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

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$\alpha_k s.t. f(x_k + \alpha_k d_k) < f(x_k)$ 则

$$x_{k+1} = x_k + \alpha_k d_k \tag{1}$$

2.2. 共轭梯度法

一般地，在第k次迭代，令

$$\begin{aligned} d_k &= -g_k + \sum_{i=0}^{k-1} \beta_i d_i \\ d_{k+1} &= -g_{k+1} + \beta_k d_k \end{aligned} \tag{2}$$

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

2.2.1. Fletcher-Reeves (FR)公式

$$\beta_k = \frac{g_k^T g_k}{g_{k-1}^T g_{k-1}} \tag{3}$$

2.2.2. Polak-Ribiere-Polyak (PRP)公式

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

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
`func`	目标优化函数	无，必须提供
`dfunc`	目标函数一阶导	无，必须提供
`x_0`	初始搜索点	(0,0)
`mod`	搜索模式	`DESCENT`（最速下降法）
`epsilon`	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

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`。

函数在宏中定义了分析用的`Log`输出，通过`#define IF_LOG`来开启。若开启，则会在调用时分别在文件`GD.log`、`CG_FR.log`和`CG_PRP.log`中记录搜索过程。

报告后续的绘图流程均借助这些 log 完成，不再赘述。

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 = 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 = dfunc(x_0); // 当前梯度

#ifdef IF_LOG
      Logger logger("WEEK3\\GD.log");
      std::string Log;
      Log += std::to_string(curx.x);
      Log += " ";
      Log += std::to_string(curx.y);
      logger.log(Log);
#endif

      // 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 = 1;
        k++;
// tc ++;
#ifdef IF_LOG
        std::string Log;
        Log += std::to_string(curx.x);
        Log += " ";
```

```
        Log += std::to_string(curx.y);
        logger.log(Log);
#endif
      }
      // std::cout << "tc:" << tc << "\n";
      return {curx, fmin};
```

4.2. 共轭梯度法

```
      int k = 0;              // 迭代次数
      double alpha = 3;         // 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
#ifdef IF_LOG
      Logger logger(std::string("WEEK3\\CG_") + (FRorPRP ? "FR" :
"PRP") + ".log");
      std::string Log;
      Log += std::to_string(curx.x);
      Log += " ";
      Log += std::to_string(curx.y);
      logger.log(Log);
#endif
      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 = 3;
          }
          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)) /
(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 = 3;
          }
          k++;
#ifdef IF_LOG
          std::string Log;
          Log += std::to_string(curx.x);
          Log += " ";
          Log += std::to_string(curx.y);
          logger.log(Log);
#endif
        }
        return {curx, fmin};
```

5. 正确性测试

见附录 `TOFtest.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

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

反映了搜索的准确度。其中偏差量

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

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

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**。

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

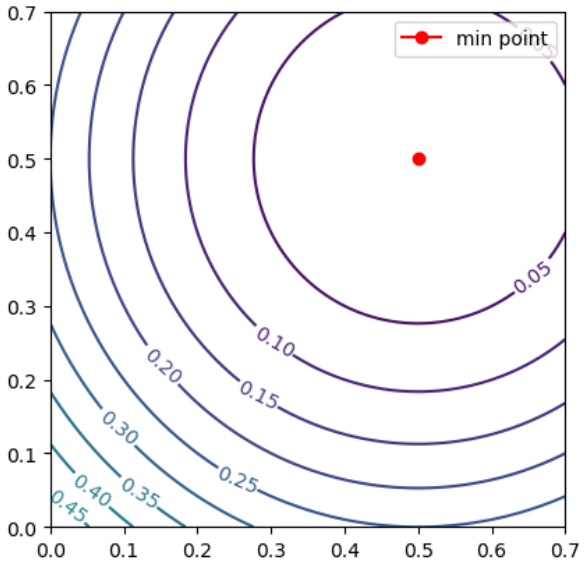
6. 各方法不同情况下的性能表现与分析

6.1. 对于最速下降法的最坏情况:

见附录 `LGNtest.cpp` 和 `draft.ipynb`

6.1.1. 测试用例:

构造测试函数$f(x,y)=x^2+\frac{y^2}{2}$，其$x\in[0,0.7],y\in[0,0.7]$范围内等值线如图:



6.1.2. 测试过程:

设置容限为10^{-5}，从点$(0,0)$开始搜索:

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
```

