基础优化算法的编程实现

简介

优化算法在源定位问题中大量使用，无线传感器网络(Wireless Sensor Networks, WSNs)中用于定位的所有距离相关(Range Based Localization)方法，即RSSI、AoA、ToA和TDoA，均采用优化算法进行源位置估计。梯度下降算法和牛顿法是优化算法的最基础方法，虽然这两种算法具有缺陷，如梯度下降法在趋近真值时收敛速度下降、牛顿法的Hessian逆矩阵求解复杂甚至无解，但由于其基础性，以及拥有大量衍生算法，因此对其定位性能的研究具有重要的对照意义。

本文在Linux平台下，采用C++语言编写梯度下降算法的实现程序。程序的基本思路是以虚基类构建计算流程，以继承类定义代价函数(Cost Function)的形式，代价函数由用户自定义，继承关系由C++的多态特性辅助。

优化算法通过三个部分实现，残差块(Residual Block)构造、代价函数(Cost Function)构造和优化计算(Optimization)。由于优化计算的框架是固定的，只有具体到残差块函数、代价函数等是不同的，因此将优化计算整体拆分为计算框架和具体内容两个部分，其中的计算框架部分由程序构造，具体内容由用户自定义。为了达成这一思路，程序以虚基类构建计算流程，以继承类定义残差块、代价函数等的形式，代价函数由用户自定义，继承关系由C++的多态特性辅助。

程序包含三个虚基类，残差块类(ResidualBlockFunction)、代价函数类(CostFunction)和优化算法管理类(OptimizationManager)，残差块类负责定义残差块的数学结构，为最基础的类；代价函数类负责构造代价函数的数学结构，该结构通过残差块构造，并且负责代价函数的导数求解，导数使用数值微分的方法进行求解；优化算法管理类统筹整个优化计算的实现过程。

虚基类名称以"User"开头，继承类名称以描述自身目标的词汇而非"User"开头。

程序结构

残差块类

残差块类包含两个部分，虚基类和继承类。虚基类名为"UserResidualBlockFunction"，继承类名为"PolyResidualBlockFunction"。

代价函数类

代价函数类包含两个部分，虚基类"UserCostFunction"和继承类"SteepestCostFunction"。

代价函数类由优化算法管理类调用，负责为优化算法管理类提供代价函数的函数值、梯度值（即一阶导数，也称为Jacobi Matrix）、黑森矩阵（Hessian Matrix，即二阶导数）等，因此其核心功能是计算代价函数值、计算代价函数的导数值。同时，代价函数类负责管理残差块，由此，代价函数类的核心功能还包括残差块的添加。

一个典型的代价函数类的虚基类如下，核心功能所指的函数包括了添加计算代价函数值`bool CostFunction`，计算代价函数的梯度值`bool DerivativesFunction`，残差块函数`void AddResidualBlock`。而其它函数是用于辅助核心功能的，包括计算代价函数对某一参数的梯度值的函数`bool GetOneDerivative`，设定迭代步长`void SetStepLength`。

在虚基类的这些函数中，有一部分是纯虚函数(Pure Virtual Function)，它们的定义交由继承类完成，并且继承类必须定义，否则程序无法完成编译。这些函数都是与用户的选择挂钩的，包括添加残差块函数和计算代价函数值的函数。残差块和代价函数必须由用户自行定义，虽然在大量研究文献中，残差块都定义为观测数值与理论数值之差，代价函数都定义为残差块的平方和，但这不代表残差块和代价函数没有其它定义方式，因此，这两个函数的定义交由继承类完成。

```
class UserCostFunction
{
    public:
        UserCostFunction(string name, int SizeObservations, int SizeVariables,
int SizeResiduals);
        ~UserCostFunction();

    public:
        // pure virtual
        virtual void AddResidualBlock(vector<double> observations) = 0;
        virtual bool CostFunction(vector<double> variables, vector<double>
&CostFunctionValues)=0;

    public:
        // virtual
        virtual bool GetOneDerivative(int VarialbleID, vector<double>
variables, double &theDerivativeValue);

    public:



        bool DerivativesFunction(vector<double> variables, vector<double>
&theDerivatives);
        void SetStepLength(double delta);

    public:
        virtual void Show() = 0;

    protected:
        vector<UserResidualBlockFunction*> ResidualBlockFunctions_;
        int SizeObservations_;
        int SizeVariables_;
        int SizeResiduals_;
```

```
        // for derivative calculation
        double delta_;

    private:
        string name_;
};
```

