10.34 应用数值方法的化工工程

MATLAB 教程

肯尼斯・比尔斯
麻省理工学院化学工程系2001年8月1日

科学计算的性质

本课程侧重于使用计算机解决化学工程问题。 我们将学习如何解决描述动量、能量和质量传递的偏微分方程，积分描述化学反应器的常微分方程，以及模拟分子的动力学和预测最小能量结构。 这些问题是用数学运算来表达的，例如偏微分和积分，计算机无法理解。 它们只知道如何在内存中存储数字，并对它们执行简单的操作，如加法、减法、乘法、除法和指数运算。 不知何故，我们需要将这些问题的高级数学描述转化为一系列基本操作的序列。

逻辑上，我们可以开发模拟算法，将每个问题分解为以下形式的线性方程组。

$$a11^* x1+ a12^*x2 + ... + a1n^*xn = b1$$

$$a21^*x1 + a22^*x2 + ... + a2n^*xn = b2$$

.

.

.

$$an1^*x1 + an2^*x2 + ... + ann^*xn = bn$$

计算机可以理解这个系统中的操作（乘法和加法），我们可以通过矩阵方程Ax= b来非常普遍地表示这组方程，其中A={a_{ij}}是左侧系数矩阵，x是解向量，b是右侧系数向量。 这种普遍的表示使我们能够以一种一致的语言传递我们特定于系统的线性方程组给已经经过优化以非常高效地解决它们的预先编写的算法。这样可以节省我们每次编写新程序时编写线性求解器的工作量。 将重复的任务委托给可重用的、预先编写的子程序的方法使得使用计算机解决复杂技术问题的想法变得可行。 它还使我们能够利用几十年来应用数学研究的成果，开发出高效的数值算法。 科学程序通常涉及特定于问题的部分，执行参数输入和结果输出，将问题转化为一系列线性代数系统，然后程序大部分时间用于解决这些线性系统。 本课程主要关注理解科学计算的理论和概念，但我们还需要知道如何将这些概念转化为可工作的程序，并将我们特定于问题的代码与高效执行所需数值操作的预先编写的例程相结合。

那么，我们如何指示计算机解决我们的特定问题呢？在基本层面上，计算机所做的就是按照指令从指定的内存中检索数字。

在一些位置上，对它们进行一些简单的代数运算，并将它们存储在一些（可能是新的）内存位置中。计算机科学家为了不让计算机用户处理内存地址或者从内存到CPU的数据传递等细节，为每种类型的计算机开发了一个名为编译器的程序，将“人类级别”的代码转换成一组详细的机器级指令（包含在一个可执行文件中），以完成任务。使用编译器，很容易编写告诉计算机执行以下操作的代码：

1. 找到一个内存空间来存储一个实数x
2. 找到一个内存空间来存储一个实数y
3. 找到一个内存空间来存储一个实数z
4. 将x的值设为2
5. 将y的值设为4
6. 将存储在位置z的值设置为2*x + 3*y，其中符号*表示乘法

在FORTRAN中，第一个现代科学编程语言，经过修改后通常为FORTRAN 77，至今仍广泛使用，你可以通过编写以下代码来完成这些任务：

```
实数 x, y, z
x = 2
y = 4
z = 2*x + 3*y
```

然而，这段代码本身可以完成所需的任务，但没有提供任何方式供用户查看结果。 一个完整的FORTRAN程序执行任务并将结果写入屏幕的代码如下：

```
IMPLICIT NONE
实数 x, y, z
x = 2
y = 4
z = 2*x + 3*y
PRINT *, 'z = ',z
END
```

当这段代码使用FORTRAN 77编译器编译时，运行可执行文件后在屏幕上的输出为：z = 16.0000。编译型编程语言只允许简单输出文本、数字和二进制数据，因此任何结果的绘图必须由单独的程序完成。实际上，编写代码、将输出存储在具有适当格式的文件中，并将该文件读入单独的绘图或分析程序的要求，导致人们在小型项目中使用像EXCEL这样的“现成”软件，这些软件不适合技术计算；毕竟，EXCEL是用于商业电子表格的！

还有其他编译型编程语言存在，大多数比FORTRAN 77更强大，这是过去的遗产，主要是因为存在高效的数值计算例程。 虽然FORTRAN 77在功能上不如现代语言，但在执行速度方面通常具有优势。 在80年代和90年代，C和C++在更广泛的计算机科学界变得非常流行，因为它们允许更方便地组织和结构化数据，并为大型程序编写高度模块化的代码。 C和C++在科学计算界从未获得相同的流行程度，主要是因为它们的实现更加注重稳健性和通用性，而对执行速度的关注较少。 许多科学程序具有相对简单的结构，因此执行速度是主要关注点。 尽管情况在今天有所改变，但FORTRAN 90的引入及其更新的FORTRAN 95使FORTRAN语言焕发了新的生机。

FORTRAN 90/95包含了C/C++的许多数据结构能力，但是它是为技术人员编写的。 它是并行科学计算的首选语言，在执行过程中将任务分配给一个或多个CPU。随着双处理器工作站和BEOWOLF类型的集群越来越受欢迎，FORTRAN 90/95和High Performance Fortran等变种仍然是我个人重型科学计算的首选编译语言。

那么为什么这门课程使用MATLAB而不是FORTRAN 90呢？ FORTRAN 90是我在编译语言中的选择；然而，为了使用方便，MATLAB作为一种解释语言，在处理小到中等规模的任务时更好。 在编译语言中，“人类级别”的命令直接转换为存储在可执行文件中的机器指令。 只有在完成了所有编译过程后（运行时调试除外），才会执行命令。 在编译语言中，需要学习用于输入/输出数据（从键盘、到屏幕、到/从文件）的命令，以及为变量命名和分配内存空间的命令（如FORTRAN中的real命令）。编译语言的开发原则是语言应该具有最少的命令和语法，以便将任何可以通过一系列更基本的指令完成的任务留给子程序来处理，而不是将其纳入语言定义中。 应用数学界编写了子程序库来执行常见的数值运算（例如BLAS和LAPACK），但是要通过操作系统特定的命令将代码链接到它们才能访问它们。 虽然概念上并不困难，但对于小项目来说，开销并不可忽视。

在一种解释型语言中，语言的开发者已经编写并编译了一个主程序，在我们的例子中就是MATLAB程序，它将实时解释我们的命令给计算机。 当我们运行MATLAB时，会出现一个窗口，我们可以在其中输入命令进行数学计算。 然后，这段代码会被逐行解释（通过机器级指令）为其他实际执行我们请求的计算的机器级指令。 由于MATLAB需要逐个解释每个命令，因此我们需要更多的机器级指令来执行某个任务，而这在编译语言中是不需要的。 对于需要尽可能高效地利用计算机资源的复杂数值模拟，编译语言因此更加优越。

使用一种解释型语言的好处是，我们不需要预先编译代码。 因此，我们可以逐个输入命令并观察它们的执行情况（这对于查找错误非常有帮助）。 我们不需要将代码链接到子程序库，因为MATLAB作为预编译的程序，已经具备了所有所需的机器级指令。 FORTRAN 77/90/95、C和C++不能绘制图形，因此如果我们想要绘制程序的结果，我们需要将数据写入一个输出文件，然后将其作为输入传递给另一个图形程序。 相比之下，MATLAB程序员已经提供了图形例程，并将其与MATLAB代码解释器一起编译，因此我们不需要进行这个额外的数据传输步骤。 一种解释型语言可以提供高效且复杂的内存管理工具，通过在幕后操作，屏蔽了程序员不需要学习其复杂的使用语法。 因此，可以使用动态内存分配来创建新变量，而不需要用户理解指针（指向内存位置的变量），这在大多数编译语言中是必需的。 最后，由于MATLAB不是根据最小命令语法原则开发的，它包含了丰富的集成数值运算。 其中一些例程被设计为非常高效地解决线性问题。 其他例程在更高的层次上操作，例如以函数f(x)作为输入，并返回满足$f(x_0)=0$的点x_0，或者从t=0开始积分常微分方程dx/dt = f(x)。

出于这些原因，人们可以在解释语言中比在编译语言中更高效地编写代码（McConnell, Steve,Code Complete, Microsoft Press, 1993和Jones, Capers,Programming Productivity, McGraw-Hill, 1986），但由于每个命令的额外解释步骤，执行速度较慢。 但是，我们之前已经注意到在科学计算中，执行速度是一个重要考虑因素，那么这样可以接受吗？ MATLAB具有几个功能来缓解这种情况。 每当MATLAB首次运行子程序时，它会保存

解释过程的结果，以便后续调用不必重复这项工作。
此外，通过减少命令行的数量，可以减少解释开销，这也是FORTRAN90/95的良好编程风格的实践。
以一个例子来说，让我们来看看将一个M乘以N的矩阵A与一个N乘以P的矩阵B相乘，得到一个M乘以P的矩阵C的操作。在FORTRAN 77中，我们首先需要声明和分配内存来存储A、B和C矩阵（以及计数器整数i_row、i_col和i_mid），然后，可能在一个子程序中执行以下代码：

```
DO i_row = 1, M
  DO i_col = 1, N
    C(i_row,i_col) = 0.0
    DO i_mid = 1, P
      C(i_row,i_col) = C(i_row,i_col) + A(i_row,i_mid)*B(i_mid,i_col)
    ENDDO
  ENDDO
ENDDO
```

如果我们简单地将每一行从FORTRAN 77翻译成MATLAB，我们将得到以下代码段：

```
for i_row = 1:M
  for i_col = 1:N
    C(i_row,i_col) = 0;
    for i_mid = 1:P
      C(i_row,i_col) = C(i_row,i_col) + A(i_row,i_mid)*B(i_mid,i_col);
    end
  end
end
```

这段代码以与FORTRAN 77完全相同的方式执行任务，但现在每一行都必须逐行解释，增加了相当大的开销。 看起来我们应该使用FORTRAN 77更好；然而，在MATLAB中，语言被扩展以允许矩阵操作，因此我们可以使用单个命令完成相同的任务：C = A*B。我们甚至不需要预先分配内存来存储C，这将由MATLAB自动处理。MATLAB的方法更加可取，不仅因为它用更少的输入（和错误的机会）完成了相同的任务。 FORTRAN 77代码依赖于基本的标量加法和乘法操作，不容易并行化。 它指示计算机按照精确的事件顺序执行矩阵乘法，计算机受到限制必须遵循这个顺序。 单个命令C = A*B请求相同的任务，但让计算机自由决定以最高效的方式完成，例如通过将问题分割到多个处理器上。 FORTRAN 90/95相对于FORTRAN 77的主要优势之一是它也允许这些整个数组操作（相应的FORTRAN 90/95代码是C = MATMUL(A,B)），因此编写快速的MATLAB代码与FORTRAN 90/95一样奖励相同的编程风格，以便产生易于并行化的代码。

MATLAB还附带一个可选的编译器，将MATLAB代码转换为C或C++，并且可以编译此代码以生成独立的可执行文件。 因此，我们可以在一个解释语言中轻松编程，一旦程序开发完成，我们可以利用编译语言提供的高效执行和可移植性。 或者，借助编译器的工具，我们可以将MATLAB代码和数值例程与FORTRAN或C/C++代码结合使用。 鉴于这些优势，MATLAB似乎是科学计算入门课程中的一个强大语言选择。

MATLAB教程目录

本教程为每个章节提供了单独的网页。 教程中列出的命令使用以百分号%开头的注释行进行解释。这些

命令可以逐个输入或粘贴到交互式MATLAB窗口中。
通过输入help命令后跟命令的名称，可以获取有关特定命令的更多信息。 输入helpwin可以打开一个通用的帮助工具，而helpdesk则提供了链接到广泛的在线文档。 有关详细信息，请参阅10.34主页上的推荐阅读部分的文本。

MATLAB 教程

第一章。 基本的MATLAB命令
1.1 基本的标量运算

首先，让我们谈谈如何向程序中添加注释（例如这一行）。 注释是我们想要添加的文本行，用于解释我们正在做什么，这样如果我们或其他人以后阅读这段代码，就更容易弄清楚代码在做什么。 在MATLAB文件中，如果一行文本中出现百分号，%，那么百分号后面的所有文本都是注释，MATLAB不会尝试将其解释为命令。 首先，让我们向屏幕上写一条消息，说明我们开始运行第1.1节。

disp('字符串')命令将文本字符串显示在屏幕上。
disp('开始第1.1节...')

