普通高等教育“十二五”规划教材

数值计算理论与实现

陈长军

华中科技大学出版社

本书的出版得到了国家自然科学基金的支持
（基金项目号：31370848）

数值计算理论与实现

陈长军

华中科技大学出版社
中国·武汉

内 容 提 要

本书介绍了数值计算的基本思路与方法，内容涵盖了非线性方程求根、线性方程组求解、矩阵本征值、插值与拟合、数值微分和积分以及常微分方程等方面。全书结构完整，叙述详细，读者阅读本书可以全面了解数值计算基础理论。另外，书中还穿插讲解了大量的算法实现程序代码，这些程序代码由作者在多年教学过程中积累而来，作者力求兼顾这些程序代码的执行效率和可阅读性，这些代码都依章节整合起来，非常方便调试，它们可以帮助读者学以致用，快速掌握现代计算方法。

图书在版编目(CIP)数据

数值计算理论与实现/陈长军. —武汉：华中科技大学出版社，2014.5
ISBN 978-7-5680-0053-6

Ⅰ.①数… Ⅱ.①陈… Ⅲ.①数值计算-高等学校-教材 Ⅳ.①O241

中国版本图书馆 CIP 数据核字(2014)第 100694 号

数值计算理论与实现 陈长军

策划编辑：周芬娜
责任编辑：余 涛
封面设计：潘 群
责任校对：封力煊
责任监印：周治超
出版发行：华中科技大学出版社(中国·武汉)
武昌喻家山 邮编：430074 电话：(027)81321915
录 排：武汉市洪山区佳年华文印部
印 刷：华中理工大学印刷厂
开 本：710mm×1000mm 1/16
印 张：11.75
字 数：250 千字
版 次：2014 年 9 月第 1 版第 1 次印刷
定 价：25.00 元

前　言

数值计算方法是一门伴随着计算机技术发展起来的新兴学科，它使用现代计算理论来处理复杂的数学问题，大大增强了人们对客观世界的认知能力。笔者从2010年春季开始，一直为华中科技大学物理学院的本科生讲授计算物理课程，在教学过程中涉及的许多数学物理方程都难以做解析计算，唯有借助数值计算手段才能够进行有效分析，其重要地位不可取代。因此，笔者多年来不断积累资料、总结经验，最终汇集成这本书。

本书有一些特别之处。首先，在讲授理论方法的同时，更侧重于算法实现，并给出了所有算法的程序代码，便于学生对刚刚学会的算法活学活用，在实践中真正掌握数值计算理论。当学习完后，书中代码也可以整合起来，组成一个完整且实用的数学函数库。其次，本书选择Fortran语言作为算法实现的编程语言。作为一种历史悠久的科学计算语言，Fortran语言发展成熟、功能完备，其计算效率与C语言相当，其在科学计算领域内的编程效率甚至超过了C语言。作为理工科学生，掌握Fortran语言很有必要。最后，本书采用一些前后连贯的思路来讲授某些知识点，如方程求根和函数求极值，最速下降法和松弛迭代法，这些表面上看似独立的问题，其实都有着相通性，在本质上可以当成一个问题来考虑。

为达到最佳的学习效果，本书推荐的阅读对象是普通高校二、三年级本科生，或者从事与数值计算相关的科学工作者。读者需要具有一定的高等数学知识，能够理解书中涉及的各类数学问题。另外，本书也可以作为工具书使用，书中介绍了许多实用算法，算法描述和代码实现都尽量做到简洁直观、便于理解，方便读者查询和测试。

