浙江大学

博士学位论文

论文题目 关于蒙特卡罗及拟蒙特卡罗方法的若干研究

作者姓名 雷桂媛

指导教师 王兴华教授

学科(专业) 计算数学

所在学院 理学院

提交日期 2003年4月

浙江大学

博 士 学 位 论 文

论文题目 关于蒙特卡罗及拟蒙特卡罗方法的若干研究

作者姓名 雷 桂 媛

指导教师 王 兴 华 教 授

学科(专业) 计 算 数 学

所在学院 理 学 院

提交日期 2003 年 4 月

关于蒙特卡罗及拟蒙特卡罗方法的若干研究

博士论文：雷桂媛

指导教师：王兴华教授

浙江大学计算数学专业

致谢

首先感谢我的导师王兴华教授，是他指引我进入蒙特卡罗和拟蒙特卡罗研究领域，并且在我的研究阶段给了我很多帮助。

同时我要感谢下述所有的计算数学教研组的成员。从我们组定期举行的讨论班上我学到了很多知识。

王兴华, 李冲, 江金生, 郑士明, 程晓良, 韩丹夫, 吴庆标, 孙方裕, 黄正达, 陈明飞, 叶兴德, 朱建新, 杨士俊, 梁克维, 李大明, 梁仙红, 王何宇, 宓湘江, 金中秋, 崔峰和谢聪聪

另外还有很多人给了我许多帮助，尤其是那些让我有机会将所学的内容应用于实践的人。金剑秋介绍我参加了关于预测的编程工作，应用遗传程序设计与自适应拟蒙特卡罗结合的方法取得了很好的效果。Anders Karlsson教授与我讨论了光在组织中传播的逆问题，使我得以将自己的优化方法用于解决这一实际问题。

最后，我要感谢我的家人和朋友。他们付出的关心和支持让我实现了求学的梦想，最终得于成为科研工作者。我的丈夫何江平博士一直鼓励我勤奋学习，在我准备第一篇论文时我们在他的实验室一起工作到深夜，那是一段令人难忘的时光。在这篇博士学位论文的准备中他也给了很多帮助，还教我画了其中的一些图。

序言

蒙特卡罗方法是研究多体复杂系统以及不确定性过程的强有力的工具。J. Dongarra和F. Sullivan[1] 在杂志"Computing in Science & Engineering"上发表文章评述了二十世纪对科学和工程计算的发展和实践最具影响力的十大算法[2]，蒙特卡罗方法榜上有名。

蒙特卡罗方法的前身Metropolis算法是由J.von Neumann，S. Ulam和N. Metropolis在第二次世界大战末为了研究裂变物质中子的扩散而提出的。"蒙特卡罗"是在1947年由N. Metropolis命名，并于1949年在N. Metropolis和S. Ulam合作的一篇文章[3]正式使用。这些科学家提出蒙特卡罗方法的初衷是用于攻克专门的物理数值模拟问题。起初人们对蒙特卡罗方法的兴趣不大，但后来这一方法很快就被应用于各个令人惊讶的方向，包括函数值极小化，计算几何，组合计数等。时至今日，在深刻的理论支持下，从物理模拟到计算复杂性的基石，与Metropolis相关的主题已经涵盖了计算科学的整个领域。而随着拟随机序列的出现，蒙特卡罗方法也已经发展到拟蒙特卡罗方法。

我们经常要面临一些有非常高维数的问题，或有很多分支路径(path)而每一个分支路径都按一定的概率发生的过程。因为蒙特卡罗方法是对问题进行采样，所以它不是严格的、精确的解法。但是我们能用相对于问题的维数而言相当小的样本得到近似精确解。事实上，虽然蒙特卡罗和拟蒙特卡罗方法可能比较耗时，但它们是解决高维问题唯一切实可行的方法。幸运的是，随着现代计算机技术的飞速发展，我们可以计算越来越复杂的问题。所以蒙特卡罗和拟蒙特卡罗算法在新的21世纪会有更光明的前景。鉴于此，我选择了蒙特卡罗和拟蒙特卡罗方法的若干主题作为研究方向。

如果您有任何意见和建议，请联系我。

雷桂媛 <guiyuanlei@hotmail.com>

2003年4月于杭州

摘要

此论文主要阐述蒙特卡罗和拟蒙特卡罗方法在积分、优化和模拟方面的应用的若干主题。

第1章和第2章是关于蒙特卡罗和拟蒙特卡罗方法的预备知识。第1章介绍了蒙特卡罗和拟蒙特卡罗积分的误差估计并阐述了拟蒙特卡罗方法的优势，同时介绍了拟蒙特卡罗的标准优化方法，最后介绍了蒙特卡罗方法的起源--Metropolis模拟方法。因为随机数发生器是蒙特卡罗和拟蒙特卡罗方法的核心之一，所以第2章介绍了伪随机数和拟随机数发生器。此章还介绍了如何生成服从一定概率分布函数的随机数。

第3章提出了B样条光滑拒绝抽样方法。第2章介绍的标准拒绝抽样方法其实跟特征函数的蒙特卡罗积分有密切的关系。而由于特征函数的不连续性，蒙特卡罗积分应有的误差精度就达不到，拒绝抽样的效果也就受到影响。我们用B样条磨光技术在不改变积分值的前提下磨光特征函数，用可微的权重函数代替特征函数，提高了采样的质量。将B样条光滑拒绝方法用于重要抽样估计，数值例子显示拟蒙特卡罗积分的精度重新达到了$O(N^{-1})$的阶，而对于蒙特卡罗积分，采用B样条光滑重要抽样，其精度也比标准积分的精度$O(N^{-\frac{1}{2}})$好。由此间接证明了B样条光滑拒绝抽样比标准拒绝抽样有效得多。

第4章是关于蒙特卡罗积分的，得出了用于多重积分的精细对偶变数蒙特卡罗(fine antithetic variables Monte Carlo，简称FAMC)方法的误差估计式。对维数是s的二阶导数连续的函数来说，FAMC方法理论误差的阶是$O(N^{-(\frac{1}{2}+\frac{2}{s})})$。我们同时也讨论了对偶变数蒙特卡罗积分(antithetic variable Monte Carlo，简称AMC)方法。对次数不高于2的多变量函数，AMC方法其误差阶是$O(N^{-\frac{1}{2}})$，但其系数比原始蒙特卡罗积分(MC)方法的误差阶的系数小。我们用C语言实现了并行计算程序，数值实验结果与理论结果吻合得很好。

第5章介绍了自适应拟蒙特卡罗全局优化(AQMC)方法。AQMC算法在不可微函数的蒙特卡罗优化算法上大大前进了一步。首先改进了局部搜索算法，使得搜索方向，搜索半径和搜索所计算的函数值个数根据已有的搜索结果而自适应调整。其次引进了种群演化的概念，根据演化度产生新个体，保证搜索的全局性。因为在搜索过程中会根据搜索中间结果进行调整，所以此方法不仅可以加快搜索速度，而且可以平衡全局和局部搜索需求。

第6章结合遗传程序设计方法和自适应拟蒙特卡罗优化方法用于预测问题。在现实生活中存在许多随时间而变化的复杂系统和现象，人们通常需要根据动态系统的观测数据建立合理的微分方程模型(动力学模型)，为系统分析、设计和未来状态的预报提供依据。我们用遗传程序设计方法优化常微分方程右端的函数，用自适应拟蒙特卡罗优化方法优化函数中的系数。在预测杭州市地区全社会用电量的实际应用中，所编写的程序取得了很好的效果。

第7章是蒙特卡罗模拟和优化的结合。首先我们介绍了光在组织中传播的蒙特卡罗模拟的完整过程，解释了如何利用第2章中介绍的随机数生成方法根据实际问题产生随机数。然后用自适应拟蒙特卡罗优化方法来解决光的传播的逆问题。在这一章我们进一步探讨了如何根据实际问题的性质来平衡全局和局部搜索需求。

附录给出了相关程序的C语言代码。

Abstract

I have studied a couple of topics on Monte Carlo and quasi-Monte Carlo methods. This dissertation covers its applications in integration, optimization and simulation.

Chapter 1 and 2 are the basic knowledge to use Monte Carlo and quasi-Monte Carlo methods. Chapter 1 presents the error bounds of Monte Carlo and quasi-Monte Carlo integration methods. By comparing these two methods, we show the advantages of quasi-Monte Carlo method. We also introduce the standard Monte Carlo random search for optimization. The last but not least application is Metropolis algorithms which is the origin of Monte Carlo method. Because the random numbers generators are the key of Monte Carlo methods and quasi-Monte Carlo methods. Chapter 2 describes the pseudo-random number generators and quasi-random number generators. How to generate non-uniform random number from its distributed function is also introduced.

Chapter 3 introduces B-spline smoothed rejection sampling method. The standard rejection sampling method which is introduced in chapter 2 is closely related to the problem of quasi-Monte Carlo integration of characteristic functions, whose accuracy may be lost due to the discontinuity of the characteristic functions. We use B-spline smoothing technique to smooth the characteristic function without changing the integral quantity and get a differentiable weight function. The method considerably improves the quality of sampling points. We apply the B-spline smoothed rejection sampling method to importance sampling. Numerical experiments show that the error size $O(N^{-1})$ is regained by using the B-spline smoothed rejection method for quasi-Monte Carlo estimate. The error bound of Monte Carlo method using B-spline smoothed importance sampling is also better than that of the

standard Monte Carlo method. So the B-spline smoothed rejection sampling method is indirectly proved to be superior to the standard rejection sampling method.

Chapter 4 is about the Monte Carlo integration. We get a theoretical error of the fine antithetic variables Monte Carlo(FAMC) method for multidimensional integration. The error size of FAMC is $O(N^{-(\frac{1}{2}+\frac{2}{s})})$ for functions having second continuous derivative, where s is the dimension of the integrand. We also give the theoretical error result of antithetic variable Monte Carlo(AMC) method for multi-variable functions whose degree is no more than two. The constant before $O(N^{-\frac{1}{2}})$ is less than that of the MC method. We realize the parallel algorithm in C for the FAMC and AMC methods. The results of the numerical experiments coincide with the theoretical results very well.

Chapter 5 introduces adaptive monte carlo method(AQMC) for global optimization. AQMC algorithm progresses in the nondifferentiable optimization. First, we develop the local search such that the search direction, search radius and number of search points are adjusted according to the previous search result. Second, we introduce the ideas of population and generate new individuals according to population evolution degree. Because the search procedure will be adjusted according to the previous result, the method not only speeds up the random search but also balances the global and local demands (adaptive equalization).

Chapter 6 combines the genetic programming with AQMC optimization method to solve the prediction problems. There are many complex systems in real life. In order to analyze, design and predict the system, we often want to model the dynamic systems of ordinary differential equations according to the observed data. We use genetic programming to optimize the the right hand functions of the ordinary differential equations. Adaptive quasi-Monte Carlo optimization methods are used to optimize the coefficients of the functions. The program for the prediction of electrical power consumption of Hangzhou city shows that the hybrid method is powerful.

In Chapter 7, we combine the Monte Carlo simulation and optimiza-

tion. We first introduce the Monte Carlo simulation of light transport in tissue, explain how to generate the random number according to the practical problems using the transform method introduced in chapter 2. Then use the AQMC method to solve the inverse problem of light transport. We further discuss how to balance the global and local search demands in practical problems.

The C codes of some programs are given in the appendix.

目录

图形列表

表格列表

第一章 蒙特卡罗和拟蒙特卡罗方法的基本知识

蒙特卡罗和拟蒙特卡罗方法在数值方法中应用得很广泛，人们研究这些方法也已经很多年了。其中最常用的应用是蒙特卡罗积分，蒙特卡罗和拟蒙特卡罗方法同样广泛地应用于优化和模拟。因为蒙特卡罗方法简单直接，所以很容易应用。这些方法同样很实用，因为方法的精度仅依赖于问题复杂性最粗糙的刻划。这章主要介绍蒙特卡罗和拟蒙特卡罗方法的的一些基本知识。

1.1 蒙特卡罗积分

关于蒙特卡罗积分，Caflisch [4]在1998年给出了一个综述。勒贝格可积函数$f(\mathbf{x})$的积分可以表示成函数f在随机点上函数值的平均值或期望值。考虑在s维单位超立方体(unit cube)$I^s = [0,1]^s$上的积分，则有

$$I = \mathrm{E}[f(\mathbf{x})] = \int_{I^s} f(\mathbf{x})\mathrm{d}\mathbf{x} = \bar{f} \tag{1.1}$$

这里$\mathbf{x}$是在单位超立方体上的均匀分布的随机数向量。

原始蒙特卡罗估计(crude Monte Carlo(MC) estimator)是基于对积分的概率解释。若有一服从均匀分布的序列ξ_n，则有经验近似估计式：

$$M = \frac{1}{N}\sum_{k=1}^{N} f(\xi_k) \tag{1.2}$$

根据强大数定律(Strong Law of Large Numbers)[5]，此估计以概率1收敛，也就是：

$$\lim_{N\to\infty} M \to I. \tag{1.3}$$

另外，此近似式是无偏的，这就意味着对任意N，M的平均值就等于I。也就是：

$$\mathrm{E}[M] = I, \tag{1.4}$$

这里是在选择的点ξ_k上取平均值。

一般来说，蒙特卡罗积分误差定义成：

$$\epsilon_N = I - M \tag{1.5}$$

这样偏差(bias)是$E[\epsilon_N]$，均方根误差($rmse$)定义成：

$$\mathrm{E}[\epsilon_N^2]^{1/2}. \tag{1.6}$$

1.1.1 误差估计

中心极限定理(CLT)[5]描述了蒙特卡罗积分的误差的大小和统计性质。

定理 1.1.1 *对任意大的N，*

$$\epsilon_N \approx \sigma N^{-1/2}\nu \tag{1.7}$$

这里ν是标准正态分布随机变量($N(0,1)$)，常数$\sigma = \sigma[f]$是f的根方差(square root of the variance)，也就是说：

$$\sigma[f] = \Big(\int_{I^s}(f(\mathbf{x}) - I)^2 \mathrm{d}\mathbf{x}\Big)^{1/2}. \tag{1.8}$$

更精确的表达式是

$$\begin{aligned}\lim_{N\to\infty} \mathrm{Prob}(a < \tfrac{\sqrt{N}}{\sigma}\epsilon_N < b) &= \mathrm{Prob}(a < \nu < b) \\ &= \int_a^b (2\pi)^{-1/2}\mathrm{e}^{-x^2/2}\mathrm{d}x.\end{aligned} \tag{1.9}$$

这就是说蒙特卡罗积分的误差是以$O(N^{-1/2})$为阶，系数是被积函数f的根方差。另外，误差的统计分布大致是一个正态分布变量。与通常的数值分析结果相比较，这是一个统计结果，它并不提供一个绝对的误差上界，而认为误差是概率意义上的估计。另一方面，此估计是一个等式，所以又是严格的。

1.1.2 蒙特卡罗方法与网格方法的比较

很多人第一次接触蒙特卡罗方法时会惊讶于此方法的可行性。为什么随机排列会比网格(grid-based method)更好？可以从几个方面来回答这个问题。首先，可以比较蒙特卡罗和网格方法如辛普森方法(Simpson's rule)的收敛阶。对于s维问题的k阶网格求积方法的收敛阶是$O(N^{-k/s})$。而蒙特卡罗积分的收敛阶是$O(N^{-1/2})$，与维数没有关系。所以在高维积分时，当维数s满足$k/s < 1/2$时，蒙特卡罗积分方法比网格积分方法精度高。

然而，对于周期解析函数(analytic function)，k的值是无穷的，也就是说上述简单解释不成立。对蒙特卡罗方法的实用性的更强有力的解释是实际应用中很难对高维问题进行网格化。对s问题最简单的超立方网格至少需要2^s个点。对$s = 20$这样在实际问题中并不很大的维数，就要有1百万多个点。另外，要对高维问题进行网格细化就更不现实了，因为每一次细分网格都要增加2^s倍的点。相比于网格方法在高维问题中碰到的这些困难，蒙特卡罗方法的精度几乎与维数无关，而且点的增加都会提高误差精度。在实际应用时，取N个点，精度$O(N^{-1/2})$并不一定能达到。但经验显示，对于中等维数(比如$s = 20$)的中等复杂的问题，只要中等规模的N值，误差阶$O(N^{-1/2})$一般都能达到。

1.2 拟蒙特卡罗积分

回想蒙特卡罗积分，用N个随机点，其积分误差阶的平均数量级是$O(N^{-1/2})$。显然，肯定存在这样的N个点，使得误差的绝对值不大于平均值。如果我们能构造这样的点集，就可以对原有的方法进行较大的改进。拟蒙特卡罗积分方法就是为了发展数值积分而提出的。它致力于构造其积分误差比平均误差显著要好的那种点集。其积分形式与蒙特卡罗积分一致，只不过所用的随机数不一样。比如，对于单位超立方体积分区域$I^s = [0,1]^s$，我们定义拟蒙特卡罗估计如下：

$$\int_{I^s} f(\mathbf{x})\mathrm{d}\mathbf{x} \approx \frac{1}{N}\sum_{k=1}^{N} f(\mathbf{x}_k) \tag{1.10}$$

此式看起来象蒙特卡罗估计，但这里用的是确定性(deterministic)的点$\mathbf{x}_1$, $\mathbf{x}_2, \cdots, \mathbf{x}_N \in I^s$。我们可以明智地选择这些点使得式(1.10)中的误差得以变小。

因此，拟蒙特卡罗积分方法的基本思想是用精选的确定性点来取代蒙特卡罗积分中的随机数点。选择点的标准取决于数值问题本身。对于数值积分而言，选择的的标准很容易找到，而且由此引进了均匀分布序列(uniformly distributed sequence)和偏差(discrepancy)的概念。

1.2.1 偏差

偏差可以看成是对均匀分布的偏离的量化度量。

记P是包含点$\mathbf{x}_1,\cdots,\mathbf{x}_N \in I^s$的点集。对任意属于$I_s$的子集$B$，我们定义

$$A(B;P)=\sum_{n=1}^{N}\chi_B(\mathbf{x}_n), \tag{1.11}$$

这里χ_B是B的特征函数(characteristic function)。这样$A(B;P)$便是计算满足$\mathbf{x}_n \in B(1\le n\le N)$的那些点的个数的函数。如果$\mathcal{B}$是属于$I^s$的勒贝格测度子集的非空集合，则点集$P$的偏差表示成：

$$D_N(\mathcal{B};P)=\sup_{B\in\mathcal{B}}|\frac{A(B;P)}{N}-\lambda_s(B)|. \tag{1.12}$$

这里λ_s是s维勒贝格测度，$D_N(\mathcal{B};P)$总有$0\le D_N(\mathcal{B};P)\le 1$。作集合$\mathcal{B}$作适当的专门化，我们有三种最重要的偏差概念。我们记$\bar{I}^s=[0,1)^s$。

定义 1.2.1 (星偏差) *点集P的星偏差(star discrepancy)可以定义为$D_N^*(P)=D_N^*(\mathbf{x}_1,\cdots,\mathbf{x}_N)=D_N(\mathcal{J}^*;P)$，这里$\mathcal{J}^*$是$\bar{I}^s$的具有形式$\prod_{i=1}^{s}[0,u_i)$的所有子区间的集合。*

定义 1.2.2 (极化偏差) *点集P的极化偏差(extreme discrepancy)可以定义为$D_N(P)=D_N(\mathbf{x}_1,\cdots,\mathbf{x}_N)=D_N(\mathcal{J};P)$，这里$\mathcal{J}$是$\bar{I}^s$的具有形式$\prod_{i=1}^{s}[u_i,v_i)$的所有子区间的集合。*

定义 1.2.3 (等方性偏差) *点集P的等方性偏差(isotropic discrepancy)可以定义为$J_N(P)=J_N(\mathbf{x}_1,\cdots,\mathbf{x}_N)=D_N(\mathcal{C};P)$，这里$\mathcal{C}$是$I^s$的所有凸子集的集合。*

偏差的性质在[6]中有阐述。下面的误差估计分析也来自这本书。

1.2.2 Koksma-Hlawka不等式

我们讨论关于式(1.10)所计算的拟蒙特卡罗积分的误差估计的重要结果。我们从一维情形开始，其中一个经典的结果就是下面的Koksma不等式。

定理 1.2.1 *如果函数f在区间$[0,1]$上有有界变分(variation)$V(f)$，则对任意N个点$x_1,\cdots,x_N\in[0,1]$，有*

$$|\frac{1}{N}\sum_{k=1}^{N}f(x_k)-\int_0^1 f(x)\mathrm{d}x|\leq V(f)D_N^*(x_1,\cdots,x_N). \tag{1.13}$$

定理 1.2.2 *如果函数f在区间$[0,1]$上是连续的，则对任意N个点$x_1,\cdots,x_N\in[0,1]$，有*

$$|\frac{1}{N}\sum_{k=1}^{N}f(x_k)-\int_0^1 f(x)\mathrm{d}x|\leq \omega(f,D_N^*(x_1,\cdots,x_N)). \tag{1.14}$$

这里$\omega(f,\delta)=\sup\limits_{\substack{\|t\|<\delta\\ x,x+t\in D}}|f(x+t)-f(x)|$是连续模(modulus of continuity)。

将Koksma不等式延伸到多维情形，便有下面的Hlawka不等式，通常叫做Koksma-Hlawka不等式。

定理 1.2.3 *如果函数f在I^s上存在Hardy-Krause意义上的有界变分$V(f)$，则对任意N个点$\mathbf{x}_1,\cdots,\mathbf{x}_N\in I^s$，有*

$$|\frac{1}{N}\sum_{k=1}^{N}f(\mathbf{x}_k)-\int_{I^s}f(\mathbf{x})\mathrm{d}\mathbf{x}|\leq V(f)D_N^*(\mathbf{x}_1,\cdots,\mathbf{x}_N)). \tag{1.15}$$

上述所有定理的证明都能在书本[6]中找到。

含N个点的拟随机序列点集的偏差的阶是$O(N^{-1}(\log N)^{s-1})$，所以拟蒙特卡罗积分的误差阶是$O(N^{-1})$。

1.2.3 拟蒙特卡罗积分的优点

以拟蒙特卡罗积分选择确定性点的特性，我们有确定的误差估计。一

般来说，我们都能根据误差精度要求事先确定计算所要用的点数。另外，用同样多的函数值(一般来说计算函数值是最费时的)，拟蒙特卡罗方法比蒙特卡罗方法有更高的精度。所以，这两点—确定性和高精度性—说明拟蒙特卡罗积分优于蒙特卡罗积分。

1.3 蒙特卡罗优化

蒙特卡罗在数值分析上另外一个应用是全局优化。Rubinstein在文[7]中首次提出蒙特卡罗搜索可以用来找不可微函数的全局极值。而拟蒙特卡罗优化则是由Niederreiter在文[8]中介绍的。设f是$R^s(s \geq 1)$的有界子集E上的有界实函数。设$\mathbf{x}_1, \cdots, \mathbf{x}_N$是$E$上的点，则

$$m_N = \max_{1 \leq n \leq N} f(\mathbf{x_n}) \tag{1.16}$$

可看作函数f在E上的上确界M的近似值。定义E上的点$\mathbf{x}_1, \cdots, \mathbf{x}_N$的散度(dispersion)如下：

$$d_N = d_N(E) = \sup_{\mathbf{x} \in E} \min_{1 \leq n \leq N} d(\mathbf{x}, \mathbf{x}_n) \tag{1.17}$$

这里$d(\mathbf{y}, \mathbf{z}) = \max\limits_{1 \leq j \leq s} |y_j - z_j|$，$\mathbf{y} = (y_1, \cdots, y_s)$, $\mathbf{z} = (z_1, \cdots, z_s)$是$R^s$上的向量。Niederreiter在文[8]中证明如下定理

$$M - m_N \leq \omega(d_N) \tag{1.18}$$

这里$\omega(t) = \sup\limits_{\substack{\mathbf{x},\mathbf{y} \in E \\ d(\mathbf{x},\mathbf{y}) \leq t}} |f(\mathbf{x}) - f(\mathbf{y})|$, $t \geq 0$是连续模(modulus of continuity)。

如果f是E上的连续函数，上述的优化方法是收敛的。为了加快搜索速度和保证搜索的全局性，我们在第5章给出自适应拟蒙特卡罗全局优化(Adaptive Quasi-Monte Carlo，AQMC)方法。我们对如何平衡局部搜索和全局搜索作了一些探讨。

1.4 蒙特卡罗模拟的Metropolis算法

蒙特卡罗模拟则在物理领域有广泛的应用，如液体蒙特卡罗模拟方法[9]，热学方程的粒子模拟[10]，玻尔兹曼(Boltzmann)方程模拟解

法[11]等。

Metropolis算法[12]是蒙特卡罗方法的起源，在物理模拟中用得非常广泛。我们在此介绍此算法。

在液体模拟中，假设系统当前是m状态，系统总能量记为$\mathcal{V}_m$。系统试着转到状态n。先计算状态n的总能量$\mathcal{V}_n$，再比较两个状态的能量。如果$\delta\mathcal{V}_{nm} = \mathcal{V}_n - \mathcal{V}_m \leq 0$，即状态$n$是比状态$m$能量低的状态。因为系统总是趋向于能量最低的状态，所以系统处于状态n的概率比系统处于状态m的概率大，此时系统无条件转到状态n。如果$\delta\mathcal{V}_{nm} = \mathcal{V}_n - \mathcal{V}_m > 0$，即状态$n$是比状态$m$能量高的状态，此时系统以概率$\rho_n/\rho_m$转到状态$n$。对于正则系统(canonical ensemble)，即NVT恒定的系统

$$\frac{\rho_n}{\rho_m} = \exp(-\beta\delta\mathcal{V}_{nm})$$

其中β是玻尔兹曼常数。所以产生一个随机数ξ，如果$\xi < \exp(-\beta\delta\mathcal{V}_{nm})$，则系统转到状态$n$，否则系统维持在原来的状态。综上所述，系统从状态$m$转到$n$的概率是：$\min(1, \exp(-\beta\delta\mathcal{V}_{nm}))$。系统从初始状态一直重复下面两步：

- 步骤1：试走步，即计算下一步的能量；
- 步骤2：计算走到下一步的概率$\min(1, \exp(-\beta\delta\mathcal{V}_{nm}))$并决定系统下一步的真正状态。

这是一个马尔科夫(Markov Chain)过程，最终系统会达到稳定状态。

其实Metropolis蒙特卡罗模拟的关键是计算系统转移到下一状态的概率。我们将在第7章介绍光的传播的蒙特卡罗模拟方法，同样强调了如何计算各事件的概率。

1.5 阅读资料

有很多书本介绍蒙特卡罗和拟蒙特卡罗方法。其中[13][14][15] 覆盖积分、优化、模拟三个方面主题，另外[16]注重于随机数发生器，还有[17]则讨论蒙特卡罗方法在生物领域的应用。

第二章 随机数发生器

随机采样是蒙特卡罗方法的核心。蒙特卡罗方法的成功当然取决于随机模型的构造，但很大程度上也取决于模型计算中随机数的性质。这一章着重于随机数发生器。

一般来说随机数分成均匀随机数(uniform random numbers)和非均匀随机数(nonuniform random numbers)。均匀随机数的目标分布(target distribution)是I上的均匀分布(uniform distribution)U。产生非均匀随机数通常是先生成均匀随机数，然后再转换成服从目标分布$F \neq U$的随机数。我们先介绍伪随机数(pseudorandom numbers)和拟随机数(quasirandom numbers)发生器，然后介绍转换方法。

2.1 伪随机数

在使用蒙特卡罗方法的早期，人们就已经认识到在现实生活中不可能有真正的“随机”数。于是人们求助于能通过一定算法及几个参数用计算机产生的伪随机数(PRN) 。

一个经典的且现在仍然广泛使用的产生均匀伪随机数的方法是线性同余方法(linear congruential method)，此方法最初由Lehmer[18]提出。线性同余方法有三个参数，其中M是值很大的正整数，a是满足$1 \leq a < M$和$\gcd(a, M) = 1$的整数，c是集合$Z_M = (0, 1, \cdots, M-1)$的任意元素。一当我们选择、确定了初始值$y_0 \in Z_M$，就可以用下列的递归公式产生一系列的数$y_0, y_1, \cdots \in Z_M$。

$$y_{n+1} \equiv a y_n + c \bmod M \qquad n = 0, 1, \cdots \tag{2.1}$$

于是线性同余伪随机数就可以用这些数除以M得到。

$$x_n = \frac{y_n}{M} \in I = [0,1) \qquad n = 0,1,\cdots \tag{2.2}$$

这里，M被称作模(modulus)，a被称作乘数(multiplier)。模的选择通常是根据计算机的字长，典型的值有$M = 2^{32}$或者Mersenne素数$M = 2^{32} - 1$。为满足更高精度的计算需求，M的值也会取2^{48}。