接下来，我们将一个变量设置为1。
x=1

这个命令既为变量x分配了内存空间（如果x尚未声明），又将值1存储在与该变量关联的内存位置中。

它还会在屏幕上显示"x = 1"。 通常，我们不希望屏幕上充斥着这样的输出，所以我们可以通过在命令结尾加上分号来使命令"不可见"。 作为一个例子，让我们使用以下命令"不可见"地将x的值更改为2，然后将结果写到屏幕上。 **x=2;**这会改变x的值，但不会写到屏幕上disp('我们已经改变了x的值。');

然后，我们通过输入"x"而不带分号来显示x的值。
x

现在，让我们看看如何声明其他变量。
y = 2*x; 这将y的值初始化为x的两倍
x = x + 1; 这将x的值增加1。
z = 2*x; 这声明了另一个变量z。

z不等于y，因为在声明每个变量时x的值发生了变化。

差异 = z - y

接下来，我们想要查看存储在内存中的变量列表。 为了做到这一点，我们使用命令"who"。

who;

我们可以通过使用"whos"来获得更多信息。
whos;

这些命令也可以用来获取关于特定变量的信息。
whos z 差异;

假设我们想要摆脱变量"差异"。
我们使用命令"clear"来做到这一点。
clear 差异;
who;

接下来，我们想要摆脱变量x和y。
同样，我们使用命令"clear"。

clear x y;
who;

通常，良好的编程风格是每行只写一个命令；然而，MATLAB
允许在一行上放置多个命令。
x = 5; y = 13; w = 2*x + y; who;

更常见的是，由于语法的长度，希望将单个命令延续到多行。 这可以通过使用三个点来实现。

z = 2*x + ...
y

最后，当使用clear命令时，我们可以一次性清除所有变量，命令为"clear
all"。
clear all;
who;它不打印任何内容，因为没有变量。

1.2. 基本向量运算

最简单但不推荐的声明变量的方法是逐个输入组件。

x(1) = 1;
x(2) = 4;
x(3) = 6;
x显示x的内容

通常最好一次性声明一个向量，这样MATLAB就知道需要从一开始分配多少内存。 对于大向量，这样做
更高效。
y = [1 4 6]与上面的代码执行相同的操作

注意，这声明了一个行向量。 要获得列向量，我们可以使用转置(对于复数x是伴随)运算符**xT = x';**将实数
行向量x转置，或者我们可以从一开始就将其作为列向量**yT = [1; 4; 6];**

为了看到行向量和列向量的维度差异，使用命令"size"
返回向量或矩阵的维度。
size(xT)
size(y)
size(yT)
命令length适用于行向量和列向量。
length(x), length(xT)

将两个向量相加或相减与标量类似。
z = x + y
w = xT - yT

将向量乘以标量同样简单直接。
v = 2*x
c = 4;
v2 = c*x

我们还可以使用. 运算符告诉MATLAB按元素方式执行给定操作。 假设我们想要设置每个y值，使得y(i) = 2*x
(i) + z(i)^2 + 1。
我们可以使用以下代码实现
y = 2.*x + z.^2 + 1

两个向量的点积和叉积可以通过以下方式计算
dot(x,y)
z=cross(x,y)

我们还可以使用符号[a : d : b]来定义一个向量。这将产生一个向量a，a + d，a + 2*d，a + 3*d，...直到我们得到一个整数n，其中a + n*d > b。看看这两个例子。
v = [0 : 0.1: 0.5];
v2 = [0 : 0.1: 0.49];

如果我们想要一个从a到b的具有N个均匀间隔点的向量，我们使用命令
"linspace(a,b,N)".
v2 = linspace(0,1,5)

有时候，我们会在程序的后面使用一个向量，但是想要在开始时将其初始化为零，并通过这样做来分配一块内存来存储它。这可以通过**v = linspace(0,0,100)';**为零的列向量分配内存来实现

最后，我们可以使用整数计数变量来访问矩阵的一个或多个元素。
v2 = [0 : 0.01 : 100];
c=v2(49)
w = v2(65:70)

清除所有

1.3. 基本矩阵运算

我们可以声明一个矩阵并直接给它赋值。
A = [1 2 3; 4 5 6; 7 8 9]
我们也可以使用逗号在一行上分隔元素。
B = [1,2,3; 4,5,6; 7,8,9]

我们可以从行向量构建矩阵
row1 = [1 2 3]; row2 = [4 5 6]; row3 = [7 8 9];
C = [row1; row2; row3]

或者从列向量构建。
column1 = [1; 4; 7];
column2 = [2; 5; 8];
column3 = [3; 6; 9];
D = [column1 column2 column3]

可以将几个矩阵连接起来形成一个更大的矩阵。
M = [A B; C D]

我们可以从矩阵中提取行向量或列向量。
row1 = C(1,:)
column2 = D(:,2)

或者，我们可以通过提取元素的子集来创建一个向量或另一个矩阵。
v = M(1:4,1)
w = M(2,2:4)
C = M(1:4,2:5)

通过使用 ' 运算符可以得到实矩阵的转置
D = A'
C, C'

对于复矩阵，' 返回伴随矩阵（转置和共轭）。通过使用 "仅转置" 命令可以去除共轭操作。'**E = D;**

E(1,2) = E(1,2) + 3*i;
E(2,1) = E(2,1) - 2*i;
E', E.'

"who" 命令列出了矩阵以及标量和向量变量。
谁

如果我们还想看到每个变量的维度，我们可以使用"whos"命令。
这告诉我们每个变量的大小和所需的存储器量。
whos

"size"命令告诉我们矩阵的大小。
M = [1 2 3 4; 5 6 7 8; 9 10 11 12];
size(M)
num_rows = size(M,1)
num_columns = size(M,2)

矩阵的加法、减法和乘法都很简单。
D = A + B
D = A - B
D = A*B

我们可以用多种方式声明矩阵。

我们可以创建一个包含m行和n列的矩阵，所有元素都为零，通过
m=3; n=4;
C = zeros(m,n)

如果我们想要创建一个N乘N的方阵，我们只需要使用一个索引。
C = zeros(n)

我们创建一个单位矩阵，所有元素都为零，除了主对角线上的元素为一。**D = eye(5)**

最后，我们可以使用. 运算符来执行逐元素操作，就像我们对向量做的那样。 下面的命令创建一个矩阵C，使得C(i,j) = 2*A(i,j) + (B(i,j))^2。
C = 2.*A + B.^2

矩阵以及所有其他变量都从内存中清除。
clear A B
whos
clear all
who

1.4. 使用字符字符串

在MATLAB中，当我们打印出结果时，通常希望用文本解释输出。 为了这个，字符字符串很有用。 在MATLAB中，字符字符串用单引号括起来。

course_name = '化工中应用数值方法的MATLAB教程'

要在字符串中插入撇号，我们将其重复两次，以避免将其与结束字符串的'
运算符混淆。
phrase2 = '课程的名称是：';
disp(phrase2), disp(course_name)

我们还可以以类似的方式组合字符串，就像处理数字的向量和矩阵一样。

word1 = '数值'; word2 = '方法'; word3='课程';
phrase3 = [word1, word2, word3]

我们可以看到这里没有包含空格，所以我们使用
phrase4 = [word1, ' ', word2, ' ', word3]

我们可以使用"int2str"命令将整数转换为字符串。
icount = 1234;
phrase5 = ['icount的值为', int2str(icount)]

同样，我们可以使用"num2str(number,k)"命令将浮点数转换为k位数的字符串。

Temp = 29.34372820092983674;
phrase6 = ['温度 = ',num2str(Temp,5)]
phrase7 = ['温度 = ',num2str(Temp,10)]

清除所有

1.5. 基本数学运算

指数运算命令
我们已经学习了如何进行加法、减法和乘法运算。 我们有时候也会使用^运算符，其中x^y表示x的y次方。**2^3, 2^3.3, 2.3^3.3, 2.3^(1/3.3), 2.3^(-1/3.3)**

平方根运算有自己的名称。
sqrt(27), sqrt(37.4)

用于分析数字符号的运算符包括
abs(2.3), abs(-2.3)返回一个数的绝对值
sign(2.3), sign(-2.3), sign(0)返回一个数的符号

计算指数和对数的命令为
a=exp(2.3)计算e^x
log(a)计算自然对数
log10(a)计算以10为底的对数

三角函数命令

可以直接调用pi的数值
pi, 2*pi

注意MATLAB以弧度计算角度

标准三角函数为
sin(0), sin(pi/2), sin(pi), sin(3*pi/2)
cos(0), cos(pi/2), cos(pi), cos(3*pi/2)
tan(pi/4), cot(pi/4), sec(pi/4), csc(pi/4)

它们的倒数是
asin(1),acos(1),atan(1),acot(1),asec(1),acsc(1)

双曲函数是
sinh(pi/4), cosh(pi/4), tanh(pi/4), coth(pi/4)
sech(pi/4), csch(pi/4)
它们的倒数是
asinh(0.5), acosh(0.5), atanh(0.5), acoth(0.5)
asech(0.5), acsch(0.5)

这些运算符可以按以下方式与向量一起使用。
x=linspace(0,pi,6)创建在0和pi之间的x值向量
y=sin(x)每个y(i) = sin(x(i))

四舍五入运算

round(x) : 返回最接近实数x的整数
round(1.1), round(1.8)

fix(x) : 返回向0方向最接近x的整数
fix(-3.1), fix(-2.9), fix(2.9), fix(3.1)

floor(x)：返回小于或等于x的最接近的整数
floor(-3.1), floor(-2.9), floor(2.9), floor(3.1)

ceil(x)：返回大于或等于x的最接近的整数
ceil(-3.1), ceil(-2.9), ceil(2.9), ceil(3.1)

rem(x,y)：返回整数除法x/y的余数
rem(3,2), rem(898,37), rem(27,3)

mod(x,y)：计算模数，即实数除法的余数
mod(28.36,2.3)

复数

使用i（或j）表示-1的平方根来声明复数。
z = 3.1-4.3*i
conj(z)返回共轭复数，conj(a+ib) = a - ib
real(z)返回复数的实部，real(a+ib) = a
imag(z)返回复数的虚部，imag(a+ib) = b
abs(z)返回绝对值（模），a^2+b^2
angle(z)返回相位角度theta，其中z = r*exp(i*theta)
abs(z)*exp(i*angle(z))返回z

对于复杂矩阵，运算符'计算伴随矩阵，即转置矩阵
并取每个元素的共轭
A = [1+i, 2+2*i; 3+3*i, 4+4*i]
A'取共轭转置（伴随操作）
A.'取转置而不共轭元素

坐标变换

2-D极坐标（theta,r）与笛卡尔坐标相关
x=1; y=1;
[theta,r] = cart2pol(x,y)
[x,y] = pol2cart(theta,r)

3-D球坐标（alpha,theta,r）由笛卡尔坐标获得
x=1; y=1; z=1;
[alpha,theta,r] = cart2sph(x,y,z)
[x,y,z] = sph2cart(alpha,theta,r)

清除所有

MATLAB 教程

第2章 编程结构
2.1. for循环

用于数值模拟的程序通常需要重复执行一组命令多次。 在MATLAB中，我们使用for循环指示计算机重复执行一段代码。 一个简单的for循环示例是**for i=1:10**重复执行i=1,2,...,10的代码

i打印循环计数器的值 **end**这结束了重复的代码部分。

计数器可以以非+1的值递增。
```
for i=1:2:10
disp(i);
end
```

这个例子显示计数器变量取值为1, 3, 5, 7, 9。 在9之后，代码尝试i=11，但由于11大于10（不小于或等于10），它不执行此次迭代的代码，而是退出for循环。

```
for i=10:-1:1
disp(i);
end
```

由于计数器整数的值在每次迭代中都会改变，for循环的常见用途是对向量或矩阵的不同元素执行给定的一组操作。 下面的示例演示了for循环的这种用法。

复杂结构可以通过嵌套循环来实现。 下面的嵌套循环结构将一个(m x p)矩阵与一个(p x n)矩阵相乘。

A = [1 2 3 4; 11 12 13 14; 21 22 23 24];A是一个3 x 4矩阵
B = [1 2 3; 11 12 13; 21 22 23; 31 32 33]; B是一个4 x 3矩阵
im = size(A,1);m是A的行数
ip = size(A,2);p是A的列数
in = size(B,2);n是B的列数
C = zeros(im,in);为包含0的m x n矩阵分配内存

现在我们来相乘这些矩阵
for i=1:im对C的每一行进行迭代
for j=1:in对每一行的元素进行迭代
for k=1:ip 对元素求和以计算C(i,j)
```
C(i,j) = C(i,j) + A(i,k)*B(k,j);
end
end
end
```
C打印代码结果
A*BMATLAB的例程做同样的事情

