暨南大学本科实验报告专用纸

课程名称 运筹学 成绩评定
实验项目名称 求单峰函数最小值 指导老师 吴乐秦
实验项目编号 1 实验项目类型 设计性 实验地点 数学系机房
学生姓名 郭彦培 学号 2022101149
学院 信息科学技术学院 系 数学系 专业 信息管理与信息系统
实验时间 2024 年 3 月 21 日上午 ~3 月 21 日中午 温度 21℃湿度 85%

1. 实验目的

基于无导数二分法、黄金分割法和 Fibonacci 法实现对单谷函数的求解

2. 实验环境

计算机：PC X64 + Android aarch64

操作系统：Windows + termux

编程语言：python + C++

IDE：Visual Studio Code

3. 具体实现

实现一个函数，接受目标函数的函数指针，初始搜索区间，方法选择器，返回求得的单谷函数的最小值。

规定命名空间`lineSearch`内的函数原型

```
double find_mininum(
  double (*func)(double x),// inputed unimodal function
  double l,                // left range
  double r,                // right range
  double acc,              // Search accuracy
  int mod                  // Search mod
);
```

其中，`func`为目标单谷函数，`l`、`r`分别为初始区间的两个端点，`acc`为搜索精度，`mod`为模式选择。

关于模式选择，命名空间`lineSearch`内提供了三个可选模式：

`BINARY`	无导数二分法
`GOLDEN_RATIO`	黄金分割法
`FIBONACCI`	Fibonacci 法

当参数不合法时，程序会抛出异常，并返回固定值`-1`：

`Illegal Range Execption`	区间不合法
`Unexpection Search Mod Exception`	未知的搜索模式
`Unknown Exception`	其他预料外错误

4. 核心代码构成

完整代码见 7.附录

4.1. 无导数二分法

```
    double inf = acc /INF_ACC_RATIO,
           //infinitesimal unit 模拟求导极小值
            x; //search index 搜索目标值
    double cul = l,//current left range 当前搜索左端点
           cur = r;//current right range 当前搜索右端点
    while(cur - cul > acc){
        double mid = ( cul + cur ) /2.0;//计算中间值
        if(func(mid + inf) > func(mid - inf)) cur = mid;
        // 若极小值在左侧，则将右边界左移
        else cul = mid;
        // 反之则右移
        x = cul;
    }
    return x;
```

4.2. 黄金分割法

```
    double x;       //search index 搜索目标值
    double cul = l,
           //current left range 当前搜索左端点
           cur = r;
           //current right range 当前搜索右端点
    double otl = func(r - GOLDEN_RATIO_VALUE * (r - l)),
           //overture point left 左侧试探点函数值
           otr = func(l + GOLDEN_RATIO_VALUE * (r - l));
           //overture point right 右侧试探点函数值
    while(cur - cul > acc){
        if(otl > otr) //最小值在左侧区间
        {
            cul = cur - GOLDEN_RATIO_VALUE * (cur - cul);
            //将区间左端点移动到原左侧黄金分割点处
            otl = otr;
            //原右侧试探点为新区间左侧试探点
            otr = func(cul + GOLDEN_RATIO_VALUE * (cur - cul));
            //计算新的右侧试探点值
        } else {    //反之...
            cur = cul + GOLDEN_RATIO_VALUE * (cur - cul);
            otr = otl;
            otl = func(cur - GOLDEN_RATIO_VALUE * (cur - cul));
        }
        x = cul;
    }
    return x;
```

其中，`GOLDEN_RATIO_VALUE`为黄金分割比，其值为$\frac{\sqrt{5}-1}{2}$

4.3. Fibonacci 法

```
    double x;       //search index 搜索目标值
    double cul = l,
           //current left range 当前搜索左端点
           cur = r;
           //current right range 当前搜索右端点
    double FN = (r - l) / acc;
           //inneed fibonacci value 斐波那契数列最大值
    int N; //total caculate times 最大计算次数
    std::vector<double> Fib(2,1);
                        //Fibonacci array 斐波那契数列

    while(Fib[Fib.size()-1] < FN)//递推计算斐波那契数列
        Fib.push_back(Fib[Fib.size()-1] + Fib[Fib.size()-2]);
    N = Fib.size() -1;

    double otl = func(l + Fib[N-2] / Fib[N] * (r - l)),
           //overture point left 左侧试探点函数值
           otr = func(l + Fib[N-1] / Fib[N] * (r - l));
           //overture point right 右侧试探点函数值

    for(int k = 0;k <= N - 2.0 ;k ++)
    {
        if(otl > otr) //最小值在左侧
        {
            cul = cul + Fib[N -k -2] / Fib[N -k] * (cur -cul);
            //将区间左端点移动到原左侧试探点处
            otl = otr;
            //原右侧试探点为新区间左侧试探点
            otr = func(cul +Fib[N -k -1] /Fib[N -k] *(cur -cul));
            //计算新的右侧试探点值
        } else {//反之...
            cur = cul +Fib[N -k -1] /Fib[N -k] *(cur -cul);
            otr = otl;
            otl = func(cul +Fib[N -k -2] /Fib[N -k] *(cur -cul));
        }
        x = cul;
    }
    return x;
```

