软件分析

聚类、挖掘和搜索

熊英飞
北京大学
2016

聚类

聚类

- 把数据按相似度分类
  - 最常见的无监督学习方法
- 如何衡量相似度
  - 欧氏距离
  - 协方差
- 如何分类
  - k-means
  - 层次聚类

$$d_{euc}(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$ 欧氏距离

$$\rho(\mathbf{x},\mathbf{y}) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$ 协方差

K-Means

- 输入：数据集，聚类数量k
- 算法
  - 随机分类k个数据到k个类别，每个数据成为该类别的中点
  - 对于每个数据
    - 计算该数据到每个类别中心的距离
    - 将该数据加入最近的类别
    - 重新计算该类别的中点

K-Means优缺点

- 优点：简单快速
- 缺点：
  - 需要指定k
  - 一开始的随机选点对结果影响很大
  - 必须能计算中点
- 针对前两个缺点有若干改进算法
  - 尝试多组k或者随机选点，选出较好的（即每个类别中的点互相接近）
  - 允许拆分和重新组合类别

层次聚类

- 一开始每一个数据是一个类别
- 每次合并两个最相似的类别
  - 如何衡量类别的相似性？
    - 中点的距离
    - 任意两点的最短距离
    - 任意两点的最大距离
    - ……
- 到达停止条件时结束
  - 类别数量少于等于k
  - 类别之间的距离都大于等于d

层次聚类优缺点

- 优点：
  - 可不计算中点
  - 终止条件更灵活
- 缺点：
  - 计算比较慢

聚类的应用：编译器测试排序

- 首先对选择特征，然后基于选择后的特征聚类
- 利用分类器排序的时候，先对类别按类别中最大值进行排序，然后依次从各个类别中选择最优测试输入
- 整体可以获得更稳定的输出

频繁模式挖掘

频繁模式挖掘

- 从一个（大型）数据库中挖掘频繁出现的子项
- 常见频繁模式挖掘
  - 频繁子集(itemset)挖掘
  - 频繁序列挖掘
  - 频繁子树挖掘
  - 频繁子图挖掘

Transaction	Items
t_1	Bread,Jelly,PeanutButter
t_2	Bread,PeanutButter
t_3	Bread,Milk,PeanutButter
t_4	Beer,Bread
t_5	Beer,Milk

{Bread, PeanutButter}是一个频繁子集

频繁模式挖掘通式

- 考虑某种数据结构S
  - S上存在包含关系
  - 任意数据项$s \in S$有大小$d(s)$，$d(s)$为一个包含0的自然数
- 输入：
  - 一组数据项的集合I
  - 一个频率的阈值t
  - 定义：一个数据项s的频率为I中包含s的数据项的个数
  - 定义：一个数据项s是频繁当且仅当s的频率 $>$ t
- 输出：一个频繁数据项的集合F，满足
  - $\forall s, s$的频率 $\geq$ t $\Leftrightarrow s \in F$

Apriori算法

- 频繁子项性质：如果一个子项是非频繁的，那么其超项也是非频繁的
- 算法：

n=1;

找到大小为1的所有频繁项，记为L_1;

do {

n++;

C = $\{ s \mid d(s) = n \wedge \exists s' \in L_{n-1}, s' \subseteq s \}$

L_n=$\{s \mid s \in C \wedge s$是频繁的$\}$

}

优化：
产生C的时候过滤掉包含非频繁子项的数据项

Apriori算法小结

- 优点：
  - 非常容易并行
  - 非常容易实现
- 缺点：
  - 需要反复扫描数据集，效率不高
- 针对特定数据结构有各种专门算法，在使用时可直接找到最新算法使用

关联规则挖掘

- 找到所有频繁且可靠的关联规则
- 关联规则$X \Rightarrow Y$
  - $X \cap Y = \emptyset$，即X和Y没有公共子项
- 频繁的关联规则
  - 同时包含X和Y的数据项数目超过阈值t
- 可靠的关联规则
  - $\frac{\text{同时包含}x\text{和}y\text{的数据项数目}}{\text{包含x的数据项数目}} >$ 阈值c

关联规则挖掘

对任意频繁子项l

对任意数据项$x \subset l$

检查x=>(l-x)是否是可靠的

基于关联规则的API使用错误查找

- Zhenmin Li and Yuanyuan Zhou. PR-Miner: Automatically Extracting Implicit Programming Rules and Detecting Violations in Large Software Code. FSE 2005.
- 很多函数是按顺序调用的。如果漏了一个函数就是Bug。

```
postgresql-8.0.1/src/backend/catalog/dependency.c:
1733 getRelationDescription (StringInfo buffer, Oid relid)
1734 {
1735    HeapTuple  relTup;
          ......
1740    relTup = SearchSysCache(...);
          ......
1796    ReleaseSysCache(relTup);
1797 }
```

基于关联规则的API使用错误查找


```
postgresql-8.0.1/src/backend/commands/tablecmds.c:
5686 AlterTableCreateToastTable(Oid relOid, bool silent)
5687 {
        ......
5692   Relation  class_rel;
        ......
5853   class_rel = heap_openr (...);
        ......
5863   simple_heap_update(class_rel, ...);
        ......
5866   CatalogUpdateIndexes(class_rel, ...);
        ......
5870   heap_close (class_rel, ...);
        ......
5891 }
```

```
linux-2.6.11/drivers/isdn/hisax/config.c:
771 void ll_stop(struct IsdnCardState *cs)
772 {
773     isdn_ctrl ic;
775     ic.command= ISDN_STAT_STOP;
776     ic.driver= cs->myid;
777     cs->iif.statcallb(&ic);
779 }
```

基于关联规则的API使用错误查找

- 从代码中提取API调用序列
- 将API编码成整数
- 应用关联规则挖掘
- 用挖掘出的规则检查代码中的规则违反
- 在Linux代码中检查了60处违反，发现了16个Bug

搜索算法

判定问题和优化问题

- 判定问题：给定输入，输出“是”或者“否”的问题
  - 程序验证
- 优化问题：给定目标空间和一个适应性（fitness）函数f，f把任意目标空间的值映射成一个实数，求让f结果尽可能大的目标空间的值