清除所有

2.2. if，case结构和关系运算符

在编写程序时，我们经常需要根据内存中变量的值做出决策。 这需要逻辑运算符，例如判断两个数字是否相等。MATLAB中常见的关系运算符有：eq(a,b)如果a等于b，则返回1，否则返回0。

eq(1,2), eq(1,1)
eq(8.7,8.7), eq(8.7,8.71)

当与向量或矩阵一起使用时，eq(a,b)返回与a和b相同大小的数组，其中元素为0表示a不等于b，为1表示a等于b。下面的示例演示了这种用法。

u = [1 2 3]; w = [4 5 6]; v = [1 2 3]; z = [1 4 3];
eq(u,w), eq(u,v), eq(u,z)
A = [1 2 3; 4 5 6; 7 8 9]; B = [1 4 3; 5 5 6; 7 9 9];
eq(A,B)
这个操作也可以使用==来调用
(1 == 2), (1 == 1), (8.7 == 8.7), (8.7 == 8.71)

ne(a,b)返回1，如果a不等于b，则返回0
ne(1,2), ne(1,1)
ne(8.7,8.7), ne(8.7,8.71)
ne(u,w), ne(u,v), ne(u,z)
ne(A,B)
另一种调用这个操作的方式是使用~=
(1 ~= 2), (1 ~= 1), (8.7 ~= 8.7), (8.7 ~= 8.71)

lt(a,b)如果a小于b则返回1，否则返回0
lt(1,2), lt(2,1), lt(1,1)
lt(8.7,8.71), lt(8.71,8.7), lt(8.7,8.7)
另一种执行此操作的方法是使用<
(1 < 2), (1 < 1), (2 < 1)

le(a,b)如果a小于或等于b则返回1，否则返回0
le(1,2), le(2,1), le(1,1)
le(8.7,8.71), le(8.71,8.7), le(8.7,8.7)
这个操作也可以使用<=来执行
(1 <= 1), (1 <= 2), (2 <= 1)

gt(a,b)如果a大于b则返回1，否则返回0
gt(1,2), gt(2,1), gt(1,1)
gt(8.7,8.71), gt(8.71,8.7), gt(8.7,8.7)
这个操作也可以使用>来执行
(1 > 2), (1 > 1), (2 > 1)

ge(a,b)如果a大于或等于b，则返回1，否则返回0
ge(1,2), ge(2,1), ge(1,1)
ge(8.7,8.71), ge(8.71,8.7), ge(8.7,8.7)
这个操作也可以使用>=来执行
(1 >= 1), (1 >= 2), (2 >= 1)

这些操作可以组合起来执行更复杂的逻辑测试。

(logic1)&(logic2)只有当logic1和logic2都不等于零时才返回0
((1==1)&(8.7==8.7))
((1==2)&(8.7==8.7))
((1>2)&(8.71>8.7))
((1<2)&(8.7<8.71))
((1>2)&(8.7>8.71))
i1 = 1; i2 = 0; i3=-1;
(i1 & i1), (i1 & i2), (i2 & i1), (i2 & i2), (i1 & i3)
((1==1)&(8.7==8.7)&(1<2))
((1==1)&(8.7==8.7)&(1>2))

通过使用命令all(vector1)，可以更容易地将此操作扩展到多个操作，该命令返回1，如果vector1的所有元素都非零，否则返回0**all([i1 i2 i3]), all([i1 i1 i3])**

如果logic1或logic2中的任何一个不等于零，或者它们都不等于零，则or(logic1,logic2)返回1。

or(i1,i2), or(i1,i3), or(i2,i2)
通过使用命令any(vector1)，可以将此操作扩展到多于两个逻辑变量，该命令返回1，如果vector1的任何元素都非零，否则返回0。

any([i1 i2 i3]), any([i2 i2 i2]), any([i1 i2 i2 i2]),

在科学计算中较少使用的是异或构造 xor(logic1,logic2)，只有当logic1或logic2中的一个非零时才返回1，但不能同时为非零。
xor(i1,i1), xor(i2,i2), xor(i1,i2)

我们使用这些关系运算来决定是否使用if结构来执行一块代码，其一般形式如下。

```
logictest1 = 0; logictest2 = 1; logictest3 = 0;
if(logictest1)
disp('执行块1');
elseif(logictest2)
disp('执行块2');
elseif(logictest3)
disp('执行块3');
else
disp('执行结束块');
end
```

如果之前的代码块都没有执行，则执行最后一个代码块。

```
logictest1 = 0; logictest2 = 0; logictest3 = 0;
如果(logictest1)
显示('执行块1');
否则如果(logictest2)
显示('执行块2');
否则如果(logictest3)
显示('执行块3');
否则
disp('执行结束块');
end
```

if循环不会执行多个代码块。 如果多个logictest变量都不等于零，则它遇到的第一个变量是它执行的变量。

```
logictest1 = 0; logictest2 = 1; logictest3 = 1;
如果(logictest1)
显示('执行块1');
否则如果(logictest2)
显示('执行块2');
否则如果(logictest3)
显示('执行块3');
否则
disp('执行结束块');
end
```

if结构经常与for循环结合使用。 例如，下面的例程将向一个矩阵的主对角线上添加一个向量的分量，该矩阵是两个矩阵A和B的和。

```
A = [1 2 3; 4 5 6; 7 8 9];
B = [11 12 13; 14 15 16; 17 18 19];
u = [10 10 10];
C=zeros(3);
for i=1:3
for j=1:3
if(i==j)
C(i,j) = A(i,j) + B(i,j) + u(i);
else
C(i,j) = A(i,j) + B(i,j);
end
end
end
```

作为if块的替代方案，可以使用case结构来选择不同的
替代方案。

```
for i=1:4
switch i;
case {1}
disp('i是一');
case {2}
disp('i是二');
case {3}
disp('i是三');
otherwise
disp('i不是一、二或三');
end
end
```

清除所有

2.3. while循环和控制语句

当逻辑测试表达式返回非零值时，WHILE循环执行一段代码块。

```
error = 283.4;
tol = 1;
factor = 0.9;
while (error > tol)
error = factor*error;
disp(error)
end
```

如果factor >= 1，那么error的值将增加，while循环将不会终止。 一种更好的方法是使用for循环来设置迭代次数的上限。 “break”命令停止最深层嵌套的for循环的迭代，并在达到条件（error < tol）时调用。

```
error = 283.4;
tol = 1;
factor = 0.9;
iter_max = 10000;
iflag = 0;表示目标未达到
for iter=1:iter_max
if(error <= tol)
iflag = 1;表示目标已达到
break;
end
```

错误 = 因子*错误;
显示(错误)
结束
如果**(iflag==0)**写一条消息说明目标未达成。
显示('目标未达成');
显示(['错误 = ' num2str(错误)]);
显示(['容差 = ',num2str(容差)]);
结束

清除所有

2.4. 屏幕输入/输出

在MATLAB中，将输出写入屏幕的基本命令是"显示"。
显示('显示命令将字符字符串写入屏幕。');

当将整数或实数写入屏幕时，应使用"int2str"和"num2str"命令
(有关详细信息，请参阅教程的第1章。
i = 2934;
x = 83.3847;
显示(['i = ' int2str(i)]);
显示(['x = ' num2str(i)]);

允许用户从键盘输入数据的标准命令是"输入"。
i = 输入('输入整数 i : ');
x = 输入('输入实数 x : ');
v = input('输入向量v：');尝试输入[1 2 3]
i, x, v

清除所有

MATLAB 教程

第三章。 基本绘图例程
3.1. 2-D图

制作2-D图的基本命令是“plot”。 以下代码绘制了函数sin(x)的图形。

```
x = linspace(0,2*pi,200);
f1 = sin(x);
plot(x,f1)
```

现在我们添加一个标题和x轴、y轴的标签
```
标题('f_1 = sin(x)的图形');
标签('x');
标签('f_1');
```

让我们更改坐标轴，使其只绘制x从0到2*pi。
坐标轴([0 2*pi -1.1 1.1]);[xmin xmax ymin ymax]

接下来，我们用cos(x)创建一个新的图形
```
f2 = cos(x);
figure;创建一个新的图形窗口
plot(x,f2);
title('f_2 = cos(x)的图像');
xlabel('x');
ylabel('f_2');
axis([0 2*pi -1.1 1.1]);
```

现在，我们将两个图像合并到一个图中
```
figure;创建一个新的图像
plot(x,f1);
hold on;告诉MATLAB不要覆盖当前图像
```

如果忘记输入hold on会发生什么？ "hold off"会取消hold状态。
```
plot(x,f2,'r');用红色曲线绘制
title('f_1 = sin(x), f_2 = cos(x)的图像');
xlabel('x');
ylabel('f_1, f_2');
axis([0 2*pi -1.1 1.1]);
```

现在我们添加一个图例。
```
legend('f_1', 'f_2');
```

如果我们想要移动图例，可以在图像窗口的"工具"菜单中打开"启用绘图编辑"，然后将图例拖动到所需位置。

最后，我们使用命令“gtext”添加一行文本，然后使用光标将其定位在图上。

```
gtext('f_1=f_2在两个位置相交');
```

命令“help plot”告诉我们如何使用不同类型的点而不是线条制作图形，以及如何选择不同的颜色。

清除所有

3.2. 三维图

首先，我们生成一个包含每个点的x和y值的网格。

x = 0:0.2:2*pi;在x轴上创建点的向量
y = 0:0.2:2*pi;在y轴上创建点的向量

现在，如果n=length(x)和m=length(y)，那么网格将包含N=n*m个网格点。 XX和YY是n行m列的矩阵，分别包含每个网格点的x和y值。
[XX,YY] = meshgrid(x,y);

从以下行中可以看出点的编号约定。
x2 = 1:5; y2 = 11:15;
[XX2,YY2] = meshgrid(x2,y2);
XX2, YY2

这表明XX2(i,j)包含x向量的第j个分量，YY2(i,j)包含y向量的第i个分量。

现在，我们生成一个函数，将每个（x，y）2-D网格点保存为单独的z轴值。
Z1 = sin(XX).*sin(YY);计算要绘制的函数值

创建一个彩色网格图
figure; mesh(XX,YY,Z1);
xlabel('x'); ylabel('y'); zlabel('z'); title('sin(x)*sin(y)');

创建一个彩色表面图
figure; surf(XX,YY,Z1);
xlabel('x'); ylabel('y'); zlabel('z'); title('sin(x)*sin(y)');

创建一个等高线图
figure; contour(XX,YY,Z1);
xlabel('x'); ylabel('y'); zlabel('z'); title('sin(x)*sin(y)');

创建一个填充等高线图，并使用条形图显示函数值
figure; contourf(XX,YY,Z1); colorbar;
xlabel('x'); ylabel('y'); zlabel('z'); title('sin(x)*sin(y)');

创建一个三维等高线图
figure; contour3(XX,YY,Z1);
xlabel('x'); ylabel('y'); zlabel('z'); title('sin(x)*sin(y)');

清除所有

3.3. 制作复杂图形

使用subplot命令，可以将多个图形合并到一个单独的图中。 我们想要创建一个包含nrow行图形和每行ncol个图形的主图。subplot(nrow,ncolumn,i)在主图中创建一个新的图形窗口，其中i是根据以下顺序在主图中的位置的数字：1 2 3 ... ncol

ncol+1 ncol+2 ncol+3 ... 2*ncol

首先，生成要绘制的数据。
x = 0:0.2:2*pi;

```
y = 0:0.2:2*pi;
f1 = sin(x);
f2 = cos(y);
[XX,YY] = meshgrid(x,y);
Z1=sin(XX).*cos(YY);
```

以下代码创建了一个包含四个子图的图形。
figure;创建一个新的图形

subplot(2,2,1);创建第一个子图窗口
```
plot(x,f1); title('sin(x)');
xlabel('x'); ylabel('sin(x)'); axis([0 2*pi -1.1 1.1]);
```

subplot(2,2,2);创建第二个子图窗口
```
plot(y,f2); title('cos(y)');
xlabel('y'); ylabel('cos(y)'); axis([0 2*pi -1.1 1.1]);
```

subplot(2,2,3);创建第三个子图窗口
```
surf(XX,YY,Z1); title('sin(x)*cos(y)');
xlabel('x'); ylabel('y'); zlabel('z');
```

subplot(2,2,4);创建第四个子图窗口
```
contourf(XX,YY,Z1); colorbar; title('sin(x)*cos(y)');
zlabel('x'); ylabel('y');
```

