JAVA 技术丛书

# Java 数值方法

## Java Number Cruncher

### The Java Programmer's Guide to Numerical Computing

[美] Ronald Mak 著

张葵葵 骆 振 熊德慧 等译

PRENTICE HALL PTR

PHEI

電子工業出版社
Publishing House of Electronics Industry
http://www.phei.com.cn

Java 技术丛书

# Java 数值方法

## Java Number Cruncher

The Java Programmer's Guide to Numerical Computing

[美] Ronald Mak 著

张葵葵 骆 振 熊德慧 等译

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

## 内容简介

本书深入讲解了有关Java的数值计算，并介绍了常见的Java数值方法及计算特性。全书分为4个部分，共16章，首先对Java的数据表示、数据类型及相关标准进行了详细的介绍。然后，作者分析了数值求根、数值求和、插值、估计、数值积分、求解微分方程等常见的数值计算；讨论了矩阵运算软件包，并讲解了各种矩阵运算的方法。最后，本书给出了一些有趣的数学计算实例。全书内容新颖，举例丰富，而且应用了大量生动的交互式图形程序。

本书适合于有兴趣学习Java数值方法的程序设计者，同时对于从事数值计算及相关工作的人员也具有很好的参考价值。



版权贸易合同登记号　图字：01-2003-0592

**图书在版编目（CIP）数据**

Java数值方法/（美）马克（Mak, R.）著；张葵葵等译. -北京：电子工业出版社，2004.1
（Java技术丛书）
书名原文：Java Number Cruncher: The Java Programmer's Guide to Numerical Computing
ISBN 7-5053-9309-X

Ⅰ.J... Ⅱ.①马... ②张... Ⅲ.JAVA语言-数值计算 Ⅳ.TP312

中国版本图书馆CIP数据核字（2003）第101386号

责任编辑：冯小贝
印　　刷：北京兴华印刷厂
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　邮编：100036
经　　销：各地新华书店
开　　本：787×980　1/16　　印张：27.5　字数：616千字
印　　次：2004年1月第1次印刷
印　　数：4000册　定价：43.00元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换；若书店售缺，请与本社发行部联系。
联系电话：（010）68279077。质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

# 译 者 序

作为一门程序设计语言，Java 在设计网络程序、企业软件、数据库系统、计算机游戏及图形处理等领域得到了广泛的应用，其强大的功能受到越来越多的程序设计者的青睐。关于这些方面的著作精彩纷呈、数不胜数，然而有关数值计算的著作却较少出现。事实上，人们可以通过 Java 进行大量的数值计算，译者正是想通过翻译这本书，使读者领略到 Java 的数值计算功能。

在这本著作中，作者解释了如何发现及避免应用程序中可能导致计算误差的编程错误。作为将纯数学转换为计算机数学的权威专家，作者通过简单、轻松的写作风格，解释了如何使用经常被忽视的 Java 计算特性。书中为读者介绍了数值求和的实用安全算法，求解方程的根，插值法和近似法，数值积分，求解微分方程，矩阵运算，以及怎样求解联立方程组。同时还给出了一些有趣的应用，例如在素数中寻找模式，随机数的生成，π 的数千位有效数的计算，以及不规则地生成分形图像，等等。

本书分为 4 个部分，共 16 章。其中第一部分对 Java 的数据表示、数据类型及相关标准进行了详细介绍。第二部分讲解了常见的求根运算、数值求和、插值、数值积分、求解微分方程等数值计算。第三部分介绍了矩阵软件包，并分析了各种矩阵运算的方法。第四部分则给出了一些有趣的数学计算实例。在每一章的结尾，作者给出了该章的主要参考文献，读者可以根据需要进行阅读和学习。

本书的第一部分由骆振翻译，第二部分的第 4 章、第 5 章由熊德慧翻译，第 6 章～第 16 章由张葵葵翻译。参与翻译工作的还有胡洪志、司功闪、彭程、李波，全书最后由张志龙进行了校对与统稿。

由于译者水平有限，书中难免有错误及不妥之处，敬请读者批评指正。

# 前　言

在我们以往所见到的 Java 编程语言中，进行数值运算依然使用的是“+”、“-”、“*”、“/”和“%”这样的运算符。这在今天来说是难以置信的，现在的 Java 编程不仅应用于网页、图形、娱乐软件，而且应用于数据库系统、计算机游戏等。