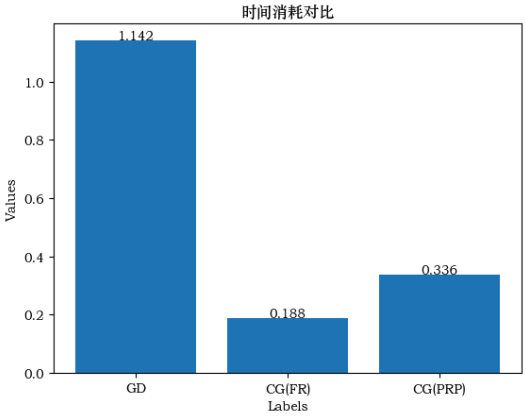
确认得到正确结果:

```
    <search data> eps:1e-05
    <Theoretical> ans:(0.5 0.5) acc:inf
    [    G D    ] ans:4.6867e-11 df(ans)9.68163e-06 df(thn):0 at:
(0.5 0.49999) acc:103.288 dev:0%
    [  C G(FR)  ] ans:2.42193e-11 df(ans)6.95978e-06 df(thn):0
```

```
at:(0.5 0.499993) acc:143.683 dev:0%
    [  C G(PBD) ] ans:3.83661e-11 df(ans)8.75969e-06 df(thn):0
at:(0.5 0.499991) acc:114.159 dev:0%
```

重复10^5次，统计三种方法代码消耗的时间：

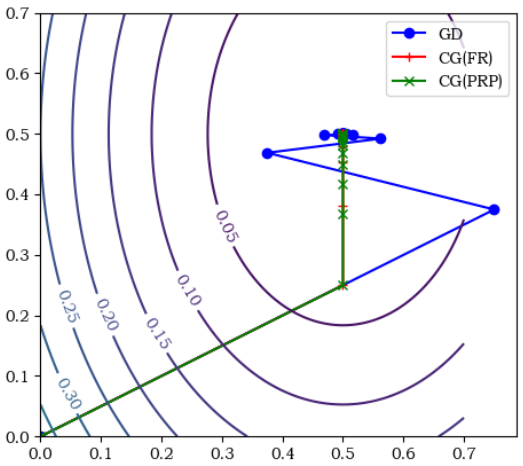
```
[    G D    ] cost:1.142s

[  C G(FR)  ] cost:0.188s

[  C G(PBD) ] cost:0.336s
```

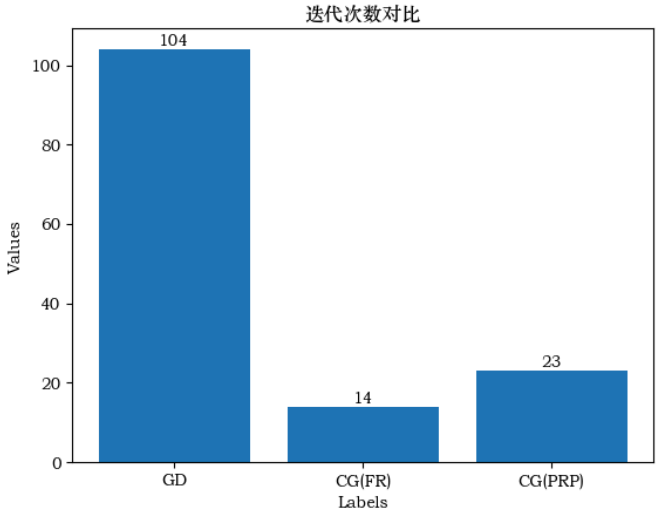


6.1.3. 测试结果：

三种方法均能找到正确的极小值点，且精度符合要求。

分析 log，可以对比三种方法使用的搜索次数和他们搜索过程的轨迹：





6.1.4. 测试分析：

从图中可以看出，最速下降法在搜索过程中不能有效地找到合适的搜索步长和下降方向，导致搜索效率较差。而共轭梯度法在搜索过程中能够更快地找到合适的搜索方向，从而更快地找到极小值点。