清除所有

MATLAB 教程

第四章。 高级矩阵运算
4.1. 稀疏矩阵

稀疏矩阵
为了展示使用稀疏矩阵所获得的效率，我们将两次使用有限差分法来求解一个偏微分方程。 首先，我们将使用我们迄今为止学到的使用完整矩阵的矩阵命令。其次，我们将使用新的命令，利用大部分元素为零的事实，大大减少解决偏微分方程所需的内存和浮点运算次数。

清除所有;从内存中删除所有现有变量

num_pts = 100;模拟中的网格点数

使用完整矩阵格式进行计算

通过使用中心有限差分法离散化一维拉普拉斯算子，得到以下矩阵。

x = 1:num_pts;x值的网格

使用包含num_pts个点且点之间间距为1的一维网格离散化PDE得到矩阵。

```
Afull=zeros(100,100);
Afull(1,1) = 1;
Afull(num_pts,num_pts) = 1;
对于内部点i=2:(num_pts-1)求和
Afull(i,i) = 2;
Afull(i,i-1) = -1;
Afull(i,i+1) = -1;
end
```

在x=1和x=num_pts处设置Dirichlet边界条件。
BC1 = -10;x(1)处的f值;
BC2 = 10;x(num_pts)处的f值;

对于内部点，我们有一个源项。
b_RHS = linspace(0,0,num_pts)';创建零列向量
b_RHS(1) = BC1;
b_RHS(num_pts) = BC2;
b_RHS(2:(num_pts-1)) = 0.05;对于内部点，b_RHS是源项

现在我们使用标准的MATLAB求解器来获得PDE在网格点上的解。
f = Afull _RHS;
figure; plot(x,f);
title('从FD-CDS方法得到的PDE解（完整矩阵）');
xlabel('x'); ylabel('f(x)');

现在让我们更仔细地看一下Afull。命令spy(A)通过在A的非零值处写入一个点来绘制矩阵A的图形。

figure;
spy(Afull); title('Afull的结构');

底部的数字nz是非零元素的数量。 我们可以看到矩阵元素中只有很小一部分是非零的。 由于我们按照规则对网格点进行编号，并将每个网格点的邻居存储在相邻位置，因此该矩阵中的非零元素位于主对角线和上下相邻的两个对角线上。 即使我们对网格点进行不规则编号，仍然只有很少的非零点。 在数值求解PDE时，我们经常遇到稀疏矩阵，即它们的点只有很小一部分是非零的。 对于这个矩阵，元素的总数是num_elements = num_pts*num_pts;

nzA = nnz(Afull);返回Afull中非零元素的数量
fraction_filled = nzA/num_elements

这意味着Afull大部分是空白空间，我们浪费了很多内存来存储我们知道是零的值。

从内存中删除除Afull以外的所有变量。
clear x f b_RHS BC1 BC2 i num_elements nzA fraction_filled;

稀疏矩阵

我们可以使用"sparse"命令将矩阵转换为稀疏格式。
Asparse = sparse(Afull)

MATLAB将稀疏矩阵存储为一个NZ乘以3的数组，其中NZ是非零元素的数量。 第一列是非零元素的行号，第二列是列号。 第三列是非零元素的实际值。 总内存使用量远小于完整矩阵格式。

whos Afull Asparse;
clear Asparse; 清除稀疏矩阵

现在我们将使用稀疏矩阵格式进行求解

接下来，我们设置网格点的值
x = 1:num_pts;网格的x值

现在我们声明矩阵A从一开始就具有稀疏矩阵结构。 首先，我们计算非零元素的数量（或者这个数量的上界）。 我们可以看到，对于每个对应于内部点的行，我们有3个值，而对于第一行和最后一行，我们只有一个值。 因此，非零元素的数量是**nzA = 3*(num_pts-2) + 2;**

现在我们使用"spalloc(m,n,nz)"为m行n列的稀疏矩阵分配内存，其中非零元素的数量不超过nz个。

A = spalloc(num_pts,num_pts,nzA);

现在我们设置A矩阵的值。
A(1,1) = 1;
A(num_pts,num_pts) = 1;
for i=2:(num_pts-1)
A(i,i) = 2;
A(i,i-1) = -1;
A(i,i+1) = -1;
end

在x=1和x=num_pts处设置Dirichlet边界条件。
BC1 = -10;x(1)处的f值;
BC2 = 10;x(num_pts)处的f值;

对于内部点，我们有一个源项。
b_RHS = linspace(0,0,num_pts)';创建零列向量
b_RHS(1) = BC1;
b_RHS(num_pts) = BC2;
b_RHS(2:(num_pts-1)) = 0.05;对于内部点，b_RHS是源项

现在，当我们调用MATLAB标准求解器时，它会自动识别A是一个稀疏矩阵，并使用能够利用这一事实的求解算法。
f = A _RHS;

figure; plot(x,f);
title('使用FD-CDS方法求解的PDE解（稀疏矩阵）');
xlabel('x'); ylabel('f(x)');
whos A Afull;

从A和Afull的行中，我们可以看到稀疏矩阵格式所需的内存远远少于完整矩阵格式。 此外，如果N是网格点的数量，我们可以看到完整矩阵的大小为N^2；而稀疏矩阵在内存中的大小仅约为3*N。 因此，随着N的增加，稀疏矩阵格式比完整矩阵格式更加高效。 对于具有数千个网格点的复杂模拟，如果不利用稀疏性，我们无法解决这些问题。 为了看到使用稀疏矩阵可以获得的执行速度提升，我们来比较以下两个矩阵相乘的算法。

用于矩阵乘法的完全矩阵算法

Bfull = 2*Afull;
Cfull = 0*Afull;为C=A*B声明内存
num_flops = 0;
for i=1:num_pts
for j=1:num_pts
for k=1:num_pts
Cfull(i,j) = Cfull(i,j) + Afull(i,k)*Bfull(k,j);
num_flops = num_flops + 1;
end
end
end
disp(['完全矩阵格式的FLOPS数量 = ', int2str(num_flops)]);

用于矩阵乘法的稀疏矩阵算法

B = 2*A;
nzB = nnz(B);B的非零元素数量
nzC_max = round(1.2*(nzA+nzB));猜测我们将需要多少内存来存储C
C = spalloc(num_pts,num_pts,nzC_max);
[iA,jA] = find(A);找到A中非零的(i,j)元素
[iB,jB] = find(B);找到B中非零元素的(i,j)位置
num_flops = 0;
for ielA = 1:nzA遍历A中的非零元素
for ielB = 1:nzB遍历B中的非零元素
if(iB(ielB)==jA(ielA))该对元素对C有贡献
i = iA(ielA);
k = jA(ielA);

```
j = jB(ielB);
C(i,j) = C(i,j) + A(i,k)*B(k,j);
num_flops = num_flops + 1;
end
end
end
disp(['稀疏矩阵格式的FLOPS数量 = ', int2str(num_flops)]);
D = Cfull - C;检查两种算法是否给出相同的结果
disp(['Cfull与C不相等的元素数量：' int2str(nnz(D))]);
```

最后，我们注意到取稀疏矩阵的逆矩阵通常会破坏大部分的稀疏
性。

```
figure;
subplot(1,2,1); spy(A); title('A的结构');
subplot(1,2,2); spy(inv(A)); title('inv(A)的结构');
```

因此，如果我们已知A和C = A*B的值，并且想要计算矩阵B，我们不使用inv(A)*C。 相反，我们使用"左矩阵除法"运算符A\C。这将返回一个等价于inv(A)*C的矩阵，但使用了能够利用MATLAB求解器的稀疏性的方法。

```
B2 = A\C;
figure; spy(B2); title('B2的结构');
```

我们可以看到，消元法引入了非常小的非零值到中心三个对角线之外的元素中。 我们可以通过保留大于某个容差值的元素来去除这些非零值。

```
tol = 1e-10;
Nel = nnz(B2);
[iB2,jB2] = find(B2);返回非零元素的位置
对于iel=1:Nel
如果(abs(B2(iB2(iel),jB2(iel))) < tol) 设置为零
B2(iB2(iel),jB2(iel)) = 0;
end
end
B2 = 稀疏(B2);减少内存存储
图; spy(B2); 标题('"清理"B2的结构');
```

由于我们不想经历中间步骤，其中我们必须存储一个具有许多非零元素的矩阵，通常我们不以这种方式计算矩阵。 相反，我们限制自己解决形式为A*x = b的线性系统，其中x和b是向量，A是一个稀疏矩阵，我们直接输入其值。 因此，我们避免了由于舍入误差产生许多非零元素而引起的内存问题。

```
清除所有
```

4.2. 常见矩阵运算/特征值

通过使用"det"计算方阵的行列式。

```
A = rand(4);创建一个随机的4x4矩阵
det(A)计算A的行列式
```

矩阵的其他常见函数有

```
rank(A)A的秩
trace(A)A的迹
norm(A)A的矩阵范数
cond(A)A的条件数
```

A_inv=inv(A)计算A的逆矩阵
A*A_inv

使用命令"eig"计算矩阵的特征值
eig(A)

如果还需要特征向量，语法为
[V,D] = eig(A)

这里，V是一个包含特征向量的矩阵，每列是一个特征向量，D是一个对角矩阵，包含特征值。

for i=1:4
eig_val = D(i,i);
eig_vect = V(:,i);
A*eig_vect - eig_val*eig_vect
end

命令"eigs(A,k)"计算矩阵A的前k个特征值，即具有最大模的k个特征值。

eigs(A,1)估计A的主特征值

同样地，主特征值的特征向量也可以用eigs计算。
[V2,D2] = eigs(A,1);
eig_vect = V2; eig_val = D2;
A*eig_vect - eig_val*eig_vect

对于稀疏矩阵，只能使用"eigs"命令。

清除所有

4.3. LU分解

线性方程组Ax=b可以使用LU分解解决，可以使用多个b向量。 在这里，我们进行分解P*A = L*U，其中P是置换矩阵（因此inv(P)=P'），L是下三角矩阵，U是上三角矩阵。 当在因式分解过程中没有进行枢轴选取时，P是一个单位矩阵（即本质上是高斯消元）。 一旦LU分解完成，可以使用以下线性代数步骤解决问题Ax=b。

A*x = b
P*A*x = P*b
L*U*x = P*b

这给出了以下两个涉及三角矩阵的线性问题，可以通过代入法解决。

L*y = P*b
U*x = y
执行LU分解的MATLAB命令是"lu"。我们使用一个随机的非奇异矩阵来演示算法。 通过添加一个单位矩阵的因子来确保非奇异性。

A = rand(10) + 5*eye(10);

执行LU分解。
[L,U,P] = lu(A);
max(P*P'-eye(10))演示P是正交矩阵
max(P*A - L*U)显示舍入误差的最大结果

比较涉及的矩阵的结构
```
figure;
subplot(2,2,1); spy(A); title('A的结构');
subplot(2,2,2); spy(P); title('P的结构');
subplot(2,2,3); spy(L); title('L的结构');
subplot(2,2,4); spy(U); title('U的结构');
```

对于稀疏矩阵，可以以完全相同的方式调用LU分解；然而，一般来说，分解后的矩阵L和U不像A那样稀疏，因此使用LU分解会损失一些效率。 这个问题在矩阵的带宽越大时变得更加严重，即非零值距离主对角线越远时。

有时候我们只需要一个近似的因式分解B=L*U，其中B与A足够接近，使得C = inv(B)*A与单位矩阵的差别不大，即C的最大特征值与最小特征值的比值小于A的比值。在这种情况下，B被称为预处理器，并且用于优化方法和解决某些类别的线性系统。 当我们进行不完全LU分解时，只计算与A中非零元素相对应的L和U的元素，或者根据不同的选项，忽略绝对值小于指定值的元素。

以下代码演示了不完全LU分解的使用方法。

使B=A，除了将某些元素设为零。
```
B=A;
```

将一些远离对角线的元素设为零。
```
for i=1:10
B(i+5:10,i) = 0;
B(1:i-5,i) = 0;
end

B=sparse(B);
[Linc,Uinc,Pinc] = luinc(B,'0');

figure;
subplot(2,2,1); spy(B); title('B的结构');
subplot(2,2,2); spy(Pinc); title('Pinc的结构');
subplot(2,2,3); spy(Linc); title('Linc的结构');
subplot(2,2,4); spy(Uinc); title('Uinc的结构');

D1 = P*A - L*U;
D2 = Pinc*B - Linc*Uinc;
tol = 1e-10;设置元素为零的容差
for i=1:10
for j=1:10
if(D1(i,j)<tol)
D1(i,j) = 0;
end
if(D2(i,j)<tol)
D2(i,j) = 0;
end
end
end
```

figure;
subplot(1,2,1); spy(D1); title('(P*A - L*U)');
subplot(1,2,2); spy(D2); title('(Pinc*B - Linc*Uinc)');

