同济大学计算数学教研室　编著

现代数值数学和计算

同济大学出版社

现代数值数学和计算

同济大学计算数学教研室　编著

同济大学出版社

内 容 提 要

本书以 MatLab 语言及编程为基础介绍常用的数值计算方法及有关的基础理论，主要内容包括科学计算与 MatLab、多项式插值与样条插值、函数逼近、数值积分与数值微分、线性方程组的直接解法、线性方程组的迭代解法、非线性方程求解、矩阵的特征值和特征向量的计算及常微分方程数值解等 9 章. 本书每章后面都有评注、习题，并配有计算机实习题，通过编程做题，大大提高学生解决实际问题的能力.

本书内容由浅入深、通俗易懂、易于教学、便于自学，可作为数学专业和工科各专业大学生、研究生的教材，也可供工程技术人员作为自学参考书.

图书在版编目(CIP)数据

现代数值数学和计算/同济大学计算数学教研室编著. —上海：同济大学出版社，2004.7

ISBN 7-5608-2832-9

Ⅰ. 现… Ⅱ. ① 陈… ② 同… Ⅲ. 数值计算—计算方法—高等学校—教材 Ⅳ. O241

中国版本图书馆 CIP 数据核字(2004)第 023265 号

现代数值数学和计算

同济大学计算数学教研室 编著

责任编辑 李炳钊 责任校对 郁 峰 封面设计 李志云

出 版
发 行 同济大学出版社

(上海四平路 1239 号 邮编 200092 电话 021-65985622)

经 销 全国各地新华书店

印 刷 江苏启东印刷厂印刷

开 本 787mm×960mm 1/16

印 张 18.25

字 数 370000

印 数 1—4500

版 次 2004 年 7 月第 1 版 2004 年 7 月第 1 次印刷

书 号 ISBN 7-5608-2832-9/O·253

定 价 21.00 元

前　言

数值计算是计算机科学与信息科学的重要内容. 大型科学计算与工程计算在国民经济发展中已占有越来越重要的地位. 1996 年美国能源部提出“加速战略计算创新计划”,其本质是将过去以实验为主的科学研究方法转变为以数值模拟为主的科学研究方法. 因此,学习和掌握计算机上常用的数值计算方法及其相关的基础理论知识对于当代大学生和研究生来说是非常必要的.

MatLab 是适合多学科、多种工作平台的功能强大、界面友好且开放性很强的大型优秀应用软件,是数值计算课程的基本教学工具. 我们遵循“以应用为目的,以必须够用为度”的原则,确定编写本书的指导思想为:强调算法形成思路及计算实践,将先进的科研成果引入教材,同时加强数值实验. 根据编者在同济大学多年的教学经验,参考国内外优秀教材,并结合当前数值计算领域中新发展起来的一些数值方法,编写了本教材. 本教材以 MatLab 语言及编程为基础,介绍常用的数值方法及其相关的基础理论,内容丰富,取材由浅入深、条理清楚、通俗易懂. 教材中的习题在随后出版的解题指南中将用 MatLab 程序语言给出详尽的解答.

全书共分九章,第一章“科学计算与 MatLab”由陈雄达编写,第二章“多项式插值与样条插值”和第三章“函数逼近”由陈素琴编写,第四章“数值积分与数值微分”由徐承龙编写,第五章“线性方程组的直接解法”和第六章“线性方程组的迭代解法”由穆祖元编写,第七章“非线性方程求解”由陈雄达编写,第八章“矩阵的特征值和特征向量的计算”由张秀艳编写,第九章“常微分方程数值解”由吴雄华编写,全书由陈素琴策划、吴雄华统校. 同济大学应用数学系的领导对本书的编写非常重视,给了我们很多帮助,同济大学出版社对本书的出版予以大力支持,在此我们表示衷心的感谢. 我们诚恳地希望使用本书的教师、学生及广大读者对本书的不足之处提出批评指正.

本书适用于工科高等院校的本科生和研究生使用,教师可根据学生的具体情况对教学内容作适当的取舍.