虚基类"UserCostFunction"的头文件部分

```
class SteepestCostFunction : virtual public UserCostFunction
{
    public:
        SteepestCostFunction(string name, int SizeObservations, int
SizeVariables, int SizeResiduals);

        ~SteepestCostFunction();

    public:
        void Show();

    public:
        virtual void AddResidualBlock(vector<double> observations);
        virtual bool CostFunction(vector<double> variables, vector<double>
&CostFunctionValues);

    public:
        virtual bool GetOneDerivative(int VarialbleID, vector<double>
variables, double &theDerivativeValue);

    private:
        string name_;
};
```

继承类"SteepestCostFunction"的头文件部分

代价函数值的计算框架

本文设定代价函数为$F(A)$，残差块为$f_i(A)$，参量以矩阵形式表达，这里假设参量数量为3，则参量为A，$A=[a_0,a_1,a_2]^T$，观测数据为x_i和y_i，假设其理论数学关系符合三阶多项式，$y_i=a_0+a_1x_i+a_2x_i^2$。虽然代价函数$F(A)$的形式可以根据用户实际使用而不同，但残差块的平方和形式仍然在大量文献中被采用，如下。

$$F(A)=\sum_{i=1}^{m}(f_i(A))^2 \tag{1}$$

残差块$f_i(A)$也面临相同的情况，虽然可以根据用户使用情况而不同，但观测值与理论值之差的形式依然是广泛使用的形式，如下。

$$f_i(A) = y_i - (a_0 + a_1 x_i + a_2 x_i^2) \tag{2}$$

采用上述形式，则代价函数可以表示如下。

$$\begin{aligned} F(A) = &(y_0 - (a_0 + a_1 x_0 + a_2 x_0^2))^2 \\ &+ (y_1 - (a_0 + a_1 x_1 + a_2 x_1^2))^2 \\ &\vdots \\ &+ (y_m - (a_0 + a_1 x_m + a_2 x_m^2))^2 \end{aligned} \tag{3}$$

代价函数值的计算由函数`bool CostFunction`完成，由于该函数是核心功能，其名称必须固定，因此该函数作为虚基类"UserCostFunction"的纯虚函数进行声明，在继承类"PolyResidualBlockFunction"中进行定义。

梯度计算框架

代价函数的梯度(Gradient)即一阶导数，也称为雅各比矩阵(Jacobi Matrix)，记为$\nabla F(A)$，本文中简化表示为J，$\nabla F(A) \equiv J$。梯度是梯度下降法、牛顿法等方法的核心参量，直接参与下降方向和步长的计算。

在梯度下降法(Gradient descent method)中，梯度直接决定了下降方向$\mathbf{h}_{sd}$和步长$StepLength$，方程如下，

$$\mathbf{h}_{sd} = -F'(A) \tag{4}$$

$$StepLength = \alpha \mathbf{h}_{sd} \tag{5}$$

其中，α是因子。

梯度定义上通过计算代价函数的偏导数得到，

$$J = \begin{bmatrix} \frac{\partial F(A)}{\partial a_0} \\ \frac{\partial F(A)}{\partial a_1} \\ \frac{\partial F(A)}{\partial a_2} \end{bmatrix} \tag{6}$$