搜索方法解决优化问题

- 搜索算法输入
  - 一个目标空间
  - 一个适应度函数fitness function
- 搜索算法输出
  - 目标空间的一个值，该值在适应度函数下最大

搜索算法的基本流程

```
while(! done()) {
  s = getNextValue()
  v = getFitness(s)
  analyzeValue(v, s)
}
printBestValue()
```

- done通常为
  - 达到一定的循环次数
  - 达到一定时间
  - fitness值不再增加
- analyzeValue通常记录下当前的值或仅记录下最佳值
- getNextValue是不同搜索算法的关键

随机搜索

- 最基本的搜索算法
- 随机产生下一个值

- 通常作为和其他搜索算法比较的基础
- 很多问题随机搜索已经能取得较好效果

元启发式搜索算法

- 采用问题相关的特定启发式规则进行搜索
  - 局部搜索
    - 爬山法
    - 模拟退火
  - 全局搜索
    - 粒子群算法
    - 遗传算法

爬山法

- 爬山法是最基本的启发式搜索算法
- 假设fitness是输入空间上的一个连续函数
- 下一个值为当前值的邻接值，直到达到局部最优点

爬山算法

vc ← 随机值

repeat

在vc的邻域中选择所有新点

计算新点的fitness，将最好的点记为vn

if fitness(vn) 好于 fitness(vc)

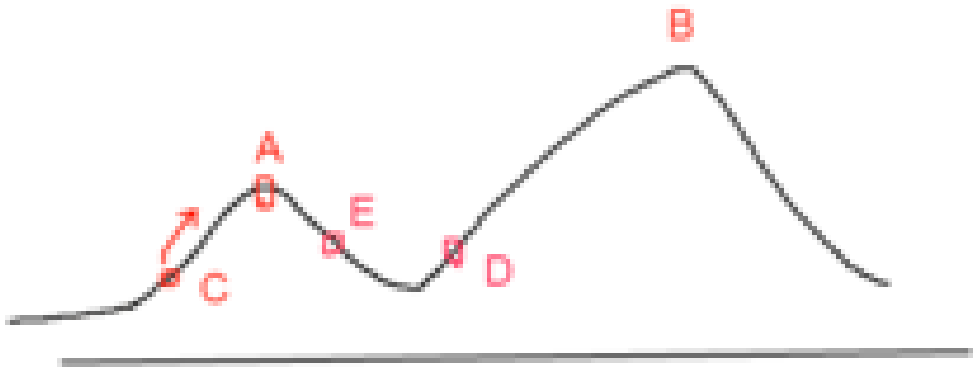
then vc ← vn

else localOptimal ← TRUE

until localOptimal

爬山法的缺点

- 只能达到局部最优，不能达到全局最优



模拟退火

- 除了上山也允许有一定几率下山

- 模拟退火的命名源自冶金学中的退火
  - 退火过快可能造成冶炼质量不好，所以要以适当速度退火
  - 模拟退火借用这个名字来说明搜索也不要收敛得太快

模拟退火算法

Let $s = s_0$

For k = 0 through k_{max} :

    $T \leftarrow k / k_{max}$

    Pick a random neighbour, $s_{new} \leftarrow$ neighbour(s)

    If $P(fitness(s), fitness(s_{new}), T) \geq$ random(0, 1)

        $s \leftarrow s_{new}$

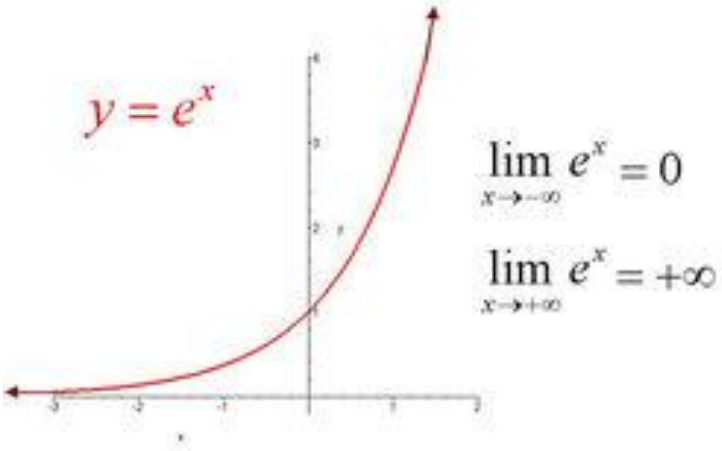
Output: the final state s

s_{new}越好，P越倾向于选择s_{new}

T越大，P越倾向于选择好的s_{new}

P的定义方法

- P通常按照Metropolis准则定义为
  - exp((*fitness*(s_{new}) - *fitness*(*s*))*T)
  - 求值大于等于1的时候概率为100%



exp函数

粒子群搜索

- 以上本地搜索算法一次只考虑一个候选值
- 全局搜索算法一次考虑多个候选值
- 粒子群算法
  - 粒子群算法(particle swarm optimization，PSO)，由Kennedy和Eberhart在1995年提出
  - 该算法模拟鸟群飞行觅食的行为，鸟之间通过集体的协作使群体达到最优目的，是一种基于Swarm Intelligence(群体智能)的优化方法。
  - 多次入选中科院和汤森路透的研究前沿

粒子群算法

- 假设目标空间是一个n维向量空间
- 空间中有若干粒子在飞行，每个粒子可以感知自己距离目标的距离（fitness）
- 粒子们不断根据自己发现的最佳位置和集体发现的最佳位置调整自己的行为

粒子群算法

- 每个结点保留两个参数
  - 结点位置z_i
  - 结点速度v_i
- 每次迭代根据下面的公式更新位置和速度
  - $v_i \leftarrow v_i + c_1 r_1 (p_i - z_i) + c_2 r_2 (p_g - z_i)$
  - $z_i \leftarrow z_i + v_i$
  - c_1, c_2为常数，r_1, r_2为随机值。
  - p_i为自己发现的最佳值，p_g为整体最佳值
- 迭代固定次数或者达到一定时间后结束

粒子群——最大速度

- 由于粒子群的飞行速度常常会过大，所以通常通过域值v_{max}来限制飞行速度

遗传算法

- 由美国密歇根大学的Holland于1975年首次提出
- 假设目标空间的每一个值是由一系列部件（基因）组成
- 整体fitness由每个部件的fitness和部件之间的局部组合决定
- 通过部件之间的组合（遗传）和演化（变异）来寻找新的值