本书由笔者一人完成，虽经反复修改，但受学识和精力所限，书中错误在所难免，望广大读者指正。

陈长军

于华中科技大学

目　录

第一章 绪 论

本章简要介绍了数值计算的重要性，以及一些必要知识，包括编程语言、绘图工具和初步的误差分析方法等。这些内容是数值计算的基础，读者需要尽快掌握，为后续学习做好准备。

第1节 数值计算简介

一般我们接触到的物理模型，具有解析解的情况非常少。比如量子力学中的薛定谔方程，能够精确求解波函数的有一维无限深势阱、方势阱、谐振子势等极少数情形，而其他特殊势函数用纯理论方法计算会非常困难。还比如，电动力学中计算可极化介质内外的空间静电势，用常规的级数展开方法只能处理球对称分布等极特殊情形，对于复杂边界问题则难以下手。另外，还有一些看似简单的物理模型，如天文学中经典的三体问题，甚至没有解析解。然而客观存在的事实却毫无疑问地表明，这种三体模型在设定初值条件以后有着确定的运动轨迹。因此，在面对复杂的物理问题时，传统的基于解析公式的理论研究方法越来越力不从心，而实际的实验物理研究又迫切需要理论指导。在这两方面需求的强力推动之下，计算物理在现代得到了快速发展，已经深入到核物理、凝聚态物理、生物物理、天体物理、光学、声学等几乎所有物理学科，发挥着越来越重要的作用。计算物理的核心正是数值计算，掌握这一现代计算方法能够大大拓展我们的视野，帮助我们深入研究各类物理问题。

目前，国际上已经有很多软件包含数值计算接口，如 Matlab 和 Mathematica，甚至还有一些独立的数值计算模块，如 IMSL、PETSc、LAPACK 等，它们调用方便，易于使用。但是，对于理工科学生来说，光会使用这些模块或接口是不够的，必须了解其内在思想。这是因为实际物理问题往往千变万化，可用的算法有很多，需要做仔细选择。例如，用泊松方程来求解空间静电场，我们通过有限差分法将其简化为线性方程组后，可以用 Gauss 消元法、LU 分解法、Jacobi 迭代法等多种算法，它们有着不同的计算效率和收敛标准，只有熟悉了它们各自的特点，才能选择最合适的算法，顺利完成计算工作。另外，有些物理问题没有现成的算法可用，或者需要在标准算法的基础上增加自定义的计算模块，或者需要代码并行化，这些都必须自行实现。此时，如果没有透彻了解标准算法的计算原理，是很难完成这个任务的。

可见学好数值计算方法很有必要，但在学习之前，我们需要先做一些准备工作。首先，我们要熟悉高等数学知识，包括微积分、线性代数和常微分方程等，它们的解析计算方法是发展数值算法的基础。其次，仅仅理解这些数值算法是不够的，必须要将

算法转化成计算机能够运行的程序代码才能真正发挥作用。因此，我们必须具备基本的计算机编程能力，能够使用主流的编程语言，如 C 语言和 Fortran 语言，以便实现算法。最后，我们还需要一定的计算机绘图技巧，当计算结果出来后，为便于研究和分析，往往需要将数据绘制成图表，因此学会使用绘图软件，将显著提高后期的工作效率。

第 2 节 编程语言

任何数值算法都需要用编程语言来实现，现在可以使用的主流编程语言有很多，如 C/C++、Java、C#、Fortran、Python 等，甚至还有一些软件带有编程功能，包括前面提到的 Matlab 和 Mathematica。它们各自的特点简要介绍如下。

C/C++功能强大，在用户界面、网络编程、3D 图形编程等许多方面都能胜任；Java 和 C# 则隐藏了底层的实现细节，编程效率极高；Fortran 自带了很多内部函数，方便处理复数、矢量和矩阵等数据对象，适于公式运算。Python、Matlab、Mathematica 都属于解释执行的编程语言，易读易懂，特别容易调试，它们还封装了各类计算模块，在人机交互、绘图、数据处理等方面都有着独特优势。

虽然现在可供选择的编程语言和编程软件有很多，各有专长，但一定要明确一点，我们是用它们来做数值计算的，因此在选择的时候，必须要在以下几方面仔细权衡。

首先，计算效率是我们需要考虑的首要因素。在以上提到的各类编程语言中，计算速度最快的当数 Fortran 语言和 C/C++语言。其他语言或者需要软件运行环境，或者需要程序解释器，都或多或少地拖慢了代码的计算效率。

