

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 求二元函数极小值 指导老师 吴乐秦
实验项目编号 3 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 5 月 17 日上午 ~ 5 月 19 日晚上 温度 33℃ 湿度 95%

目录

1. 实验目的	2
2. 实验原理与理论分析	2
2.1. 最速下降法	2
2.2. 共轭梯度法	2
3. 代码框架	3
4. 核心代码构成	5
4.1. 最速下降法	5
4.2. 共轭梯度法	6
5. 正确性测试	8
5.1. 测试数据准备	8
5.2. 测试结果	8
6. 各方法不同情况下的性能表现与分析	10
6.1. 对于最速下降法的最坏情况:	10
6.2. 对于复杂目标函数进行搜索:	11
7. 附录	11
7.1. 代码	11
7.2. 仓库	11

暨南大学本科实验报告专用纸(附页)

1. 实验目的

实现利用梯度法求解二元函数最小值的函数，并对比不同方法间、同方法内不同推导式之间的性能差异。

2. 实验原理与理论分析

本次实验选用最速下降法和共轭梯度法

2.1. 最速下降法

对于当前搜索点 x_k ，有梯度 $d_k = -\nabla f(x_k)$ 。利用一维搜索取合适的步长因子 α_k s.t. $f(x_k + \alpha_k d_k) < f(x_k)$ 则

$$x_{k+1} = x_k + \alpha_k d_k \quad (1)$$

2.2. 共轭梯度法

一般地，在第 k 次迭代，令

$$d_k = -g_k + \sum_{i=0}^{k-1} \beta_i d_i \quad (2)$$

$$d_{k+1} = -g_{k+1} + \beta_k d_k$$

，则选择 β_i s.t. $d_k^T G d_i = 0$ 则有不同的 β_k 推导式：

2.2.1. Fletcher-Reeves (FR)公式

$$\beta_k = \frac{g_k^T g_k}{g_{k-1}^T g_{k-1}} \quad (3)$$

2.2.2. Polak-Ribiere-Polyak (PRP)公式

$$\beta_k = \frac{g_{k+1}^T (g_{k+1} - g_k)}{g_k^T g_k} \quad (4)$$

3. 代码框架

编码利用 C++ 完成，遵循 C++17 标准

规定命名空间 `lineSearch` 内的函数原型

```
std::pair<Corrdinate,double> find_mininum(  
    double (*func)(Corrdinate),      //目标函数  
    Corrdinate (*dfunc)(Corrdinate), //目标函数梯度  
    Corrdinate x_0,                   //初始搜索点  
    int mod = GD,                     //搜索模式  
    double epsilon = _epsilon        //容限  
)
```

其中:

参数	用途	默认值
<code>func</code>	目标优化函数	无，必须提供
<code>dfunc</code>	目标函数一阶导	无，必须提供
<code>x_0</code>	初始搜索点	(0,0)
<code>mod</code>	搜索模式	DESCENT（最速下降法）
<code>epsilon</code>	容限	10^{-3}

返回值为一个 `std::pair<double,double>` 类型对象，分别存储了搜索到的 x_{km} 与对应的最小函数值 f_{\min}

其中，内建类库 `Corrdinate` 提供了坐标向量相关的运算成员函数、进行了运算符重载，并对形如 `Corrdinate x_0 = {1,1}` 的列表初始化提供了支持。

关于模式选择，命名空间 `ODSearch` 内提供了三个可选模式：

GD	最速下降法
CG	割线法

暨南大学本科实验报告专用纸(附页)

当参数不合法或执行出错时，程序会抛出异常。若无法继续计算，则返回固定值 -1：

异常	why?	类型
Epsilon out of Precision Exception	给定容限精度溢出	错误
Coordinate out of Precision Warning	搜索坐标精度溢出	警告
Unexpection Search Mod Exception	未知的搜索模式	错误
Unknown Exception	其他预料外错误	错误

以下是一些函数调用例子：

```
pair<Corrdinate, double> ans =  
ODSearch::find_mininum(f, df, {0, 0}, ODSearch::GD, 0.0019);  
//用最速下降法从点(0,0)搜索函数 f 的最小值，精度为 0.0019  
Corrdinate ans =  
lineSearch::find_mininum(f, df, {0, 0}, ODSearch::CG, 0.0019).first;  
//用共轭梯度法从点(0,0)搜索函数 f 的最小值，精度为 0.0019, 返回搜索到的 x
```

