近的根。
- 3、用牛顿法计算 $\sqrt{3}$ 具有 4 位有效数字的近似值。
- 4、用牛顿法计算 $P(z) = z^4 - 3z^3 + 20z^2 + 44z + 54 = 0$ 方程在 $z_0 = 2.5 + 4.5i$ 附近的零点。（注 z 为复变量，精度可以自行定义）
- 5、用高斯-约当消元法求下列方程的解：

$$\begin{cases} x_1 + 2x_2 + x_3 = 2 \\ -2x_1 - 2x_2 - x_3 = -3 \\ 2x_1 - 3x_2 - 2x_3 = -1 \end{cases}$$

- 6、调用 MATLAB 的 LU 函数，对下列两个矩阵进行 LU 分解，并观察 L、U、P 的结果，解释为什么会出现错乱，怎么看懂出来的结果。

$$A = \begin{pmatrix} 2 & -1 & 1 \\ -1 & -2 & 3 \\ 1 & 3 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 4 \\ 5 & 2 & 1 \end{pmatrix}$$

- 7、编写 MATLAB 代码求解下列矩阵的 $\|\cdot\|_\infty, \|\cdot\|_1, \|\cdot\|_2$ 和不同范数下的条件数。（提示：求特征值函数为 **eig**，条件数函数为 **cond**）

$$A = \begin{pmatrix} 2 & -1 & 1 \\ -1 & -2 & 3 \\ 1 & 3 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 4 \\ 5 & 2 & 1 \end{pmatrix}$$

- 8、编写小段 MATLAB 代码来判断下列两个方程组的雅克比迭代、高斯-赛代尔迭代的收敛可能性。

$$(a) \begin{cases} x_1 + 0.4x_2 + 0.4x_3 = 1 \\ 0.4x_1 + x_2 + 0.8x_3 = 2 \\ 0.4x_1 + 0.8x_2 + x_3 = 3 \end{cases} \quad (b) \begin{cases} x_1 + 2x_2 - 2x_3 = 1 \\ x_1 + x_2 + x_3 = 1 \\ 2x_1 + 2x_2 + x_3 = 1 \end{cases}$$

9、用幂法、反幂法分别求下列矩阵的绝对值最大、最小的特征根和特征向量。

$$A = \begin{pmatrix} 2 & -1 & 1 \\ -1 & -2 & 3 \\ 1 & 3 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 4 \\ 5 & 2 & 1 \end{pmatrix} \quad C = \begin{pmatrix} 5 & 1 & 2 \\ 1 & 9 & 4 \\ 2 & 4 & 20 \end{pmatrix}$$

