目录

第三章 分支语句和编程设计.....1
3.1 自上而下的编程方法简介.....1
3.3 关系运算符和逻辑运算符.....4
3.3.1 关系运算符.....4
3.3.2 小心==和~=运算符.....5
3.3.3 逻辑运算符.....6
例 3.1.....7
3.3.4 逻辑函数.....7
测试 3.1.....7
3.4 选择结构(分支语句).....8
3.4.1 if 结构.....8
3.4.2 if 结构举例.....9
例 3.2.....9
例 3.3.....12
3.4.3 关于 if 结构使用的注意事项.....13
例 3.4.....14
3.4.4 switch 结构.....15
3.4.5 try/catch 结构的应用.....16
测试 3.2.....17
3.5 附加的画图特性.....17
3.5.1 控制 x，y 轴绘图的上下限.....18
3.5.2 在同一坐标系内画出多个图象.....20
3.5.3 创建多个图象.....20
3.5.4 子图象.....21
3.5.5 对画线的增强控制.....22
3.5.6 文本字符串的高级控制.....22
3.5.7 极坐标图象.....23
例 3.5.....24
例 3.6.....25
例 3.7.....26
3.5.8 注释并保存图象.....28
测试 3.3.....30
3.6 程序调试的进一步说明.....30
3.7 总结.....33
3.7.1 好的编程习惯的总结.....34
3.7.2 matlab 总结.....34
3.8 练习.....34
3.1.....34
3.2.....34
3.3.....34
3.4.....35
3.5.....35
3.6.....35
3.7.....35
3.8.....35
3.9.....35
3.10.....36
3.11.....36
3.12.....36
3.13.....36

3.14.....37

第三章 分支语句和编程设计

在前面的章节中，我们开发了几个完全运转的 MATLAB 程序。但是这些程序都十分简单，包括一系列的 MATLAB 语句，这些语句按照固定的顺序一个接一个的执行。像这样的程序我们称之顺序结构程序。它首先读取输入，然后运算得到所需结果，打印出结果，并退出。至于要多次重复运算程序的某些部分是没有办法的，也不能根据输入的值，有选择地执行程序的某些部分。

在下面的两章中，我们将向大家介绍大量的 MATLAB 语句，这些语句允许我们来控制中语句的执行顺序。有两大类控制顺序结构：**选择结构**，用选择执行特定的语句；**循环结构**，用于重复执行特定部分的代码。选择结构将会本章讨论，循环结构我们将会在第四章讨论。

随着选择和循环介绍，我们的程序也将变得复杂，对于解决问题来说，将会变得简单。为了帮助大家避免在编程过程中出现大量的错误，我们将向大家介绍正式的编程步骤，即自上而下的编程方法。我们也会向大家介绍一些普通的算法开发工具即伪代码。

3.1 自上而下的编程方法简介

假设你是在工厂工作的工程师，为了解决某些问题，你要编写一个程序。你如何开始呢？当遇到一个新问题时，我们的心里会自然而然的产生这样的想法：马上坐在计算机前，开始编程，而不用浪费大量的时间思考我们所要解决的问题是什么？用这种不切实际的想法来编一些非常小的程序可能会成功。但在现实中，问题可能会非常的大，程序员再用这种方法编程将会陷入困境。对于一个大的程序来说，在编写代码之前你要通盘的思考你所要面临的问题和解决的方法。在本节中，我们将向大家介绍正式的编程设计步骤，然后应用这个步骤来编写本书所有的大的应用程序。对于我们所遇到一些简单的例子来说，这个步骤好像有些画蛇添足。但是当我们解决的问题变得越来越大的时侯，这个步骤将会变得异常重要。

当我还没有毕业的时侯，一个教授喜欢说：“编程很简单，因为我知道在编程的过程的困难”。当我们离开学校，在工厂从事于大规模软件工程编写时，我深深地理解了它所说的话。我发现在工作中我遇到的大多数困难都是对所要解决问题的理解。一旦你真正理解了问题，你就会把这个问题分解成许多小的问题，更加易于管理的小块，然后逐一解决某一个小块。自上而下的编程方法是我们正规编程设计的基础。我们现在向大家介绍这些在图 3.1 说明的步骤细节。步骤如下:

1.清晰地陈述你所要解决的问题

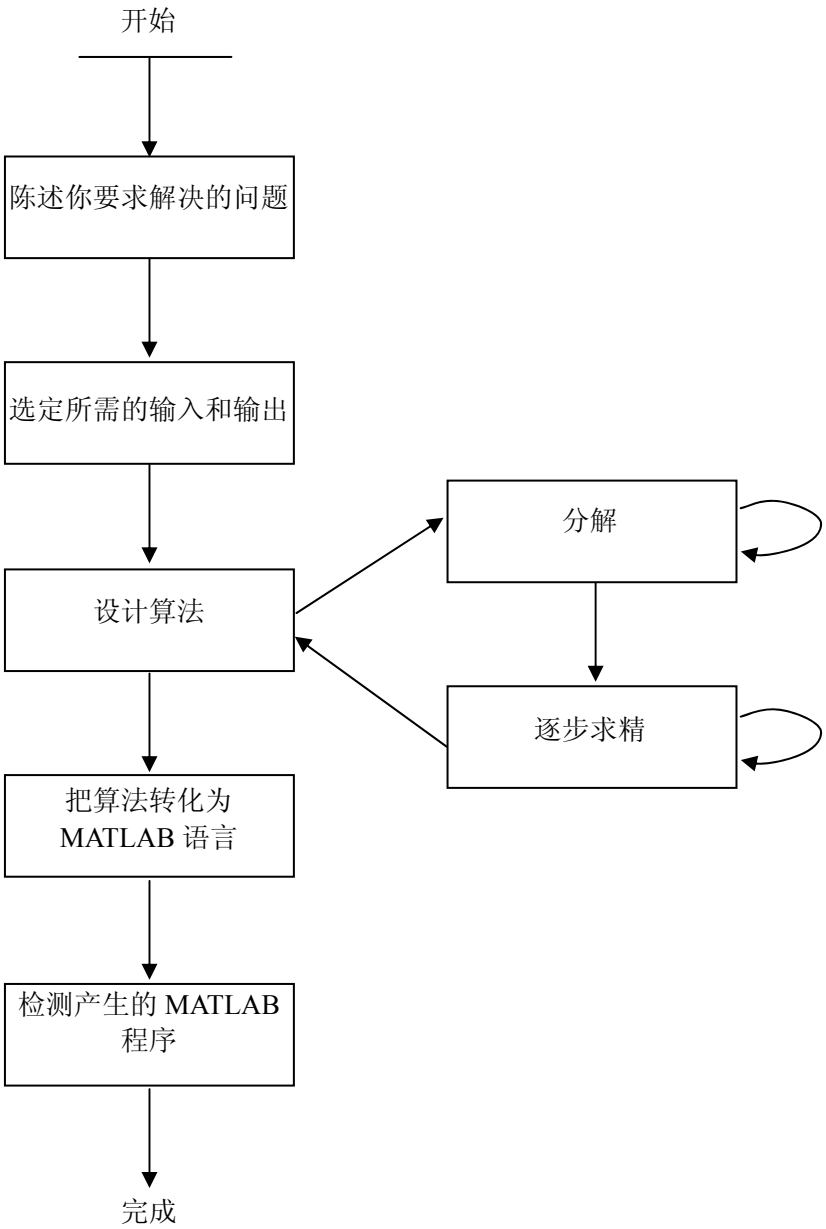
编写的程序大多数情况下要满足一些感觉上的需要，但这种需要不一定能够被人清晰地表达出来。例如，用户需要一个解线性方程组的表达式。像这样的要求就不够清楚，程序员就很难编出一个使他满意的程序。他必须弄清楚要有多少问题需要解决?在这些方程式中有没有对称的形式使我们的开发变得简单?程序设计者必须和使用者讨论所需的程序，他们必须要对完成的任务有一个精确细致的描述。对问题清晰的描述可以防止误解，并且能够帮助程序员合理的组织他的思想。上面的例子对问题合适的陈述应为:

设计一个用于解决联立线性方程组的程序，这些方程中未知数的系数为实数，最多有 20 个未知数。

2.定义程序所需的输入量和程序所产生的输出量

指定输入量和输出量，只有这样新的程序才能适应全过程计划。在这个例子中方程式的系数可能有其预先存在的顺序，我们的新程序必须能按照顺序读取它们。相似地，也需要产生出这个程序所要求的结果，即输出量，我们还要以一定的格式打印出来。

3.设计你的程序得以实现的算法



算法是指为某个问题找到答案一步接一步的程序。在这个阶段自上而下的编程方法发挥了作用。编程设计者开始对这个问题进行逻辑划分，把它逐步分解为一个又一个子工作。这个过程叫做分解(decomposition)。如果一些子工作还是比较大，设计者还可以把他它分解成更小的块。这个过程将会继续到问题被分解成许多简单且易理解的小块为止。

在问题被分解成小块之后，每一个小块要被进一步的求精，这个过程叫做逐步求精(stepwise refinement)。在这个过程中，设计者开始于对本小块代码总括性的描述，然后开始一步一步地定义所需的函数，越来越具体，直到他能够转化为 MATLAB 语句。逐步求精的过程中，我们要用到的伪代码将会在下节为大家介绍。

在算法开发过程中，这个方法是非常有用的。如果设计者真正理解了解决问题这个些步骤，他将会对问题进行分解和逐步求精。

4.把算法转化为代码

如果分解和逐步求精的过程已经顺利完成，那么这一步将会异常地简单。所有程序员都会将伪代码一句一句地转化为合适地 MATLAB 语句。

5 检测产生的 MATLAB 程序

这一步是真正的拦路虎。首先，程序的每一部分将会被单独地检测，如果有可能的话，整个程序还要被检测一遍。在我们检测程序时，我们必须证明所有合法输入数据值都

能够正常运行。用标准的输入值检测程序，看它是否产生了值。如果在一个程序中执行的算法包含了不同的分支，你必须检测每一个分支，以保证产生正确的答案。大程序在交付大众使用之前，必须经过一系列地检测(图 3.2)。检测的第一步有时被称为单元检测(unit testing)。在单元检测过程中，程序的子程序将会被独立地检测以证明它的正确性。当单元检测结束之后，这个程序将进行一系列的组合，把独立的子程序联合产生出最后的程序。程序第一步的联合通常只包括很少的子程序。通过组合这些子程序，经常用检查子程序或函数之间的联系。在一系列地组合过程中，越来越多的子程序被加了进来，直到整个程序的完成。在每一次组合的过程中，每一个错误都会被发现并在进行下一次组合之前纠正过来。

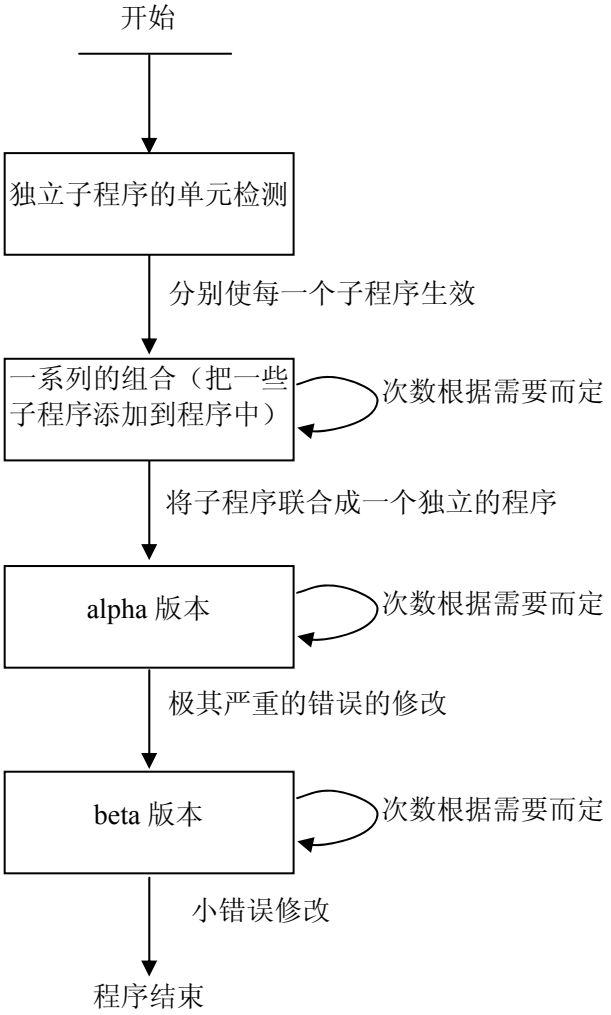


图 3.2 大程序典型地调试过程

在整个程序被组合之后，调试继续进行。程序第一个版本我们通常称之为“alpha 版本”。程序员和其他有机会接近它的人可以想尽一切办法应用它，以发现其中的漏洞，然后改正之。当许许多多大的错误从程序中去除，一个新的版本出现了，我们称之“beta 版本”。beta 版本就要公开地发行给天天需要这个程序工作的人。这些用户使这个程序在不同的环境下，在不同的输入条件下工作，会发现许多的错误，并报告给程序员。当这些错误被更正后，这个程序就能够发行给公众使用了。因为本书中的程序都比较小，没有必要进行上述的大规模的检测。但是我们会遵循基本的调试原则。

程序设计的基本步骤如下:

1.清晰地陈述出你要解决的问题。
2.确定程序所需地输入量和程序所产生的输出量。
3.为你的程序设计算法
4.将算法转化为 MATLAB 语句

5.调试 MATLAB 程序

好的编程习惯
遵循上面的步骤编写可靠，易理解的 MATLAB 程序。

在大的编程项目中，花在编程序的时间是出奇的少。Frederick P Brooks 在他的 the Mythical Man-Month 书中写道，对于大的软件工程来说，三分之一的时间花在计划如何来做上(第一步到第三步)，六分之一的时间花在编写程序上，近一半的时间用来调试程序。而我们能做的只有压缩调试用的时间。在计划阶段做好充分的准备和在编程过程使用良好的编程习惯，这样会大大降低我们调试所用的时间。好的编程习惯能减少出错的数量，也能使别人迅速地找出其中的错误。