但是，看一下B的特征值和近似分解的特征值。
Bapprox = Pinc'*Linc*Uinc;
eigs(B) B矩阵的特征值
C = Bapprox\B;inv(Bapprox)*B（不要对稀疏矩阵使用"inv"）
eigs(C)

清除所有

4.4. QR分解

通过调用"qr"命令执行因子分解A*P = Q*R，其中P是置换矩阵，Q是正交矩阵，R是上三角矩阵。

A = rand(6);
[Q,R,P] = qr(A);

Q*Q' 显示Q是正交的
A*P - Q*R

figure;
subplot(2,2,1); spy(A); title('A的结构');
subplot(2,2,2); spy(P); title('P的结构');
subplot(2,2,3); spy(Q); title('Q的结构');
subplot(2,2,4); spy(R); title('R的结构');

如果需要分解A=QR（即P=1），则命令为：**[Q,R] = qr(A);**

figure;
subplot(2,2,1); spy(A); title('A的结构');
subplot(2,2,2); spy(Q); title('Q的结构');
subplot(2,2,3); spy(R); title('R的结构');

A - Q*R

清除所有

4.5. Cholesky分解

如果A是一个共轭转置矩阵（即A=A'），那么我们知道所有的特征值都是实数。如果此外，所有的特征值都大于零，那么对于所有的向量x，x'*A*x > 0，我们称A是正定的。 在这种情况下，可以进行Cholesky分解，即A = R'*R，其中R是上三角矩阵。 这相当于写成A = L*L'，其中L是下三角矩阵。

首先，我们使用以下正定矩阵。
Ndim=10;
Afull=zeros(Ndim,Ndim);
for i=1:Ndim在内部点上求和
Afull(i,i) = 2;
if(i>1)

```
Afull(i,i-1) = -1;
end
if(i<Ndim)
Afull(i,i+1) = -1;
end
end

Rfull = chol(Afull);
D = Afull - Rfull'*Rfull; eig(D)

figure;
subplot(1,2,1); spy(Afull); title('Afull的结构');
subplot(1,2,2); spy(Rfull); title('Rfull的结构');
```

对于稀疏矩阵，我们可以进行不完全的Cholesky分解，得到一个不会丢失稀疏性的近似分解，可以用作预处理器。 在这个特殊情况下，由于矩阵高度结构化，不完全分解与完全分解相同。**Asparse = sparse(Afull);Rsparse = cholinc(Asparse,'0');D2 = Asparse - Rsparse'*Rsparse; eig(D2)**

```
figure;
subplot(1,2,1); spy(Asparse); title('Asparse的结构');
subplot(1,2,2); spy(Rsparse); title('Rsparse的结构');

清除所有
```

4.6. 奇异值分解

特征值和特征向量仅对方阵定义。 将特征值的概念推广到非方阵通常很有用。 矩阵A的奇异值分解(SVD)定义为A = U*D*V'，其中D是包含奇异值的(m x n)对角矩阵，U是包含右特征向量的(m x m)矩阵，V'是左特征向量的转置共轭(n x n)矩阵。

在MATLAB中，使用"svd"进行奇异值分解

```
A = [1 2 3 4; 11 12 13 14; 21 22 23 24];
[U,D,V] = svd(A);
D, U, V
```

U*D*V' 证明分解有效

```
清除所有
```

MATLAB 教程

第5章. 文件输入/输出
5.1. 保存/读取二进制文件和调用操作系统

在使用MATLAB时，无论是运行m文件还是进行交互计算，都有一个主内存结构，MATLAB用它来跟踪所有变量的值。 可以将这个内存空间以二进制格式写入文件，以便将计算结果存储起来以供以后使用。当你需要中断MATLAB会话时，这通常很有用。 以下命令演示了如何使用这个存储选项来创建二进制.mat文件。

首先，让我们定义一些要保存的变量。
```
num_pts =10;
Afull=zeros(num_pts,num_pts);
Afull(1,1) = 1;
Afull(num_pts,num_pts) = 1;
对于内部点i=2:(num_pts-1)求和
Afull(i,i) = 2;
Afull(i,i-1) = -1;
Afull(i,i+1) = -1;
end
b = linspace(0,1,num_pts)';
x = Afull  ;
```

whos;显示内存内容

"save"命令将内存中的数据保存到指定的二进制文件中。
```
save mem_store1.mat;
```

```
clear all;
```
whos;内存中没有存储变量

ls *.mat显示目录中的所有.mat文件

"load"命令将存储在指定二进制文件中的数据加载到内存中。
```
load mem_store1.mat;
```
whos;我们可以看到数据已经再次加载

如果我们想要删除这个文件，可以使用"delete"命令。
```
delete mem_store1.mat;
ls *.mat
```

在上面的命令中，我使用了路径名来指定目录。 我们可以使用"pwd"命令查看当前默认目录。

pwd显示当前目录

我们可以使用"cd"命令切换到另一个目录。
cd ..向上移动一个目录
pwd
ls列出目录中的文件
cd MATLAB_tutorial;目录名称可能因人而异
pwd; ls

我们还可以使用"save"命令将选定的变量保存到二进制文件中。
save mem_store2.mat Afull;

清除所有
查看变量

加载mem_store2.mat
查看变量

删除mem_store2.mat

清除所有

5.2. 与ASCII文件的数据输入/输出

首先，让我们定义一些要保存的变量。
num_pts =10;

Afull=zeros(num_pts,num_pts);
Afull(1,1) = 1;
Afull(num_pts,num_pts) = 1;
对于内部点i=2:(num_pts-1)求和
Afull(i,i) = 2;
Afull(i,i-1) = -1;
Afull(i,i+1) = -1;
end

b = linspace(0,1,num_pts)';
x = Afull ;

查看变量；显示内存内容

现在，让我们将Afull的内容写入一个可以读取的文件中。

一种选项是使用带有选项-ascii的"save"命令，它使用ASCII格式写入文件。

save store1.dat Afull -ascii;
查看 **store1.dat** 文件的内容

我们也可以以这种方式加载文件。 ASCII文件filename.dat的内容存储在MATLAB变量filename中。 这是从实验或其他程序导入数据到MATLAB的好方法。加载 store**1**.dat 文件；

如果我们添加选项-double，数据将以更高精度的两倍数字打印出来。

删除 store1.dat 文件；
save store1.dat Afull -ascii -double;
查看 store1.dat 文件

我们可以使用这个命令来处理多个变量，但我们可以看到没有添加空格。
delete store1.dat;
save store1.dat Afull b x -ascii;
type store1.dat查看文件内容
delete store1.dat删除文件

MATLAB还允许使用类似于C语言的命令来进行更复杂的格式化文件的输入/输出。

首先，我们列出目录中的所有文件。
ls

接下来，我们创建输出文件并为其分配一个标签
使用具有以下语法的"fopen"命令
FID = fopen(FILENAME,PERMISSION)
其中PERMISSION通常是以下之一：
'r' = 只读
'w' = 写入（如果需要则创建）
'a' = 追加（如果需要则创建）
'r+' = 读取和写入（不创建）
'w+' = 创建用于读取和写入
'a+' = 读取和追加（如果需要则创建）
FID_out = fopen('test_io.dat','w');
ls

现在，我们使用"fprintf"命令将b向量作为列向量打印到输出文件中。
在格式字符串中，'
'表示换行，10.5f指定了小数点后5位数字和总字段宽度为10的浮点数输出。
对于i=1:length(b)
fprintf(FID_out,'10.5f ',b(i));end

现在我们关闭文件并显示结果。
fclose(FID_out);
disp('test_io.dat的内容：');
type test_io.dat;

为了避免使用for循环，也可以加载MATLAB的"fprintf"。

FID_out = fopen('test_io.dat','a');fprin
tf(FID_out,'');fprintf(FID_out,'10.5
f ',x);fclose(FID_out);

disp('test_io.dat的内容：');
type test_io.dat;

我们还可以使用"fprintf"来打印矩阵。
C = [1 2 3; 4 5 6; 7 8 9; 10 11 12];FID_out =
fopen('test_io.dat','a');fprintf(FID_out,'');
for i = 1:size(C,1)

fprintf(FID_out,'5.0f 5.0f 5.0f ',C(i,:));endfclos
e(FID_out);

disp('test_io.dat的内容：');
type test_io.dat;

我们可以使用"fscanf"从格式化文件中读取数据，它的工作
方式类似于"fprintf"。

首先，我们以只读方式打开文件。
FID_in = fopen('test_io.dat');

现在我们将向量b读入变量b_new。 首先，我们为向量分配空间，然后逐个读取值。**b_new = linspace(0, 0,num_pts)';for i=1:num_pts**

b_new(i) = fscanf(FID_in,'f',1);
end
b_new

现在使用MATLAB中可能的重载将x读入x_new。
x_new = linspace(0,0,num_pts)';
x_new = fscanf(FID_in,'f',num_pts);
x_new

最后，我们将矩阵C读入C_new。
C_new = zeros(4,3);
for i=1:size(C,1)
for j=1:size(C,2)
C_new(i,j) = fscanf(FID_in,'f',1);
end
end
C_new

fclose(FID_in);

清除所有

MATLAB 教程

第6章 编写和调用函数

在本章中，我们讨论如何使用多个源代码文件来构建程序的结构。 首先，介绍了MATLAB中代码文件的工作原理。 在编译语言（如FORTRAN、C或C++）中，代码可以存储在一个或多个源文件中，在编译时链接在一起形成一个可执行文件。 MATLAB作为一种解释性语言，以更加开放的方式处理多个源文件。 MATLAB代码被组织成扩展名为.m的ASCII文件（也称为m文件）。 MATLAB 6具有集成的文字处理和调试工具，是编辑m文件的首选模式，尽管也可以使用其他ASCII编辑器，如vi或emacs。

有两种不同类型的m文件。 最简单的一种是脚本文件，只是一组MATLAB命令。 当通过在交互提示符处键入其名称来执行脚本文件时，MATLAB会读取并执行m文件中的命令，就像手动输入它们一样。 这就好像将m文件的内容剪切并粘贴到MATLAB命令窗口中一样。 本节概述了使用这种类型的m文件的方法。

第二种m文件，在第6.2节中讨论，包含一个与m文件同名的单个函数。 这个m文件包含了一个独立的代码段，具有明确定义的输入/输出接口；也就是说，可以通过传递一个虚拟参数列表arg1，arg2，...来调用它。 它将返回输出值out1，out2，...。 函数m文件的第一行非注释行包含函数头，格式如下：**function [out1,out2,...] = filename(arg1,arg2,...);**

m文件以return命令结束，该命令将程序执行返回到调用函数的位置。 无论是在交互式命令提示符下还是在另一个m文件中，只要使用命令invoked with the command :[outvar**1**,outvar**2**,...] = filename**(**var**1**,var**2**,...**)**，函数代码就会被执行。

通过将输入映射到虚拟参数：arg1 = var1，arg2 = var2等，在函数体内，输出值被赋给变量out1，out2等。当遇到return时，out1，out2等的当前值 被映射到变量outvar1，outvar2等 在函数被调用的地方。 MATLAB允许在编写函数时使用可变长度的参数和输出变量列表。 例如，该函数也可以通过命令outvar1 = filename**(**var1,var2,...**)**来调用。

在这种情况下，只返回一个输出变量，该变量在退出时包含函数变量out1的值。 输入和输出参数可以是字符串、标量数值、向量、矩阵或更高级的数据结构。

为什么要使用函数？ 众所周知，在计算机科学课程中，将一个大程序分割成多个执行单个明确定义和注释的任务的过程，可以使程序更易于阅读、修改，并且更具抗错误性。 在MATLAB中，首先编写一个主程序文件，可以是脚本文件或者更好的是返回一个整数的函数m文件（该整数可能表示程序成功返回1，不完整执行返回0，运行时错误返回负值），这是程序的入口点。 然后，该程序文件通过将其他m文件作为函数调用来调用代码。 但是，如果没有编译过程将所有源代码文件链接在一起，那么MATLAB在调用函数时如何知道在哪里查找函数呢？