- 例：用遗传算法解SAT问题
- 每个部件是对一个变量的赋值
- Fitness定义为当前满足的clause的数量

遗传算法

- 初始化：计算初代种群
  - 如：对所有变量随机赋true/false，生成n组
- 个体评价：计算适应度函数
  - 计算每组满足的clause数量
- 选择：选择可以繁殖的个体
  - 选择满足最多clause的n组
- 交叉：将若干个体的基因交换
  - 将n组两两组合，每组交换前一半变量的值
- 变异：将个体的基因改动
  - 每组随机翻转n个变量的值
- 下一代群体：取新值和老值的并集为下一代群体

编码

- 如何把目标空间编码成基因？
- 简单方法：根据二进制编码，每个二进制位是一个基因
  - 不能很好的表现整数的特征
- Gray Code：数值增加1，二进制位差别是1
- 最好针对特定问题设计编码方法

Decimal	Binary	Gray	Gray as Decimal
0	000	000	0
1	001	001	1
2	010	011	3
3	011	010	2
4	100	110	6
5	101	111	7
6	110	101	5
7	111	100	4

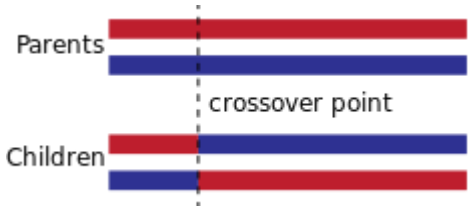
初始化

- 随机生成
- 等密度均匀生成

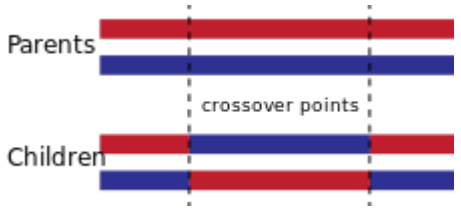
选择

- 基本方法：选择fitness最好的n个个体
  - 不利于物种多样性
- 轮盘法：每一轮，每个个体有$\frac{\text{该个体适应度}}{\text{总适应度}}$的概率被选择。重复n轮
  - 适应度差异较大（通常出现在前几轮），仍然不利于物种多样性
- 线性排序：先排序，个体被选择几率根据名次线性递减
- 竞技场选择：每次随机的从当前种群中挑选两个个体，以概率p选择较好个体
  - 避免排序的开销

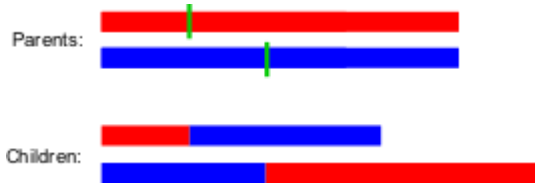
交叉



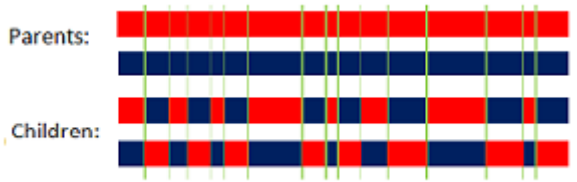
One-point crossover



Two-point crossover



Cut and splice
适用于个体长度不固定的情况



Uniform crossover
每个基因有50%的概率从父亲继承

通常需要针对问题设计交叉方法或者不使用交叉

变异

- 随机挑选n个基因，替换他们为新的随机基因
- 新的随机基因可以符合某种概率分布
  - 均匀分布
  - 正态分布
  - 非均匀分布——上一轮随机产生过的基因降低概率
- 非均匀变异
  - 一开始n值较大
  - 之后n值逐渐变小

Hyper Heuristic/Meta Optimization

- 问题1：这么多种搜索算法，怎么知道哪个比较好？
- 问题2：每个搜索算法都有这么多参数，应该如何设置？

- 把搜索算法和他们的参数作为目标空间
- 给定训练集，fitness为搜索算法限定时间内在训练集上找到的最好结果
- 利用搜索算法找出最好的搜索算法和参数
- 可以看做是一个机器学习的过程

启发式搜索vs随机搜索

- 启发式搜索不一定就比随机搜索更快
  - 随机搜索判断出当前解不如最优解的时候就抛弃当前解
  - 启发式搜索必须完成完整的fitness计算并选择下一个值
    - 如果这个过程耗时较长，那么启发式搜索验证的个体数可能显著少于随机搜索
- 案例：基于遗传算法的程序缺陷修复

搜索算法应用举例: 基于搜索的浮点测试输入生成

Daming Zou, Ran Wang, Yingfei Xiong, Lu Zhang, Zhendong Su, Hong Mei. A Genetic Algorithm for Detecting Significant Floating-Point Inaccuracies. ICSE 2015