其次，我们需要考虑的是在科学计算方面的易用性。数值计算中会涉及许多数学公式，相应地也会包含各类数据操作，如矩阵加减、复数运算等。在这些方面，Fortran语言具有天然优势。比如，矢量 $\boldsymbol{a}$ 和矢量 $\boldsymbol{b}$ 相加得到矢量 $\boldsymbol{c}$，Fortran 语言仅仅需要一条语句 c=a+b 就可以完成运算；而 C/C++语言则必须加上循环语句才行。又比如，矩阵 $\boldsymbol{a}$ 和矩阵 $\boldsymbol{b}$ 相乘得到矩阵 $\boldsymbol{c}$，Fortran 语言也仅仅需要一条语句 c=matmul(a,b)就可以完成；而 C/C++语言需要加上两层循环语句(用户自定义新的函数可以完成同样的工作，但同时也会降低代码的可读性)。复数运算同样如此，因为自带复数数据类型(含实部和虚部)，Fortran 语言处理复数的加减和乘法都非常方便。因此，要执行同样的计算工作，用 Fortran 语言写出来的代码往往要比 C/C++语言的简单，科学工作者可以因此节省一定的代码编写和维护工作，把有限的思维劳动投入到非常重要的算法设计与数值模拟中。

最后，我们要考虑的是高级语言特性——面向对象的编程思想。比如，不同数据类型的变量或数组能否封装在一起方便管理？完成同一任务的多个子程序能否合并成一个模块以便重复利用？相似功能的函数或子程序，其参数类型和数目能否改变

以增强其多态性？Fortran(Fortran90 以后版本)和 C++都具备这些高级特性，使用它们都可以写出结构性强、便于维护、重复利用率高的代码。

综合以上各方面分析，Fortran 语言在数值计算领域显示出了很好的实用性，因此，本书中的所有算法都用 Fortran 语言实现。

Fortran 编译环境有很多，如 MinGW、Compaq Visual Fortran 和 Intel Visual Fortran。其中 Compaq Visual Fortran 和 Intel Visual Fortran 是集成开发环境(见图 1-1)，编译和调试代码都比较方便；而 MinGW 是终端类型的编译器，功能较少，但它是免费的，体积小巧，易于下载和安装。