3.2 伪代码的应用

作为我们设计步骤的一部分，描述出你要执行的算法是非常必要的。算法的描述有一种标准形式，能让你和大家都能理解，这种描述将帮助你的内容转化为 MATLAB 代码。我们用于描述算法的标准形式叫做构造(constructs 有时也称 structure)。用这些结构描述出的算法，我们称之为结构化算法。当在我们在 MATLAB 程序中执行这个算法时，产生的程序叫做结构化程序。

我们可以用伪代码的形式建立算法的结构。伪代码是 MATLAB 和英语的混合体。和 MATLAB 一样，它是结构化的，一行表达一个明确的意思或代码的片段，但每一行的描述用的是英语或其他人类语言。伪代码的每一行都应用普通简单且易于理解的英语或中文描述。因为修改简单灵活，所以伪代码在开发算法的过程中非常的有用。因为伪代码给编辑器或字处理器(通常用于编写 MATLAB 程序)的，而不需要其他的可视化功能。例如下面是例 2.3 的算法伪代码

Prompt user to enter temperature in degrees Fahrenheit
Read temperature in degrees Fahrenheit(temp_f)
temp_k in kelvins ← (5/9) * (temp_f - 32) + 273.15
Write temperature in kelvins

注意用向左指的箭头←替代等号(=)指出一个值将存储到对应的变量中，这样就避免了赋值号与等号的混淆。在把它们转化为 MATLAB 代码之前，伪代码将有助于你思想的组织。

3.3 关系运算符和逻辑运算符

选择结构的运算由一个表达式控制的，这个表达式的结果只有 true(1)和 false(0)。有两种形式的运算符可以在 MATLAB 中关系得到 true/false：关系运算符和逻辑运算符。

跟 C 语言一样，MATLAB 没有布尔型和逻辑数据类型。MATLAB 把 0 值作为结果 false，把所有的非 0 值作为结果 ture。

3.3.1 关系运算符

关系运算符是指两数值或字符操作数的运算符，这种运算将会根据两操作数的关系产生结果 true 或 false。关系运算的基本形式如下

$$a_1 \; op \; a_2$$

其中 a_1 和 a_2 是算术表达式，变量或字符串，op 代表表 3.1 中的关系运算符中的一个。如果两者的关系为真（true）时，那么这个运算将会返回 1 值；否则将会返回 0 值。

表 3.1 关系运算符

运算符	运算
==	等于
~=	不等于
>	大于
>=	大于或等于

<	小于
<=	小于或等于

下面是一些关系运算和它的结果运算结果

3 < 4	1
3 <= 4	1
3 == 4	0
3 > 4	0
4 <= 4	1
'A' < 'B'	1

最后一个运算得到的结果为 1，是因为字符之间的求值要按照子母表的顺序。

关系运算符也可用于标量与数组的比较。例如，如果 $a=\begin{bmatrix}1 & 0\\ -2 & 1\end{bmatrix}$ 和 b=0，那么表达式 a>b 将会产生结果 $\begin{bmatrix}1 & 0\\ 1 & 0\end{bmatrix}$。关系运算符也可比较两个关系运算符，只要两个数组具有相同的大小。例如 $a=\begin{bmatrix}1 & 0\\ -2 & 1\end{bmatrix}$，$b=\begin{bmatrix}0 & 2\\ -2 & -1\end{bmatrix}$，表达式 a>=b 将会产生结果 $\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}$。如果这个数组具有不同的大小，那么将会产生运行时错误。

注意因为字符串实际上是字符的数组，关系运算符也比较两个相同长度的字符串。如果它们有不同的长度，比较运算将会产生一个错误。在第六章中我们将会学到一个更普遍的方法。等于关系运算符由两个等号组成，而赋值运算符只有一个等号。它们是完全不同的两个符号，初学者极易混淆。符号==是一个比较运算符，返回一个逻辑数，而符号=是将等号右边的表达式的值赋给左边的变量。当进行比较运算的时候，初学者经常用误用符号=。

常见编程错误

小心谨慎不要混淆了等于关系运算符（==）和赋值运算符（=）。

在运算的层次中，关系运算在所有数学运算的之后进行。所以下面两个表达式是等价的，均产生结果 1。

```
7 + 3 < 2 + 11
(7 + 3)< (2 + 11)
```

3.3.2 小心==和~=运算符

等于运算符（==）如果两变量值相同将会返回变量值 1，如果不同将返回 0。

不等运算符（~=）如果两变量值不同则返回 1，相则返回 0。

用这两个运算符比较两个字符串他是安全的，不会出现错误。但对两个数字数据的比较，将可能产生异想不到的错误。两个理论上相等的数不能有一丝一毫的差别，而在计算机计算的过程中出现了近似的现象，从而可能在判断相等与不相等的过程中产生错误，这种错误叫做 round off 错误。例如，考虑下面的两个数，两者均应等于 0。

```
a = 0;
b = sin(pi);
```

因为这两个数在理论上相等的，所以关系式 a==b 应当返回值 1。但在事实上，MATLAB 计算所产生的结果的是

```
>> a = 0;
>> b = sin(pi);
>> a == b
ans =
     0
```

MATLAB 报告了 a 和 b 不同因为他产生了一个 round off 错误，在计算中 sin(pi)产生了结果 1.2246×10^{-16} 而不是 0。两个理论上相等的值因为 round off 错误而失之发生了细微的

差别。

我们可以通过检测两数之间在一定的范围内是不是近似相等，在这个精确范围内可能会产生 round off 错误。例如测试

```
>> abs(a - b) < 1.0E-14
ans =
     1
```

将会产生正确的结果，不管在 a 与 b 的计算中产不产生的 round off 错误。

好的编程习惯

在我们检测两数值是否相等时一定要小心，因为 round off 错误可能会使两个本来应该相等的值不相等了。这时你可以在 round off 错误的范围内它是不是近似相等。

3.3.3 逻辑运算符

逻辑运算符是联系一个或二个逻辑操作数并能产生一个逻辑结果的运算符。有三个二元运算符：分别为 AND，OR 和异或运算符，还有一个一元运算符 NOT。二元逻辑运算的基本形式

$$l_1 \text{ op } l_2$$

一元逻辑运算的基本形式为

$$\text{op } l_1$$

l_1 和 l_2 代表表达式或变量，op 代表表 3.2 中的逻辑运算符。如果 l_1 和 l_2 的逻辑运算关系为 true，那么运算将会返回值 1，否则将会产生 0。

表 3.2 逻辑运算符

&	逻辑与
|	逻辑或
xor	逻辑与或
~	逻辑非

运算的结果总结在真值表 3.3 中，它向我们展示每一种运算所有可能的结果。如果一个数的值不为 0，那么 MATLAB 将把看作 true，如

表 3.3 逻辑真值表

输入		与	或	异或	非
l_1	l_2	l_1 & l_2	$l_1 \mid l_2$	$xor(l_1,l_2)$	$\sim l_1$
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

果它为 0，则其为 false。所以~5 的结果为 0，~0 的结果为 1。

标量和数组之间也可进行逻辑运算。例如，$a=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$，b=0，那么表达式 a&b 将会产生结果$\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}$。两数组之间也可进行逻辑运算，只要它们具有相同的大小。例如，$a=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$，$b=\begin{bmatrix}1 & 1\\ 0 & 0\end{bmatrix}$则 a|b 产生的结果$\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}$。如果两个数的大小不相同，那么将会产生运行时错误。

在运算的顺序中，逻辑运算在所有的数学运算和关系运算之后进行。

表达式中的运算顺序如下：

1.所有的数学运算按照前面描述的顺序的进行。
2.从左向右依次进行关系运算
3.执行所有~运算
4.从左向右依次进行&运算
5.从左向右依次进行|运算和数学运算一样，括号能够改变括号的默认顺序。

下面是关于逻辑运算的一些例子。

例 3.1

假设下面有三个变量被初始和一些表达式及其运算结果。

value1 = 1
value2 = 0
value3 = -10

逻辑表达式	结果
(a) ~value1	0
(b) value1 | value2	1
(c) value1 & value2	0
(d) value1 & value2 | value3	1
(e) value1 & (value2 | value3)	1
(f) ~(value1 & value3)	0

因为~运算在其它的逻辑运算之前进行，那么(f)中的括号是必须的。如果去掉括号的话，(f)表达式将等价于(~value1)&value3。

3.3.4 逻辑函数

MATLAB 中有大量的逻辑函数，在条件满足时，函数返回 1。条件不满足时，返回 0。这些函数和逻辑运算与关系联合在组成选择结构和循环结构。表 3.4 列出了一系列的逻辑函数。

表 3.4 **MATLAB** 逻辑函数

函数	用途
ischar(a)	a 是字符数组返回 1，否则返回 0
isempty(a)	a 是空数组返回 1，否则返回 0
isinf(a)	a 是无穷大，则返回 1，否则返回 0
isnan(a)	a 不是一个数则返 1，否则返回 0
isnumeric(a)	a 是一个数值数组返回 1，否则返回 0

测试 3.1

本测试提供了一个快速的检查方式，看你是否掌握了 3.3 的基本内容。如果你对本测试有疑问，你可以重读 3.3，问你的老师，或和同学们一起讨论。在附录 B 中可以找到本测试的答案。

变量 a，b，c，d 定义如下，计算后面的表达式。

a = 20;　　b = -2;
c = 0;　　d = 1;

1. a > b　　2. b > d;
3. a > b & c > d　　4. a == b
5. a & b > c　　6. ~~b

变量 a，b，c，d 定义如下，计算后面的表达式。

a = 2;　　$b = \begin{bmatrix} 1 & -2 \\ -0 & 10 \end{bmatrix}$

$c = \begin{bmatrix} 0 & 1 \\ 2 & 0 \end{bmatrix}$　　$d = \begin{bmatrix} -2 & 1 & 2 \\ 0 & 1 & 0 \end{bmatrix}$

7. ~(a > b);
8. a > c & b > c;
9. c <= d

变量 a，b，c，d 定义如下，计算后面的表达式。

a = 2; b = 3;
c = 10; d = 0;

10. a*b^2 > a*c
11. d | b b > a
12. (d | b) > a

变量 a，b，c，d 定义如下，计算后面的表达式。

a = 20; b = -2;
c = 0; d = 'Test';

13. isinf(a/b)
14. isinf(a/c)
15. a > b & ischar(d)
16. isempty(c)

3.4 选择结构(分支语句)

选择结构可以使 MATLAB 选择性执行指定区域内的代码(称之为语句块 blocks)，而跳过其他区域的代码。选择结构在 MATLAB 中有三种具体的形式:if 结构，switch 结构和 try/catch 结构。

3.4.1 if 结构

if 结构的基本形式如下:

```
if control_expr_1
   Statement 1
   Statement 2   } Block 1
   ...
elseif control_expr_2
   Statement 1
   Statement 2   } Block 2
   ...
else
   Statement 1
   Statement 2   } Block 3
   ...
end
```

其中 control expression 控制 if 结构的运算。如果 control_expr_1 的值非 0，那么程序将会执行语句块 1(block1)，然后跳到 end 后面的第一个可执行语句继续执行。否则，程序将会检测 control_expr_2 的值，。如果 control_expr_2 的值非 0，那么程序将会执行语句块 2(block2)，然后跳到 end 后面的第一个可执行语句继续执行。如果所有的控制表达式(control expression)均为 0，那么程序将会执行与 else 相关的语句块。

在一个 if 结构中，可以有任意个 elseif 语句，但 else 语句最多有一个。只要上面每一个控制表达式均为 0，那么下一个控制表达式将会被检测。一旦其中的一个表达式的值非 0，对应的语句块就要被执行，然后跳到 end 后面的第一个可执行语句继续执行。如果所有的控制表达式(controlexpression)均为 0，那么程序将会执行 else 语句。如果没有 else 语句，程序将会执行 end 后面的语句，而不执行 if 结构中的部分。

注意 MATLAB 在 if 结构中的关键字 end 与第二章中提到的返回已知下标最大值函数 end 完全不同。matlab 通过 end 在 M 文件中的上下文来区分开它的两个用途。在大多数情况下，控制表达式均可以联合关系运算符和逻辑运算符。正像我们在本章早些时侯学到的，当对应的条件为真时，关系运算和逻辑运算将会产生 1，否则产生 0。所以当一个运算条件为真时，运算结果为非 0，则对应的语句块，就会被执行。

例如，一元二次方程的基本形式如下:

$$ax^2 + bx + c = 0 \qquad (3.1)$$

其解为

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \qquad (3.2)$$

其中 $b^2 - 4ac$ 是我们熟知的判别式，当 $b^2 - 4ac>0$ 时，方程式有两个不同的实数根，当 $b^2 - 4ac=0$，有两个相同的实数根，当 $b^2 - 4ac<0$ 时，方程式有两个不同的复根。

假设我们检测某一元二次根的情况，并告诉使用者这个方程有两个复根，还是两个相等的实根和两个不相等的实根。用伪代码这个结构的形式如下:

```
if (b^2 - 4*a*c) < 0
    Write msg that equation has two complex roots.
elseif (b^2 - 4*a*c) ==0
    Write msg that equation has two identical real roots.
else
    Write msg that equation has two distinct real roots.
end
```

转化为 MATLAB 语言:

```
if (b^2 - 4*a*c) < 0
    disp('This equation has two complex roots.');
elseif (b^2 - 4*a*c) == 0
    disp('This equation has two identical real roots.');
else
    disp('This equation has two distinct real roots.');
end
```

回忆一下，判断为真时，关系运算符将会返回一个非 0 值，从而导致对应语句的执行。

为增加程序的可读性，在 if 结构中的语句块中最好缩进 2 到 3 个空格，而实际上没有必要。

好的编程习惯

if 结构体经常缩进 2 到 3 个空格，以增强程序的可读性。

你可以在一行内写完一个完整的 if 结构，只需把结构的每一部分后面加上分号或逗号，所以下面的两个结构是等价的:

```
if x < 0
    y = abs(x);
end
```

和

```
if x < 0; y = abs(x); end
```

但是这种方式只适用于简单的结构。

3.4.2 if 结构举例

我们将用两个例子来说明 if 结构的用途

例 3.2

求一元二次方程的根

设计并编写一个程序，用来求解一元二次方程的根。

答案:

我们将本章开头介绍的方法进行编程。

1.陈述问题这个问题的陈述非常的简单，我们要求一元二次方程的根，不管它的根是实根还是复根，有一个根还是两个根。

2.定义输入和输出

本程序的输入应为系数 a，b，c

$$ax^2 + bx + c = 0 \qquad (3.1)$$

输出量应为两个不相等的实数。两个相等的实数或两个复数。

3.写出算法本程序可分为三大块，它的函数分别为输入，运算过程和输出。

我们把每一个大块分解成更小的，更细微的工作。根据判别式的值，可能有三种计算途径，

读取输入的数据
计算出根
输入出根

所以我们要用到有三种选项的 if 结构。产生的伪代码如下

```
Prompt the user for the coefficients a, b, and c.
Read a, b, and c
discriminant ← b^2 - 4*a*c
if discriminat > 0
    x1 ← (-b + sqrt(discriminant)) / (2*a)
    x1 ← (-b - sqrt(discriminant)) / (2*a)
    Write msg that equation has two distinct real roots.
    Write out the two roots.
elseif discriminant == 0
    x1 ← -b / (2*a)
    Write msg that equation has two identical real roots.
    Write out the repeated roots.
else
    real_part ← -b / (2*a)
    imag_part ← sqrt(abs(discriminant)) / (2*a)
    Write msg that equation has two complex roots.
    Write out the two roots.
end
```

4.把算法转化为 MATLAB 语言

```
%Script file: calc_roots.m
%
% Purpose:
% This program solves for the roots of a quadratic equation
% of the form a*x^2 + b*x + c = 0. It calculates the answers
% regardless of the type of roots that the equation possesses.
%
% Record of revisions:
%       Date          Programmer        Description of change
%     ======        ============      =================
%    12/04/98       S. J. Chapman      Original code
%
% Define variables:
% a                  --Coefficient of x^2 term of equation
% b                  --Coefficient of x term of equation
% c                  --Constant term of equation
% discriminant       --Discriminant of the equation
% imag_part          --Imag part of equation (for complex roots)
% real_part          --Real part of equation (for complex roots)
% x1                 --First solution of equation (for real roots)
% x2                 --Second solution of equation (for real roots)
% Prompt the user for the coefficients of the equation
disp ('This program solves for the roots of a quadratic ');
disp ('equation of the form A*X^2 + B*X + C = 0.');
a = input('Enter the coefficient A: ');
b = input('Enter the coefficient B: ');
```

```
c = input('Enter the coefficient C: ');
% Calculate discriminant
discriminant = b^2 - 4 * a * c;
% Solve for the roots, depending on the vlaue of the discriminant.
if discriminant > 0    % there are two real roots, so ...
    x1 = (-b + sqrt(discriminant)) / (2*a);
    x2 = (-b - sqrt(discriminant)) / (2*a);
    disp('This equation has two real roots:');
    fprintf('x1 = %f\n', x1);
    fprintf('x2 = %f\n', x2);
elseif discriminant == 0 % there is one repeated root, so ...
    x1 = ( -b ) / (2*a);
    disp('This equation has two identical real roots:');
    fprintf('x1 = x2 = %f\n', x1);
else % there are complex roots, so ...
    real_part = (-b) / (2*a);
    imag_part = sqrt( abs(discriminant)) / (2*a);
    disp('This equation has complex roots:');
    fprintf('x1 = %f + i %f \n',real_part, imag_part);
    fprintf('x1 + %f - i %f \n', real_part, imag_part);
end
```

5.检测这个程序

下一步，我们必须输入实数来检测这个程序。因这个程序有三个可能的路径。所以在我们确信每一人路径都工作正常之前，必须把这三个路径检测一遍。从式子(3.2)中，我们可以有用下面的方法来验证程序的正确性。

x2 + 5x + 6 = 0	x = -2, and x = -=3
x2 + 4x + 4 = 0	x = -2
x2 + 2x + 5 = 0	x = -1 ± i2

如果输入上面三个方程的系数得到对应的结果，则说明程序是正确的。

```
>> calc_roots
This program solves for the roots of a quadratic
equation of the form A*X^2 + B*X + C = 0.
Enter the coefficient A: 1
Enter the coefficient B: 5
Enter the coefficient C: 6
This equation has two real roots:
x1 = -2.000000
x2 = -3.000000
>> calc_roots
This program solves for the roots of a quadratic
equation of the form A*X^2 + B*X + C = 0.
Enter the coefficient A: 1
Enter the coefficient B: 4
Enter the coefficient C: 4
This equation has two identical real roots:
x1 = x2 = -2.000000
>> calc_roots
This program solves for the roots of a quadratic
equation of the form A*X^2 + B*X + C = 0.
Enter the coefficient A: 1
Enter the coefficient B: 2
Enter the coefficient C: 5
This equation has complex roots:
x1 = -1.000000 + i 2.000000
x1 + -1.000000 - i 2.000000
```

在三种不同的情况下，程序都给出了正确的结果。

例 3.3

编写一个程序，求以 x，y 为自变量函数 f(x，y)的值。函数 f(x，y)的定义如下:

$$f(x,y) = \begin{cases} x+y & x\geq 0 \text{ and } y\geq 0 \\ x+y^2 & x\geq 0 \text{ and } y<0 \\ x^2+y & x<0 \text{ and } y\geq 0 \\ x^2+y^2 & x<0 \text{ and } y<0 \end{cases}$$

答案:

根据自变量 x 和 y 的正负符号的不同，而采取不同的求值表达式。为选取合适的表达式，检查用户输入的 x，y 的正负符号是必要的。

1.陈述问题

这个问题的陈述非常简单:根据用户输入的 x，y，求函数 $f(x，y)$的值。

2.确定输入输出量

程序的输入量为函数的自变量 x，y。输出量为函数值 $f(x，y)$。

3.设计算法这个问题可以把他分解成三个大块，即输入，计算过程，和输出。

我们把这三大块再分解成小的，精细的工作。在计算 f(x，y)时，我们有 4 种选择，选哪一种取决于 x，y 的值。所以及逻辑上我们要用 4 个选择的 if 结构来实现。产生的伪代码如下:

```
Prompt the user for the values x and y
Read x and y
if x≥0 and y≥0
    fun ← x + y
elseif x≥0 and y<0
    fun ← x + y^2
elseif x<0 and y≥0
    fun ← x^2 + y
else
    fun ← x^2 + y^2
end
Write out f(x,y)
```

4.转化为 MATLAB 语句

最终的代码如下

```
% Scripte file: funxy.m
%
% Purpose:
% This program solves the function f(x,y) for a
% user-specified x and y, where f(x,y) is defined as:
%                          _
%                         |
%                         | x + y                    x>=0 and y>=0
%                         | x + y^2                  x>=0 and y<0
%                f(x,y)=  | x^2 + y                  x<0 and y>=0
%                         | x^2 + y^2                x<0 and y<0
%                         |_
%
% Record of revisions:
%      Date               Programmer          Description of change
%    =====              ==========          =================
%   12/05/98            S.J.Chapman         Original code
%
% Define variables:
% x                      --First independent variable
```

```
% y                       --Second independent variable
% fun                     --Resulting function
% Prompt the user for the values x and y
x = input('Enter the x coefficient: ');
y = input('Enter the y coefficient: ');
% Calculate the function f(x,y) based upon
% the signs of x and y.
if x>=0 & y>=0
    fun = x + y;
elseif x>=0 & y<0
    fun = x + y^2;
elseif x<0 & y>=0
    fun = x^2 + y;
else
    fun = x^2 + y^2;
end
% Write the value of the function.
disp(['The vlaue of the function is ' num2str(fun)]);
```

5.检测程序

下一步，我们必须输入实数来检测这个程序。因这个程序有四个可能的路径。所以在我们确信每一人路径都工作正常之前，必须把这四个路径检测一遍。我们分别取 4 个象限内的值(2，3)，(-2,3)，(2，-3)和(-2，-3)。我们用手工计算可得

$f(2,3) = 2 + 3 = 5$

$f(2,-3) = 2 + (-3)^2 = 11$

$f(-2,3) = (-2)^2 + 3 = 7$

$f(-2, -3) = (-2)^2 + (-3)^2 = 13$

当程序被编程后，运行 4 次并输入相应的值，运算结果如下:

```
>> funxy
Enter the x coefficient: 2
Enter the y coefficient: 3
The vlaue of the function is 5
>> funxy
Enter the x coefficient: 2
Enter the y coefficient: -3
The vlaue of the function is 11
>> funxy
Enter the x coefficient: -2
Enter the y coefficient: 3
The vlaue of the function is 7
>> funxy
Enter the x coefficient: -2
Enter the y coefficient: -3
The vlaue of the function is 13
```

程序在 4 种可能的情况下均产生了正确的结果。

3.4.3 关于 if 结构使用的注意事项

if 结构是非常灵活的，它必须含有一个 if 语句和一个 end 语句。中间可以有任意个 elseif 语句，也可以有一个 else 语句。联合它的这些特性，我们可以创建出我们需要的各种各样的选择结构。

还有 if 语句是可以嵌套的。如果 if 结构完全是另一个 if 结构的一个语句块，我们就称两者为嵌套关系。下面是两个 if 语句的嵌套。

```
if x > 0
    ...
    if y < 0
        ...
```

```
        end
        ...
    end
```

MATLAB 翻译器经常把把已知的 end 语句和它最近的 if 语句联合在一起，所以第一个 end 语句和 if y<0 最靠近，而第二个 end 与 if x>0 最接近。对于一个编写正确的程序，它能工作正常。但如果程序员编写出错误，它将会使编译器出现混淆性错误信息提示。例如，假设我们编写一个大的程序，包括如下的一个结构:

```
...
if (test1)
    ...
    if (test2)
        ...
        if (test3)
            ...
        end
        ...
    end
    ...
end
```

这个程序包括了三个嵌套的 if 结构，在这个结构中可能有上千行的代码。现在假设第一个 end 在编辑区域突然被删除，那么 MATLAB 编译器将会自动将第二个 end 与最里面的 if (test3)结构联合起来，第三个 end 将会和中间的 if(test2)联合起来。当编译器翻译到达文件结束的时候，那将发现第一个 if(test1)结构将永远没有结束，然后编译器就会产生一个错误提示信息，即缺少一个 end。但是，它不能告诉你问题发生在什么地方，这就使我们必须回过头去看整个程序，来找问题。

在大多数情况下，执行一个算法，即可以用多个 else if 语句，也可以用 if 语句的嵌套。在这种情况下，程序员可以选择他喜欢的方式。

例 3.4

给出等级分数

假设我们要编写一个程序，输入一个数值分数，输出等级分数，即是 A 级，B 级和 C 级

```
   grade > 95      A
95 ≥ grade > 86    B
86 ≥ grade > 76    C
76 ≥ grade > 66    D
66 ≥ grade >  0    F
```

用两种方式写出这个程序，第一种方式用多个 elseif 语句，第二种方式用 if 的嵌套。

答案:

(a)用多个 elseif 语句

```
if grade > 95.0
    disp('The grade is A.');
elseif grade > 86.0
    disp('The grade is B.');
elseif grade >76.0
    disp('The grade is C.');
elseif grade > 66.0
    disp('The grade is D.');
else
    disp('The grade is F.');
end
```

(b)用 if 嵌套结构

```
if grade > 95.0
  disp('The grade is A.');
```

```
else
    if grade > 86.0
        disp('The grade is B.');
    else
        if grade > 76.0
            disp('The grade is C.');
        else
            if grade > 66.0
                disp('The grade is D.');
            else
                disp('The grade is F.');
            end
        end
    end
end
```

从上面的例子中，我们可以看到如果有多个选项的话，在一个 if 结构中用到多个 else if 语句将会比 if 的嵌套结构简单的多。

好的编程习惯

对于有许多选项的选择结构来说，最好在一个 if 结构中使用多个 elseif 语句，尽量不用 if 的嵌套结构。

3.4.4 switch 结构

switch 结构是另一种形式的选择结构。程序员可以根据一个单精度整形数，字符或逻辑表达式的值来选择执行特定的代码语句块。

```
switch (switch_expr)
case case_expr_1,
    Statement 1
    Statement 2   } Block 1
    ...
case case_expr_2
    Statement 1
    Statement 2   } Block 2
    ...
...
otherwise,
    Statement 1
    Statement 2   } Block n
    ...
end
```

如果 switch_expr 的值与 case_expr_1 相符，那么第一个代码块将会被执行，然后程序将会跳到 switch 结构后的第一个语句。如果 switch_expr 的值与 case_expr_2 相符，那么第二个代码块将会被执行，然后程序将会跳到 switch 结构后的第一个语句。在这个结构中，用相同的方法来对待其他的情况。otherwise 语句块是可选的。如果它存在的话，当 switch_expr 的值与其他所有的选项都不相符时，这个语句块将会被执行。如果它不存在，且 witch_expr 的值与其他所有的选项都不相符，那么这个结构中的任何一个语句块都不会被执行。这种情况下的结果可以看作没有选择结构，直接执行 MATLAB 语言。

如果说 switch_expr 有很多值可以导致相同代码的执行，那么这些值可以括在同一括号内，如下所示。如果这个 switch 表达式和表中任何一个表达式相匹配，那么这个语句块将会被执行。

```
switch (switch_expr)
case {case_expr_1, case_expr_2, case_expr_3},
```

```
    Statement 1
    Statement 2   Block 1
    ...
otherwise,
    Statement 1
    Statement 2   Block n
    ...
end
```

switch_expr 和每一个 case_expr 既可以是数值，也可以是字符值。

注意在大多情况下只有一个语句块会被执行。当一个语句块被执行后，编译器就会跳到 end 语句后的第一个语句开始执行。如果 switch 表达和多个 case 表达式相对应，那么只有他们中的第一个将会被执行。

让我们看一个简单的关于 switch 结构的例子。下面的语句用来判断 1 到 10 之间的数是奇数还是偶数。它用来说明一系列的 case 选项值的应用和 otherwise 语块的应用。

```
switch (value)
case {1, 3, 5, 7, 9},
    disp('The value is odd.');
case {2, 4, 6, 8, 10},
    disp('The value is even.');
otherwise,
    disp('The value is out of range.');
end
```

3.4.5 try/catch 结构的应用

try/catch 结构是选择结构的一种特殊形式，用于捕捉错误。一般地，当一个 MATLAB 程序在运行时遇到了一个错误，这个程序就会中止执行。try/catch 结构修改了这个默认行为。