定义上，梯度的各个元素需要通过求导计算得到，但是实际应用中不一定能够获得导数，常用数值微分替代求导。

数值微分求导方法有多种，最基础的方法以代价函数的泰勒展开为基础，通过泰勒多项式的一阶项，达到在参量A附近近似的目标。

梯度计算由函数`bool DerivativesFunction`、`bool GetOneDerivative`和`void SetStepLength`共同完成。其中，`bool DerivativesFunction`总管所有函数，负责计算梯度矩阵，参量A由用户提供，梯度数据以容器"Vector"的形式保存，通过函数的形式参数`vector<double> &CostFunctionValues`以引用传递方式返回。

函数`bool GetOneDerivative`是梯度计算中使用的功能型函数，负责计算参数矩阵A中第i个参数的导数值，参数编号通过函数形式参数`int VarialbleID`导入，计算完成的导数值通过形式参数`double &theDerivativeValue`以引用传递的方式返回。具体的求导计算即在该函数中进行，本文使用一阶泰勒展开式近似代价函数，并计算导数值。导数计算方法如下，

$$\frac{\partial F(a_0,a_1,a_2)}{\partial a_0}_{|a_0=a_{0,i}} = \frac{1}{2h_0}(F((a_{0,i}+h_0),a_1,a_2) - F((a_{0,i}-h_0),a_1,a_2)) \tag{7}$$

但本文目前使用的计算方程如下，

$$\frac{\partial F(a_0,a_1,a_2)}{\partial a_0}_{|a_0=a_{0,i}} = \frac{1}{h_0}(F((a_{0,i}+h_0),a_1,a_2) - F(a_{0,i},a_1,a_2)) \tag{8}$$

其中，$a_{0,i}$表示第i次计算导数时，用户提供的参量A的a_0分量，$A=[a_{0,i},a_{1,i},a_{2,i}]^T$，$h_0$为步长$\mathbf{h}=[h_0,h_1,h_2]^T$的第一个分量。

函数`void SetStepLength`用来设定步长$\mathbf{h}=[h_0,h_1,h_2]^T$，虽然步长含有三个分量，但是在本程序中，设定步长一致，仅用一个变量代表。

Hessian计算框架

Hessian矩阵，记为H，是代价函数$F(A)$的二阶导数，$H=F''(A)$。实际计算中，代价函数的二阶导数可能难以求解，或者不存在，因此使用数值方法计算。

Hessian矩阵是一个$n\times n$矩阵，n为参数数量，其元素记为$\frac{\partial^2 F(A)}{\partial a_j \partial a_k}$，$H$中编号为$(j,k)$的元素的计算方程如下，

$$\frac{\partial^2 F(A)}{\partial a_j \partial a_k} = \sum_{i=1}^{m}(\frac{\partial f_i}{\partial a_j}(A)\frac{\partial f_i}{\partial a_k}(A) + \frac{\partial^2 f_i(A)}{\partial a_j \partial a_k}) \tag{9}$$

方程中包含一阶偏导数$\frac{\partial f_i}{\partial a_j}(A)$和二阶偏导数$\frac{\partial^2 f_i(A)}{\partial a_j \partial a_k}$，一阶偏导数的计算与最速下降法一样，使用一阶泰勒展开式近似，

$$\frac{\partial f_i}{\partial a_j}(A) = \frac{f_i(A;a_j+\Delta a_j) - f_i(A;a_j-\Delta a_j)}{2\Delta a_j} \tag{10}$$

其中，$(A;a_j+\Delta a_j)=(a_0,\cdots,a_j+\Delta a_j,\cdots,a_n)$。二阶偏导数的计算同样使用一阶泰勒展开式近似，

$$\frac{\partial^2 f_i(A)}{\partial a_j \partial a_k} = \frac{\frac{\partial f_i}{\partial a_j}(A;a_k+\Delta a_k) - \frac{\partial f_i}{\partial a_j}(A;a_k-\Delta a_k)}{2\Delta a_k} \tag{11}$$