2.2 拟随机数

无论伪随机数用什么方法产生，它的局限性在于这些随机数总是一个有限长的循环集合，而且序列偏差的上确界达到最大值。所以若能产生低偏差的确定性序列是很有用的，产生的序列应该具有这样的性质，即任意长的子序列都能均匀地填充函数空间。

人们已经产生了若干种满足这个要求的序列，如Van der Corput序列，Halton 序列[19]，Faure 序列[20]，Sobol' 序列[21]和Niederreiter的(t,s)序列[22]。我们称这些序列为拟随机序列(quasirandom sequences)。伪随机序列是为了模拟随机性，而拟随机序列更致力于均匀性。但对(有限长)的拟随机序列只能尽量均匀填充单位超立方体。文[4]和[23]给出了关于拟随机序列的综述。

2.2.1 Halton序列

Halton序列是Van der Corput序列的推广。Halton序列是通过将一系列整数表示成某个基(base)的数位(digit)的形式，然后将这些数位按反序排列再在前面加小数点而得到的值。我们将s维Halton序列表示成$\mathbf{x}_1, \mathbf{x}_2, \cdots$，其中每一个随机数是一个$s$维向量，即$\mathbf{x}_i = (x_{i1}, x_{i2}, \cdots, x_{is})$。生成Halton序列的简单易行的步骤如下：首先选择s个基$b_1, b_2, \cdots, b_s$，比如选择前s个素数。然后对某个整数m，将m表示成以b_j为基的数位，再将这些数位按反序排列再在前面加小数点得到新的值，便是序列中某个随机数向量的第j个元素。即对某个整数m有以下步骤：

1. 选择适当大的t_{mj}，对每个基将m表示成:

$$m = \sum_{k=0}^{t_{mj}} a_{mk} b_j^k, \qquad j = 1, \cdots, s$$

2. 将数位按反序排列再在前面加小数点得到新的值

$$x_{ij} = \sum_{k=0}^{t_{mj}} a_{mk} b_j^{k-t_{mj}-1}, \qquad j = 1, \cdots, s$$

3. 置$m = m + 1$，然后重复1，2两步

举个例子，比如$s = 3$，即维数是3，设$m = 15$，选择2，3和5作为基。我们有$15 = 1111_2$，$15 = 120_3$和$15 = 30_5$，于是便有第一个随机数向量$x = (0.1111_2, 0.021_3, 0.03_5)$，即$(0.937500, 0.259259, 0.120000)$。将$m$加1，如此循环，便得到整个序列。

2.2.2 Sobol'序列

Sobol'序列是基于一组叫做“直接数(direction numbers)”的数v_i而构造的。设m_i是小于2^i的正奇数，则

$$v_i = \frac{m_i}{2^i}$$

数v_i(同时m_i)的生成借助于系数只为0或1的简单多项式(primitive polynomial)。多项式表示成

$$f(z) = z^p + c_1 z^{p-1} + \cdots + c_{p-1} z + c_p \tag{2.3}$$

对$i > p$，我们有递归公式

$$v_i = c_1 v_{i-1} \oplus c_2 v_{i-2} \oplus \cdots \oplus c_p v_{i-p} \oplus \lfloor v_{i-p}/2^p \rfloor \tag{2.4}$$

这里$\oplus$表示二进制按位异或(exclusive-or)。对于m_i，对等的递归公式是

$$m_i = 2c_1 m_{i-1} \oplus 2^2 c_2 m_{i-2} \oplus \cdots \oplus 2^p c_p m_{i-p} \oplus m_{i-p} \tag{2.5}$$

举个例子，取简单多项式

$$x^4 + x + 1$$

对应的递归公式是

$$m_i = 8m_{i-3} \oplus 16m_{i-4} \oplus m_{i-4}$$

如果我们取m的初值为$m_1 = 1$，$m_2 = 1$，$m_3 = 3$和$m_4 = 13$，则有

$$\begin{aligned} m_5 &= 8 \oplus 16 \oplus 1 \\ &= 01000_2 \oplus 10000_2 \oplus 00001_2 \\ &= 11001_2 \\ &= 25 \end{aligned}$$

则可以得到Sobol'序列的第i个数

$$x_i = b_1 v_1 \oplus b_2 v_2 \oplus b_3 v_3 \oplus \cdots,$$

这里$\cdots b_3 b_2 b_1$是i的二进制表示形式。

Antonov和Saleev[24]提出了另一种产生Sobol'序列的简单方法。此方法用下面的递归公式

$$x_i = g_1 v_1 \oplus g_2 v_2 \oplus g_3 v_3 \oplus \cdots, \tag{2.6}$$

其中$\cdots g_3 g_2 g_1$是以i为变量的格雷码(Gray code)的二进制表示。格雷码$G(i)$是以非负整数i为变量的函数，使得$G(i)$ 和$G(i+1)$的二进制表示只有一个数位不同，即在$G(i)$的最右一个零位上不同。Antonbov和Saleev使用的二进制表示的格雷码是利用下式产生的。

$$\cdots g_3 g_2 g_1 = \cdots b_3 b_2 b_1 \oplus b_4 b_3 b_2.$$

(这是最常用的格雷码，能产生一系列函数值$0, 1, 3, 2, 6, 7, 5, 4, \cdots$)于是产生Sobol'序列的(2.6)式可以用下列式子代替。

$$x_i = x_{i-1} \oplus v_r$$

这里r的选择是使b_r是$i-1$的二进制表示的最右零位。

Bratley和Fox在文[25]中讨论了起始$m_1, m_2, \cdots$点的取值(此章例子中所取的简单多项式对应的m的起始值满足那些标准)。

图2.1是一个两维的Sobol'序列，图2.2是Sobol'序列与伪随机数序列的比较。函数$ran2$是书[26]中一个伪随机数发生器。书[26]中的Sobol'序列的C语言代码经过修改，放在附录中。

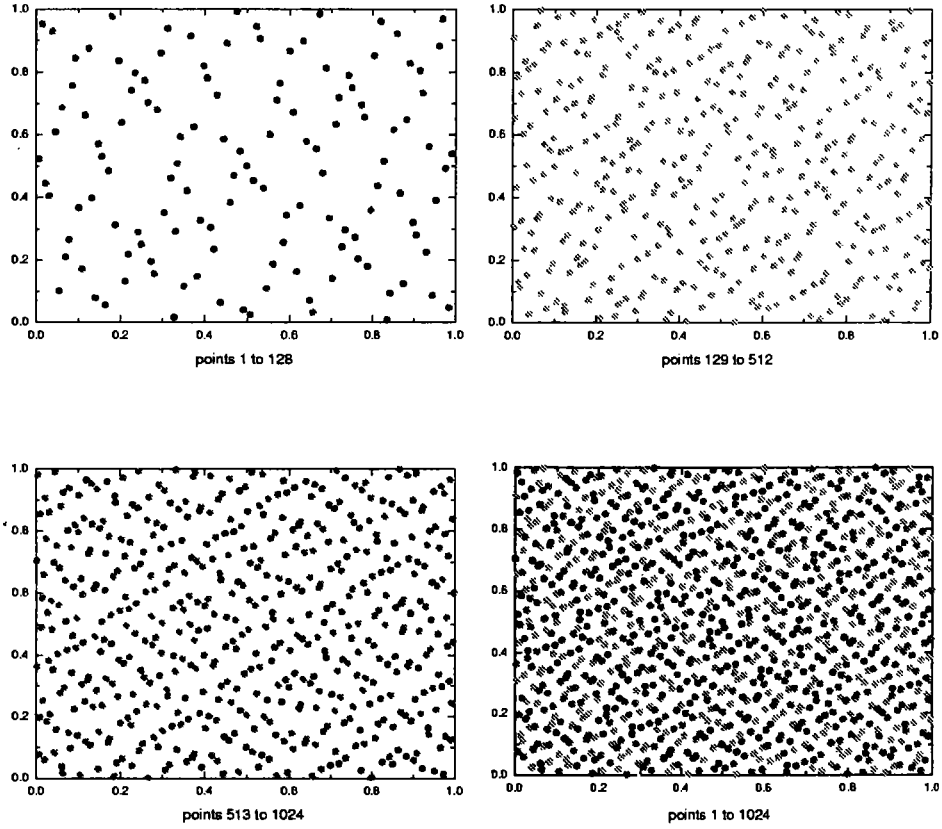

图 2.1: 两维Sobol'随机数，新产生的相继的点分布在先前产生的点的空隙中。

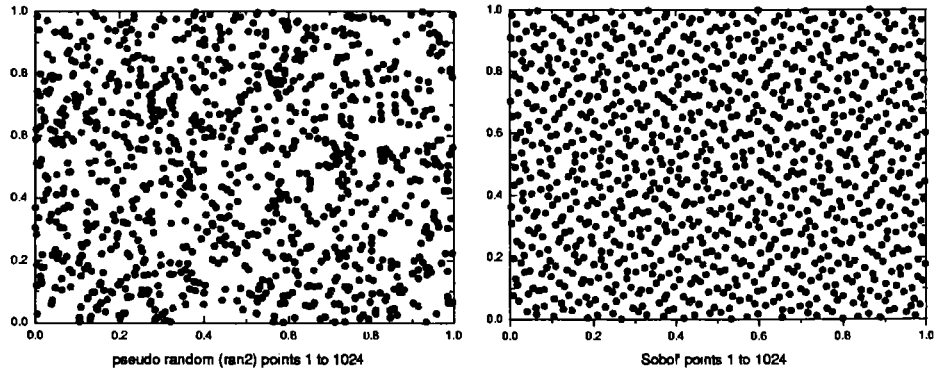

图 2.2: 拟随机数序列比伪随机数序列有更好的均匀性。

2.2.3 Niderreiter的(t,m,s)网格和(t,s)序列

这一节我们介绍(t,m,s)网格和(t,s)序列。这种序列是非常规则分布的序列。

为了定义此序列，我们记维数$s \geq 1$，整数$b \geq 2$，并称$\bar{I}^s$的子区间E

$$E = \prod_{i=1}^{s} [a_i b^{-d_i}, (a_i + 1)b^{-d_i})$$

为以b为基的基本区间(elementary interval)，其中对任意$1 \leq i \leq s$有$a_i, d_i \in \mathbb{Z}$且满足$d_i \geq 0, 0 \leq a_i < b^{d_i}$。

定义 2.2.1 **((t,m,s)网格)** 设t是满足$0 \leq t \leq m$的整数。对任意以b为基的勒贝格测度为$\lambda_s(E) = b^{t-m}$的基本区间E，如果$\bar{I}^s$中的包含b^m个点的点集P满足$A(E,P) = b^t$，则称P为(t,m,s)网格。

定义 2.2.2 **((t,s)序列)** 设t是满足$t \geq 0$的整数。$\bar{I}^s$上的$\mathbf{x}_0, \mathbf{x}_1, \cdots$是一个无穷序列。如果对所有的$k \geq 0$和$m > t$，满足$kb^m \leq n < (k+1)b^m$的点$\mathbf{x}_n$组成的点集都是以$b$为基的$(t,m,s)$网格，则称序列$\mathbf{x}_0, \mathbf{x}_1, \cdots$是$(t,s)$序列。

2.3 从均匀随机数到非均匀随机数的转换方法

从均匀随机数通过转换能产生服从一定概率分布函数的非均匀分布随机数。此节介绍的方法是普适的，因为可以用来产生绝大多数分布的随机数。

2.3.1 累积分布函数(CDF)求逆方法

如果x是有连续的概率累积分布函数(CDF)P_x的随机标量，则由下式产生的随机变量有$U(0,1)$分布。

$$\xi = P_x(x) = \int_a^x p(t)\mathrm{d}t \tag{2.7}$$

这一常识可以让我们在均匀分布随机变量U和随机变量x之间建立一个简单关系，那就是：

$$x = P_x^{-1}(\xi).$$

我们称这一直接了当的转换方式为CDF求逆技术。因为分布函数求逆容易计算，所以此方法是很好的产生随机变量的方法。然而，因为计算某些分布函数的逆函数并不容易，CDF求逆方法不象预想得那样使用广泛。

2.3.2 拒绝方法(rejection method)

令$p(\mathbf{x})$是定义在I^s上的概率密度函数。标准的拒绝采样算法如下描述：

1. 选择γ使得$\gamma \geq \sup_{\mathbf{x}\in I^s} p(\mathbf{x})$.
2. 重复下列步骤直至N个点被接受：
   (a) 从$U([0,1]^{s+1})$采样随机向量$(\mathbf{x}_t, y_t)$，
   (b) 如果有$y_t < \gamma^{-1}p(\mathbf{x}_t)$则接受点$\mathbf{x}_t$；
   否则，拒绝这些点。

跟其它产生非均匀随机数方法类似，拒绝采样方法也依赖于所使用的均匀随机数的均匀性。

第三章 B样条光滑拒绝采样方法及其在重要抽样中的应用

拒绝采样方法是蒙特卡罗方法中最流行的采样算法之一。拒绝采样方法其实跟特征函数(characteristic functions)的蒙特卡罗积分有密切的关系。而由于特征函数的不连续性，蒙特卡罗积分应有的误差精度会达不到，拒绝采样的效果也就受到影响。我们在文[27]中提出了B样条光滑拒绝采样方法，通过用B样条磨光技术在不改变积分值的前提下磨光特征函数。我们将B样条光滑拒绝采样方法用于重要性抽样方法中，数值例子显示拟蒙特卡罗积分的精度重新达到了$O(N^{-1})$的阶，而对于蒙特卡罗积分，采用B样条光滑重要抽样，其精度也比标准积分的精度$O(N^{-\frac{1}{2}})$好。由此证明，B样条光滑拒绝抽样比标准拒绝抽样有效得多。

3.1 引言

标准拒绝采样算法在诸如决策过程，服从密度分布函数的采样等拟蒙特卡罗实际应用中有非常重要的地位。但是它并不如理论结果所期望的那么有效。

拒绝采样算法可以解释成对特征函数的积分。回想第1章所介绍的拟蒙特卡罗积分，其积分误差可用Koksma-Hlawka不等式描述，误差精度的阶一般为$O(N^{-1})$。但是除非函数定义域是边平行于坐标轴的长方形，否则特征函数的变分是无穷的。这样积分误差的上界就不能用Koksma-Hlawka不等式来估计，$O(N^{-1})$的理论误差也就无法达到。

我们用B样条磨光技术磨光特征函数，重新获得了$O(N^{-1})$误差阶。在第3.2.3节我们介绍B样条光滑拒绝抽样方法，在第3.3.2节中将B样条光滑拒绝抽样方法用在重要性抽样中。在第3.4节中给出了数值实验。

3.2 拒绝方法

3.2.1 标准拒绝方法可以解释成特征函数的积分

我们讨论在第2.3.2节中介绍的标准拒绝方法。利用贝叶斯公式，接受点的密度函数$p_{accept}(\mathbf{x})$可以解释成如下的蒙特卡罗积分：

$$p_{\text{accept}}(\mathbf{x}) = \frac{\int_0^1 \chi(y < \gamma^{-1}p(\mathbf{x}))\mathrm{d}y}{\int_{I^s}[\int_0^1 \chi(y < \gamma^{-1}p(\mathbf{x}))\mathrm{d}y]\mathrm{d}\mathbf{x}} = \frac{p(\mathbf{x})/\gamma}{1/\gamma} = p(\mathbf{x}).$$

其中$\chi(y < \gamma^{-1}p(\mathbf{x}))$是如下定义的特征函数：

$$\chi(y < \gamma^{-1}p(\mathbf{x})) = \begin{cases} 1, & if\ y < \gamma^{-1}p(\mathbf{x}), \\ 0, & otherwise. \end{cases} \tag{3.1}$$

由此可见标准拒绝方法接受的点所组成s维点集P是服从密度函数$p(\mathbf{x})$的无穷序列。

但是序列P的前N个元素的性质如何？为了说明这一点，我们引进文[28]中介绍的更具一般性的偏差概念。

定义 3.2.1 (F偏差) 假设$P_N = \{\mathbf{x}_i, , i = 1, \ldots, N\}$是$I^s$上的点集，$F_N(\mathbf{x})$是此点集的经验分布*(empirical distribution)*，即

$$F_N(\mathbf{x}) = (1/N)\sum_{i=1}^{N} \chi\{\mathbf{x}_i \le \mathbf{x}\}$$

则对应累积分布函数*(cumulative distribution function)*$F(\mathbf{x})$的F偏差定义为：

$$D_F(P_N) = \sup_{\mathbf{x}\in I^s} |F_N(\mathbf{x}) - F(\mathbf{x})|. \tag{3.2}$$

F偏差是累积分布函数$F(\mathbf{x})$对点集P_N的刻划好坏的衡量。如文献[29]所述，F偏差实际上是特征函数的拟蒙特卡罗积分的误差，由于特征函数的不连续性，理论误差阶一般只有$O(N^{-1/(s+1)})$。文[29] [30]给出了用连续但不可微函数代替不连续特征函数的光滑拒绝方法，我们在第3.2.3节提出了用可微函数代替特征函数的B样条光滑拒绝抽样方法。

3.2.2 特征函数的磨光

文献[31]介绍了用于磨光函数的B样条磨光技术。众所周知，对任何被积函数$f(x), x \in R$，我们称

$$f_h(x) = \frac{1}{h}\int_{x-\frac{h}{2}}^{x+\frac{h}{2}} f(t)\mathrm{d}t \tag{3.3}$$

为平均函数(average function)。记$D^{-1}f(x) = \int_a^x f(t)\mathrm{d}t$，我们有

$$f_h(x) = h^{-1}\delta_h D^{-1} f(x)$$

其中$\delta_h F(x) = F(x+\frac{h}{2}) - F(x-\frac{h}{2})$。将磨光算子$h^{-1}\delta_h D^{-1}$ (h是磨光宽度)应用于一些简单的基本不连续函数，比如$f(x) = x_+$，则有

$$\begin{aligned} f_h(x) &= \tfrac{1}{2h}[(x+\tfrac{h}{2})_+^2 - (x-\tfrac{h}{2})_+^2] \\ &= \begin{cases} 0, & if\ x \le -\frac{h}{2}, \\ (x+\frac{h}{2})^2/2h, & if\ -\frac{h}{2} < x \le \frac{h}{2}, \\ x, & if\ x > \frac{h}{2}. \end{cases} \end{aligned}$$

显然平均函数$f_h(x)$是连续函数。当h足够小时，函数$f_h(x)$是函数$f(x)$的近似，它们的差别在于区间$[x-\frac{h}{2}, x+\frac{h}{2}]$上的函数值不同。

定理 3.2.1 *如果函数$f_h(x)$是如式(3.3)所定义的函数$f(x)$的平均函数，则有*

$$f_h(x) \approx f(x).$$

及$\lim_{h\to 0} f_h(x) = f(x)$。

我们称上述的光滑技术为B样条光滑技术。

3.2.3 B样条光滑拒绝抽样方法

我们同样可以将B样条磨光技术用于式(3.1)所定义的特征函数$\chi(y < \gamma^{-1}p(\mathbf{x}))$。先将特征函数改写成：

$$W_0(\mathbf{x}, y) = (\gamma^{-1}p(\mathbf{x}) - y)_+^0, \qquad 0 \le y \le 1.$$

然后将磨光算子$(2h)^{-1}\delta_{2h}D^{-1}$应用于$W_0(\mathbf{x}, y)$，这里磨光宽度是$2h$，得

到，

$$\begin{aligned} W_\delta(\mathbf{x},y) &= (2h)^{-1}\delta_{2h}D^{-1}W_0(\mathbf{x},y) \\ &= (2h)^{-1}[(\gamma^{-1}p(\mathbf{x})-y+h)_+ - ((\gamma^{-1}p(\mathbf{x})-y-h)_+] \end{aligned}$$

记$f_1(y)=(\gamma^{-1}p(\mathbf{x})-y+h)_+$及$f_2(y)=(\gamma^{-1}p(\mathbf{x})-y-h)_+$，再将磨光算子$h^{-1}\delta_h D^{-1}$分别应用于$f_1(y)$和$f_2(y)$，我们得到可微的权重函数。

$$W_{\delta\delta}(\mathbf{x},y)=(2h)^{-1}[h^{-1}\delta_h D^{-1}f_1(y)-h^{-1}\delta_h D^{-1}f_2(y)]$$

$$= \begin{cases} 1 & , 0\le y<\gamma^{-1}p(\mathbf{x})-\frac{3h}{2} \\ \frac{[(\gamma^{-1}p(\mathbf{x})-y+h)-\frac{(\gamma^{-1}p(\mathbf{x})-y-\frac{h}{2})^2}{2h}]}{2h} & , \gamma^{-1}p(\mathbf{x})-\frac{3h}{2}\le y<\gamma^{-1}p(\mathbf{x})-\frac{h}{2}, \\ \frac{(\gamma^{-1}p(\mathbf{x})-y+h)}{2h} & , \gamma^{-1}p(\mathbf{x})-\frac{h}{2}\le y<\gamma^{-1}p(\mathbf{x})+\frac{h}{2}, \\ \frac{(\gamma^{-1}p(\mathbf{x})-y+\frac{3h}{2})^2}{4h^2} & , \gamma^{-1}p(\mathbf{x})+\frac{h}{2}\le y<\gamma^{-1}p(\mathbf{x})+\frac{3h}{2}, \\ 0 & , \gamma^{-1}p(\mathbf{x})+\frac{3h}{2}\le y\le 1. \end{cases} \tag{3.4}$$

函数$W_{\delta\delta}(\mathbf{x},y)$就是我们用来代替特征函数$\chi(y<\gamma^{-1}p(\mathbf{x}))$的，称其为权重函数。于是改进的B样条光滑拒绝抽样算法可以描述如下：

1. 选择满足$\gamma\ge\sup_{\mathbf{x}\in I^s}p(\mathbf{x})$的$\gamma$。
2. 重复下述步骤直到权重w_t的和达到值N:
   (a) 从$U([0,1]^{s+1})$采样得到$(\mathbf{x}_t,y_t)$;
   (b) 将点$\mathbf{x}_t$的权重置为$w_t=W_{\delta\delta}(\mathbf{x}_t,y_t)$，即点$\mathbf{x}_t$被接受的概率是$w_t$。

定理 3.2.2 *$W_{\delta\delta}(\mathbf{x},y)$是由式(3.4)定义的权重函数，$\chi(y<\gamma^{-1}p(\mathbf{x}))$是由式(3.1)定义的特征函数。我们有*

$$\int_0^1 W_{\delta\delta}(\mathbf{x},y)\mathrm{d}y=\int_0^1\chi(y<\gamma^{-1}p(\mathbf{x}))\mathrm{d}y$$

即B样条光滑拒绝抽样方法所生成的序列服从概率分布$p(\mathbf{x})$。

证明: 令$t=\gamma^{-1}p(\mathbf{x})-\frac{3h}{2}$，则

$$\begin{aligned}\int_0^1 W_{\delta\delta}(\mathbf{x},y)\mathrm{d}y &= \int_0^t 1\mathrm{d}y+\int_t^{t+h}\frac{[(t-y+\frac{5h}{2})-\frac{(t-y+h)^2}{2h}]}{2h}\mathrm{d}y+\int_{t+h}^{t+2h}\frac{(t-y+\frac{5h}{2})}{2h}\mathrm{d}y\\ &\quad+\int_{t+2h}^{t+3h}\frac{(t-y+3h)^2}{4h^2}\mathrm{d}y\\ &= t+\frac{11h}{12}+\frac{h}{2}+\frac{h}{12}=t+\frac{3h}{2}=\gamma^{-1}p(\mathbf{x})\end{aligned}$$

所以有$\int_0^1 W_{\delta\delta}(\mathbf{x},y)\mathrm{d}y=\int_0^1\chi(y<\gamma^{-1}p(\mathbf{x}))\mathrm{d}y$。

也就是说用B样条光滑拒绝抽样方法所生成的序列服从概率分布$p(\mathbf{x})$。

我们将在第3.3.2节中将B样条光滑拒绝抽样方法用于拟蒙特卡罗积分的重要性抽样。

3.3 重要性抽样

3.3.1 标准重要性抽样

重要性抽样(importance sampling)也许是蒙特卡罗中最重要的方差缩减(variance reduction)技术。将积分$I(f)$改写成:

$$I(f)=\int_{I^s}f(\mathbf{x})\mathrm{d}\mathbf{x}=\int_{I^s}\frac{f(\mathbf{x})}{p(\mathbf{x})}p(\mathbf{x})\mathrm{d}\mathbf{x},$$

这里函数$p(\mathbf{x})$称重要性函数(importance function)，重要性函数的选择要尽量符合函数$f(\mathbf{x})$在空间I^s上的性质。标准的重要性抽样估计可写成:

$$I_N^{(IS)}=\frac{1}{N}\sum_{i=1}^{N}\frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)},\tag{3.5}$$

这里$\mathbf{x}_1,\ldots,\mathbf{x}_N$是服从密度函数$p(\mathbf{x})$的采样点。拒绝采样就是用于产生服从分布$p(\mathbf{x})$的采样点的实用方法。然而由于特征函数的不连续性，从而使用拟蒙特卡罗方法能得到的改善无法达到。我们将B样条光滑拒绝抽样方法应用于重要性抽样，重新得到了$O(N^{-1})$的误差阶。

3.3.2 用B样条光滑拒绝抽样方法改进重要性抽样

我们将积分$I(f)$改写成:

$$\begin{aligned} I(f) = \int_{I^s} f(\mathbf{x})\mathrm{d}\mathbf{x} &= \gamma \int_{I^s} \frac{f(\mathbf{x})}{p(\mathbf{x})}\gamma^{-1}p(\mathbf{x})\mathrm{d}\mathbf{x} \\ &= \gamma \int_{I^s} \frac{f(\mathbf{x})}{p(\mathbf{x})}[\int_0^1 \chi(y < \gamma^{-1}p(\mathbf{x})\mathrm{d}y]\mathrm{d}\mathbf{x} \\ &= \gamma \int_{I^s} \frac{f(\mathbf{x})}{p(\mathbf{x})}[\int_0^1 W_{\delta\delta}(\mathbf{x},y)\mathrm{d}y]\mathrm{d}\mathbf{x} \\ &\approx \frac{\gamma}{N^*}\sum_{i=1}^{N^*} W_{\delta\delta}(\mathbf{x}_i,y_i)\frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)} \end{aligned}$$

于是改进的拟蒙特卡罗重要性抽样估计可以定义成:

$$I_N^{(BIS)} = \frac{1}{N}\sum_{i=1}^{N^*} W_{\delta\delta}(\mathbf{x}_i,y_i)\frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)}, \tag{3.6}$$

这里$W_{\delta\delta}(\mathbf{x},y)$由式(3.4)定义，$N^*$的选择使得接受权重$w_i$的和不超过$N$。显然

$$N \approx N^*/\gamma$$

改进的蒙特卡罗和拟蒙特卡罗重要性抽样的数值例子在下一节中给出。

3.4 数值实验

在这一节中，我们比较蒙特卡罗和拟蒙特卡罗的标准估计，标准重要性估计和B样条光滑拒绝抽样估计。我们选择了几个经典函数试验这些估计以证明B样条光滑拒绝抽样方法的有效性，在此仅举一例。下面的计算公式分别来自于式(1.2)，式(3.5) 和式(3.6):

原始蒙特卡罗(Crude Monte Carlo):
$Y_N^{(1)} = (1/N)\sum_{i=1}^N f(\mathbf{x}_i)$，$\mathbf{x}_i \sim U([0,1]^s)$;

标准拒绝方法(Standard rejection method):
$Y_N^{(2)} = (1/N)\sum_{i=1}^N f(\mathbf{x}_i)/p(\mathbf{x}_i)$，$\mathbf{x}_i \sim p(\mathbf{x}_i)$，对接受的点

B样条光滑拒绝抽样(B-spline smooth rej.):
$Y_N^{(3)} = (1/N)\sum_{i=1}^{N^*} W_{\delta\delta}(\mathbf{x}_i,y_i)f(\mathbf{x}_i)/p(\mathbf{x}_i)$。

