优化算法大作业

代码结构框架

函数范数及公式

范数公式

L0 范数

L0 范数表示向量中非零元素的个数。

$$\|x\|_0 = \text{number of non-zero entries in } x$$

L1 范数

L1 范数是向量元素绝对值之和。

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$

L2 范数

L2 范数或欧几里得范数是向量元素平方和的平方根。

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

优化问题的公式

最小二乘问题

最小二乘问题通常用于回归分析，目标是最小化预测值与实际值之间差的平方和。

$$f(x) = \frac{1}{2}\|Ax - b\|^2$$

逻辑回归

逻辑回归用于二分类问题，目标是最小化逻辑损失，通常用交叉熵表示。

$$f(x) = \frac{1}{m} \sum_{i=1}^{m} \log(1 + \exp(-y_i(a_i^T x)))$$

其中 y_i 是实例 i 的类标签，a_i 是实例 i 的特征向量。

步长选择策略

Armijo 步长策略

Armijo步长策略，也称为回溯线搜索，通过逐步缩小步长，直到满足一定条件。

```
# Armijo伪代码
initialize t = 1, beta < 1, alpha > 0
while f(x + t * delta_x) > f(x) + alpha * t * gradient(f)(x)^T * delta_x:
    t = beta * t
```

BB 步长策略

BB步长策略，也称为Barzilai-Borwein步长策略，通过考虑两次迭代间解的变化来自适应调整步长。该方法基于简单的梯度信息来估算有效的步长，通常可以加快收敛速度，尤其是在处理大规模优化问题时。
python

```
# BB伪代码
if first_iteration:
    t = initial_step_size  # 初始步长
else:
    s = x_new - x_old  # x的差异
    y = grad(x_new) - grad(x_old)  # 梯度的差异
    t = (s^T * s) / (s^T * y)  # 更新步长，利用梯度变化和解变化的比例
```

项目描述

本项目旨在开发一个软件包，解决复合优化问题：

$$\min_x \{f(x) + h(x)\}$$

其中 $f(x)$ 是一个可微分函数，$h(x)$ 是一个可以轻松计算近邻算子的函数。本项目特别关注使用梯度方法和适当的步长选择策略进行迭代更新。

迭代更新公式

更新使用的公式为：

$$x_{k+1} = \mathrm{prox}_h(x_k - t_k \nabla f(x_k))$$

这里，t_k 是每次迭代中选择的步长，根据不同的策略进行选择。

步长选择策略

步长选择包括：

- **Armijo步长**：通过满足Armijo条件来调整步长，确保每次迭代足够的下降。
- **BB步长**：利用Barzilai-Borwein方法来调整步长，特别针对梯度震荡问题进行优化。

核心部件

- **Function 类**：一个抽象基类，定义了函数的基本接口。
- **ProximalOperator 类**：一个抽象基类，定义了近邻算子的基本接口。
- **LeastSquares 类**：实现最小二乘问题的求解。
- **L1NormProx 类**：实现L1范数的近邻算子。
- **ProximalGradientOptimizer 类**：实现了近邻梯度方法（PGM）的优化器。

项目结果：

- **迭代次数**
- **目标函数值**
- **计算时间**

目录结构

- **PGM/**
  - **include/** *# 包含所有头文件*
    - `Function.h` *# Function类定义*
    - `ProximalOperator.h` *# ProximalOperator类定义*
    - `LeastSquares.h` *# LeastSquares类定义*
    - `L1NormProx.h` *# L1NormProx类定义*
    - `ProximalGradientOptimizer.h` *# ProximalGradientOptimizer类定义*
  - **src/** *# 包含所有源代码文件*
    - `Function.cpp` *# Function类实现*
    - `LeastSquares.cpp` *# LeastSquares类实现*
    - `L1NormProx.cpp` *# L1NormProx类实现*
    - `ProximalGradientOptimizer.cpp` *# ProximalGradientOptimizer类实现*
    - `main.cpp` *# 程序入口*

核心组件描述

Function 类

- **文件位置**：`include/Function.h` 和 `src/Function.cpp`
- **功能描述**：
  - 抽象基类，定义了函数的接口。
  - 包含纯虚函数 `evaluate` 用于计算函数值。

ProximalOperator 类

- **文件位置**：`include/ProximalOperator.h` 和 `src/L1NormProx.cpp`
- **功能描述**：
  - 抽象基类，定义了近邻算子的接口。
  - 包含函数 `apply` 用于应用算子。

LeastSquares 类

- **文件位置**：`include/LeastSquares.h` 和 `src/LeastSquares.cpp`
- **功能描述**：
    - 继承自 `Function` 类，实现最小二乘问题的求解。

L1NormProx 类

- **文件位置**：`include/L1NormProx.h` 和 `src/L1NormProx.cpp`
- **功能描述**：
    - 继承自 `ProximalOperator` 类，实现L1范数的近邻算子。

ProximalGradientOptimizer 类

- **文件位置**：`include/ProximalGradientOptimizer.h` 和 `src/ProximalGradientOptimizer.cpp`
- **功能描述**：
    - 实现了近邻梯度方法（PGM）的优化器。
    - 包含方法 `optimize` 用于执行优化。

main 函数

- **文件位置**：`src/main.cpp`
- **功能描述**：
    - 创建 `LeastSquares` 和 `L1NormProx` 对象。
    - 使用 `ProximalGradientOptimizer` 执行优化。
    - 输出优化结果。

结果汇总分析

迭代算法结束循环的容忍度设置为1e-3(通过梯度二范数，判断循环结束)

L1范数，最小二乘，Armoji准则

编译命令

```
g++ -o pgm_l1_ls_arm_small PGM/main.cpp PGM/src/LeastSquares.cpp PGM/src/L1NormProx.cpp PGM/src/L2
```

```
g++ -o pgm_l1_ls_arm_big PGM/main.cpp PGM/src/LeastSquares.cpp PGM/src/L1NormProx.cpp PGM/src/L2No
```

简单数据算法验证

1.参数设置

矩阵 A：

矩阵 A 是一个 2x2 的矩阵，具体值如下：

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 1 \end{bmatrix}$$

向量 b:

向量 b 是一个 2x1 的向量，具体值如下：

$$b = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

初始向量 x:

初始向量 x 也是一个 2x1 的向量，用作优化算法的起始点，具体值如下：

$$x = \begin{bmatrix} 1000 \\ 2000 \end{bmatrix}$$

2.运行结果

迭代次数 (Iterations): 75

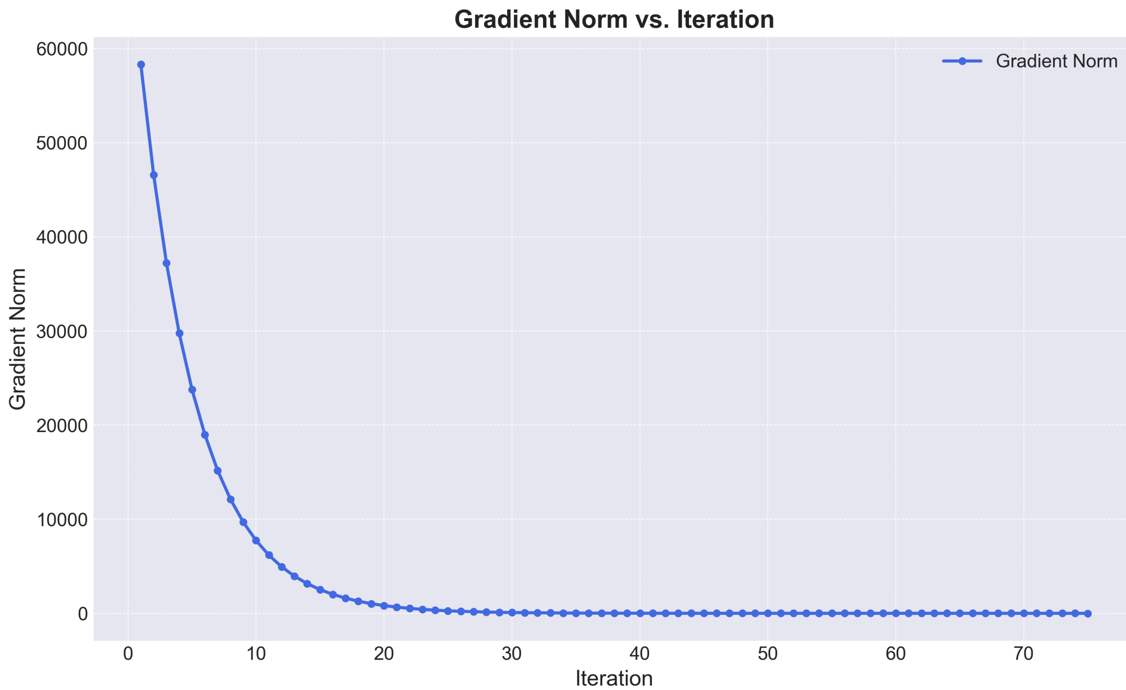
目标值 (Objective Value): 1.5995

解 (Solution): [1.1985, -0.3998]