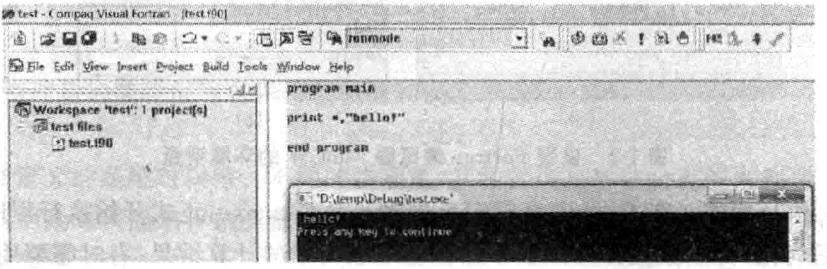


图 1-1 Compaq Visual Fortran 集成开发环境

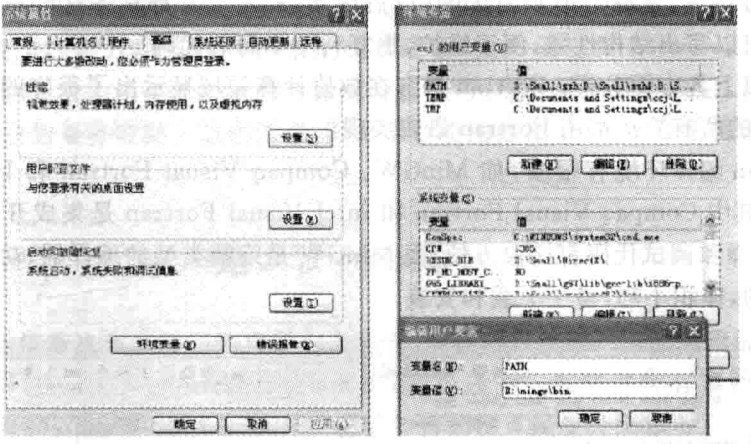
下面简单介绍一下 MinGW 的安装和编译过程。

(1) 下载安装器。进入 MinGW 的官方网站 http://www.mingw.org/，点击右上角的 Download Installer，下载进程会自动开始下载文件 mingw-get-setup.exe。

(2) 下载并安装编译器。mingw-get-setup.exe 下载完成后，双击它，会打开一个对话框，用户需要设定安装目录并选择编译器种类，这里只需勾选 mingw-basic 和 mingw-gfortran，随后会下载和安装 Fortran 编译器。

(3) 设置环境变量。编译器安装好以后，需要把编译器所在路径添加到系统的 path 环境变量中去，以便可以在任意路径上执行编译命令。具体操作过程是：用鼠标右键单击桌面上的“我的电脑”图标，选择右键菜单中的“属性”，系统会弹出图 1-2(a)所示的“系统属性”窗口。选择“高级”选项卡，再单击面板上的“环境变量”按钮，就会出现图 1-2(b)所示的“环境变量”窗口，最后单击窗口中的 PATH 一栏的标签，在 PATH 变量值最后加上 MinGW 编译器可执行文件的所在路径(如 D:\mingw\bin)，单击“确定”按钮。然后在 Windows 操作系统中打开 DOS 命令行窗口，输入 gfortran 命令并执行。如果出现提示信息 gfortran: fatal error: no input files，则表明 MinGW 环境变量设置成功。

(4) 编译代码。编译过程非常简单，假如用于测试的程序代码 hello.f90 在目录 D:\program\下，打开 DOS 命令行窗口，输入命令 D: 进入代码所在磁盘，然后输入命令 cd D:\program\ 进入代码所在目录，最后执行命令 gfortran hello.f90，就可以使用 gfortran 编译器对代码文件 hello.f90 进行编译。编译完成后，在代码所在目录下会生成一个可执行文件 a.exe。



(a) (b)

图 1-2 设置 Fortran 编译器 MinGW 的环境变量

(5) 执行程序。在 DOS 命令行窗口中输入命令 a. exe,正式开始执行程序。

最后讨论一点,数值计算过程中往往会生成大量的计算结果,有时需要用绘图软件进行直观展示。与编程语言的选择一样,现在绘图软件的选择同样很多。如 Matlab、Mathematica、Origin 都具有强大的绘图功能,但它们过于庞大复杂,不利于初学者。本书推荐使用 Gnuplot(见图 1-3),这是一个免费的绘图软件,可以从它的主页 http://www.gnuplot.info/下载。它操作简单且功能强大,能够绘制各种类型的二维和三维图形,许多学术期刊上的图片都是由它绘制的。另外,Gnuplot 还可以整合多个文件中的数据,生成连续动画。这对于分析波动问题、热传导问题等许多动