对于给定的N，计算这些估计的m个样本，记作$Y_N^{(j)}(k)$对所有满足$1 \leq k \leq m$的k(每个样本用的点是单个序列中相继的一段随机数)。于是积分值$I(f)$的最终估计可定义成$\hat{I}^{(j)} = (1/m)\sum_{k=1}^{m} Y_N^{(j)}(k)$。我们总能用下列两种形式来描述数值误差，这就是经验标准离差(standard deviation(sd))和经验均方根误差(root mean square error($rmse$))，分别定义如下：

$$sd(\hat{\sigma}^{(j)}) = \sqrt{\frac{1}{(m-1)}\sum_{k=1}^{m}[Y_N^{(j)}(k) - \hat{I}^{(j)}]^2}, j = 1,2,3. \tag{3.7}$$

$$rmse(\hat{\sigma}^{(j)}) = \sqrt{\frac{1}{m}\sum_{k=1}^{m}[Y_N^{(j)}(k) - I]^2}, j = 1,2,3. \tag{3.8}$$

我们用文[32]中的Halton序列来产生拟随机数点，文[26]中的*ran2*函数来产生伪随机数点。令$m = 75$，也就是每个估计运行75次，用75个不同的随机数子序列。我们画图时对变量取了对数，这样斜率(值已在图中标出)就对应于误差阶。

例1. 此数值例子是在区间$I^7 = [0,1]^7$上的函数

$$f_1(\mathbf{x}) = e^{1-(\sin^2(\frac{\pi}{2}x_1)+\sin^2(\frac{\pi}{2}x_2)+\sin^2(\frac{\pi}{2}x_3))}\arcsin(\sin(1) + \frac{x_1 + \cdots + x_7}{200})$$

的蒙特卡罗和拟蒙特卡罗积分。取的重要性函数是：

$$p_1(\mathbf{x}) = \frac{1}{\eta}e^{1-(\sin^2(\frac{\pi}{2}x_1)+\sin^2(\frac{\pi}{2}x_2)+\sin^2(\frac{\pi}{2}x_3))},$$

这里$\eta = \int_{I^7} e^{1-(\sin^2(\frac{\pi}{2}x_1)+\sin^2(\frac{\pi}{2}x_2)+\sin^2(\frac{\pi}{2}x_3))}\mathrm{d}\mathbf{x} = e \cdot (\int_0^1 e^{-(\sin^2(\frac{\pi}{2}x)}\mathrm{d}x)^3$，这是一维积分，其值可以高精度地算得。

数值例子分别在拒绝抽样中用拟随机数和伪随机数计算所得的重要抽样估计的sd误差在图3.1和图3.2中。

计算结果显示：

- 用同样多的随机数点，拟蒙特卡罗方法比蒙特卡罗方法的误差小(无论用不用重要性抽样)。
- 无论是对拟蒙特卡罗方法还是蒙特卡罗方法，B样条光滑拒绝抽样方法都比标准拒绝抽样方法好

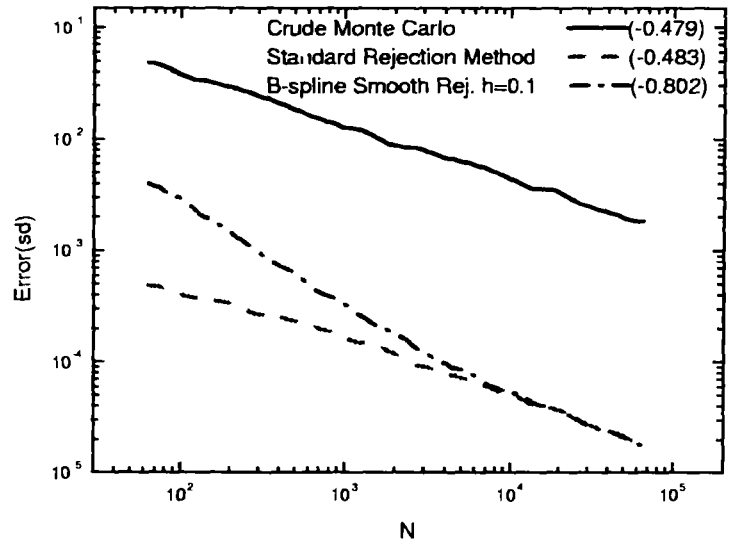

图 3.1: 数值例子用伪随机数计算所得的重要抽样估计的sd误差。

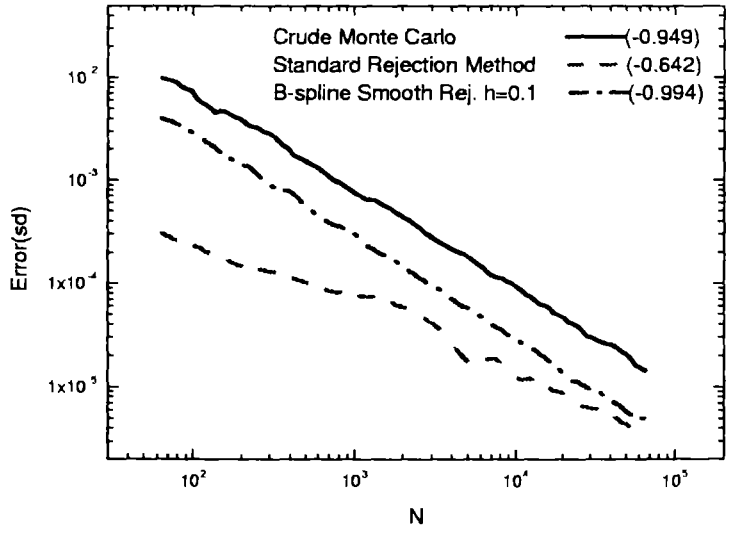

图 3.2: 数值例子用拟随机数计算所得的重要抽样估计的sd误差。

- 对蒙特卡罗方法，用B样条光滑拒绝抽样的重要抽样估计也提高了精度，比原始蒙特卡罗的精度$O(N^{-\frac{1}{2}})$好。

3.5 结语

综前所述，B样条光滑拒绝抽样方法改进了标准拒绝方法，从拟蒙特卡罗重要抽样估计重新达到精度$O(N^{-1})$可见一斑。同时我们也发现将B样条光滑拒绝抽样用在蒙特卡罗重要抽样估计中，也能提高积分精度，如文中的数值例子，误差精度达到$O(N^{-0.8})$，比原始蒙特卡罗积分的精度$O(N^{-0.5})$好得多。虽然，对于文中的数值例子，我们大可以直接用原始拟蒙特卡罗积分方法，不必费尽周折地因为要用重要抽样而用拒绝抽样。但对于如决策过程等，必须用到拒绝抽样方法，此时用B样条光滑拒绝抽样就显得很重要了，效果比标准拒绝抽样好得多。

第四章 精细对偶变数蒙特卡罗积分方法及其并行程序实现

在这一章我们得出了用于多重积分的精细对偶变数蒙特卡罗(fine antithetic variables Monte Carlo，简称FAMC)方法的误差估计式。对维数是s的二阶导数连续的函数来说，FAMC方法理论误差的阶是$O(N^{-(\frac{1}{2}+\frac{2}{s})})$。我们同时也讨论了对偶变数蒙特卡罗积分(antithetic variable Monte Carlo，简称AMC)方法。对次数不高于2的多变量函数，AMC方法其误差阶是$O(N^{-\frac{1}{2}})$，但其系数比原始蒙特卡罗积分(MC)方法的误差阶的系数小。我们用C语言实现了并行计算程序，数值实验结果与理论结果吻合得很好。

4.1 引言

考虑绝对收敛的高维积分

$$I=\int_{I^s} f(\mathbf{x})\mathrm{dx} \tag{4.1}$$

的数值估计，这里$I^s=[0,1]^s$是s维超立方体。则原始蒙特卡罗求积公式(crude Monte Carlo(MC) estimator)

$$M=\frac{1}{N}\sum_{k=1}^{N} f(\xi_k) \tag{4.2}$$

可以用来估计I的值。这里点$\xi_1,\xi_2,\cdots,\xi_N$是I^s上独立均匀分布随机数。在文[4][6]中已经证明原始蒙特卡罗方法的误差阶是$O(N^{-\frac{1}{2}})$。文[33]给出了另一个积分估计式，称为对偶变数蒙特卡罗(antithetic variables Monte Carlo(AMC))。

$$A=\frac{1}{2N}\sum_{k=1}^{N}[f(\xi_k)+f(2\mathbf{c}-\xi_k)] \tag{4.3}$$

这里点$\mathbf{c} = (\frac{1}{2},\cdots,\frac{1}{2})^{\mathrm{T}}$是$I^s$的中心点。对偶变数方法是由Hammersley等人在文[34]中首先提出。我们将空间I^s用均匀网格分成$N = n^s$个子超立方体D_k，记$\mathbf{c}_k$为D_k的中心点。定义点$\mathbf{d}_k$为各子空间中离原点$\mathbf{O} = (0,\cdots,0)^{\mathrm{T}}$最近的点，令$\eta_k = \mathbf{d}_k + \xi_k/N^{1/s}$，则精细对偶变数蒙特卡罗估计(fine antithetic variables Monte Carlo(FAMC) estimator)定义如下：

$$F = \frac{1}{2N}\sum_{k=1}^{N}[f(\eta_k) + f(2\mathbf{c}_k - \eta_k)] \tag{4.4}$$

S.Haber在[35]提出了同样的方法并给出定理证明二阶导数连续的函数的误差阶是$O(N^{-(\frac{1}{2}+\frac{2}{s})})$，我们在第4.2节中给出了定理及我们的证明。推论指出AMC方法的误差阶$O(N^{-\frac{1}{2}})$前的系数比MC方法同样的误差阶前的系数小。因为对偶变数方法要求计算的函数值个数与维数s有关，所以传统的串行程序对较高维数的问题计算非常花时。我们在第4.3节给出了C语言实现的并行程序。第4.4节给出的数值结果很好地吻合了理论结果。

4.2 精细对偶变数蒙特卡罗积分方法的理论结果

对点$\mathbf{x} = (x_1,\cdots,x_s)^{\mathrm{T}} \in R^s$记模$\|\mathbf{x}\| = (\sum_{i=1}^{s}(x_i)^2)^{\frac{1}{2}}$，假设$D$是$R^s$中的凸域，$\delta > 0$，则函数$f : D \to R$的连续模(modulus of continuity)$\omega(f,\delta)$和二阶连续模(second order modulus of continuity)$\Omega(f,\delta)$分别定义如下：

$$\omega(f,\delta) = \sup_{\substack{\|\mathbf{t}\|<\delta \\ \mathbf{x},\mathbf{x}+\mathbf{t}\in D}} |f(\mathbf{x}+\mathbf{t}) - f(\mathbf{x})| \tag{4.5}$$

$$\Omega(f,\delta) = \sup_{\substack{\|\mathbf{t}\|<\delta \\ \mathbf{x}\pm\mathbf{t}\in D}} |f(\mathbf{x}+\mathbf{t}) - 2f(\mathbf{x}) + f(\mathbf{x}-\mathbf{t})| \tag{4.6}$$

对函数$f : D \to R^s$，连续模则定义为：

$$\omega(f,\delta) = (\sum_{i=1}^{s}\omega(f_i,\delta)^2)^{\frac{1}{2}} \tag{4.7}$$

我们将FAMC估计简记为F，AMC估计简记为A，I是引言中定义的精确积分，下面给出理论结果。

引理 4.2.1 记$C^1(D)$为凸域D上一阶导数连续的函数类，假设函数$f \in$

$C^1(D)$，则二阶连续模$\Omega(f,\delta)$和连续模$\omega(f,\delta)$ 两者之间有如下不等式。

$$\Omega(f,\delta) \le 2\delta\omega(f',\delta) \tag{4.8}$$

这里$\delta > 0$。

证明. 对所有的$\mathbf{x} \pm \mathbf{t} \in D$，存在$\theta_i \in (0,1), i = 1,2$，使得

$$\begin{aligned} f(\mathbf{x}+\mathbf{t}) - f(\mathbf{x}) &= \sum_{i=1}^{s} \frac{\partial f(\mathbf{x}+\theta_1\mathbf{t})}{\partial x_i} t_i, \\ f(\mathbf{x}-\mathbf{t}) - f(\mathbf{x}) &= -\sum_{i=1}^{s} \frac{\partial f(\mathbf{x}-\theta_2\mathbf{t})}{\partial x_i} t_i. \\ \Rightarrow \quad f(\mathbf{x}+\mathbf{t}) - 2f(\mathbf{x}) + f(\mathbf{x}-\mathbf{t}) &= \sum_{i=1}^{s} \left(\frac{\partial f(\mathbf{x}+\theta_1\mathbf{t})}{\partial x_i} - \frac{\partial f(\mathbf{x}-\theta_2\mathbf{t})}{\partial x_i}\right) t_i. \end{aligned}$$

对所有的$\|\mathbf{t}\| < \delta$，运用柯西不等式，便得

$$\begin{aligned} \Omega(f,\delta)^2 &\le \sum_{i=1}^{s} \left(\left| \frac{\partial f(\mathbf{x}+\theta_1 i)}{\partial x_i} - \frac{\partial f(\mathbf{x})}{\partial x_i} \right| + \left| \frac{\partial f(\mathbf{x})}{\partial x_i} - \frac{\partial f(\mathbf{x}-\theta_2\mathbf{t})}{\partial x_i} \right| \right)^2 \sum_{i=1}^{s} (t_i)^2 \\ &\le \sum_{i=1}^{s} [2\omega(\frac{\partial f}{\partial x_i}, \delta)]^2 \cdot \delta^2 \end{aligned}$$

所以$\Omega(f,\delta) \le 2\delta\omega(f',\delta)$成立。

定理 4.2.2 记$C(I^s)$为$I^s = [0,1]^s$上连续函数类，函数$f \in C(I^s)$，则FAMC估计F的方差可以用二阶连续模$\Omega(f,\delta)$来估计

$$\mathrm{E}(F-I)^2 < \frac{1}{4N}\Omega(f, \frac{1}{2N^{1/s}})^2 \tag{4.9}$$

这里N是所使用的随机数点数。

证明. 记

$$\begin{aligned} F_k &= \frac{1}{2N}[f(\eta_k) + f(2\mathbf{c}_k - \eta_k)], \\ I_k &= \int_{D_k} f(\mathbf{x})\mathrm{d}\mathbf{x}. \end{aligned}$$

将$\eta_k = \mathbf{d}_k + \xi_k/N^{1/s}$改写成$\xi_k = N^{1/s}(\eta_k - \mathbf{d}_k)$，代入下式得到

$$\begin{aligned} \mathrm{E}F_k &= \frac{1}{2N}\int_{I^s}|f(\eta_k) + f(2\mathbf{c}_k - \eta_k)|\mathrm{d}\xi_k \\ &= \frac{1}{2}\int_{D_k}[f(\eta_k) + f(2\mathbf{c}_k - \eta_k)]\mathrm{d}\eta_k \\ &= I_k. \end{aligned}$$

将F_k和I_k代入$\mathrm{E}(F_k - I_k)^2$，然后加一项$2f(\mathbf{c}_k)$减一项$2f(\mathbf{c}_k)$，便得到F_k的方差的值。

$$\begin{aligned} \mathrm{E}(F_k - I_k)^2 = & N\int_{D_k}(F_k - I_k)^2\mathrm{d}\eta_k \\ = & N\int_{D_k}\Big\{\Big[\frac{1}{2N}(f(\eta_k) - 2f(\mathbf{c}_k) + f(2\mathbf{c}_k - \eta_k))\Big] \\ & -\Big[\frac{1}{2}\int_{D_k}(f(\mathbf{x}) - 2f(\mathbf{c}_k) + f(2\mathbf{c}_k - \mathbf{x}))\mathrm{d}\mathbf{x}\Big]\Big\}^2\mathrm{d}\eta_k \end{aligned}$$

将平方项展开，并整理整个式子，得

$$\begin{aligned} \mathrm{E}(F_k - I_k)^2 = & \frac{1}{4N}\int_{D_k}[f(\mathbf{x}) - 2f(\mathbf{c}_k) + f(2\mathbf{c}_k - \mathbf{x})]^2\mathrm{d}\mathbf{x} \\ & -\frac{1}{4}\Big\{\int_{D_k}[f(\mathbf{x}) - 2f(\mathbf{c}_k) + f(2\mathbf{c}_k - \mathbf{x})]\mathrm{d}\mathbf{x}\Big\}^2 \\ < & \frac{1}{4N}\int_{D_k}[f(\mathbf{x}) - 2f(\mathbf{c}_k) + f(2\mathbf{c}_k - \mathbf{x})]^2\mathrm{d}\mathbf{x} \end{aligned}$$

因为$\mathbf{c}_k$是D_k的中心点，所以对$\mathbf{x} \in D_k$有$\|\mathbf{x} - \mathbf{c}_k\| < \frac{1}{2N^{1/s}}$。根据$\Omega(f,\delta)$的定义，又有$f(\mathbf{x}) - 2f(\mathbf{c}_k) + f(2\mathbf{c}_k - \mathbf{x}) \le \Omega(f, \frac{1}{2N^{1/s}})$。所以

$$\begin{aligned} \mathrm{E}(F_k - I_k)^2 \le & \frac{1}{4N}\int_{D_k}\Omega(f, \frac{1}{2N^{1/s}})^2\mathrm{d}\mathbf{x} \\ = & \frac{1}{4N^2}\Omega(f, \frac{1}{2N^{1/s}})^2 \end{aligned}$$

因为随机数点是相互独立的，所以当$i \neq k$时$\mathrm{Cov}(F_i - I_i)(F_k - I_k) = 0$。将所有的$\mathrm{E}(F_k - I_k)^2$相加，得到

$$\mathrm{E}(F-I)^2 = \sum_{k=1}^{N} \mathrm{E}(F_k - I_k)^2 < \frac{1}{4N}\Omega(f, \frac{1}{2N^{1/s}})^2$$

定理证明完毕。

将引理4.2.1应用于定理4.2.2，得到

推论 4.2.3 记$C^1(I^s)$为$I^s = [0,1]^s$上一阶导数连续的函数类，如果$f \in C^1(I^s)$，则F的方差可以用连续模$\omega(f,\delta)$来估计：

$$\mathrm{E}(F-I)^2 < \frac{1}{4N^{1+2/s}}\omega(f', \frac{1}{2N^{1/s}})^2 \tag{4.10}$$

这里N是所用的随机数点数。

用类似的推导方法，对函数类$f \in C^2(I^s)$，我们可以估计$\mathrm{E}(F-I)^2$的阶为$O(\frac{1}{N^{1+\frac{4}{s}}})$，下面是精确的结果。

定理 4.2.4 记$C^2(I^s)$为$I^s = [0,1]^s$上二阶导数连续的函数类，假设$f \in C^2(I^s)$，则F的方差有如下精确估计式：

$$\mathrm{E}(F-I)^2 = \frac{1}{288}\int_{I^s}\sum_{i=1}^{s}\sum_{j=1}^{s}(1-\frac{3}{5}\delta_{ij})\Big(\frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j}\Big)^2 \mathrm{d}\mathbf{x}\cdot\frac{1}{N^{1+\frac{4}{s}}} + o(\frac{1}{N^{1+\frac{4}{s}}}), \tag{4.11}$$

其中$\delta_{ij} = \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases}$是Kronecleer符号，$N$是所用的随机数点数。

证明. 从定理4.2.2的证明我们知道

$$\begin{aligned} \mathrm{E}(F_k - I_k)^2 = \; & \frac{1}{4N}\int_{D_k}[f(\mathbf{x}) - 2f(\mathbf{c}_k) + f(2\mathbf{c}_k - \mathbf{x})]^2 \mathrm{d}\mathbf{x} \\ & -\frac{1}{4}\Big\{\int_{D_k}[f(\mathbf{x}) - 2f(\mathbf{c}_k) + f(2\mathbf{c}_k - \mathbf{x})]\mathrm{d}\mathbf{x}\Big\}^2. \end{aligned} \tag{4.12}$$

现在计算

$$\begin{aligned} & f(\mathbf{x})-2f(\mathbf{c}_k)+f(2\mathbf{c}_k-\mathbf{x}) \\ = & (\mathbf{x}-\mathbf{c}_k)^{\mathrm{T}}f''(\mathbf{c}_k)(\mathbf{x}-\mathbf{c}_k)+o(\frac{1}{N^{\frac{2}{s}}}) \\ = & \sum_{i=1}^{s}\sum_{j=1}^{s}\frac{\partial^2 f(\mathbf{c}_k)}{\partial x_i\partial x_j}(x_i-c_{ki})(x_j-c_{kj})+o(\frac{1}{N^{\frac{2}{s}}}) \end{aligned}$$

因为$\mathbf{c}_k$是D_k的中心点，所以$\int_{D_k}(x_i-c_{ki})(x_j-c_{kj})\mathrm{d}\mathbf{x}=0$，于是

$$\begin{aligned} & \int_{D_k}[f(\mathbf{x})-2f(\mathbf{c}_k)+f(2\mathbf{c}_k-\mathbf{x})]\mathrm{d}\mathbf{x} \\ = & \sum_{i=1}^{s}\frac{\partial^2 f(\mathbf{c}_k)}{(\partial x_i)^2}\int_{D_k}(x_i-c_{ki})^2\mathrm{d}x+o(\frac{1}{N^{1+\frac{2}{s}}}) \qquad (4.13) \\ = & \frac{1}{12N^{1+\frac{2}{s}}}\sum_{i=1}^{s}\frac{\partial^2 f(\mathbf{c}_k)}{(\partial x_i)^2}+o(\frac{1}{N^{1+\frac{2}{s}}}) \end{aligned}$$

同时$|f(\mathbf{x})-2f(\mathbf{c}_k)+f(2\mathbf{c}_k-\mathbf{x})|^2$在$D_k$上的积分是

$$\begin{aligned} & \int_{D_k}|f(\mathbf{x})-2f(\mathbf{c}_k)+f(2\mathbf{c}_k-\mathbf{x})|^2\mathrm{d}\mathbf{x} \\ = & \int_{D_k}\sum_{i=1}^{s}\sum_{j=1}^{s}\frac{\partial^2 f(\mathbf{c}_k)}{\partial x_i\partial x_j}(x_i-c_{ki})(x_j-c_{kj})\sum_{m=1}^{s}\sum_{n=1}^{s}\frac{\partial^2 f(\mathbf{c}_k)}{\partial x_m\partial x_n}(x_m-c_{km})(x_n-c_{kn})\mathrm{d}\mathbf{x} \\ & +o(\frac{1}{N^{1+\frac{4}{s}}}) \end{aligned}$$

只有那些满足$\begin{cases} m\neq i \\ j=i \\ n-m \end{cases}$或$\begin{cases} j\neq i \\ m=i \\ n-j \end{cases}$或$\begin{cases} j\neq i \\ n=i \\ m-j \end{cases}$或$n=m=j=i$的项非零。

所以

$$\int_{D_k}[f(\mathbf{x})-2f(\mathbf{c}_k)+f(2\mathbf{c}_k-\mathbf{x})]^2\mathrm{d}\mathbf{x}$$

$$=\sum_{i=1}^{s}\sum_{m\neq i}\frac{\partial^2 f(\mathbf{c}_k)}{(\partial x_i)^2}\frac{\partial^2 f(\mathbf{c}_k)}{(\partial x_m)^2}\int_{D_k}(x_i-c_{ki})^2(x_m-c_{km})^2\mathrm{d}\mathbf{x}$$

$$+2\sum_{i=1}^{s}\sum_{j\neq i}\Big(\frac{\partial^2 f(\mathbf{c}_k)}{\partial x_i\partial x_j}\Big)^2\int_{D_k}(x_i-c_{ki})^2(x_j-c_{kj})^2\mathrm{d}\mathbf{x}$$

$$+\sum_{i=1}^{s}\Big(\frac{\partial^2 f(\mathbf{c}_k)}{(\partial x_i)^2}\Big)^2\int_{D_k}(x_i-c_{ki})^4\mathrm{d}\mathbf{x}+o(\frac{1}{N^{1+\frac{4}{s}}})$$

$$=\frac{1}{144N^{1+\frac{4}{s}}}\sum_{i=1}^{s}\sum_{m\neq i}\frac{\partial^2 f(\mathbf{c}_k)}{(\partial x_i)^2}\frac{\partial^2 f(\mathbf{c}_k)}{(\partial x_m)^2}+\frac{2}{144N^{1+\frac{4}{s}}}\sum_{i=1}^{s}\sum_{j\neq i}\Big(\frac{\partial^2 f(\mathbf{c}_k)}{\partial x_i\partial x_j}\Big)^2$$

$$+\frac{1}{80N^{1+\frac{4}{s}}}\sum_{i=1}^{s}\Big(\frac{\partial^2 f(\mathbf{c}_k)}{(\partial x_i)^2}\Big)^2+o(\frac{1}{N^{1+\frac{4}{s}}})$$

在i的求和公式下加一项减一项$\Big(\frac{\partial^2 f(\mathbf{c}_k)}{(\partial x_i)^2}\Big)^2$，得到

$$\int_{D_k}[f(\mathbf{x})-2f(\mathbf{c}_k)+f(2\mathbf{c}_k-\mathbf{x})]^2\mathrm{d}\mathbf{x}$$

$$=\frac{1}{144N^{1+\frac{4}{s}}}\sum_{i=1}^{s}\sum_{m=1}^{s}\frac{\partial^2 f(\mathbf{c}_k)}{(\partial x_i)^2}\frac{\partial^2 f(\mathbf{c}_k)}{(\partial x_m)^2}+\frac{2}{144N^{1+\frac{4}{s}}}\sum_{i=1}^{s}\sum_{j=1}^{s}\Big(\frac{\partial^2 f(\mathbf{c}_k)}{\partial x_i\partial x_j}\Big)^2$$

$$+(\frac{1}{80}-\frac{3}{144})\frac{1}{N^{1+\frac{4}{s}}}\sum_{i=1}^{s}\Big(\frac{\partial^2 f(\mathbf{c}_k)}{(\partial x_i)^2}\Big)^2+o(\frac{1}{N^{1+\frac{4}{s}}})$$

$$=\frac{1}{144N^{1+\frac{4}{s}}}\Big(\sum_{i=1}^{s}\frac{\partial^2 f(\mathbf{c}_k)}{(\partial x_i)^2}\Big)^2+\frac{1}{72N^{1+\frac{4}{s}}}\sum_{i=1}^{s}\sum_{j=1}^{s}\Big(\frac{\partial^2 f(\mathbf{c}_k)}{\partial x_i\partial x_j}\Big)^2$$

$$-\frac{1}{120N^{1+\frac{4}{s}}}\sum_{i=1}^{s}\Big(\frac{\partial^2 f(\mathbf{c}_k)}{(\partial x_i)^2}\Big)^2+o(\frac{1}{N^{1+\frac{4}{s}}}) \tag{4.14}$$

现在把式(4.14) 和式(4.13)代入式(4.12)，有

$$\mathrm{E}(F_k - I_k)^2 = \frac{1}{288N^{2+\frac{4}{s}}}\sum_{i=1}^{s}\sum_{j=1}^{s}\Big(\frac{\partial^2 f(\mathbf{c}_k)}{\partial x_i \partial x_j}\Big)^2 - \frac{1}{480N^{2+\frac{4}{s}}}\sum_{i=1}^{s}\Big(\frac{\partial^2 f(\mathbf{c}_k)}{(\partial x_i)^2}\Big)^2 + o(\frac{1}{N^{2+\frac{4}{s}}})$$

所以

$$\begin{aligned}\mathrm{E}(F-I)^2 &= \sum_{k=1}^{N}\mathrm{E}(F_k - I_k)^2 \\ &= \frac{1}{288N^{2+\frac{4}{s}}}\sum_{k=1}^{N}\sum_{i=1}^{s}\sum_{j=1}^{s}(1-\frac{3}{5}\delta_{ij})\Big(\frac{\partial^2 f(\mathbf{c}_k)}{\partial x_i \partial x_j}\Big)^2 + o(\frac{1}{N^{1+\frac{4}{s}}}) \\ &= \frac{1}{288N^{1+\frac{4}{s}}}\lim_{N\to+\infty}\frac{1}{N}\sum_{k=1}^{N}\Big[\sum_{i=1}^{s}\sum_{j=1}^{s}(1-\frac{3}{5}\delta_{ij})\Big(\frac{\partial^2 f(\mathbf{c}_k)}{\partial x_i \partial x_j}\Big)^2\Big] + o(\frac{1}{N^{1+\frac{4}{s}}})\end{aligned}$$