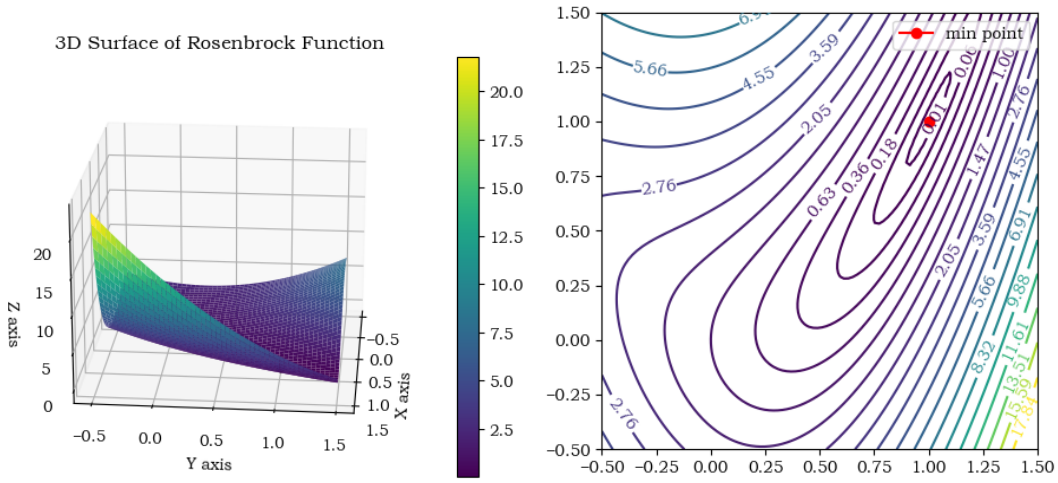
6.2. 对于一般目标函数进行搜索:

见附录`CMFtest.cpp`

6.2.1. 测试用例:

$$f(x,y) = (1-x)^2 + 3(y-x^2)^2 \tag{7}$$

其在全局范围内有最小值点$(1,1)$，$f(1,1)=0$。其 3D 图像与等值线如下:



6.2.2. 测试过程:

设置容限为10^{-5}，从点$(0,0)$开始搜索： 分别调用

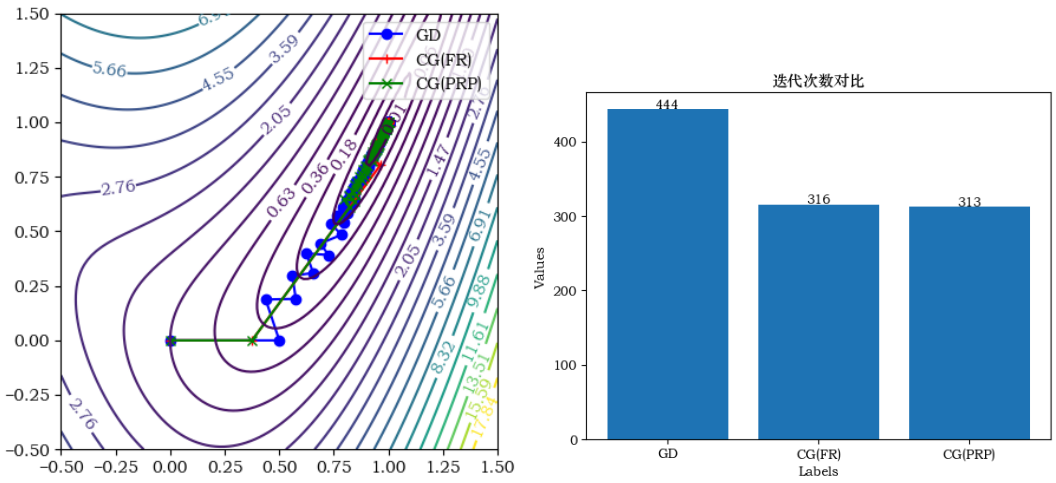
```
ODSearch::find_mininum(f, df,{0.0,0.0}, ODSearch::GD,eps)
ODSearch::find_mininum(f, df,{0.0,0.0}, ODSearch::CG,eps)//
FRorPRP = 1
ODSearch::find_mininum(f, df,{0.0,0.0}, ODSearch::CG,eps)//
FRorPRP = 0
```