编写这本书的目的是提醒现在的编程人员，尤其是 Java 编程人员，称为“number crunching”的计算机的确具有强大的数值计算功能。实际上，数值计算是大多数程序的基础，例如大多数的图形应用或者交互式计算机游戏都需要数据的处理，哪怕只是少量数据的处理。当然，有关科学、数学、统计和财经的程序则更加偏重于数值计算。

所以，本书适合于不仅了解标准的 API 应用——JFC、RMI、JSP、EJB、JDBC，而且熟悉数值运算的 Java 编程人员。我们可能永远不知道什么时候舍入误差将影响计算结果。有时候，我们将课本上抄来的“正确”公式编入程序后却得到错误的答案。

撰写这本书的另一个原因是作者喜欢将纯数学和计算机数学分成两部分。一方面数学是一个严格、抽象的世界，在这个世界中，绝对地证明一个计算的正确性是可能的。而从另一方面来说，计算机世界中的计算快速且准确。因此，数学家和计算机科学家可以一起设计出更有效的方法，能够使计算机在绝大多数情况下进行数学运算时得到正确的答案。

这本书是用来介绍数值运算的，虽然其中介绍了如何将许多重要的数值算法编写到 Java 程序中，但本书并不是一本关于数值方法或数值分析的著作。我们将检验这些算法，以了解它们是如何工作的以及为什么是有效的。同时，书中也给出了许多交互式程序和图形程序的算法。在讲解了如何避免一些浮点数和整数运算的陷阱之后，我们将研究求解 $x$ 方程、进行插值和积分、求解微分方程及线性系统方程的程序。

当然，数值计算不是这本书的全部内容，作者还用几章介绍了其他一些相关主题，包括计算 $\pi$ 的数千个数位，使用不同的方法产生随机数，在素数中寻找模式，以及生成美丽的分形图像。

本书的所有交互式程序既可以作为 applet（小程序）运行，又可以作为独立程序运行。作者已经在一些平台上测试了这些程序，这些平台包括运行在 Windows、Linux 和 Solaris 上的 Netscape 4.7 浏览器，运行在 PC 上的 Microsoft Internet Explorer 6.0，以及运行在 Macintosh 上的 Microsoft Internet Explorer 5.1。作者也已

经在 Windows 98 上利用 JDK 1.2、1.3 和 1.4 测试了相关的程序。当然，我们不能保证所有的程序都能工作得很好，但所有程序的源代码以及编译和运行的说明都是可以下载的。

作者严格地编写了书中的所有程序。读者可以按照任意方式使用源代码，但注意这不是经过完全测试的商业质量的代码。

## 怎样下载源代码

这本书所有程序的源代码以及怎样编译和运行的说明，都可以从 Prentice PTR 的网站 http://www.phptr.com/mak 上下载。

读者也可以从作者的网站 http://www.apropos-logic.com/nc/上下载源代码和相关说明，并且可以运行所有的 applet。

# 目　录

## 第一部分　正确的运算公式却导致出现错误的结果


**第 1 章　浮点数而非实数** …… 3
1.1　舍入误差 …… 3
1.2　误差放大 …… 5
1.3　实数和浮点数的比较 …… 8
1.4　精度和准确度 …… 10
1.5　浮点运算不遵守代数定律 …… 11
1.6　整数运算的情况 …… 16
参考文献 …… 17

**第 2 章　整数类型的情况分析** …… 18
2.1　整数类型及其运算 …… 18
2.2　带符号量值法与补码的比较 …… 19
2.3　纯数学中的整数与 Java 中整数的比较 …… 20
2.4　封装类 …… 24
2.5　整数的除法和求余 …… 26
2.6　整数指数 …… 27
参考文献 …… 29

**第 3 章　浮点标准** …… 30
3.1　浮点格式 …… 30
3.2　非规范化数 …… 33
3.3　分解浮点数 …… 34
3.4　浮点运算 …… 52
3.5　±0、±∞和 NaN …… 54
3.6　无异常 …… 58
3.7　重新分析舍入误差 …… 59
3.8　严格或非严格浮点运算 …… 61
3.9　计算机的最小正数值$\varepsilon$ …… 62

3.10 误差分析 …… 64
参考文献 …… 64

## 第二部分 迭代计算

**第4章 数列求和** …… 69
4.1 求和的实质——大小问题 …… 69
4.2 Kahan 求和算法 …… 78
4.3 任意顺序的数列求和 …… 82
4.4 不同符号加数的求和 …… 87
4.5 计算的内部细节 …… 91
4.6 求和算法总结 …… 100
参考文献 …… 101