如果一个错误发生在这个结构的 try 语句块中，那么程序将会执行 catch 语句块，程序将不会中断。它将帮助程序员控制程序中的错误，而不用使程序中断。

Try/catch 结构的基本形式如下:

```
try
    Statement 1
    Statement 2   Try Block
    ...
catch
    Statement 1
    Statement 2   Catch Block
    ...
end
```

当程序运行到 try/catch 语句块，在 try 语句块中的一些语句将会被执行。如果没有错误出现，catch 语句块将会被跳过。另一方面，如果错误发生在一个 try 语句块，那么程序将中止执行 try 语句块，并立即执行 catch 语句块。

下面有一个包含 try/catch 结构程序。它能创建一个数组，并询问用户显示数组中的哪能一个元素。用户提供一个下标，那么这个程序将会显示对应的数组元素 try 语句块一般会在这个程序中执行，只有当 try 语句块执行出错，catch 语句块将会发生错误。

```
% Initialize array
a = [ 1 -3 2 5];
try
    % Try to display an element
    index = input('Enter subscript of element to display: ');
    disp(['a(' int2str(index) ') = ' num2str(a(index))] );
```

```
catch
    % If we get here an error occurred
    disp( ['Illegal subscript: ' int2str(index)] );
end
```

这个程序的执行结果如下：

```
>> try_catch
Enter subscript of element to display: 3
a(3) = 2
>> try_catch
Enter subscript of element to display: 8
Illegal subscript: 8
```

测试 3.2

本测试提供了一个快速的检查方式，看你是否掌握了 3.4 的基本内容。如果你对本测试有疑问，你可以重读 3.4，问你的老师，或和同学们一起讨论。在附录 B 中可以找到本测试的答案。

根据下面的描述编写对应的 MATLAB 语句。

1. 如果 x 大于等于 0，把 x 的平方根赋值于变量 sqrt_x，并打印出结果。否则，打印出一条关于平方根函数参数的错误信息。并把 sqrt_x 归零。

2. 变量 fun 由 n/m 计算得到，如果 m 的绝对值小于 $1.0e^{-300}$，打印出除数为 0，否则计算并打印出 fun 值。

3. 租用一个交通工具前 100 公里 0.50 美元每公里，在下面的 200 公里中 2.30 美元每分钟，越过 300 公里的部分一律按 0.20 美元每公里。已知公里数，编写对应的 MATLAB 语句计算出总费用和平均每公里的费用。

检测下面的 matlab 语句，是对是错?正确的，输出结果如何，错误的，错在哪里?

4.
```
if volts > 125
    disp('WARNING: High voltage on line.');
    if volts < 105
        disp('WARNING: Low voltage on line.');
    else
        disp('Line voltage is within tolerances.');
    end
```

5.
```
color = 'yellow';
switch( color);
case 'red',
    disp('Stop now!');
case 'yellow',
    disp('Prepare to stop.');
case 'green',
    disp('Proceed through intersection.');
otherwise,
    disp('Illegal color encountered.');
end
```

6.
```
if temperature > 37
    disp('Human body temperature exceeded.');
elseif temperature > 100
    disp('Boiling point of water exceeded.');
end
```

3.5 附加的画图特性

在本节中，我们将讨论简单的二维图象(在第二章我们已有所介绍)的附加特性。这些特性将允许我们控制 x，y 轴上的值的范围，在一个坐标系内打印多个图象，或创建多个

图，或在一个图象窗口内创建多个子图像，或提供更加强大的轨迹文本字符控制。还有，我们将向大家如何创建极坐标。

3.5.1 控制 x，y 轴绘图的上下限

在默认的情况下，图象的 X，Y 轴的范围宽到能显示输入值的每一个点。但是有时只显示这些数据的一部分非常有用，这时你可以应用 axis 命令/函数。

axis 命令/函数的一些形式展示在表 3.5 中。其中两个最重要的形式在表中用黑体字标出——它允许程序员设定和修改坐标的上下限。所有形式的完全列表将会在 MATLAB 的在线文件中找到。

为了说明 axis 的应用，我们将画出函数 f(x)=sinx 从-2π 到 2π 之间的图象，然后限定坐标的区域为 0≤x≤π，0≤y≤1。

表 3.5axis 函数/命令的形式

命令	功能
v=axis	此函数将会返回一个 4 元素行向量[xmin xmax ymin ymax]，其中 xmin xmax ymin ymax 代表 x，y 轴的上下限
axis([xmin xmax ymin ymax])	**xmin xmax** 设定横轴的下限及上限， **ymin ymax** 设定纵轴的下限及上限
axis equal	将横轴纵轴的尺度比例设成相同值
axis square	横轴及纵轴比例是 1:1
axis normal	以预设值画纵轴及横轴
axis off	将纵轴及横轴取消
axis on	这个命令打开所有的轴标签，核对符号，背景(默认情形)

一些 MATLAB 命令似乎不能确定它是个函数还是一个命令。例如，有时 axis 它好像是命令，有时它好像是函数。有时我们把它当作命令:axis on，在其他时候，我们把他当作函数:axis([0 20 0 35])。遇到这样的情况怎么办?

一个简单的答案是 MATLAB 命令是通过函数来实现的。MALTAB 编译器无论什么时候遇到这个命令，它都能转化为相应的函数。它把命令直接当作函数来用，而不是应用命令语法。下面的两个语句是等价的：

```
axis on;
axis ('on');
```

无论什么时候 MATLAB 遇到一个命令时，它都会转化一个函数，当命令的参数当作字符串看作相对应函数的参数。所以编译器翻译如下命令:

```
garbage 1 2 3
```

为

```
garbage ('1', '2', '3')
```

注意只有带有字符参数的函数才能当作命令。带有数字参数的函数只能被当作函数。这就是为什么 axis 有时当作命令，有时被当作函数。

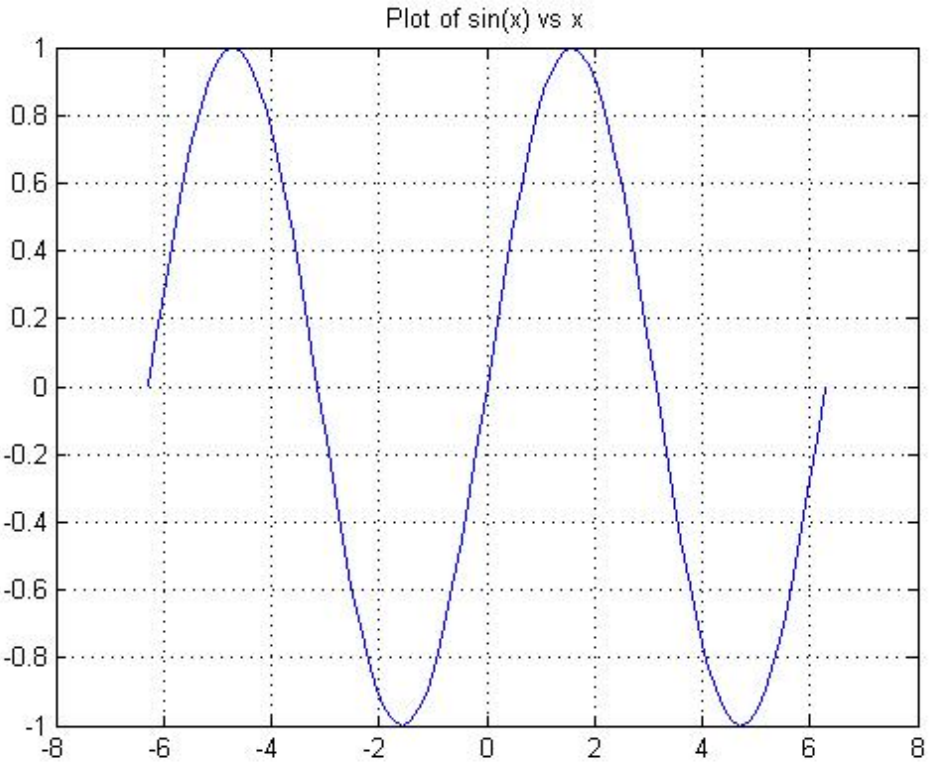
```
x=-2*pi:pi/20:2*pi;
y=sin(x);
plot(x,y);
title('Plot of sin(x) vs x');
```

当前图象坐标轴的上下限的大小由函数 axis 得到。

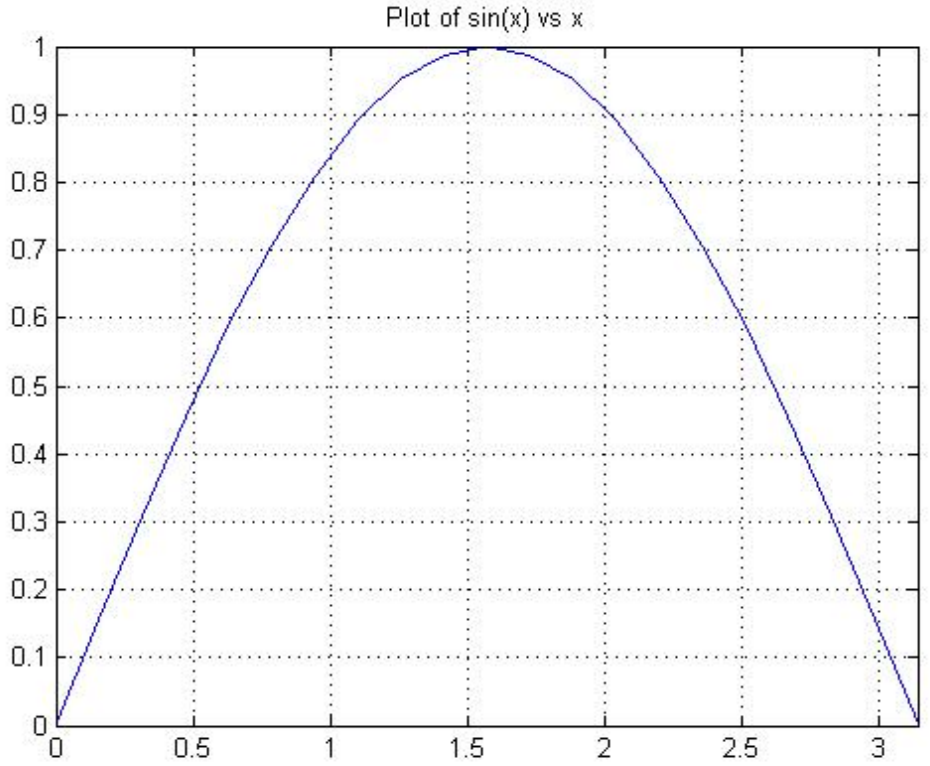
```
>> limits=axis
limits =
     -8      8     -1      1
```

修改坐标轴的上下限可以调用函数 axis([0 pi 0 1])。

当这个函数执行后，产生的图象如图 3.3（b）所示。



(a)



(b)

图 3.3 (a)以 x 为自变量的 sinx 的图象 （b）画图区域为［0 π 0 1］

3.5.2 在同一坐标系内画出多个图象

在一般情况下，创建一个新的图象就要用到一个 plot 命令，前面的数据就会自动消失。这种行为可以通过使用 hold 命令得到修改。当 hold on 命令执行后，所有的新的图象都会叠加在原来存在的图象。hold off 命令可恢复默认情况，用新的图象来替代原来的图象。

例如，在同一坐标轴内的画出 sinx 和 cosx 的图象。产生的图象如图 3.4 所示。

```
x=-pi:pi/20:pi;
y1=sin(x);
y2=cos(x);
plot(x,y1,'b-');
hold on;
plot(x,y2,'k--');
hold off;
legend ('sin x','cos x');
```

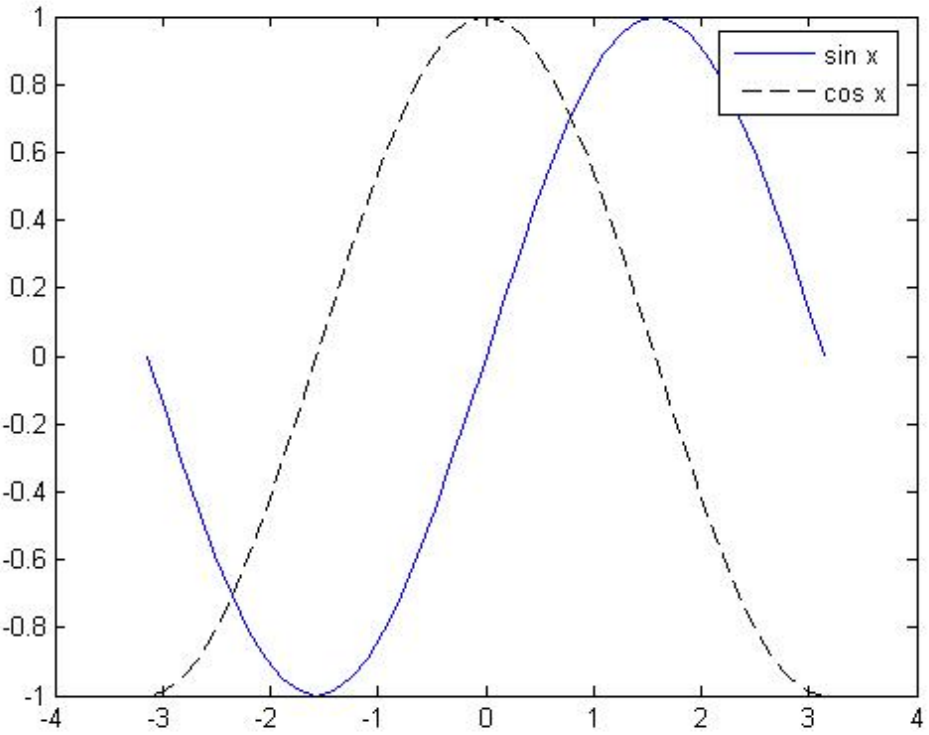


图 3．4 用 hold 命令在一个坐标轴内画出两个函数的图象

3.5.3 创建多个图象

MATLAB 可以创建多个图象窗口，每个窗口都有不同的数据。我们用**图象数**来区分这些图象窗口，**图象数**是一个小的正整数。第一个图象窗口称为图 1，第二个图象窗口为图 2，依次类推。这些窗口中的一个称为当前图象窗口，所有的新的画图命令将会展示在那个窗口中。