计算时间 (CPU Time): 0 秒

3.算法收敛情况

算法收敛情况如下图所示：



收敛速度较快，在70左右达到了容忍度的要求，二范数梯度呈现下降趋势。

随机数据算法验证

1.参数设置

矩阵 A 是一个大小为 512×512 的矩阵，用于存储随机生成的实数值。每个元素 $A_{i,j}$ 都从一个均匀分布的范围 $[-10, 10]$ 中随机选取。矩阵定义如下：

$$A \in \mathbb{R}^{512 \times 512}$$

向量 b:

向量 b 是一个长度为 512 的向量，其元素也是从同一均匀分布 $[-10, 10]$ 中随机生成的。向量定义如下：

$$b \in \mathbb{R}^{512}$$

初始猜测向量 x:

初始猜测向量 x 是一个长度为 512 的向量，初始时所有元素都设置为 0。这个向量用于开始某种优化算法的计算。向量定义如下：

$$x = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{512}$$

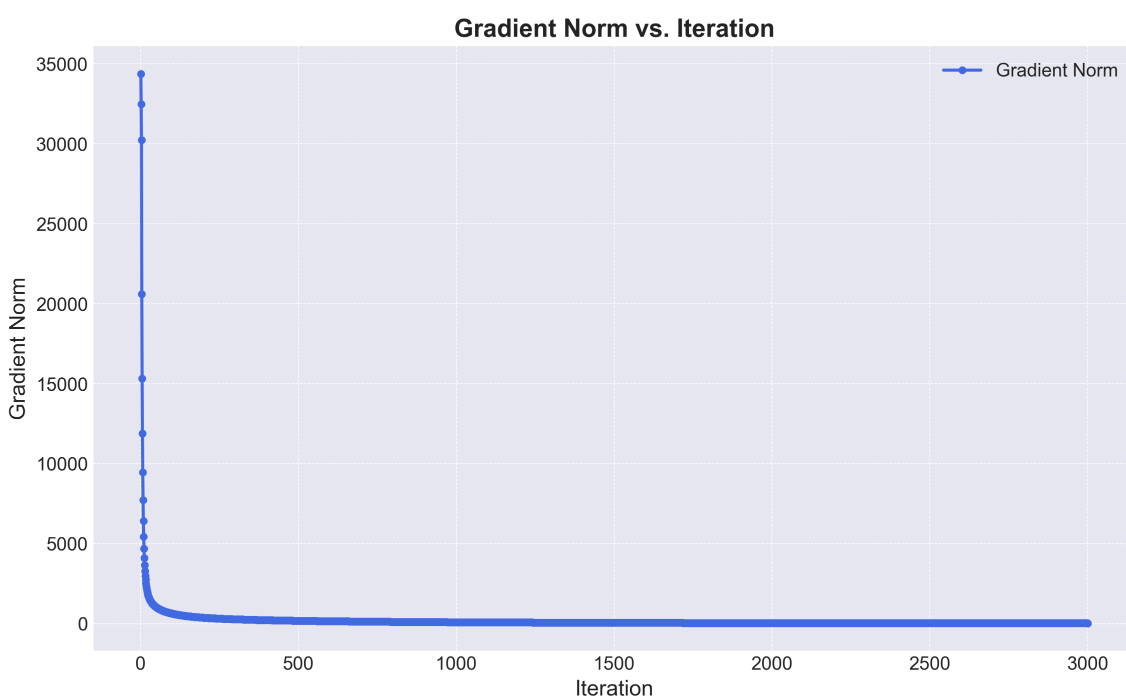
2.运行结果

迭代次数 (Iterations): 3000

目标值 (Objective Value): 194.949

计算时间 (CPU Time): 10.982 秒

3.算法收敛情况
算法收敛情况如下图所示：
3.算法收敛情况
算法收敛情况如下图所示：



收敛速度很快，但在最后迭代结束都没有达到容忍度要求，二范数梯度呈现下降趋势。

L1范数，最小二乘，bb准则

编译命令

```
g++ -o pgm_l1_ls_bb_small PGM/main.cpp PGM/src/LeastSquares.cpp PGM/src/L1NormProx.cpp PGM/src/L2N
```

```
g++ -o pgm_l1_ls_bb_big PGM/main.cpp PGM/src/LeastSquares.cpp PGM/src/L1NormProx.cpp PGM/src/L2Nor
```

简单数据算法验证

1.参数设置

矩阵 A：

矩阵 A 是一个 2x2 的矩阵，具体值如下：

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 1 \end{bmatrix}$$

向量 b:

向量 b 是一个 2x1 的向量，具体值如下：

$$b = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

初始向量 x:

初始向量 x 也是一个 2x1 的向量，用作优化算法的起始点，具体值如下：

$$x = \begin{bmatrix} 1000 \\ 2000 \end{bmatrix}$$

2.运行结果

迭代次数 (Iterations): 7

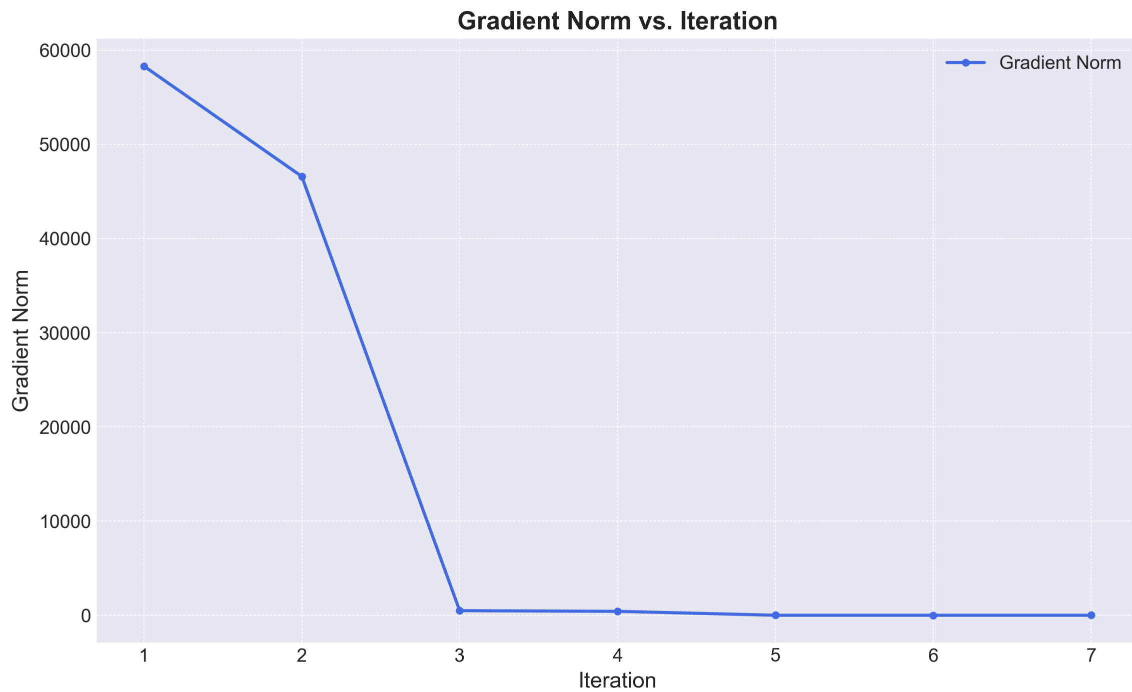
目标值 (Objective Value): 1.5995

解 (Solution): [1.1985, -0.3998]

计算时间 (CPU Time): 0 秒

3.算法收敛情况

算法收敛情况如下图所示：



收敛速度很快，在第7步左右达到了容忍度的要求，二范数梯度呈现下降趋势。

随机数据算法验证

1.参数设置

矩阵 A 是一个大小为 512×512 的矩阵，用于存储随机生成的实数值。每个元素 $A_{i,j}$ 都从一个均匀分布的范围 $[-10, 10]$ 中随机选取。矩阵定义如下：

$$A \in \mathbb{R}^{512 \times 512}$$

向量 b:

向量 b 是一个长度为 512 的向量，其元素也是从同一均匀分布 $[-10, 10]$ 中随机生成的。向量定义如下：

$$b \in \mathbb{R}^{512}$$

初始猜测向量 x:

初始猜测向量 x 是一个长度为 512 的向量，初始时所有元素都设置为 0。这个向量用于开始某种优化算法的计算。向量定义如下：

$$x = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{512}$$

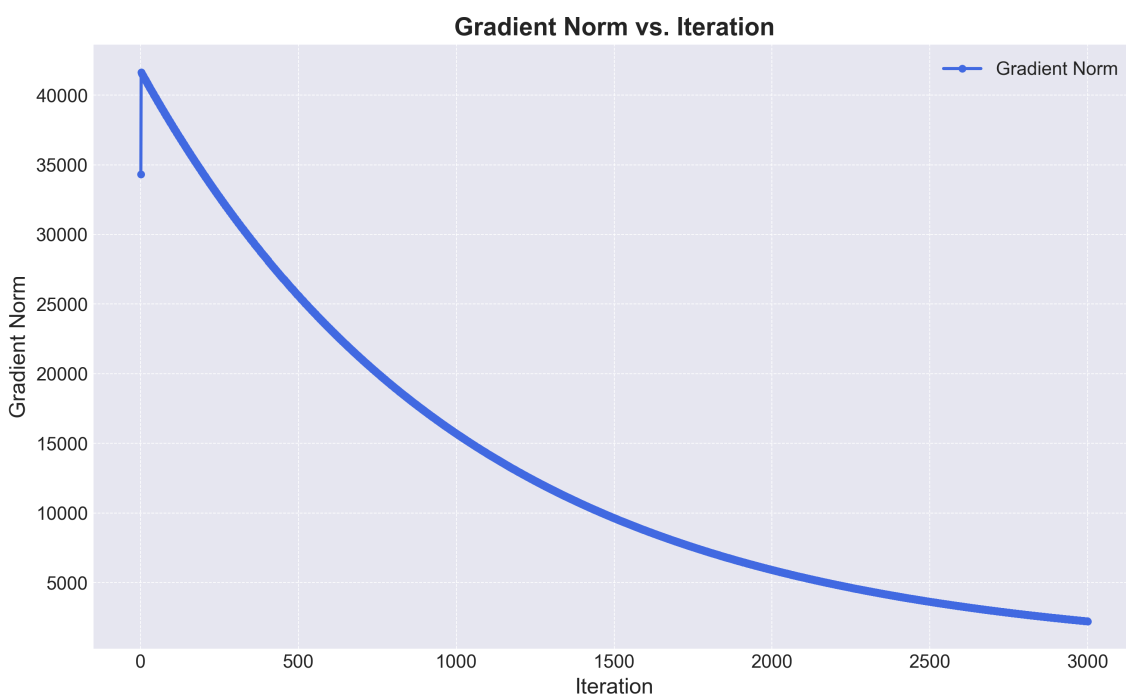
2.运行结果

迭代次数 (Iterations): 3000

目标值 (Objective Value): 218.916

计算时间 (CPU Time): 5.512 秒

3.算法收敛情况
算法收敛情况如下图所示：



收敛速度较慢，但在最后迭代结束都没有达到容忍度要求，二范数梯度呈现下降趋势，优化进行不彻底，可能是步长调整策略存在问题，后续在逻辑回归目标函数中对于BB算法步长调整策略进行修改。

尝试数据	tk选取	Iter次数	min f + g	CPU_TIME
small	Armijo	76	1.5995	0 s
small	BB-step	7	1.5995	0 s
big	Armijo	3000	194.949	10.982 s
big	BB-step	3000	218.916	5.512 s

L2范数，最小二乘，Armoji准则

编译命令

```
g++ -o pgm_l1_ls_arm_small PGM/main.cpp PGM/src/LeastSquares.cpp PGM/src/L1NormProx.cpp PGM/src/L2
```

```
g++ -o pgm_l1_ls_arm_big PGM/main.cpp PGM/src/LeastSquares.cpp PGM/src/L1NormProx.cpp PGM/src/L2No
```

简单数据算法验证

1.参数设置
***矩阵 A*：**
矩阵 A 是一个 2x2 的矩阵，具体值如下：

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 1 \end{bmatrix}$$

向量 b:
向量 b 是一个 2x1 的向量，具体值如下：

$$b = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

初始向量 x:
初始向量 x 也是一个 2x1 的向量，用作优化算法的起始点，具体值如下：

$$x = \begin{bmatrix} 1000 \\ 2000 \end{bmatrix}$$

2.运行结果

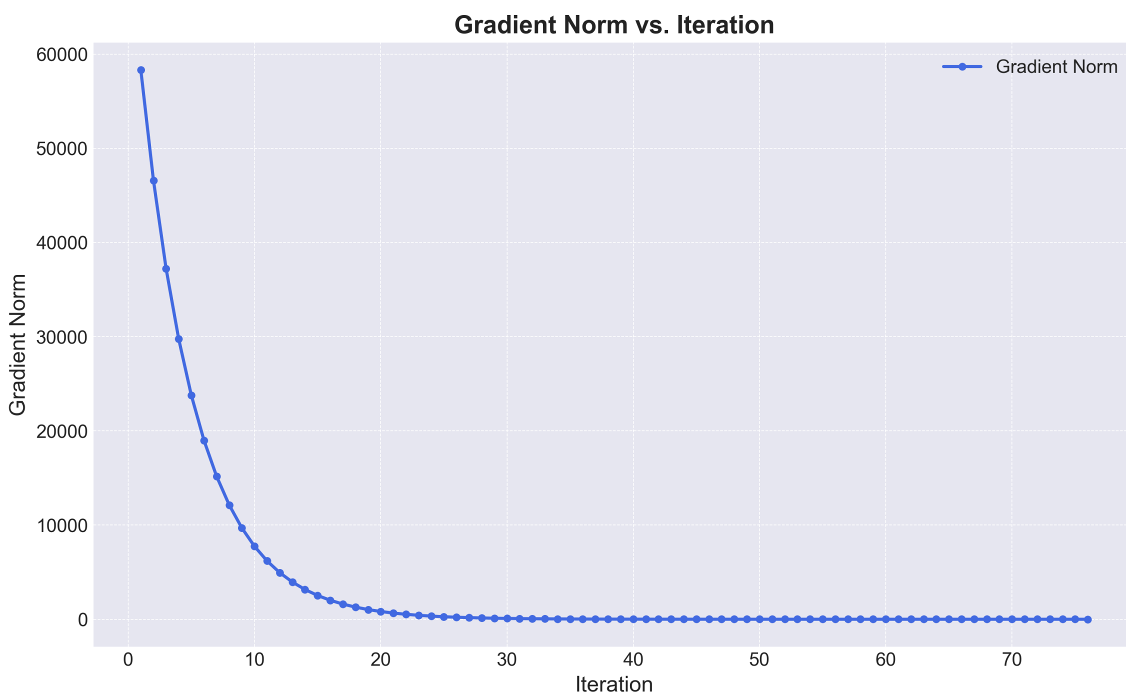
迭代次数 (Iterations): 76

目标值 (Objective Value): 1.26468

解 (Solution): [1.19982 -0.399799]

计算时间 (CPU Time): 0.013 秒

3.算法收敛情况
算法收敛情况如下图所示：



小规模问题收敛速度较快，能比较好的收敛。

随机数据算法验证

1.参数设置

矩阵 A 是一个大小为 512×512 的矩阵，用于存储随机生成的实数值。每个元素 $A_{i,j}$ 都从一

个均匀分布的范围 $[-10, 10]$ 中随机选取。矩阵定义如下：

$$A \in \mathbb{R}^{512\times 512}$$

向量 b:

向量 b 是一个长度为 512 的向量，其元素也是从同一均匀分布 $[-10, 10]$ 中随机生成的。向量定义如下：

$$b \in \mathbb{R}^{512}$$

初始猜测向量 x:

初始猜测向量 x 是一个长度为 512 的向量，初始时所有元素都设置为 0。这个向量用于开始某种优化算法的计算。向量定义如下：

$$x = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{512}$$

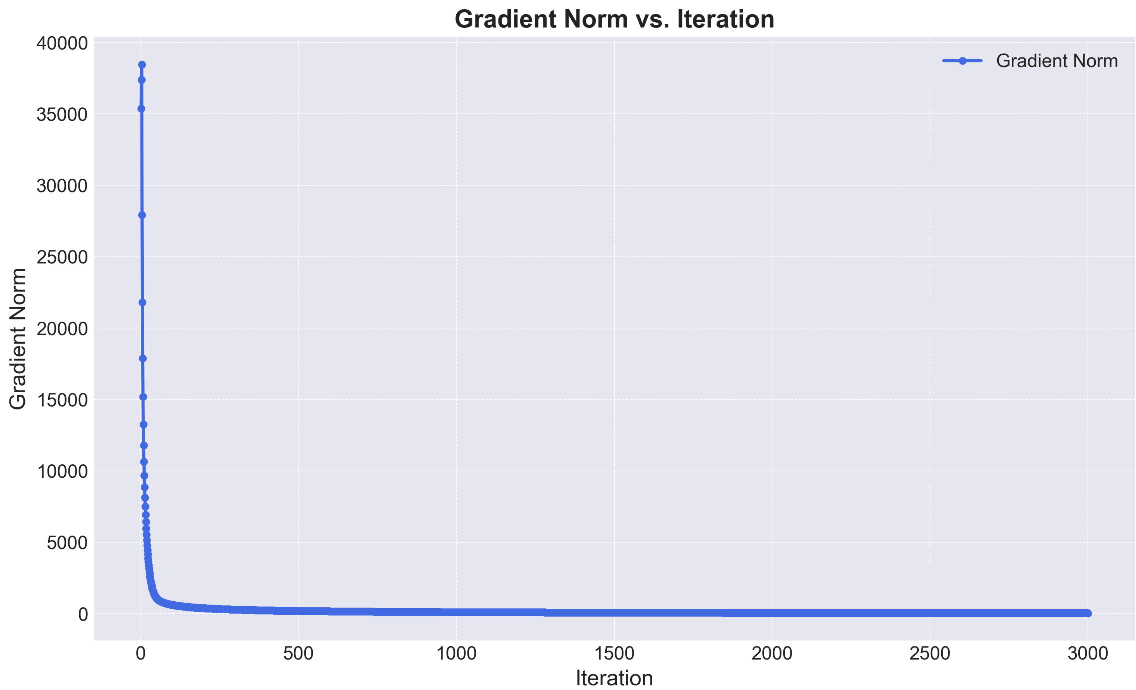
2.运行结果

迭代次数 (Iterations): 3000

目标值 (Objective Value): 54.0401

计算时间 (CPU Time): 11.127 秒

3.算法收敛情况
算法收敛情况如下图所示：



收敛速度较快，性能比较好的收敛。

L2范数，最小二乘，bb准则

编译命令

```
g++ -o pgm_l2_ls_bb_small PGM/main.cpp PGM/src/LeastSquares.cpp PGM/src/L1NormProx.cpp PGM/src/L2N
```

```
g++ -o pgm_l2_ls_bb_big PGM/main.cpp PGM/src/LeastSquares.cpp PGM/src/L1NormProx.cpp PGM/src/L2Nor
```