因为函数f的二阶导数连续，所以$\sum_{i=1}^{s}\sum_{j=1}^{s}(1-\frac{3}{5}\delta_{ij})\Big(\frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j}\Big)^2$也是连续函数。于是将对下标$k$的求和改成积分，得到

$$\mathrm{E}(F-I)^2 = \frac{1}{288N^{1+\frac{4}{s}}}\int_{I^s}\sum_{i=1}^{s}\sum_{j=1}^{s}(1-\frac{3}{5}\delta_{ij})\Big(\frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j}\Big)^2 \mathrm{d}\mathbf{x} + o(\frac{1}{N^{1+\frac{4}{s}}})$$

证明完毕。

只要将定理4.2.4作适当修改，就可以得到

推论 4.2.5 *记$\Pi_2(I^s)$为$I^s=[0,1]^s$上次数不高于2的多变量函数类，假设$f\in\Pi_2(I^s)$，则AMC估计的方差有如下精确估计式:*

$$\mathrm{E}(A-I)^2 = \frac{1}{288N}\sum_{i=1}^{s}\sum_{j=1}^{s}(1-\frac{3}{5}\delta_{ij})\Big(\frac{\partial^2 f(\mathbf{c})}{\partial x_i \partial x_j}\Big)^2 \tag{4.15}$$

这里$\mathbf{c}=(\frac{1}{2},\cdots,\frac{1}{2})^{\mathrm{T}}$，$N$是所用的随机数点数。

从定理4.2.4得知FAMC方法的均方根误差阶是$O(N^{-(\frac{1}{2}+\frac{2}{s})})$。这个误差精

度比MC方法的高。推论4.2.5 告诉我们AMC方法的误差阶$O(N^{-\frac{1}{2}})$的系数比MC方法的误差阶的系数小。尤其对线性函数，AMC方法的误差为零。我们在第4.4节中给出数值实验。

4.3 对偶变数蒙特卡罗积分方法的并行程序

我们编写了MPI并行程序，并在中国科学院科学与工程计算国家重点实验室的“大规模科学计算研究”微机机群系统(主机名为lssc.cc.ac.cn)上运行做数值实验。

串行程序N个随机数的函数值在一个微机上计算，并行程序是分在$noprocs$个处理器上计算。为了保证随机数的独立性，我们只用一个随机数发生器产生一条序列，所以在各处理器之间要传递随机数种子。MPI程序的初始化用下列语句

```
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD,&nid);
MPI_Comm_size(MPI_COMM_WORLD,&noprocs);
```

其中nid是赋给每个处理器的序号，为0至$noprocs-1$的值。我们让每个处理器一次产生一批共sub_seq_len个随机数,会有不凑整的数，记为$remainder$。

```
procs_step=sub_seq_len*noprocs;
remainder=N%sub_seq_len;
size=N-remainder;
```

产生随机数后然后把随机数发生器的种子($seeds[NTAB+3]$)传给下一个处理器，再来计算这sub_seq_len个随机数的函数值，于是在这个处理器(nid)计算函数值的同时，下一个处理器(nid_after)可以重复同样的操作，即产生随机数，传递随机数种子，计算函数值。这样除了产生sub_seq_len个随机数及传递随机数种子的时间外，其他时间各处理器都在并行计算函数值。每个处理器在产生随机数之前要知道从哪个处理器(nid_before)那里接收种子。

因为对AMC和FAMC方法，需要把产生的随机数映射到每个小区间上，所以随机数的序号$(0-(N-1))$是很重要的，我们用nid来控制这个序号，下面的积分估计调用(MC())中$k+m$为随机数的序号。

```
for(k=nid*sub_seq_len;k<N;k+=procs_step) {...
if(k+sub_seq_len<=size)
  { for(m=0;m<sub_seq_len;m++)
```

```
            MC(k+m,n,random[m],interval,sum);
    }
  else
    { for(m=0;m<remainder;m++)
            MC(k+m,n,random[m],interval,sum);
    }
  ... }
```

可以看出，第0号处理器的k是从0开始的，所以N个随机数中的第0个随机数必须在第0号处理器上计算。且每次N个随机数计算完毕，开始新一轮N个随机数的计算时都必须在0号处理器上进行。所以我们必须记下处理最后一个随机数(序号为$N-1$)的处理器序号($last\_nid$)，当它产生完sub_seq_len个随机数或$remainder$个随机数后将随机数种子传给第0号处理器。而0号处理器在接收随机数种子时也要判断是从$noprocs-1$号处理器还是从处理第$N-1$个随机数的处理器接收，用$last_nid$记0号处理器的随机数种子的接收源。

```
for(i=0;i<runs;i++) {for(j=1;j<=step;j++) {
 N=points_step[j];
 ...
 last_nid =noprocs-1;
 for(k=nid*sub_seq_len;k<N;k+=procs_step)
 {/* receive the random seeds*/
  if(nid)   /*nid_before=(nid+noprocs-1)%noprocs;*/
      nid_before=nid-1;
  else
      nid_before=last_nid;
  MPI_Irecv(seeds,NTAB+3,MPI_LONG,nid_before,10,MPI_COMM_WORLD,&
        req_recv_seeds);
  MPI_Wait(&req_recv_seeds,&status);
  ...
  nid_after =(nid+1)%noprocs;
  /*in general the (noprocs-1)'th process send the seeds to process 0*/
  last_nid =noprocs-1;
  /*the process who deal with the N'th random number will send the seeds to process 0
    then process 0 start the first random number of new successive N random points
  */
  if((k+sub_seq_len==N)||k+remainder==N)
  {   nid_after =0;
      last_nid =nid;
      /*boadcast so that process 0 know the change of last_nid*/
      MPI_Bcast(&last_nid,1,MPI_INT,nid,MPI_COMM_WORLD);
  }
  MPI_Isend(seeds,NTAB+3,MPI_LONG,nid_after,10,MPI_COMM_WORLD,&
```

```
        req_send_seeds);
  ...
}
...
}/*end of j: step*/ ... } /*end of i :runs*/
```

在各处理器上算完N个随机数的函数值后用

```
MPI_Reduce(sum,G_sum,4,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
```

语句将部分函数值和再求和。
并行程序以

```
MPI_Finalize();
```

语句结束。完整的程序列在附录中。

4.4 数值实验

我们用FAMC方法和AMC方法计算了很多经典函数，并跟MC方法的结果进行比较。记$Y_N^{(1)} = M$，$Y_N^{(2)} = A$，$Y_N^{(3)} = F$。令$m = 75$，即每种估计运行75次，计算式(3.7)所定义的*sd*误差和式(3.8)所定义的均方根误差*rmse*。用的伪随机数是文[26]中的*ran2*发生器产生的。数值例子均来自文[36][37]。

例1.

$$I_1 = \int_0^1 \int_0^1 \int_0^1 \int_0^1 \frac{4x_1x_3^2 \exp(2x_1x_3)}{(1+x_2+x_4)^2} \mathrm{d}x_1\mathrm{d}x_2\mathrm{d}x_3\mathrm{d}x_4 \tag{4.16}$$

例2.

$$I_2 = \int_0^1 \cdots \int_0^1 \prod_{i=1}^{s} \frac{1+3(x_i)^2}{2} \mathrm{d}x_1 \cdots \mathrm{d}x_s, s = 10 \tag{4.17}$$

例3.

$$I_3 = \int_0^1 \cdots \int_0^1 \exp \sum_{i=1}^{s} \frac{x_i}{i} \mathrm{d}x_1 \cdots \mathrm{d}x_s, s = 15 \tag{4.18}$$

积分精确值分别是$I_1 = 0.5753$，$I_2 = 1.0$，$I_3 = 5.610253495$。我们记误差阶形式为$O(N^\alpha)$，从理论结果可知$\alpha(\mathrm{MC}) = \alpha(\mathrm{AMC}) = -\frac{1}{2}$以及$\alpha(\mathrm{FAMC}) = -(\frac{1}{2} + \frac{2}{s})$。FAMC方法的*rmse*误差列在表4.1–4.3中。表的第一行是N的大小，即在估计积分值时用了$N = n^s$个随机数点。α(FAMC)的理论值和对应的数值计算*rmse*误差的斜率值(使用的数据取过对数)列在

表 4.1: 数值例子1的$rmse$误差，维数$s = 4$

	16	81	256	625	1296	2401	4096
MC	0.26816	0.12726	0.07522	0.04744	0.03278	0.02726	0.01828
AMC	0.19763	0.08531	0.04605	0.03064	0.02021	0.01497	0.01050
FAMC	0.09145	0.01912	0.00774	0.00302	0.00140	0.00082	0.00043

表 4.2: 数值例子2的$rmse$误差，维数$s = 10$

	1024	59049	1048576	9765625	60466176	282475249
MC	0.070045	0.010355	0.002235	0.000735	0.000299	0.000127
AMC	0.042890	0.006604	0.001491	0.000522	0.000186	0.000085
FAMC	0.017122	0.001541	0.000209	0.000041	0.000013	0.000004

表4.4中。而对于AMC方法和MC方法，误差sd画在图4.1–4.6中。图中数据是取过对数后的值，这样斜率便对应α值，截距对应误差阶的系数。

我们的计算结果显示：

- 从表4.1–4.3知FAMC方法的误差阶远小于AMC方法和MC方法的误差差；
- 表4.4中线性拟合的斜率与理论α(FAMC)吻合得很好。比如，当维数$s = 10$，α(FAMC) $= -0.7$，斜率是-0.67；
- 图4.1–4.6显示AMC方法与MC方法的比较，其误差阶$O(N^{-\frac{1}{2}})$(对应斜率)相同，但前者的系数(对应截距)比后者的系数小。

表 4.3: 数值例子3的$rmse$误差，维数$s = 15$

	32768	14348907	1073741824
MC	0.0093009	0.0004967	0.0000651
AMC	0.0023853	0.0001140	0.0000118
FAMC	0.0006000	0.0000134	0.0000008

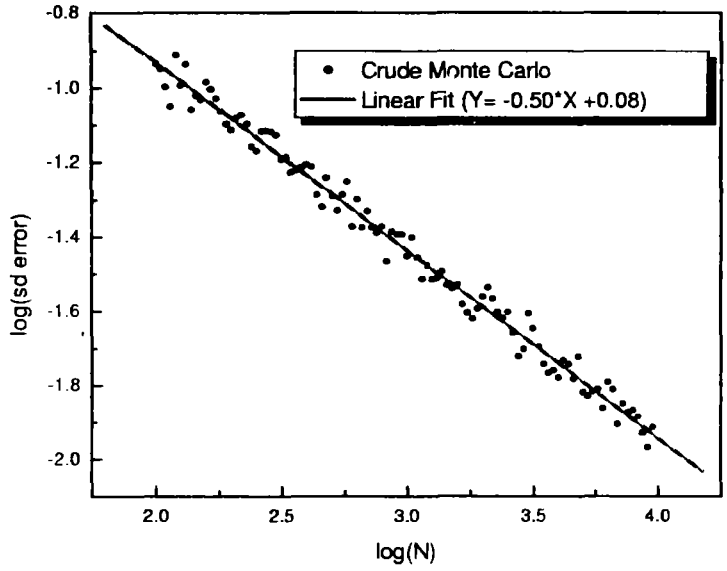

图 4.1: 数值例子1用MC估计的sd误差

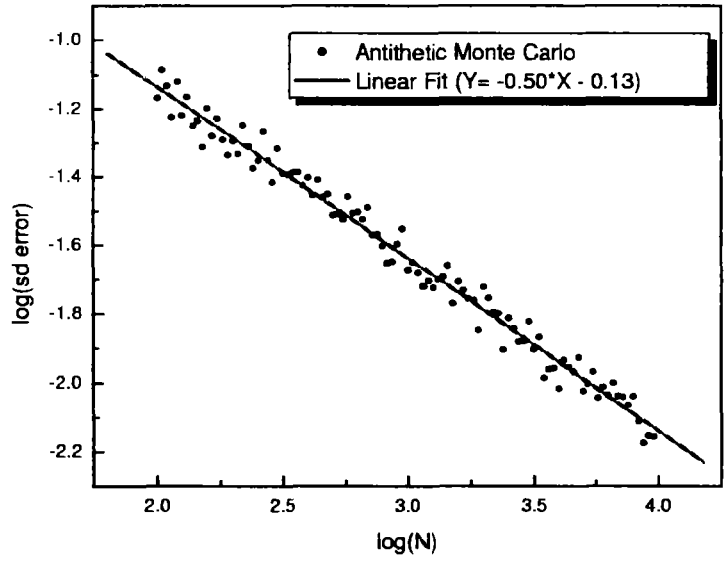

图 4.2: 数值例子1用AMC估计的sd误差

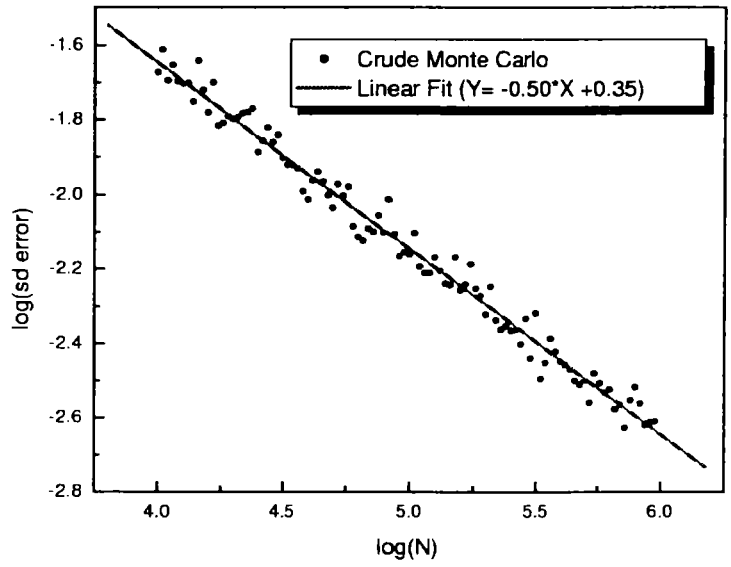

图 4.3: 数值例子2用MC估计的sd误差

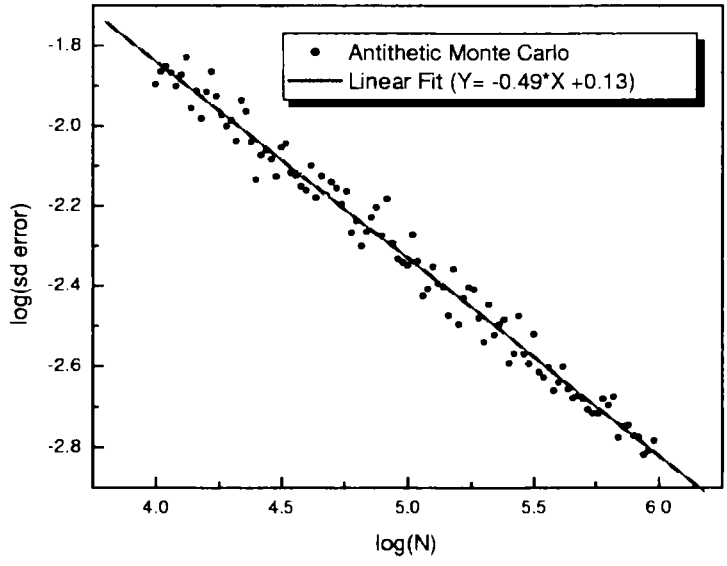

图 4.4: 数值例子2用AMC估计的sd误差

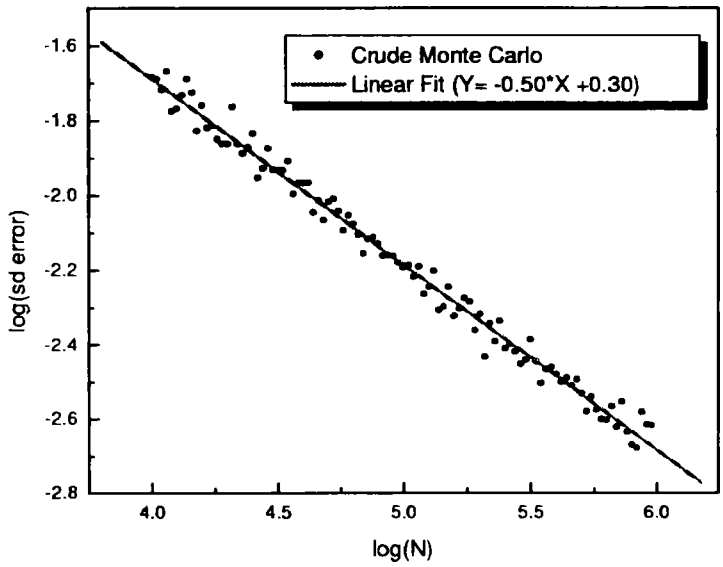

图 4.5: 数值例子3用MC估计的sd误差

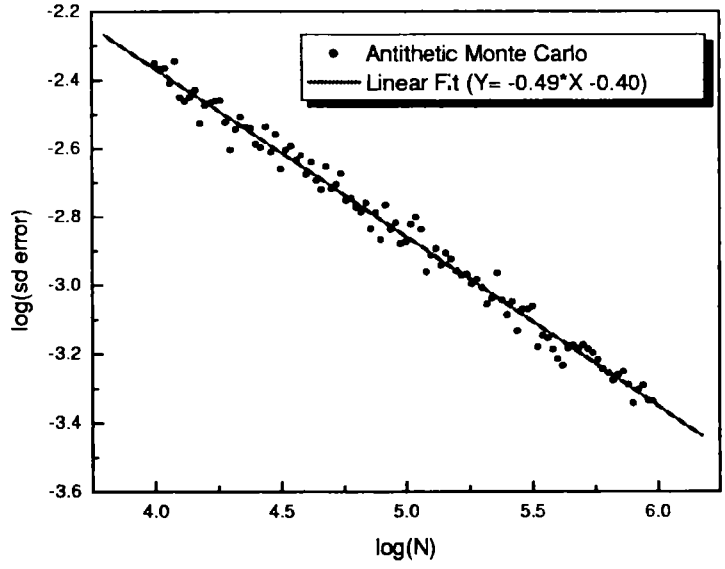

图 4.6: 数值例子3用AMC估计的sd误差

表 4.4: 理论α(FAMC)值与FAMC数值估计的$rmse$误差(线性拟合的斜率)的比较

	s	α(FAMC)	slope
Example 1	4	-1.00	-0.96
Example 2	10	-0.70	-0.67
Example 3	15	-0.63	-0.64

4.5 结语

无论是从理论结果还是数值实验，我们都能得出结论：对二阶导数连续的函数类，FAMC估计的误差阶是$O(N^{-(\frac{1}{2}+\frac{2}{s})})$，优于AMC估计和MC估计的误差阶。数值实验同时印证了“AMC估计与MC估计有相同的误差阶$O(N^{-\frac{1}{2}})$，但前者阶的系数比后者阶的系数小”的理论结果。

第五章 自适应拟蒙特卡罗全局优化方法

在不可微优化应用中拟蒙特卡罗搜索是非常有用的。我们提出了自适应拟蒙特卡罗全局优化((AQMC))算法。首先，在文[38]中提出，在局部搜索中，自适应技术根据搜索中间结果来调整局部搜索的搜索方向和搜索半径以及搜索所计算的函数值个数，从而使局部搜索得以快速找到局部极值。然后，我们在文[39]中发展了自适应技术，即引进遗传算法中关于种群进化的概念，根据种群的演化度(evolution degree)自适应地增加新个体。对于具有低偏差的拟随机序列，新产生的点分布在之前产生的点的空隙中，这样就能保证函数的定义域空间能被均匀地搜索从而找到全局极值。总而言之，AQMC方法不仅可以加快局部搜索，还可以自适应平衡全局与局部搜索的需求。

5.1 引言

第1章中所提的简单优化算法虽然收敛，然而，收敛速度一般来说是非常慢的。为了加快搜索速度，Niederreiter和Peart在[40]文提出了一种叫做“局域化搜索”的技术，我们将其简称为LQMC方法。王元和方开泰[41]在1990年提出的序贯数论优化算法与此方法的思想一致。我们称王的方法为SNTO方法。

LQMC方法和SNTO方法的有效性很大程度上取决于条件$d_N < \varepsilon$是否满足，这里ε是局部搜索的搜索半径，取小于$\frac{1}{2}$的正数。文[42]指出散度$d_N \geq \frac{1}{2}N^{-1/s}$是$I^s = [0,1]^s$上$N$个点的散度(dispersion)能取得的最小界。因此N至少得取ε^{-s}数量级，这些算法才能收敛。另外，如果函数有很多局部极大值，尤其当局部极大值很接近M，那么“局域化搜索”就可能陷入局部极值而无法跳出，也就是找不到全局极大值。

在这一章我们提出了自适应拟蒙特卡罗全局优化(AQMC)算法，在局部搜索和种群演化过程中都使用了自适应技术。AQMC算法不仅较大地加

快了局部搜索速度，也平衡了全局与局部搜索的需求(自适应均衡)。算法描述在第5.3节，数值实验在第5.4节。

5.2 从LQMC到LAQMC

假设f是定义在超矩形区域$E=[\mathbf{a},\mathbf{b}],\mathbf{a},\mathbf{b}\in R^s$上。Niederreiter的局部搜索(LQMC)算法可描述如下：

- 步骤1(初始化): 产生N个拟随机数，用式$f(\mathbf{x}_m)=\max\limits_{1\le n\le N} f(\mathbf{x}_n)$找到最大值点$\mathbf{x}_m$；
- 步骤2(映射): 将这N个点映射到以$\mathbf{x}_m$为中心，ε_i为半径的s维超立方体内

$$g_C(\mathbf{x})=\mathbf{x}_m+\varepsilon_i(2\mathbf{x}-(\mathbf{a}+\mathbf{b}))$$

- 步骤3: 寻找新的最大值点$\mathbf{x}_m$使其满足

$$f(\mathbf{x}_m)=\max(f(\mathbf{x}_m),\max_{1\le n\le N} f(g_C(\mathbf{x}_n)))$$

- 步骤4: 重复步骤2和步骤3直到搜索半径接近零。

这里N的数量级是$O(\varepsilon^{-s})$，搜索半径ε_i每一次映射逐次递减，一般令$\varepsilon_i=\varepsilon^i, 0<\varepsilon<1/2$(见图5.3)。我们称每一个映射步为一代(generation)。

为了加快搜索速度，我们改进了局部搜索算法使得我们用少得多的随机数点就能找到局部极值。我们称改进后的局部搜索算法为LAQMC方法。LAQMC方法有时也能跳出局部极值而找到全局极值。

LAQMC与LQMC算法在三个方面有很大不同。在记号上我们用下标i表示第i代。搜索方向和搜索半径ε_{ik}将根据已有的搜索结果调整，另外，每次局部搜索中所用的随机数点N_i与搜索半径ε_{ik}成正比。

对选中做局部搜索的个体$\mathbf{x}_{ik}$，将点$\mathbf{x}_1,\cdots,\mathbf{x}_N$的前$N_i$个点用函数$g_C: E\to C$映射到$\mathbf{x}_{ik}$的邻域。.

$$1\le N_i=\lfloor c_2\times N\times\max\{\varepsilon_{ik},c_1\}\rfloor\le N,\quad 0<c_1\le 1, 0<c_2\le 1 \tag{5.1}$$

$$g_C(\mathbf{x})=\mathbf{c}+\varepsilon_{ik}(2\mathbf{x}-(\mathbf{a}+\mathbf{b}))\quad \mathbf{x}\in E \tag{5.2}$$

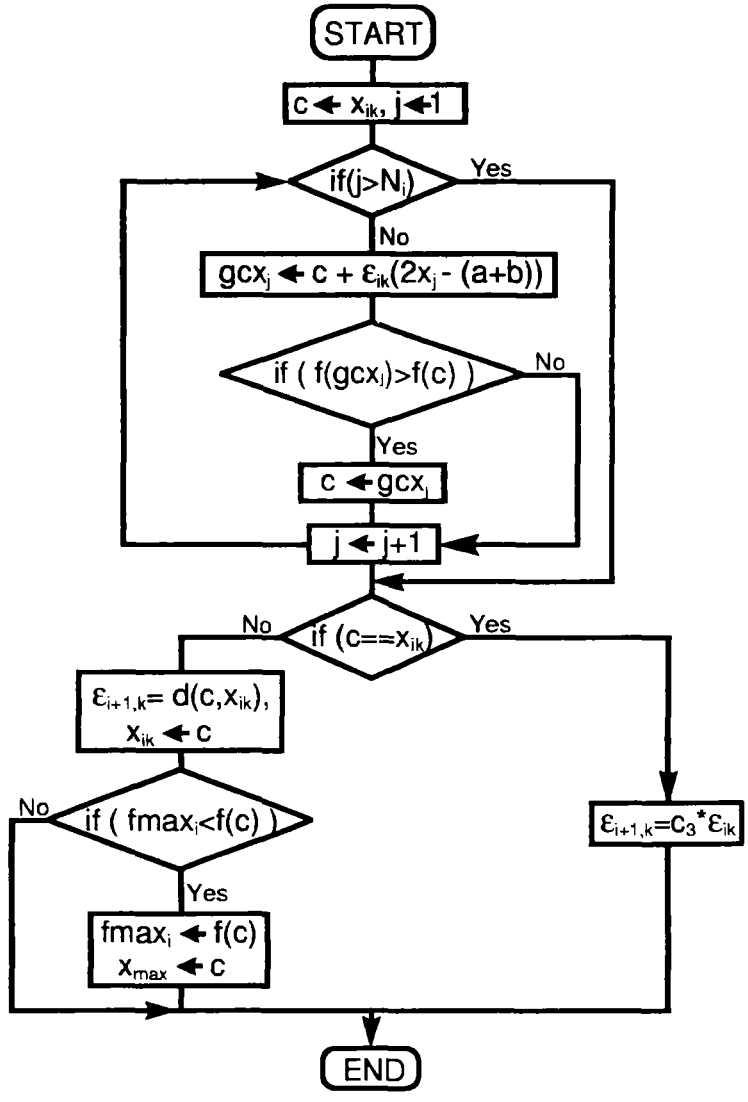

图 5.1: 自适应拟蒙特卡罗全局优化的局部搜索算法(LAQMC)的流程图

这里$[x]$表示小于x的最大整数。值$\mathbf{c}$最初设置为$\mathbf{x}_{ik}$，如果有$f(g_C(\mathbf{x}_j)) > f(\mathbf{c}), j = 1, \cdots, N_i$，则将$\mathbf{c}$重置为$g_C(\mathbf{x}_j)$。如图5.1的流程图(flow chart)所示，下一次第k个个体的搜索半径$\varepsilon_{i+1,k}$会随着这次的搜索结果而调整。如果局部搜索中找到了比$f(\mathbf{x}_{ik})$大的函数值，则令$\varepsilon_{i+1,k} = d(\mathbf{c}, \mathbf{x}_{ik})$，同时点$\mathbf{x}_{ik}$被新点$\mathbf{c}$取代；否则，搜索半径将递减，令$\varepsilon_{i+1,k} = c_3 \times \varepsilon_{ik}$，这里$0 < c_3 < \varepsilon_0$，经验显示取值$c_3 = \varepsilon_0^3$较好。

LQMC

```
r=0.25;//search radius
a=x[m];//the center

for(i=1;i<=N;i++)
 {
  gcx[i]=a+r*(x[i]-1);
 }


r=r*r;
```

LAQMC

```
r=0.25;//search radius
a=x[m];//the center
Ni=int(c2*N*max(r,c1));
for(i=1;i<=Ni;i++)
 {gcx[i]=a+r*(x[i]-1);
  if(f(gcx[i])-f(a)>1.0E-8)
      a=gcx[i];
 }
tempx=fabs(x[m]-a);
if(tempx>1.0E-8)
    r=tempx;
else
    r=c3*r;//c3<1;
```