我们用 figure 函数来选择当前窗口。这个函数的形式为“figure(n)”，其中 n 代表图象数。当这个函数被执行后，图 n 将会变为当前图象，执行所有的画图命令。如果这个图象窗口不存在，那么 matlab 将会自动创建。当前图象也可以用鼠标单击选择。

gcf 函数用于当于返回当前图象数。当你需要知道当前图象数时，你就把这个函数写入 M 文件中。

下面的命令用于说明图函数的应用。它将创建两个图象，第一个用来展示 e^x 的图象，第二个用来展示 e^{-x} 的图象。

```
figure(1);
x=x:0.05:2;
y1=exp(x);
plot(x,y1);
figure(2);
y2=exp(-x);
plot(x,y2);
```

3.5.4 子图象

在一个图象窗口中有一系列的坐标系，创建出多个子图象。创建子图象要用到 subplot 命令其形式如下

```
subplot(m,n,p)
```

这个命令在当前图象窗口创建了 m×n 个子图象，按 m 行，n 列排列，并选择子图象 p 来接受当前所有画图命令。

这些子图象以从左向右从上到下编号。例如，命令 subplot(2,3,4)将会创建 6 个子图象，而且 subplot 4 是当前子图象。

如果 subplot 命令创建的新坐标系与原来的坐标系相冲突，那么原来的坐标系将会被自动删除。

下面的命令将会在同一窗口中创建两个子图象，每一个子图象独立地展示不同的图象。

产生的图象为图 3.5。

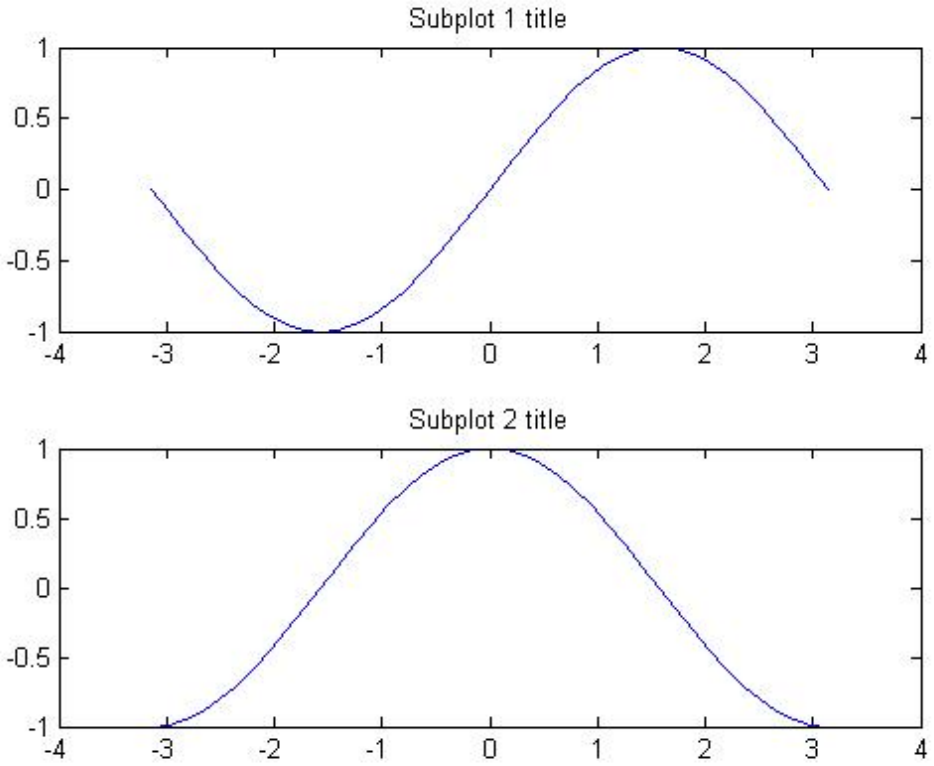


图 3.5 包含多个子图象的图象

```
figure(1);
subplot(2,1,1);
```

```
x=-pi:pi/20:pi;
y=sin(x);
plot(x,y);
title('Subplot 1 title');
subplot(2,1,2);
x=-pi:pi/20:pi;
y=cos(x);
plot(x,y);
title('Subplot 2 title');
```

3.5.5 对画线的增强控制

在第二章中，我们学习了如何设置画线的颜色，样式，符号形式。我们还可以设置其中的 4 种附加的属性。

属性	说明
. LineWidth	用来指定线的宽度
. MarkerEdgeColor	用来指定标识表面的颜色
. MarkerFaceColor	填充标识的颜色
. MarkerSize	指定标识的大小

在 plot 命令中，在自变量和函数之后被指定，形式如下：

```
plot(x,y,'PropertyName',value,...)
```

例如，下面的命令将画出一个图象，轨迹的宽度为 3，颜色为黑色，圆圈标识的宽度为 6，每个标识为红色边缘和绿色内核，如图 3.6。

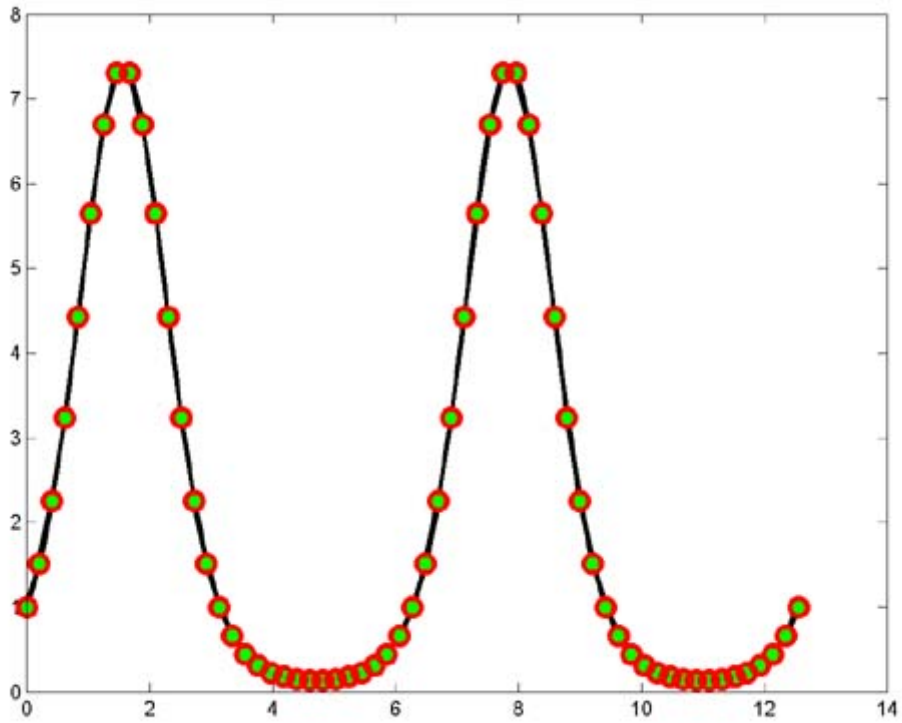


图 3.6 这个图象用于说明 LineWidth 和 Marker 的属性

3.5.6 文本字符串的高级控制

我们在画图中可能要用到文本字符串(比如标题，坐标轴标签)，这些字符串我们可以

用黑体，斜体来格式化，也包括特殊的希腊或数学符号。

文本的字体通可以通过 stream modifiers 修改。一个 stream modifier 是一个特殊的字符序列，

用来告诉编译器改变它的行为。最普通的 stream modifiers 是:

. \bf	黑体
. \it	斜体
. \rm	恢复正常字体
. \fontname	字体的名字
. \fontsize	字体的大小
. _{xxx}	xxx 做为某字符的上标
. ^{xxx}	xxx 做为某字符的下标

一旦一个 stream modifier 插入一个文本字符串中，它持续发挥作用，直到这个字符串的结束或消失。如果一个 modifier 后在跟着一个{}，只有{}中的文本起作用。

特殊的希腊字母或数学符号也可用在文本字符串中。通过嵌入特殊的转义序列来创建这些字符。这些转义序列是支持 T_EX 语言的特殊序列的一个子集。在表 3.6 中向大家展示一些转义序列代码的例子。所有转义序列可以在 matlab 在线帮助文本中找到。

如果要打印转义符\，{，}，_，或^就必须在前面加上一个反斜杠。

下面的例子用于说明 stream modifier 和特殊字符的应用。

字符串	结果
\tau_{ind} versus \omega_{\itm} \theta varies from 0\circ to 90\circ \bf{B}_{\itS}	τ_{ind} versus ω_m θ varies from 0^o to 90^o $\mathbf{B}_S$

表 3.6 精选的希腊符号和数学符号

字符序列	符号	字符序列	符号	字符序列	符号
\alpha	α			\int	$\int$
\beta	β			\cong	$\cong$
\gamma	γ	\Gamma	Γ	\sim	$\sim$
\delta	δ	\Delta	Δ	\infty	∞
\epsilon	ε			\pm	$\pm$
\eta	η			\leq	$\leq$
\theta	θ			\geq	$\geq$
\lamda	λ	\Lamda	Λ	\neq	$\neq$
\mu	μ			\propto	$\propto$
\nu	ν			\div	$\div$
\pi	π	\Pi	Π	\circ	$\circ$
\phi	φ			\leftrightarrow	$\leftrightarrow$
\rho	ρ			\leftarrow	$\leftarrow$
\sigma	σ	\Sigma	Σ	\rightarrow	$\rightarrow$
\tau	τ			\uparrow	$\uparrow$
\omega	ω	\Omega	Ω	\downarrow	$\downarrow$

3.5.7 极坐标图象

Matlab 中包括一个重要的函数叫做 polar，它用于在极坐标系中画图。这个函数的基本形式如下:

```
polar(theta,r)
```

其是 theta 代表一个弧度角数组，r 代表一个距离数组。它用来画以角度为自变量的函数的极坐标图是非常有用的。

例 3.5

心形麦克风

为舞台表演设计的麦克风大多都是定向麦克风，它能够增大来自演唱者的信号，抑制后面观众的噪声信号。一个心形麦克风的增益 gain 是关于角度 θ 的函数，关系式如下

$$Gain=2g(1+\cos\theta) \qquad (3.3)$$

其中 g 是和特定的心形麦克风有关的常量。θ 是声源和麦克风之间的夹角。假设一个麦克风的 g 是 0.5，画出函数 $Gain$ 的极坐标图。

答案：我们必须计算出与角度对应的函数值，然后画出相应的极坐标图。产生的结果如图 3.7 所示。注意这种麦克风叫做心形麦克风，所以得出来曲线的形状像颗心。

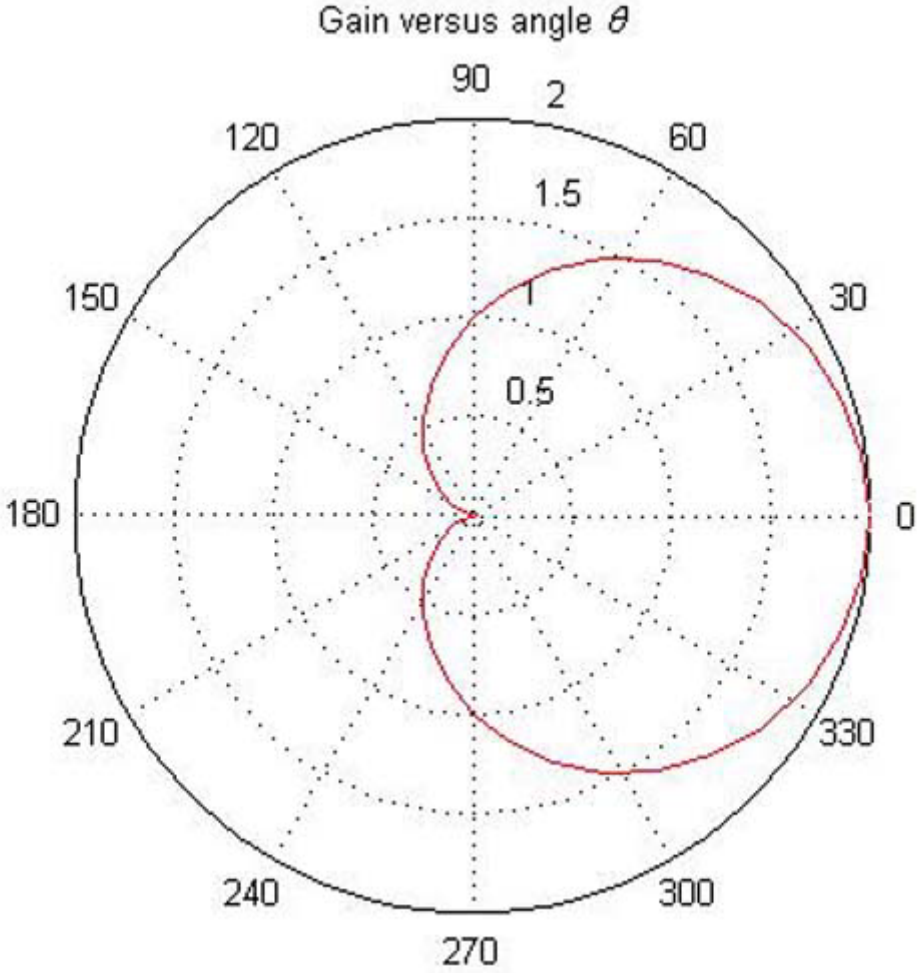


图 3.7 心形麦克风的增益图象

代码如下：

```
% Script file: microphone.m
%
% Purpose:
% This program plots the gain pattern of a cardioid
% microphone.
%
% Record of revisions:
% Date Programmer Description of change
% ==== ========== =====================
% 12/10/97 S. J. Chapman Original code
%
```

```
% Define variables:
% g -- Microphone gain constant
% gain -- Gain as a function of angle
% theta -- Angle from microphone axis (radians)
% Calculate gain versus angle
g = 0.5;
theta = 0:pi/20:2*pi;
gain = 2*g*(1+cos(theta));
% Plot gain
polar (theta,gain,'r-');
title ('Gain versus angle \it\theta');
```