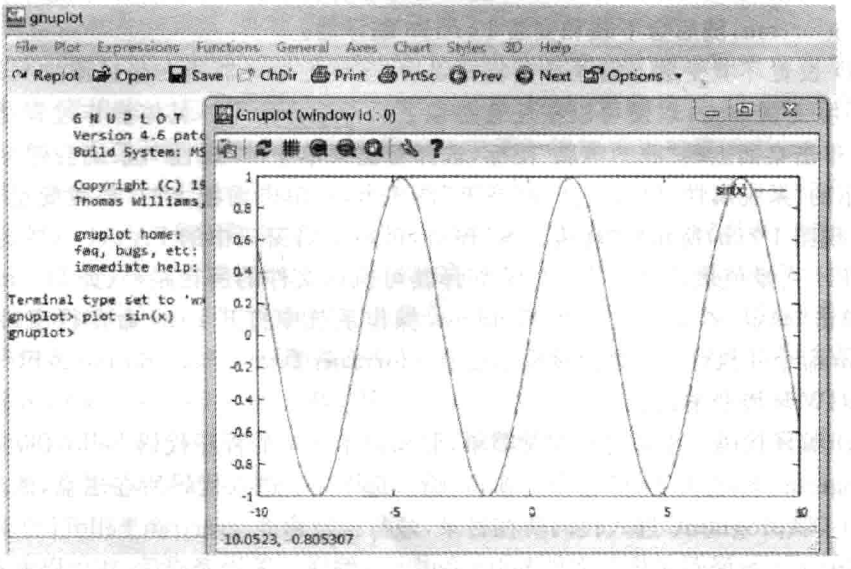


图 1-3 Gnuplot 绘图软件

态物理问题，都是非常有用的。

第3节 误差分析

如同实验数据会有误差一样，数值计算结果同样会有误差。这一小节我们来讨论一下关于误差的问题，包括误差的定义、分类，以及控制误差的手段。

误差表示近似值与准确值之间的差异，反映了近似值的准确程度。如果用 x 表示一个物理量的实际值（准确值），x^* 表示该物理量的实验测量值或者理论计算值（都可以视为近似值），那么误差 e 的定义式为

$$e=|x-x^*| \tag{1.1}$$

反过来，物理量的准确值也可以用近似值和误差来表示，即

$$x=x^*\pm e \tag{1.2}$$

上面定义的是绝对误差。不同的物理量，其实际大小往往会有较大差别，这时直接用上面的绝对误差来评估测量或计算结果的准确度是不合适的。例如，测量之后得到大小分别为 100 和 1000 的两个物理量，如果绝对误差都是 10，显然后者的测量结果更为准确。因此，我们有必要考虑物理量自身的大小，定义一个新的误差标准，这就是相对误差。它的大小就是绝对误差与近似值之间的比值，即

$$\varepsilon=\frac{e}{x^*} \tag{1.3}$$

上面给出了一般的误差定义。在实际测量或者计算过程中，形成这些误差的原因主要有以下几个方面。

（1）模型误差。实际的物理问题往往非常复杂，难以考虑所有因素，为了能够更有效地进行研究，我们不得不先建立一个简化模型后再做分析。这个简化过程必然会引入模型误差。例如，观察一个重物的自由下落过程，打算通过测量其下落距离和下落时间来得到本地的重力加速度。我们可以建立三种模型，分别是真空中的匀加速直线运动、考虑空气阻力的变加速直线运动和考虑非惯性系中科里奥利力的曲线运动。这些不同的物理模型，都有着固有的模型误差。

（2）观测误差。前面的物理模型中往往带有需要实验测量的参数，如长度、时间、速度、体积、密度等，不同的测量仪器或者测量手段，都会引入观测误差，随之传递到最终的计算结果中。

（3）舍入误差。数值计算离不开计算机的帮助，现在的高速计算机，其运行速度是人脑无法企及的，但是，计算机只能以有限位数来保存数据，比如一些无穷循环小数，计算机在保存它们时必须在小数点后的有限位置作截断（四舍五入），由此造成的误差称为舍入误差。实际计算过程中，某些变量可能会被迭代成百上千次，这样累积之后的舍入误差是相当可观的。

（4）截断误差。数值计算中还会碰到一些复杂函数，它们或带有三角函数，或带

有指数，或带有对数，我们常常需要对它们进行泰勒展开，以便将其转化为易于处理的多项式，这在插值、数值微积分、常微分方程等章节中都会介绍。这样做固然方便，但经过截断的泰勒展开式毕竟是近似的，伴随而来的是截断误差，其高阶余项就反映了这部分误差的大小。