MATLAB的程序内存包含一个搜索路径列表，可以使用命令path查看其内容，该列表存储了目录的名称，这些目录被告知包含函数m文件。 最初，路径列表仅包含保存内置MATLAB函数（如sin**()**，e**x**p**()**等）的目录。 如第6.2节所示，可以使用命令addpath将包含当前项目m文件的每个目录的名称添加到此列表中。 然后，当MATLAB代码解释器遇到一个函数，比如名称为filename的函数时，它从路径列表的顶部开始，逐个搜索每个目录中的filename.m文件。 当找到它时，它以相应的方式执行文件的代码。

如上所述。 因此，m文件的名称和函数的名称必须一致；实际上，只有文件名才起作用。

6.1. 编写和运行m文件

虽然MATLAB可以在命令行中交互运行，但你可以通过编写一个包含你想要MATLAB按照顺序执行的命令的文本文件来编写一个MATLAB程序。 一个包含MATLAB程序的文本文件的标准后缀是.m。在MATLAB命令窗口中，选择下拉菜单文件 -> 新建 -> M文件打开集成的MATLAB文本编辑器以编写m文件。 这个工具与文字处理器非常相似，因此在这里不详细解释编写和保存m文件的使用方法。

作为示例，将此部分作为一个名为"MATLAB_tutorial_c6s1.m"的文件，该文件只包含以下可执行命令。

```
file_name = 'MATLAB_tutorial_c6s1.m';
disp(['开始 ' file_name ]);
j = 5;
for i=1:5
j = j - 1;
disp([int2str(i) ' ' int2str(j)]);
end
disp(['完成 ' file_name]);
```

我们可以通过输入文件名来从提示符运行这个m文件
>> MATLAB_tutorial_c6s1

如果我们现在输入"whos"，我们会看到在程序结束后仍然保留在内存中的变量。 这是因为我们将m文件编写为脚本文件，在文件中简单地收集了几个命令，然后在运行脚本时逐个执行它们，就像我们在交互式会话窗口中输入它们一样。 m文件的更常见用法是将一系列命令隔离在一个独立的函数中，如下一节所述。

6.2. 使用函数进行结构化编程

非结构化编程方法

文件 unstructured.m

在本节中，让我们演示使用子程序编写结构化、组织良好的程序。 我们为一个特别简单和熟悉的情况进行了演示，即简单的一维PDE问题，在这个m文件中，我们将所有命令组合到一个单独的文件中解决问题。 这种"非结构化"方法对于非常小的程序来说是可以的，但是随着程序规模的增长，很快就会变得混乱不堪。

```
num_pts = 100;网格点的数量
x = 1:num_pts;x值的网格
```

现在我们设置矩阵的值，用于离散化带有Dirichlet边界条件的PDE。

```
nzA = 3*(num_pts-2) + 2;非零元素的数量
A = spalloc(num_pts,num_pts,nzA);分配内存
```

设置值

```
A(1,1) = 1;
A(num_pts,num_pts) = 1;
```

对于 i=2:(num_pts-1)
A(i,i) = 2;
A(i,i-1) = -1;
A(i,i+1) = -1;
end

接下来，我们设置每个边界处的函数值。
BC1 = -10;x(1)处的f的值;
BC2 = 10;x(num_pts)处的f的值;

现在我们创建右手边问题的向量。
b_RHS = linspace(0,0,num_pts)';创建零列向量
b_RHS(1) = BC1;
b_RHS(num_pts) = BC2;
b_RHS(2:(num_pts-1)) = 0.05;对于内部点，b_RHS是源项

现在，我们调用标准的MATLAB求解器。
f = A _RHS;

然后，我们绘制结果的图表。
figure; plot(x,f);
title('使用FD-CDS方法（稀疏矩阵）求解的PDE');
xlabel('x'); ylabel('f(x)');

虽然将所有命令放在一起的方法对于这个小程序有效，但对于大型程序来说非常笨拙。

清除所有

结构化编程方法

文件structured.m

注意：在运行此文件之前，其他包含子程序的M文件必须已经存在。首先，我们定义点的数量**num_pts = 100;**

现在，我们创建一个包含网格点的向量。
x = 1:num_pts;

在MATLAB中，每个函数都存储在一个同名的单独的M文件中。 当你在交互式会话提示符或其他脚本或函数M文件中调用该函数时，MATLAB会在一个包含函数的目录列表中搜索，直到找到一个具有适当名称的M文件。 然后，它执行该M文件中包含的MATLAB代码。 当我们编写包含函数的M文件时，在使用它们之前，我们必须告诉MATLAB它们的位置；也就是说，我们必须将它们所在的目录名称添加到搜索路径中。

我们可以使用命令"path"来检查当前搜索路径的内容。
路径

命令"pwd"返回当前目录。
当前目录

我们使用命令"addpath"将包含我们子程序的目录添加到搜索列表中。
我们可以使用"rmpath"命令从路径中移除一个目录。
addpath(pwd);
path

下面的函数计算矩阵A。 函数调用的语法如下：[out1,out2,...] = func_name(in1,in2,...)，其中输入参数是变量in1,in2,... 函数的输出存储在out1,out2,...中。 在我们的情况下，输入是矩阵A的维度num_pts，输出变量是A和iflag，一个整数用于告知我们代码是否成功执行。

[A,iflag] = c6s2_get_A(num_pts);
if(iflag ~= 1) then error
disp(['c6s2_get_A返回错误标志：' int2str(iflag)]);
end

我们还可以从下面的代码中看到，在名为i的函数中存在一个局部变量
对于调用点的i值没有任何改变。
i = 1234;
[A,iflag] = c6s2_get_A(num_pts);
i

接下来，我们要求用户在边界处输入函数值。
BC1 = input('输入x = 1处的函数值：');
BC2 = input('输入x = num_pts处的函数值：');
source = input('输入源项的值：');

现在我们调用另一个子程序来计算右手边的向量。
[b_RHS,iflag] = c6s2_get_b_RHS(num_pts,BC1,BC2,source);

现在我们解决这个系统。
f = A _RHS;

然后，我们绘制输出的图形。
figure; plot(x,f);
phrase1 = ['带有源项的PDE解，source = ' num2str(source)];
phrase1 = [phrase1 ', BC1 = ' num2str(BC1)];
phrase1 = [phrase1 ', BC2 = ' num2str(BC2)];
title(phrase1); xlabel('x'); ylabel('f(x)');

然后，为了清理，我们清除内存

清除所有

文件 c6s2_get_A.m

m文件的第一行声明了子程序的名称和输入/输出结构，使用"function"命令。

function [A,iflag] = c6s2_get_A(Ndim);

iflag = 0;表示工作未完成

如果Ndim < 1，则会出现错误，因为我们不能有一个小于1的矩阵维度。

if(Ndim<1) 表示错误

A=-1;
iflag = -1;
我们将控制权返回到调用此子程序的m文件，而不执行其余的代码。

return;
end

首先，我们使用稀疏矩阵格式声明A。
nzA = 3*(Ndim-2) + 2;非零元素的数量
A = spalloc(Ndim,Ndim,nzA);分配内存
A(1,1) = 1;
A(Ndim,Ndim) = 1;
for i=2:(Ndim-1)
A(i,i) = 2;
A(i,i-1) = -1;
A(i,i+1) = -1;
end

iflag = 1;表示工作完成且成功

return;返回控制权给调用此程序的m文件

文件c6s2_get_b_RHS.m
function [b_RHS,iflag] = c6s2_get_b_RHS(num_pts,BC1,BC2,source);
iflag = 0; 表示工作未完成

if(num_pts < 3)点数不足
iflag = -1;
b_RHS = -1;
return;
end

我们为b_RHS分配空间并初始化为零。
b_RHS = linspace(0,0,num_pts)';

现在，我们从边界条件中指定第一个和最后一个分量。
b_RHS(1) = BC1;
b_RHS(num_pts) = BC2;

接下来，我们指定内部点。
for i=2:(num_pts-1)
b_RHS(i) = source;
end

iflag=1;表示成功完成

返回;

6.3. 内联函数

有时候，我们不想费力写一个单独的m文件来定义一个函数。 在这些情况下，我们可以定义一个内联函数。 假设我们想定义函数f1(x) = 2*x + 3*x^2

我们可以使用以下方式定义这个函数
f1 = inline('2*x + 3*x^2');

然后，我们可以直接调用这个函数
f1(1), f1(23)

我们也可以使用向量和矩阵作为输入来定义函数。
invariant2 = inline('(trace(A)*trace(A) - trace(A*A))/2');
A = rand(3);
invariant2(A)

我们可以通过输入函数名来检查函数的定义
invariant2

虽然这很方便，但内联函数的执行速度相对较慢。

清除所有变量;
尝试
invariant2
catch
disp('我们可以看到内联函数也被清除了');
end

6.4. 函数作为函数参数

下面列出的函数trig_func_1，返回给定a、b和
x值的f(x) = a*sin(x) + b*cos(x)的值。

下面列出的函数plot_trig_1，在0到2*pi的区间上绘制一个函数。

以下代码要求用户输入a和b的值，然后使用plot_trig绘制
trig_func_1。
将函数名作为参数包含在列表中。
disp('绘制 a*sin(x) + b*cos(x) ...');
a = input('输入 a： ');
b = input('输入 b： ');
func_name = 'trig_func_1';

确保当前路径在路径中
addpath(pwd)

plot_trig_1(func_name,a,b);

清除所有

文件 trig_func_1.m
function f_val = trig_func_1(x,a,b);
f_val = a*sin(x) + b*cos(x);

返回;

文件 plot_trig_1.m
function iflag = plot_trig_1(func_name,a,b);
iflag = 0;表示未完成

首先，从0到2*pi创建一个x向量
```
num_pts = 100;
x = linspace(0,2*pi,num_pts);
```

接下来，创建一个函数值的向量。 我们间接地评估参数函数
使用"feval"命令。
```
f = linspace(0,0,num_pts);
for i=1:num_pts
f(i) = feval(func_name,x(i),a,b);
end
```

然后，我们绘制图形。
```
figure;
plot(x,f);
xlabel('角度（弧度）');
ylabel('函数值');

返回;
```

MATLAB 教程

第7章。 数据结构和输入断言
7.1. 用户定义的数据结构

向量和矩阵不是MATLAB提供的唯一将数据组合成单个实体的方式。 用户定义的数据结构也可用，使程序员能够创建混合数字、字符串和数组的变量类型。 例如，让我们创建一个包含单个学生信息的数据结构。

我们将存储姓名、状态（年级和系别）、作业和考试成绩以及最终课程成绩。

首先，我们可以定义一个NameData结构来存储姓名。 在这里，"."运算符用于在Structure.Field的情况下告诉MATLAB访问结构"Structure"中名为"Field"的字段。

```
NameData.First = 'John';
NameData.MI = 'J';
NameData.Last = 'Doe';
```

现在我们创建一个带有姓名字段的StudentData结构。
```
StudentData.Name = NameData;
```

现在我们初始化结构的其余部分。
```
StudentData.Status = '化工研究生1年级';
StudentData.HW = 10;
StudentData.Exam = linspace(100,100,3);
```

现在我们可以查看结构StudentData的内容
```
StudentData
StudentData.Name
StudentData.Exam
```

我们可以对结构的元素进行操作。
```
StudentData.Exam(3) = 0;
StudentData.Exam
StudentData.Name.First = '简';
StudentData.Name
```

我们还可以创建结构数组
```
num_students = 5;
for i=1:num_students
ClassData(i) = StudentData;
end
ClassData
ClassData(2)
```

结构体可以像标量、向量和矩阵一样作为参数传递给函数。 在这种情况下，我们使用下面列出的函数pass_or_fail。

```
message = pass_or_fail(ClassData(2));
message
```

文件pass_or_fail.m

```
function message = pass_or_fail(StudentData)
Exam_avg = mean(StudentData.Exam);

if(Exam_avg >= 70)
message = '你通过了！';
else
message = '你失败了！';
end

返回;
```

7.2. 输入断言例程

良好的编程风格要求进行防御性编程，即在运行时错误导致程序执行停止或崩溃之前，预测和检测可能的错误。 这样可以将当前数据保存到磁盘或采取纠正措施以避免灾难性故障。 一个常见的错误来源可以通过确保每个子程序通过其参数列表接收到的数据具有适当的类型来避免，例如，参数1应该是一个实数、正数、标量整数，参数2应该是一个实数、非负的长度为N的列向量。以下的m文件对自动化这个检查过程很有用，并提供了一个标量输入函数，以便从键盘上输入数据。

assert_scalar.m

```
function [iflag_assert,message] = assert_scalar( ...
i_error,value,name,func_name, ...
check_real,check_sign,check_int,i_error);
```

