README

问题

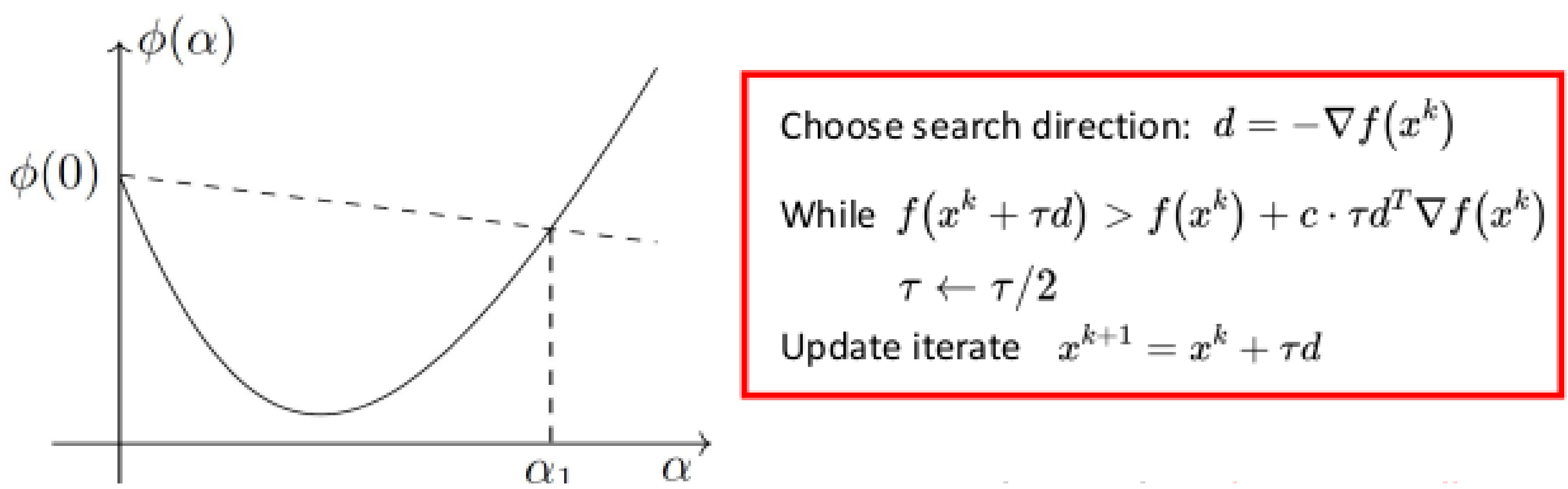
最小化

$$f(\mathbf{x})=f(x_1,x_2,\ldots,x_N)=\sum_{i=1}^{N/2}\left[100\left(x_{2i-1}^2-x_{2i}\right)^2+\left(x_{2i-1}-1\right)^2\right]$$