提高效率、减小误差一直是我们的目标。在上述误差中，除了观测误差外，其余三项误差（模型误差、舍入误差和截断误差）都是数值计算中需要控制的。模型误差在算法设计阶段就需要考虑。物理模型越完善，相应误差就越小。舍入误差取决于程序中的数据类型，比如一个单精度浮点数由 4 个字节的内存来记录，一个双精度浮点数由 8 个字节的内存来记录，用于存储的字节越多，则舍入误差越小。截断误差则由泰勒展开式中的步长和阶数决定，步长越小，阶数越高，则截断误差越小。需要指出的是，按照以上方式来减少误差，可以让计算结果更为准确，但必然会增加相应的计算量。在计算时间和计算精度之间，我们应该寻找一个合理的平衡点。

最后，我们来探讨一下误差的传播和累积。了解这一点非常重要，因为即使针对的是同一个问题，使用不同的计算思路也会给出完全不同的计算结果。下面来看一个简单的例子，计算一个积分

$$I_n = \int_0^1 x^n \mathrm{e}^{x-1}\,\mathrm{d}x, \quad n = 1,2,\cdots,20 \tag{1.4}$$

要计算不同大小的 n 对应的积分值 I_n，用解析方法是很难计算的，但可以先利用分部积分将这个公式变换一下形式，即

$$\int_0^1 x^n \mathrm{e}^{x-1}\,\mathrm{d}x = x^n \mathrm{e}^{x-1}\Big|_0^1 - n\int_0^1 \mathrm{e}^{x-1}x^{n-1}\,\mathrm{d}x = 1 - n\int_0^1 \mathrm{e}^{x-1}x^{n-1}\,\mathrm{d}x \tag{1.5}$$

这样，原来的积分就被转换为一个迭代公式，即

$$I_n = 1 - nI_{n-1} \tag{1.6}$$

只要知道了 I_0，就可以利用这一迭代公式计算以后所有的积分值 I_n。

Fortran 代码如下：

```
program main
  implicit none
  integer :: n
  real*8 :: I(0:20)
  I(0)=1.0d0 - exp(-1.0d0)
  do n=1,20
    I(n)=1.0d0 - dble(n) * I(n-1)
  end do
  print "(4(a2,i2,a2,f7.3,5x))",("I(",n,")=",I(n),n=1,20)
end program
```

图 1-4 所示的是输出结果。

```
I(1)=  0.368   I(2)=  0.264   I(3)=  0.207   I(4)=  0.171
I(5)=  0.146   I(6)=  0.127   I(7)=  0.112   I(8)=  0.101
I(9)=  0.092   I(10)=  0.084   I(11)=  0.077   I(12)=  0.072
I(13)=  0.067   I(14)=  0.063   I(15)=  0.059   I(16)=  0.055
I(17)=  0.057   I(18)= -0.029   I(19)=  1.560   I(20)= -30.192
```

图 1-4 利用迭代公式(1.6)计算的积分值(I_1 到 I_{20})

我们发现,从 $n=18$ 开始,积分值变得不稳定起来,那么最终的结果 I_{20} 是否准确呢?我们需要从初值 I_0 开始做分析,即

$$I_0=1-\frac{1}{e} \tag{1.7}$$

假定计算机在存储 I_0 的时候,引入了舍入误差 ε,则以后每一步的积分结果(近似值,全部用 * 标记)及相应误差可以计算如下:

$$\begin{cases} I_0^* = I_0 \pm \varepsilon \\ I_1^* = 1 - I_0^* = 1 - I_0 \pm \varepsilon \\ I_2^* = 1 - 2I_1^* = 1 - 2(1 - I_0 \pm \varepsilon) = -1 + 2I_0 \pm 2\varepsilon \\ I_3^* = 1 - 3I_2^* = 1 - 3(-1 + 2I_0 + 2\varepsilon) = 4 - 3 \times 2I_0 \pm 3 \times 2\varepsilon \\ \quad \vdots \end{cases} \tag{1.8}$$