确认得到正确结果:

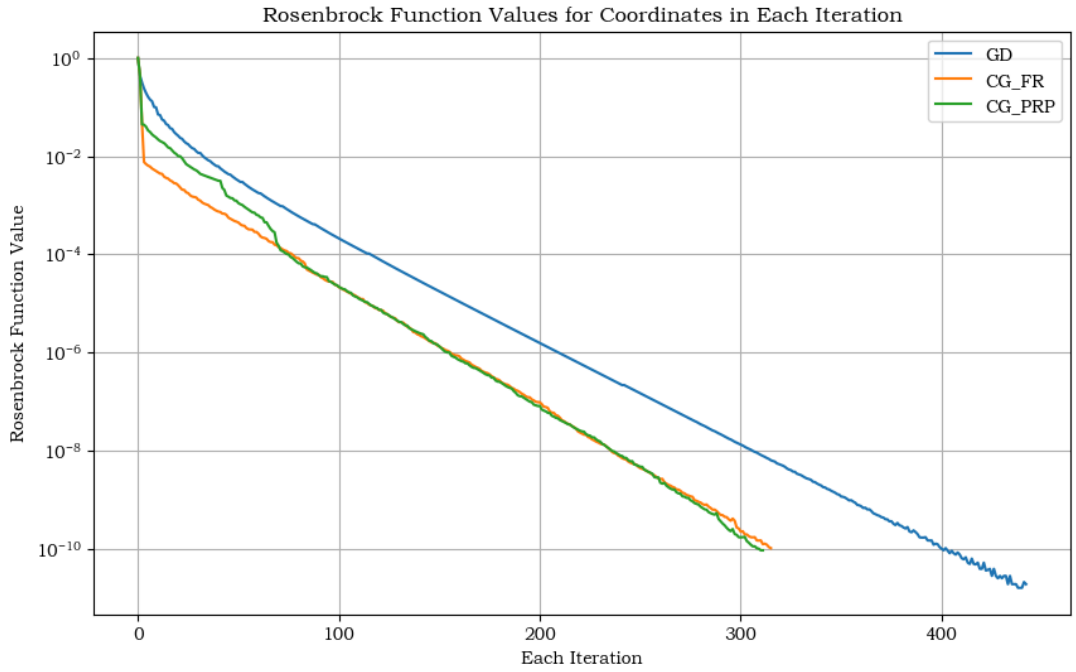
```
  <search data> eps:1e-05
  <Theoretical> ans:(1 1) acc:inf
  [    G D    ] ans:6.05069e-11 df(ans)9.58326e-06 df(thn):0 at:
(0.999993 0.999984) acc:104.349 dev:0%
  [  C G(FR)  ] ans:5.24562e-11 df(ans)8.08791e-06 df(thn):0 at:
(0.999993 0.999985) acc:123.641 dev:0%
  [  C G(PBD) ] ans:8.65653e-11 df(ans)9.57498e-06 df(thn):0 at:
(0.999991 0.999981) acc:104.439 dev:0%
```