例 3.6

电器工程低通滤波电路

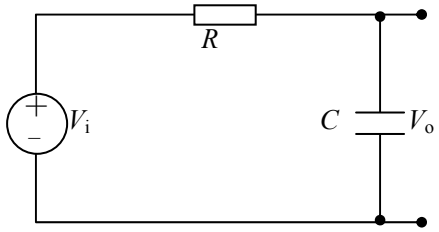


图 3.8 简单的低通滤波电路

上图是向大家展示的一个简单的低通滤波电路。这个电路是由一个电阻和一个电容组成。输出电压 V_0 与输入电压 V_i 的电压比为

$$\frac{V_o}{V_i} = \frac{1}{1+j2\pi fRC} \qquad (3.4)$$

其中 V_i 是在频率 f 下的正弦输入电压。R 代表电阻，单位为欧姆。C 代表电容，单位为法拉。j 为$\sqrt{-1}$

假设 R=16 kΩ，电容 C=1 μF，画出这个滤波器，振幅与频率的关系图。由于频率和振幅的关系图两者的跨度都非常的大，按照惯例，两者均使用对数标度，另外相位的取值范围非常的小，所以对相位我们应用线性标度。

所以，我们将用 loglog 命令来画频率响应，用 semilogx 来画相位响应图。我们将在一个画图窗口内画出两个子图象。

代码如下：

```
% Script file: plot_filter.m
%
% Purpose:
% This program plots the amplitude and phase responses
% of a low-padd RC filter.
%
% Record of revisions:
% Date Programmer Description of change
% ==== ========== ====================
% 12/29/98 S. J. Chapman Original code
%
% Define variables:
% amp       -- Amplitude response
% C         -- Capacitiance (farads)
% f         -- Frequency of input signal (Hz)
% phase     -- Phase response
% R         -- Resistance (ohms)
% res       -- Vo/Vi
% Initialize R & C
```

```
R = 16000; % 16 k ohms
C = 1.0E-6; % 1 uF
% Create array of input frequencies
f = 1:2:1000;
% Calculate response
res = 1 ./ ( 1 + j*2*pi*f*R*C );
% Calculate amplitude response
amp = abs(res);
% Calculate phase response
phase = angle(res);
% Create plots
subplot(2,1,1);
loglog( f, amp );
title('Amplitude Response');
xlabel('Frequency (Hz)');
ylabel('Output/Input Ratio');
grid on;
subplot(2,1,2);
semilogx( f, phase );
title('Phase Response');
xlabel('Frequency (Hz)');
ylabel('Output-Input Phase (rad)');
grid on;
```

得到的结果如图 3.9 所示。注意这个电路叫做低通滤波电路，是因为在低频下，电压很少衰减，在高频下，电压衰减的很厉害。

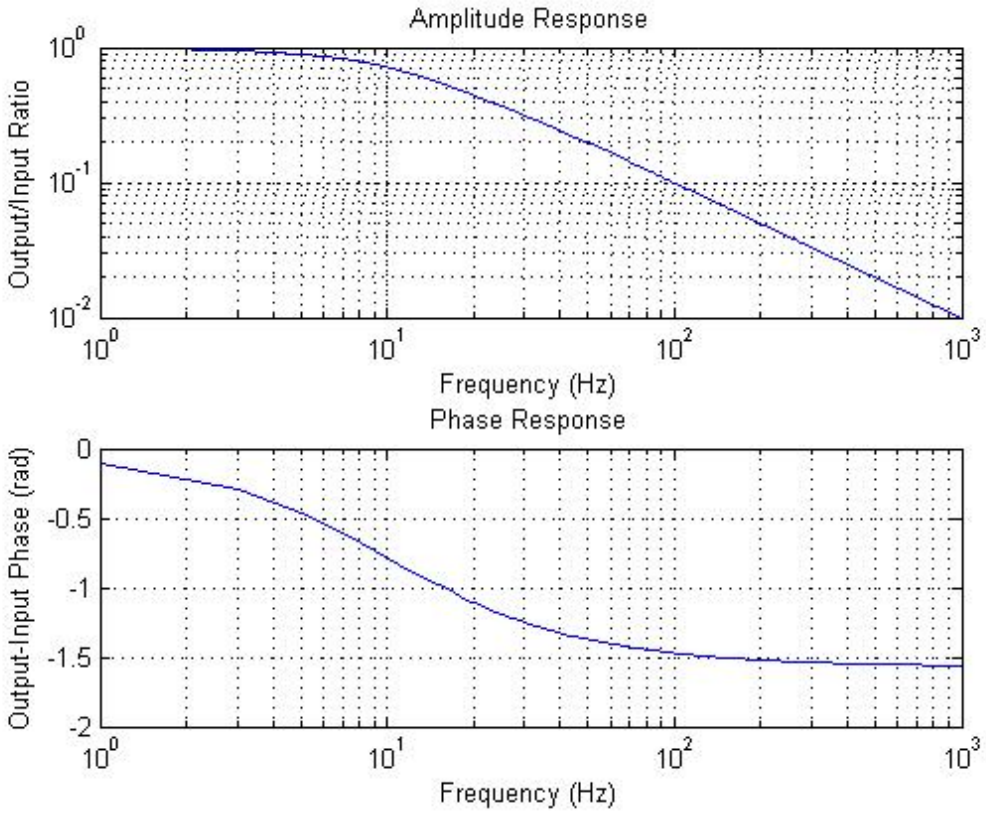


图 3.9

例 3.7

热力学：理想气体定律

理想气体是指发生在分子之间的碰撞均为弹性碰撞。你可以把理想气体中的每一个分

子想象成一个刚性小弹，每次碰撞，总动能不会改变。这样的气体可以用三个变量来描述：绝对气压（*P*），体积（*V*）和绝对温度（*T*）。三者之间的关系式就是我们所熟知的理想气体定律。

$$PV=nRT \qquad (3.5)$$

P 代表气压，单位为千帕，*V* 代表气体的体积，单位为升，*n* 代表分子的摩尔数，*T* 代表绝对温度，单位为开。

假设一理想气体样品在 273K 温度下，有一摩尔分子，回答相关问题。

（a）当气压从 1 到 1000 千帕变化，气体的体积将会如何变化？设置合适的坐标，画出这个气体的压力——体积图象。

（b）假设这个气体的温度上升到 373K，气体体积将会随气压如何变化。在与（a）相同的坐标系内，画出气体的压力——体积图象。轨迹用虚绿线，宽度为 2pixel。在图象上包含有一个大标题，x，y 轴的标签，还有各轨迹的图例。

答案：因为我们画的值都有一千个因子，所以一个普通线性尺度坐标不能画出有效的图象。所以，我们在画图时，用 log-log 标度。注意我们必须在相同的坐标系下，画出两个曲线，所以我们必须在画完第一个图象后加入 hold on 命令，当所有画图结束后，用上 hold off 命令。我们也必须指定轨迹的颜色，样式和宽度，并指定标签为黑体。

下面的程序创建了气压的函数 *V*(气体的体积)的图象。注意那些控制图象样式的语句，我们已用黑体标出。

```
% Script file: ideal_gas.m
%
% Purpose:
%    This program plots the pressure versus volumn of an
%    ideal gas.
%
% Record of revisions:
%    Date            Programmer          Description of change
%    ====            ==========          =====================
%    07/17/00      S.J.Chapman         Original code
%
% Define variables:
% n               --Number of atoms (mol)
% P               --Pressure (kPa)
% R               --Ideal gas constant (L kPa/mol K)
% T               --Temperature (K)
% V               --volume (L)
% Initialize nRT
n = 1;                % Moles of atoms
R = 9.314;            % Ideal gas constant
T = 273;              % Temperature (K)
% Create array of input pressures. Note that this
% array must be guite dense to catch the major
% changes in volume at low pressures.
P = 1:0.1:1000;
% Calculate volumes
V = (n * R * T) ./ P;
% Create first plot.
figure(1);
loglog(P, V, 'r-', 'LineWidth', 2);
title('\bfVolume vs Pressure in an Ideal Gas');
xlabel('\bfPressure (kPa)');
ylabel('bfVolume (L)');
grid on;
hold on;
% Now increase temperature
T = 373;              % Temperature (K)
```

```
% Calculate volumes
V = (n * R * T) ./ P;
% Add second line to plot
figure(1);
loglog(P, V, 'b--', 'LineWidth', 2);
hold off;
% Add legend
legend('T = 273 K', 'T = 373 K');
```

产生的 V-P 图象如图 3.10 所示。

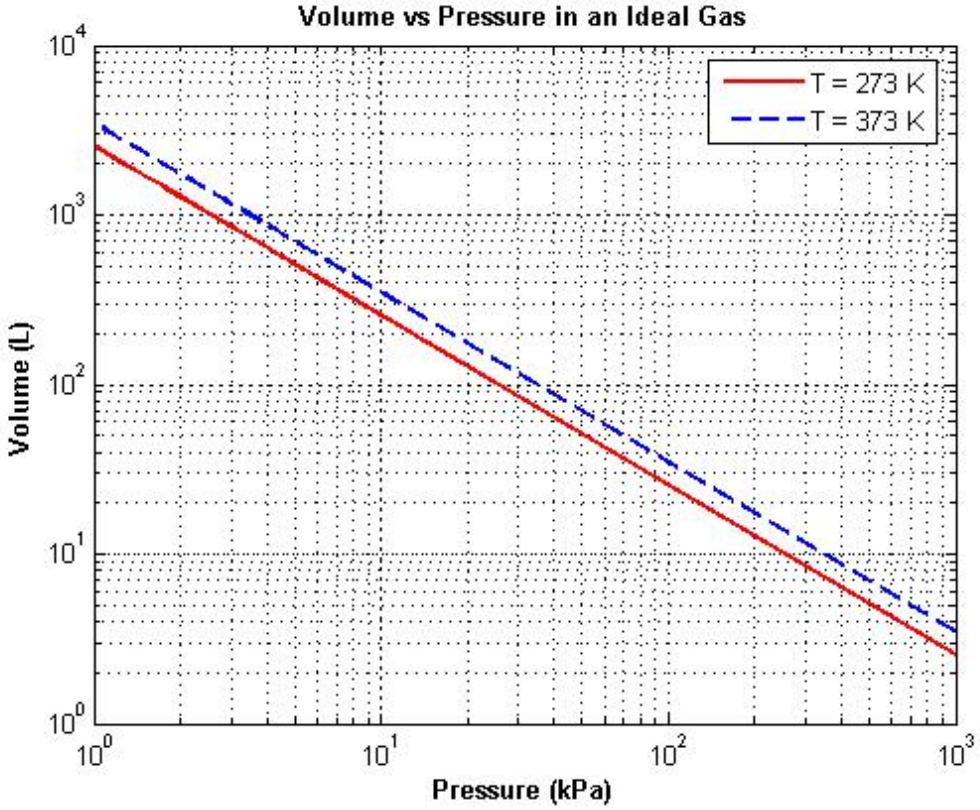


图 3.10 理想气体的 V-P 图象

3.5.8 注释并保存图象

一旦 matlab 成功创建一个图象，那么用户就可以运画图工具条上的 GUI 工具来编辑和注释这些图象。图 3.11 向大家展示了这些可用的工具，它允许我们添加直线，带箭头的线，还有文本。当工具条中的编辑按钮(↖)被选中，注释和翻译工具将会变得可用。

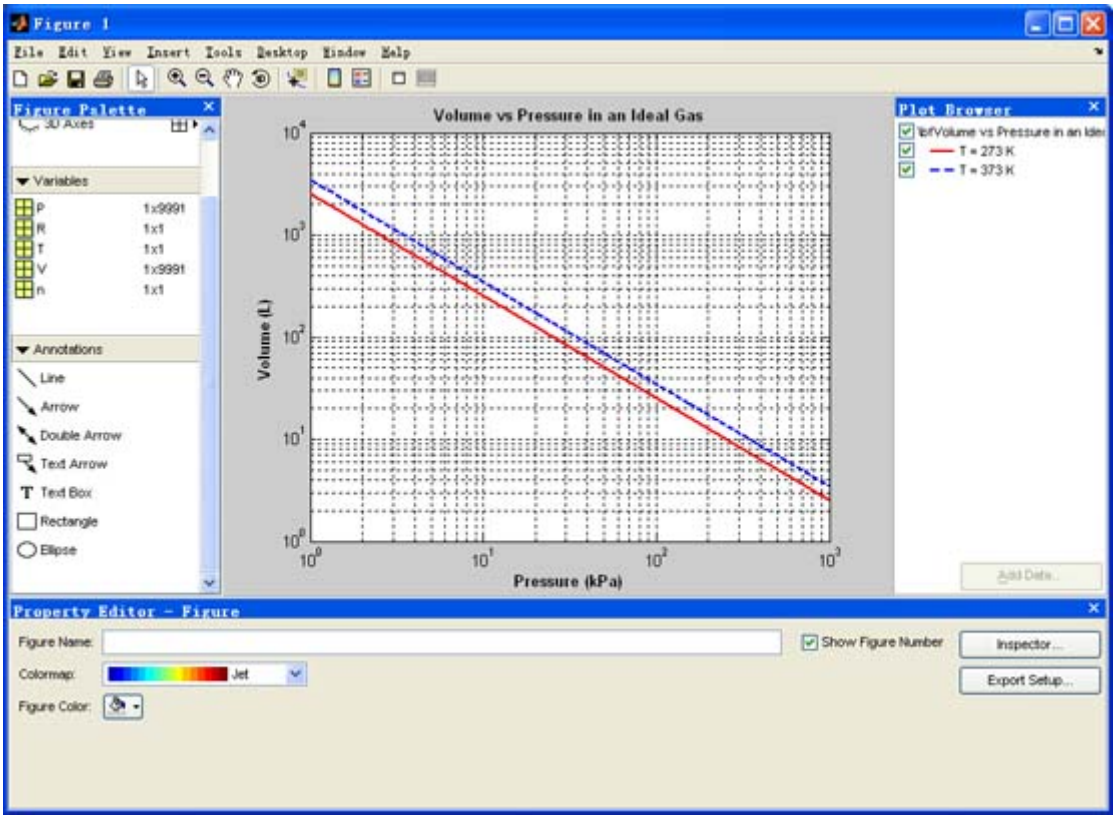


图 3.11 图工具条中的编辑工具

还有，当编辑按钮被按下，单击图象中的任何一条线或一个文本，它们将会处于可编辑状态，如果双击它们将会弹出一个属性窗口，允许我们修改这个对象的每一项属性。图 3.12 是图 3.10 经用户编辑修改后得到的，用户把绿线改成了 3pixel 宽的虚线，并加上了箭头和注释。