图 5.2: LQMC算法与LAQMC算法对比的部分C语言代码

我们给出LAQMC方法和LQMC方法的部分C语言代码(图5.2)，为简便起见，设维数s为1。

在第5.4节中对LAQMC方法和LQMC方法的数值试验显示：AQMC算法的自适应局部搜索大大节省了计算量。

5.3 自适应拟蒙特卡罗全局优化算法

但是对于全局优化问题，局部搜索是不够的。我们借用遗传算法(Genetic Algorithms)[43][44][45][46] 中的种群演化的概念，提出了自适应拟蒙特卡罗全局优化算法。我们取无穷长拟蒙特卡罗序列的前N(对大的维数s，N可以取相对较小的值)个点$\mathbf{x}_1,\cdots,\mathbf{x}_N$映射到函数空间$E$上作为初始种群，每一个点是一个个体。首先计算每个个体的适应度(fitness)，然后选择一个个体(选择概率正比于适应度)做自适应局部搜索。根据种群演化度(evolution degree)Ed自适应地增加新的个体，Ed的值大，引进新个体的概率就大。如我们在第2.2节所讨论的，新产生的相继拟随机数点填充在之前产生的点在函数空间E的分布的空隙中，这就保证函数空间E能被均匀地搜索从而找到全局极大值。记$fmax_i = \max\limits_{\substack{1\le k\le i\\ 1\le j\le N}} f(\mathbf{x}_{kj})$，而对应的极大值点是$\mathbf{x}_{max}$。则非递减序列$fmax_1, fmax_2,\cdots$可以看作函数$f$的上确界$M$的近似值。

定义 5.3.1 (适应度(fitness)) 令$F_{ij} = f(\mathbf{x}_{ij}) - Cmin$，这里$Cmin$是至当前代所有个体的函数值的最小值，$f(\mathbf{x}_{ij})$是第$i$代种群第$j$个个体的函数值，则

$$p_{ij} = F_{ij} / \sum_{k=1}^{N} F_{ik}$$

表示第i代种群第j个个体的适应度。显然

$$p_{ij} \geq 0, j = 1, \cdots, N, \quad \sum_{j=1}^{N} p_{ij} = 1.$$

定义 5.3.2 (演化度(evolution degree)) 第i代种群所有个体的平均函数值记作$m_i = \sum_{j=1}^{N} f(\mathbf{x}_{ij})/N, i = 1, \cdots$。$m_0$的初始值置为$m_1$，一旦第$i$代种群中差的个体被序列中新产生的点取代，令$m_0 = m_i$。则

$$Ed_i = |1 - m_i/m_0| \quad i = 1, \cdots$$

称作第i代种群的演化度。

演化度Ed反应的是局部搜索能力的饱和度。对初始种群，$Ed = 0$，产生新个体的的概率为0，局部搜索占绝对优势。当局部搜索逐步提高极大值的近似值，局部搜索能找到更优点的可能性降低，此时演化度Ed增加，产生新个体的概率得到提高。当产生新的一批拟随机数点时，种群得到了更新，重新置演化度$Ed = 0$，开始新的演化。由此可以看出，我们更注重局部搜索，引进新个体只是为了保证搜索不致于陷在某个局部极值点的邻域。

现在我们将AQMC算法完整地描述如下：

- 步骤1:

  1. 产生序列的最初N个点$\mathbf{x}_1, \cdots, \mathbf{x}_N$，令$i = 1$，记初始种群的个体$\mathbf{x}_{ij} = \mathbf{x}_j$，搜索半径为$\varepsilon_{ij} = \varepsilon_0 (0 < \varepsilon_0 < \frac{1}{2})$，$j = 1, \cdots, N$，计算适度度$p_{ij}$；
  2. 令$fmax_i = f(\mathbf{x}_{ik}) = \max\limits_{1 \leq j \leq N} f(\mathbf{x}_{ij})$，对应的最大值点为$\mathbf{x}_{max} = \mathbf{x}_{ik}$；
  3. 计算m_1的值并令$m_0 = m_1$；

- 步骤2:

  如果(满足停止条件):

  程序结束

  否则:

  (a) $i = i + 1$

  (b) 根据适应度p_{ij}选择一个个体做自适应局部搜索(LAQMC)，$fmax_i$ 的值在局部搜索后可能被改变;

- 步骤3: 计算p_{ij}, m_i, Ed_i;

- 步骤4:

  1. 产生随机数$newp$;
  2. 如果($newp < Ed_i$):
     (a) 从序列产生$c_4 \times N(0 < c_4 \leq 1)$个相继的拟随机数替代第$i$代种群中较差的个体，新产生的个体的搜索半径$\varepsilon_{ij}$取初始值$\varepsilon_0$;
     (b) 记新产生的个体的最大函数值为$tempx$，如果有$fmax_i < tempx$，则令$fmax_i = tempx$;
     (c) 计算p_{ij}和m_i，令$m_0 = m_i$;
  3. 转向步骤2。

程序停止的条件可以有不同的方式。比如，如果在若干代后函数最大值$fmax_i$还没有改变就停止程序。我们也可以事先设置总的演化代数。另外，有很多实际问题是寻找最优参数，也就是说最大值是已知的，我们就可以控制$fmax_i$和全局极值之间的误差。

5.4 数值实验

我们用AQMC算法对若干经典函数做了数值实验，所用的拟随机数是Sobol'序列[26][25]。下面的数值例子中f_1来自文[44]。我们将AQMC算法的结果与LQMC和SNTO算法的结果进行了比较，AQMC算法的有效性是非常明显的。

例1:

$$f_1 = 100(x_1^2 - x_2)^2 + (1 - x_1)^2, \quad -2.408 \leq x_i \leq 2.408 \tag{5.3}$$

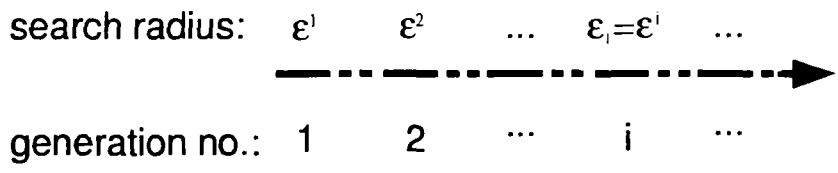

图 5.3: Niederreiter的局域化搜索(LQMC)每代的搜索半径的变化

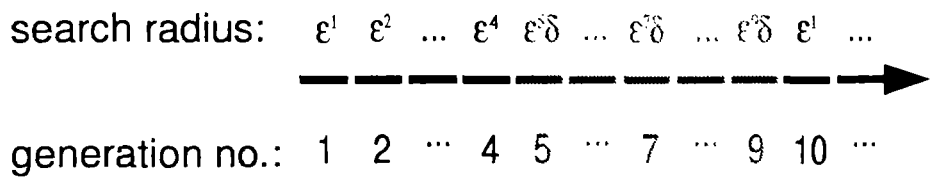

图 5.4: Niederreiter的局域化搜索(LQMC)外迭代方法的每代的搜索半径的变化(见式(5.4))，比较图5.3

是一个很难极小化的函数。全局极小值点在(1.0, 1.0)点，全局极小值是0。我们取$N = 64, \varepsilon_0 = 0.25, \delta = 4.0$并运行了81920次。对LQMC方法，全局极小值和近似值之间的误差均大于阶$O(10^{-4})$。

对LQMC方法的外迭代过程[40]，如果设(见图5.4)

$$\left.\begin{array}{ll} \varepsilon_i = \varepsilon_{i-1} \times \delta & \text{if}(!i\%5) \\ \varepsilon_i = \varepsilon_0 & \text{if}(!i\%10) \end{array}\right\} \tag{5.4}$$

则在第45代找到最小值(见图5.5 LQMC)。同时，若用AQMC的自适应局部搜索(LAQMC)，参数$c_1 = 1.0$, $c_2 = 1.0$, $c_3 = \varepsilon_0^3$，则最小值在第4代找到(见图5.5 LAQMC)，且81920次的运行中有41885次(51.13%)能找到全局极小值。定义err为精确的极小值和近似的极小值之间的误差，图5.5显示AQMC的自适应局部搜索LAQMC比Niederreiter的局域化搜索方法LQMC强。

此外，AQMC的自适应局部搜索LAQMC可以避免LQMC可能陷入局部极值的缺陷。我们来考察下面的函数：

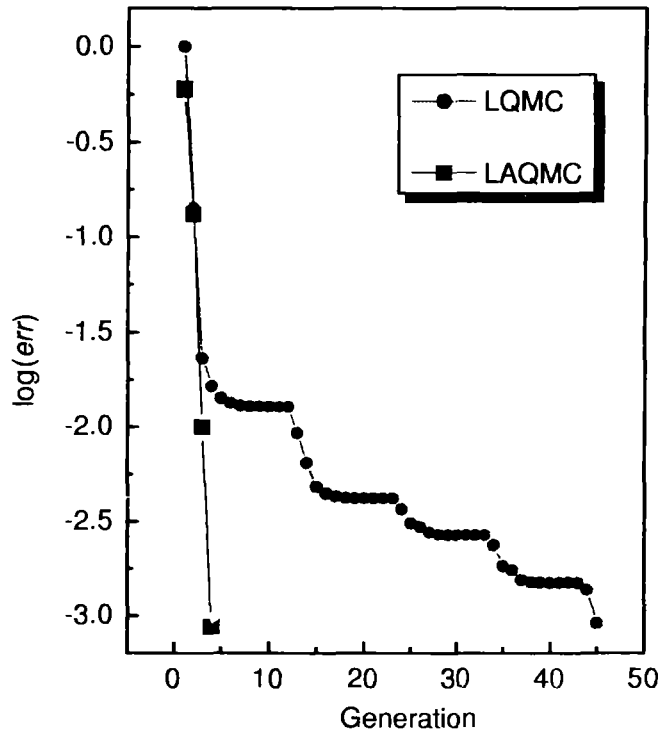

图 5.5: LQMC方法和LAQMC方法对函数$f_1(s=2)$的误差比较

例2:

$$f_2 = sA + \sum_{i=1}^{s}[x_i^2 - A\cos(2\pi x_i)], \quad -4.0 \le x_i \le 5.0, \ A \in R \tag{5.5}$$

此函数全局极小值是0。这里我们设常数$A=8$。对于二维($s=2$)情形，全局极小值点在$(0,0)$点，且在区域$[0,1]\times[0,1]$存在许多局部极小值点(见图5.6)。对LQMC方法，运行81920次有36320次(44.33%)找不到全局极小值。如果用LQMC的外迭代过程，搜索半径ε_i设置成式(5.4)的值，当随机数点数取$N=64$，此方法在第11代找到一个局部极小值点且停留在这一局部极小值点(见图5.7, LQMC N=64)；只有当N增加到200，此方法到第25代(计算了5000个函数值)才找到全局极小值点(见图5.7, LQMC N=200)。而对于AQMC的自适应局部搜索LAQMC，情形与函数f_1类似，

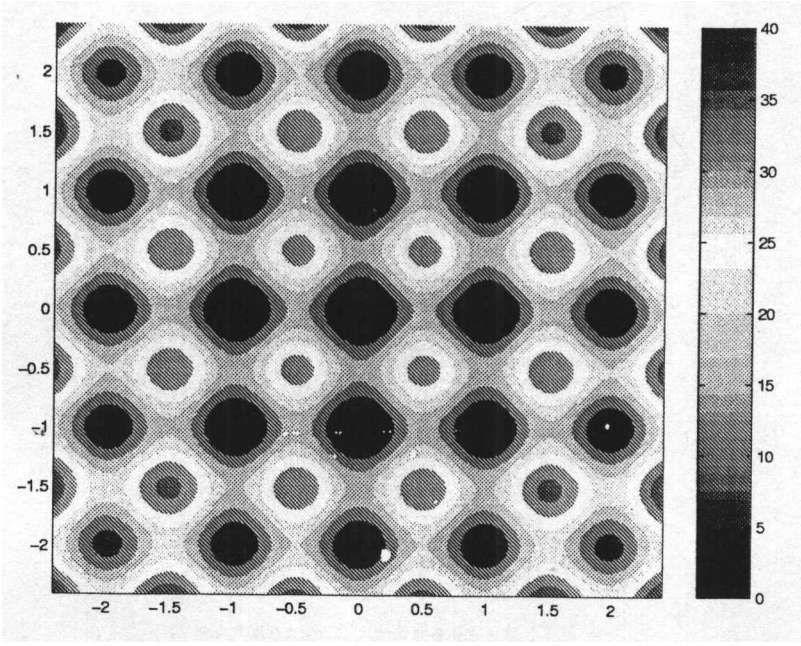

图 5.6: 数值例子2中函数的等高图

在第5代就找到了全局极小值(见图5.7, LAQMC N=64)。LAQMC方法的成功得益于自适应调整搜索方向和搜索半径以及搜索所计算的函数值个数。

AQMC方法不仅在局部搜索速度上快于LQMC方法，在全局搜索能力上更胜一筹。对于式(5.5)定义的函数f_2，如果维数$s = 6$，LQMC方法几乎找不到全局极小值。对于$N = 1024$，$\varepsilon_0 = 0.25$的情形，运行8192次有8092次(98.78%)找不到全局极小值。但AQMC最终能找到全局极小值。表5.1是AQMC方法对函数f_2在维数$s = 6$时的一个搜索结果，其中N_p是找到全局极小值所计算的函数值的个数，$fmin$是所找到的函数f_2的全局极小值的近似值。注意LQMC对于函数f_2的二维情形计算了5000个函数值，所以这里计算的函数值个数绝对值虽大，但与LQMC的二维情形相比，就不大了。

现在我们来比较AQMC方法和SNTO方法的结果。下面两个函数来自文[28]。

例3:

$$f_3(x,y,z,u) = \exp(xyzu)\sin(x+y+z+u), \quad (x,y,z,u) \in I^4.$$

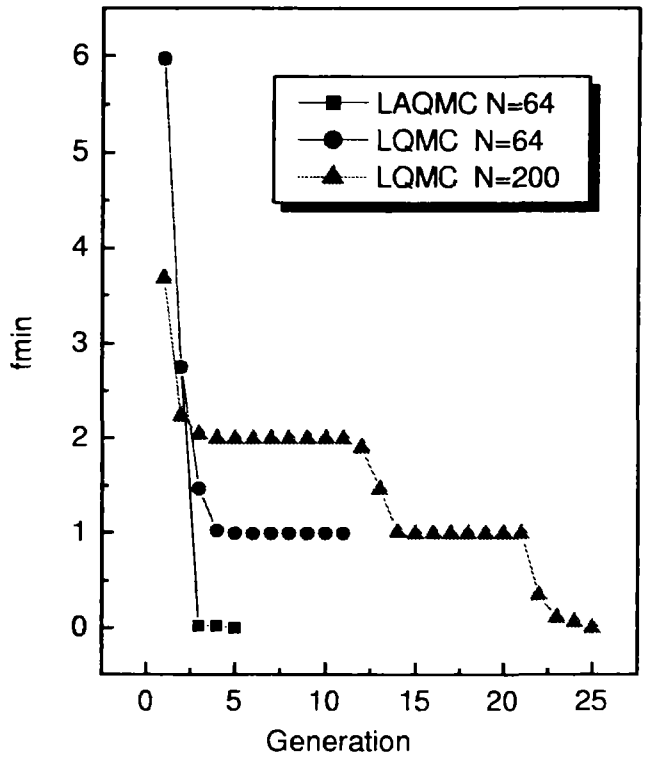

图 5.7: LQMC方法和LAQMC方法求函数$f_2(s=2)$的全局极小值的近似值$(fmin)$的比较

例4:

$$f_4(x,y,z,u) = -(x-\frac{3}{11})^2-(y-\frac{6}{13})^2-(z-\frac{12}{23})^2-(u-\frac{8}{37})^2,\ (x,y,z,u)\in I^4.$$

我们已知函数f_3的全局极大值是1.0261986，函数f_4的极大值是0。文[28]中的表3.4和表3.5显示无论对函数f_3还是函数f_4，SNTO方法都要在计算2000多个函数值后才能达到误差精度$O(10^{-7})$。而达到相同的精度，AQMC方法只需计算少于400个函数值。由此可见，AQMC方法的有效性是显著的。表5.2和表5.3 是AQMC方法的搜索结果。表中所有的记号都在第5.3节有所说明。

表 5.1: AQMC方法对函数f_2维数$s = 6$的全局优化结果。与LQMC对维数$s = 2$的结果相比，N_p的值并不大

c_1	c_2	c_3	c_4	N_p	$fmin$
0.040000	1.0	0.0625	0.25	134254	0.0000018688
0.050000	1.0	0.0625	0.25	145119	0.0000052123
0.080000	1.0	0.0625	0.25	190176	0.0000018547

表 5.2: AQMC方法对函数f_3的搜索结果(参数$c_1 = 0.5, c_2 = 1.0, c_3 = 0.0625, c_4 = 0.25$)

N_i	$fmax_i$	x	y	z	u
64	1.0170369	0.2187500	0.3437500	0.5312500	0.5937500
32	1.0250066	0.4375000	0.4375000	0.4375000	0.3125000
32	1.0250066	0.4375000	0.4375000	0.4375000	0.3125000
32	1.0256147	0.4340668	0.4310455	0.4357147	0.3428497
32	1.0261741	0.4150982	0.3969021	0.4091587	0.4149303
32	1.0261741	0.4150982	0.3969021	0.4091587	0.4149303
32	1.0261921	0.4139015	0.4027446	0.4100738	0.4123259
32	1.0261969	0.4100674	0.4065787	0.4098912	0.4125084
32	1.0261973	0.4115052	0.4080165	0.4084534	0.4120292
32	1.0261983	0.4106066	0.4085556	0.4100709	0.4104117
352					

表 5.3: AQMC方法对函数f_4的搜索结果(参数$c_1 = 0.5, c_2 = 1.0, c_3 = 0.015625, c_4 = 0.25$)

N_i	$fmax_i$	x	y	z	u
64	-0.06343331	0.32812500	0.67187500	0.45312500	0.10937500
32	-0.01104883	0.31250000	0.46875000	0.59375000	0.28125000
32	-0.01104883	0.31250000	0.46875000	0.59375000	0.28125000
32	-0.00684690	0.30773926	0.46875000	0.57788086	0.26538086
32	-0.00004844	0.27798462	0.46279907	0.52432251	0.21975708
32	-0.00004844	0.27798462	0.46279907	0.52432251	0.21975708
32	-0.00000507	0.27474183	0.46206683	0.52254421	0.21651429
32	-0.00000211	0.27291776	0.46105346	0.52274688	0.21712231
32	-0.00000013	0.27286076	0.46156648	0.52154983	0.21649529
320					

数值结果显示：

- AQMC方法是全局优化算法，而LQMC和SNTO方法可能陷入局部极值；
- AQMC的局部搜索LAQMC速度比LQMC和SNTO方法快差不多5倍；
- AQMC方法的种群大小远比LQMC和SNTO方法的采样大小少。

5.5 结语

如数值实验所示，自适应拟蒙特卡罗全局优化方法是一个全局算法。其自适应局部搜索充分利用搜索过的函数值自动调整搜索方向，能从一个点快速跳到其临近的局部极值点。对于连续性好的函数，局部搜索算法尤其加快了搜索速度。文中大多数的数值例子只要自适应局部搜索就能找到全局极值。另外根据种群的演化度可以在搜索过程中引进新个体，保证整个搜索算法收敛。

我们建议将AQMC的局部搜索方法用于遗传算法的局部搜索中，这两种方法的结合会有一定的前景。第6章就是结合遗传程序设计和AQMC的例子。

第六章 遗传程序设计与自适应拟蒙特卡罗优化方法在预测中的应用

在现实生活中存在许多随时间而变化的复杂系统和现象，人们通常需要根据动态系统的观测数据建立合理的微分方程模型(动力学模型)，为系统分析、设计和未来状态的预报提供依据。在这一章我们结合遗传程序设计和自适应拟蒙特卡罗全局优化方法解决预测问题。遗传程序设计算法用来优化常微分方程组右端的函数(模型结构)，自适应拟蒙特卡罗全局优化方法则用来优化函数的系数(模型参数)。

6.1 动态系统抽象模型

尽管那些随时间而变化的复杂系统存在于工程技术、经济管理、自然科学和社会科学等各种领域，我们都可以将预测问题抽象地概括成：由一组历史数据来决定新的数据。

$$\begin{pmatrix} x_1(t_1) & \cdots & x_1(t_m) \\ \vdots & \cdots & \vdots \\ x_n(t_1) & \cdots & x_n(t_m) \end{pmatrix} \Rightarrow \begin{pmatrix} x_1(t_{m+1}) & \cdots & x_1(t_{m+num}) \\ \vdots & \cdots & \vdots \\ x_n(t_{m+1}) & \cdots & x_n(t_{m+num}) \end{pmatrix}$$

这里有n个变量，$x_i(t_j)$表示第i个变量在t_j时刻的值。我们希望从$t_1,\cdots,t_m$时刻的历史数据知道各变量在$t_{m+1},\cdots,t_{m+num}$时刻的值。

由于各变量之间相互作用，所以这是一个复杂的动态系统。而常微分

方程组(ordinary differential equations) 可以描述这样一个动态系统。

$$\begin{cases} \dfrac{\mathrm{d}x_1}{\mathrm{d}t} = f_1(t, x_1, x_2, \cdots, x_n) \\ \dfrac{\mathrm{d}x_2}{\mathrm{d}t} = f_2(t, x_1, x_2, \cdots, x_n) \\ \quad \vdots \\ \dfrac{\mathrm{d}x_n}{\mathrm{d}t} = f_n(t, x_1, x_2, \cdots, x_n) \end{cases} \tag{6.1}$$

只要有常微分方程组，我们便可以用欧拉方法(Euler method)等方法从某一时间步的数据得到下一时间步的数据(回归数据(regression data))。

$$x_i'(t + \Delta t) = x_i(t) + f_i(t, x_1(t), x_2(t), \cdots, x_n(t)) \times \Delta t \tag{6.2}$$

我们的目标就是要找到“好”的函数f_i使得回归数据与实际数据拟合得很好。优化目标函数如下：

$$fit = \sum_{i=1}^{n} \sum_{j=1}^{m} (x_i'(t_j) - x_i(t_j))^2 \tag{6.3}$$

所谓“好”的函数就是使上式达到极小值0，一旦找到了好的函数f_i，我们就可以用欧拉方法(式(6.2))得到需要预测的数据。

6.2 遗传程序设计

遗传程序设计(Genetic Programming)[47][48]是从遗传算法发展来的，通过增加结构的复杂性可以更灵活地处理遗传算法中优化对象的表示问题。不同于常规遗传算法那样采用确定长度的染色体串，遗传程序设计中用到的一般是规模和形状能够动态变化的分层的计算机程序(computer program)。这里我们的优化对象是函数(数学表达式)，种群包含N个个体，每个个体是n个方程组。经过演化，可以从前一代种群得到下一代种群。

我们优化的对象是方程组，即n个函数，函数在计算机程序中可以用树这种数据结构表示。遗传程序设计算法的过程与标准遗传算法的一样，但这里演化算子操作的对象是树，而不是一般确定长度的染色体串(数值)。把优化对象树的表达方式和演化算子的操作方式讲明白之后，其它的演化过

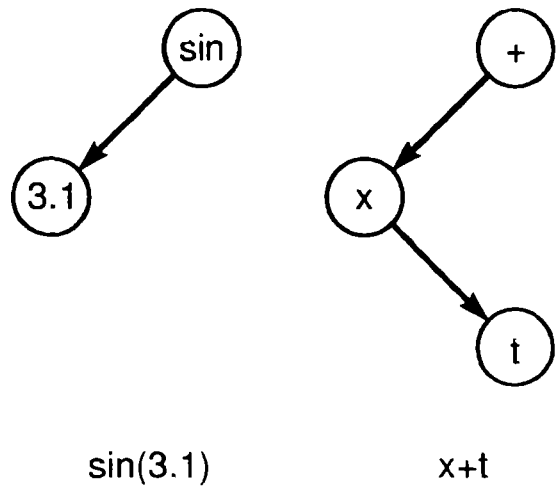

图 6.1: 二叉树表示函数，运算符的左子女是该运算符的第一个变量，运算符的左子女的右子女是该运算符的第二个变量

程与标准的遗传算法一样，步骤如下：

- 步骤1: (初始化)建立种群，种群包含N个个体，即生成N个方程组，每个方程组是n个函数(数学表达式)，即n棵树；
- 步骤2: 对每个个体根据式(6.3)计算适应度值；
- 步骤3: 由复制、变异、交叉算子产生下一代种群的个体(树)；
- 步骤4: 由AQMC算法优化函数系数;
- 步骤5: 转到步骤2。

我们用C++语言[49]编写程序，此节将结合部分C++代码说明程序的要点。首先说明如何用树来表示数学表达式，然后说明如何计算函数值，最后说明三种演化算子对树的操作方式。系数优化将在下一节单独给出。

每一个函数用一棵二叉树(binary tree)表示。如图6.1，对于单目运算符，只有一个左子女，如图中左边的树表示sin 3.1。而对于双目运算符，运算符结点的左子女是该运算符的第一个变量，运算符的左子女的右子女是该运算符的第二个变量。如图中右边的树表示$x+t$，其中第二个变量t是第一个变量x的右子女。之所以用这种二叉树而不是把二个变量都当作运算符结点的子女是为了代码的通用性。因为对于有三个分支的结构，如计算机

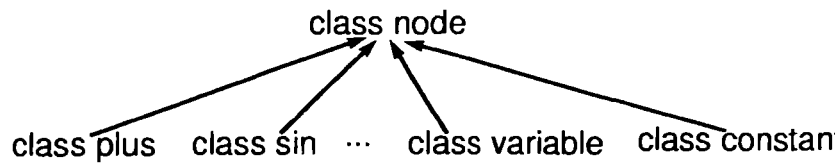

图 6.2: 运算符类，常数类和变量类都由基类“node”派生而来

程序，如果把分支都当作根结点的子女，则根结点就得有三个子女；对于有四个分支的结构，根结点得有四个子女；···。而我们这样表示的二叉树可以用统一的数据结构来表示这些不定分支的的计算机程序：把第三个分支当作第二个分支的右子女，第四个分支当作第三个分支的右子女···。

树是用链表(link)来表示的，因为已将所有的表达式用二叉树表示，每个结点只有两个子女，所以每个结点只需用两个指针(pointer)分别指向左子女和右子女。但是我们这里树的结点包含运算符“+”，“−”，“×”，“/”，“ sin”，“cos”，“ln”，“exp”，常数(如图6.1中“3.1”)和变量，这些元素不能用统一的数据结构来处理。C++语言中具有封装性、继承性和多态性的“类”(class) 非常适合用来表示这种异质结点的链表(树)。我们定义一个基类(base class)结点“node”，运算符类、常数类“constant”和变量类“variable” 都从基类“node”派生而来(图6.2)。把所有结点的共同的性质在基类中定义，如都有两个指针，都有整型变量“rank”来标记这些元素的序号，如“rank=0”表示这个结点是“+”，“rank=8”表示这个结点是常数，“rank=9”表示这个结点是变量“t”。事实上在用链表表示的树中并不存放“+”，“t”，“3.1”这些我们在图6.1中看到的表达式，放的只是标记的序号。基类中还有三个虚函数(virtual function)，copy_node()是用来拷贝结点得到相同的结点，eval()是计算以此结点为根的树的函数的值，estr()是返回以此结点为根的函数的数学表达式。但派生类又有各自不同的成员，如常数类“constant”有一个成员是储存这个常数的数值的。同时因为对数学运算符，常数和变量的操作方式不一样，基类中的虚函数在派生类中也将被重载。虚函数的重载将在后面函数值的计算中说明。

```
class node
{ public:
    friend class function;
    friend class individual;
    node *right,* left ;
    int rank;//node rank
```

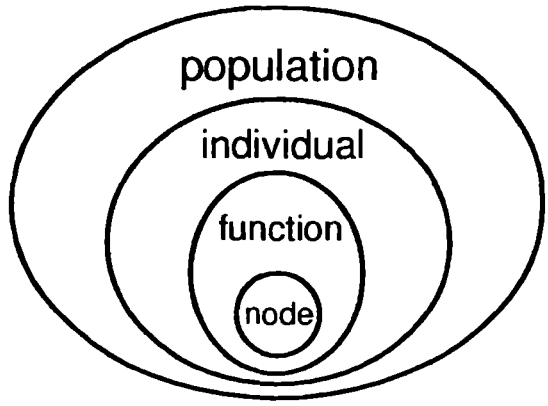

图 6.3: 所有类的关系，结点类(node)是函数类(function)的成员，函数类(function)是个体类(individual)的成员，依此类推……

```
    int tag;//if operand,tag is the number of variable node(int Rank)
    node(int Rank,int Tag)
   { rank=Rank;
     tag=Tag;
     right=NULL;
     left=NULL;
   }
   virtual node *copy_node(){return NULL;}// copy the node
   virtual double eval(){return 0.;}//get the value of the tree (function)
   virtual char *estr(){return NULL;}// get the expression of the tree
};

class constant:public node
{ public:
   double random_const;
   ...
};
```