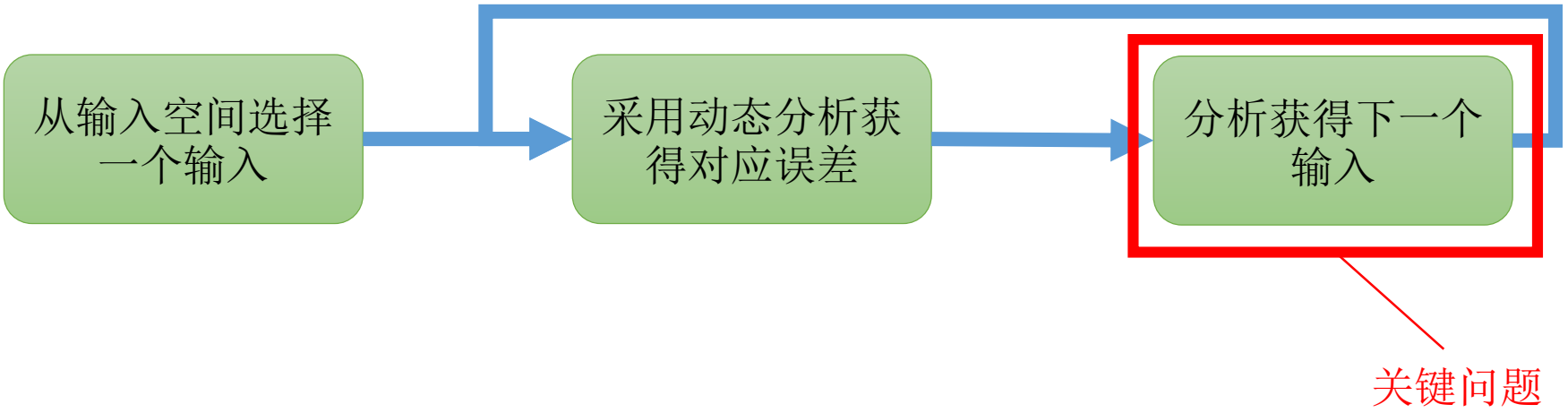
问题背景

- 浮点误差会导致严重的后果。
- 在海湾战争中，爱国者导弹由于累积的浮点误差拦截失败，造成了28人丧生。

已有方法

- 通过一次运行程序，可以知道该运行所产生的误差
- 如何知道一个程序有没有较大误差？

基于搜索的误差查找



实证研究

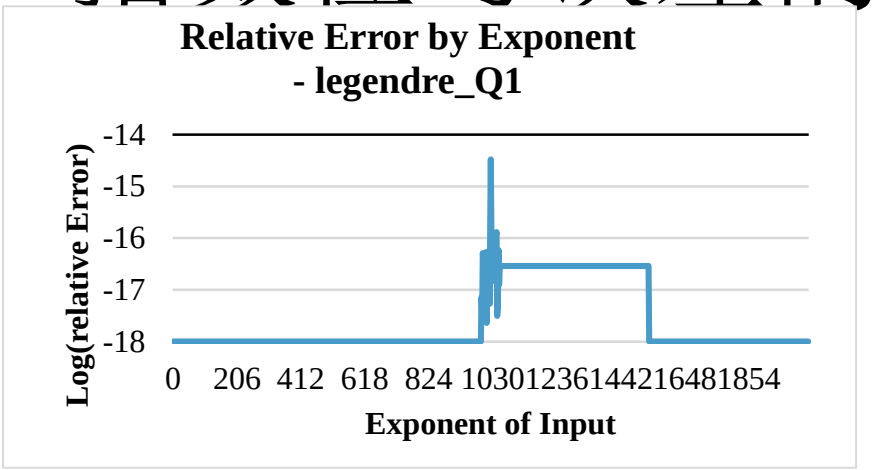
- 从GSL中选取4个函数，仅变化浮点数的尾数位或指数位，观察它们和内部误差之间的关系

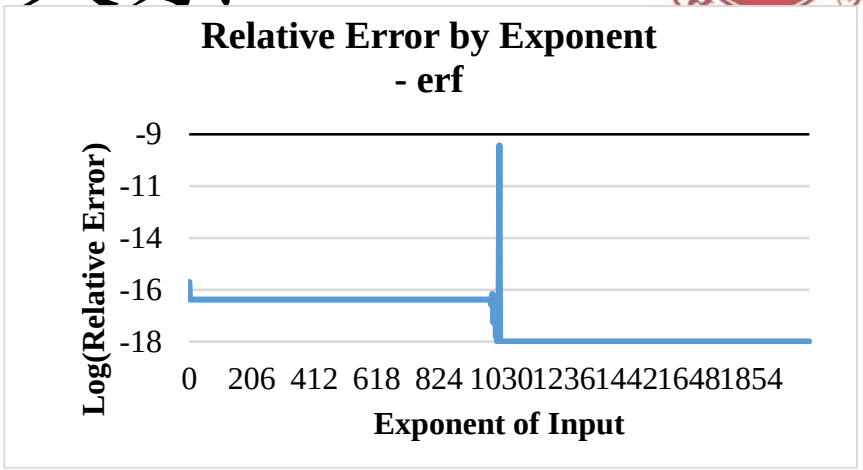
IEEE 754 Floating-Point Representation

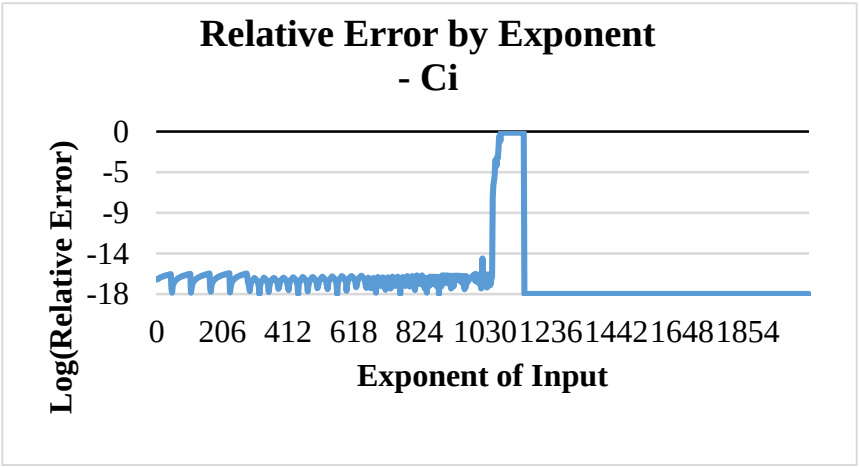
	Sign	Exponent	Significand
Single Precision	1	8	23
Double Precision	1	11	52

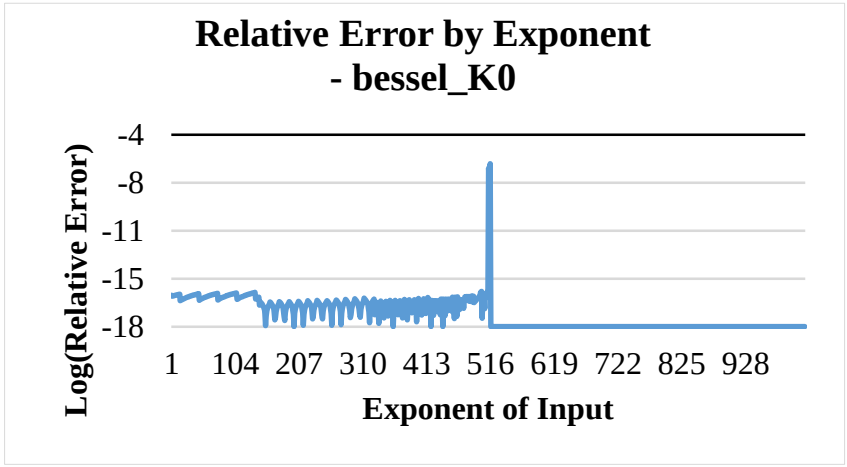
$$1.2345 = \underbrace{12345}_{\text{significand}} \times \underbrace{10}_{\text{base}}^{\overbrace{-4}^{\text{exponent}}}$$

