Lab 4 - 非线性最小二乘

一、实验要求

本次实验我们实现 Gauss Newton 法求解简单的非线性最小二乘问题。

二、理论基础

Gauss Newton 法来源于 Newton Rapson法，后者利用二次型逼近目标函数。为了进行二次逼近，NR法需要计算目标函数的 Hessian，但这并不是个容易完成的任务。GN方法利用最小二乘问题本身的特点，通过计算 Jacobian 来近似 Hessian 。

NR法采用二阶 Taylor 展开近似目标函数：

$$F(x+\Delta x)=F(x)+J_F\Delta x+\frac{1}{2}\Delta x^T H_F\Delta x$$

最小二乘问题具有$F(x)=\|R(x)\|_2^2$ 的形式，对R进行一阶Taylor展开，可以得到：

$$\begin{aligned}F(x+\Delta x)&=\|R(x+\Delta x)\|_2^2\\&\approx\|R(x)+J_R\Delta x\|_2^2\\&=\|R(x)\|_2^2+2R^TJ_R\Delta x+\Delta x^TJ_R^TJ_R\Delta x\\&=F(x)+J_F\Delta x+\Delta x^TJ_R^TJ_R\Delta x\end{aligned}$$

将两种展开方式进行对比，可知在最小二乘问题里$H_F\approx 2J_R^TJ_R$。

继续依照NR法的思路，每一步迭代中，我们求解如下的标准方程：

$$J_R^TJ_R\Delta x+J_R^TR=0$$

这一标准方程对应于求$J_R^T\Delta x=-R$的（线性）最小二乘解，可以采用共轭梯度法求解。本次实验中，可以直接采用`OpenCV`提供的矩阵算法完成。

得到Δx 后，以它作为下降方向，我们进行线性搜索求出合适的步长α，更新x： $x\leftarrow x+\alpha\Delta x$

这样就得到了GN法，GN法的算法框架如下：

$$\begin{aligned}&x\leftarrow x_0\\&n\leftarrow 0\\&while\quad n<n_{max}:\\&\qquad\Delta x\leftarrow Solution\quad of\quad J_R\Delta x=-R:\\&\qquad\quad Conjugate\quad Gradient\quad or\quad Other\\&\qquad if\quad \|R\|_\infty\le\epsilon_r\vee\|\Delta x\|_\infty\le\epsilon_g\quad return\quad x\\&\qquad\alpha\leftarrow\arg\min_\alpha\{x+\alpha\Delta x\}\\&\qquad x\leftarrow x+\alpha\Delta x\\&\qquad n\leftarrow n+1\end{aligned}$$

作为测试，我们求解从三维点云拟合椭球的问题，我们的椭球模型是：

$$\frac{x^2}{A^2}+\frac{y^2}{B^2}+\frac{z^2}{C^2}=1$$

下载测试数据文件ellipse753.txt，其中包含了 753 个含有噪音的点，你需要建立目标函数，然后利用你的优化器最小化目标函数，从这个点云恢复出参数 A、B、C 。

三、运行环境

编译环境：Visual Studio 2017

运行环境：Windows10

四、具体实现

4.1 接口设计

我需要实现名为 `Solver2721` 的类，这个类需要继承自接口类 `GaussNewtonSolver` 。
`GaussNewtonSolver` 的定义如下：

```
class GaussNewtonSolver {
public:
    virtual double solve(
                ResidualFunction *f, // 目标函数
                double *X,           // 输入作为初值，输出作为结果
                GaussNewtonParams param = GaussNewtonParams(), // 优化参数
                GaussNewtonReport *report = nullptr // 优化结果报告
                ) = 0;
};
```

- 当你的类的 `solve` 函数被调用时，它将采用 Gauss Newton 法最小化函数 `f` 。
- 输入时数组 `X` 内包含初始点，并且在优化后将被修改为最优点，维度需要与目标函数定义的维度一致。
- `param` 是 GN 算法运行的各种相关参数。
- 如果 `report` 不是空指针，优化完成后应当把相关的报告记录到 `report` 。
- 函数返回值是优化得到的目标函数的最优值。

目标函数通过继承接口类 `ResidualFunction` 进行定义：

```
class ResidualFunction {
public:
    virtual int nR() const = 0;
    virtual int nX() const = 0;
    virtual void eval(double *R, double *J, double *X) = 0;
};
```