编者于同济园

2003.12

目　录

第一章　科学计算与 MatLab

第一节　科学计算的意义

数值计算是随着计算机的出现和大规模计算的需求而发展起来的一门新兴学科. 数值计算主要考虑各种数学模型及其算法，这些数学模型是为了解决各类应用领域特别是科学与工程计算领域的实际问题而提出的. 为此，数值计算有时也称为科学计算或科学工程计算. 随着科学技术的发展，计算机的性能和算法的效率都有了飞速的提高，要求求解的实际问题规模也成倍地扩大，其数学模型日趋复杂. 通常，这些数学模型不能够精确地求解，这时要简化模型并且提出相应的数值解法，然后在计算机上实现它并作实际检验. 随着计算能力的迅猛提高(包括硬件的性能提高和各种高效的算法的出现)，人们能够计算的实际问题的规模越来越大，并同时期待能解决一些超大规模的挑战性问题，如基因测序、全球天气模拟等等. 对于同一个问题，不同的算法在计算性能上可能相差百万倍甚至更多，科学计算的主要任务是设计高效可靠的数值算法. 例如，用一个每秒钟计算一亿次浮点运算的计算机求解一个 20 阶的线性方程组，克拉姆(Cramer)法则至少需要 30 万年，而用高斯(Gauss)消去法只不过用几秒钟而已. 这个事实说明了两个问题：一方面，计算方法效率的提高速度要比计算机性能的提高快得多；另一方面，选择高效率的计算方法无疑是极其重要的.

科学与工程计算领域中的问题求解一般要经历下面的几个过程. 首先根据实际问题构造相应的数学模型问题，然后根据问题特点选择计算方法，编制程序，最后在计算机上求解. 在这个过程中存在着许多与真实情况的差异，称为**误差**. 构造数学问题时，经常要忽略某些次要因素，这时提出的数学模型问题是原始问题的近似，这个过程产生的误差称为**模型误差**；构造出的模型中可能包含若干参数，它们往往只能通过观测得到，这时产生的误差称为**观测误差**；把数学问题转化为能够用计算机求解的数值问题引起的误差称为**截断误差**或**方法误差**；在计算机上做有限精度的运算产生的误差称为**舍入误差**.

科学计算的主要研究内容即是提出数值问题，设计高效的算法，并探讨方法误差和舍入误差对近似解的影响. **数值问题**是要求对有限个输入数据计算得出有限个输出数据，这些输出数据通常称为**数值解**，或者也可以理解为**近似解**. **数值算法**是求解问题的数值解的方法，它由有限个明确的无歧义的操作组成对输入数据的变换，其中每一个操作都是计算机能够完成的. 一个算法只有在保证可靠的前提下才有可能评

价其性能的好坏，通俗地讲，可靠性方面包含诸如算法的收敛性、稳定性、误差估计等多方面的内容. 评价一个算法的优劣应该考虑其时间复杂度(即占用的计算机时间)、空间复杂度(即占用的计算机存储空间)以及逻辑复杂度(即程序开发的周期长短及维护的难易程度).

由于各种科学计算问题最后通常都归结为求解一些基本的问题，数值算法领域的许多工作者为这些基本问题设计了一些相对固定的高效的算法，并把它们设计成简单的容易调用的功能函数并形成软件包. 但由于实际问题的复杂性及算法自身的适应性，调用者必须自行选择适合自己的功能. 现在数值计算领域流行的软件包括 Maple, Mathematica, MatLab 等；更多软件可以在网上查询(http://www.netlib.org)，其中的 MatLab 软件是在工程计算界被广泛使用的软件之一.

鉴于实际问题的复杂性，通常将其具体分解为一系列的子问题进行研究. 本课程主要涉及如下几方面的内容：

第一章为数值计算的基本知识及 MatLab 软件简介；其后各章内容包括函数的插值与逼近、数值积分与数值微分、线性方程组的直接求解和迭代求解、非线性方程的求解、矩阵特征值问题的求解、常微分方程的数值解.

第二节　误差基础知识

人们常用相对误差、绝对误差或有效数字来说明一个近似值的准确程度. 这些概念在科学计算中被广泛应用，下面我们对有关概念作一介绍.

