数学实验 exp10 实验报告

计65 赖金霖2016011377

1. 问题背景

对于方阵An×n，我们可以定义它的积和式per(A)如下：

其中π为1~n的排列。本次实验的目的是通过几种方法，计算随机生成的0/1矩阵的积和式，并比较各方法的优劣。

1. Naïve算法

很显然，我们可以枚举π，直接进行计算，这种算法的时间复杂度为O(n!)，我们在接下来的内容中称其为naïve算法。

1. Ryser算法

对这个问题，现在已知的最优的确定性算法叫做Ryser算法，它把per(A)变换为

这样，我们只要枚举{1,…,n}的子集，就可以计算出per(A)了。注意到枚举子集的复杂度为O(2n)，而枚举子集后的计算是O(n2)的，所以总时间复杂度为O(n22n)。当我们使用格雷码枚举子集时，每次枚举的集合S和上一次枚举的集合S’之间只相差一个元素，所以可以设bi=Σj∈Saij，每次枚举后bi最多只用加或减一个元素。这样做之后的时间复杂度为O(n2n)，接下来的Ryser算法就是指这一算法。

1. MC方法

0/1矩阵的积和式的计算是困难的，可以考虑采用MC方法近似。对于一个无偏估计X， 设X1~XN与X同分布且相互独立，而Y=ΣNi=1 Xi，那么如果我们需要Y是一个以1-δ的概率产生一个相对误差不超过𝜀的估计：

则需要满足

其中E[X2]/E[X]2被称为critical ratio。

1. GG算法

GG算法是per(A)的一个无偏估计，它的形式如下

其中⊙为对应项相乘，M-1,1为和A一样大的，各元素以0.5概率取-1或1的矩阵。可以证明，GG算法的critical ratio不超过3n/2。通常行列式通过高斯消去计算的复杂度为O(n3)。

1. KKLLL算法

KKLLL算法也是per(A)的一个无偏估计，它的形式如下

其中⊙为对应项相乘，M-0.5-0.5√3i-,0.5+0.5√3i,1为和A一样大的，各元素以1/3概率取-0.5-0.5√3i-,0.5+0.5√3i或1的矩阵。可以证明，KKLLL算法的critical ratio不超过2n/2。

1. Normal估计

只要M矩阵的各元素是0均值1方差的随机变量，就能构成一个per(A)的无偏估计。本次实验中尝试采用N(0,1)随机产生M矩阵，形式如下

1. GG算法和KKLLL算法的理论效率

设Ryser算法的时间为

GG算法的时间不超过

KKLLL算法的时间不超过

其中C1、C2、C3为常数。在具体实现上，我们可以认为C1、C2和C3是十分接近的数值（彼此的倍数关系很小）。很显然当n很小时，Ryser算法更优，随着n增大，GG算法和KKLLL算法效率会变好。

我们可以计算在不同的δ和𝜀下，使得GG算法优于Ryser算法的理论最小的n：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 𝜀  δ | 0.2 | 0.1 | 0.05 | 0.01 |
| 0.2 | 88 | 100 | 111 | 136 |
| 0.1 | 91 | 103 | 114 | 139 |
| 0.05 | 94 | 105 | 116 | 141 |
| 0.01 | 97 | 108 | 119 | 144 |

同理，使得KKLLL算法优于Ryser算法的理论最小的n为：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 𝜀  δ | 0.2 | 0.1 | 0.05 | 0.01 |
| 0.2 | 31 | 36 | 40 | 51 |
| 0.1 | 32 | 37 | 42 | 52 |
| 0.05 | 33 | 38 | 43 | 53 |
| 0.01 | 35 | 39 | 44 | 55 |

容易看出，在普通的电脑能计算的规模下（如n≤20时），GG算法和KKLLL算法没有比Ryser算法更优越。当要计算很大的矩阵的积和式时，GG算法和KKLLL算法能优于Ryser算法。