当这个图象的编辑和注释完成后，你可以一种可修改的格式存储整个图象，方法选择图象窗口中的”file/save as”菜单项。产生的图象文件(*.fig)包含了用于重建这个图象的所有信息，也就是说在未来的任何时侯你都可以轻松的重建这个图象。

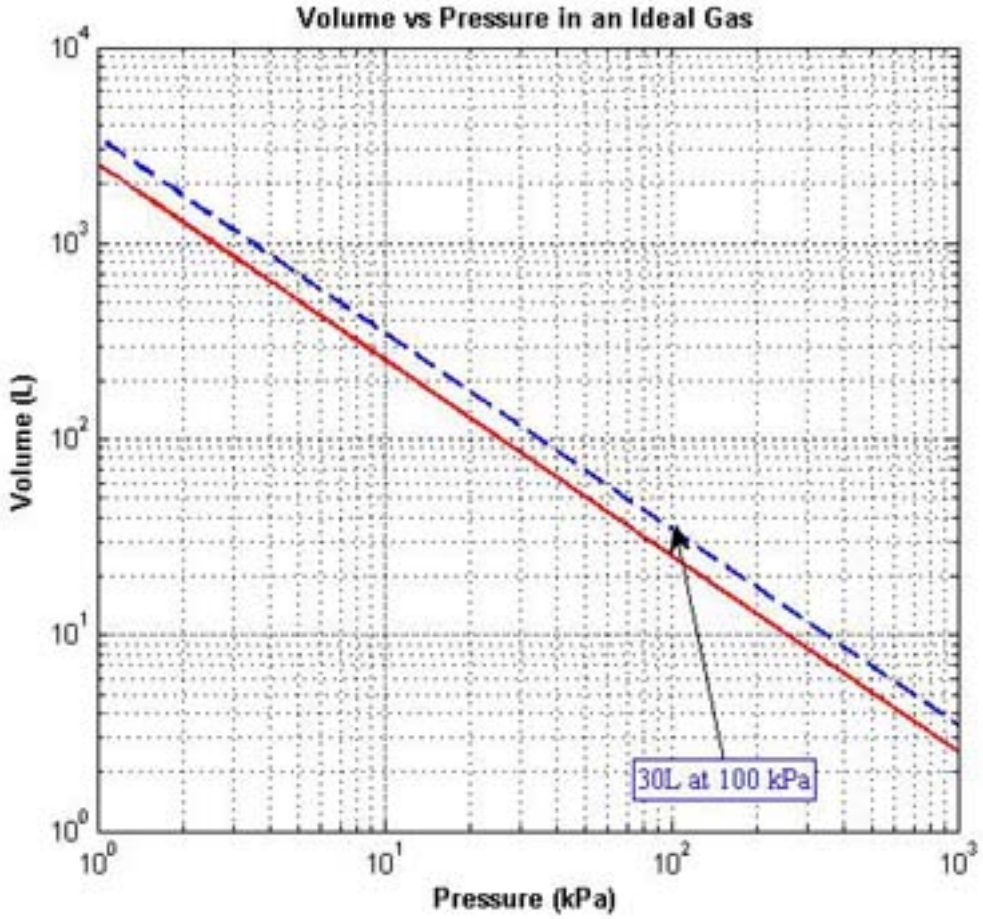


图 3.12 应用图工具条中的纺辑工具改变了蓝线的样式并添加了注释

测试 3.3

本测试提供了一个快速的检查方式，看你是否掌握了 3.5 的基本内容。如果你对本测试有疑问，你可以重读 3.5，问你的老师，或和同学们一起讨论。在附录 B 中可以找到本测试的答案。

1. 编写 matlab 语句，画出 sinx 和 cos2x 在 0 到 2π 之间的图象，其中步增量为 π/10。这个些点由 2pixel 宽的红线相边，每一个点用 6pixel 宽的绿色圆圈标记。

2. 应用图象编辑工作改变标记符号为黑色方块。添加注释并用带箭头的线指向 x=π 的点。用 matlab 文本字符编写以下表达式

3. $f(x) = \sin\theta\cos 2\phi$

4. Plot of Σx^2 versus x

下面的文本字符将产生怎样的表达式

5. '\tau\it_{m}'

6. '\bf\itx_{1}^{ 2} + x_{2}^{ 2} \rm(units:\bfm^{2}\rm)'

7. 如何在文本字符串产生反斜杠(\)?

3.6 程序调试的进一步说明

在含有选择结构和循环结构的程序出错的概率要比只含简单的顺序结构的程序出错的概率大得多。在完成了程序设计的步骤之后，无论多大的一个程序，在第一次运行时都很难通过。假如我们创建了一个程序并调试它，只发现这个程序的输出是错误的。我们怎样找到这些错误并修改它呢?

一旦程序包含了循环和选择结构，找到错语的最好的方法是应用 matlab 支持的符号调

试器(symbolic debugger)。这个调试器将会整合到 matlab 编辑器中。

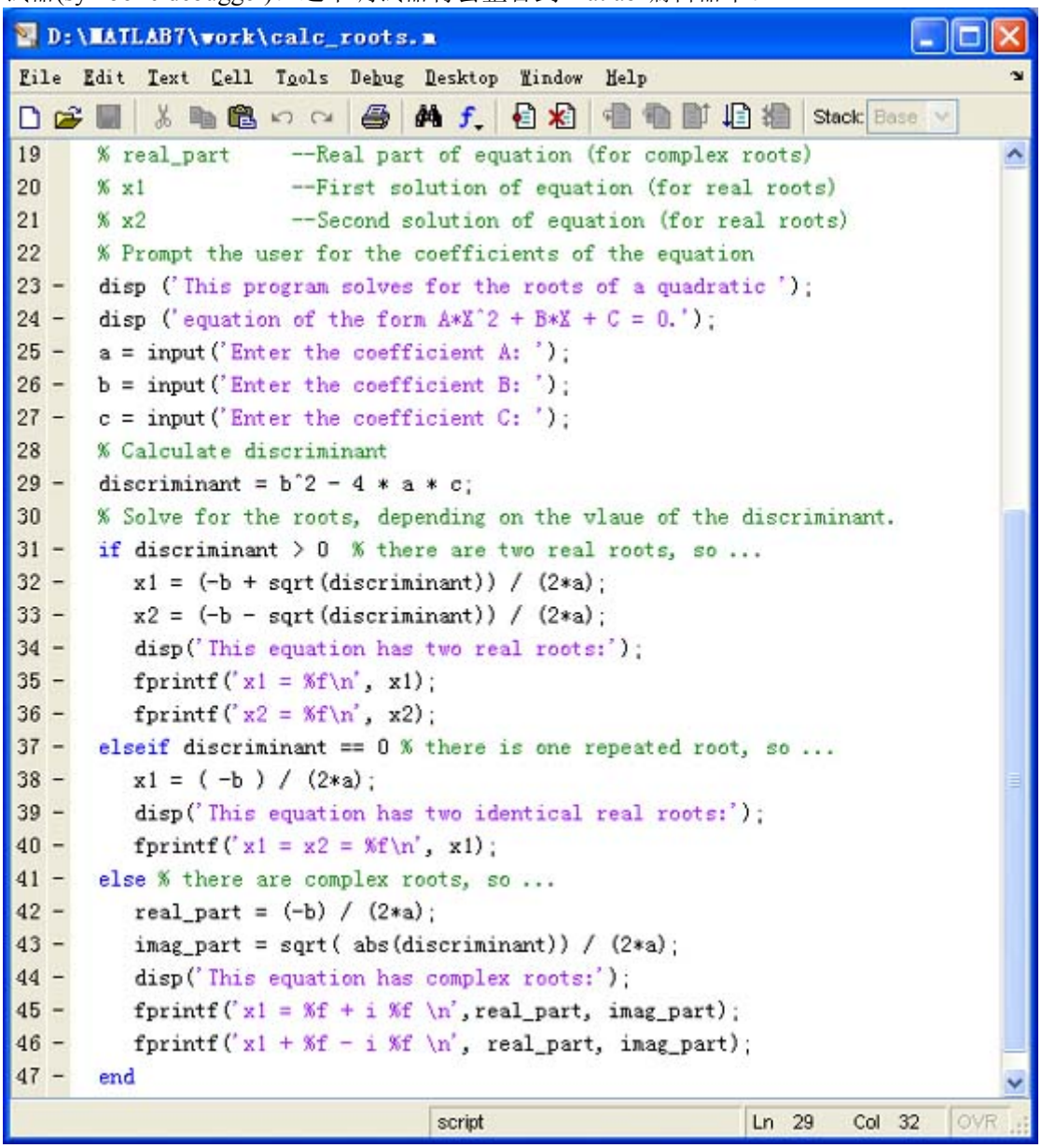


图 3.13 matlab 的编辑/调试窗口

应用这个调试器，首先应该选择“file/open”打开你在 MATLAB 命令窗口中要调试的程序。当一个文件被打开，编辑器就加载了这个文件，代码根据语法的不同出现不同的颜色。在这个文件中评论显示为绿色，变量和数字显示为黑色，字符串显示为红色，语言的关键字显示为蓝色。图 3.13 向大家展示的是含有文件 calc_roots.m 的编辑/调试窗口。

当一个程序执行时，我们想知道什么事情发生了。为了达到此目的，我们可以用鼠标右击你所关心的行并选择”set/clear breakpoint”选项。当一个断点被设置后，一个红色的点将会出现在行的左边，如图 3.14 所示。

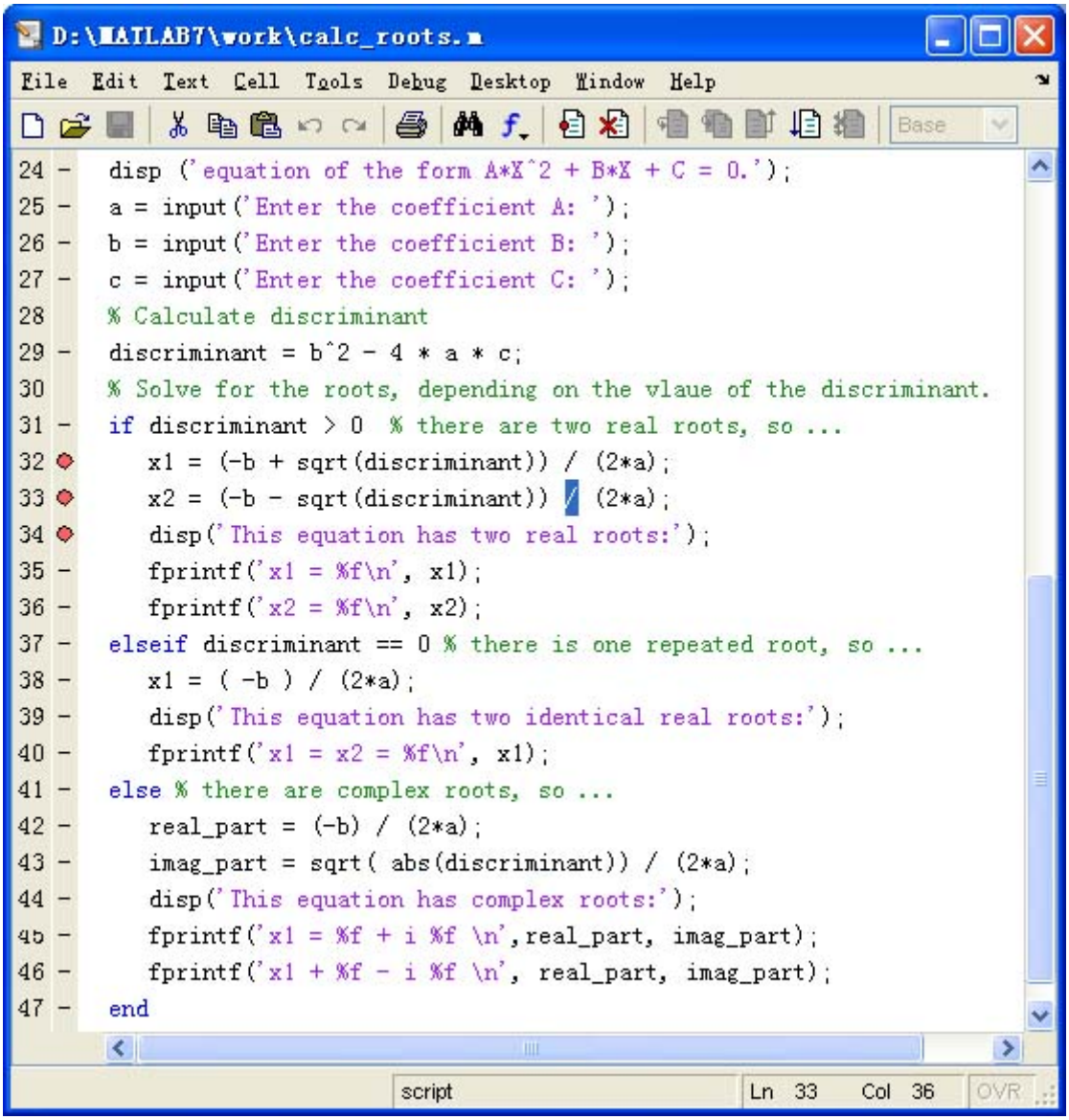


图 3.14 这个窗口断点已设置

一旦这些断点被设置，在命令窗口键入 calc_roots 将会像往常一样执行这个程序。这个程序将会运行到第一个断点并在那里停止。在调试的过程中将会有一个绿色的箭头将会出现在当前行。如图 3.15 所示。

一旦到达某个断点程序员可以通过在命令窗口中键入变量名的方法检查或修改在工作区内的任一变量。当程序员对程序的这一点感到满意时，可以通过重复按 F10 一行一行调试，也可以按 F5 运行到下一个断点。它总是能检测程序中的每一个断点中的任何一个变量的值。