一、误差的来源

我们把通过任何途径得到的数据或模型与真实情况之间的差异称为**误差**. 误差的来源经常是多方面的. 在建立数学模型的过程中，不可避免要忽略一些次要的因素，因而数学模型往往只是对实际问题的一种近似的表达，这二者之间的差异称为**模型误差**. 同时，数学模型中可能包含一些参数，它们可以通过仪器观测得到或通过经验得到，这种数据间的误差称为**观测误差**. 数值分析通常假定数学模型真实地反映了客观实际，因而这两类误差在数值分析的内容中并不常出现.

数学模型问题通常要转化为数值问题才能被求解，经常使用的转化手段往往有**离散化**等方法. 我们称这种数值问题与数学模型之间的误差为**截断误差**或**方法误差**，通常引起方法误差的原因在于必须在有限的步骤内在计算机上得到结果. 在用计算机实现数值方法的过程中，由于计算机表示的浮点数是固定的有限字长，因此，计算机并不能精确地表示所有的数，这样不仅原始输入数据有误差，中间计算的数据及最终输出结果也必然有误差. 这种因为计算机有限字长引起的误差称为**舍入误差**，原始

数据的误差导致最终结果也有误差的过程称为**误差传播**.

二、误差度量

假设 x 是真值，$\bar{x}$ 是它的近似值，则称 $\Delta x = x - \bar{x}$ 为该近似值的**绝对误差**，或简称**误差**. 一般说来，真值通常是求不出来的，因此也不可能知道 Δx 的值，而只能有如下估计：

$$|\Delta x| = |x - \bar{x}| \leqslant \varepsilon,$$

数 ε 称为**绝对误差限**或**误差限**. 如此就有

$$\bar{x} - \varepsilon \leqslant x \leqslant \bar{x} + \varepsilon,$$

在工程上也记为 $x = \bar{x} \pm \varepsilon$. 误差限给出了真值的范围，但并不能很好地表示近似值的精确程度. 例如，测量珠峰高度为 8848m，误差不超过 1m；在测量运动员的身高时就绝对不可以用这个误差限，否则，其结果是没有任何意义的. 同样的误差限对于不同的数据，其反应近似真实的程度可以完全相反，因此，必须同时考虑真值的大小.

若 x 是不为零的真值，$\bar{x}$ 是它的近似值，则称 $\delta x = \Delta x / x = (x - \bar{x})/x$ 为该近似值的**相对误差**(真值为零的情况没有定义). 若已知数 ε_r 满足

$$|\delta x| = \frac{|x - \bar{x}|}{|x|} \leqslant \varepsilon_r,$$

则称 ε_r 为**相对误差限**. 由于真值难以求出，通常也使用 $\delta x = \Delta x / \bar{x}$，假如 $\bar{x}$ 也非零.

三、有效数字

当 x 有很多位数字，为规定其近似数的表示法，使得用它表示的近似数自身就指明了其误差的大小，为此引入有效数字的概念.

设十进制数有如下的标准形式：

$$x = \pm 10^m \times 0.x_1 x_2 \cdots x_n x_{n+1} \cdots,$$

其中，m 为整数，$\{x_i\} \subseteq \{0,1,2,\cdots,9\}$ 且 $x_1 \neq 0$. 对 x 四舍五入保留 n 位数字，得到近似值 $\bar{x}$：

$$\bar{x} = \begin{cases} \pm 10^m \times 0.x_1 x_2 \cdots x_n, & x_{n+1} \leqslant 4, \\ \pm 10^m \times 0.x_1 x_2 \cdots (x_n + 1), & x_{n+1} \geqslant 5. \end{cases}$$

容易证明，四舍五入的误差限满足

$$|x - \bar{x}| \leqslant 10^m \times \left(\frac{1}{2} \times 10^{-n}\right) = \frac{1}{2} \times 10^{m-n}.$$

设 x 的近似值 $\bar{x}$ 有如下标准形式

$$\bar{x}=\pm 10^m\times 0.x_1x_2\cdots x_n\cdots x_p,$$

其中，m 为整数，$\{x_i\}\subseteq\{0,1,2,\cdots,9\}$ 且 $x_1\neq 0$，$p\geqslant n$. 如果有

$$|x-\bar{x}|\leqslant\frac{1}{2}\times 10^{m-n},$$

则称 $\bar{x}$ 为 x 的具有 n 位**有效数字**的近似数，其中，$x_1,x_2,\cdots,x_n$ 分别称为第 1 到第 n 位有效数字. 当 $p=n$ 时，称 $\bar{x}$ 为**有效数**. 有效数的误差限是末位数单位的一半，其本身就体现了误差界，因此有效数末尾是不可以随便添加零的.