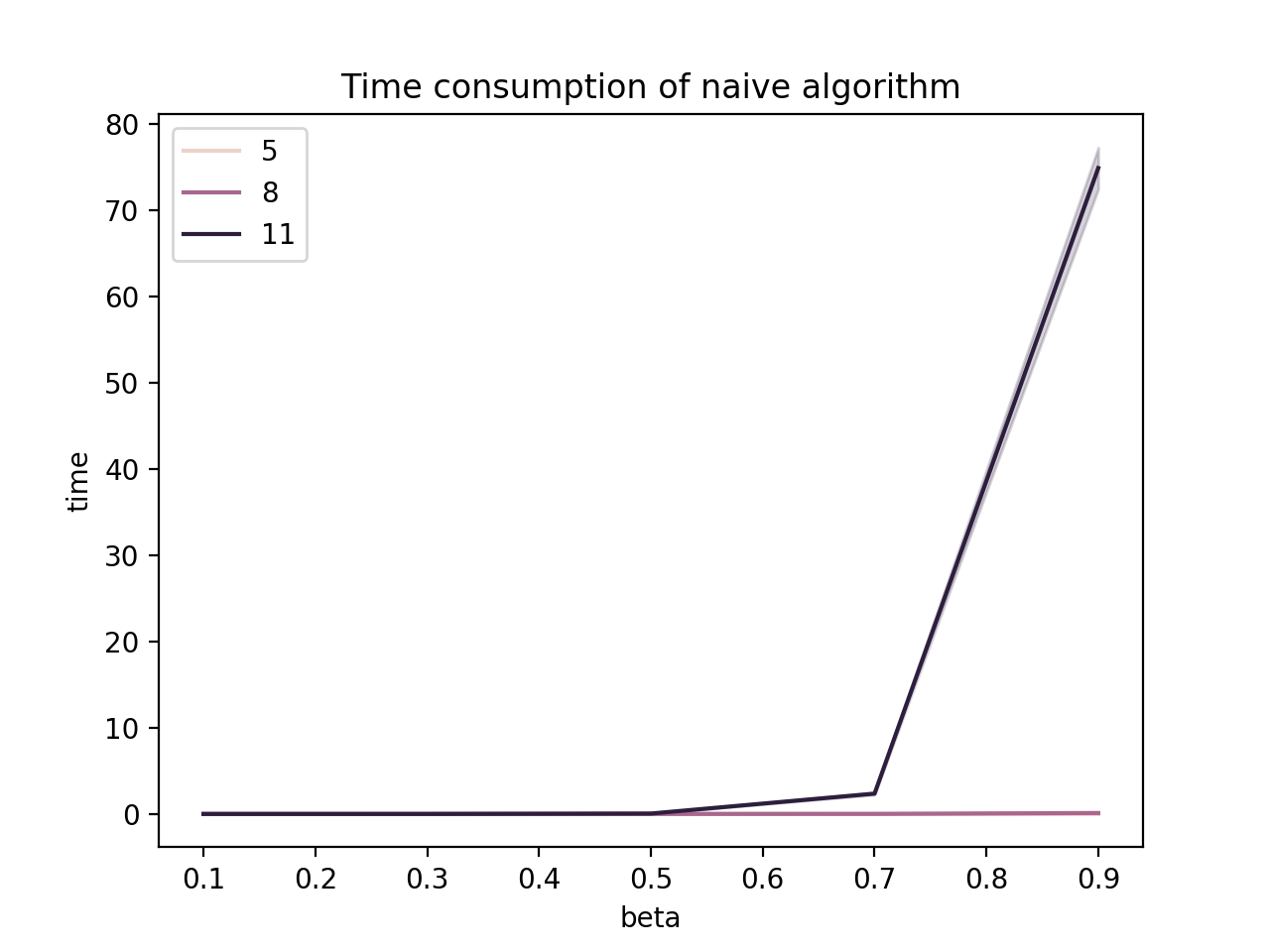
1. 不同算法的效率对比

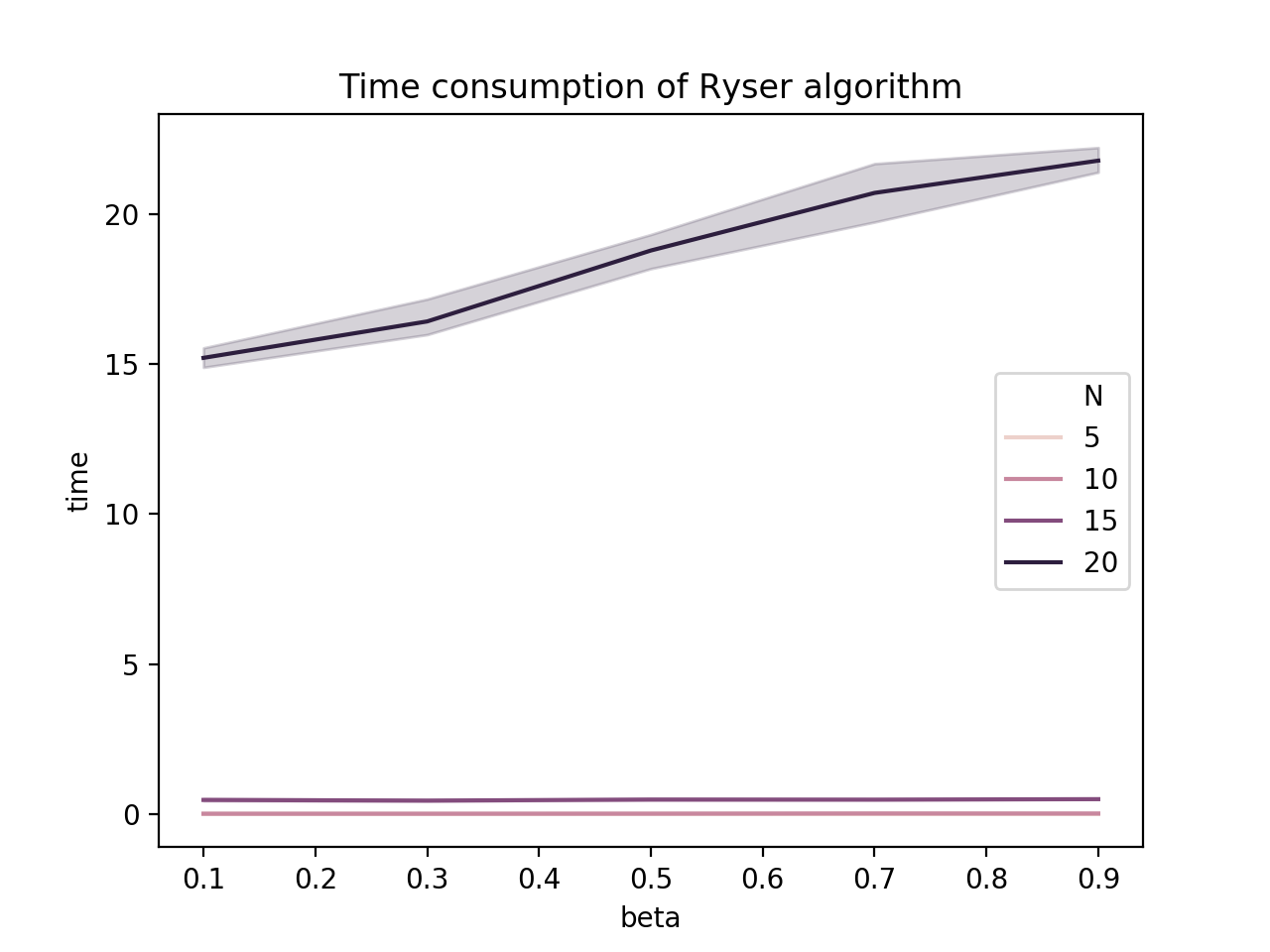
在实验中，我们随机生成矩阵A，A中每个元素为1的概率记为β。

需要注意的是，由于我使用的是不适用于直接科学计算的Python（