简单数据算法验证

1.参数设置

***矩阵 A*：**

矩阵 A 是一个 2x2 的矩阵，具体值如下：

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 1 \end{bmatrix}$$

向量 b:
向量 b 是一个 2x1 的向量，具体值如下：

$$b = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

初始向量 x:
初始向量 x 也是一个 2x1 的向量，用作优化算法的起始点，具体值如下：

$$x = \begin{bmatrix} 1000 \\ 2000 \end{bmatrix}$$

2.运行结果

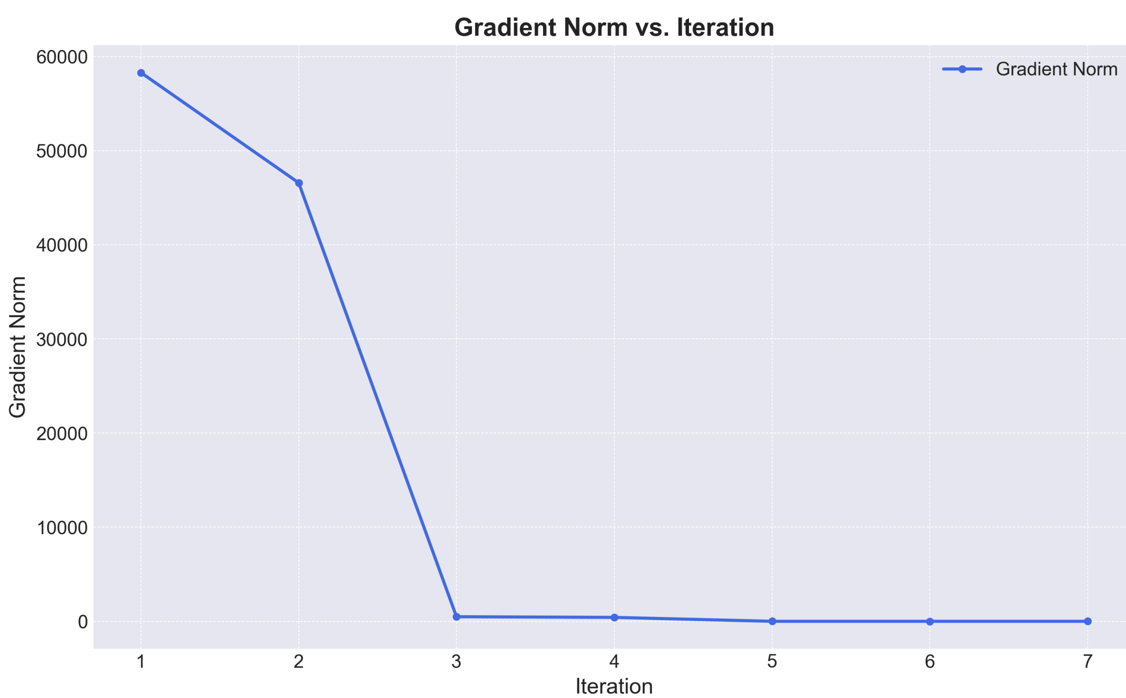
迭代次数 (Iterations): 7

目标值 (Objective Value): 1.26468

解 (Solution): [1.19982 -0.399799]

计算时间 (CPU Time): 0 秒

3.算法收敛情况
算法收敛情况如下图所示：



小规模问题，收敛速度较快，性能比较好的收敛。

随机数据算法验证

1.参数设置

矩阵 A 是一个大小为 512×512 的矩阵，用于存储随机生成的实数值。每个元素 $A_{i,j}$ 都从一个均匀分布的范围 $[-10, 10]$ 中随机选取。矩阵定义如下：

$$A \in \mathbb{R}^{512\times512}$$

向量 b:

向量 b 是一个长度为 512 的向量，其元素也是从同一均匀分布 $[-10, 10]$ 中随机生成的。向量定义如下：

$$b \in \mathbb{R}^{512}$$

初始猜测向量 x:

初始猜测向量 x 是一个长度为 512 的向量，初始时所有元素都设置为 0。这个向量用于开始某种优化算法的计算。向量定义如下：

$$x = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{512}$$

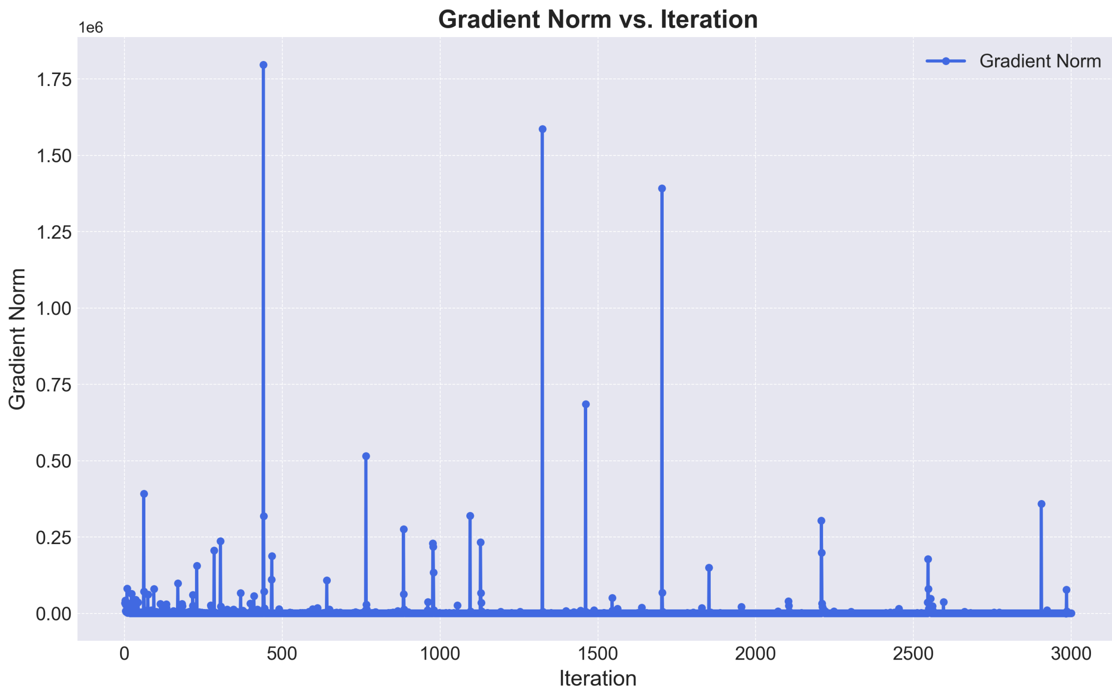
2.运行结果

迭代次数 (Iterations): 3000

目标值 (Objective Value): 27.1448

计算时间 (CPU Time): 5.646 秒

3.算法收敛情况
算法收敛情况如下图所示：



出现了震荡的现象，因此在后续逻辑函数中，对于bb策略的步长选择方式进行了调整，解决了这个问题。

尝试数据	tk选取	Iter次数	min f + g	CPU_TIME
small	Armijo	76	1.26468	0.02 s
small	BB-step	7	1.26468	0 s
big	Armijo	3000	54.0401	1.127 s
big	BB-step	3000	27.1448	5.646 s

L0范数，最小二乘，Armoji准则

编译命令

```
g++ -o pgm_l0_ls_arm_small PGM/main.cpp PGM/src/LeastSquares.cpp PGM/src/L1NormProx.cpp PGM/src/L2
```

```
g++ -o pgm_l0_ls_arm_big PGM/main.cpp PGM/src/LeastSquares.cpp PGM/src/L1NormProx.cpp PGM/src/L2No
```

简单数据算法验证

1.参数设置

矩阵 A：

矩阵 A 是一个 2x2 的矩阵，具体值如下：

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 1 \end{bmatrix}$$

向量 b:

向量 b 是一个 2x1 的向量，具体值如下：

$$b = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

初始向量 x:

初始向量 x 也是一个 2x1 的向量，用作优化算法的起始点，具体值如下：

$$x = \begin{bmatrix} 1000 \\ 2000 \end{bmatrix}$$

2.运行结果

迭代次数 (Iterations): 75

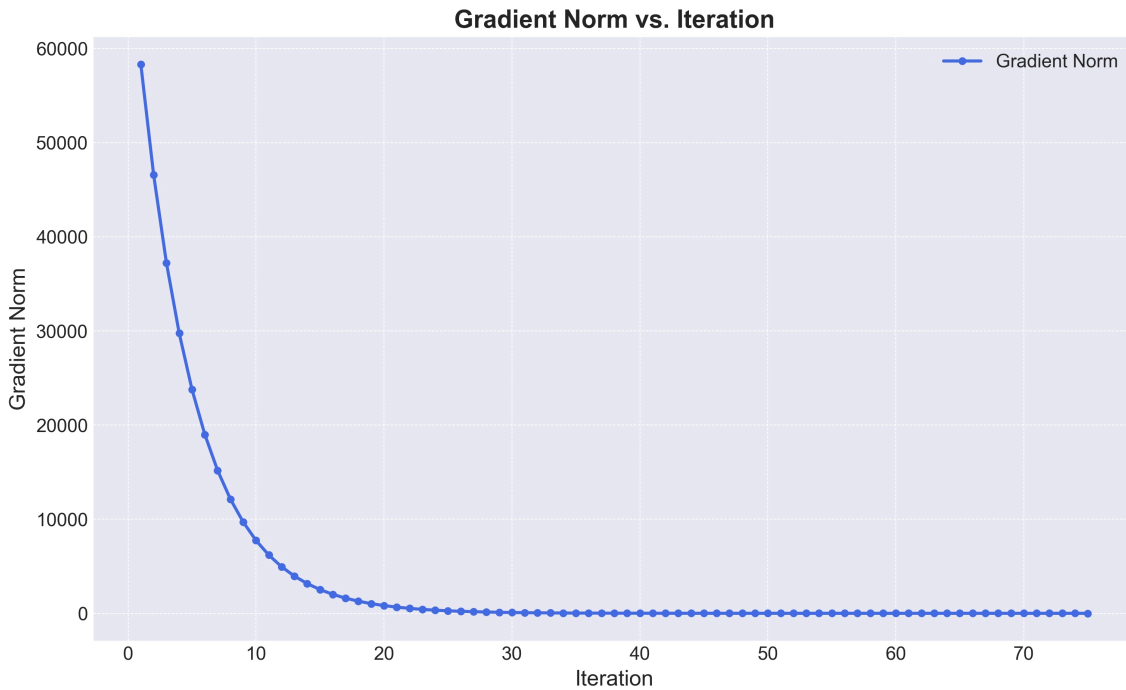
目标值 (Objective Value): 2

解 (Solution): [1.19994 -0.400094]

计算时间 (CPU Time): 0 秒

3.算法收敛情况

算法收敛情况如下图所示：



在76步的时候收敛，二范数性质较好。

随机数据算法验证

1.参数设置

矩阵 A 是一个大小为 512×512 的矩阵，用于存储随机生成的实数值。每个元素 $A_{i,j}$ 都从一个均匀分布的范围 $[-10, 10]$ 中随机选取。矩阵定义如下：

$$A \in \mathbb{R}^{512\times512}$$

向量 b:

向量 b 是一个长度为 512 的向量，其元素也是从同一均匀分布 $[-10, 10]$ 中随机生成的。向量定义如下：

$$b \in \mathbb{R}^{512}$$

初始猜测向量 x:

初始猜测向量 x 是一个长度为 512 的向量，初始时所有元素都设置为 0。这个向量用于开始某种优化算法的计算。向量定义如下：

$$
x = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{512}
$$

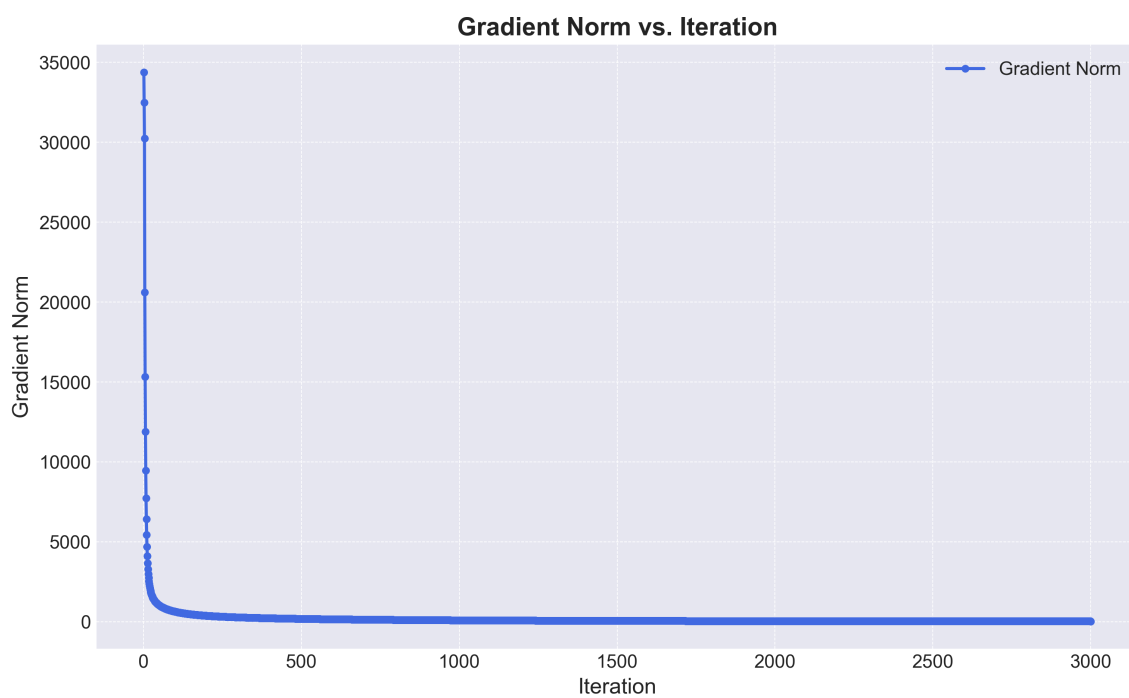
2.运行结果

迭代次数 (Iterations): 3000

目标值 (Objective Value): 530.192

计算时间 (CPU Time): 10.961 秒

3.算法收敛情况
算法收敛情况如下图所示：



能够收敛，但是因为问题规模问题在达到3000步的时候还是没有满足容忍度的限制。

L0范数，最小二乘，bb准则

编译命令

```
g++ -o pgm_l2_ls_bb_small PGM/main.cpp PGM/src/LeastSquares.cpp PGM/src/L1NormProx.cpp PGM/src/L2N
```

```
g++ -o pgm_l2_ls_bb_big PGM/main.cpp PGM/src/LeastSquares.cpp PGM/src/L1NormProx.cpp PGM/src/L2Nor
```

简单数据算法验证

1.参数设置

矩阵 A：

矩阵 A 是一个 2x2 的矩阵，具体值如下：

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 1 \end{bmatrix}$$

向量 b:

向量 b 是一个 2x1 的向量，具体值如下：

$$b = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

初始向量 x:

初始向量 x 也是一个 2x1 的向量，用作优化算法的起始点，具体值如下：

$$x = \begin{bmatrix} 1000 \\ 2000 \end{bmatrix}$$

2.运行结果

迭代次数 (Iterations): 7

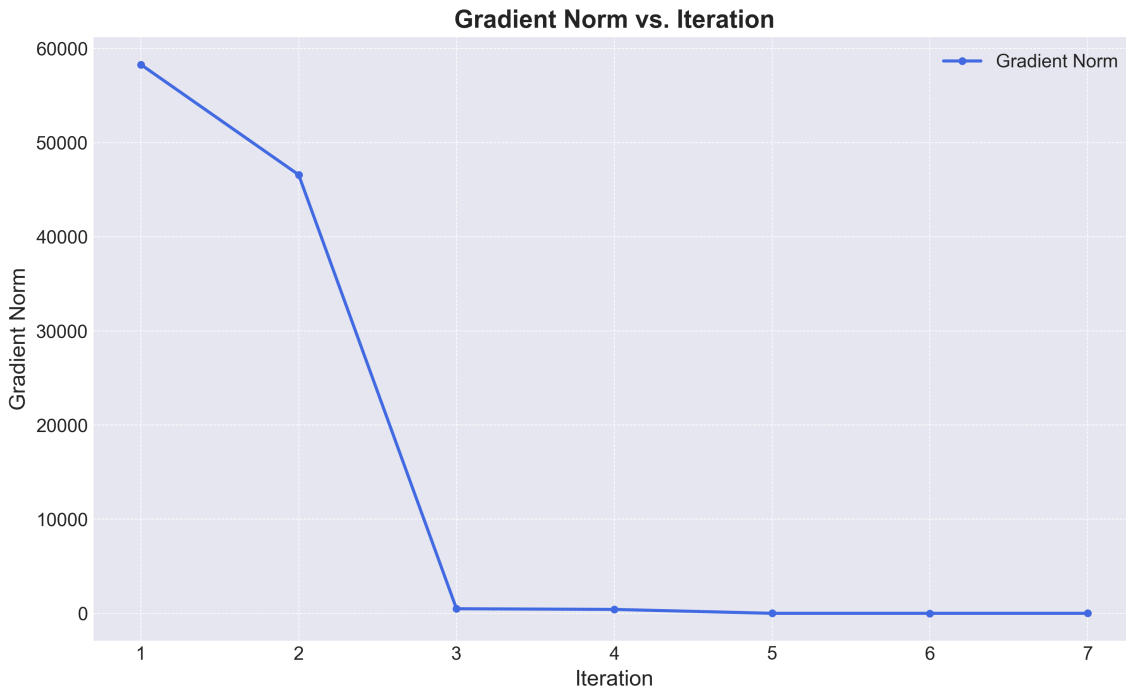
目标值 (Objective Value): 2

解 (Solution): [1.2 -0.4]