6.2.3. 测试结果

分析 log，可以对比三种方法使用的搜索次数和他们搜索过程的轨迹：



以对数坐标画出三种方法的收敛曲线：



6.2.4. 测试分析：

从结果来看，最速下降法的折线特征明显，所求下降方向有明显偏差，而两种梯度法均解决了这个问题。

对于两种不同的梯度法公式，其速度表现相似。从图中可以看出，FR 公式在末端搜索的效率较高，而 PRP 公式在搜索初期效率较高。

7. 附录

7.1. 代码

7.1.1. 核心`core.h`

```
/**
 * @author
 * JNU,Guo Yanpei,github@GYPpro
 * https://github.com/GYPpro/optimizeLec
 * @file
 * /optimizeLec/WEEK2/core.h
 * @brief
 * a functional lib solving One-dimensional search
 */

#ifndef _ONE_DIMENSIONAL_SEARCH_
#define _ONE_DIMENSIONAL_SEARCH_

#include <math.h>
#include <algorithm>
#include <vector>
#include <string>
#include <iostream>
#include <initializer_list>

#define IF_LOG

#include <fstream>

class Logger
{
private:
  std::ofstream logFile;

public:
  Logger(const std::string &filename)
  {
    logFile.open(filename, std::ios::out); // 打开文件用于写入,
fugai
    if (!logFile.is_open())
    {
      std::cerr << "Error opening log file: " << filename <<
```

```
std::endl;
      exit(EXIT_FAILURE);
    }
  }

  ~Logger()
  {
    if (logFile.is_open())
    {
      logFile.close();
    }
  }

  void log(const std::string &message)
  {
    // 输出到控制台
    // std::cout << message << std::endl;

    // 写入到日志文件
#ifdef IF_LOG
    logFile << message << std::endl;
#endif
  }
};

namespace ODSearch
{

  const int GD = 1; // 最速下降法
  const int CG = 2; // 共轭梯度法

  const double _epsilon = 1e-3; // 默认容限

  int FRorPRP = 1; // 共轭梯度方向公式选择，1 为 FR，0 为 PBD

  /**
   * @brief
   * a Corrdinate class as 2D vector
   */
  class Corrdinate
  {
  public:
```

```
    double x;
    double y;

    Corrdinate(double x = 0, double y = 0) : x(x), y(y) {}

    Corrdinate(std::initializer_list<double> l)
    {
      auto it = l.begin();
      x = *it;
      y = *(++it);
    }

    Corrdinate operator-(Corrdinate c)
    {
      return Corrdinate(x - c.x, y - c.y);
    }

    Corrdinate operator-()
    {
      return Corrdinate(-x, -y);
    }

    // 数乘
    Corrdinate operator*(double a)
    {
      return Corrdinate(x * a, y * a);
    }

    // 加
    Corrdinate operator+(Corrdinate c)
    {
      return Corrdinate(x + c.x, y + c.y);
    }

    // 点乘
    double operator*(Corrdinate c)
    {
      return x * c.x + y * c.y;
    }

    // 求模
    double norm()
```

```
{
  return sqrt(x * x + y * y);
}

// 单位化
Corrdinate normalize()
{
  double n = norm();
  return Corrdinate(x / n, y / n);
}

// 除
Corrdinate operator/(double a)
{
  return Corrdinate(x / a, y / a);
}

Corrdinate operator+=(const Corrdinate c)
{
  x += c.x;
  y += c.y;
  return *this;
}

Corrdinate operator-=(const Corrdinate c)
{
  x -= c.x;
  y -= c.y;
  return *this;
}

Corrdinate operator*=(double a)
{
  x *= a;
  y *= a;
  return *this;
}

Corrdinate operator/=(double a)
{
  x /= a;
  y /= a;
```

```
      return *this;
    }
  };

  /**
   * @brief
   * set the FRorPRP
   */
  void setFRorPRP(int mod)
  {
    FRorPRP = mod;
  }

  double abs(Corrdinate c)
  {
    return sqrt(c.x * c.x + c.y * c.y);
  }

  /**
   * @attention
   * function will return -1 and throw exceptions while getting
illegal input
   * @brief
   * Finding the minnum num of a One-dimensional function
   */
  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
  )
  {
    if (epsilon <= 1e-14)
    {
      throw "Epsilon out of Precision Exception";
      return {{0, 0}, -1};
    }
    switch (mod)
    {
    case GD:
    {
```

```
      int k = 0;              // 迭代次数
      double alpha = 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 = dfunc(x_0); // 当前梯度

#ifdef IF_LOG
      Logger logger("WEEK3\\GD.log");
      std::string Log;
      Log += std::to_string(curx.x);
      Log += " ";
      Log += std::to_string(curx.y);
      logger.log(Log);
#endif

      // 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 = 1;
        k++;
// tc ++;
#ifdef IF_LOG
        std::string Log;
        Log += std::to_string(curx.x);
        Log += " ";
        Log += std::to_string(curx.y);
        logger.log(Log);
#endif
      }
      // std::cout << "tc:" << tc << "\n";
```

```
      return {curx, fmin};
    }
    break;

    case CG:
    {
      int k = 0;              // 迭代次数
      double alpha = 3;         // 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
#ifdef IF_LOG
      Logger logger(std::string("WEEK3\\CG_") + (FRorPRP ? "FR" :
"PRP") + ".log");
      std::string Log;
      Log += std::to_string(curx.x);
      Log += " ";
      Log += std::to_string(curx.y);
      logger.log(Log);
#endif

      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 = 3;
        }
        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)) /
(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 = 3;
        }
        k++;
#ifdef IF_LOG
        std::string Log;
        Log += std::to_string(curx.x);
        Log += " ";
        Log += std::to_string(curx.y);
        logger.log(Log);
#endif
      }
      return {curx, fmin};
    }
    break;
```

```
    default:
    {
      throw "Unexpection Search Mod Exception";
      return {{0, 0}, -1};
    }
    break;
    }
    throw "Unknown Exception";
    return {{0, 0}, -1};
  }
}
#endif
```