的 X 向量

方法

Armijo line search



程序如下

```C++
#include<bits/stdc++.h>
using namespace std;

using vd=vector<double>;
constexpr int max_iter=2000000;
constexpr double minError=0.01;
double function_value(int N,vd& X){

    double res=0;
    for(int i=1;i<=N/2;i++){//x1, x2,...,xn
        res+=100*(pow(X[2*i-2],2)-X[2*i-1])*(pow(X[2*i-2],2)-X[2*i-1])
                +(X[2*i-2]-1)*(X[2*i-2]-1);
    }
    return res;
}

vd diff_function(int N,vd& X){
    vd res(N);
    int bound=N/2;
    for(int i=1;i<=N/2;i++){

        res[2*i-2]=400*(pow(X[2*i-2],2)-X[2*i-1])*X[2*i-2]+2*(X[2*i-2]-1);
        res[2*i-1]=-200*(pow(X[2*i-2],2)-X[2*i-1]);
```

```
    }
    if(N%2==1){
        res[N-1]=400*(pow(X[N-2],2)-X[N-1])*X[N-2]+2*(X[N-2]-1);
    }
    return res;
}



double norm2(vd& w){
    double res;
    for(double i:w)
    {
        res+=i*i;
    }
    return sqrt(res);
}
//TODO Goldstein 准则
double goldsteinsearch(vd& Xk_iter,vd& X_k,double X_kV,int  N,vd& d,double alpham,double rho,double t)
{
    double c=0.1;
    double b=alpham;//原步长
    srand(time(NULL));//设置随机数种子，使每次产生的随机序列不同
    double rd=rand()%(1000)/(double)(1000);//小数点后三位
    double alpha=b*rd;//
    bool flag=true;
    double dp=0;//点乘之和
    double a=0.0;//调节系数
    for(int i=0;i<N;i++){
        dp+=-d[i]*d[i];

    }
    while(flag){
        for(int m=0;m<N;m++){
            Xk_iter[m]=X_k[m]+t*d[m];//update
        }
        double iterV=function_value(N,Xk_iter);
        if(iterV-X_kV<=rho*alpha*dp) {
            if(iterV-X_kV>=(1-rho)*alpha*dp){
                flag=false;
            }
            else{
                if(b<alpham){
                    alpha=(a+b)/2;
                }
                else{
                    alpha=t*alpha;
                }
            }
        }
        else{
            b=alpha;
            alpha=(a+b)/2;
        }


    }
    return alpha;

}
double dot_vd(vd &d,int N){
    double res=0;
    for(int l=0;l<N;l++){
        res+=d[l]*d[l];
    }
    return res;
}
void coutO(const string& str,double val){
```

```
    cout<<str<<" = "<<val<<endl;
}

void Armijio_line_search_G(vd& initV,double t,int N)
{
    double c=0.1;

    vd X_k=initV;
    int inner_iter=0;
    double error=100;
    for(int i=0;i<max_iter&&(error>minError);i++){

        vd d=diff_function(N,X_k);//梯度
        error=norm2(d);
        coutO("error",error);
        for(int di=0;di<d.size();di++){
            d[di]*=-1;//负梯度
        }
        vd Xk_iter(N);
        for(int m=0;m<N;m++){
            Xk_iter[m]=X_k[m]+t*d[m];//update
        }
//          goldsteinsearch(vd& Xk_iter,vd& X_k,double X_kV,int  N,vd& d,double alpham,double rho,double t)

        double X_kV=function_value(N,X_k);
        bool flag=true;
        t= goldsteinsearch(Xk_iter,X_k,X_kV,N,d,t,c,0.5);
        //在迭代前需要设定下一步的值 note
        for(int m=0;m<N;m++){
            Xk_iter[m]=X_k[m]+t*d[m];//update
        }

        for(int m=0;m<N;m++){
            X_k[m]=X_k[m]+t*d[m];

        }
        //2-norm
        inner_iter++;

    }
    for(auto an:X_k){
        cout<<an<<endl;
    }
    cout<<"inner_iter= "<<inner_iter<<endl;

}

void Armijio_line_search(vd& initV,double t,int N)
{
    double c=0.01;

    vd X_k=initV;
    int inner_iter=0;
    double error=100;
    for(int i=0;i<max_iter&&(error>minError);i++){

        vd d=diff_function(N,X_k);//梯度
        error=norm2(d);
        coutO("error",error);
        for(int di=0;di<d.size();di++){
            d[di]*=-1;//负梯度
        }
        vd Xk_iter(N);

        double X_kV=function_value(N,X_k);
        bool flag=true;
        //在迭代前需要设定下一步的值 note
        for(int m=0;m<N;m++){
```

```
            Xk_iter[m]=X_k[m]+t*d[m];//update
        }
        // Armijo Condition 计算步长
        while(function_value(N,Xk_iter)>(X_kV+c*t*dot_vd(d,N))){
            t/=2;//t->0
            for(int m=0;m<N;m++){
                Xk_iter[m]=X_k[m]+t*d[m];//update
            }

            double delta=0;//这里 note =0 不然会叠加

        }
        for(int m=0;m<N;m++){
            X_k[m]=X_k[m]+t*d[m];

        }
        //2-norm
        inner_iter++;

    }
    for(auto an:X_k){
        cout<<an<<endl;
    }
    cout<<"inner_iter= "<<inner_iter<<endl;

}


int main(){
    int N=4;
    // vx X(1,2);//t tau
    // X.resize(N);
    // X<<1,2;
    vd X{-3.2,4.0,1.5,3} ;
    double t=0.1;
    Armijio_line_search(X,t,N);
//    Armijio_line_search_G(X,t,N);
    return 0;
}
```