且没有像Python库函数一样调用C语言），所以时间效率要比其他语言如Matlab低。

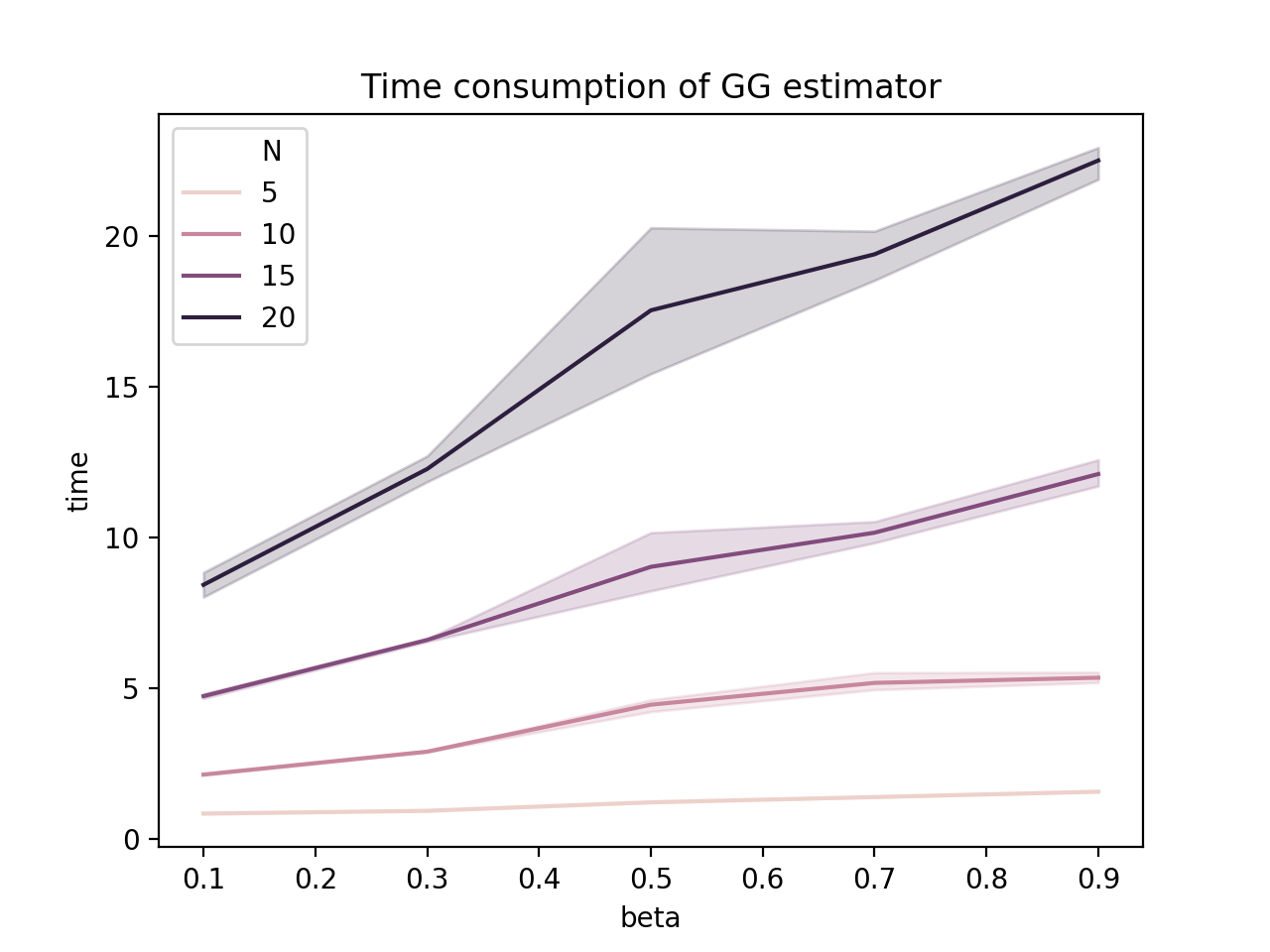
由于GG算法、KKLLL算法和Normal估计理论上要求的N都很大，在个人电脑上计算不现实，所以对所有MC方法，如果不加说明，取N=20000。