十四、综合练习（二）

一类迭代法的 MATLAB 编程

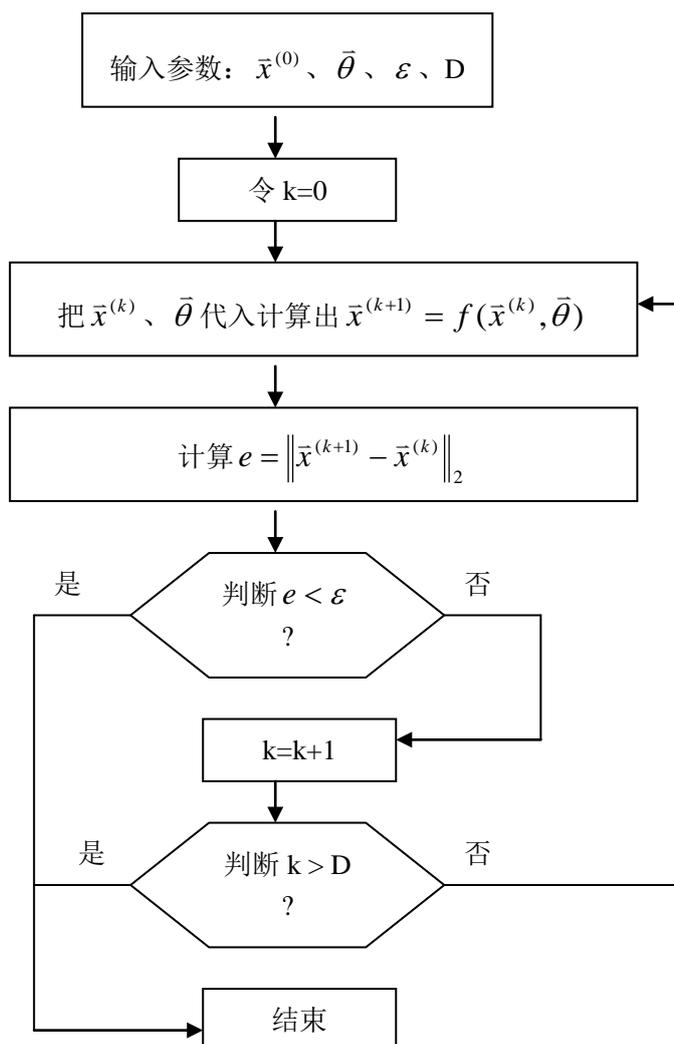
设有迭代表达式如下：

$$\bar{x}^{(k+1)} = f(\bar{x}^{(k)}, \bar{\theta})$$

其中，初始向量 $\bar{x}^{(0)}$ 和参数向量 $\bar{\theta}$ 为已知值。用 MATLAB 编程求出迭代 D 次的所有 $\bar{x}^{(k)}, k = 1, \dots, D$ 。在迭代循环中，提前结束循环的条件是： $(0 < \varepsilon < 1)$

$$\|\bar{x}^{(k+1)} - \bar{x}^{(k)}\|_2 < \varepsilon$$

这是一个典型的迭代算法框架，求解非线性方程的二分法、牛顿法、迭代法；求解线性方程组的迭代法；求解矩阵特征值的幂法、反幂法等等均属于这一类。从编程题目的思路和要求分析，画出其流程图如下：



用 MATLAB 编程实现上述的流程图，其中迭代循环用 for 或 while 语句均可。当 D 未知时，采用 while 循环；D 已知时，for 循环更明了一些。为一般性起见，下面用 while 循环实现。另外在计算 f() 函数中，若 f() 函数较复杂，则一般把 f() 函数编写成子函数，这样程序模块化，使得结构、调试排错等更加简单容易。

MATLAB 实现的源码如下：（源码保存到 “ddf.m” 文件中）

```

1      % MATLAB实现通用迭代法
2      % 输入参数：
3      %   x0 列向量，也可以是单个数值，看具体问题
4      %   e0 数值，迭代退出的误差
5      %   D 数值，最大迭代次数。没确定最大迭代次数的话，D=-1
6      %   Theta 参数向量，可以有，也可以无，看具体问题。不需要额外参数的话，Theta=[]
7      % 输出参数：
8      %   XX 矩阵，每一列存放一个 xk 向量
9      %   D1 数值，实际迭代次数
10     function [XX,D1] = ddf(x0,e0,D,Theta)
11     N = length(x0); %获得 x0 中的成员个数
12     if (D ~= -1), XX = zeros(N,D); else XX = []; end; %初始化 XX 矩阵
13     k = 0;
14     while (1)
15         x1 = ddf_fun(x0,Theta); %调用子函数计算 x(k+1)
16         XX(:,k+1) = x1;
17         e1 = norm(x1-x0); %计算迭代误差
18         if (e1<e0), break; end;
19         if (k>=D), break; end;
20         k = k+1;
21         x0 = x1;
22     end;
23     D1 = k;

```

对于具体的迭代表达式，需要编写不同的 *ddf_fun* 子函数。下面举 2 个例子。

1、用迭代法求解非线性方程的根

非线性方程：

$$\cos(x) - x = 0 \Rightarrow x^{(k+1)} = \cos(x^{(k)})$$

设 $x^{(0)} = 1$ ， $D=1000$ ， $\varepsilon = 0.001$ 迭代表达式中不需要额外参数，所以 $\text{Theta} = []$ 。其

ddf_fun 子函数编写如下：（源码保存到 “ddf_fun.m” 文件中）

```

1      function x1=ddf_fun(x0,Theta)
2      x1 = cos(x0);

```

在 MATLAB 的 Command Window 中输入如下语句进行程序调试和使用。

```

Command Window
>> x0 = 1; D = 1000; e0 = 0.001;
>> [XX, D1]=ddf(x0, e0, D, []);
>> D1

D1 =

    16

>> XX(D1)

ans =

    0.7396

>> cos(XX(D1))

ans =

    0.7388
    
```

Callouts in the image:

- 设置输入参数 (Set input parameters) - points to the first line of code.
- 调用 ddf 函数 (Call ddf function) - points to the second line of code.
- 查看实际迭代次数 (Check actual iteration count) - points to the output '16'.
- 查看最终解 (Check final solution) - points to the output '0.7396'.
- 验证解的正确性 (Verify the correctness of the solution) - points to the output '0.7388'.