**第5章 求方程的根** …… 102
5.1 解析解与计算机解的比较 …… 102
5.2 函数关系式 …… 103
5.3 对分算法 …… 106
5.4 试位算法 …… 117
5.5 改进的试位算法 …… 124
5.6 割线算法 …… 128
5.7 牛顿算法 …… 134
5.8 不动点迭代 …… 141
5.9 重根的双重麻烦 …… 150
5.10 求根算法的比较 …… 151
参考文献 …… 153

**第6章 插值和逼近** …… 154
6.1 幂级数和牛顿级数 …… 154
6.2 多项式插值函数 …… 155
6.3 差商 …… 157
6.4 构造插值函数 …… 158
6.5 最小平方线性逼近 …… 166
6.6 构造回归线 …… 167
参考文献 …… 175

**第 7 章　数值积分** …… 176
7.1　回到基础知识 …… 176
7.2　梯形算法 …… 177
7.3　辛普森算法 …… 184
参考文献 …… 189

**第 8 章　微分方程的数值解** …… 190
8.1　回到基础知识 …… 190
8.2　微分方程类 …… 192
8.3　欧拉算法 …… 196
8.4　预估校正算法 …… 205
8.5　四阶龙格-库塔算法 …… 210
参考文献 …… 214

## 第三部分　矩阵软件包

**第 9 章　基本的矩阵操作** …… 217
9.1　矩阵 …… 217
9.2　方阵 …… 227
9.3　单位矩阵 …… 230
9.4　行向量 …… 230
9.5　列向量 …… 234
9.6　图形转换矩阵 …… 237
9.7　三维空间中立方体的旋转 …… 240
参考文献 …… 256

**第 10 章　求解线性系统方程** …… 257
10.1　高斯消元法 …… 257
10.2　高斯消元法存在的问题 …… 259
10.3　部分变换 …… 260
10.4　标定 …… 261
10.5　LU 分解 …… 262
10.6　迭代的改进 …… 265
10.7　求解线性系统方程类 …… 265
10.8　LU 分解的测试程序 …… 274
10.9　多项式回归 …… 277
参考文献 …… 286

**第 11 章　矩阵求逆、行列式和条件数** …… 288
11.1　矩阵的行列式 …… 288
11.2　矩阵的逆 …… 288
11.3　矩阵的范数和条件数 …… 289
11.4　逆矩阵类 …… 289
11.5　希尔伯特矩阵 …… 292
11.6　求解算法的比较 …… 295
参考文献 …… 299

## 第四部分　计算的乐趣

**第 12 章　大数** …… 303
12.1　大整数 …… 303
12.2　一个非常大的素数 …… 304
12.3　大整数和密码技术 …… 308
12.4　大十进制数 …… 309
12.5　大十进制函数 …… 309
参考文献 …… 321

**第 13 章　计算π** …… 323
13.1　π 值的估计与 Ramanujan 的公式 …… 323
13.2　生成π的反正切公式 …… 328
13.3　生成 10 亿位数 …… 336
参考文献 …… 345

**第 14 章　生成随机数** …… 347
14.1　伪随机数 …… 347
14.2　均匀分布的随机数 …… 348
14.3　正态分布的随机数 …… 348
14.4　指数分布的随机数 …… 360
14.5　蒙特卡洛、Buffon 针算法与 π …… 366
参考文献 …… 373

**第 15 章　素数** …… 374
15.1　Eratosthenes 筛选法和因子分解 …… 374
15.2　同余与模运算 …… 378
15.3　Lucas 测试 …… 382

15.4 Miller-Rabin 测试 …… 389
15.5 联合素数测试 …… 397
15.6 素数生成 …… 399
15.7 素数模式 …… 404
参考文献 …… 406

第 16 章 分形 …… 408
16.1 不动点迭代和轨迹 …… 408
16.2 分支与实函数 $f(x)=x^2+c$ …… 409
16.3 茹利亚集与复变函数 $f(z)=z^2+c$ …… 414
16.4 复平面的牛顿算法 …… 422
16.5 芒德布罗集 …… 424
参考文献 …… 428