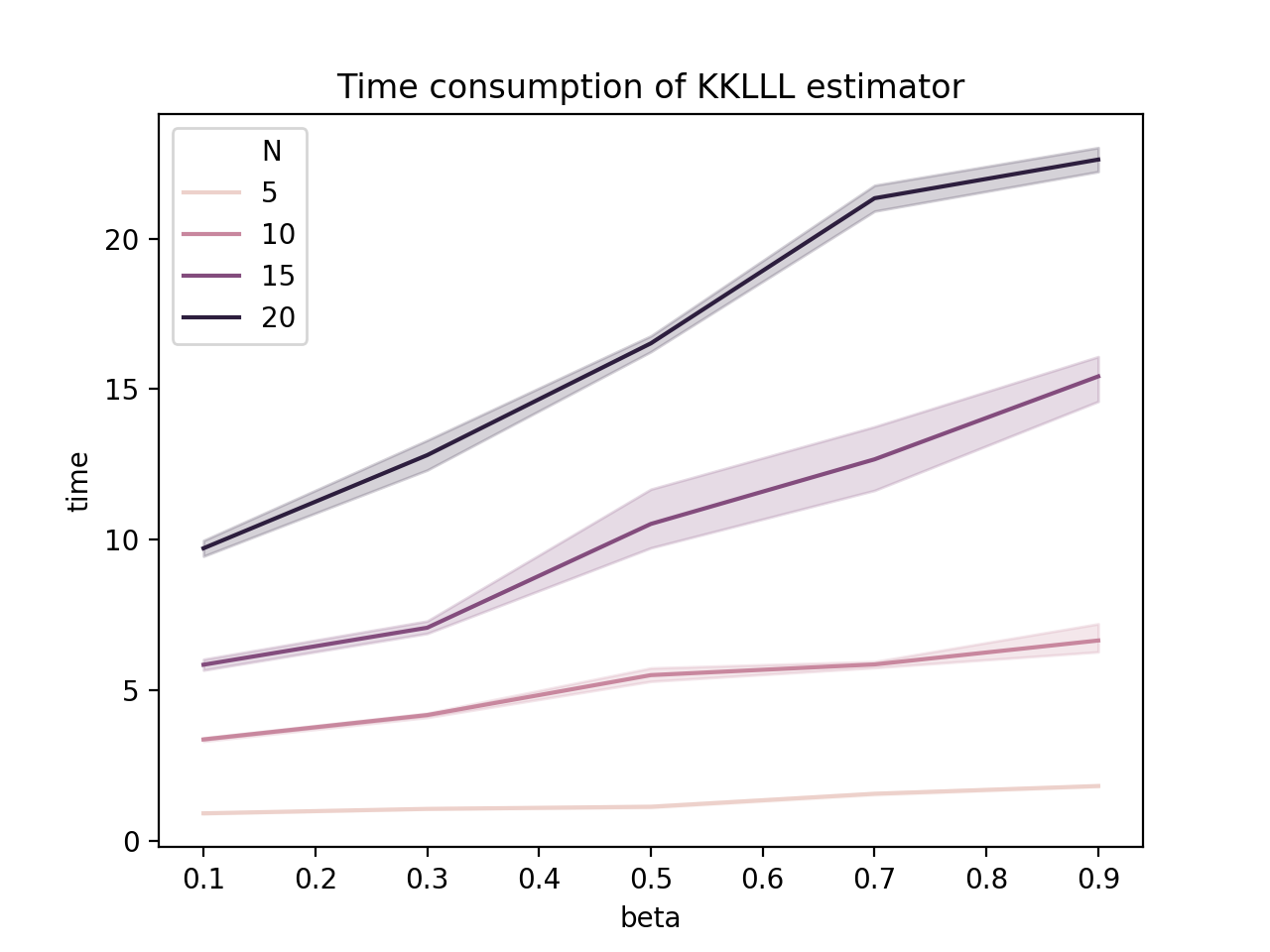
1. 时间效率（本部分图中的N指n，曲线附近error band为ci）

Naïve算法的时间效率与β、n的关系如下：

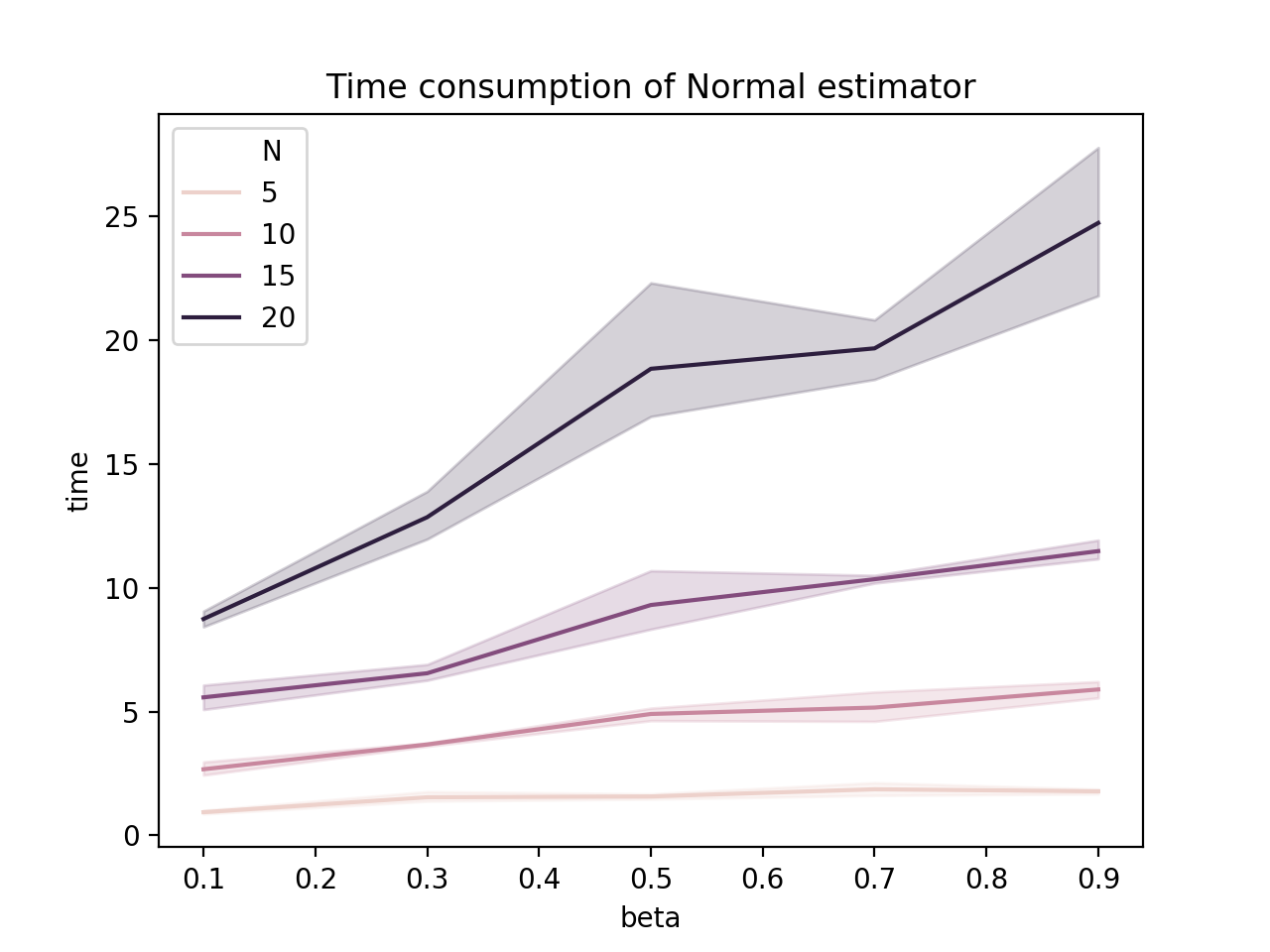
Ryser算法的时间效率与β、n的关系如下：

由于Naïve算法和Ryser算法是指数算法，所以N的提升对时间的影响很大。此外，当n较大时，可以明显看出算法执行时间随β的增大而增大，这可能是因为更多的加减运算导致的。