因为最大误差设为 0.001，因此程序迭代到 16 次就符合条件退出了。尝试把最大误差设为 1.0e-10，就会得出非常精确的解。

2、用幂法求解矩阵的谱半径和对应的特征向量

幂法的迭代表达式为：

$$\bar{v}^{(k+1)} = A\bar{v}^{(k)}, \quad \lambda^{(k)} = \frac{1}{N} \sum_{i=1}^N \frac{\bar{v}_i^{(k+1)}}{\bar{v}_i^{(k)}}$$

设矩阵 A 的行数和列数均为 N， $x^{(0)} = rand(N,1)$ ，D=1000， $\varepsilon = 0.001$ ，迭代表达式中需要额外参数 A，所以 Theta = A。

由于幂法的迭代表达式中，判断退出条件的应该是 Lamda 的值，所以把 ddf 函数中的判断语句进行更改。除了误差判断外，ddf_fun 子函数由于很简单，所以直接写在主程序中。更改后，完整的幂法程序如下：（保存到“mifa.m”文件）

```

1      % MAILAB实现幂法，在通用迭代法上进行更改
2      % 输入参数：
3      %   x0 列向量，初始随机向量
4      %   e0 数值，迭代退出的误差
5      %   D 数值，最大迭代次数
6      %   Theta 参数向量，幂法中就是待求谱半径的矩阵A
7      % 输出参数：
8      %   Lamda 数值，求解出来的谱半径值
9      %   V1 列向量，归一化后的特征向量
10     %   D1 数值，实际迭代次数
11     function [Lamda,V1,D1] = mifa(x0,e0,D,Theta)
12     -   Lamda0 = 1;
13     -   k = 0;
14     -   while (1)
15     -       x1 = Theta * x0; %计算 x(k+1)
16     -       Lamda = mean(x1./x0); %求谱半径
17     -       %只有迭代2次后才计算误差
18     -       if (k>1), e1=norm(Lamda-Lamda0); else e1 = e0+1; end;
19     -       if (e1<e0), break; end;
20     -       if (k>=D), break; end;
21     -       k = k+1;
22     -       x0 = x1;
23     -       Lamda0 = Lamda;
24     -   end;
25     -   D1 = k;
26     -   V1 = x1/max(abs(x1));

```

上述程序中取消了 *ddf_fun* 子函数，因此只需要一个主函数即可。这样程序变得更见简短明了，但是损失了通用性。可以根据具体问题选择 *ddf_fun* 子函数需不需要，一般来说当迭代表达式很复杂时，最好用 *ddf_fun* 子函数。

在 MATLAB 的 Command Window 中输入如下语句进行程序调试和使用。

```

Command Window
>> A=[1 2 1; 2 1 2; 3 4 2]

A =

     1     2     1
     2     1     2
     3     4     2

>> x0=rand(size(A,1),1); D=1000; e0=0.001;
>> [Lamda,V1,D1]=mifa(x0,e0,D,A);
>> D1

D1 =

     6

>> Lamda

Lamda =

     5.8101

>> V1

V1 =

     0.4604
     0.6072
     1.0000

>> eig(A)

ans =

     5.8100
    -0.3548
    -1.4552
    
```

矩阵 A

x0 为随机列向量

调用幂法函数

查看实际迭代次数

查看谱半径

精确的谱半径

查看特征向量

3、用二分法求解非线性方程

二分法的迭代表达式稍微复杂一些，区间的两个端点的迭代更新由二分法的前提条件判断决定，难以写成简单的数学表达式。因此根据迭代法的通用编程框架，把二分法流程列出如下：

- 1) 输入参数：初始区间端点 a 、 b 、最大迭代次数 D 、最大误差 $e0$ 、非线性方程的函数名称 ff （即方程为 $ff(x)=0$ ）
- 2) 计算 $ay = ff(a)$ 、 $by = ff(b)$ ，若 $(ay*by > 0)$ 则非法区间，退出。
- 3) 令 $x0 = (a + b)/2$ ，计算 $y0 = ff(x0)$
- 4) 若 $(y0*ay > 0)$ 则 $b = x0$ ， $by = y0$ ，否则 $a = x0$ ， $ay = y0$
- 5) 迭代次数加 1，若到达最大迭代次数则退出
- 6) 若 $(\|a - b\|_2 < e0)$ 则退出，否则跳转到第 3) 步。