四、向前误差和向后误差

数值计算结果的可信程度至少部分依赖于输入数据的准确程度. 设想，如果输入数据只有 4 位有效数字，一般情况下只能期望最终结果的有效数字能达到 4 位而不是更多，不管计算过程如何的精确. 那么，最终结果是如何依赖于输入数据和计算过程的呢？

下面举例说明上述的这个过程. 假设要计算 $y=f(x)$，其中，f 是某个函数，往往得到某个值 $\bar{y}$ 而不是真值 y. 把 $\Delta y=y-\bar{y}$ 称为**向前误差**. 向前误差通常取决于计算方式，并不很好估计，在最弱的条件下只能得到一些不实用的误差界. 换一种方式考虑此问题，可以问：在输入的数据有多大的改变时，假设计算过程精确就可以得到计算值 $\bar{y}$？或者在数学上可以说成：如何估计 $\Delta x=x-\bar{x}$ 使得 $\bar{y}=f(\bar{x})$？把 Δx 称为**向后误差**. 从这方面讲，一个好的近似解应该是一个稍稍变化了的问题的精确解.

如图 1-1 所示，其中的 $\bar{f}$ 是实际上的计算方式，$\bar{f}(x)=f(\bar{x})$ 是因为 $\bar{x}$ 而特意挑选出来的.

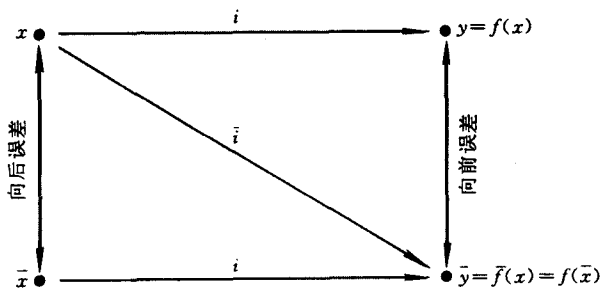

图 1-1 向前误差和向后误差

五、计算机的浮点数系

计算机内部通常使用浮点数进行实数的运算. 由于浮点数是只有有限字长的二进制数，大部分实数存入计算机时需要做四舍五入，由此引起的误差称为**舍入误差**. 一个浮点数的表示由正负号、小数形式的尾数以及为确定小数点位置的阶三部分组成. 例如单精度实数用 32 位的二进制表示，其中，符号占 1 位，尾数占 23 位，阶数占 8 位. 这样一个规范化的计算机单精度数(除零外)可以写成如下形式：

$$\pm 2^{p}\times(0.\alpha_1\alpha_2\alpha_3\cdots\alpha_{23})_2,\quad |p|\leqslant 2^7-1,\quad p\in\boldsymbol{Z},\quad \alpha_i\in\{0,1\}.$$

上面记号中，$\boldsymbol{Z}$ 表示整数集. 二进制的非零数字只有 1，所以 $\alpha_1\equiv 1$；阶数的 8 位中须有 1 位表示阶数的符号，所以阶数的值占 7 位. 凡是能够写成上述形式的数称为**机器数**. 设机器数 a 有上述形式，则与之相邻的机器数为 $b=a+2^{p-23}$ 和 $c=a-2^{p-23}$. 这样区间 (c,a) 和 (a,b) 中的数无法准确表示，计算机通常按规定用与之最近的机器数表示.

设实数 x 在机器中的浮点(float)表示为 $fl(x)$，把 $x-fl(x)$ 称为**舍入误差**. 如当 $x\in\left[\frac{c+a}{2},\frac{a+b}{2}\right)=[a-2^{p-1-23},a+2^{p-1-23})$ 时，用 a 表示 x，即 $fl(x)=a$. 其相对误差满足

$$|e_r|=\left|\frac{x-fl(x)}{fl(x)}\right|\leqslant\frac{2^{p-1-23}}{2^{p-1}}=2^{-23}\approx 10^{-6.9}.$$

上式表明单精度实数有六至七位有效数字.