以下是 Corrdinate 库使用的例子：

```
Corrdinate a = {1, 1};  
Corrdinate b = {2, 2};  
Corrdinate c = a + b;  
//c = {3, 3}  
double d = a * b;  
//d = 1*2 + 1*2 = 4  
  
Coordinate d = a * 2;  
//d = {2, 2}  
  
Coordinate e = a / 2;  
//e = {0.5, 0.5}
```

使用 setFRorPRP 设置共轭梯度法的推导式，setFRorPRP(1) 为 FR，setFRorPRP(0) 为 PRP。

4. 核心代码构成

完整代码见 7.附录

4.1. 最速下降法

```
int k = 0; //迭代次数
double alpha = 0.1; //初始步长因子
Coordinate curx = x_0; //当前搜索点
double fmin = func(x_0); //当前函数值最小值
Coordinate grad = dfunc(x_0); //当前梯度

// int tc = 0;
while(grad.norm() > epsilon)
{
    if(k > 10 && (curx.norm() < 1e-20 || curx.norm() > 1e20))
    {throw "Coordinate out of Precision Warning";}

    //二分线性搜索确定可选步长因子
    while(!(func(curx - grad * alpha) < func(curx)))
        alpha = alpha / 2.0;
    fmin = func(curx - grad * alpha);
    curx -= grad * alpha;
    grad = dfunc(curx);
    alpha = 0.1;
    k ++;
    // tc ++;
}
// std::cout << "tc:" << tc << "\n";
return {curx, fmin};
```

4.2. 共轭梯度法

```
int k = 0; //迭代次数
double alpha = 0.1; //初始步长因子
Corrdinate curx = x_0; //当前搜索点
double fmin = func(x_0); //当前函数值最小值
Corrdinate grad_k = dfunc(x_0); //当前梯度
Corrdinate grad_k_1 = grad_k; //上一次梯度
Corrdinate d_k = -grad_k; //搜索方向
Corrdinate d_k_1 = d_k; //上一次搜索方向

while(grad_k.norm() > epsilon)
{
    if(k > 10 && (curx.norm() < 1e-20 || curx.norm() > 1e20))
    {throw "Coordinate out of Precision Warning";}

    if(k == 0)
    {
        //二分线性搜索确定可选步长因子
        while(!(func(curx + d_k * alpha) < func(curx)))
            alpha = alpha / 2.0;
        fmin = func(curx + d_k * alpha);
        curx += d_k * alpha;
        grad_k_1 = grad_k;
        grad_k = dfunc(curx);
        d_k = -grad_k;
        alpha = 0.1;
    }
    else
    {
        if (FRorPRP == 1)
        {
            //FR 公式
            double beta = (grad_k * grad_k) / (grad_k_1 *
grad_k_1);
            d_k = -grad_k + d_k * beta;
        }
        else
        {
            //PRP 公式
            double beta = (grad_k * (grad_k - grad_k_1)) /
```

暨南大学本科实验报告专用纸(附页)

```
(grad_k_1 * grad_k_1);
    d_k = -grad_k + d_k * beta;
}

//二分线性搜索确定可选步长因子
while(!(func(curx + d_k * alpha) < func(curx)))
    alpha = alpha / 2.0;
fmin = func(curx + d_k * alpha);
curx += d_k * alpha;
grad_k_1 = grad_k;
grad_k = dfunc(curx);
d_k = -grad_k;
alpha = 0.1;
}
k ++;
}
return {curx,fmin};
```

5. 正确性测试

见附录 T0Ftest.cpp

5.1. 测试数据准备

测试用的目标函数为一个最小值在 xOy 上的点 (dev, dev) 的二次函数, 即

```
double f(Corrdinate x)
{
    return (x.x - dev.x) * (x.x - dev.x) + (x.y - dev.y) * (x.y - dev.y);
}
```

测试程序将随机生成一系列的偏移值 `dev` 和容限 `eps`, 并分别调用

```
ODSearch::find_mininum(f, df, {0.0, 0.0}, ODSearch::GD, eps)
ODSearch::find_mininum(f, df, {0.0, 0.0}, ODSearch::CG, eps) //
FRorPRP = 1
ODSearch::find_mininum(f, df, {0.0, 0.0}, ODSearch::CG, eps) //
FRorPRP = 0
```