一开始打算用 eigen 库写，但发现无法调试，显示以下报错

```C++
useSophus: /usr/include/eigen3/Eigen/src/Core/PlainObjectBase.h:285: void Eigen::PlainObjectBase<Derived>::resize
Matrix<double, 3, 1>; Eigen::Index = long int]: Assertion `(!(RowsAtCompileTime!=Dynamic) || (rows==RowsAtCompile
olsAtCompileTime)) && (!(RowsAtCompileTime==Dynamic && MaxRowsAtCompileTime!=Dynamic) || (rows<=MaxRowsAtCompile
ompileTime!=Dynamic) || (cols<=MaxColsAtCompileTime)) && rows>=0 && cols>=0 && "Invalid sizes when resizing a mat
Aborted (core dumped)
```

没有解决……

后来用标准库写，要写的东西挺多，调试调了半天，发现一个变量没有事先初始化，导致一直无法出循环

结果分析

τ 初始值为 0.1 情况下，改变梯度变化最大值限制

梯度变化<	初始化值	结果	迭代次数
0.1	{-1.2,2.0}	{0.89，0.80}	5652
0.01	同上	{0.98,0.97}	12331
0.001	同上	{0.99,0.99}	19633

显而易见，最大值越小，精度越高，迭代次数也随之增加；

改变初始值

梯度变化<	初始化值	结果	迭代次数
0.01	{-3.2,4.0}	{0.98，0.97}	30102

可以看到，迭代次数迅速提升，说明初始值对于算法的快速性影响较大；

改变变量维度为 4

梯度变化<	初始化值	结果	迭代次数
0.01	{-3.2,4.0,60,3}	{-1.41， 2.0，-1.35,1.85}	2000000
0.01	{-3.2,4.0,1.5,3 }	{0.995284 0.990572 1.01021 1.02056}	35512

变量增加，对比前面相同梯度变化最大值限制，迭代次数变化较小，该算法在多变量优化方面较好

,但这是建立在**初值条件较好的情况下**

麻烦问下助教这是什么原因,有什么好的解决方法

Todo

Goldstein 准则调试实现

使用 matplot-cpp 绘图

参考

https://blog.csdn.net/weixin_45735391/article/details/118705597

https://blog.csdn.net/qq_39779233/article/details/119973486

最优化方法：无约束梯度问题----最速下降法理论与C++实现_weixin_35338624的博客…
背景优化问题的关键在于每次更新时方向、步长的选择（比较简单的方式是固定步长，缺陷步长选太小收敛速度慢，太大则在靠近最优值时剧烈震荡）。最速下降方法：方向选择的是负梯度方向，步长通过极小化 g(k) = f(x0+k*dx) 确定（这里 g(k) 是关于k这个常数的一元函数，可以通过
blog.csdn.net

[笔记整理] 一维搜索_罗西的思考的博客-CSDN博客_一维搜索
本文是阅读Alink源码期间在网上查找资料做的笔记整理，把找到的算法实现加了一些注解。
blog.csdn.net

最速下降法steepest descent详细解释_微电子学与固体电子学-俞驰的博客-CSDN博客
---------------------先闲聊几句---------------------------------[1]首次提出了梯度下降法和最速下降法，既然柯西写出来了，所以这两个算法肯定不个一个东西,它们的区别是学习率是否恒定。[4]提出了GoldStein法则Wolfe准则以及Goldstein[5][8]给出了具体的代码实现,[6][7]中给出了手算
blog.csdn.net

Rosenbrock函数_蓝净云的博客-CSDN博客_rosenbrock函数
定义在数学最优化中，Rosenbrock函数是一个用来测试最优化算法性能的非凸函数，由Howard Harry Rosenbrock在1960年提出 。也称为Rosenbrock山谷或Rosenbrock香蕉函数，也简称为香蕉函数。Rosenbrock函数的定义如下：f(x,y)=(a−x)2+b(y−x2)e2.f(x,y) = (a-x)^2+b(y-
blog.csdn.net