这个m文件包含了逻辑检查，用于断言输入值是一种标量数值类型。
该函数接收变量的值和名称，断言的函数名称，以及四个整数标志，其用法如下：

i_error：控制测试失败时的操作
如果i_error非零，则使用error()
MATLAB命令用于停止执行，否则只返回适当的负数。
如果i_error > 1，则在调用error()之前将当前状态转储到dump_error.mat。

check_real：检查输入数字是否为实数。请参阅函数
标题后的表格以获取这些情况标志的设置值
check_real = i_real（确保输入为实数）
check_real = i_imag（确保输入为纯虚数）
check_real的任何其他值（尤其是0）都不会进行检查

```
check_real
i_real = 1;
i_imag = -1;
```

check_sign：检查输入值的符号，请参考函数头后的表格设置这些情况标志的值

check_sign = i_pos（确保输入为正数）
check_sign = i_nonneg（确保输入为非负数）
check_sign = i_neg（确保输入为负数）
check_sign = i_nonpos（确保输入为非正数）
check_sign = i_nonzero（确保输入为非零数）
check_sign = i_zero（确保输入为零）

check_sign的任何其他值（尤其是0）都不进行检查

check_sign
i_pos = 1;
i_nonneg = 2;
i_neg = -1;
i_nonpos = -2;
i_nonzero = 3;
i_zero = -3;

check_int：检查输入是否为整数
如果为1，则检查输入是否为整数
其他任何值，则不进行检查

肯尼斯・比尔斯
麻省理工学院
化学工程系

7/2/2001

截至2001年7月21日的版本

```
function [iflag_assert,message] = assert_scalar( ...
i_error,value,name,func_name, ...
check_real,check_sign,check_int);

iflag_assert = 0;
message = 'false';
```

首先，设置检查整数标志的情况值。

```
check_real
i_real = 1;
i_imag = -1;
check_sign
i_pos = 1;
i_nonneg = 2;
i_neg = -1;
i_nonpos = -2;
i_nonzero = 3;
i_zero = -3;
```

检查输入是否为数字而不是字符串。

```
if(~isnumeric(value))
message = [ func_name, ': ', ...
name, ' is not numeric'];
iflag_assert = -1;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end
```

检查是否为标量。
```
if(max(size(value)) ~= 1)
message = [ func_name, ': ', ...
name, ' is not scalar'];
iflag_assert = -2;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end
```

然后，检查是否为实数。
```
switch check_real;

case {i_real}
if(~isreal(value))
message = [ func_name, ': ', ...
name, ' 不是实数'];
iflag_assert = -3;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end

case {i_imag}
if(real(value))
message = [ func_name, ': ', ...
name, ' 不是虚数'];
iflag_assert = -3;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end
end
```

接下来，检查符号。
```
switch check_sign;

case {i_pos}
if(value <= 0)
message = [ func_name, ': ', ...
name, ' 不是正数'];
iflag_assert = -4;
```

```
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end

情况 {i_nonneg}
如果(value < 0)
message = [ func_name, ': ', ...
name, ' 不是非负数'];
iflag_assert = -4;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end

情况 {i_neg}
如果(value >= 0)
message = [ func_name, ': ', ...
name, ' 不是负数'];
iflag_assert = -4;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end

情况 {i_nonpos}
如果(value > 0)
message = [ func_name, ': ', ...
name, ' 不是非正数'];
iflag_assert = -4;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end

情况 {i_nonzero}
如果(value == 0)
message = [ func_name, ': ', ...
```

```
name, ' 不是非零的'];
iflag_assert = -4;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end

情况 {i_zero}
如果(value ~= 0)
message = [ func_name, ': ', ...
name, ' 不是零'];
iflag_assert = -4;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end
end
```

最后，检查确保它是一个整数。

```
如果(check_int == 1)如果(round(value
) ~= value)message = [ func_name, ': '
, ...name, ' 不是整数'];iflag_assert = -5
;if(i_error ~= 0)if(i_error > 1)save dum
p_error.mat;enderror(message);elser
eturn;endend
```

设置标志以成功通过所有检查

```
iflag_assert = 1;
message = 'true';

返回;
```

assert_vector.m

```
function [iflag_assert, message] = ...
assert_vector( ...
i_error,value,name,func_name,num_dim, ...
```

check_real,check_sign,check_int,check_column);

这个m文件包含逻辑检查，以确保输入值是给定类型的向量。
该函数接收变量的值和名称，以及调用该函数的函数的名称

断言，向量应该具有的维度，以及具有以下用途的五个整数标志：

i_error：控制测试失败时的操作如果i_error非零
，则使用error()函数停止执行MATLAB命令，否
则只返回适当的负数。

如果i_error大于1，在调用error()函数之前创建
文件dump_error.mat

check_real：检查输入是否为实数请参考函数头后的表格
以获取这些情况标志的设置值

check_real = i_real (确保输入是实数)check_real = i_i
mag (确保输入是纯虚数)

check_real的任何其他值（尤其是0）都不
会进行检查

check_real
i_real = 1;
i_imag = -1;

check_sign：检查输入的符号，请参考函数头后的表格以获取这些情况标志的设置值

check_sign = i_pos (确保输入为正数)check_sign = i_nonn
eg (确保输入为非负数)check_sign = i_neg (确保输入为负数)chec
k_sign = i_nonpos (确保输入为非正数)check_sign = i_nonz
ero (确保输入为非零数)check_sign = i_zero (确保输入为零)che
ck_sign的任何其他值（尤其是0）都不会进行检查

check_sign
i_pos = 1;
i_nonneg = 2;
i_neg = -1;
i_nonpos = -2;
i_nonzero = 3;
i_zero = -3;

check_int：检查输入是否为整数
如果为1，则检查输入是否为整数
其他任何值，则不进行检查

check_column：检查输入是否为列向量check_column = i_column (
确保输入为列向量)

check_column = i_row (确保输入为行向量)
任何其他值都不进行检查

check_column
i_column = 1;

i_row = -1;

如果维度num_dim设置为零，则不检查向量的维度。

肯尼斯・比尔斯
麻省理工学院
化学工程系

7/2/2001

截至2001年7月21日的版本

function [iflag_assert,message] = ...
assert_vector(...
i_error,value,name,func_name,num_dim, ...
check_real,check_sign,check_int,check_column);

首先，设置检查整数标志的情况值。
check_real
i_real = 1;
i_imag = -1;
check_sign
i_pos = 1;
i_nonneg = 2;
i_neg = -1;
i_nonpos = -2;
i_nonzero = 3;
i_zero = -3;
check_column
i_column = 1;
i_row = -1;

iflag_assert = 0;
message = 'false';

检查输入是否为数字而不是字符串。
if(~isnumeric(value))
message = [func_name, ': ', ...
name, '不是数字'];
iflag_assert = -1;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end

检查是否是正确长度的向量。
num_rows = size(value,1);
num_columns = size(value,2);
如果是多维数组
if(length(size(value)) > 2)
message = [func_name, ': ', ...
name, '有太多的下标'];

```
iflag_assert = -2;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end
```

如果行数和列数都不等于1，则值为矩阵而不是向量。

```
如果（并且（（num_rows ~= 1），（num_columns ~= 1））
message = [ func_name, ': ', ...
name, '不是向量'];
iflag_assert = -2;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end
```

如果向量的维度不正确

```
if(num_dim ~= 0)
if(length(value) ~= num_dim)
message = [ func_name, ': ', ...
name, '长度不正确'];
iflag_assert = -2;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end
end
```

检查向量是否为正确类型（例如列向量）

```
switch check_column;

情况 {i_column}
```

检查确保它是一个列向量

```
如果（num_columns > 1）
message = [ func_name, ': ', ...
name, '不是一个列向量'];
iflag_assert = -2;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
```

```
else
return;
end
end

情况 {i_row}
如果（num_rows > 1）
message = [ func_name, ': ', ...
name, '不是一个行向量'];
iflag_assert = -2;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end
end
```

然后，检查所有元素是否具有正确的复数类型。

```
switch check_real;

情况 {i_real}
```

如果值的任何元素不是实数

```
如果（任何（~isreal（value））））
message = [ func_name, ': ', ...
name, '不是实数'];
iflag_assert = -3;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end

情况 {i_imag}
```

如果value的任何元素不是纯虚数

```
如果(any(real(value)))
message = [ func_name, ': ', ...
name, ' 不是虚数'];
iflag_assert = -3;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end
end
```

接下来，检查符号。
switch check_sign;

情况 {i_pos}
如果value的任何元素不是正数
如果(any(value <= 0))
message = [func_name, ': ', ...
name, ' 不是正数'];
iflag_assert = -4;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end

情况 {i_nonneg}
如果value的任何元素是负数
如果(any(value < 0))
message = [func_name, ': ', ...
name, ' 不是非负数'];
iflag_assert = -4;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end

情况 {i_neg}
如果value的任何元素不是负数
如果(any(value >= 0))
message = [func_name, ': ', ...
name, ' 不是负数'];
iflag_assert = -4;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end

情况 {i_nonpos}
如果value的任何元素是正数
如果(any(value > 0))
message = [func_name, ': ', ...
name, ' 不是非正数'];
iflag_assert = -4;
if(i_error ~= 0)

```
如果(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end

情况 {i_nonzero}
如果value的任何元素是零
如果(any(value == 0))
message = [ func_name, ': ', ...
name, ' 不是非零'];
iflag_assert = -4;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end

情况 {i_zero}
如果value的任何元素非零
如果(any(value ~= 0))
message = [ func_name, ': ', ...
name, ' 不为零'];
iflag_assert = -4;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end
end

最后，检查确保它是一个整数。
如果(check_int == 1)如果(any(round(val
ue) ~= value))message = [ func_name,
 ': ', ...name, ' 不是整数'];iflag_assert = -
5;if(i_error ~= 0)if(i_error > 1)save du
mp_error.mat;enderror(message);els
ereturn;endend
```

设置标志以成功通过所有检查

iflag_assert = 1;
message = 'true';

返回;

assert_matrix.m

function [iflag_assert,message] = assert_matrix(...
i_error,value,name,func_name, ...
num_rows,num_columns, ...
check_real,check_sign,check_int);

这个m文件包含了逻辑检查，用于断言输入值是给定类型的矩阵。
该函数传递了变量的值和名称，以及进行断言的函数的名称，矩阵应该具有的维度，以及四个整数标志的用法如下：

i_error：控制测试失败时的操作
如果i_error非零，则使用error()
MATLAB命令用于停止执行，否则只返回适当的负数。
如果i_error大于1，在调用error()函数之前创
建文件dump_error.mat

check_real：检查输入是否为实数，请参阅函数头后的表格以获取这些情况标志的设置值

check_real = i_real（确保输入为实数）check_real = i_
imag（确保输入为纯虚数）

check_real的任何其他值（尤其是0）都不
会进行检查

check_real
i_real = 1;
i_imag = -1;

check_sign：检查输入的符号，请参阅函数头后
的表格以获取这些情况标志的设置值

check_sign = i_pos (确保输入为正数)check_sign = i_nonn
eg (确保输入为非负数)check_sign = i_neg (确保输入为负数)chec
k_sign = i_nonpos (确保输入为非正数)check_sign = i_nonz
ero (确保输入为非零数)check_sign = i_zero (确保输入为零)che
ck_sign的任何其他值（尤其是0）都不会进行检查

check_sign
i_pos = 1;
i_nonneg = 2;
i_neg = -1;
i_nonpos = -2;
i_nonzero = 3;
i_zero = -3;

check_int：检查输入值是否为整数

如果等于1，则检查输入是否为整数
其他任何值，则不进行检查

如果维度num_rows或num_columns
设置为零，则不检查矩阵的该维度。

肯尼斯·比尔斯
麻省理工学院
化学工程系

7/2/2001

截至2001年7月21日的版本

```
function [iflag_assert,message] = assert_matrix( ...
i_error,value,name,func_name, ...
num_rows,num_columns, ...
check_real,check_sign,check_int);
```

首先，设置检查整数标志的情况值。
check_real

```
i_real = 1;
i_imag = -1;
check_sign
i_pos = 1;
i_nonneg = 2;
i_neg = -1;
i_nonpos = -2;
i_nonzero = 3;
i_zero = -3;

iflag_assert = 0;
message = 'false';
```