按照上述流程，编写 MATLAB 源码如下：（保存到“eff.m”文件）

```
1 % MATLAB实现二分法，在通用迭代法上进行更改
2 % 输入参数：
3 % a,b 数值，初始区间端点
4 % e0 数值，最大允许误差
5 % D 数值，最大迭代次数
6 % ff 字符串，方程对应的函数名称
7 % 输出参数：
8 % x0 数值，方程的解
9 % D1 数值，实际迭代次数
10 function [x0,D1] = eff(a,b,e0,D,ff)
11 x0 = Inf; D1 = 0; %非法解
12 ay = feval(ff,a); by = feval(ff,b);
13 if (ay*by > 0), return; end; %不符合二分法前提条件，非法区间
14 while (1)
15 x0 = (a+b)/2; y0 = feval(ff,x0);
16 if (y0*ay > 0), a = x0; ay = y0; else b = x0; by = y0; end;
17 D1 = D1+1;
18 if (D1>D), break; end;
19 if (norm(a-b) < e0), break; end;
20 end;
```

其中用到了字符串指针，由 ff 变量指向待求解的方程函数名称。该函数名称可以任意取，编写如下：（保存到“eff_func.m”文件）

```
1 % 二分法函数需调用的子函数，代表方程表达式
2 function y = eff_func(x)
3 y = x - cos(x);
```

假定要求方程：

$$x - \cos(x) = 0$$

的解，那么到 MATLAB 的 Command Window 中输入如下调试语句进行测试：

```
Command Window
>> a=0.5; b=1; e0=0.001; D=1000; ff='eff_func';
>> [x0,D1] = eff(a,b,e0,D,ff);
>> D1
D1 =
    9
>> x0
x0 =
    0.7393
```

实际迭代次数

方程的根

可以看到，用二分法求解的方程根与前面用迭代法求解的结果精度差不多，因为最大允许误差均为 0.001。

Q14.1 迭代思维编程

- 1) 重复上述三个问题的源码，以及测试，要求达到一样的结果。
- 2) 改变迭代的最大允许误差，例如改成 0.00001、1.0e-8 等，观看结果的精度。
- 3) 读懂上述三个问题的思路和源码，模仿这些源码编写一个用牛顿法程序求解如下方程：

$$x - \cos(x) = 0$$

按照牛顿法，该方程的迭代表式为：

$$x^{(k+1)} = x^{(k)} - \frac{x^{(k)} - \cos(x^{(k)})}{1 + \sin(x^{(k)})} = \frac{\sin(x^{(k)}) + \cos(x^{(k)})}{1 + \sin(x^{(k)})}$$

迭代初始值取为： $x^{(0)} = 0.5$ 。

- 4) 编写一个用雅克比迭代法程序求解如下线性方程组：

$$\begin{cases} x_1 + 0.4x_2 + 0.4x_3 = 1 \\ 0.4x_1 + x_2 + 0.8x_3 = 2 \\ 0.4x_1 + 0.8x_2 + x_3 = 3 \end{cases}$$

雅克比法的迭代表式为： $(A = L + D + U)$

$$\vec{x}^{(k+1)} = -D^{-1}(L + U)\vec{x}^{(k)} + D^{-1}\vec{b}$$

附：实验报告样例

MATLAB 基础（1）

9月22日 10:00-11:30

实验人：XXX 实验地点：信息楼 121

实验目标：熟悉 MATLAB 语言的编程环境、基本编程指令、变量等。

实验步骤：

- 1 进入 MATLAB 环境；
- 2 熟悉各种菜单命令（详细说明。。。。。）；
- 3 MATLAB 语言基本语法形式（详细说明。。。。。）；
- 4 MATLAB 语言变量、算术运算、关键词及在线帮助等（详细说明。。。。。）；
- 5 把上机内容拷贝到 U 盘。

实验题解答：

Q1.1

Q1.2

Q1.3

Q1.4

图形界面与数据分析：

本次实验无

实验结果：

通过本次实验，我基本上对 MATLAB 语言环境有了一个较感性的认识，能够运用 MATLAB 语言进行简单的算术运算和简单编程。达到了基本的实验目标。