指数位与误差的关系


Relative Error by Exponent
- legendre_Q1
Log(relative Error)
-14
-15
-16
-17
-18
0 206 412 618 824 1030 1236 1442 1648 1854
Exponent of Input


Relative Error by Exponent
- erf
Log(Relative Error)
-9
-11
-14
-16
-18
0 206 412 618 824 1030 1236 1442 1648 1854
Exponent of Input


Relative Error by Exponent
- Ci
Log(Relative Error)
0
-5
-9
-14
-18
0 206 412 618 824 1030 1236 1442 1648 1854
Exponent of Input


Relative Error by Exponent
- bessel_K0
Log(Relative Error)
-4
-8
-11
-15
-18
1 104 207 310 413 516 619 722 825 928
Exponent of Input

主要发现 1

指数部分对误差的大小有重大影响。

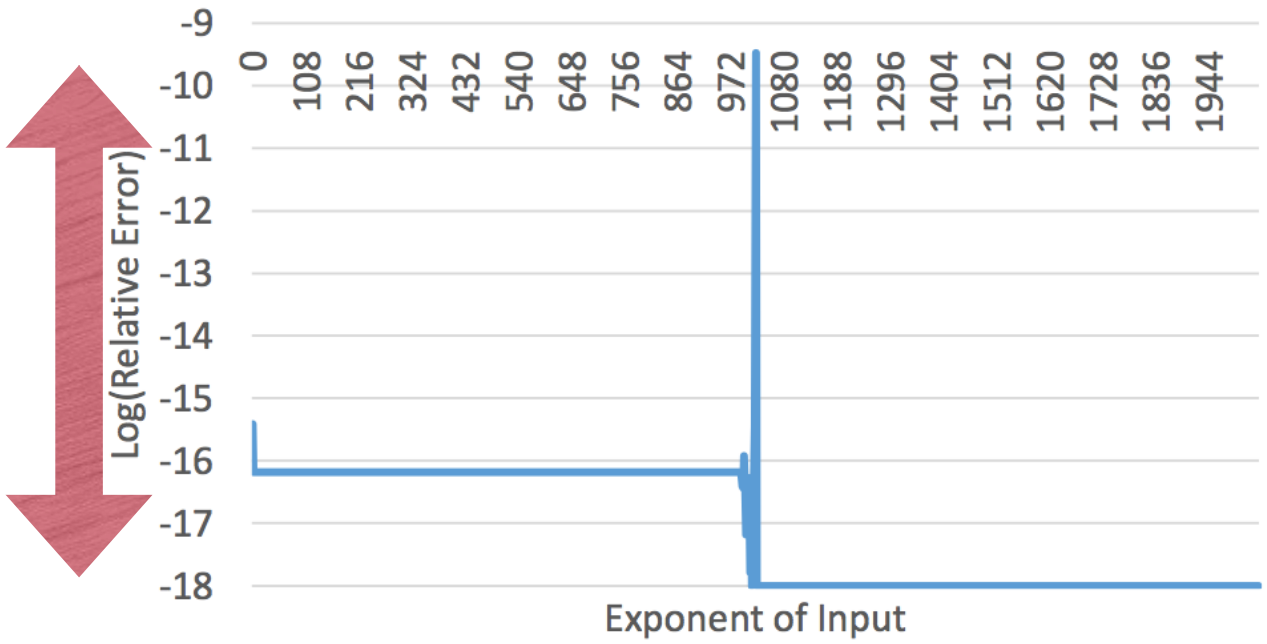


Fig. 1. erf at significand 0x34873b27b23c6

主要发现 2

能触发大误差的指数往往只存在于一个很小的范围。