函数值的计算是通过每个结点的成员函数eval()实现的。每棵树都有一个指针(node *root)指向这棵树的根结点，通过这个指针可以循着根结点的左子女及其左子女的子女找到这棵树的所有结点(树的周游)。我们只需调用根结点的eval()函数，C++便会根据根结点是属于哪种结点类自动调用相关的函数来计算函数值。我们前面说过结点有三种类型，即运算符，常数和变量，所以对不同的类型，虚函数eval()得有不同的处理方式，这就是我们说的虚函数重载。比如对于运算符结点"+"，前面曾提到它的第一个变量是其左子女，第二个变量是其左子女的右子女，于是就应该找到这两个变量

的结点，将以这两个结点为根的子树的值相加。

```
double plus::eval()
      { node *p=left;
        p=p->right;
        return left->eval()+p->eval();
       }
```

显然这是一个递归调用，因为在程序中要计算两棵子树的函数值。对于常数只需取出结点中储存的值就可以了。

```
double constant::eval()
       { return random_const;
       }
```

最关键的是变量结点，因为我们要从外界输入变量(如"t"，"x_1"等)的值，希望如平常的习惯直接调用$f_1(x)$就能得到函数f_1的值。我们用一个类"symbol_table"中的数组"table"来储存变量的表达式和值。在变量类"variable"中有一个成员"index"是用来对"symbol_table"的数组"table"进行索引的，如"index=0"表示对应数组"table" 中第一个元素即"t"，同时存有"t"这个字符串和它的值。

```
symbol_table sym_tab;
 class symbol_table
 {private:
    struct info
    { char var_name[3];/*to store the expression of the variable*/
      double var_value;/*to store the value of the variable*/
    };
    info table[n+1];//t,x1,x2
    int table_index;
 public:
    symbol_table();
    void add_value(int index,double r);
    void add_variable(char *s);
    char *get_name(int index);
    double get_value(int index);/*to get the value of the variable by index*/
    void clear();
};
```

变量结点的值是通过

```
double variable::eval()
{return sym_tab.get_value(index);
}
```

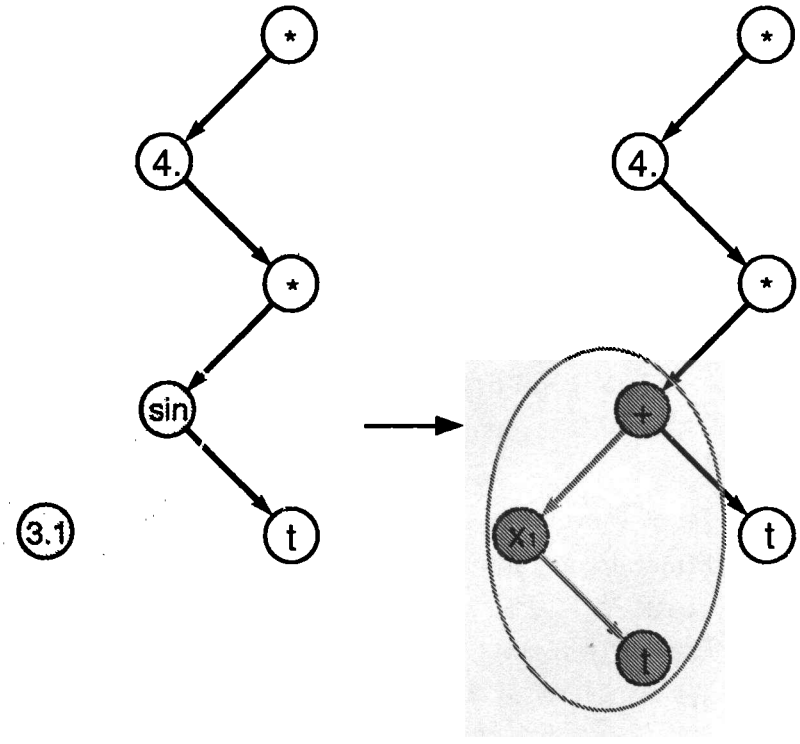

图 6.4: 种群演化的变异算子。先随机选择树的一个结点，该结点的左子树"sin(3.1)"被新产生的树"$x_1 + t$"所代替，该结点的右子女("t")保持不变。

来获得。为了直接调用$f_1(x)$就能得到函数f_1的值，我们用了运算符重载(operator overloading)方法：

```
double function::operator()(double x[])
{double temp;
 for(int i=0;i<=n;i++)
   sym_tab.add_value(i,x[i]);/*x[0]=t, x[1]=x1*/
 temp=root->eval();
 return temp;
}
```

可以看出这段程序里调用了类"symbol_table"的函数add_value()，这样就将x(x在这里是一个数组)的值传给数组"table"，更新了所有变量的值，然后调用树的根结点的eval()函数，通过递归调用得到我们需要的函数值。

用类似的原理，我们可以通过调用root->estr()来写出树所表示的数学

表达式，此函数也是用递归方法，对不同类的结点也需对虚函数进行重载。

至此，我们已经将函数的表达和函数值的计算作了讲解，有了类"node"(表示树结点)，类"function"(表示一个函数)，我们还需要另外一些类如类"individual"(表示一个方程组)和类"population"(表示种群，即一组方程组)，这些类组成一个从下到上的包含关系(member)(图6.3)，如类"node"是类"function"的成员，类"function"是类"individual"的成员，依此类推……这样整个完整的数据结构便架好了。接下来解释演化算子对树的操作方式。

三个演化算子(evolution operator)分别是复制算子(duplicate operator)，变异算子(mutate operator) 和交叉算子(cross operator)，它们的操作对象都是树(表示函数)。复制算子是简单地拷贝一棵树，即生成一棵完全相同的树。变异算子(图6.4)对被选中做变异的树先随机选择树的一个结点(如图6.4中的结点"sin")，然后寻找该结点的左子树("sin(3.1)")，即该结点及其以该结点的左子女为根的树，不包含该结点的右子女(结点"t")，然后用一棵新树替换此子树。之所以不能替换该结点的右子女是因为该结点的右子女(结点"t")是该结点的父母("$*$")的变量(与前面关于二叉树的描述相对应)。图6.4中子树"sin(3.1)"被新产生的树"$x_1 + t$"所代替，原来的函数"$4.*(\sin(3.1)*t)$"变异成函数"$4.*((x_1+t)*t)$"。交叉算子(图6.5)是对被选中做交叉的两棵树先随机选择两棵树上各一个结点，然后寻找这些结点的左子树，分别交换这两棵左子树。所有对树的操作都要注意保持结点的右子女，使得树所表示的函数表达式有意义。

程序中多个地方用了树的周游算法，需要用到堆栈(stack)，堆栈其实是一个数组，只是对这个数组定义了一组函数，使得对它的数据的操作方式是先进后出。为了堆栈能处理不同的数据类型，如有时要处理的是指针，有时要处理的是数值，我们使用了模板(template)。如前面提到的二叉树可以表示多个分支，使用模板也是使代码更通用的做法。

```
template <class Tdatatype,int m0> class TStack
{public:
    friend class function;
    TStack(int t_ini,int StackDepth_ini)
   { t=t_ini;
    StackDepth=StackDepth_ini;
    }
```

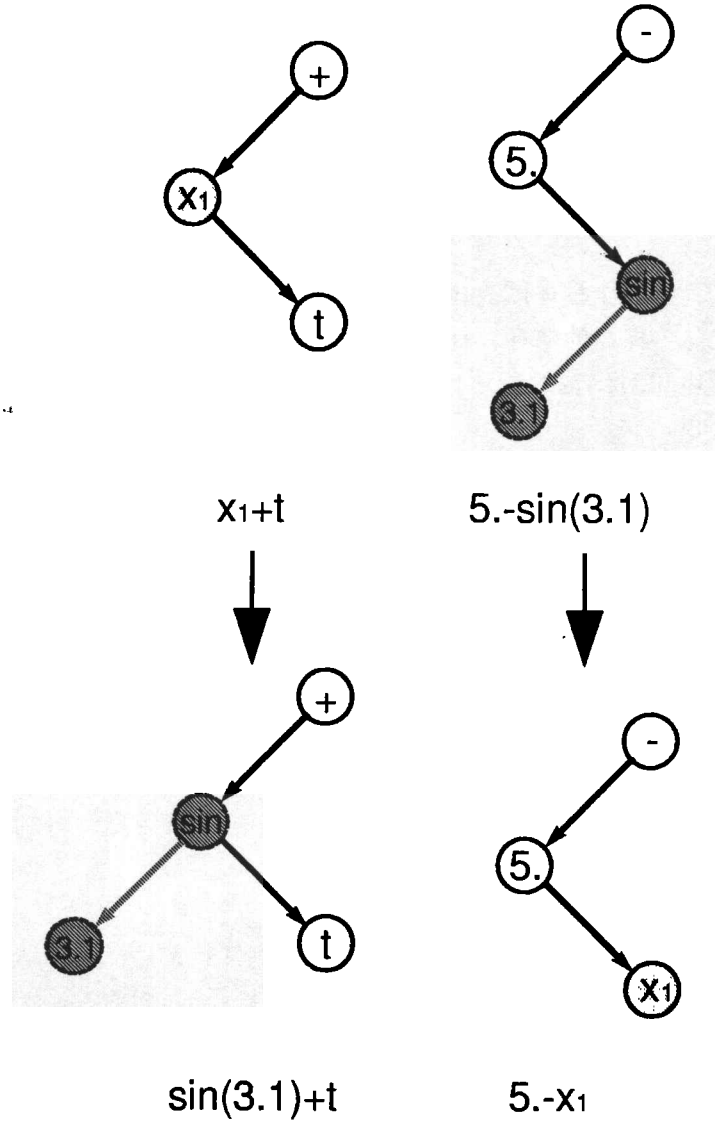

图 6.5: 种群演化的交叉算子。先随机选择两棵树上各一个结点“x_1”和“sin”，然后寻找这些结点的左子树“x_1”和“sin(3.1)”，分别交换这两棵左子树，右子女保持不变。

```
void reset_position(int position)
{ t=position;}
void push(Tdatatype X);
void pop();
void top(Tdatatype *X);
```

```
    int sempty();
    Tdatatype ptop();
private:
    Tdatatype s[m0+1];//s[1-m0]
    int t;
    int StackDepth;
};
```

可以看出整个程序就是围绕如何处理复杂优化对象(这里是函数，在计算机中就是一棵树的储存，计算等)的结构，这也是遗传程序设计要解决的问题。所以说对结构的表达以及相对应的演化算子的操作方式是遗传程序设计的核心。

6.3 系数优化

我们在遗传程序中嵌入了第5章介绍的自适应拟蒙特卡罗全局优化算法(AQMC)用于系数优化(coefficient optimization)。即使函数有好的结构但系数不好，这样的函数在演化过程中也会被丢弃。比如图6.6中的函数"$\sin(0.5*t)+3.*x_1$"，只要把函数中的变量x_1的系数由3.改成5.就能很好地拟合系统的数据。但若不进行参数优化，原来的函数就可能因为对历史数据拟合得不好而在演化过程中就被抛弃了。所以当我们得到新的一代种群后便要在向下一代的演化前进行系数优化。

系数优化过程是首先周游树(函数)，找到常数结点，记录结点的地址，然后在一定范围内改变这些系数的值并计算新的适应度函数值。我们利用AQMC算法来做系数优化。适应度(fitness)函数与前节遗传程序设计中的适应度函数相同。

6.4 一个实际应用

我们用遗传程序设计和自适应拟蒙特卡罗全局优化算法相结合预测了杭州市地区2000年全社会用电量。我们手头有1990年至1999年的用电量历史数据，然后用此文的方法预测2000年的用电量。当然，为了测试方法的有效性，2000年的数据是已知的。表6.1比较了实际数据和一个运行结果的预测数据，相对误差是4.5%(实际需求是误差小于10%)。我们运行的结果误差一般都在6%左右，且所用的时间相当短。在与若干其它预测方法如神经网络等方法的比较中，此方法预测精度是最优秀的。其它方法对于突然的改变无法预测，如从1999年到2000年用电量有了比较大的增长，而之前的

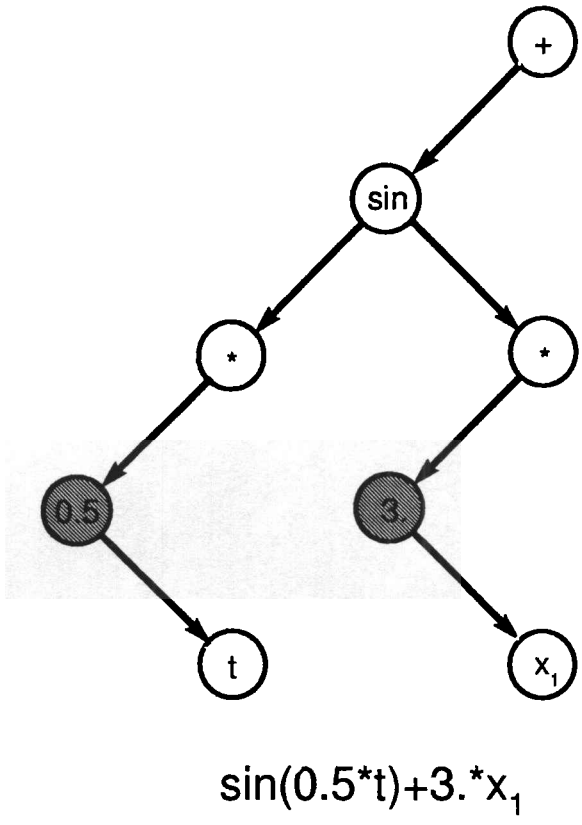

图 6.6: 系数优化：周游树(函数)，找到常数结点并记录结点的地址，然后利用AQMC算法优化这些系数

表 6.1: 杭州市地区2000年全社会用电量预测，相对误差4.5%

t (year)	x_1 (10^7 wh)	x_1' (10^7 wh)
1990	485278.06	
1991	535014.80	548806.96
1992	604257.63	605577.47
1993	680473.09	684688.80
1994	784010.28	771858.12
1995	851131.16	890411.27
1996	930382.96	967341.10
1997	992755.26	1058242.41
1998	1048329.91	1129830.92
1999	1168747.17	1193650.49
2000	1394604.41	1332031.53

增长量比较平缓，用神经网络方法根据历史数据就无法很好地预测2000年的用电量。我们的程序能预测到这种变化，这得归功于常微分方程能很好地刻划复杂变化的系统。而遗传程序设计是优化常微分方程的有力工具，尤其是对多变量的情形，比传统的灰色理论等有很大的优势。

第七章 光在组织中传播的蒙特卡罗模拟及其逆问题

光的传播的方法各种各样，蒙特卡罗方法以其容易实现，容易编程，在处理宏观组织上的优势而得到广泛应用。相对于有限时域差分方法(finite difference time domain(FDTD))等精细方法，蒙特卡罗模拟方法花时较少，更适合作为求逆问题的正向方法。光传播的蒙特卡罗模拟为光在混沌组织中的传播提供了一种灵活且严格的解法。在这一章我们先介绍光在组织中传播的蒙特卡罗模拟方法，然后运用自适应拟蒙特卡罗全局优化方法解光的传播的逆问题。

7.1 引言

光与生物组织相互作用的知识对于设计新的诊断方法和借助镭射的医疗方法是非常重要的。对光在组织中的传播研究得越透彻，设计的方法就会越精细。

蒙特卡罗模拟是建立在输运方程基础上的概率方法。光在组织中传播时，在不同的介质里会发生不同程度的吸收和散射。我们通常用折射率(refraction index)n，吸收系数(absorption coefficient)μ_a，散射系数(scattering coefficient)μ_s和各向异性因子g来描述介质的的光学性质。吸收系数和散射系数分别用来表示每单位光程光子被吸收和散射的概率。吸收系数和散射系数之和$\mu_a + \mu_s$的倒数$1/(\mu_a + \mu_s)$可以解释成光子与介质的相互作用的平均自由程，量$\mu_t = \mu_a + \mu_s$叫做总衰减系数(attenuation coefficient)。而g因子(g-factor) 定义成散射角度的余弦值的平均值。上述这些光学参数在蒙特卡罗模拟中作为输入参数，从而决定光子在介质中的运动。

在这一章讨论的光的传播有一些基本假定。首先假定光分布是稳态的，由此变化和发射的时间短于1纳秒(十亿分之一秒)的光学性质都忽略

不计。其次，所有介质的光学性质都是均匀的。对介质有均匀的折射率的假定可以保证光在组织中被散射或吸收前以直线传播。第三，组织的几何形状可以近似成无限长有一定厚度的平行平板，这种假定要求入射光的宽度要比组织的厚度小。同时假定边界是光滑的，镜面反射遵循Fresnel法则(Fresnel's law)。对这种平板模型的模拟方法可以进一步用到分层组织中或推广到无限厚度的情形。最后一个假定是不考虑光的极化现象。

7.2 蒙特卡罗模拟

蒙特卡罗方法跟踪随机走走的光子包的轨迹，散射与吸收事件由散射系数μ_s与吸收系数μ_a及相函数(phase function)$p(\mathrm{s},\mathrm{s}')$来确定概率。模拟的关键是确定光在两个散射事件发生间隔内走过的平均自由程以及散射角度，另外还要处理内部(边界)反射。

Prahl在文[50]中简要描述了蒙特卡罗模拟的主要公式，并给出了模拟过程的流程图(图7.1)。

光子的吸收和散射由μ_a和μ_s决定，光子在任何与组织作用的位置被吸收的概率是$\mu_a/(\mu_a+\mu_s)$。除非μ_a非常小，光子在若干散射事件后存活的概率很小。这就意味着要跟踪非常多的光子才能使在离开光源很远的位置的模拟达到可以接受的精度。为了用少得多的光子数能提高模拟精度，利用了一种方差缩减技术(variance reduction technology)。原始的方法是在吸收事件中终止光子的传播(光子就死亡了)，用方差缩减技术时用光子包(photon packet)取代光子，光子包的权重初始设为1，在吸收事件中比例为μ_a/μ_t的光子包被吸收，剩下的部分被散射。光子包一直被跟踪直到在轮盘赌(roulette method)时它被终止，即当光子包的权重小于某个阈值(比如0.001)时，光子有$\frac{1}{m}$概率在轮盘赌中存活。如果光子存活，它的权重增加为原来的m倍以保证所有入射光的光能守恒。

光子包在发生吸收和散射前会传播一段距离。现在我们来讨论如何用第2章中的连续累积分布函数求逆(CDF)方法产生步长(step size)$\triangle s$。

根据衰减系数μ_t的定义，光子在传播距离s和$s+\mathrm{d}s$之间每单位路程与介质发生相互作用的概率是$\mu_t\mathrm{d}s$，表示成概率式子是：

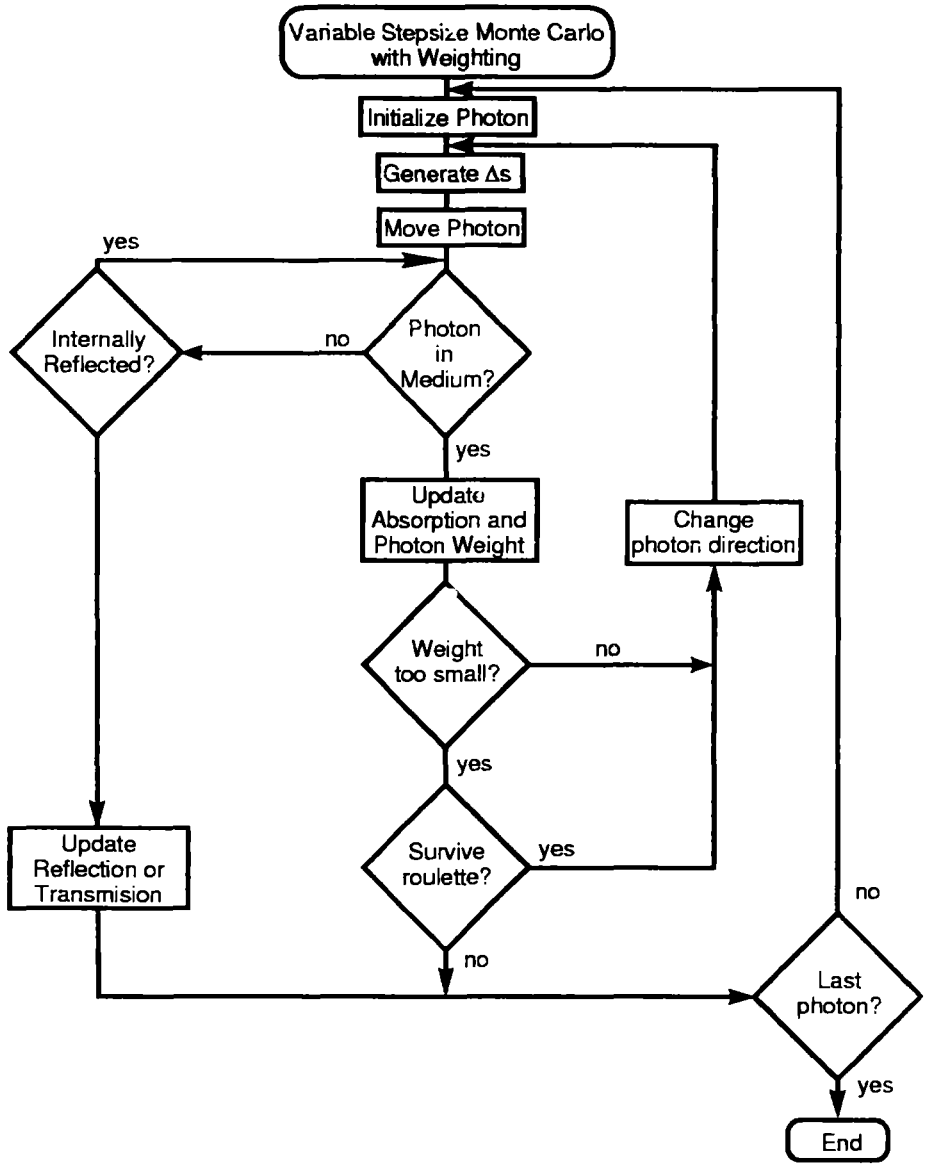

图 7.1: 光传播蒙特卡罗模拟的流程图。光子一旦入射，传播$\triangle s$的路程后发生散射，吸收，继续传播，内部传播或透射(出组织)。光子将一直运动直到光子从组织中跑出或被组织吸收。如果光子从组织中跑出，则记录光子反射或透射的位置。如果光子被吸收，则记录光子被吸收的位置。上述过程一直重复直到一定量的光子全部传播完毕。如果传播的光子数趋向无穷，则记录的反射量、透射量和吸收量的分布便接近真实值。

$$\mu_t \mathrm{d}s = \frac{-\mathrm{d}P(S \geq s)}{P(S \geq s)} \tag{7.1}$$

将方程两边积分，得

$$\mu_t s = -\ln P(S \geq s)$$

所以

$$P(s) = 1 - \exp(-\mu_t s)$$

利用式(2.7)，我们有

$$s = \frac{-\ln(1-\xi)}{\mu_t}$$

这里ξ是在区间$[0,1]$上均匀分布的随机数，所以此式与下式等效

$$s = \frac{-\ln \xi}{\mu_t} \tag{7.2}$$

我们用六个参数表示光子包的位置，三个参数表示三维笛卡尔坐标，另个三个参数表示光子包传播方向的余弦值。所以光子从位置(x,y,z)往方向(μ_x,μ_y,μ_z)传播$\triangle s$长距离后新的位置(x',y',z')计算如下：

$$\begin{aligned} x' &= x + \mu_x \triangle s \\ y' &= y + \mu_y \triangle s \\ z' &= z + \mu_z \triangle s \end{aligned} \tag{7.3}$$

如果光子包传播透过边界到达有不同折射率的介质时有可能发生反射。我们假定组织的几何形状可以近似成x和y方向平行，z方向厚度为τ的平板。光子内部反射的概率由Fresnel反射系数(reflection coefficient)$R(\theta_i)$决定。

$$R(\theta_i) = \frac{1}{2}\left[\frac{\sin^2(\theta_i - \theta_t)}{\sin^2(\theta_i + \theta_t)} + \frac{\tan^2(\theta_i - \theta_t)}{\tan^2(\theta_i + \theta_t)}\right] \tag{7.4}$$

这里$\theta_i = \cos^{-1}\mu_z$是入射角，$\theta_t$是透射角，它们的关系由Shell法则决定。

$$n_i \sin\theta_i = n_t \sin\theta_t \tag{7.5}$$

其中n_i和n_t分别是入射介质和透射介质的折射率。我们用均匀分布在0和1之间的随机数ξ来决定光子是反射还是透射。如果$\xi < R(\theta_i)$，则光子内部反射，否则光子跑出组织。如果光子是从组织顶部跑出，则记为后散射光；如果光子从组织底部跑出，则记为透射光。如果光子在位置(x'', y'', z'')是

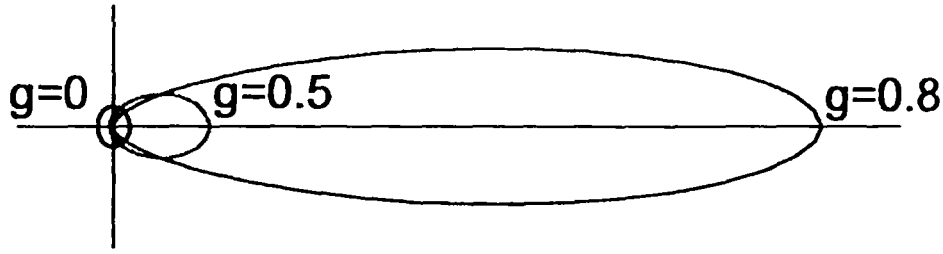

图 7.2: 对不同的g因子Henyey-Greenstein相函数的形状

内部反射的，则反射光的位置只需改变z轴的值。

$$(x'', y'', z'') = \begin{cases} (x, y, -z) & if \ \ z < 0, \\ (x, y, 2\tau - z) & if \ \ z > \tau. \end{cases} \tag{7.6}$$

而光子新的传播方向(μ_x', μ_y', μ_z')为

$$(\mu_x', \mu_y', \mu_z') = (\mu_x, \mu_y, -\mu_z) \tag{7.7}$$

μ_x和μ_y的值都保持不变。

如果光子包传播后仍在组织中或在边界反射回组织内部，则光子包的一部分要被吸收。剩余的光子包如果权重小于某一个值，则做轮盘赌。如果光子包的权重较大不需做轮盘赌或在轮盘赌中幸存下来，则它将被散射。散射的偏转角(deflection angle)θ可以根据相函数产生。相函数(phase function)$p(\mathrm{s}, \mathrm{s}')$ 描述光子从方向s'散射到方向s的概率，为了强调它是角度相关的，相函数有时被写成$p(cos\theta)$的形式。最常用的用于混沌介质的相函数叫做Henyey-Greenstein相函数(phase function)[51]:

$$p(\cos\theta) = \frac{(1-g^2)}{2(1+g^2-2g\cos\theta)^{3/2}} \tag{7.8}$$

这里g叫作散射各向异性因子。对不同的g因子Henyey-Greenstein相函数的形状画在图7.2中。 将式(7.8)代入式(2.7)，求解$\cos\theta$的值得:

$$\cos\theta = \frac{1}{2g}[1+g^2-(\frac{1-g^2}{1-g+2g\xi})^2] \tag{7.9}$$

而对于各向同性的介质，$p(\cos\theta)=\frac{1}{2}$，所以

$$\cos\theta = 2\xi - 1 \tag{7.10}$$

散射的方位角(azimuthal scattering angle)是均匀分布在区间$0<\phi<2\pi$，所以有

$$\phi = 2\pi\xi \tag{7.11}$$

如果光子从传播的方向(μ_x,μ_y,μ_z)以角度$(\theta,\ \phi)$散射，则新的传播方向(μ'_x,μ'_y,μ'_z)可以计算如下：

$$\begin{aligned}
\mu'_x &= \frac{\sin\theta}{\sqrt{1-\mu_z^2}}(\mu_x\mu_z\cos\phi-\mu_y\sin\phi)+\mu_x\cos\theta \\
\mu'_y &= \frac{\sin\theta}{\sqrt{1-\mu_z^2}}(\mu_y\mu_z\cos\phi+\mu_x\sin\phi)+\mu_y\cos\theta \\
\mu'_z &= -\sin\theta\cos\phi\sqrt{1-\mu_z^2}+\mu_z\cos\theta
\end{aligned} \tag{7.12}$$