7.1.2. 测试代码

7.1.2.1. `TOFtest.cpp`

```
/**
 * @file TOFtest.cpp
 * @brief True or False test
 */

#include <iostream>
#include <stdlib.h>
#include "core.h"
using namespace std;

using ODSearch::Corrdinate;

int tc = 1;       // test case
Corrdinate dev = 0.03; // deviation

double f(Corrdinate x)
{
    return (x.x - dev.x) * (x.x - dev.x) + (x.y - dev.y) * (x.y -
dev.y);
}
Corrdinate df(Corrdinate x)
{
    return Corrdinate(2 * (x.x - dev.x), 2 * (x.y - dev.y));
}
int main()
```

```
{
    double acc = 0.001;
    Corrdinate thn = dev;
    double eps = 1e-5;
    srand(1145);

    auto randint = [](int l, int r) -> int
    {
        return (int)((rand() * (r - l)) / (RAND_MAX) + l);
    };

    while (tc--)
    {

        if(tc == 5)
            tc -= 0;
        dev = {((double)randint(1, 100)) / 50.0,
((double)randint(1, 100)) / 50.0};
        thn = dev;
        eps = pow(0.1, abs(randint(1, 10)));

        cout << "\n----Test Cases" << 10 - tc << "----\n";

        cout << "<search data> eps:" << eps << "\n";

        cout << "<Theoretical> ans:(" << thn.x << " " << thn.y <<
") acc:"
             << "inf\n";
        acc = eps;
        try
        {
            auto ans = ODSearch::find_mininum(f, df,{0.0,0.0},
ODSearch::GD,eps);
            cout << "[    G D    ] ans:" << ans.second << "
df(ans)"<<df(ans.first).norm()<<" df(thn):"<<df(thn).norm()<< "
at:(" << ans.first.x << " " << ans.first.y << ") acc:" << (acc /
(df(thn)-df(ans.first)).norm()) * 100 << " dev:" << max(0.0,
(df(thn)-df(ans.first)).norm() - acc) / acc * 100 << "%\n";
            ans = ODSearch::find_mininum(f, df,{0.0,0.0},
ODSearch::CG,eps);
            cout << "[  C G(FR)  ] ans:" << ans.second << "
df(ans)"<<df(ans.first).norm()<<" df(thn):"<<df(thn).norm()<< "
```

```
at:(" << ans.first.x << " " << ans.first.y << ") acc:" << (acc /
(df(thn)-df(ans.first)).norm()) * 100 << " dev:" << max(0.0,
(df(thn)-df(ans.first)).norm() - acc) / acc * 100 << "%\n";
            ODSearch::FRorPRP = 0;
            ans = ODSearch::find_mininum(f, df,{0.0,0.0},
ODSearch::CG,eps);
            cout << "[  C G(PBD) ] ans:" << ans.second << "
df(ans)"<<df(ans.first).norm()<<" df(thn):"<<df(thn).norm()<< "
at:(" << ans.first.x << " " << ans.first.y << ") acc:" << (acc /
(df(thn)-df(ans.first)).norm()) * 100 << " dev:" << max(0.0,
(df(thn)-df(ans.first)).norm() - acc) / acc * 100 << "%\n";
            ODSearch::FRorPRP = 1;
        }
        catch(const std::exception& e)
        {
            std::cout << e.what() << '\n';
        }



        // cout << "[SECANT ] ans:" << ans.second << " at:" <<
ans.first << " acc:" << (acc / abs(thn - ans.first)) * 100 << "
dev:" << max(0.0, abs(thn - ans.first) - acc) / acc * 100 << "%
\n";
    }
    system("pause");
}
```