`nR` 函数需要返回余项向量的维度。类似的， `nX` 函数需要返回变量 `X` 的维度。
`eval` 运行时读入 `X` 将计算得到的余项和Jacobian写入 `R` 和 `J` 。

- 计算得到的**余项向量**需要写入到 `R` ， `R` 需要预先分配好，长度为 `nR` 。
- 计算得到的 Jacobian 需要写入到 `J` ， `J` 需要预先分配好，大小为 `nR*nX` 。
- 输入的 `X` 是一个长度为 `nX` 的数组，包含所有的变量。

注意：你的优化器需要负责在优化开始前分配好 `R` 和 `J` 需要的空间，在结束后销毁。 `X` 作为输入，由用户（调用 `solve` 的程序）分配并填充好初始值。

优化的参数通过如下的结构体定义，各个参数的含义见注释。

```
struct GaussNewtonParams{
    GaussNewtonParams() :
       exact_line_search(false),
       gradient_tolerance(1e-5),
```

```
        residual_tolerance(1e-5),
        max_iter(1000),
        verbose(false)
    {}
    bool exact_line_search; // 使用精确线性搜索还是近似线性搜索
    double gradient_tolerance; // 梯度阈值，当前梯度小于这个阈值时停止迭代
    double residual_tolerance; // 余项阈值，当前余项小于这个阈值时停止迭代
    int max_iter; // 最大迭代步数
    bool verbose; // 是否打印每步迭代的信息
};
```

优化结果的详细信息储存的结构体定义如下，含义见注释。

```
struct GaussNewtonReport {
    enum StopType {
        STOP_GRAD_TOL,        // 梯度达到阈值
        STOP_RESIDUAL_TOL,    // 余项达到阈值
        STOP_NO_CONVERGE,     // 不收敛
        STOP_NUMERIC_FAILURE // 其它数值错误
    };
    StopType stop_type; // 优化终止的原因
    double n_iter;      // 迭代次数
};
```

4.2 类`Solver2721`

这部分需要完成继承自接口类`GaussNewtonSolver`的类`Solver2721`，首先定义变量并分配空间。

```
double *x = X;
double alpha = 1;
int nR = f->nR();
int nX = f->nX();
double *J = new double[nR*nX];
double *R = new double[nR];
double *delta_x = new double[nR];
int count;
```

进入迭代并创建我们需要的量。

```
for (count = 0; count< param.max_iter; count++);
```

```
f->eval(R, J, x);
Mat mat_R(nR, 1, CV_64FC1, R);
Mat mat_J(nR, nX, CV_64FC1, J);
Mat delta_x(nX, 1, CV_64FC1);
cv::solve(mat_J, mat_R, delta_x, DECOMP_SVD);

double Res = mat_R.at<double>(0, 0);
double Gra = delta_x.at<double>(0, 0);

for (int i = 0; i < nR; i++) {
    double r = abs(mat_R.at<double>(i, 0));
    if ( r > Res) Res = r;
}
```

```
for (int i = 0; i < nX; i++) {
    double g = abs(delta_x.at<double>(i, 0));
    if (g > Gra) Gra = g;
}
```

判断梯度或余项是否达到阈值，若达到阈值则终止迭代。

```
if (Gra <= param.gradient_tolerance) {
    report->stop_type = report->STOP_GRAD_TOL;
    report->n_iter = count;
    return 0;
}

if (Res <= param.residual_tolerance) {
    report->stop_type = report->STOP_RESIDUAL_TOL;
    report->n_iter = count;
    return 0;
}
```

计算更新`x`的值：

```
for (int i = 0; i < nX; i++) {
    x[i] += alpha * delta_x.at<double>(i, 0);
}
```

如果达到迭代次数但仍未终止，说明不收敛：

```
report->stop_type = report->STOP_NO_CONVERGE;
report->n_iter = count;
return 1;
```