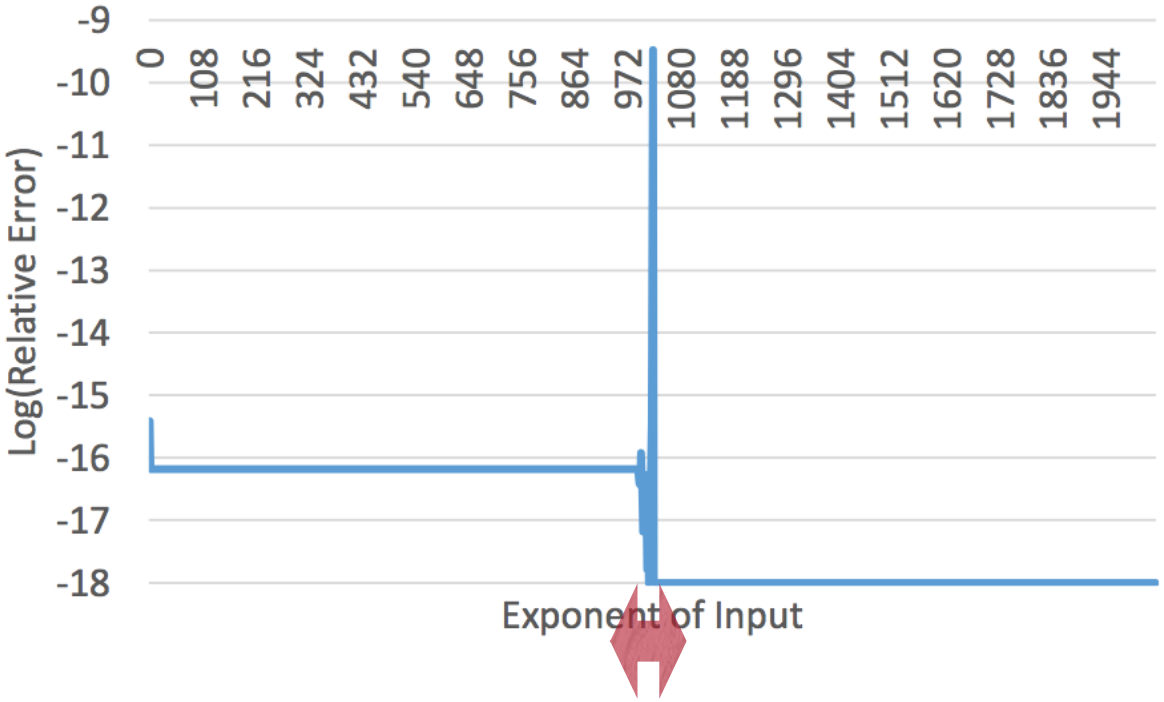


Fig. 1. erf at significand 0x34873b27b23c6

主要发现 3

在大误差的附近常常有高于平均误差的小波动。

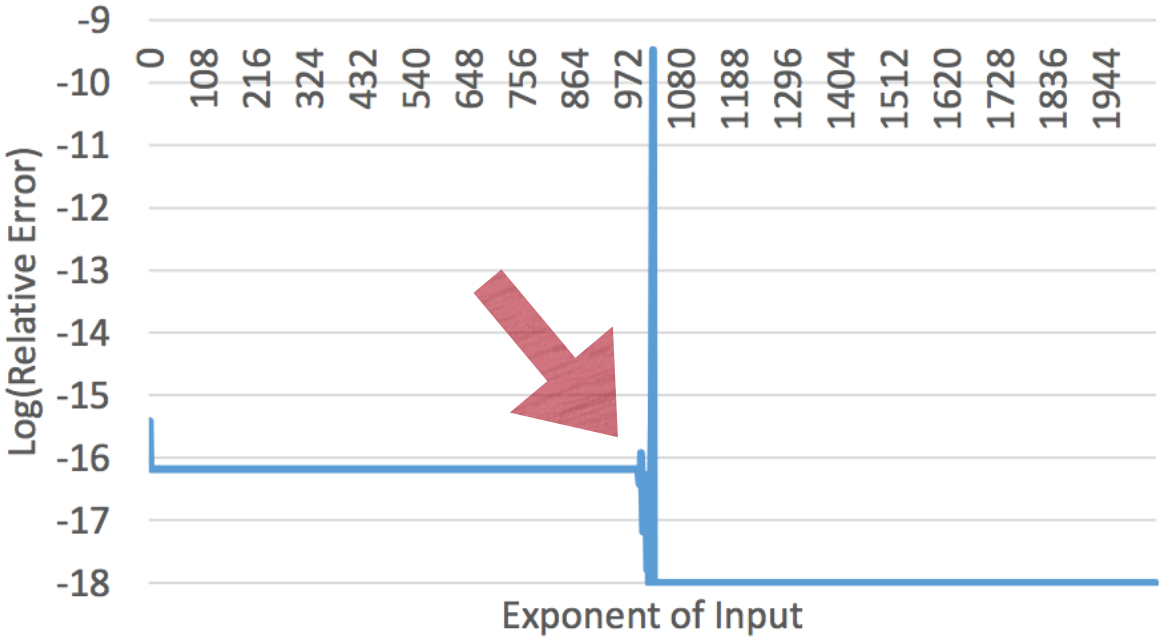


Fig. 1. erf at significand 0x34873b27b23c6

主要发现 4

大误差常常出现在浮点数中段位置。

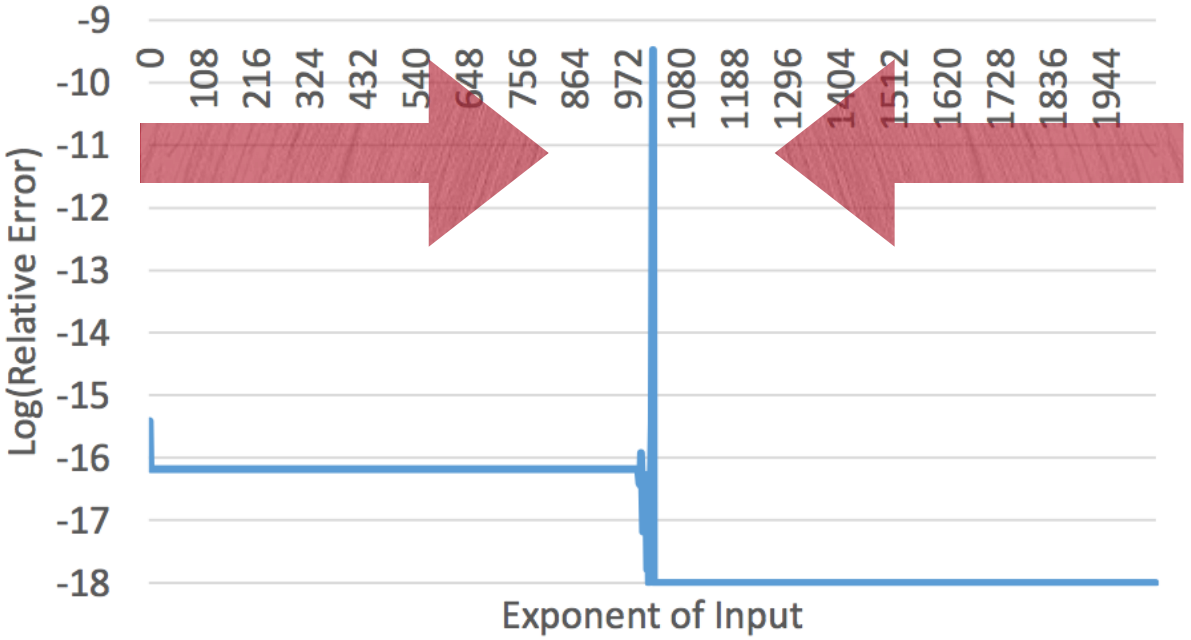


Fig. 1. erf at significand 0x34873b27b23c6

尾数位和浮点数之间的关系

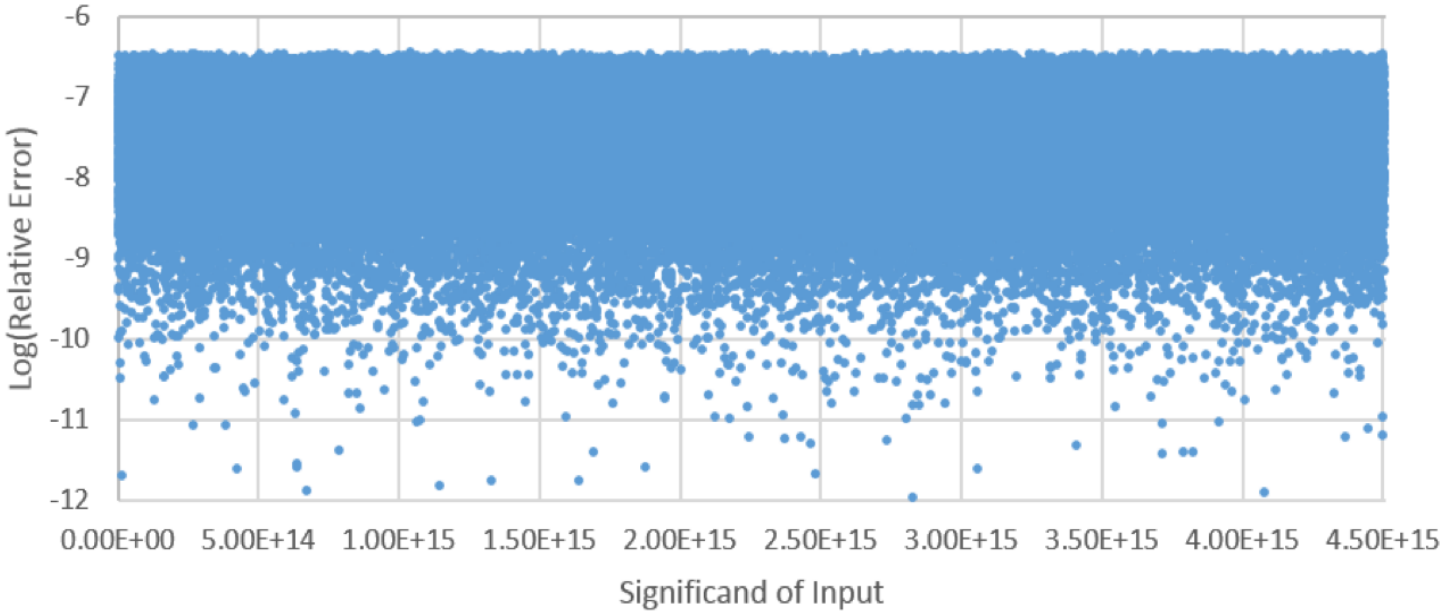


Fig. 2. erf at exponent 1023

主要发现 5

尾数位对浮点误差有显著影响。

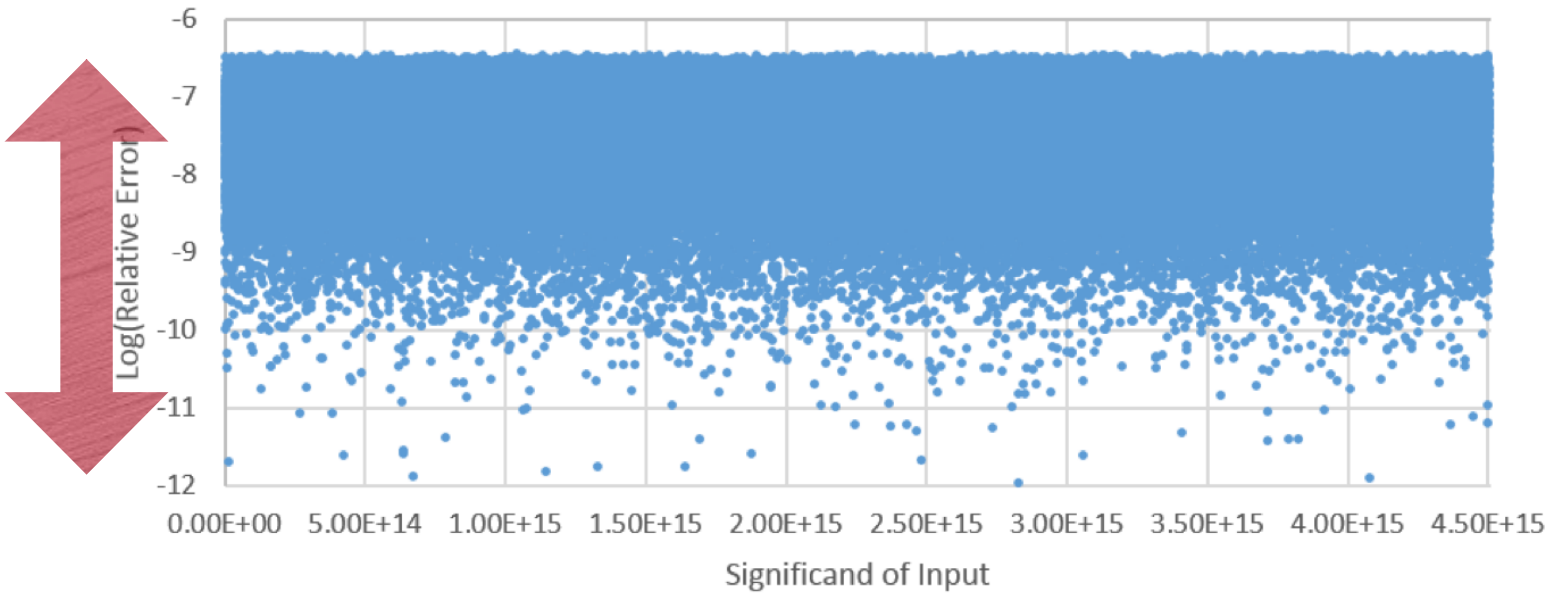


Fig. 2. erf at exponent 1023

主要发现 6

大量尾数都能触发大误差，并且在数轴上呈均匀分布。

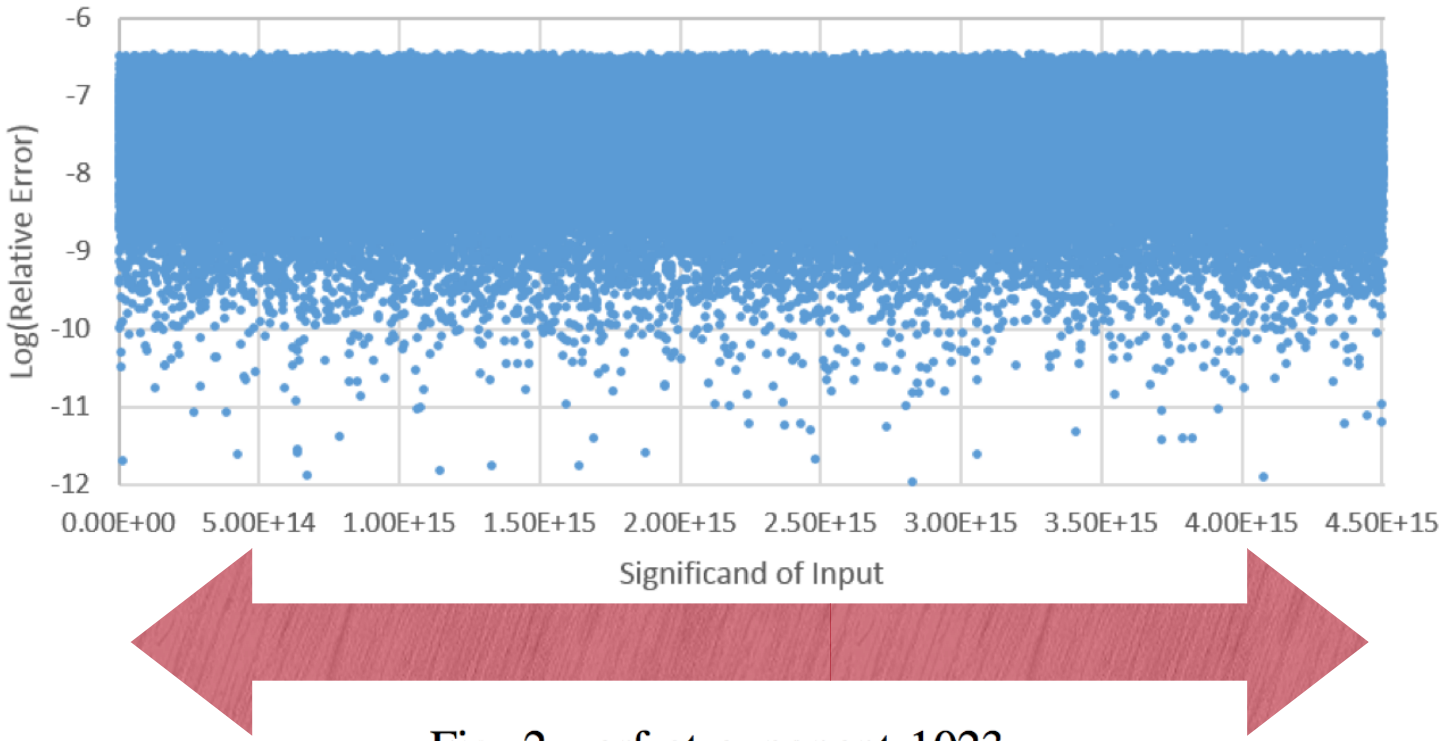


Fig. 2. erf at exponent 1023

局部敏感型遗传算法

- 根据以上特征，本研究设计了局部敏感型遗传算法。
- 对于指数位
  - 初始化时更多生成中位数附近。
  - 数值变异。