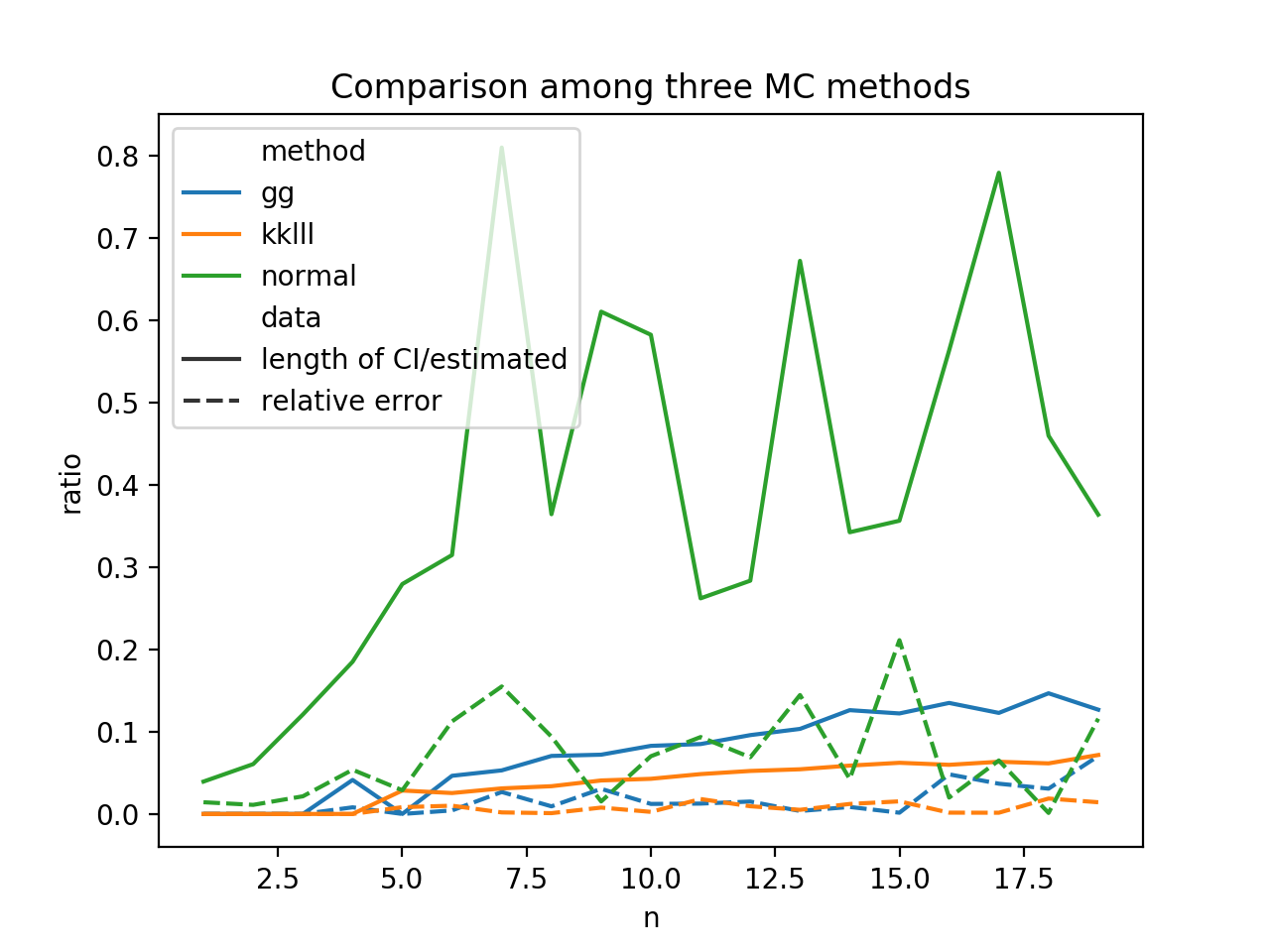
GG算法的时间效率与β、n的关系如下：

KKLLL算法的时间效率与β、n的关系如下：

Normal估计的时间效率与β、n的关系如下：

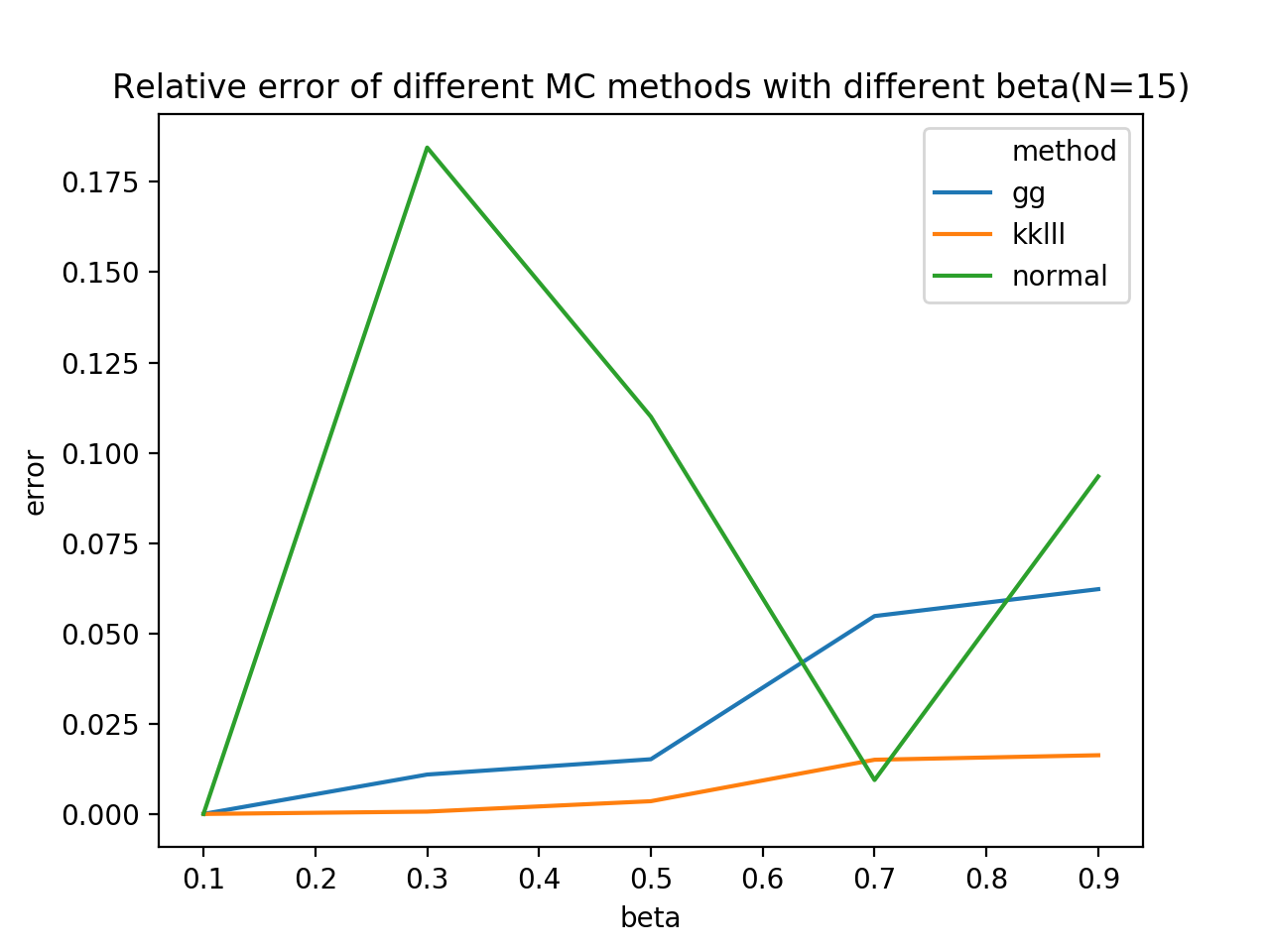
可以看出，MC方法的运行时间都随β和N的增大而增大。其中β越大，矩阵约稠密，计算行列式的速度越慢；N越大，矩阵规模越大，行列式计算也越慢。

1. 置信区间与相对误差

为了刻画MC方法的结果的精确度，我们引入两个变量“置信区间相对长度RL”和“相对误差RE”。设真实值为real，估计值为estimated，置信区间长度为length，则有