4.3 类`ObjectFunction`

构造函数和析构函数，首先读取坐标点数据，在此之前新定义`double *x,*y,*z;`用来存储坐标数据：

```
ObjectFunction(){
    FILE* file;
    fopen_s(&file, "ellipse753.txt", "r");//读入文件
    x = new double[number];
    y = new double[number];
    z = new double[number];
    for (int i = 0; i < number; i++) {//存储坐标点
        fscanf_s(file, "%lf", &x[i]);
        fscanf_s(file, "%lf", &y[i]);
        fscanf_s(file, "%lf", &z[i]);
    }
}

~ObjectFunction(){//结束后销毁坐标数据
    delete x;
    delete y;
    delete z;
}
```

定义两个函数用于计算平方值和立方值便于调用以简化程序：

```
double Square(double x) {
    return x * x;
}

double Cube(double x) {
    return x * x * x;
}
```

计算余项表达式的值：

```
double Residual_value(double *eclipse, int i) {
    double A = eclipse[0];
    double B = eclipse[1];
    double C = eclipse[2];
    return 1 - 1.0 / Square(A)*Square(x[i]) - 1.0 / Square(B)*Square(y[i]) - 1.0
/ Square(C)*Square(z[i]);
}
```

下面继承接口类`ResidualFunction`进行定义，这里体现了我们本次实验利用的是非线性最小二乘法：

```
virtual int nR() const {
    return number;//返回余项向量的维度
}

virtual int nX() const {
    return dimension;//返回变量X的维度。
}

virtual void eval(double *R, double *J, double *eclipse) {
    for (int i = 0; i < number; i++) {
        R[i] = Residual_value(eclipse, i);//计算余项值
        J[i * 3 + 0] = -2 * Square(x[i])/Cube(eclipse[0]);
        J[i * 3 + 1] = -2 * Square(y[i])/Cube(eclipse[1]);
        J[i * 3 + 2] = -2 * Square(z[i])/Cube(eclipse[2]);
    }//读入X将计算得到的余项和Jacobian写入R和J
}
```

4.4 主程序

首先创建新变量存储椭球模型的参数

```
#define number 753
#define dimension 3

double *Eclipse = new double[3];
for (int i = 0; i < 3; i++) {
    Eclipse[i] = 1;
}
ResidualFunction* f= new ObjectFunction();
GaussNewtonParams param = GaussNewtonParams();
GaussNewtonReport report;
```

调用函数：

```
Solver2721* solver = new Solver2721();
solver->solve(f, Eclipse, param, &report);
```

打印结果：

```
cout << "迭代次数: " << report.n_iter << endl;
cout << "A:" << " " << Eclipse[0] << endl;
cout << "B:" << " " << Eclipse[1] << endl;
cout << "C:" << " " << Eclipse[2] << endl;
if (report.stop_type == report.STOP_RESIDUAL_TOL) {
    cout << "StopType: 余项达到阈值" << endl;
}
else if (report.stop_type == report.STOP_GRAD_TOL) {
    cout << "StopType: 梯度达到阈值" << endl;
}
else if (report.stop_type == report.STOP_NO_CONVERGE) {
    cout << "StopType: 不收敛" << endl;
}
else {
    cout << "StopType: 其它数值错误" << endl;
}
```

五、实验结果及分析

5.1运行结果

运行发现程序输出的结果较为符合预期，迭代共进行`6`次，终止的原因是梯度达到阈值。

```
迭代次数: 6
A: 2.94404
B: 2.30504
C: 1.79783
StopType: 梯度达到阈值
```

5.2问题分析与改进

在本次实验中，有很多有待改进之处，比如在确定精确线性搜索还是近似线性搜索时，我并未对步长α作进一步的探讨，稍微尝试了一下后便浅尝辄止了，最终也只选择了固定的步长，另外也还有很多参量的选取操作不够完善。

六、心得体会

本次实验难度较低，实现起来比较方便，和上次实验相比有一个共同点，对数理能力的要求较高，只有对算法中各个参量的含义和内涵有了比较透彻的理解，知其然并知其所以然，才能对算法本身有较为深刻的见解，也才有可能进一步提出优化和改进之处。现在学期过去了小半部分，经过课堂的学习和实验的回顾，我发现课程对于数学能力的要求是比较高的，也非常锻炼思维和实践能力，将一个巧妙的数学算法通过代码的形式呈现出来是一件非常有成就感的事情。

七、参考文献

[1]. https://www.cnblogs.com/xiaoboge/p/9713312.html