可见即使第一步的舍入误差非常小,如 $\varepsilon=10^{-15}$,但到了第 20 步,误差将放大到 20! 倍。这样大的误差显然令计算结果没有意义。

如果我们换个思路,结果会怎样呢?将迭代公式变换如下:

$$I_{n-1}=\frac{1-I_n}{n} \tag{1.9}$$

然后从 I_{20}(令积分值为 0.0)开始逆向迭代到 I_0。Fortran 代码如下:

```
program main
implicit none
integer :: n
real*8 :: I(20)
I(20)=0.0d0
do n=20,2,-1
   I(n-1)=(1.0d0-I(n))/dble(n)
end do
print "(4(a2,i2,a2,f7.3,5x))",("I(",n,")=",I(n),n=1,20)
end program
```

图 1-5 所示的是输出结果。

I(1)=	0.368	I(2)=	0.264	I(3)=	0.207	I(4)=	0.171
I(5)=	0.146	I(6)=	0.127	I(7)=	0.112	I(8)=	0.101
I(9)=	0.092	I(10)=	0.084	I(11)=	0.077	I(12)=	0.072
I(13)=	0.067	I(14)=	0.063	I(15)=	0.059	I(16)=	0.056
I(17)=	0.053	I(18)=	0.050	I(19)=	0.050	I(20)=	0.000

图 1-5 利用迭代公式(1.9)计算的积分值(I_{20}到 I_1)

结果发现,各个积分值都非常稳定。我们也可以用之前的分析方法来看看误差在迭代过程中是如何传递的。假定 $n=20$ 的时候,初值 I_{20}的误差为 ε,则到了 $n=0$ 时,I_0 的误差变为$\pm\varepsilon/20!$,即迭代步数越多,误差反而越小。由此我们发现,计算思路对计算结果有着巨大影响,在数值计算过程中,首先要做的是估计误差,分析计算结果的可靠性。

第二章　非线性方程寻根与函数优化

这一章我们来学习如何寻找一个方程的根，这原本是一个数学问题，但在物理学上却有着广泛的应用。比如质点系的平衡位置、分子的最优结构等，与这些物理问题相关的方程或方程组往往非常复杂，传统的解析求解已经力不从心，需要借助数值计算的手段来研究这些问题，寻找答案。

在介绍算法以前，先准备两个源代码文件 Main. f90 和 Comphy_Findroot. f90，其中 Main. f90 的功能是启动程序，定义求根方程，并调用求根模块来计算方程的根。初始内容如下，以后会不断完善。

```
program main
  use Comphy_Findroot      ! 导入方程求根计算模块
  implicit none
  real*8 :: x              ! 方程的自变量
  integer :: imethod,niter
  external:: myfunc        ! 定义函数 f(x),待求根方程即为 f(x)=0

  write(*,"(/,'Input initial x: ',$)"); read(*,*) x              ! 设定自变量初值
  write(*,"(/,'Input iteration number: ',$)"); read(*,*) niter   ! 设定迭代步数
  do
    print *
    print "(a)","All methods"
    print *,"1: Bisection method"        ! 二分法
    print *,"2: Jacobi Iteration"        ! Jacobi 迭代法
    print *,"3: Post Acceleration"       ! 事后加速算法
    print *,"4: Atiken Acceleration"     ! Atiken 加速算法
    print *,"5: Newton Iteration"        ! Newton 迭代法
    write (*,"(/,'Select method: ',$)"); read(*,"(i8)") imethod     ! 选择方法
    print *

    select case(imethod)
! ==========在这里调用具体的求根程序
! ==========
    case default
        exit
    end select
  end do
end program
```

Comphy_Findroot.f90 文件则是用来执行具体的求根算法，初始内容如下。

```
module Comphy_Findroot
implicit none
contains
! 在这里写具体的求根算法
end module
```