# 第一部分

## 正确的运算公式却导致出现错误的结果

将数学或者统计学课本中的原有公式编写到程序中，几乎必然会导致错误的结果。这本书的第一部分将介绍有关基本数值运算的一些陷阱。

第 1 章讨论通常的浮点数以及它们与数学上的实数的区别。如果不理解这些区别（如舍入误差的发生），或是不遵守代数中的一些基本规则，那么将导致计算出错。

第 2 章着眼于看上去不会有任何问题的整数类型。这种类型与数学上的整数表现不完全一样，如加、减、乘这样的数学运算是在时钟表盘上而不是在数线上进行的。

最后，第 3 章分析了 Java 如何实现其浮点型。其中查看了 IEEE 754 浮点标准，并讨论了 Java 满足其规定的程度。

# 第 1 章　浮点数而非实数

当早期的编程语言 FORTRAN 和 ALGOL 的设计者们把其中一种数据类型命名为 REAL 时，仅仅是为了方便或者是他们认为这样命名十分正确吗？

那么，Java 的 float 类型与数学体系的实数有哪些相似之处呢？就此而言，int 类型与数学集合中的整数（所有整数）又有哪些相似之处呢？我们知道，诸如溢出和舍入误差这样的小麻烦，实际上可能会导致更多的潜在问题，那么，还会出现什么样的陷阱呢？

## 1.1　舍入误差

考虑一般的分数$\frac{1}{2}$、$\frac{1}{3}$、$\frac{1}{4}$、$\frac{1}{5}$、$\frac{1}{6}$和$\frac{1}{7}$。在十进制或者以 10 为基数的数字体系中，可把$\frac{1}{2}$、$\frac{1}{4}$和$\frac{1}{5}$分别表示为 0.5、0.25 和 0.2。在十进制中，其分母可以整除 10 的幂的任意分数，都可以表示为小数。但分母 3 不能整除 10 的幂，所以$\frac{1}{3}$的小数部分是无限重复的，即 0.333 3...。分母 6 同样不能整除 10 的幂，所以$\frac{1}{6}$是 0.166 66...。分母 7 同样也是，因此$\frac{1}{7}$是 1.142 857 142 857...，其中最后 6 位数字无限循环出现。

像大多数现代计算机一样，Java 虚拟机使用二进制或者以 2 为基数的数字体系。将这些分数表示为以 2 为基数的形式，其结果会怎样呢？程序 1.1 打印并对一些这样的值进行了求和。

**程序清单 1.1　分数**

```
package numbercruncher.program1_1;

/**
 * PROGRAM 1-1: Fractions
 *
 * Print and sum the values of the fractions 1/2, 1/3, 1/4, and 1/5
 * to look for any roundoff errors.
 */
public class Fractions
{
    private static final float HALF    = 1/2f;
    private static final float THIRD   = 1/3f;
    private static final float QUARTER = 1/4f;
    private static final float FIFTH   = 1/5f;
```

```
    private static final float SIXTH   = 1/6f;
    private static final float SEVENTH = 1/7f;

    private static final int FACTOR = 840;

    public static void main(String args[])
    {
        System.out.println("1/2 = " + HALF);
        System.out.println("1/3 = " + THIRD);
        System.out.println("1/4 = " + QUARTER);
        System.out.println("1/5 = " + FIFTH);
        System.out.println("1/6 = " + SIXTH);
        System.out.println("1/7 = " + SEVENTH);

        float sum = 0;
        System.out.println();

        for (int i = 0; i < FACTOR; ++i) sum += HALF;
        System.out.println("1/2 summed " + FACTOR + " times = " + sum +
                           " (should be " + FACTOR/2 + ")");

        sum = 0;
        for (int i = 0; i < FACTOR; ++i) sum += THIRD;
        System.out.println("1/3 summed " + FACTOR + " times = " + sum +
                           " (should be " + FACTOR/3 + ")");

        sum = 0;
        for (int i = 0; i < FACTOR; ++i) sum += QUARTER;
        System.out.println("1/4 summed " + FACTOR + " times = " + sum +
                           " (should be " + FACTOR/4 + ")");

        sum = 0;
        for (int i = 0; i < FACTOR; ++i) sum += FIFTH;
        System.out.println("1/5 summed " + FACTOR + " times = " + sum +
                           " (should be " + FACTOR/5 + ")");

        sum = 0;
        for (int i = 0; i < FACTOR; ++i) sum += SIXTH;
        System.out.println("1/6 summed " + FACTOR + " times = " + sum +
                           " (should be " + FACTOR/6 + ")");

        sum = 0;
        for (int i = 0; i < FACTOR; ++i) sum += SEVENTH;
        System.out.println("1/7 summed " + FACTOR + " times = " + sum +
                           " (should be " + FACTOR/7 + ")");
    }
}
```