将一阶偏导数带入方程，得到二阶偏导数的实际计算方程，

$$\frac{\partial^2 f_i(A)}{\partial a_j \partial a_k} = \frac{\frac{f_i(A;a_j+\Delta a_j,a_k+\Delta a_k)-f_i(A;a_j-\Delta a_j,a_k+\Delta a_k)}{2\Delta a_j} - \frac{f_i(A;a_j+\Delta a_j,a_k-\Delta a_k)-f_i(A;a_j-\Delta a_j,a_k-\Delta a_k)}{2\Delta a_j}}{2\Delta a_k} \tag{12}$$

化简得到，

$$\begin{aligned}&\frac{\partial^2 f_i(A)}{\partial a_j \partial a_k} = \\ &\frac{f_i(A;a_j+\Delta a_j,a_k+\Delta a_k) - f_i(A;a_j-\Delta a_j,a_k+\Delta a_k) - (f_i(A;a_j+\Delta a_j,a_k-\Delta a_k) - f_i(A;a_j-\Delta a_j,a_k-\Delta a_k))}{4\Delta a_k \Delta a_k}\end{aligned} \tag{13}$$

将上述公式带回Hessian矩阵元素的计算公式中，替换二阶偏导数部分，

$$\sum_{i=1}^{m}\frac{\partial^2 f_i(A)}{\partial a_j \partial a_k}=\frac{1}{4\Delta a_j \Delta a_k}(\sum_{i=1}^{m} f_i(A;a_j+\Delta a_j,a_k+\Delta a_k)-\sum_{i=1}^{m} f_i(A;a_j-\Delta a_j,a_k+\Delta a_k) \\ -\sum_{i=1}^{m} f_i(A;a_j+\Delta a_j,a_k-\Delta a_k)+\sum_{i=1}^{m} f_i(A;a_j-\Delta a_j,a_k-\Delta a_k)) \tag{14}$$

由此，可应用于实际编程计算的Hessian矩阵计算公式如下，

$$\frac{\partial^2 F(A)}{\partial a_j \partial a_k}=\sum_{i=1}^{m}\{\frac{f_i(A;a_j+\Delta a_j)-f_i(A;a_j-\Delta a_j)}{2\Delta a_j}\cdot\frac{f_i(A;a_k+\Delta a_k)-f_i(A;a_k-\Delta a_k)}{2\Delta a_k} \\ +\frac{1}{4\Delta a_j \Delta a_k}[f_i(A;a_j+\Delta a_j,a_k+\Delta a_k)-f_i(A;a_j-\Delta a_j,a_k+\Delta a_k) \\ -f_i(A;a_j+\Delta a_j,a_k-\Delta a_k)+f_i(A;a_j-\Delta a_j,a_k-\Delta a_k)]\} \tag{15}$$

令$f_i(A;a_j+\Delta a_j)=f_i(\Delta a_j)$，$f_i(A;a_j-\Delta a_j)=f_i(-\Delta a_j)$，
$f_i(A;a_j+\Delta a_j,a_k+\Delta a_k)=f_i(\Delta a_j,\Delta a_k)$，$f_i(A;a_j+\Delta a_j,a_k-\Delta a_k)=f_i(\Delta a_j,-\Delta a_k)$，
简化方程形式如下，

$$\frac{\partial^2 F(A)}{\partial a_j \partial a_k}=\sum_{i=1}^{m}\{\frac{f_i(\Delta a_j)-f_i(-\Delta a_j)}{2\Delta a_j}\cdot\frac{f_i(\Delta a_k)-f_i(-\Delta a_k)}{2\Delta a_k} \\ +\frac{1}{4\Delta a_j \Delta a_k}[f_i(\Delta a_j,\Delta a_k)-f_i(-\Delta a_j,\Delta a_k)-f_i(\Delta a_j,-\Delta a_k)+f_i(-\Delta a_j,-\Delta a_k)]\} \tag{16}$$

优化算法管理类

优化算法管理类包含两个部分，虚基类"UserOptimizationManager"和继承类"SteepestOptimizationManager"。