第1节 二 分 法

二分法是最简单的方程求根方法，它通过不断细分区间来搜索方程的根。其具体思路如下。

(1) 将需要求根的方程转化为 $f(x)=0$ 的形式，即通过移项将方程右边归零。这样，只要找到一个 x，使得函数 $f(x)$ 的值等于 0，那么方程的根也就找到了。

(2) 划定搜索区间 $[a,b]$，并计算区间边界处的函数值 $f(a)$ 和 $f(b)$，如果 $f(a)f(b)>0$，则认为区间中没有根存在，需要重新定义区间 $[a,b]$（仅适用于单数根情形）。反之，如果 $f(a)f(b)<0$，则可以确定区间中有根存在，后面需要做的就是不断细分区间。

(3) 在区间 $[a,b]$ 中取中点 $x_0=(a+b)/2$，如图 2-1 所示，并计算函数值 $f(x_0)$，如果 $f(a)f(x_0)<0$，则认为根在左半区间 $[a, x_0]$，反之，根在右半区间 $[x_0,b]$（同样仅适用于单根情形）。

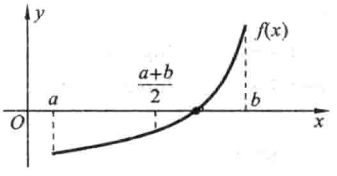


图 2-1 二分法示意图

(4) 如果根在左半区间 $[a, x_0]$，再次取其中点 $x_1=(a+x_0)/2$，并计算函数值 $f(x_1)$，如果 $f(a)f(x_1)<0$，则根在区间 $[a, x_1]$，反之，根在区间 $[x_1, x_0]$。如果根在右半区间 $[x_0,b]$，取其中点 $x_1=(x_0+b)/2$，并计算函数值 $f(x_1)$，如果 $f(x_0)f(x_1)<0$，则根在左半区间 $[x_0, x_1]$，反之，根在区间 $[x_1, b]$（仅适用于单根情形）。

(5) 不断重复第 4 步，直至计算结果满足收敛条件（x_k 表示 k 次迭代时的 x 值，ε 表示误差标准）

$$|x_k-x_{k-1}|<\varepsilon \quad 或 \quad |f(x_k)|<\varepsilon \tag{2.1}$$

在模块文件 Comphy_Findroot.f90 中加入下面的二分法程序代码，参数 func 是一个子程序，定义了求根方程。niter 则指定了二分法的迭代步数，x 则为迭代初值（区间中心位置）。

```
subroutine Bisection(func,niter,x)
  implicit none
  real*8,intent(inout):: x
```

```
integer,intent(in):: niter
integer:: iter
real*8:: xup,xlow,fx,flow,fup
interface
  subroutine func(x,fx,dfx,gx,dgx)
    real*8,intent(in) :: x
    real*8,intent(out) :: fx
    real*8,intent(out),optional :: dfx,gx,dgx
  end subroutine
end interface

xlow=x-1.5d0; xup=x+1.5d0          ! 根据迭代初值设置二分法区间(宽度 3.0)
call func(xlow,flow); call func(xup,fup)
if (flow*fup > 0.0d0) then          ! 如果函数 f(x)在上下边界同号,则返回
  print *,"There is no root in the interval!"; return
end if

print "(a)","iter   x   f(x)   |dx|"      ! dx 即为当前区间宽度
do iter=1,niter
   x=(xlow+xup)/2.0d0
   call func(x,fx);
   print "(i5,3f10.3)",iter,x,fx,abs(xup-xlow)
   if (fx*flow < 0.0d0) then     ! 根据中点的函数值 f(x)来重新定义新的区间
      xup=x; fup=fx
   else
      xlow=x; flow=fx
   end if
end do
end subroutine
```

主程序中需要加入下面的语句以调用二分法。

```
select case(imethod)
case (1)
      call Bisection(myfunc,niter,x)     ! 调用二分法求根
case default
      exit
end select
```