二进制阶数最高为 2^7-1 相应于十进制的 38，即 $(2^7-1)\lg 2$. 因此，单精度实数(除零外)的数量级不大于 10^{38} 且不小于 10^{-38}. 当输入、输出或中间数据太大而无法表示时，计算过程将会非正常停止，此现象称为**上溢**(overflow)；当数据太小而只能用零表示时，计算机将此数置零，精度损失，此现象称为**下溢**(underflow). 下溢并不总是有害的，在做浮点运算时，需要考虑数据运算可能产生的上溢及有害的下溢.

第三节　数值计算应注意的问题

舍入误差在实际计算中几乎是不可避免的，定量地分析舍入误差的积累过程往往都是非常繁杂的. 一个可行的办法是研究舍入误差是否能够得到控制或者不影响得到可靠的结果. 一个算法，如果在一定的条件下，其舍入误差在整个运算过程中能够得到控制或者舍入误差的增长不影响产生可靠的结果，则称该算法是**数值稳定**的，否则称为**数值不稳定**的.

例 1.1 计算 $S_n=\int_0^1 \frac{x^n}{x+5}\mathrm{d}x$，其中，$n=0,1,2,\cdots,8$.

解 由于

$$S_n+5S_{n-1}=\int_0^1 \frac{x^n+5x^{n-1}}{x+5}\mathrm{d}x=\int_0^1 x^{n-1}\mathrm{d}x=\frac{1}{n},$$

取 $S_0=\ln 6-\ln 5=0.182$，利用公式 $S_n=\frac{1}{n}-5S_{n-1}$ 可以逐步得到如下的值（精确到小数点后 3 位）：

$$S_1=0.090,\quad S_2=0.050,\quad S_3=0.083,\quad S_4=-0.165,$$
$$S_5=1.025,\quad S_6=-4.958,\quad S_7=24.933,\quad S_8=-124.540.$$

易知

$$\frac{1}{6(n+1)}=\int_0^1 \frac{x^n}{6}\mathrm{d}x\leqslant S_n\leqslant\int_0^1 \frac{x^n}{5}\mathrm{d}x=\frac{1}{5(n+1)},$$

所以上述的 8 个计算结果中，那些负的或大于 1 的结果都是错误的.（当然，其余的结果也可能有比较大的误差.）

下面分析造成这种计算结果的原因. 假设 S_n 的真值为 S_n^*，误差为 ε_n，即 $S_n=S_n^*-\varepsilon_n$. 对于真值，也有等式 $S_n^*+5S_{n-1}^*=\frac{1}{n}$，综合计算式，有

$$\varepsilon_n=-5\varepsilon_{n-1}.$$

这意味着哪怕开始只有一点误差，这个误差就能随着计算的进行以 5 倍的速度增长，该计算过程是不稳定的.

换一种方式，把计算方式改为

$$S_{n-1}=\frac{1}{5n}-\frac{1}{5}S_n,\quad n=8,7,\cdots,1.$$

S_8 可以用上面的估计式计算，例如，$S_8=\left(\frac{1}{6\times 9}+\frac{1}{5\times 9}\right)\Big/2\approx 0.020$. 逐步计算得

$$S_7=0.021,\quad S_6=0.024,\quad S_5=0.028,\quad S_4=0.034,$$
$$S_3=0.043,\quad S_2=0.058,\quad S_1=0.088,\quad S_0=0.182.$$

这样的计算结果和实际是很相近的. 对 S_8 不同的估计方式，最后得到的结果也相似：这个事实只需对递推计算公式进行类似的误差分析就可以得到.

误差的传播在一些实际的问题中经常是很复杂的，不像上面的例子可以得到一

个误差传播的具体的公式. 通过对误差传播规律的简单分析，在数值计算中应该注意如下的基本问题.