5. 正确性测试

完整测试代码见 7.附录

5.1. 测试数据准备

测试用的目标函数为一个在`x`轴平移了`dev`的二次函数，即:

```
double dev = 0.03; // deviation
double f(double a)
{
    return (a - dev) * (a - dev);
}
```

测试程序将随机生成一系列的偏移值`dev`，和对应的合法搜索区间`l`,`r`、准确度`acc`，并分别调用

```
lineSearch::find_mininum(f, l, r, acc, lineSearch::BINARY);
lineSearch::find_mininum(f, l, r, acc, lineSearch::GOLDEN_RATIO);
 lineSearch::find_mininum(f, l, r, acc, lin eSearch::FIBONACCI);
```

随后分析并输出结果。

规定理论值为`thn`，当前答案为`ans`

下面是 10 次测试的结果，其中当前精准度

$$\text{acc}_{\text{当前}} = \frac{\text{acc}}{|\text{thn} - \text{ans}|} * 100\% \tag{1}$$

反映了搜索的准确度。其中偏差量

$$\text{dev} = \frac{\max(0, |\text{thn} - \text{ans}| - \text{acc})}{\text{acc}} * 100\% \tag{2}$$

反应了搜索结果与目标的偏差是否在可接受范围内。

$\text{acc} > 100\%$且$\text{dev} = 0$时可以视为解是可接受的。

5.2. 测试结果：

测试次数取 5 时输出如下:

```
----Test Cases1----
< search data > l:-3.38 r:3.96 acc:1e-09
< Theoretical > ans:0.24 acc:inf
[Binary search] ans:0.24 acc:641.04 dev:0%
[0.618  method] ans:0.24 acc:665.21 dev:0%
[  Fibonacci  ] ans:0.24 acc:238.612 dev:0%

----Test Cases2----
< search data > l:-1.76 r:3.4 acc:1e-07
< Theoretical > ans:1.38 acc:inf
[Binary search] ans:1.38 acc:246.724 dev:0%
[0.618  method] ans:1.38 acc:247.304 dev:0%
[  Fibonacci  ] ans:1.38 acc:267.991 dev:0%

----Test Cases3----
< search data > l:-1.64 r:2.58 acc:0.0001
< Theoretical > ans:0.56 acc:inf
[Binary search] ans:0.559956 acc:225.986 dev:0%
[0.618  method] ans:0.559955 acc:221.363 dev:0%
[  Fibonacci  ] ans:0.559919 acc:123.319 dev:0%

----Test Cases4----
< search data > l:-1.12 r:3.38 acc:0.001
< Theoretical > ans:1.24 acc:inf
[Binary search] ans:1.23986 acc:731.429 dev:0%
[0.618  method] ans:1.23937 acc:159.73 dev:0%
[  Fibonacci  ] ans:1.23942 acc:173.462 dev:0%

----Test Cases5----
< search data > l:-2.5 r:2.86 acc:1e-05
< Theoretical > ans:0.74 acc:inf
[Binary search] ans:0.739996 acc:273.067 dev:0%
[0.618  method] ans:0.739998 acc:427.762 dev:0%
[  Fibonacci  ] ans:0.739996 acc:281.095 dev:0%
```

可以看到对于不同的参数，程序的`acc`与`dev`均在可接受范围内，因此可以认为搜索算法实现正确。

6. 各方法不同情况下的性能表现与分析

完整测试代码见 7.附录

6.1. 对于一般单谷函数，进行大范围区间搜索：

这一项测试针对大部分搜索场景，考验算法的区间下降速度。

测试函数：$f(x)=\begin{cases}\sqrt{x+1.03} & (x\geq -0.48)\\ \sqrt{0.07-x} & (x<-0.48)\end{cases}$

测试函数性质：在$\mathbb{R}$上为单谷函数，最小值在$x=-0.48$处取得，$f(0.48)\simeq 0.7416198463187$

测试内容：给定相同的测试参数，分别进行十次搜索，统计平均时间消耗。

测试参数：`l = -1e5,r = 1e5,acc = 0.00001`

搜索结果：

```
  [Binary search] ans:-0.480004 acc:273.601 dev:0%
  [0.618  method] ans:-0.480005 acc:221.192 dev:0%
  [  Fibonacci  ] ans:-0.48001 acc:104.605 dev:0%
```