7.1.2.2. `WTHtest.cpp`

```
/**
 * @file WTHtest.cpp
 * @brief 最速下降法的最坏情况测试
 */

#include <iostream>
#include <stdlib.h>
#include "core.h"
#include <time.h>
#include <math.h>
using namespace std;
using ODSearch::Corrdinate;
```

```
int N = 1e5;                  // test case
Corrdinate dev = {0.5, 0.5}; // deviation

double f(Corrdinate x)
{
    return (x.x - dev.x) * (x.x - dev.x) + (x.y - dev.y) * (x.y -
dev.y) / 2;
}
Corrdinate df(Corrdinate x)
{
    return Corrdinate(2 * (x.x - dev.x), (x.y - dev.y));
}
int main()
{
    int tc = 0;
    Corrdinate thn = dev;
    double eps = 1e-5;

    cout << "<search data> eps:" << eps << "\n";

    cout << "<Theoretical> ans:(" << thn.x << " " << thn.y << ")
acc:"
         << "inf\n";
    double acc = eps;
    auto ans = ODSearch::find_mininum(f, df, {0.0, 0.0},
ODSearch::GD, eps);
    cout << "[    G D    ] ans:" << ans.second << " df(ans)" <<
df(ans.first).norm() << " df(thn):" << df(thn).norm() << " at:("
<< ans.first.x << " " << ans.first.y << ") acc:" << (acc /
(df(thn) - df(ans.first)).norm()) * 100 << " dev:" << max(0.0,
(df(thn) - df(ans.first)).norm() - acc) / acc * 100 << "%\n";
    ans = ODSearch::find_mininum(f, df, {0.0, 0.0}, ODSearch::CG,
eps);
    cout << "[  C G(FR)  ] ans:" << ans.second << " df(ans)" <<
df(ans.first).norm() << " df(thn):" << df(thn).norm() << " at:("
<< ans.first.x << " " << ans.first.y << ") acc:" << (acc /
(df(thn) - df(ans.first)).norm()) * 100 << " dev:" << max(0.0,
(df(thn) - df(ans.first)).norm() - acc) / acc * 100 << "%\n";
    ODSearch::FRorPRP = 0;
    ans = ODSearch::find_mininum(f, df, {0.0, 0.0}, ODSearch::CG,
eps);
```

```
    cout << "[  C G(PBD) ] ans:" << ans.second << " df(ans)" <<
df(ans.first).norm() << " df(thn):" << df(thn).norm() << " at:("
<< ans.first.x << " " << ans.first.y << ") acc:" << (acc /
(df(thn) - df(ans.first)).norm()) * 100 << " dev:" << max(0.0,
(df(thn) - df(ans.first)).norm() - acc) / acc * 100 << "%\n";
    ODSearch::FRorPRP = 1;
    int begin = clock();
    while (N > tc++)
    {

        ans = ODSearch::find_mininum(f, df, {0.0, 0.0},
ODSearch::GD, eps);
    }
    int end = clock();
    tc = 0;
    cout << "[    G D    ] cost:" << double(end - begin) /
CLOCKS_PER_SEC << "s" << "\n";
    begin = clock();
    while (N > tc++)
    {

        ans = ODSearch::find_mininum(f, df, {0.0, 0.0},
ODSearch::CG, eps);
    }
    end = clock();
    tc = 0;
    cout << "[  C G(FR)  ] cost:" << double(end - begin) /
CLOCKS_PER_SEC << "s" << "\n";
    begin = clock();
    while (N > tc++)
    {

        ODSearch::FRorPRP = 0;
        ans = ODSearch::find_mininum(f, df, {0.0, 0.0},
ODSearch::CG, eps);

        ODSearch::FRorPRP = 1;
    }
    end = clock();
    tc = 10;
    cout << "[  C G(PBD) ] cost:" << double(end - begin) /
CLOCKS_PER_SEC << "s" << "\n";
```

```
    cout << ans.first.x << "\n";
    system("pause");
}
```