检查输入是否为数字而不是字符串。

```
if(~isnumeric(value))
message = [ func_name, ': ', ...
name, ' is not numeric'];
iflag_assert = -1;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end
```

检查是否为正确长度的矩阵。
如果是多维数组

```
如果（length（size（value））> 2）
message = [ func_name, ': ', ...
name，'有太多的下标'];
iflag_assert = -2;
if（i_error～= 0）
if(i_error > 1)
```

```
save dump_error.mat;
end
error(message);
else
return;
end
end
```

检查值的行数是否正确

```
if(num_rows ~= 0)
if(size(value,1) ~= num_rows)
message = [ func_name, ': ', ...
name, '的行数不正确'];
iflag_assert = -2;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end
end
```

检查值的列数是否正确

```
if(num_columns ~= 0)
if(size(value,2) ~= num_columns)
message = [ func_name, ': ', ...
name, '的列数不正确'];
iflag_assert = -2;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end
end
```

然后，检查所有元素是否具有正确的复数类型。

```
switch check_real;

情况 {i_real}
```

如果值的任何元素不是实数

```
if(any(~isreal(value)))
message = [ func_name, ': ', ...
name, '不是实数'];
iflag_assert = -3;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
```

结束
结束

情况 {i_imag}
如果value的任何元素不是纯虚数
如果(any(real(value)))
message = [func_name, ': ', ...
name, ' 不是虚数'];
iflag_assert = -3;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end
end

接下来，检查符号。
switch check_sign;

情况 {i_pos}
如果value的任何元素不是正数
如果(any(value <= 0))
message = [func_name, ': ', ...
name, ' 不是正数'];
iflag_assert = -4;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end

情况 {i_nonneg}
如果value的任何元素是负数
如果(any(value < 0))
message = [func_name, ': ', ...
name, ' 不是非负数'];
iflag_assert = -4;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end

情况 {i_neg}
如果任何一个值不是负数

```
如果(任何一个值大于等于0)
消息 = [函数名, ': ', ...
名称, '不是负数'];
iflag_assert = -4;
如果(i_error ~= 0)
如果(i_error > 1)
保存 dump_error.mat;
结束
错误(消息);
否则
返回;
结束
结束

情况 {i_nonpos}
如果任何一个值是正数
如果(任何一个值大于0)
消息 = [函数名, ': ', ...
名称, '不是非正数'];
iflag_assert = -4;
如果(i_error ~= 0)
如果(i_error > 1)
保存 dump_error.mat;
结束
错误(消息);
否则
返回;
结束
结束

情况 {i_nonzero}
如果value的任何元素是零
如果(any(value == 0))
message = [ func_name, ': ', ...
name, ' 不是非零'];
iflag_assert = -4;
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end

情况 {i_zero}
如果任何一个值不是零
如果(任何一个值不等于0)
消息 = [函数名, ': ', ...
名称, '不是零'];
iflag_assert = -4;
如果(i_error ~= 0)
如果(i_error > 1)
保存 dump_error.mat;
结束
错误(消息);
否则
```

```
return;
end
end
end
```

最后，检查确保它是一个整数。

```
如果(check_int == 1)如果(any(round(val
ue) ~= value))message = [ func_name,
 ': ', ...name, ' 不是整数'];iflag_assert = -
5;if(i_error ~= 0)if(i_error > 1)save du
mp_error.mat;enderror(message);els
ereturn;endend
```

设置标志以成功通过所有检查

```
iflag_assert = 1;
message = 'true';

返回;
```

assert_structure.m

```
function [iflag_assert,message] = assert_structure(...
i_error,Struct,struct_name,func_name,StructType);
```

这个MATLAB m文件对数据结构进行断言。 它使用assert_scalar、assert_vector和assert_matrix来检查字段。

输入：=
======
i_error控制测试失败时的操作
如果i_error非零，则使用error()
MATLAB命令用于停止执行，否则只返回适当的负数。
如果i_error > 1，则在调用error()之前将当前状态转储到dump_error.mat。
Struct这是要检查的结构
struct_name结构的名称
func_name进行断言的函数的名称
StructType这是一个包含每个字段的类型数据的结构。
.num_fields是字段的总数
然后，对于i = 1,2, ..., StructType.num_fields，我们有：
.field(i).name字段的名称
.field(i).is_numeric如果非零，则字段是数值型的
.field(i).num_rows字段中的行数
.field(i).num_columns字段中的列数
.field(i).check_real的值通过了check_real的断言
.field(i).check_sign的值通过了check_sign的断言
.field(i).check_int的值通过了check_int的断言

输出：===
====
iflag_assert是一个整数标志，用于告知断言的结果消息是一个传递描述结果的消息

肯尼斯・比尔斯
麻省理工学院
化学工程系

7/2/2001

截至2001年7月25日的版本

```
function [iflag_assert,message] = assert_structure(...
i_error,Struct,struct_name,func_name,StructType);

iflag_assert = 0;
message = 'false';

首先，检查Struct是否为一个结构体
if(~isstruct(Struct))
iflag_assert = -1;
message = [func_name, ': ', struct_name, ...
'不是一个结构体'];
if(i_error ~= 0)
if(i_error > 1);
save dump_error.mat;
end
error(message);
else
return;
end
end

现在，对于每个字段，执行所需的断言。
for ifield = 1:StructType.num_fields

将当前字段类型设置为快捷方式
FieldType = StructType.field(ifield);

检查它是否存在于Struct中
if(~isfield(Struct,FieldType.name))
iflag_assert = -2;
message = [func_name, ': ', struct_name, ...
' does not contain ', FieldType.name];
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end

提取字段的值
value = getfield(Struct,FieldType.name);
```

如果字段应该是数值类型
if(FieldType.is_numeric ~= 0)

检查字段是否为数值类型
if(~isnumeric(value))
iflag_assert = -3;
message = [func_name, ': ', ...
struct_name, '.', FieldType.name, ...
' 不是数值类型'];
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end
end

根据字段值的数组维度决定使用哪个断言语句如果num_rows和num_columns都设置为零，则不检查该字段的维度。

如果(FieldType.num_rows == 0)且(FieldTyp
e.num_columns == 0)成立

message = [func_name, ': ', struct_na
me,'.',FieldType.name, ...]
'未检查维度';
if(i_error ~= 0)
disp(message);
end

否则，进行维度检查，确保它是标量、向量或矩阵（即二维数组）。否则

检查它是否为多维数组**if(length(size(value)) > 2**
)iflag_assert = -4;message = [**f**unc_name**,**
': ', struct_name**,'.',F**ield**T**y**p**e**.**name**, ...**]

'是多维数组';
if(i_error ~= 0)
if(i_error > 1)
save dump_error.mat;
end
error(message);
else
return;
end

否则，如果是标量
否则如果(FieldType.num_rows == 1)且(FieldType
.num_columns == 1)成立assert_scalar(i_e
rror,value, [struct_name,'.',FieldType.na
me], ...

```
func_name,FieldType.check_real, ...
FieldType.check_sign,FieldType.check_int);

如果是列向量
elseif (and((FieldType.num_rows > 1), ...
(FieldType.num_columns == 1)))
维度=FieldType.num_rows;
check_column = 1;
assert_vector(i_error,value, ...
[struct_name，'。'，FieldType.name]，...
func_name,dim,FieldType.check_real, ...
FieldType.check_sign,FieldType.check_int, ...
check_column);

如果是行向量
elseif (and((FieldType.num_rows == 1), ...
(FieldType.num_columns > 1)))
维度=FieldType.num_columns;
check_column = -1;
assert_vector(i_error,value, ...
[struct_name，'。'，FieldType.name]，...
func_name,dim,FieldType.check_real, ...
FieldType.check_sign,FieldType.check_int, ...
check_column);

否则，是一个矩阵
else
assert_matrix(i_error,value, ...
[struct_name，'。'，FieldType.name]，...
func_name, ...
FieldType.num_rows,FieldType.num_columns, ...
FieldType.check_real,FieldType.check_sign, ...
FieldType.check_int);

end选择断言例程
end如果执行维度检查
end如果（FieldType.is_numeric！= 0）
end对字段的循环

设置成功断言的返回结果

iflag_assert = 1;
message = 'true';

返回;
```

get_input_scalar.m

function value = get_input_scalar(prompt, ...
check_real,check_sign,check_int);

这个MATLAB m文件从用户那里获取适当类型的输入标量值。 它一遍又一遍地要求输入，直到输入正确类型的值为止。

肯尼斯・比尔斯
麻省理工学院
化学工程系

7/2/2001

截至2001年7月25日的版本

```
function value = get_input_scalar(prompt, ...
check_real,check_sign,check_int);

func_name = 'get_input_scalar';
name = 'trial_value';

input_OK = 0;

while (input_OK ~= 1)
trial_value = input(prompt);
[iflag_assert,message] = ...
assert_scalar(0,trial_value, ...
name,func_name, ...
check_real,check_sign,check_int);
if(iflag_assert == 1)
input_OK = 1;
value = trial_value;
else
disp(message);
end
end

返回;
```

MATLAB 教程

第8章 MATLAB编译器

前几章讨论了在MATLAB环境中进行编程的内容。 已经注意到MATLAB是一种解释性语言，这意味着每个命令在执行过程中都会逐个转换为机器级指令。 虽然这使得可以在交互和批处理模式下编程，但在运行时转换命令所需的额外开销是不希望的。 此外，任何用MATLAB编写的程序只能在安装有MATLAB的计算机上运行，因此可移植性有限。 MATLAB包括一个可选的编译器，通过将m文件转换为C或C++代码，并可选择将此代码与其数学和图形库链接起来，生成一个独立的可执行文件，可以在任何具有兼容操作系统平台的计算机上运行，而无需解释开销。 在本节中，我们演示MATLAB编译器如何从第6.4节的简单示例中生成一个独立的可执行文件。请注意，包含主程序的程序已从以前的脚本文件版本重新编写，因为MATLAB编译器只能与函数m文件一起使用。 第一个文件是一个名为make_file.m的脚本文件，从交互提示符执行以进行编译；或者，可以手动输入命令mcc ...。

make_file.m

这个MATLAB脚本m文件调用编译器将MATLAB源代码文件转换为C，将目标文件与MATLAB图形库链接，然后生成一个独立的可执行文件。

肯尼斯・比尔斯
麻省理工学院
化学工程系

7/31/2001

mcc -B sgl ...
make_plot_trig ...
plot_trig_1 ...
trig_func_1 ...
get_input_scalar ...
assert_scalar

make_plot_trig.m (主程序文件)

make_plot_trig.m

这个MATLAB m文件绘制了一个一般函数
f(x) = a*sin(x) + b*cos(x)
对于用户选择的a和b的值。

肯尼斯・比尔斯
麻省理工学院
化学工程系

7/31/2001

function iflag_main = make_plot_trig();

iflag_main = 0;表示没有完成

```
disp('运行make_plot_trig ...');
disp(' ');
disp('这个程序在[0,2*pi]范围内绘制了一个图形');
disp('函数为：');
disp('f(x) = a*sin(x) + b*cos(x)');
disp('对于用户输入的实数a和b的值');
disp(' ');
```

以下代码要求用户输入a和b的值，然后使用plot_trig函数将trig_func_1绘制成图形，将函数名作为参数包含在列表中。

```
prompt = '输入a的值：';
check_real=1; check_sign=0; check_int=0;
a = get_input_scalar(prompt, ...
check_real,check_sign,check_int);

prompt = '输入b的值：';
check_real=1; check_sign=0; check_int=0;
b = get_input_scalar(prompt, ...
check_real,check_sign,check_int);
```

现在我们调用生成图形的程序。

```
func_name = 'trig_func_1';
plot_trig_1(func_name,a,b);
```

现在我们要求用户在退出程序之前按下一个键。

```
暂停

iflag_main = 1;

返回;
```

plot_trig_1.m

```
function iflag = plot_trig_1(func_name,a,b);

iflag = 0;
```

表示未完成

首先，从0到2*pi创建一个x向量

```
num_pts = 100;
x = linspace(0,2*pi,num_pts);
```

接下来，创建一个函数值的向量。 我们间接地评估参数函数
使用"feval"命令。

```
f = linspace(0,0,num_pts);
for i=1:num_pts
f(i) = feval(func_name,x(i),a,b);
end
```

然后，我们绘制图形。

```
figure;
plot(x,f);
xlabel('角度（弧度）');
ylabel('函数值');
```

```
return;
```

trig_func_1.m

```
function f_val = trig_func_1(x,a,b);
f_val = a*sin(x) + b*cos(x);

return;
```