如果角度太接近直角(比如$|\mu_z|>0.99999$)，必须用如下公式修正传播方向。

$$\begin{aligned}
\mu'_x &= \sin\theta\cos\phi \\
\mu'_y &= \sin\theta\sin\phi \\
\mu'_z &= \frac{\mu_z}{|\mu_z|}\cos\phi
\end{aligned} \tag{7.13}$$

光子包将一直传播直到跑出组织或全部被组织吸收（在轮盘赌中死亡），光子包每次被吸收时，记录光子被吸收的位置和吸收的量。如果光子包从组织中跑出，则记录光子包反射或透射的位置。以上所述的就是一个光子包传播的完整过程。这样的过程要多次重复直到一定量的光子全部传播完毕。如果传播的光子包数趋向无穷，则记录的反射量、透射量和吸收量的分布便接近真实值。

7.3 逆问题

光学医疗仪器的设计是通过测得光在组织中的传播来反推组织的光学性质比如折射率(n)，吸收系数(μ_a)，散射系数(μ_s)和散射各向异性因子(g)，即需要解光传播的逆问题。我们用多层介质蒙特卡罗模拟程序(MCML) [52]解正向问题，用第5章的自适应拟蒙特卡罗全局优化算法求逆问题。

我们在引言中提到平板模型可以进一步用到分层组织中，MCML程序就是计算多层组织的蒙特卡罗模拟程序。除了前节描述的过程，MCML考

虑了多层组织之间的内部反射和透射问题。为简便起见，我们只计算介质的层数为1的组织，组织上面和下面的介质均为玻璃。我们定义适应度函数为

$$fit = \sum_{n=0}^{num-1} (Tr[n] - Tr_0[n])^2 \tag{7.14}$$

式中Tr_0为实际测得的透射分布量(也可以取反射分布量，但透射分布量在实验中易测得)。在没有实验数据的情况下，可以用蒙特卡罗模拟的值代替实际值。这里就是取光学参数为真实的值时的蒙特卡罗模拟的透射分布量，Tr是光学参数在一定范围内每次模拟的透射分布量。我们的目标是要寻找能最小化适应度函数fit的光学参数，即寻找使fit达到最小值0的光学参数。每一次运行，取随机分布的光学参数，将它们作为MCML的输入参数，经过MCML模拟，输出透射分布量，计算fit值。

我们在用自适应拟蒙特卡罗全局优化(AQMC)算法时在平衡全局优化和局部优化方面做了一些探索。对于这个实际物理问题，其优化目标函数的值是急剧变化的，且存在大量局部极小值，所以在搜索过程中需要更多的引进新个体。第5章中介绍的AQMC算法是着重局部搜索的，虽然根据演化度(evolution degree) 来引进新个体(在函数空间产生新的随机数)，但演化度反应的是局部搜索能力的饱和度，只有当局部搜索很难再找到更优的极值时引进新个体的概率才大。而且当产生新的一批拟随机数点时，种群得到了更新，重新置演化度为0，这时即使我们在引进新个体的搜索过程中找到了一个比原来的全局极小值近似值小得多的近似值，也要转到局部搜索中去，而不是继续引进新个体。所以根据演化度来引进新个体的算法大多数时间是在执行局部搜索，对于需要更多引进新个体的问题找到全局极值的速度就慢了。

我们应该有一个刻划局部搜索能力和引进新个体的搜索能力的度量，这种度量更注重个体的改进。如果在局部搜索中找到了更好的极值点，就提高局部搜索的概率；如果在引进新个体的搜索中找到了好得多的极值点，就提高引进新个体的概率。我们使用性能/代价比(performance/cost ratio)来刻划搜索能力。

定义 7.3.1 (性能/代价比) *记$max_initial$为初始种群中目标函数最大值，若在经过计算N_c个函数值后找到新的最大值$max_current$，则定义性能/代价比为:*

$$eff = (max\_current - max\_initial)/N_c$$

表 7.1: 用AQMC算法解光在组织中传播的逆问题的一个数值结果

	n	μ_a	μ_s	g
origin	1.375	1	100	0.9
approximation	1.375	1.13	101.	0.9

于是我们可以分别计算局部搜索的性能/代价比$lseff$和引进新个体的搜索的性能/代价比$nieff$，用这两个比率来决定产生新个体的概率$newp$。可以对两个搜索赋予权重lsw和niw，用来决定局部搜索和引进新个体各自占的比重。于是

```
if( lseff <1.0E−10 && nieff< 1.0E−10)
   newp=niw;
else
   newp=niw*nieff/(lsw*lseff+niw*nieff);
```

然后按概率运行局部搜索或产生新个体，即产生随机数rnd，如果$rnd < newp$，则产生一批新个体。

```
do{
   rnd=(double)(rand())/RAND_MAX;
   if(rnd<newp)
    { /*generate new individual*/
      ...
    }
   else
     { /*local search*/
       ...
     }
}while(fglobalmax[S]<−1.0E−5);
```

整个优化程序见附录。

我们做的是单层四个参数联合的优化。对平板的厚度为1cm，光学参数为$n = 1.375$，$\mu_a = 1$，$\mu_s = 100$，$g = 0.9$的组织，设置四个光学参数的搜索范围为$n \in [1.0, 2.0]$，$\mu_a \in [0.01, 3]$(为了避免程序永远循环，吸收系数的值不能设为0)，$\mu_s \in [2, 200]$和$g \in [0.0, 1.2]$ (实际上g的物理值是[-1, 1]，这里取了正值自变量区域，同时为了考虑局部搜索的映射中自变量会溢出边界，区域的右端值放大了)，我们的AQMC算法运行了263次MCML后找到了较好的光学参数近似值，列在表7.1中。

这一方法同样可以运用于多层组织，但效果不如单层的效果好。比较第5章中的数值例子的结果，可见对实际的物理问题，优化方法还得研究问题本身的性质。我们希望今后能结合实验数据做一些工作，继续探讨蒙特卡罗搜索的局部和全局的平衡，根据具体问题提出有效的度量。

第八章 附录：程序代码

8.1 Sobol'序列发生器的C语言代码

此程序从[26]改编，能产生最大维数为160维的Sobol'序列。输入文件sobol_para.txt中放的是数组mdeg[](简单多项式的次数)和ip[](多项式的系数组成的二进制的值)。

```
#include "math.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAXBIT 30
#define MAXDIM 160
#define S 2

/*When n is negative, internally initializes a set of
MAXBIT direction numbers for each of MAXDIM different
Sobol' sequences. When n is positive (but <= MAXDIM),
returns as the vector x[1..n] the next values from n
of these sequences.
(n must not be changed between initializations.)*/

int IMIN(int a,int b)
{
    return a<b ? a:b;
}

void sobseq(int *n, double x[])
{ FILE *fr1;
  int j,k,l;
  unsigned long i,im,ipp;
  static double fac;
  static unsigned long in,ix[MAXDIM+1],*iu[MAXBIT+1];
```

```
static unsigned long mdeg[MAXDIM+1];
static unsigned long ip[MAXDIM+1];
static unsigned long iv[MAXDIM*MAXBIT+1];
long temp;

if(*n < 0)
{ //read mdeg[] and ip[]
 if((fr1=fopen("sobol_para.txt","r"))==NULL)
 {
     printf("\nFile_not_found!");
     exit(0);
 }
 for(i=1;i<=MAXDIM;i++)
     fscanf(fr1,"%d",&mdeg[i]);//read mdeg[]
 for(i=1;i<=MAXDIM;i++)
     fscanf(fr1,"%d",&ip[i]);//read ip[]
 fclose(fr1);
//set the values of iv[][//
srand((unsigned)time(NULL));//give the rand seed
for(i=1;i<=mdeg[MAXDIM];i++)
{
 for(j=1;j<=MAXDIM;j++)
 {temp=2*(int((1L<<(i-1))*float(rand())/RAND_MAX)+1)-1;
  iv[(i-1)*MAXDIM+j]=temp;
 }
}
/* Initialize ; don't return a vector.*/
for (k=1;k<=MAXDIM;k++)
    ix[k]=0;
in=0;
if (iv[1] != 1)
    return;
fac=1.0/(1L << MAXBIT);
for (j=1,k=0;j<=MAXBIT;j++,k+=MAXDIM)
    iu[j] = &iv[k];
/*To allow both 1D and 2D addressing.*/
for (k=1;k<=MAXDIM;k++)
{
    for (j=1;j<=mdeg[k];j++)
        iu[j][k] <<= (MAXBIT-j);
    /*Stored values only require normalization.*/
    for (j=mdeg[k]+1;j<=MAXBIT;j++)
   {
```

```
            /*Use the recurrence to get other values.*/
            ipp=ip[k];
            i=iu[j-mdeg[k]][k];
            i ^= (i >> mdeg[k]);
            for (l=mdeg[k]-1;l>=1;l--)
          {
                if (ipp & 1)
                    i ^= iu[j-l][k];
                ipp >>= 1;
          }
            iu[j][k]=i;
        }
    }
  }
  else
  {
   /*Calculate the next vector in the sequence.*/
   im=in++;
   for (j=1;j<=MAXBIT;j++)
   {
        /*Find the rightmost zero bit.*/
        if (!(im & 1))
            break;
        im >>= 1;
   }
    if (j > MAXBIT)
        printf("\nMAXBIT_too_small_in_sobseq\n");
   im=(j-1)*MAXDIM;
   for (k=1;k<=IMIN(*n,MAXDIM);k++)
   {
    /*XOR the appropriate direction number into each
    component of the vector and convert to a floating
    number.*/
    ix[k] ^= iv[im+k];
    x[k]=ix[k]*fac;
   }
  }

}


void main( )
{FILE *fw1;
static double xsob[S+1];
```

```
int i,j;
int ini,run;
ini=-1;
sobseq(&ini,xsob);
if( (fw1 = fopen( "sobol_points", "w" )) == NULL )
 { printf( "The file 'sobol_points' was not opened\n" );
   exit(0);
 }
for(i=1;i<=1024;i++)
{run=S;
sobseq(&run,xsob);
for(j=1;j<=S;j++)
    fprintf(fw1,"%f\t",xsob[j]);
fprintf(fw1,"\n");
}//end of i
fclose(fw1);
}

// begin of input file sobol_para.txt
1
2
3 3
4 4
 5 5 5 5 5 5
 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8
8 8 8 8 8 8
 9 9 9 9 9 9 9 9 9 9
 9 9 9 9 9 9 9 9 9 9
 9 9 9 9 9 9 9 9 9 9
 9 9 9 9 9 9 9 9 9 9
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
0
1
1 2
1 4
 2 4 7 11 13 14
```

```
 1  13  16  19  22  25
  1  4  7  8  14  19  21  28  31  32  37  41  42  50  55
56  59  62
  14  21  22  38  47  49  50  52  56  67  70  84  97  103
115  122
  8  13  16  22  25  44  47  52  55  59  62  67  74  81
 82  87  91  94  103  104  109  122 124  137  138  143
 145  152  157  167  173  176  181  182  185  191  194
 199  218  220  227  229  230  234  236  241  244  253
  4  13  19  22  50  55  64  69  98  107  115  121  127
 134  140  145  152  158  161  171  181  194  199  203
 208  227  242  251  253  265  266  274  283  289  295
 301  316  319  324  346  352  361  367  382  395  398
 400  412  419  422  426  428  433  446  454  457  472
493  505  508
// end of input file
```

8.2 Halton序列发生器的C语言代码

此程序是从已有的根据文[53]编写的Fortran程序转成C代码的。

```
//============the program to generate Halton sequence====//
#include "math.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAXBIT 64
#define MAXDIM 40
#define MAXNUM 1000
#define S 2  /*dimension*/

/*input:n, if the first time to run the subroutine, *n=-S;*/
void halseq(int *n, double x[])
{   int j,s;
    static double prime[MAXDIM+1]={0.0,2.0,3.0,5.0,7.0,11.0,13.0,17.0,19.0,23.0,\
                    29.0,31.0,37.0,41.0,43.0,47.0,53.0,59.0,61.0,67.0,71.0,73.0,\
                      79.0,83.0,89.0,97.0,101.0,103.0,107.0,109.0,113.0,127.0,\
                        131.0,137.0,139.0,149.0,151.0,157.0,163.0,167.0,173.0};

    static double xhal[MAXDIM+1],E,Delta,Tiny=1.0;
    static unsigned char Flag[2]={0,0};
    double T,F,G,H;
```

```
if(*n < 0)
{ /*FIRST CHECKS WHETHER THE USER-SUPPLIED DIMENSION "
    DIMEN"
    OF THE QUASIRANDOM VECTORS IS ACCEPTABLE
   ( STRICTLY BETWEEN 0 AND 41):
    IF SO, FLAG(1)=.TRUE.*/
    s=-*n;/*the dimension*/
    Tiny=1L>>MAXBIT;
    if((s>=1)&&(s<=40))
        Flag[0]=1;
    else
   {
        printf("The dimension must between 1 and 40!\n");
        exit(0);
   }
    /* COMPUTE AND CHECK TOLERANCE*/
    E = 0.9* (1.0/ ((double)MAXNUM*prime[s])-10.0*Tiny);
    Delta = 100.0*Tiny*(double)(MAXNUM+1)*log10((double)MAXNUM);
    if(Delta <= (0.09* (E-10.0*Tiny)))
        Flag[1]=1;
    else
   {
        printf("The dimension must between 1 and 40!\n");
        exit(0);
   }
    /*NOW COMPUTE FIRST VECTOR*/
    for(j=1;j<=s;j++)
   {
        prime[j]=1.0/prime[j];
        xhal[j]=prime[j];
        x[j]=xhal[j];
   }
}

else
{
    s=*n;
    for(j=1;j<=s;j++)
   {
        T = prime[j];
        F = 1.0 - xhal[j];
        G = 1.0;
        H = T;
        while ((F-H)<E)
```

```
            {
                G = H;
                H = H*T;
            }
              xhal[j] = G + H - F;
              x[j]=xhal[j];
        } /*end of for(j=1;j<=s;j++)*/
    } /*end of else*/
}

void main( )
{
    FILE *result;
    double xhal[S+1];
    int i,j;
    int ini,run;
    int num_random=120;
    if( ( result  = fopen( "halton.dat", "w" )) == NULL )
    {    printf( "The_file_'halton.dat'_was_not_opened\n" );
         exit(0);
    }
     fprintf(result,"Halton_sequence .....\ n");
    /* initialize  and generate the  first  random number*/
     ini=-S;
     halseq(&ini,xhal);
     for(j=1;j<=S;j++)
          fprintf(result,"%f\t",xhal[j]);
     fprintf(result,"\n");
     fclose(result);
     for(i=2;i<=num_random;i++)
     { if( ( result  = fopen( "halton.dat", "a" )) == NULL )
         {    printf( "The_file_'halton.dat'_was_not_opened\n" );
              exit(0);
         }
          run=S;
          halseq(&run,xhal);
          for(j=1;j<=S;j++)
            fprintf(result,"%f\t",xhal[j]);
          fprintf(result,"\n");
          fclose(result);
     } //end of i
}
```

8.3 蒙特卡罗(MC)，对偶变数蒙特卡罗(AMC)，精细对偶变数蒙特卡罗(AMC)估计的并行程序

我们写的是MPI并行程序，可以在机群上运行。此节的代码除了子程序ran2()来自[26]，其余代码全为作者所写，作者保留版权。

```
/*
 Copyright 2003 by Guiyuan Lei
 All rights reserved. All codes except the ran2() in this section can be used in the
       case that user cite this thesis.
 Monte Carlo integration in parallel programming(MPI)
 all the processes use a same sequence,
 passing the random seeds and serial of random number(n) through the processes.
 the pseudorandom sequence is generated by rans()
 from book "Numerical recipes in C: The art of scientific computing"
 seeds:
 static long idum2=123456789;
 static long iy=0;
 static long iv[NTAB];
*/
#include "math.h"
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAXBIT 64
#define MAXDIM 40
#define MAXNUM 1000000
#define S 15  //dimension=15
#define Pi 2*arcsin(1.)
/*length of sub-sequence generated by each process*/
#define Max_Len 32768

#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
```

```
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)
/*Uniformally distributed random sequence generator
  p282 Chapter 7. Random Numbers in "Numerical Recipes in C"
*/
static long idum2=123456789;
static long iy=0;
static long iv[NTAB];

/*Long period (> 2 * 10^18 ) random number generator of L'Ecuyer with Bays-
        Durham shuffle
  and added safeguards. Returns a uniform random deviate between 0.0 and 1.0 (
        exclusive of
  the endpoint values). Call with idum a negative integer to initialize ; thereafter .
        do not alter
  idum between successive deviates in a sequence. RNMX should approximate the
        largest oating
  value that is less than 1.
*/
float ran2(long *idum)
{
    int j;
    long k;
    float temp;
    if (*idum <= 0)
    { /* Initialize */
        /*Be sure to prevent idum = 0*/
        if (-(*idum) < 1)
          * idum=1;
        else
          * idum = -(*idum);
        idum2=(*idum);
        for (j=NTAB+7;j>=0;j--)
        { /*Load the shuffle table (after 8 warm-ups)*/
            k=(*idum)/IQ1;
          * idum=IA1*(*idum-k*IQ1)-k*IR1;
            if (*idum < 0)
              * idum += IM1;
            if (j < NTAB)
                iv[j] = *idum;
        }
        iy=iv[0];
```

```
    }
    /*Start here when not initializing .*/
    k=(*idum)/IQ1;
    /* Compute idum=(IA1*idum) % IM1 without over ows by Schrage's method.*/
    *idum=IA1*(*idum-k*IQ1)-k*IR1;
    if (*idum < 0)
      * idum += IM1;
    k=idum2/IQ2;
    /*Compute idum2=(IA2*idum) % IM2 likewise.*/
    idum2=IA2*(idum2-k*IQ2)-k*IR2;
    if (idum2 < 0)
        idum2 += IM2;
    j=iy/NDIV;/*Will be in the range 0..NTAB-1.*/
    iy=iv[j]-idum2;/* Here idum is shuffled, idum and idum2 are combined to generate
        output.*/
    iv[j] = *idum;
    if (iy < 1)
        iy += IMM1;
    if ((temp=AM*iy) > RNMX)
        return RNMX; /*Because users don't expect endpoint values.*/
    else
        return temp;
}

/*the integrand from:
  Math. Comput. Modelling Vol. 23, No, 8/9, pp. 87-96. 1996
  p92 Example2,Figure2,5
*/
double f(double x[])
{/*use the first S dimension of x[], just 1-S elements*/
    double f;
    int i;
    double sum_x=0.;
    for(i=1;i<=S;i++)
        sum_x+=x[i]/i;
    f=exp(sum_x);
    return f;
}
/*Calculating function value and sum the value usint CMC.AMC.FAMC method*/
void MC(long temp_serial, int n, double x[],double interval,double sum[])
{
    int l;
    double f_x;
    double x_k[S+1];
```

```
    double c_k[S+1];
    f_x=f(x);/*function value of x*/
    sum[1]+=f_x;/*Crude Monte Carlo estimate*/
    sum[2]+=f_x;
    for(l=1;l<=S;l++)
        c_k[l]=1.0- x[l]; /*the antithetic variables of x*/
    sum[2]+=f(c_k);/*Antithetic Monte Carlo estimate*/
    for(l=1;l<=S;l++)
    { /*The domain of function is devided into N=n^S subcube D_k
                c_k[l]: leftest border of subcube k*/
        c_k[l]=temp_serial%n;
        temp_serial=temp_serial/n;
        x_k[l]=(c_k[l]+x[l])*interval;
    }
    sum[3]+=f(x_k);
    for(l=1;l<=S;l++)
    { /*Here c_k[l] is centre of subcube k
            interval is the interval of subcube*/
        c_k[l]=(0.5+c_k[l])*interval;
        x_k[l]=2*c_k[l]- x_k[l]; /*the antithetic variables*/
    }
    sum[3]+=f(x_k);/*Fine Antithetic Monte Carlo estimate*/
}
/*end of calculating function value usint CMC,AMC,FAMC method*/
/*Linear fit of the data (X,T), least square error method*/
void Fit_linear(long X[],double T[],int count,double A[])
{
    int i;
    double Xi;
    double Ti;
    int sum_true=1;/*if sum the data*/
    double sum_Xi=0.0;
    double sum_Xi_square=0.0;
    double sum_Ti=0.;
    double sum_Xi_Ti=0.;
    double a;/*slope*/
    double b;/*intercept*/
    for(i=1;i<=count;i++)
    {
        sum_true=1;/*to sum the data*/
        Ti=T[i];
        if(Ti<1.0E-20)
            sum_true=0;
        /*to calculate slope*/
```

```
            if(sum_true)
            {
                Xi=log10(X[i]);
                sum_Xi+=Xi;/*sum x*/
                sum_Xi_square+=Xi*Xi;/*sum x^2*/
                Ti=log10(T[i]);
                sum_Ti+=Ti;/*sum T*/
                sum_Xi_Ti+=Xi*Ti;/*sum x*T*/
            } /*end of if(sum_true)*/
            else
                count--;
        } /*end of i(step)*/

        a=(count*sum_Xi_Ti-sum_Xi*sum_Ti)/(count*sum_Xi_square-sum_Xi*sum_Xi);
        b=(sum_Ti*sum_Xi_square-sum_Xi*sum_Xi_Ti)/(count*sum_Xi_square-sum_Xi*
            sum_Xi);
        A[0]=a;
        A[1]=b;
    }
    /*Empirical standard deviate(sd) error*/
    double sd_error(double s_run[],int runs)
    {
        int l;
        double temp;
        double proximate_int_average=0.;
        for(l=0;l<runs;l++)
            proximate_int_average+=s_run[l];
        proximate_int_average/=runs;
        temp=0.0;
        for(l=0;l<runs;l++)
            temp+=(s_run[l]-proximate_int_average)*(s_run[l]-proximate_int_average);
        temp=sqrt(temp/(runs-1));
        return temp;
    }
    /*Empirical root mean square error(rmse)*/
    double rmse(double s_run[],double exact_int,int runs)
    {
        int l;
        double temp=0.;
        for(l=0;l<runs;l++)
            temp+=(s_run[l]-exact_int)*(s_run[l]-exact_int);
        temp=sqrt(temp/runs);
        return temp;
    }
```

```
main(int argc,char** argv)
{
    FILE *f_value,*f_rmse, *f_sd;
    char str[20]; /* file name*/
    double s_step_run[4][4][76];
    double variance_step[4];
    int step=3;
    long points_step[4];
    long N;/*number of random point points*/
    int runs=75;/*compute the root mean square error over 75 runs*/
    /*the sum of function value for Crude MC. AMC and FAMC methods*/
    double sum[4],G_sum[4];
    long i,j,k,m;

    int l;/*index for dimension*/
    int n;/*number of sub-interval*/

    double interval;
    double exact_int=5.6102534948577798;/*for s=15*/
    /*exact_int=3.0060133559748561; //for s=4*/
    long ini;

    double A[2];/*the coefficient of linear fit*/
    /*each process calculate sub_seq_len random points every procs_step random points
        */
    long procs_step;
    long size,remainder;/*size. loop size*/
    int sub_seq_len  Max_Len;/*no more than Max_Len*/
    double random[Max_Len][S+1];

    int nid,nid_before,nid_after,noprocs,last_nid;/*for parallel programming*/
    MPI_Status status;
    long seeds[NTAB+3];/*iv[NTAB],idum,iy,ini*/
    MPI_Request req_send_seeds,req_recv_seeds;

    /*call MPI initialization*/
    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&nid);
    MPI_Comm_size(MPI_COMM_WORLD,&noprocs);

    for(j=2;j<=step+1;j++)
    {
        N=1;
```

```
    for(i=0;i<S;i++)
        N*=j;
    points_step[j-1]=N;/*set calculating points for each step*/
}

if(nid==noprocs-1)
{
    for(i=1;i<=3;i++)/*deal with three methods:CMC=1,AMC=2,FAMC=3*/
    { sprintf(str,"f_value-%d.dat",i);
        f_value=fopen(str, "w" );
        for(j=1;j<=step;j++)
            fprintf(f_value,"%ld\t\t",points_step[j]);
        fprintf(f_value,"\n");
        fclose(f_value);
    }
    ini=-S;/*the value of ini will be changed by ran2()*/
    for(m=1;m<=2500;m++)/*ingore the first 2500 random number*/
        ran2(&ini);
    /*envelope the seeds*/
    for(m=0;m<NTAB;m++)
        seeds[m]=iv[m];
    seeds[m]=idum2;
    seeds[m+1]=iy;
    seeds[m+2]=ini;
    nid_after=(nid+1)%noprocs;
    MPI_Isend(seeds,NTAB+3,MPI_LONG,nid_after,10,MPI_COMM_WORLD,&
     req_send_seeds);
}
/*generate next random numbers and perform integration*/
for(i=0;i<runs;i++)
{
    for(j=1;j<=step;j++)
    {
        N=points_step[j];
        n=j+1;
        interval=1./n;
        /*calculate the sum on distributed noprocs computer*/
        for(k=0;k<4;k++)
            sum[k]=0.;
        /*the number of random points each process should calculate*/
        procs_step=sub_seq_len*noprocs;
        remainder=N%sub_seq_len;
        size=N-remainder;
        last_nid=noprocs-1;
```

```
/*==========loop for size==========*/
for(k=nid*sub_seq_len;k<N;k+=procs_step)
{ /*receive the random seeds*/
    if(nid)  /*nid_before=(nid+noprocs-1)%noprocs;*/
        nid_before=nid-1;
    else
        nid_before=last_nid;
    MPI_Irecv(seeds,NTAB+3,MPI_LONG,nid_before,10,
MPI_COMM_WORLD,&req_recv_seeds);
    MPI_Wait(&req_recv_seeds,&status);
    /*unenvelope the seeds*/
    for(m=0;m<NTAB;m++)
        iv[m]=seeds[m];
    idum2=seeds[m];
    iy=seeds[m+1];
    ini=seeds[m+2];

    if(k+sub_seq_len<=size)
   {
        for(m=0;m<sub_seq_len;m++) /*generate sub-sequence*/
         {    for(l=1;l<=S;l++)
                random[m][l]=ran2(&ini);
     }
   }
    else
   {
        for(m=0;m<remainder;m++) //generate remainder points*/
     {
          for(l=1;l<=S;l++)
              random[m][l]=ran2(&ini);
     }
   }
    /*send the seeds
      envelope the seeds*/
    for(m=0;m<NTAB;m++)
        seeds[m]=iv[m];
    seeds[m]=idum2;
    seeds[m+1]=iy;
    seeds[m+2]=ini;

    nid_after=(nid+1)%noprocs;
    /*in general the (noprocs-1)'th process send the seeds to process 0*/
    last_nid=noprocs-1;
    /*the process who deal with the N'th random number will send the
```

```
seeds to process 0
        then process 0 start the first random number of new successive N
random points
      */
      if((k+sub_seq_len==N)||k+remainder==N)
    {
        nid_after =0;
        last_nid =nid;
        /*boadcast so that process 0 know the change of last_nid*/
        MPI_Bcast(&last_nid,1,MPI_INT,nid,MPI_COMM_WORLD);
    }
      MPI_Isend(seeds,NTAB+3,MPI_LONG,nid_after,10,
MPI_COMM_WORLD,&req_send_seeds);
      /* calculate the function value and sum them*/
      if(k+sub_seq_len<=size)
    {
        for(m=0;m<sub_seq_len;m++)
            MC(k+m,n,random[m],interval,sum);
    }
      else
    {
        for(m=0;m<remainder;m++)
            MC(k+m,n,random[m],interval,sum);
    }
  } /*end of k: loop size of sub_seq_len*/

   /*send the data to process 0*/
   MPI_Reduce(sum,G_sum,4,MPI_DOUBLE,MPI_SUM,0,
MPI_COMM_WORLD);
   if(nid==0)
  {
      for(k=1;k<=3;k++)
          s_step_run[k][j][i]=G_sum[k];
  }
} /*end of j: step*/
 if(nid==0)
{
   for(j=1;j<=step;j++)
  {
      N=points_step[j];
      s_step_run [1][j][i]=s_step_run [1][j][i]/N;
      for(k=2;k<=3;k++)
          s_step_run[k][j][i]=s_step_run[k][j][i]/N/2.;
  }
```

```
        for(k=1;k<=3;k++)/*deal with three method:CMC=1,AMC=2,FAMC=3*/
      {  sprintf(str,"f_value-%d.dat",k);
           f_value=fopen(str, "a" );
           for(j=1;j<=step;j++)
              fprintf(f_value,"%.20f\t",s_step_run[k][j][i]);
           fprintf(f_value,"\n");
           fclose(f_value);
      }
   } /*end of if(nid==0)*/
} /*end of i :runs*/

 if(nid==0)
{
    for(k=1;k<=3;k++)
   {
        sprintf(str,"rmse-%d.dat",k);
        f_rmse= fopen(str, "w" );
        for(i=1;i<=step;i++)
      {
           fprintf(f_rmse,"%ld\t",points_step[i]);
           variance_step[i]=rmse(s_step_run[k][i], exact_int,runs);
           fprintf(f_rmse,"%.10f\t",variance_step[i]);
           fprintf(f_rmse,"\n");
      }
        Fit_linear(points_step,variance_step,step,A);
        fprintf(f_rmse,"\nFit_Linear...\n");
        fprintf(f_rmse,"Y=%.10f*X+%.10f\n",A[0],A[1]);
        fclose(f_rmse);
   } /*end of k*/
    for(k=1;k<=3;k++)
   {
        sprintf(str,"sd_error-%d.dat",k);
        f_sd= fopen(str, "w" );
        for(i=1;i<=step;i++)
      {
           fprintf(f_sd,"%ld\t",points_step[i]);
           variance_step[i]=sd_error(s_step_run[k][i],runs);
           fprintf(f_sd,"%.10f\t",variance_step[i]);
           fprintf(f_sd,"\n");
      }

        Fit_linear(points_step,variance_step,step,A);
        fprintf(f_sd,"\nFit_Linear...\n");
        fprintf(f_sd,"Y=%.10f*X+%.10f\n",A[0],A[1]);
```

```
            fclose(f_sd);
        } /*end of k*/
    } /*end of if(nid==0),analysize*/
    MPI_Finalize();/*end of MPI*/
}
```

