

课 程 大 作 业

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **姓 名:** | 张凯珺  李宗宇  拓欣 | | **学 号：** | 923130830237  922114740127 |
| **课 程:** | 基于解释性语言的高效科学计算 | | | |
| **题 目:** | 求复杂函数在积分 | | | |
| **任课教师:** | 周毅 | 能源与动力工程学院 | | |

2024 年 4 月

# 使用蒙特卡洛法通过Numba和Julia两种方式实现题目三

我组选择题目三：求复杂函数在积分，实现方法为Numba和Julia。两种语言的源代码如下：

【Python代码】

【Julia代码】

下面我们首先介绍我们的三种算法实现，然后介绍我们在工程上采取的一些手段。

## 方法一：蒙特卡洛法的一般实现

接下来逐行解释算法：

1、1~3行分别调用了从timeit模块中导入timeit函数，该函数用于测量小段Python代码的执行时间；导入numpy库，并用np作为缩写，（numpy是一个用于处理数组、矩阵和进行数学运算的库）；从numba库中导入jit编译器装饰器，它可以用来优化函数性能，特别适用于numpy数组操作。

第二段程序设置了这个题目的一些参数，其中layers变量适用于第三种方法，方法一未涉及。

定义一个函数func，使用numpy的multiply函数来执行乘法操作，先计算2sin(x)，通过嵌套multiply函数实现。再计算x^3 + x^2 + 2x + 3。这里通过嵌套add和power函数实现。

计算积分（调用函数）：使用numpy的random.uniform函数生成times个在[bottom, top]均匀分布的随机数，赋值给变量x。这些随机数代表了x的值。调用之前定义的func函数，并将x作为参数传入，得到的结果赋值给变量y。此时y包含了与x对应的函数值，首先计算y的平均值np.mean(y)，然后计算区间长度np.subtract(top, bottom)，最后将两者相乘得到dist。这个dist可以看作是函数在[bottom, top]区间上的积分的近似，这里用到数学里的第一积分中值定理！

【y x拟合图】

总结：

这个函数的主要目的是计算一个数学函数在指定区间[bottom, top]上的积分的近似值。它通过以下步骤实现：

1.生成大量（times个）在指定区间内均匀分布的随机数x。

2.使用func函数计算这些x值对应的y值。

3.计算y值的平均值。

4.将y的平均值乘以区间的长度（top - bottom），得到积分的近似值。

## 方法二：重要性采样法

源代码解释如下：

y的生成是基于均匀分布的。

计算函数值与概率分布值的比例。

积分值（重要性采样的近似值），这里主要应用概率统计与数理统计的专业知识！期望值使用积分形式。

应用到被积函数值与对应概率密度的比值，它们的平均值乘以区间长度给出了积分的近似值。

【y x拟合图】

总结：

该方法专业性较强，可能涉及到专门的采样方法，比如逆变换采样、接受-拒绝采样或者马尔科夫链蒙特卡洛方法（MCMC）等，但对于该问题来说就是使用正常期望值的计算方法即可！

## 方法三：分层采样法

源代码解释如下：

使用前述的layers变量，表示将积分区间分成的层数。这里设为10的4次方，即10000层。层数越多结果越详细，当然计算量和计算时间因此也会增加。

使用numba的jit装饰器来优化这个函数，使其能够更快地运行。

使用numpy的arange函数生成一个从0到layers-1的整数序列，代表每一层的索引。

分别计算当前层的起始边界、结束边界，在计算当前层需要生成的随机数的数量，这里使用整除运算符//来确保每层生成的随机数数量相同，同时在当前层对应的区间内生成指定数量的随机数，赋值给lay\_sample。

分层计算：计算当前层随机样本对应的函数值的平均值，即单层积分的近似值。

将当前层的积分值乘以区间长度再除以层数，得到加权平均值，并累加到dist中，得到结果！

【y x拟合图】

总结：

这个layer函数实现的是分层采样（stratified sampling）的思想，用于估计函数在指定区间上的积分值。分层采样的基本思想是将整个积分区间分成若干个子区间（或层），然后在每个子区间内独立地生成随机样本，并计算每个子区间的积分近似值，最后将这些近似值加权平均得到整个区间的积分近似值。在这个函数中，layers变量表示分层的数量，times // layers确保每个子区间内生成的随机样本数量大致相同。然后，通过一个循环，执行上述操作。

## 工程手段一：Numba JIT及多线程

我们选择的两种实现方式之一就是Numba，所以我们去了解了一下Numba的特性，为以上算法开启了JIT、nopython编译模式、nogil多并发模式 和parallel并行计算模式，速度提升极大

【不同模式下的速度对比图】

在Numba的加持下，我们的代码执行速度可以接近原生C/CPP执行速度。

## 工程手段二：通过NVIDIA CUDA使用显卡进行计算

由于GPU精于浮点运算，这正迎合了我们计算亿个以上浮点数的需求，因此我们分别尝试了Numba的CUDA JIT以及为赋能Numpy GPU计算能力的CuPy，在算法不变的基础上将运算过程移植到了GPU上，在NVIDIA RTX 3060 Laptop显卡下进行了测试，性能提升数个量级。

【CPU与GPU速度对比图】

而这三种算法在CUDA的加持下速度差距也越来越大

【GPU下速度对比图】

## 总结

在我们研究这一题目的过程中，我们接触到了许多过去未曾接触到的东西。最初我们实现了相关算法，但是计算速度却相当的慢，虽然我们按照老师课上所教授的那样运用了Numba的*nopython=true*功能，但是得到的速度提升却很有限。在查看了电脑的CPU使用情况后我们发现只有一个核在跑，于是我们去查询资料并知道了*parallel*等功能，在开启后程序的执行速度突飞猛进，所有的CPU核心都被使用。

【开启前后速度与CPU使用情况对吧】

在解决了工程上的一些问题后，我们又开始通过翻阅专业书籍和搜集论文等手段了解到了许多基于蒙特卡洛法的积分计算改进方法，例如我们已经实现的重要性采样法和分层采样法，借助这些算法我们的计算速度和结果精确程度得到了很大的提升。

【不同算法计算速度/结果对比图】

除此之外，我们还了解到了通过遗传算法提升结果准确率、通过神经网络迭代等方法，但因为时间不足且我们目前大一、大二所学到的知识不足于让我们深入理解相关算法故我们没有进行实现。

于是我们又开始着手于研究通过工程手段进一步提升计算量，通过增大计算量来提升结果精度并减少计算时间，受到人工智能技术的启发，我们尝试了通过NVIDIA CUDA来进行计算并且成果斐然。

【CPU/GPU计算速度对比图】

我们相信一定还有更多方法可以用来提升精度以及减少计算用时，我们的作业代码已经开源到<https://github.com/Hobr/monte-carlo-integration>供大家互相学习交流以及提出改进意见，我们也希望以此为契机来让我们更加深入地学习和参与到科学计算这一领域中来。在此我们非常的感谢老师您的指导。