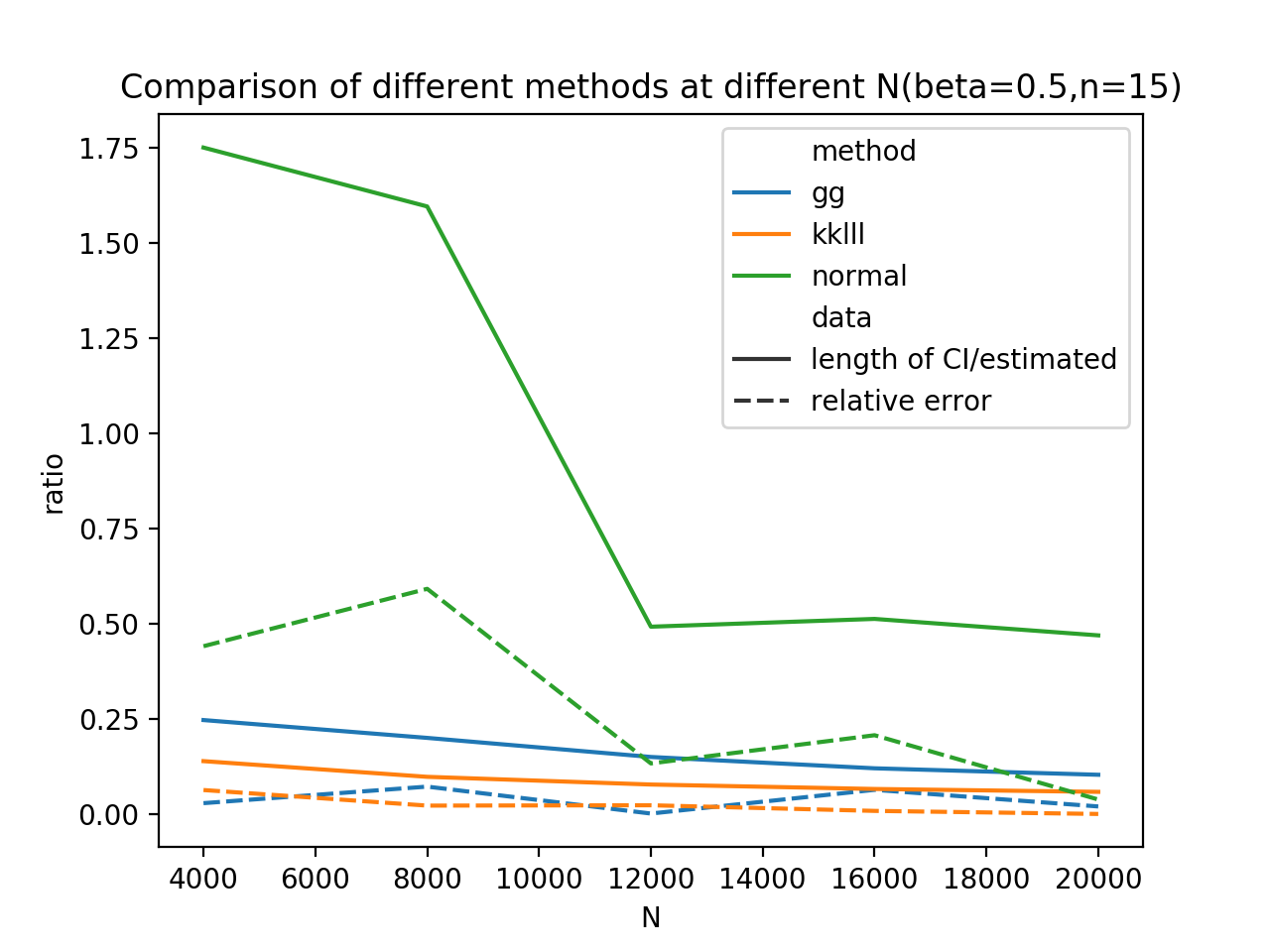
对不同的算法，在β=0.5时，不同的矩阵规模n下，RL和RE如下图：

可以看出，Normal估计（绿色）的相对置信区间最大，这可能是它不作为实际应用的理由。而GG算法（蓝色）和KKLLL算法（橙色）中，KKLLL算法无论是在置信区间上还是相对误差上都比GG算法优。此外，随着n的增大，各算法的估计偏差也变大，这是因为实验中的样本数N是固定的，而为了达到一定的精度而所需的样本数随n的增大而增大。

当N=15时，各MC算法的相对误差随β的曲线如下

从总体趋势上看，当β越高时，误差越大。

1. MC方法的N的影响

容易知道，对同一个MC方法，不同的N能产生不同的精度，我们可以对比各MC方法在不同的N下的RL和RE

在β和n固定的情况下，对per(A)估计随N的增大而变准。

1. 详细数值

在n=20，β=0.5，N=20000时，随机执行一次，各算法的结果为

|  |  |  |  |
| --- | --- | --- | --- |
| 方法 | 结果 | 相对置信区间长度RL | 相对误差RE |
| Ryser | 4461838126392 |  |  |
| GG | 4413282651905 | 0.1437 | 0.0109 |
| KKLLL | 4581015416762 | 0.0823 | 0.0267 |
| Normal | 5840527380910 | 0.8341 | 0.3090 |

由于只执行了一次，数值仅供参考。

对不同算法分别执行10次，平均时间分别为

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 方法 | Ryser | GG | KKLLL | Normal |
| 时间 | 17.544s | 14.191s | 16.561s | 16.842s |

可以看出，在MC方法中，只涉及整数运算的GG算法要比复数运算的KKLLL算法和实数运算的Normal算法更快。

1. 代码

代码可以在https://github.com/lll6924/math\_exp/tree/master/exp10下找到（per.py和calculator.py）。

1. 参考资料
2. <https://en.wikipedia.org/wiki/Computing_the_permanent#Approximate_computation>

2． Liang, H. , Shi, L. , Bai, F. , & Liu, X. . (2007). Random path method with pivoting for computing permanents of matrices. Applied Mathematics and Computation, 185(1), 59-71.

3. https://icerm.brown.edu/materials/Slides/sp-s14-w4/Permanent\_estimators\_via\_random\_matrices\_]\_Mark\_Rudelson,\_Univeristy\_of\_Michigan.pdf