8.4 AQMC算法解光的传播的逆问题的代码

此节的代码除了涉及的MCML，其余代码全为作者所写，作者保留版权。

此代码以多层组织的蒙特卡罗模拟程序(MCML)为正向问题的方法，用AQMC算法解光的传播的逆问题。根据局部搜索和引进新个体各自的性能/代价比来决定两者发生的概率。

```
/*******************************
Copyright 2003 by Guiyuan Lei
All rights reserved. All codes except the MCML in this section can be used in the case
        that user cite this thesis.
AQMC for inverse problem of light transport(optimize the optical parameters)
We use the Monte Carlo Multi-Layer(MCML) simulation programmer as the forward
        method.

The Monte Carlo random search method for the global maximum
(if to find the global minimum, use the minus of the fitness function).
Do local search or generate new individuals according to their performance/cost ratio.
We use the Sobol' sequence in optimization(so we need sobol.c file in compiling)

You can download MCML programs from the website
http://omlc.ogi.edu/software/mc/index.html

MCML's inputfile is sample.mci, output file is sample.mco
in this program we use sample.mco as the inputfile and do optimization
all the file list:
      mcmlaqmc.c  //this file
      sobol.h
      sobol.c     //sobol generator
      mcml.h
      mcmlgo.c
      mcmlio.c
      mcmlnr.c
*/
```

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include "mcml.h"
#include "sobol.h"

#define N 100
#define S 4

/*Declare before they are used in main()*/
void sobseq(int *n, double x[]);

FILE *GetFile(char *);
short ReadNumRuns(FILE* );
void ReadParm(FILE* , InputStruct * );
void CheckParm(FILE* , InputStruct * );
void InitOutputData(InputStruct, OutStruct *);
void FreeData(InputStruct, OutStruct *);
double Rspecular(LayerStruct * );
void LaunchPhoton(double, LayerStruct *, PhotonStruct *);
void HopDropSpin(InputStruct *,PhotonStruct *,OutStruct *);
void SumScaleResult(InputStruct, OutStruct *);
void WriteResult(InputStruct, OutStruct, char *);

InputStruct in_parm;
InputStruct in_parm_opti;

OutStruct out_parm;
OutStruct out_parm_opti;

/*The pseudorandom number generator*/
unsigned long Y1=3115,Y2=3115;
unsigned long  m1=1L<<31,m2=1L<<30;
unsigned long a1=65539,a2=410092949;
int b1=0,b2=1;
unsigned long  GambleMAX1=1L<<31-1,GambleMAX2=1L<<30-1;

unsigned long Gamblerand1()
{
    Y1=(Y1*a1+b1)%m1;
    return Y1;
}
```

```
unsigned long Gamblerand2()
{
    Y2=(Y2*a2+b2)%m2;
    return Y2;
}
/*end of the pseudorandom number generating*/

double DMAX(double a,double b)
{
    return a>b ? a:b;
}

//==========mcml.c==========//
void DoOneRun(InputStruct *In_Ptr,OutStruct *out_parm)
{
    register long i_photon;
    /* index to photon. register for speed.*/
    /* distribution of photons.*/
    PhotonStruct photon;
    long num_photons = In_Ptr->num_photons, photon_rep=10;
    out_parm->Rsp = Rspecular(In_Ptr->layerspecs);
    i_photon = num_photons;
    do
    {
        LaunchPhoton(out_parm->Rsp, In_Ptr->layerspecs, &photon);
        do
            HopDropSpin(In_Ptr, &photon, out_parm);
        while (!photon.dead);
    }
    while(--i_photon);
}



/*the range of variable of the function*/
double xrange[S][2]={{0.01,3},{2,200},{0.0,1.2},{1.0,2.0}};
/*************************************************************
    Report time and write results.
*************************************************************/
void ReportResult(InputStruct In_Parm, OutStruct Out_Parm)
{
    char time_report[STRLEN];
```

```
    strcpy(time_report, "_Simulation_time_of_this_run.");
    PunchTime(1, time_report);
    SumScaleResult(In_Parm, &Out_Parm);
    WriteResult(In_Parm, Out_Parm, time_report);
}

/*************************************************************
    function definition
*************************************************************/
double f(double x[S])
{
    int i;
    double f;
    f=0.;
    in_parm.layerspecs[1].mua=x[0];
    in_parm.layerspecs[1].mus=x[1];
    in_parm.layerspecs[1].g=x[2];
    in_parm.layerspecs[1].n=x[3];
    InitOutputData(in_parm, &out_parm_opti);

    DoOneRun(&in_parm,&out_parm_opti);
    ReportResult(in_parm, out_parm_opti);

    for(i=0;i<50;i++)/*nr==50*/
        f+=(out_parm.Tt_r[i]-out_parm_opti.Tt_r[i])*(out_parm.Tt_r[i]-out_parm_opti.
         Tt_r[i]);
    f=-f;/*to find the minimum of f equal to find the maximum of -f*/
    return f;
}

/*************************************************************
    calculate fitness
*************************************************************/
void fitness(double fv[N+1],double pf[N+1])
{
    double Cmin;
    double Sumf=0.;
    int i;
    Cmin=fv[1];
    Sumf+=fv[1];
    for( i=2;i<=N;i++)
    {
        Sumf+=fv[i];
        if((Cmin-fv[i])>1.0E-8)
```

```
            Cmin=fv[i];
    }
    Sumf=Sumf-Cmin*N;
    for( i=1;i<=N;i++)
        pf[i]=(fv[i]-Cmin)/Sumf;
}

/************************************************************
    select individual to do local search
************************************************************/
int individual_select (double pf[])
{
    int m;
    double gamblep;
    double gamblesum;
    gamblep=(double)(Gamblerand1())/GambleMAX1;
    gamblesum=0.0;
    for(m=1;m<=N;m++)
    {
        gamblesum=gamblesum+pf[m];
        if((gamblesum-gamblep)>1.0E-20)
            break;
    }
    if(m>N)
        m=N;
    return m;
}

/************************************************************
  roulette , select individual to be replaced by new point
************************************************************/
int individual_replace (double pf[])
{
    int m;
    double gamblep;
    double gamblesum;
    gamblep=(double)(Gamblerand2())/GambleMAX2;
    gamblesum=0.0;
    for(m=1;m<=N;m++)
    { /*the bigger fitness ,the less to be replaced*/
        gamblesum=gamblesum+(1-pf[m])/(N-1);
        if((gamblesum-gamblep)>1.0E-20)
            break;
    }
```

```
    if(m>N)
        m=N;
    return m;
}

/*************************************************************
    adaptive local search of AQMC method
*************************************************************/
void laqmc(double x[N+1][S],double c[S],double *fv_c, int Ni,double *search_step,
        double c3,int *vanish)
{
    double gcx[N+1][S];
    double flocalmax=*fv_c;
    int j,k;
    int indomain;
    double radius=*search_step;
    double tempx;
    for(j=0;j<S;j++)
        gcx[0][j]=c[j];

    for(j=1;j<=Ni;j++)
    {
        for(k=0;k<S;k++)
            gcx[j][k]=gcx[0][k]+radius*(2*x[j][k]-(xrange[k][0]+xrange[k][1]));
        indomain=1;
        for(k=0;k<S;k++)
        {
            if((xrange[k][0]-gcx[j][k])>1.0E-6||(gcx[j][k]-xrange[k][1])>1.0E-6)
                indomain=0;
        }
        if(indomain)
        {
            tempx=f(gcx[j]);
            if((tempx-flocalmax)>1.0E-8)
            {
                flocalmax=tempx;
                for(k=0;k<S;k++)
                    gcx[0][k]=gcx[j][k]; /*to store the tempory max value*/
            }
        } /*end of indomain*/
        else
        {
            (* vanish)++;
            printf("\t%dth_point_not_in_the_domain\t",j);
```

```
            printf("(%f,%f,%f,%f)\n",gcx[j][0], gcx[j][1], gcx[j][2], gcx[j][3]) ;
        }
    } /*end of for j*/
    if((flocalmax-*fv_c)>1.0E-8)
    {
        radius=fabs(c[0]-gcx[0][0]) /(xrange[0][1]-xrange[0][0]) ;
        c[0]=gcx[0][0];
        for(k=1;k<S;k++)
        {
            tempx=fabs(c[k]-gcx[0][k])/(xrange[k][1]-xrange[k][0]) ;
            if((tempx-radius)>1.0E-10)
                radius=tempx;
            c[k]=gcx[0][k]; //to store the tempory max value
        }
        * search_step=radius;
        * fv_c=flocalmax;
    }
    else
        * search_step=(*search_step)*c3;
}

/*************************************************************
    report the result
*************************************************************/
void report(double fglobalmax[S+1],int count, int vanish)
{
    int j;
    FILE *result;
    if( ( result  = fopen( "mcmlmin", "a" )) == NULL )
    {
        printf( "The_file_'mcmlmin'_was_not_opened\n" );
        exit(0);
    }
    fprintf( result,"___mcmlmin=%.10f\n",-fglobalmax[S]);
    for(j=0;j<S;j++)
        fprintf(result,"x[%d]=%.10f\t",j,fglobalmax[j]);
    fprintf(result,"\n");
    fprintf(result,"count(the_calculated_function_value_number)=%d\n",count-vanish
        );
    fprintf(result,"
        ==========\n");
    fclose(result);
}
```

```
/*************************************************************
    look for the global maximum
*************************************************************/
void fmax()
{
    FILE *result;
    int vanish=0;
    int count=0;/*to count the calculation function value number*/
    double x[N+1][S];/*the initial N Sobol' random number*/
    double fx[N+1][S];/*variable(x) of each individual*/
    double pf[N+1];/*fitness of each individual*/
    double fv[N+1];/*function value of each individual*/
    double ss[N+1];/*search radius of each individual*/
    double fglobalmax[S+1];/*to store the varible and max value of the function*/
    int fglobalposition ;
    int i,j,k,m;
    int Ni;/*the search number in local search*/
    double epslon=0.25;/*the initial value of search radius*/
    /*parameters of AQMC optimization method*/
    double c1=0.04;
    double c2=1.0;
    double c3=epslon*epslon;
    double c4=0.04;
    int newN=c4*N;
    double radius=1.0;
    double temp;
    double evolution;
    double lseff=1.0, nieff=1.0;/*local search and new individual performance/cost
        ratio*/
    double ls_max_initial,ls_max_current, ni_max_initial ,ni_max_current;
    double ls_N=0,ni_N=0;/*the count of local search and new individual*/
    double newp=0.2;/*the probability to generate new individuals*/
    double rnd;/*random number*/
    int ini,run;

    static double xsob[S+1];
    srand((unsigned)time(NULL));/*give the rand seed*/

    printf( "The_Monte_Carlo_random_search_method_for_the_global_optimum_starts
        ...\n");
    if( ( result  = fopen( "mcmlmin", "w" )) == NULL )
    {
        printf( "The_file_'mcmlmin'_was_not_opened\n" );
        exit(0);
```

```
}
 fprintf ( result ,"===The_Monte_Carlo_random_search_method_for_the_global_
     optimum===\n\n");
 fprintf ( result ,"N(population_size)=%d\n",N);
 fprintf ( result ,"epslon(the_initial _search_radius)=%f\n",epslon);
 fprintf ( result ,"c1=%f\tc2=%f\tc3=%f\tc4=%f\n",c1,c2,c3,c4);
 fprintf ( result ,"\n");
 fprintf ( result ,"\n");
 fprintf ( result ,"=====================\n");
 fclose ( result );

 i=1;
 printf("Generation_%d(initial_population)...\n",i);
 /*generate N quasirandom dimension S*/
 ini=-1;
sobseq(&ini,xsob);/*the Sobol' generator  initilization */
run=S;
count=count+N;
for( i=1;i<=N;i++)
{
     sobseq(&run,xsob);
     for(j=0;j<S;j++)
    {
          x[i][j]=xrange[j][0]+(xrange[j][1]-xrange[j][0]) *xsob[j+1];
          fx[i][j]=x[i][j];
    }
     fv[i]=f(fx[i]);
     printf("\t_%dth_point:_%f(%f,%f,%f,%f)\n",i,-fv[i],x[i][0], x[i][1], x[i][2], x[i
      ][3]) ;
     ss[i]=epslon;
}
/*end of generating N quasirandom, dimension S*/

/*globalmax*/
 fglobalposition =1;
for( i=2;i<=N;i++)
{
     if((fv[i]-fv[ fglobalposition ])>1.0E-8)
          fglobalposition =i;
}
for(i=0;i<S;i++)
     fglobalmax[i]=fx[ fglobalposition ][ i ];
fglobalmax[S]=fv[fglobalposition ];
/*to calculate the fitness*/
```

```
fitness (fv, pf);
i=1;
report(fglobalmax,count,vanish);
ni_max_initial =fglobalmax[S];
ls_max_initial =fglobalmax[S];
ls_max_current=fglobalmax[S];
ni_max_current=fglobalmax[S];
i=2;
do
{
    printf("Generation_%d...\n",i);
    printf("\t_probability_to_generate_new_individuals_%f\n",newp);
    /*  fitness  */
    rnd=(double)(rand())/RAND_MAX;
    if(rnd<newp)/*generate new individual*/
   { printf("\tgenerate_%d_new_points...\n",newN);
      count=count+newN;
      ni_N+=newN;
      /*to find the invidicual to be replaced by new point*/
      for(j=1;j<newN;j++)
     {
         do
        {
            m=individual_replace(pf);
        }
        while(m==fglobalposition);/*keep the best individual*/
        /*to generate one new point*/
        sobseq(&run,xsob);
        for(k=0;k<S;k++)
            fx[m][k]=xrange[k][0]+(xrange[k][1]-xrange[k][0])*xsob[k+1];
        fv[m]=f(fx[m]);
        printf("\t_%dth_point:_%f(%f,%f,%f,%f)\n",j,-fv[m],fx[m][0],fx[m][1],
    fx[m][2],fx[m][3]);
        ss[m]=epslon;
        if((fv[m]-fglobalmax[S])>1.0E-8)
        {
            fglobalposition =m;
            fglobalmax[S]=fv[m];
            for(k=0;k<S;k++)
                fglobalmax[k]=fx[fglobalposition ][k];
            report(fglobalmax,count,vanish);
            ni_max_current=fv[m];
            printf("\tFind_max_when_generate_new_individual!\n");
        }
```

```
        } /*end of generating  New points*/
          nieff =(ni_max_current-ni_max_initial)/ni_N;
          printf("\tnew_individuals_performance/cost_ratio_%f\n",nieff);
          fitness (fv,pf);
      } /*end of if(newp .... )*/
       else /* local search*/
      { /*to select one point for local searching*/
          m=individual_select(pf);
          Ni=(int)(c2*N*DMAX(ss[m],c1));
          ls_N+=Ni;
          count=count+Ni;
          temp=fv[m];
          printf("\tlocal_search,_%d_points...\n",Ni);
          /* local search*/
          laqmc(x,fx[m],&(fv[m]),Ni,&(ss[m]),c3,&vanish);
          if(fv[m]-temp>1.0E-5)/*if find new local max*/
        {  fitness (fv,pf);
        }
          /* global max*/
          if((fv[m]-fglobalmax[S])>1.0E-8)
        {
              for(j=0;j<S;j++)
                  fglobalmax[j]=fx[m][j];
              fglobalmax[S]=fv[m];
              fglobalposition =m;
              report(fglobalmax,count,vanish);
              ls_max_current=fv[m];
              printf("\tFind_max_in_local_search!\n");
        }
          lseff =(ls_max_current-ls_max_initial)/ls_N;/*performance/cost ratio of
       local search*/
          printf("\tlocal_performance/cost_ratio_%f\n",lseff);
      } /*end of local search*/
       /*calculate the probability of generating new points*/
       if( lseff <1.0E-10 && nieff< 1.0E-10)
          newp=0.2;
       else
          newp=0.2*nieff/(0.8*lseff+0.2*nieff);
       i++;
   } while(fglobalmax[S]<-1.0E-5);
}

time_t PunchTime(char F, char *Msg)
{
```

```
#if GNUCC
    return(0);
#else
static clock_t ut0; /* user time reference. */
    static time_t  rt0; /* real time reference. */
    double secs;
    char s[STRLEN];

    if(F==0)
    {
        ut0 = clock();
        rt0 = time(NULL);
        return(0);
    }
    else if(F==1)
    {
        secs = (clock() - ut0)/(double)CLOCKS_PER_SEC;
        if (secs<0)
            secs=0; /* clock() can overflow. */
        sprintf(s, "User time: %8.0lf sec = %8.2lf hr. %s\n",
                secs, secs/3600.0, Msg);
        puts(s);
        strcpy(Msg, s);
        return(difftime(time(NULL), rt0));
    }
    else if(F==2)
        return(difftime(time(NULL), rt0));
    else
        return(0);
#endif
}

/*************************************************************
 * Get the file name of the input data file from the
 * argument to the command line.
 ****/
void GetFnameFromArgv(int argc, char * argv[], char * input_filename)
{
    if(argc>=2)
    { /* filename in command line */
        strcpy(input_filename, argv[1]);
    }
    else
        input_filename[0] = '\0';
```

```
}


void main(int argc, char *argv[])
{
    char input_filename[STRLEN];
    FILE *input_file_ptr;
    short num_runs;/* number of independent runs. */
    /*InputStruct in_parm;
    OutStruct out_parm;*/
    GetFnameFromArgv(argc, argv, input_filename);
     input_file_ptr  = GetFile(input_filename);
    CheckParm(input_file_ptr, &in_parm);
    num_runs=ReadNumRuns(input_file_ptr);
    ReadParm(input_file_ptr, &in_parm);
    ReadParm(input_file_ptr, &in_parm_opti);

    InitOutputData(in_parm, &out_parm);
    DoOneRun(&in_parm,&out_parm);
    ReportResult(in_parm, out_parm);

    fmax();
    FreeData(in_parm, &out_parm);
    FreeData(in_parm_opti, &out_parm_opti);
    fclose ( input_file_ptr );
}
```

参考文献

[1] J. Dongarra and F. Sullivan. Guest editors' introduction: the top 10 algorithms. *Computing in Science and Engineering*, 2(1):22-23, January/February 2000.

[2] B. Cipra. The best of the 20th century: editors name top 10 algorithms. *SIAM News*, 33(4):1, 2000.

[3] N. Metropolis and S. Ulam. The monte carlo method. *J. Am. stat. Ass.*, 44:335-341, 1949.

[4] R. E. Caflisch. Monte carlo and quasi-monte carlo methods. *Acta Numerica*, 7:1–49, 1998.

[5] W. Feller. *An introduction to probability theory and its applications: Vol. I.* Wiley, New York, 1971.

[6] H. Niederreiter. *Random number generation and quasi-Monte Carlo methods.* SIAM, Philadelphia, 1992.

[7] R. Y. Rubinstein. *Simulation and the Monte Carlo method.* Wiley, New York, 1981.

[8] H. Niederreiter. *Studies in pure Mathematics(To the memory of Paul Turán)*, pages 523–529. Birkhäuser Verlag, Basel, 1983.

[9] M. P. Allen and D. J. Tildesley. *Computer simulation of liquids.* Oxford University Press, New York, 1989.

[10] W. J. Morokoff and R. E. Caflisch. A quasi-monte carlo approach of particle simulation of the heat equation. *SIAM J. Numer. Anal.*, 30(6):1558–1573, 1993.

[11] C. Lécot and I. Coulibaly. A quasi-monte carlo scheme using nets for a linear boltzmann equation. *SIAM J. Numer. Anal.*, 35(1):51–70, 1998.

[12] M. N. Rosenbluth A. H. Teller N. Metropolis, A. W. Rosenbluth and E. Teller. Equation of state calculations by fast computing machines. *J. chem. Phys.*, 21:1087–1092, 1953.

[13] B. H. Sendov and I. dimov, editors. *International youth workshop on Monte Carlo methods and parallel algorithms.* World Scientific Publishing Co. Ltd., Singapore, 1989.

[14] H. Niederreiter and P. J. Shiue, editors. *Monte Carlo and quasi-Monte Carlo methods in scientific computing.* Springer-Verlag, New York, 1995.

[15] G. Larcher H. Niederreiter, P. Hellekalek and P. Zinterhof, editors. *Monte Carlo and Quasi-Monte Carlo methods 1996.* Springer-Verlag, New York, 1998.

[16] J.E. Gentle. *Random number generation and Monte Carlo methods.* Springer-Verlag, New York, 1998.

[17] B. F. J. Manly. *Randomization, bootstrap and Monte Carlo methods in biology.* Chapman and Hall, London, 2nd edition, 1997.

[18] D. H. Lehmer. *Proc. 2nd sympos. on large-scale digital calculating machinery*, pages 141–146. Harvard University Press, Cambridge, 1951.

[19] J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluation multi-dimensional integrals. *Numer. Math.*, 2:84–90, 1960.

[20] H. Faure. Discrépance de suites associées à un système de numération (en dimension s). *Acta Arith.*, 41:337–351, 1982.

[21] I.M. Sobol'. The distribution of points in a cube and the approximate evaluation of integrals. *USSR Comp. Math. and Math. Phys.*, 7:86–112, 1967.

[22] H. Niederreiter. Point sets and sequences with small discrepancy. *Monatsh. Math.*, 104:273–337, 1987.

[23] H. Niederreiter. Quasi-monte carlo methods and pseudo-random numbers. *Bull. Amer. Math. Soc.*, 84(6):957–1041, 1978.

[24] I. A. Antonov and V. M. Saleev. An economic method of computing lp_τ-sequences. *USSR Comput. Math. Math. Phys.*, 19:252–256, 1979.

[25] P. Bratley and B. L. Fox. Algorithms 659 implementing sobol's quasirandom sequence generator. *ACM Trans. Math. Software*, 14(1):88–100, 1988.

[26] W.T. Vetterling W.H. Press, S.A. Teukolsky and B.P. Flannery. *Numerical recipes in C: The art of scientific computing*. Cambridge University Press, New York, 1992.

[27] G. Y. Lei. B-splines smoothed rejection sampling method and its applications in quasi-monte carlo integration. *Journal of Zhejiang University, Science*, 3:339–343, 2002.

[28] K. T. Fang and Y. Wang. *Number-theoretic methods in statistics*. Chapman & Hall, London, 1994.

[29] X. Q. Wang. Improving the rejection sampling method in quasi-monte carlo methods. *Journal of Computational and Applied Mathematics*, 114:231–246, 2000.

[30] B. Moskowitz and R.E. Caflisch. Smoothness and dimension reduction in quasi-monte carlo methods. *Math. Comput. Modelling*, 23(8-9):37–54, 1996.

[31] L. Schumaker. *Spline functions: basic theory*. Wiley, New York, 1981.

[32] B. L. Fox P. Bratley and H. Niederreiter. Implementation and tests of low-discrepancy sequences. *ACM Transactions on Modeling and Computer Simulation*, 2(3):195–213, 1992.

[33] J. M. Hammersley and D. C. Handscomb. *Monte Carlo methods*. Methuen & Co Ltd, Lodon, 1964.

[34] J. M. Hammersley and K. W. Morton. A new monte carlo technique: antithetic variates. *Proc. Camb. Phil. Soc.*, 52:449–475, 1956.

[35] S. Haber. A modified monte carlo quadrature. ii. *Math. Comp.*, 21:388 397, 1967.

[36] A. Karaivanova and I. Dimov. Error analysis of an adaptive monte carlo method for numerical integration. *Mathematics and Computers in Simulation*, 47:201–213, 1998.

[37] I. M. Sobol' and A. V. Tutunnikov. A variance reducing multiplier for monte carlo integration. *Math. Comput. Modelling*, 23(8-9):87–96, 1996.

[38] G. Y. Lei. Adaptive quasi-monte carlo method for multiple-extrema optimization. *Control Theory and Applications*, 19:431–434, 2002.

[39] G.Y. Lei. Adaptive random search in quasi-monte carlo methods for global optimization. *Computers and Mathematics with Applications*, 43:747–754, 2002.

[40] H. Niederreiter and P. Peart. Localization of search in quasi-monte carlo methods for global optimization. *SIAM J. Sci. Stat. Comput.*, 7(2):660–664, 1986.

[41] Y. Wang and K. T. Fang. Number theoretic methods in applied statistics. *Chinese Ann. Math. Ser.*, B,11:41–55, 859–914, 1990.

[42] H. Niederreiter. *Topics in classical number theory*, pages 1163 1208. North-Holland, Amsterdam, 1984.

[43] J. H. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, Ann Arbor, 1975.

[44] K. A. DeJong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975.

[45] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley, Reading, Mass., 1989.

[46] Z. Michalewicz. *Genetic algorithms + data Structures = evolution programs.* Springer-Verlag, Berlin, 1994.

[47] J. R. Koza. *Genetic programming.* MIT Press, Cambridge, Mass., 1992.

[48] B. Souček. *Dynamic, genetic, and chaotic programming.* John Wiley & Sons, 1992.

[49] Y. K. Wan. *C++ language and the object-oriented programming.* The press of Tsinghua University, 1998.

[50] S. L. Jacques S. A. Prahl, M. Keijzer and A. J. Welch. A monte carlo model of light propagation in tissue. *SPIE Institute Series*, 5:102 111, 1989.

[51] L. G. Henyey and J. L. Greenstein. Diffuse radiation in the galaxy. *Astrophys. J.*, 93:70–83, 1941.

[52] S. L. Jacques L. H. Wang and L. Q. Zheng. Mcml - monte carlo modeling of photon transport in multi-layered tissues. *Computer Methods and Programs in Biomedicine*, 47:131–146, 1995.

[53] J. H. Halton and G. B. Smith. Algorith 247: Radical-inverse quasi-random point sequence. *Comm. ACM*, 7:701–702, 1964.

索引