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

$$acc = \frac{eps}{|\nabla f(thn) - \nabla f(ans)|} \times 100\% \quad (5)$$

反映了搜索的准确度。其中偏差量

$$dev = \frac{\max(0, |\nabla f(thn) - \nabla f(ans)| - eps)}{eps} \times 100\% \quad (6)$$

反应了搜索结果与目标的偏差是否在可接受范围内。

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

5.2. 测试结果

以 1145 为 STL 随机数生成器种子进行了 100 次测试, 结果全部正确。

暨南大学本科实验报告专用纸(附页)

以下是前 4 次测试的结果:

```
----Test Cases1----
<search data> eps:1e-08
<Theoretical> ans:(0.24 1.62) acc:inf
[   G D   ] ans:2.35666e-17 df(ans)9.70908e-09 df(thn):0 at:
(0.24 1.62) acc:102.996 dev:0%
[  C G(FR) ] ans:1.87195e-17 df(ans)8.65321e-09 df(thn):0 at:
(0.24 1.62) acc:115.564 dev:0%
[  C G(PBD) ] ans:1.71726e-17 df(ans)8.28796e-09 df(thn):0 at:
(0.24 1.62) acc:120.657 dev:0%

----Test Cases2----
<search data> eps:1e-06
<Theoretical> ans:(1.82 1.38) acc:inf
[   G D   ] ans:2.2071e-13 df(ans)9.39596e-07 df(thn):0 at:
(1.82 1.38) acc:106.429 dev:0%
[  C G(FR) ] ans:1.80157e-13 df(ans)8.48899e-07 df(thn):0 at:
(1.82 1.38) acc:117.8 dev:0%
[  C G(PBD) ] ans:1.92346e-13 df(ans)8.77145e-07 df(thn):0 at:
(1.82 1.38) acc:114.006 dev:0%

----Test Cases3----
<search data> eps:0.001
<Theoretical> ans:(0.04 1.4) acc:inf
[   G D   ] ans:2.06581e-07 df(ans)0.000909023 df(thn):0 at:
(0.039987 1.39955) acc:110.008 dev:0%
[  C G(FR) ] ans:1.97734e-07 df(ans)0.000889346 df(thn):0 at:
(0.0399873 1.39956) acc:112.442 dev:0%
[  C G(PBD) ] ans:1.7197e-07 df(ans)0.000829386 df(thn):0 at:
(0.0399882 1.39959) acc:120.571 dev:0%

----Test Cases4----
<search data> eps:0.0001
<Theoretical> ans:(0.22 0.04) acc:inf
[   G D   ] ans:2.1568e-09 df(ans)9.28826e-05 df(thn):0 at:
(0.219954 0.0399917) acc:107.663 dev:0%
[  C G(FR) ] ans:2.48069e-09 df(ans)9.96131e-05 df(thn):0 at:
(0.219951 0.0399911) acc:100.388 dev:0%
[  C G(PBD) ] ans:2.06457e-09 df(ans)9.08751e-05 df(thn):0 at:
(0.219955 0.0399919) acc:110.041 dev:0%
```

程序均正确地找到了极小值点, 且精度符合要求

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. 对于最速下降法的最坏情况：

见附录 LGNtest.cpp

6.1.1. 测试过程：

6.1.2. 测试分析：

暨南大学本科实验报告专用纸(附页)

6.2. 对于复杂目标函数进行搜索：

见附录 `CMFtest.cpp`

6.2.1. 测试过程：

6.2.2. 测试分析：

7. 附录

7.1. 代码

7.1.1. 核心 `core.h`

7.1.2. 测试代码

7.1.2.1. `TOFtest.cpp`

7.1.2.2. `CMFtest.cpp`

7.1.2.3. `LGNtest.cpp`

7.2. 仓库

全部代码、与 x86 可执行程序均同步在本人的 `github`：

`https://github.com/GYPpro/optimizeLec`

本次实验报告存放在 `/WEE2` 文件夹下

声明：本实验报告所有代码与测试均由本人独立完成，修改和 `commit` 记录均在 `repo` 上公开。