一、避免相近的数相减

在数值计算中，两个相近的数相减时有效数字会损失. 例如，计算

$$y=\sqrt{x+1}-\sqrt{x},$$

其中，x 是比较大的数，例如，$x=1000$. 取四位有效数字计算，有

$$y=\sqrt{1001}-\sqrt{1000}=31.64-31.62=0.02.$$

在计算过程中，可以看到每个根号的计算都有 4 位有效数字，相减之后结果只有 1 位有效数字，相对误差变得很大，严重影响了结果的精确程度. 事实上，y 可以有如下的等价计算公式：

$$y=\frac{1}{\sqrt{x+1}+\sqrt{x}}.$$

按此公式计算可得 $y=0.01581$，仍旧有 3 位有效数字. 可见数学上等价的计算公式在实际计算上是不等价的. 变换公式可以有很多种，例如：

$$\frac{1}{x}-\frac{1}{x+1}=\frac{1}{x(x+1)},$$

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

$$\ln(x-\sqrt{x^2-1})=-\ln(x+\sqrt{x^2-1}),$$

和

$$\sin(x+\varepsilon)-\sin x=2\cos\left(x+\frac{\varepsilon}{2}\right)\sin\frac{\varepsilon}{2}$$

等等. 当 x 比较大或 ε 比较小时，上述各等式右边的计算方式都要比左边的有效.

二、避免量级相差太大的两数相除

计算大数除以小数或小数除以大数时，容易出现计算机溢出的情形，使得计算过程非正常中断或者中间数据没有任何有效数字. 在这种情况下，有必要在量级上对这两个数做一些处理.

三、避免大数和小数相加减

在数值计算中，有时候会碰到数量级相差很大的两个数相加或相减. 计算机做加减法是要对阶的，即把这两个数都写成同一个阶数的表示形式，再对其尾数相加减.

例如，假设十进制 5 位数机器上做下面的加法：

$$12345+0.7.$$

计算机做加法时，要把这两个数都写成尾数小于 1 的同阶的数，即

$$0.12345\times10^5+0.000007\times10^5,$$

但是计算机只能表示五位尾数，因此，第二个加数在计算机上就等于 0，这种情况称为“大数吃掉小数”.

例 1.2 计算下面级数的部分和

$$1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}.$$

解 最自然的计算方式是设一个部分和为 S，依次把第 k 项加到 S 上. 对于比较大的 n，因为调和级数是发散的，S 会越来越大(趋向无穷). 当我们把第 k 项加到 S 上时，所做的加法是

$$S+\frac{1}{k}.$$

依照上面的分析，只要 k 足够大，就会出现“大数吃掉小数”，以后的求部分和的运算更是如此. 因此，实际计算上这个部分和并不会越来越大而趋向无穷，而是停在某一个大数上，其后的所有项都被吃掉！

我们可以采取一些措施来防止大数和小数相加，例如上面的例子，可以从后面往前面做加法，也可以在中间加括号等.

四、简化计算步骤

同样的结果在数学上可以有不同的表达形式，在计算机上它们的结果可能是完全不一样的. 例如，计算多项式

$$p_n(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0,$$

如果直接采用上面的公式逐项求和，那么计算第 $n-k+1$ 项需要 k 个乘法，因此总运算量为 $\frac{1}{2}n(n+1)$ 个乘法和 n 个加法. 多项式的求值也可以改成下面的形式

$$p_n(x)=x(x\cdot\cdots\cdot(x(a_nx+a_{n-1})+a_{n-2})+\cdots+a_1)+a_0,$$

这时总运算量为 n 个乘法和 n 个加法. 这个算法叫做秦九韶算法，或称霍姆(Horner)算法.

一个好的算法不仅应当是数值稳定的算法，还应当是一个高精度的算法和一个高效的算法，然而这些要求常常不能兼备，某些情况下甚至是相互矛盾、相互制约的.

第四节　MatLab 简介

MatLab 源于 Matrix Laboratory 一词，意为矩阵实验室. MatLab 软件是一个功能非常强大的科学计算软件. 早期的 MatLab 是一个专门为方便调用 LINPACK 和 EISPACK 软件包而做的界面程序；最新的 MatLab 版本含有科学计算、符号计算、图形处理等功能，可以很方便地处理各类矩阵及多项式运算、线性方程组求解、微分方程数值解、插值拟合、统计和优化等问题，并且可以针对用户提供问题的特点自己选择合适的算法.