调试器的另处一个重要特性是可在 Breakpoints 菜单中找到。这个菜单包括两个项目：“stop if Error”和“stop if warning”。如果程序中发生了一个错误，这个错误导致了电脑死机或产生了错误信息，程序员可以打开这些选项，并执行这个程序。这个程序将会运行到错误或警告的断点并停在那儿，它允许程序员检查变量的每一个值，并帮助找出出错的原因。当一个错误被发现，程序员能用编辑器来更正这个 MALTAB 程序，并把更新的版本存到磁盘上，在调试没结束之前，它必须重复以上的动作。这个步骤将会重复下去直到这个程序没有错误出错。

现在花一定的时间来熟悉这个调试器——这是值得的。

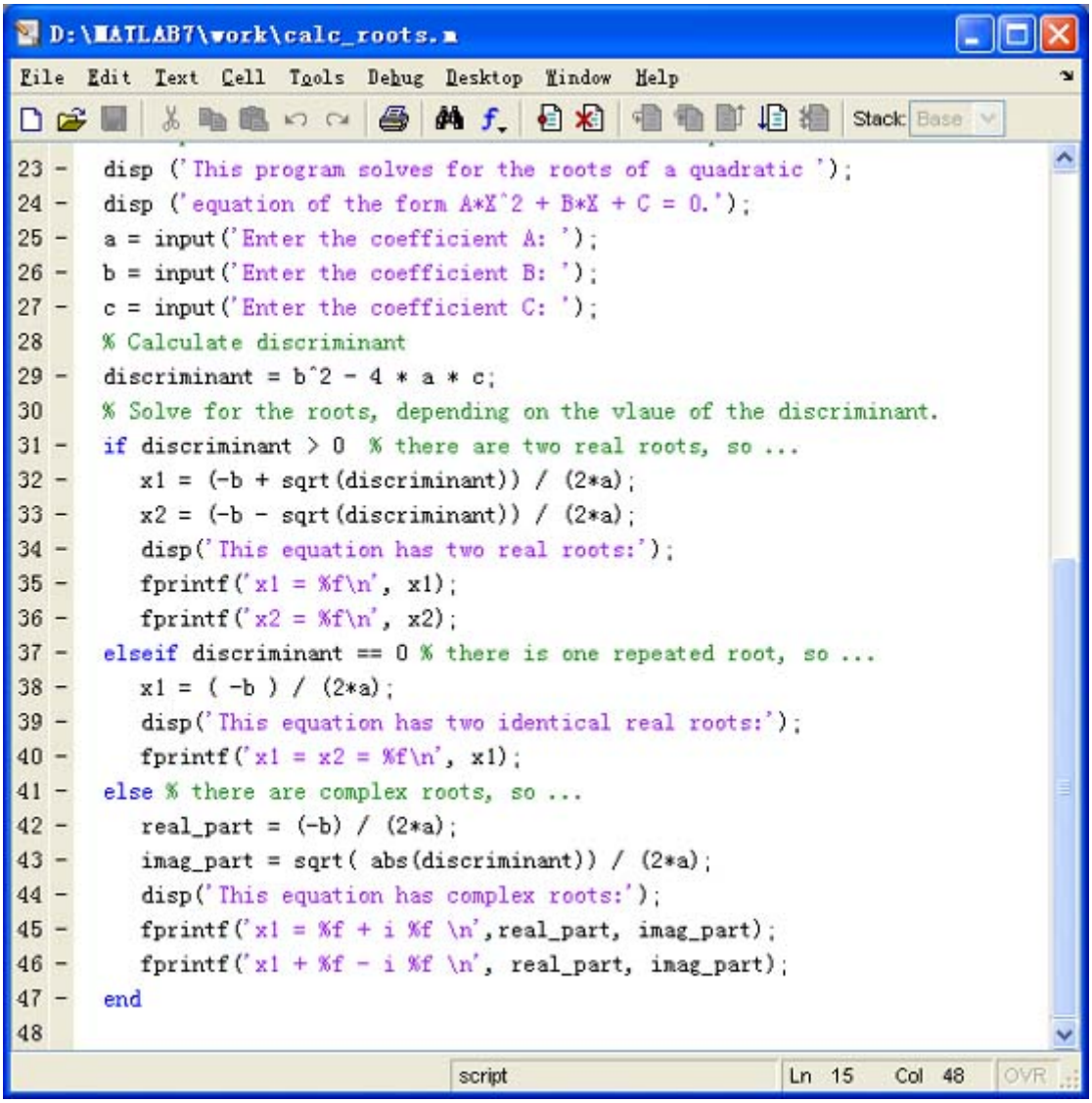


图 3.15 在调试的过程中一个绿色的小箭头将会出现在当前行的左侧

3.7 总结

在本章中，我们向大家展示了基本的 matlab 选择结构，还有控制这个结构的关系运算符和逻辑运算符。这个结构的其本类型是 if 结构。这个结构非常的灵活。如果这个结构需要的话，它可以跟任意多个 elseif 语句，if 结构可以进行嵌套组成更复杂的结构。第二种选择结构是 switch 结构，它提供多项选择。第三种选择结构是 try/catch 结构。它用于跳过错误以保证程序的继续进行。

第三章我们向大家介绍了更多的画图方法。axis 命令允许程序员指定 X，Y 轴的取值范围。hold 命令允许程序员把后面的图象叠加到原来的图象上打印。图命令允许程序员创建和选择多个图象窗口。subplot 命令允许程序在一个图象窗中创建多个子图象。

还有，我们学习如何控制画图的附加功能，例如线的宽度和符号的颜色。这些属性可由指定的“propertyname”和值 Value 决定，“propertyname”和值 Value 将出现在 plot 命令的数据后。运用流编辑器和转义序列将会增强对文本字符串的控制。用流字符串允许程序员指定相应的特性，例如字符的粗斜体，上下标和字体大小，字体类别。我们可以应用转义序列允许在文本中加入特殊的字符，比如说希腊字符和数学符号。

3.7.1 好的编程习惯的总结

在有选择结构和循环结构的编程中，要遵循以下的编程指导思想。如果你长期坚持这些原则，

你的代码将会有很少的错误，有了错误也易于修改，而且在以后修改程序时，也使别人易于理解。

1 . 在我们检测两数值是否相等时一定要小心，因为 round off 错误可能会使两个本来应该相等的值不相等了。这时你可以在 round off 错误的范围内它是不是近似相等。

2 . 遵守基本编程设计步骤来编写可靠，易理解的 matlab 的程序。

3 . 在 if 结构和 switch 语句中，语句块要缩进两个空格

3.7.2 matlab 总结

下面的总结列举了本章出现的所有特殊符号，命令和函数，后面跟的是简短的描述。

v=axis	此函数将会返回一个 4 元素行向量[xmin xmax ymin ymax]，其中 xmin xmax ymin ymax 代表 x，y 轴的上下限
axis([xmin xmax ymin ymax])	以 xmin xmax 设定横轴的下限及上限，以 ymin ymax 设定纵轴的下限及上限
axis equal	将横轴纵轴的尺度比例设成相同值
axis square	横轴及纵轴比例是 1:1
axis normal	以预设值画纵轴及横轴
axis off	将纵轴及横轴取消
axis on	这个命令打开所有的轴标签，核对符号，背景(默认情形)

3.8 练习

3.1

正弦函数的定义为 $\tan\theta=\sin\theta/\cos\theta$ 这个表达能求出角的正弦值，只要 $\cos\theta$ 的值不要太接近 0。假设 θ 用度为单位，编写相应的 matlab 语句来计算 $\tan\theta$ 的值，只要 $\cos\theta$ 大于等于 10^{-20}，如要小于 10^{-20}，那么打印出错误提示。

3.2

下面的语句用来判断一个人的体温是否处于危险状态（温度用的是华氏计量）。这些语句是否正确？如果不正确，指出错在那里？

```
if temp < 97.5
    disp('Temperature below normal');
elseif temp > 97.5
    disp('Temperature normal');
elseif temp > 99.5
    disp('Temperature slightly high');
elseif temp > 103.0
    disp('Temperature dangerously high');
end
```

3.3

在邮局发一个包裹，不超过两英磅的则收款为 10 美元。超过两英磅每英磅按 3.75 美元来计费，如果包裹的重量超过了 70 英磅，超过了 70 英磅的部分，每英磅的价格为 1.0 美元。如果超过了 100 英磅则拒绝邮递。编写一个程序，输入包裹的重量，输出它的邮费。

3.4

在例 3.3 中我们编写了一个程序用以计算 $f(x,y)$的值。这个函数的定义如下

$$f(x,y)=\begin{cases}x+y & x\ge 0 \text{ and } y\ge 0\\ x+y^2 & x\ge 0 \text{ and } y<0\\ x^2+y & x<0 \text{ and } y\ge 0\\ x^2+y^2 & x<0 \text{ and } y<0\end{cases}$$

在这里我们要求用 if 的嵌套结构来编写这个程序。

3.5

编写一个程序用以计算以下的函数

$$y(x)=\ln\frac{1}{1-x}$$

x 为自变量，x 的值小于 1。

3.6

编写一个程序允许使用者输入一个字符串，这个字符必须是一个星期中的一天(即"Sunday","Monday","Tuesday"等)，应用 switch 结构把这些字符串转化为相应的数字，以星期天为第一天，以星期六为最后一天。如果输入不是这七个字符串中的一个，那么输出提示信息。

3.7

理想气体定律。理想气体定律定义在例 3.7 中出现。假设 1 mol 的理想气体的体积为 10L，编写程序画出 P－T 图，温度的变化为 250K 到 400K。

3.8

天线增益模式。一个抛物面微波接收天线的接受增益 G 是关于执物面的反射角度 θ 的函数，

$$G(\theta)=|\mathrm{sinc}4\theta| \quad (-\frac{\pi}{2}\le\theta\le\frac{\pi}{2}) \qquad (3.6)$$

其中 sinc$x$=sin$x$/x。用极坐标画出 G 的图象。

3.9

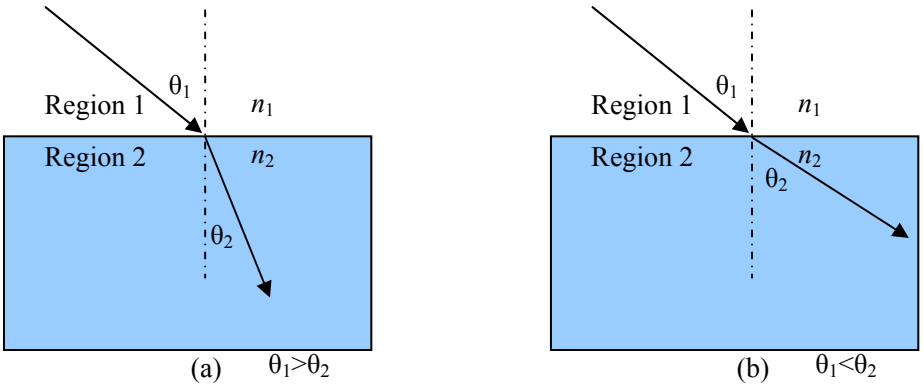
当光通过不同的介质时，就会发生折射如图(a)(b)。已知满足公式

$$n_1\sin\theta_1=n_2\sin\theta_2 \qquad (3.7)$$

n_1，n_2 分别代表介质 1，2 的折射率，θ_1 代表入射角，θ_2 代表折射角。

$$\theta_2=\sin^{-1}\left(\frac{n_1}{n_2}\sin\theta_1\right) \qquad (3.8)$$

如果 $n_1>n_2$，则入射光将会全部返回 1 介质中，而进入不了 2 介质中。编写一个程序来显示入射光在其中一介质内的边界，还有折射光在另一介质内范围。



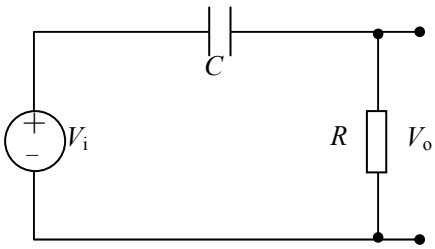
用下面的数据测试你的程序
(a) $n_1 = 1.0$, $n_2 = 1.7$, $\theta_1 = 45°$
(b) $n_1 = 1.7$, $n_2 = 1.0$, $\theta_1 = 45°$

3.10

假设复合函数定义如下
$f(t)=(0.5-0.25i)t-1.0$
计算出相应的函数 f 的幅度与相位。定义域为 $0 \leqslant t \leqslant 4$。

3.11

这是一个高通滤波器。



已知高通滤波器的输出输入电压比为

$$\frac{V_o}{V_i} = \frac{j2\pi fRC}{1+j2\pi fRC} \qquad (3.9)$$

计算并画出高通滤波器随频率变化输出电压的幅度和相位。

3.12

阿基米德螺旋。阿基米德螺旋用极坐标可描述为

$$r = k\theta \qquad (3.10)$$

r 是指到原点的距离，θ 为角度，单位为弧度。已知 $0 \leqslant \theta \leqslant 6\pi$，$k$=0.5，画出它的极坐标图。

3.13

内燃机的输出功率。内燃机的输出功率满足以下公式

$$P = \tau_{IND}\omega_m \qquad (3.11)$$

已知 $\omega_m = 188.5(1-e^{-0.2t})rad/s$，

$\tau_{IND} = 10e^{-0.2t} N \cdot m$，

0<t<10s。

画出两个函数 P 关于 t 的图象，第一个用线性尺度，第二个用对数尺度。

3.14

画轨道。一颗卫星绕地球运行，卫星的轨道是椭圆形的，而地球就处于这个椭圆的某一个焦点上。卫星的轨迹方程满足下式

$$r = \frac{P}{1 - \varepsilon \cos\theta} \qquad (3.12)$$

r 与 θ 分加代表卫星距地球的距离和两者形成的交角，P 是体现轨道大小的参数，ε 是来决定轨道形状的参数，ε 为 0 则轨道是圆形的，$0 \leqslant \varepsilon \leqslant 1$ 则说明轨道是椭圆形的。如果 $\varepsilon > 1$，则卫星要做离心运动。

已知卫星的 p=1000km，画出卫星的轨迹，已知

(a) ε=0；(b) ε=0.25；(c) ε=0.5

每一颗卫星到地球最近距离是多少?最远距离是多少?比较这三幅图，说出 p 代表意义是什么?