输出：

```
1/2 = 0.5
1/3 = 0.33333334
1/4 = 0.25
1/5 = 0.2
1/6 = 0.16666667
1/7 = 0.14285715

1/2 summed 840 times = 420.0 (should be 420)
1/3 summed 840 times = 279.99915 (should be 280)
1/4 summed 840 times = 210.0 (should be 210)
1/5 summed 840 times = 167.99858 (should be 168)
1/6 summed 840 times = 139.99957 (should be 140)
1/7 summed 840 times = 120.001114 (should be 120)
```

在程序中我们给一些数字附加上“f”，使其成为单精度浮点数。通过这种方式，1/2f 采用浮点运算而不是整数运算。

在程序的第一组输出行中，$\frac{1}{2}$、$\frac{1}{4}$和$\frac{1}{5}$的结果正好等于实际值。$\frac{1}{3}$的结果有一些舍入误差——它最右边的数字获得进位，但误差相当小。同样，$\frac{1}{6}$和$\frac{1}{7}$也存在小舍入误差。

从程序的第二组输出行中我们可以看出，当使用即使是小误差的数进行累加时将会出现的情况。虽然$\frac{1}{3}$的结果是一个正误差，但它的累加结果却是一个负误差。实际上，最初的$\frac{1}{5}$存在一个小的隐藏误差，所以累加结果是一个舍去误差，即负误差。$\frac{1}{6}$的累加结果同样是一个舍去误差，$\frac{1}{7}$的累加结果则是一个进位误差，即正误差。但是很明显，$\frac{1}{2}$和$\frac{1}{4}$没有误差，它们的分母正好是 2 的幂：$\frac{1}{2}=2^{-1}$、$\frac{1}{4}=2^{-2}$。

## 1.2 误差放大

在查看一个有关舍入误差的例子之前（这个例子不涉及任何循环），让我们先定义几个术语：

**定义：绝对误差=|计算值-正确值|**

**定义：相对误差=$\frac{\text{绝对误差}}{\text{正确值}}$**

**定义：误差百分比=相对误差×100%**

相对误差是绝对误差相对正确值的比率。我们可以通过比较不同误差的相对误差来对其进行有意义的比较。误差百分比是将一个相对误差表示为百分数。

例如，我们来看应该怎样正确表示$\frac{1}{2}$。分数$\frac{10\,000\,001}{20\,000\,000}$应该非常接近$\frac{1}{2}$，它的误差仅为$\frac{1}{20\,000\,000}$。对这个差值求倒数，我们得到差值的分母，即 20 000 000。公式形式为

$$\frac{1}{\left(\frac{10\,000\,001}{20\,000\,000}\right)-\left(\frac{1}{2}\right)}=\frac{1}{\left(\frac{1}{20\,000\,000}\right)}=20\,000\,000$$

我们假设所进行计算的正确结果应为$\frac{1}{2}$，但得到的结果却为$\frac{10\,000\,001}{20\,000\,000}$，那么其绝对误差为$\frac{1}{20\,000\,000}$。

程序 1.2 演示了这样的运算，其中变量 a 等于$\frac{10\,000\,001}{20\,000\,000}$，而变量 b 等于$\frac{1}{2}$。请参见程序清单 1.2。

**程序清单 1.2　舍入误差**

```
package numbercruncher.program1_2;

/**
 * PROGRAM 1-2: Roundoff Errors
 *
 * Demonstrate how a tiny roundoff error
 * can explode into a much larger one.
 */
public class RoundoffErrors
{
    public static void main(String args[])
    {
        float denominator = 20000000;
        float a           = 10000001/denominator;
        float b           = 1/2f;
        float diff1       = Math.abs(a - b);
        float pctError1   = 100*diff1/b;

        float inverse     = 1/diff1;
        float diff2       = Math.abs(inverse - denominator);
        float pctError2   = 100*diff2/denominator;

        System.out.println("         a = " + a);
        System.out.println("         b = " + b);
        System.out.println("     diff1 = " + diff1);
        System.out.println("pctError1 = " + pctError1 + "%");

        System.out.println();
        System.out.println("  inverse = " + inverse);
        System.out.println("     diff2 = " + diff2);
        System.out.println("pctError2 = " + pctError2 + "%");
```