MatLab 的数据类型包括：数、字符串、矩阵、单元型数据和结构型数据. 后两种实际上是复合数据类型，而数和字符串都可以看成为矩阵的特例，因此，矩阵数据类型是最有代表性的类型，这也是 MatLab 名字的来源.

MatLab 的变量不必事先说明，也不需要指定类型，它会根据变量所涉及到的操作来决定变量的类型. 任何以字母开头，包含字母、数字或下划线并且长度少于 32 的字符串都可以作为变量的名字. 变量名区分大小写，并且不能与系统的关键字和内部函数同名. 通常 MatLab 中的变量(系统的和用户的)都以约定俗成的简写命名. 例如：

```
≫  x = 3
```

该命令定义了一个叫做 x 的变量并赋值 3，其中≫为系统提示符. 回车运行后，系统会显示 x 的值，并且出现新的提示符. 若不想显示 x 的值，可以写

```
≫  x = 3;
```

行尾的分号起抑制显示的作用. 也可以定义其他类型的变量，如

```
≫  s = 'hello world!';  A = [1 2 3; 2 3 4; 3 4 5];
```

可以看到，s 是一个字符串，字符串的常量由单引号括起来；A 是一个矩阵，以[ ]为标识，同一行的元素以空格或逗号分隔，行与行之间以分号或回车分隔. s 和 A 的赋值可写在同一行. 所有变量都需要赋值后才能参加运算. MatLab 有自己预定义的变量，经常使用的有：

```
pi,    i,j,    eps,    NaN,    Inf,
```

分别表示圆周率、虚根、浮点运算的相对精度，Not-a-Number(不定型)和无穷大.

MatLab 的一个主要特点是向量运算. 例如:

```
≫  w = 0:pi/2:2 * pi;
≫  t = sin(w);
```

在上面的命令中,首先定义了一个向量 w,w 是以 0 为初值,π/2 为步长,2π 为终值的向量,即 w=[0,π/2,π,3π/2,2π]. t 是 w 的正弦值,即 t 是一个向量,它的每个分量是 w 相应分量的正弦,所以,t=[0 1 0 −1 0]. MatLab 的许多简单数值函数都有向量运算的功能,如三角和反三角函数(sin,cos,tan,asin,acos 等,后两个为反正弦和反余弦),对数和指数函数(log,log10,exp 等,前两个为自然对数和常用对数)和其他函数(abs,sqrt,sign 等,分别表示绝对值、算术平方根和符号函数). 一般地,在 MatLab 中,a:s:b 表示以 a 为初值,s 为步长,b 为上界(或下界,视 s 的正负而定)的向量,若 b−a 不刚好是 s 的整数倍,则该向量中不出现 b. 例如,1:2:6 的实际的值为向量[1 3 5],而 1:−2:−6 的值为[1 −1 −3 −5].

一、向量和矩阵的基本运算

下面的各种命令产生不同的向量:

```
≫  x = ones(5,1);
≫  y = zeros(5,1);
≫  z = 1:5;
```

其中,x 是元素全为 1 的 5 行 1 列的向量,y 是元素全为 0 的 5 行 1 列的向量,而 z 是向量[1 2 3 4 5]. 命令 ones 和 zeros 有两个参数,例如可以写 ones(m,n)表示一个 m 行 n 列元素全为 1 的矩阵,若写 ones(1,n)则表示行向量. 向量的输入也可以用括号形式,如

```
≫  u = [2 3 4 5 0];
```

向量的加减和数乘运算和数学表达式一样,例如 x+y 和 pi * u 等. 向量的内积可以用如下方式实现:

```
≫  z * u'
```

首先,u′把向量 u 转置为列向量,z 和 u′相乘得到内积的值,即 40. MatLab 系统还可以进行向量的按分量相乘除,或做乘幂运算. 例如

```
≫  z . * u
```

该命令产生一个向量,其每个分量是 z 和 u 对应分量相乘,即[2 6 12 20 0]. 这个运算也称点乘,相应地还有点除(z. /u)和点幂(u. ^3). 因此内积也可以写为

```
≫  sum(z. * u)
```

函数 sum 作用在向量上会返回该向量所有分量的和. 类似的函数还有 prod(所有分量的积)、max、min(所有分量的最大最小值)等. 上述各命令对于同样阶数的矩