- 对于尾数位
  - 随机生成
  - 随机变异

实验评估

- 有效性检验：在6个经典浮点数程序上测试。
- 结果证明LSGA可以区分stable和unstable程序，初步证明了算法的可行性与有效性。

检验结果

	Newton	Inv	Root	Poly	Exp	Cos
Stable?	stable	stable	unstable	unstable	unstable	unstable
Max. Error Detected	2.8E-16	3.2E-16	8.1E+76	7.1E-11	1.0E+00	9.2E-01

实验评估

- 从GSL中选取154个函数。
- 与随机算法，标准遗传算法对比。

实验评估

检测到最大误差

Total	RAND	STD	LSGA	Tied
154	11 (7%)	24 (16%)	105 (68%)	14 (9%)

Sign Test 结果

	n_+	n_-	N	p
LSGA vs. RAND	127	12	139	< 4.14e-22
LSGA vs. STD	110	30	140	< 2.46e-11
STD vs. RAND	93	40	133	< 6.52e-06

实验评估

- 找到潜在内部精度问题的能力。
- 如何定义潜在问题？
  - 相对误差大于0.1%
  - 绝对误差大于预估绝对误差10倍以上

实验评估

TABLE VII
FUNCTIONS WITH POTENTIAL BUGS

Name	Relative Error	Estimated Absolute Error	Reported Absolute Error
airy_Ai_deriv	1.54E+06	1.04E-06	1.35E+00
airy_Ai_deriv_scaled	1.54E+06	1.04E-06	1.35E+00
clausen	5.52E-02	6.37E-17	2.31E-02
eta	9.58E+13	1.27E+37	2.71E+50
exprel_2	2.85E+00	4.44E-16	7.41E-01
gamma	1.07E-02	6.94E-14	1.05E-01
synchrotron_1	5.35E-03	4.47E-14	3.07E-04
synchrotron_2	3.67E-03	6.39E-14	1.86E-04
zeta	9.58E+13	3.41E+18	1.19E+32
zetam1	1.42E-02	1.51E+19	7.42E+30
bessel_Knu	6.08E-03	3.33E+22	9.05E+34
bessel_Knu_scaled	6.08E-03	2.66E+22	9.05E+34
beta	9.21E-03	4.91E-13	2.04E-01
ellint_E	8.92E-03	1.58E-16	3.14E-03
ellint_F	8.79E-03	1.86E-16	3.64E-03
gamma_inc_Q	1.36E+13	8.88E-16	1.25E-12
hyperg_0F1	5.80E+06	2.08E+37	7.33E+49
hyperg_2F0	4.35E-03	5.20E+02	3.19E+12