7.1.2.3. `CMFtest.cpp`

```
/**
 * @file CMFtest.cpp
 * @brief Complex Model Funtion test
 */

#include <iostream>
#include <stdlib.h>
#include "core.h"
#include <time.h>
#include <math.h>
using namespace std;
using ODSearch::Corrdinate;

int N = 50;               // test case
Corrdinate dev = {1, 1}; // deviation
int k = 1e5;

double f(Corrdinate x)
{
    return (1 - x.x) * (1 - x.x) + 3 * (x.y - x.x * x.x) * (x.y -
x.x * x.x);
}
Corrdinate df(Corrdinate x)
{
    return Corrdinate(-2 * (1 - x.x) - 12.0 * (x.x * x.y - x.x *
x.x * x.x), 6 * x.y - 6.0 * x.x * x.x);
}
int main()
{
    int tc = 0;
    Corrdinate thn = dev;
    double eps = 1e-5;
    cout << "<search data> eps:" << eps << "\n";

    cout << "<Theoretical> ans:(" << thn.x << " " << thn.y << ")
acc:"
```

```
          << "inf\n";
    double acc = eps;
    auto ans = ODSearch::find_mininum(f, df, {0.0, 0.0},
ODSearch::GD, eps);
    cout << "[    G D    ] ans:" << ans.second << " df(ans)" <<
df(ans.first).norm() << " df(thn):" << df(thn).norm() << " at:("
<< ans.first.x << " " << ans.first.y << ") acc:" << (acc /
(df(thn) - df(ans.first)).norm()) * 100 << " dev:" << max(0.0,
(df(thn) - df(ans.first)).norm() - acc) / acc * 100 << "%\n";
    ans = ODSearch::find_mininum(f, df, {0.0, 0.0}, ODSearch::CG,
eps);
    cout << "[  C G(FR)  ] ans:" << ans.second << " df(ans)" <<
df(ans.first).norm() << " df(thn):" << df(thn).norm() << " at:("
<< ans.first.x << " " << ans.first.y << ") acc:" << (acc /
(df(thn) - df(ans.first)).norm()) * 100 << " dev:" << max(0.0,
(df(thn) - df(ans.first)).norm() - acc) / acc * 100 << "%\n";
    ODSearch::FRorPRP = 0;
    ans = ODSearch::find_mininum(f, df, {0.0, 0.0}, ODSearch::CG,
eps);
    cout << "[  C G(PBD) ] ans:" << ans.second << " df(ans)" <<
df(ans.first).norm() << " df(thn):" << df(thn).norm() << " at:("
<< ans.first.x << " " << ans.first.y << ") acc:" << (acc /
(df(thn) - df(ans.first)).norm()) * 100 << " dev:" << max(0.0,
(df(thn) - df(ans.first)).norm() - acc) / acc * 100 << "%\n";
    ODSearch::FRorPRP = 1;
    int begin = clock();
    while (N > tc++)
    {

        ans = ODSearch::find_mininum(f, df, {0.0, 0.0},
ODSearch::GD, eps);
    }
    int end = clock();
    tc = 0;
    cout << "[    G D    ] cost:" << double(end - begin) /
CLOCKS_PER_SEC << "s" << "\n";
    begin = clock();
    while (N > tc++)
    {

        ans = ODSearch::find_mininum(f, df, {0.0, 0.0},
ODSearch::CG, eps);
```

```
    }
    end = clock();
    tc = 0;
    cout << "[  C G(FR)  ] cost:" << double(end - begin) /
CLOCKS_PER_SEC << "s" << "\n";
    begin = clock();
    while (N > tc++)
    {

        ODSearch::FRorPRP = 0;
        ans = ODSearch::find_mininum(f, df, {0.0, 0.0},
ODSearch::CG, eps);

        ODSearch::FRorPRP = 1;
    }
    end = clock();
    tc = 10;
    cout << "[  C G(PBD) ] cost:" << double(end - begin) /
CLOCKS_PER_SEC << "s" << "\n";
    cout << ans.first.x << "\n";
    system("pause");
}
```

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