计算时间 (CPU Time): 0 秒

3.算法收敛情况

算法收敛情况如下图所示：



小规模问题，能够收敛，且只用了7步完成收敛。

随机数据算法验证

1.参数设置

矩阵 A 是一个大小为 512×512 的矩阵，用于存储随机生成的实数值。每个元素 $A_{i,j}$ 都从一个均匀分布的范围 $[-10, 10]$ 中随机选取。矩阵定义如下：

$$A \in \mathbb{R}^{512\times512}$$

向量 b:

向量 b 是一个长度为 512 的向量，其元素也是从同一均匀分布 $[-10, 10]$ 中随机生成的。向量定义如下：

$$b \in \mathbb{R}^{512}$$

初始猜测向量 x:

初始猜测向量 x 是一个长度为 512 的向量，初始时所有元素都设置为 0。这个向量用于开始某种优化算法的计算。向量定义如下：

$$
x = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{512}
$$

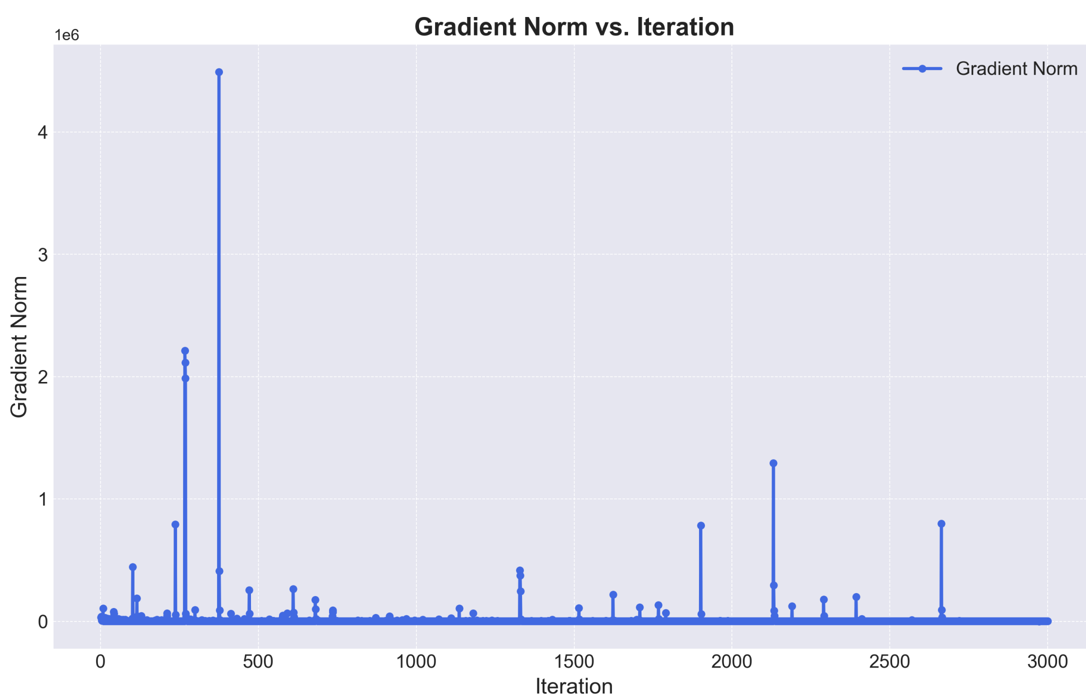
2.运行结果

迭代次数 (Iterations): 3000

目标值 (Objective Value): 596.487

计算时间 (CPU Time): 5.646 秒

3.算法收敛情况
算法收敛情况如下图所示：



由于BB策略的步长选择问题，出现了震荡的现象，后续在逻辑回归函数中解决了这个问题。

尝试数据	tk选取	Iter次数	min f + g	CPU_TIME
small	Armijo	75	2	0.02 s

尝试数据	tk选取	Iter次数	min f + g	CPU_TIME
small	BB-step	7	2	0 s
big	Armijo	3000	530.192	10.961 s
big	BB-step	3000	596.487	5.646 s

针对使用BB算法求解较大规模问题，梯度二范数出现震荡的问题，通过调整BB策略的步长选择方式解决了这个问题。

逻辑回归函数的验证

解决在最小二乘使用bb策略出现的震荡问题。

L1范数，逻辑回归，Armoji准则

编译命令

```
g++ -o pgm_l1_lr_arm_big PGM/main.cpp PGM/src/LeastSquares.cpp PGM/src/L1NormProx.cpp PGM/src/L2No
```

随机数据算法验证

1.参数设置

矩阵 A 是一个大小为 512×512 的矩阵，用于存储随机生成的实数值。每个元素 $A_{i,j}$ 都从一个均匀分布的范围 $[-10, 10]$ 中随机选取。矩阵定义如下：

$$A \in \mathbb{R}^{512 \times 512}$$

向量 b:

向量 b 是一个长度为 512 的向量，其元素也是从同一均匀分布 $[-10, 10]$ 中随机生成的。向量定义如下：

$$b \in \mathbb{R}^{512}$$

初始猜测向量 x:

初始猜测向量 x 是一个长度为 512 的向量，初始时所有元素都设置为 0。这个向量用于开始某种优化算法的计算。向量定义如下：

$$x = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{512}$$

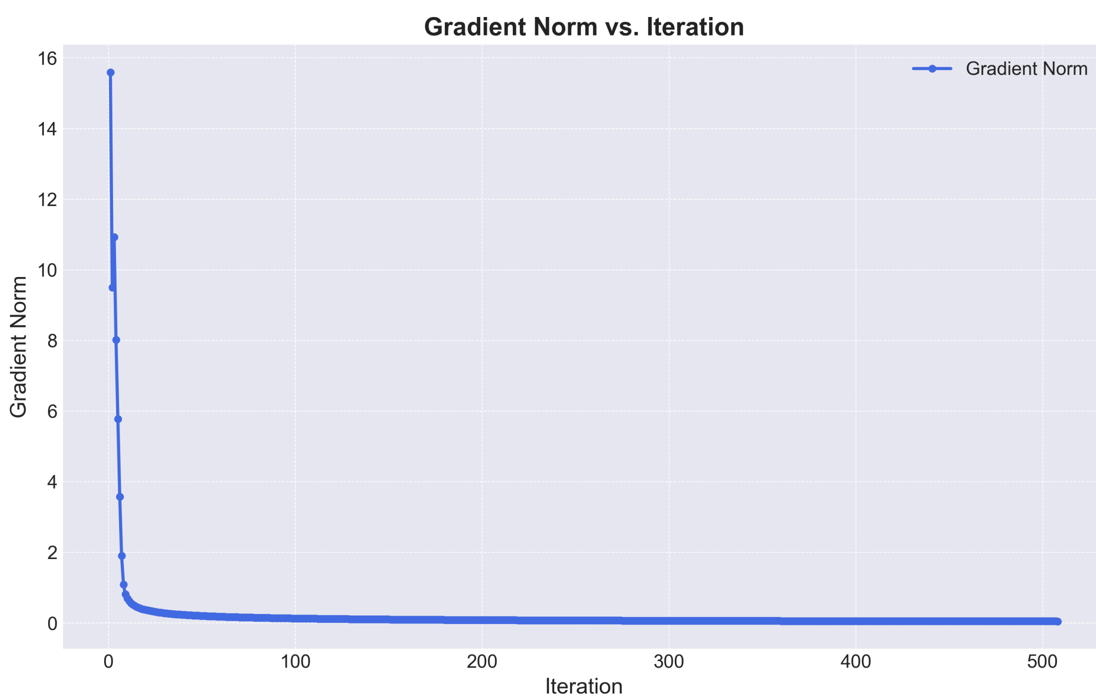
2.运行结果

迭代次数 (Iterations): 374

目标值 (Objective Value): 21.2822

计算时间 (CPU Time): 1.432 秒

3.算法收敛情况
算法收敛情况如下图所示：



能够平稳的收敛，大致在374步的时候达到容忍度要求，求解实现较长。

L1范数，最小二乘，bb准则

编译命令

```
g++ -o pgm_l1_lr_bb_big PGM/main.cpp PGM/src/LeastSquares.cpp PGM/src/L1NormProx.cpp PGM/src/L2Nor
```

随机数据算法验证

1.参数设置

矩阵 A 是一个大小为 512×512 的矩阵，用于存储随机生成的实数值。每个元素 $A_{i,j}$ 都从一个均匀分布的范围 $[-10, 10]$ 中随机选取。矩阵定义如下：

$$A \in \mathbb{R}^{512 \times 512}$$

向量 b:

向量 b 是一个长度为 512 的向量，其元素也是从同一均匀分布 $[-10, 10]$ 中随机生成的。向量定义如下：

$$b \in \mathbb{R}^{512}$$

初始猜测向量 x:

初始猜测向量 x 是一个长度为 512 的向量，初始时所有元素都设置为 0。这个向量用于开始某种优化算法的计算。向量定义如下：

$$x = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{512}$$

2.运行结果

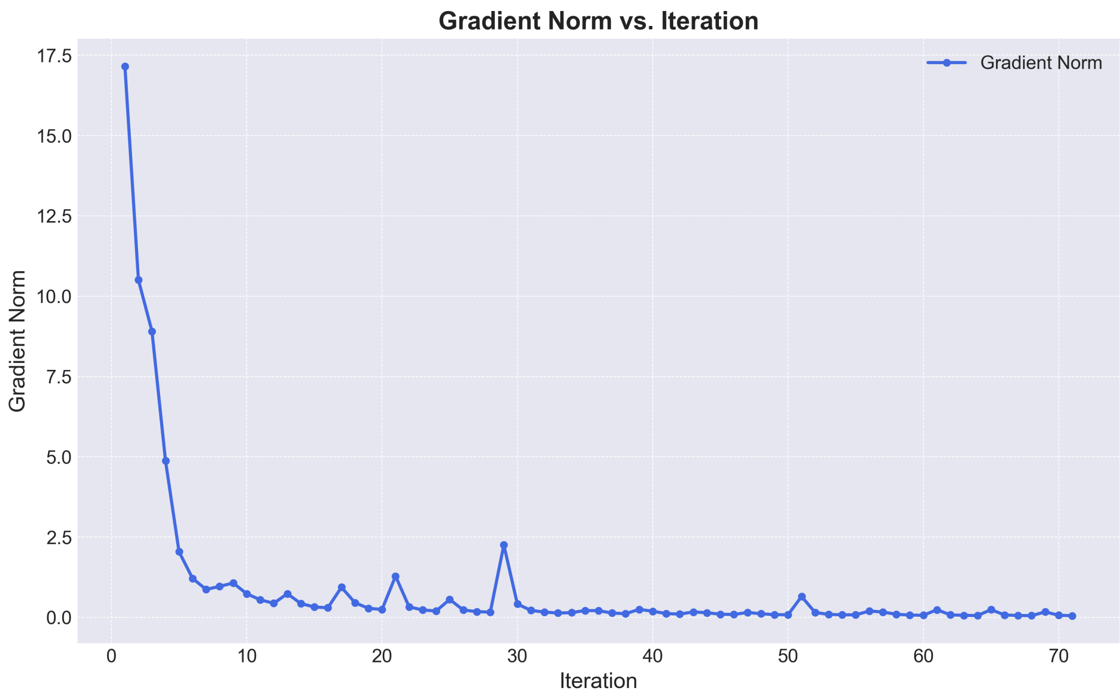
迭代次数 (Iterations): 90

目标值 (Objective Value): 18.4637

计算时间 (CPU Time): 0.287 秒

3.算法收敛情况

算法收敛情况如下图所示：



能够少许震荡，大致在90步的时候达到容忍度要求，求解时间较短，解决了在最小二乘函数中的震荡问题。

尝试数据	tk选取	Iter次数	min f + g	CPU_TIME
big	Armijo	374	21.2822	1.432 s
big	BB-step	90	18.4637	0.287 s

L2范数，最小二乘，Armoji准则

编译命令

```
g++ -o pgm_l2_lr_arm_big PGM/main.cpp PGM/src/LeastSquares.cpp PGM/src/L1NormProx.cpp PGM/src/L2No
```

随机数据算法验证

1.参数设置

矩阵 A 是一个大小为 512×512 的矩阵，用于存储随机生成的实数值。每个元素 $A_{i,j}$ 都从一个均匀分布的范围 $[-10, 10]$ 中随机选取。矩阵定义如下：

$$A \in \mathbb{R}^{512\times512}$$

向量 b:

向量 b 是一个长度为 512 的向量，其元素也是从同一均匀分布 $[-10, 10]$ 中随机生成的。向量定义如下：

$$b \in \mathbb{R}^{512}$$

初始猜测向量 x:

初始猜测向量 x 是一个长度为 512 的向量，初始时所有元素都设置为 0。这个向量用于开始某种优化算法的计算。向量定义如下：

$$x = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{512}$$

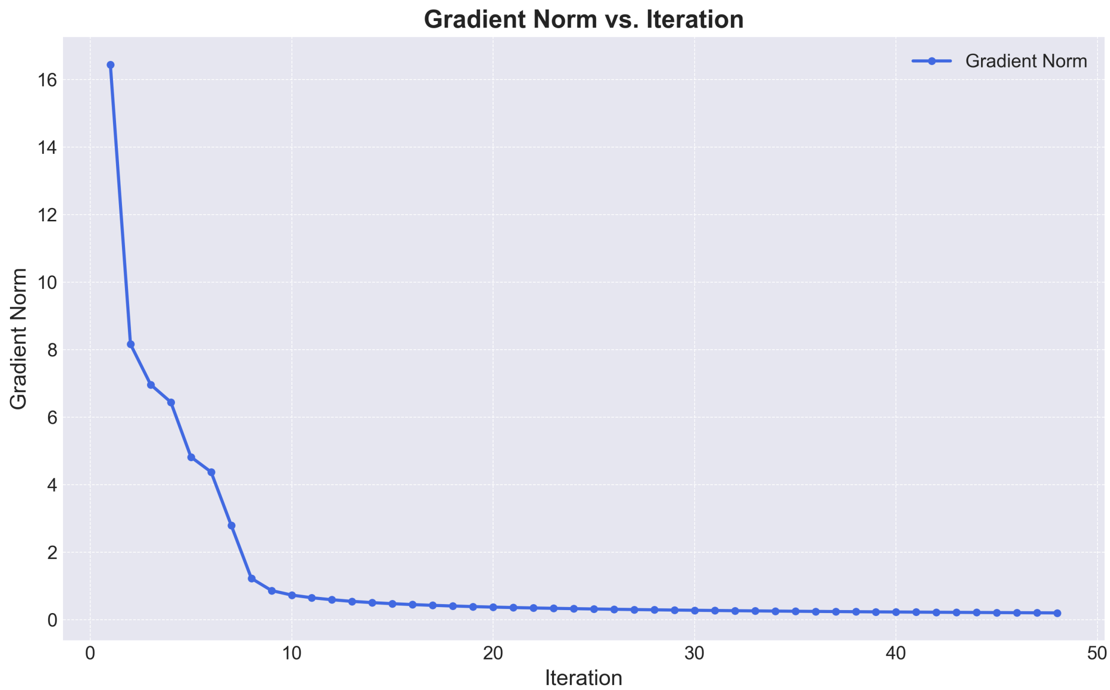
2.运行结果

迭代次数 (Iterations): 463

目标值 (Objective Value): 21.2355

计算时间 (CPU Time): 1.236 秒

3.算法收敛情况



大致在463步的时候达到容忍度要求，求解时间较短，解决了在最小二乘函数中的震荡问题。

L2范数，最小二乘，bb准则

编译命令

```
g++ -o pgm_l2_lr_bb_big PGM/main.cpp PGM/src/LeastSquares.cpp PGM/src/L1NormProx.cpp PGM/src/L2Nor
```

随机数据算法验证

1.参数设置

矩阵 A 是一个大小为 512×512 的矩阵，用于存储随机生成的实数值。每个元素 $A_{i,j}$ 都从一个均匀分布的范围 $[-10, 10]$ 中随机选取。矩阵定义如下：

$$A \in \mathbb{R}^{512\times 512}$$

向量 b:

向量 b 是一个长度为 512 的向量，其元素也是从同一均匀分布 $[-10, 10]$ 中随机生成的。向量

定义如下：

$$b \in \mathbb{R}^{512}$$

初始猜测向量 x:

初始猜测向量 x 是一个长度为 512 的向量，初始时所有元素都设置为 0。这个向量用于开始某种优化算法的计算。向量定义如下：

$$x = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{512}$$

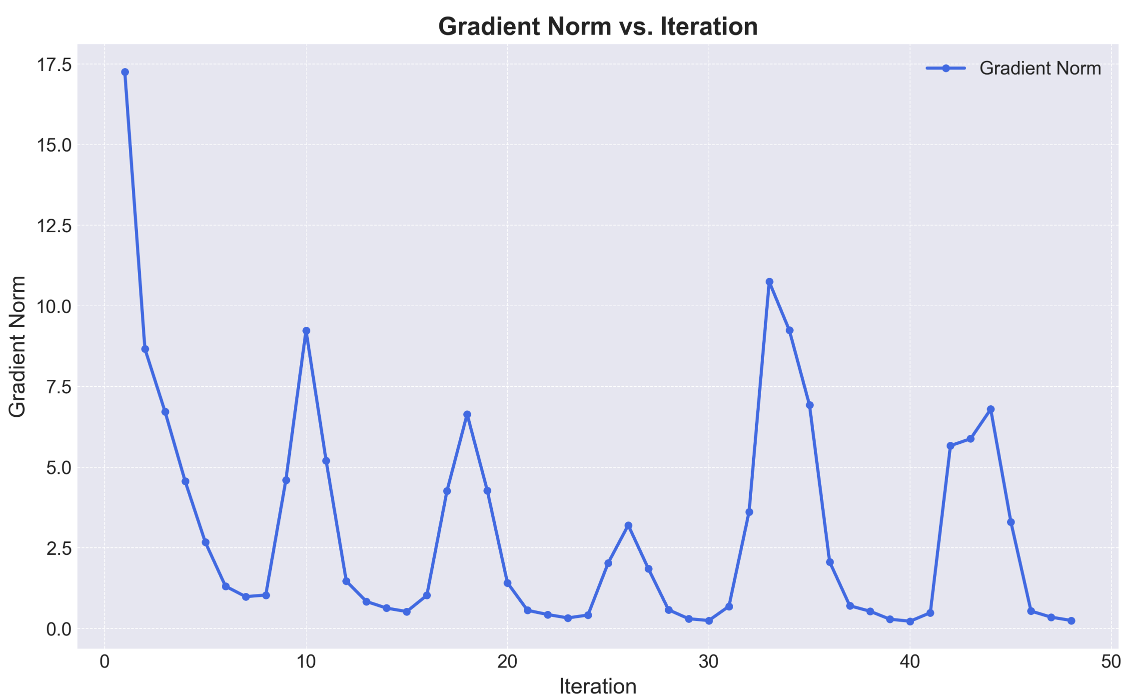
2.运行结果

迭代次数 (Iterations): 66

目标值 (Objective Value): 19.1149

计算时间 (CPU Time): 0.406 秒

3.算法收敛情况



有少许震荡，大致在66步的达到容忍度条件，可能容忍度条件比较小，步长选择没有特别调整，梯度并没有一直下降。

尝试数据	tk选取	Iter次数	min f + g	CPU_TIME
big	Armijo	463	21.2355	1.236 s
big	BB-step	66	19.1149	0.406 s

L0范数，最小二乘，Armoji准则

编译命令

```
g++ -o pgm_l0_lr_arm_big PGM/main.cpp PGM/src/LeastSquares.cpp PGM/src/L1NormProx.cpp PGM/src/L2No
```

随机数据算法验证

1.参数设置

矩阵 A 是一个大小为 512×512 的矩阵，用于存储随机生成的实数值。每个元素 $A_{i,j}$ 都从一个均匀分布的范围 $[-10, 10]$ 中随机选取。矩阵定义如下：

$$A \in \mathbb{R}^{512\times512}$$

向量 b:

向量 b 是一个长度为 512 的向量，其元素也是从同一均匀分布 $[-10, 10]$ 中随机生成的。向量定义如下：

$$b \in \mathbb{R}^{512}$$

初始猜测向量 x:

初始猜测向量 x 是一个长度为 512 的向量，初始时所有元素都设置为 0。这个向量用于开始某种优化算法的计算。向量定义如下：

$$x = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{512}$$

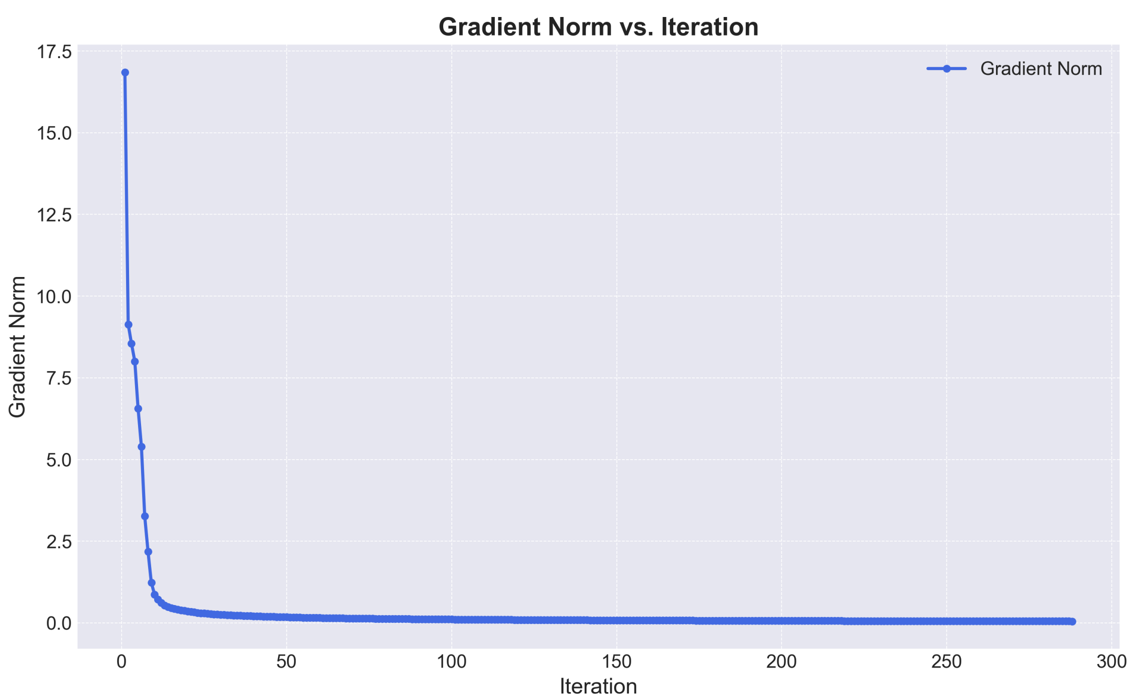
2.运行结果

迭代次数 (Iterations): 514

目标值 (Objective Value): 521.83

计算时间 (CPU Time): 1.968 秒

3.算法收敛情况
算法收敛情况如下图所示：



大致在288步的时候达到容忍度要求，求解时间相当于bb算法更长，梯度下降的速度较慢。

L0范数，最小二乘，bb准则

编译命令

```
g++ -o pgm_l0_lr_bb_big PGM/main.cpp PGM/src/LeastSquares.cpp PGM/src/L1NormProx.cpp PGM/src/L2Nor
```

随机数据算法验证

1.参数设置

矩阵 A 是一个大小为 512×512 的矩阵，用于存储随机生成的实数值。每个元素 $A_{i,j}$ 都从一个均匀分布的范围 $[-10, 10]$ 中随机选取。矩阵定义如下：

$$A \in \mathbb{R}^{512\times512}$$

向量 b:

向量 b 是一个长度为 512 的向量，其元素也是从同一均匀分布 $[-10, 10]$ 中随机生成的。向量定义如下：

$$b \in \mathbb{R}^{512}$$

初始猜测向量 x:

初始猜测向量 x 是一个长度为 512 的向量，初始时所有元素都设置为 0。这个向量用于开始某种优化算法的计算。向量定义如下：

$$x = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{512}$$

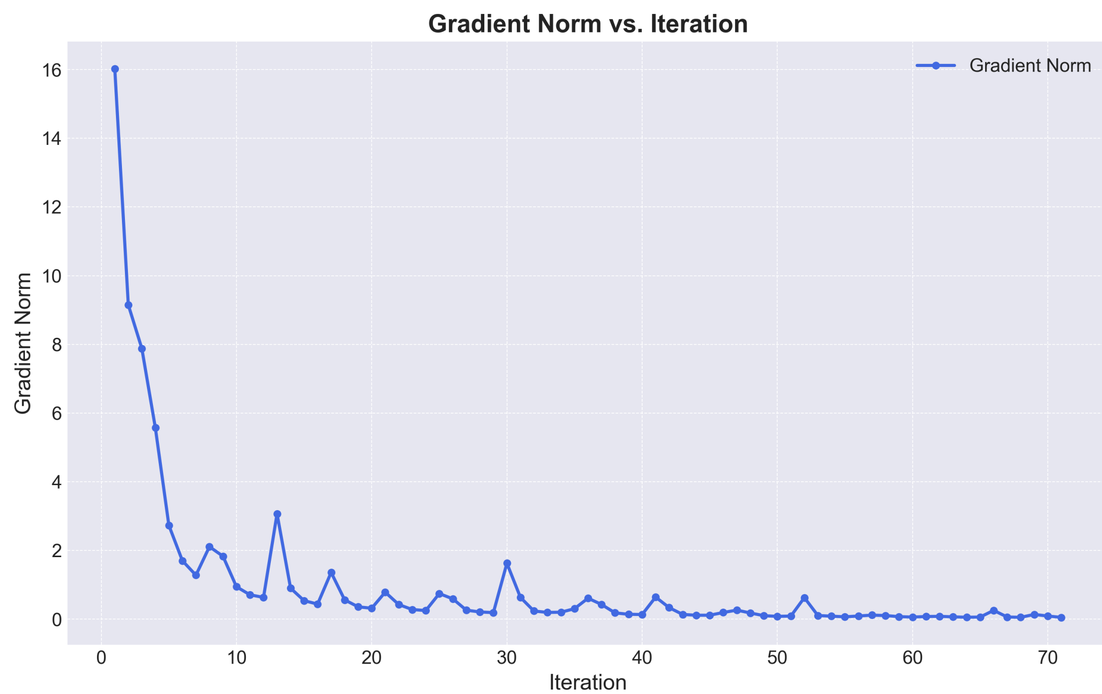
2.运行结果

迭代次数 (Iterations): 71

目标值 (Objective Value): 520.331

计算时间 (CPU Time): 0.416 秒

3.算法收敛情况
算法收敛情况如下图所示：



尝试数据	tk选取	Iter次数
big	Armijo	514
big	BB-step	71
大致在71步的时候达到容忍度要求，求解时间较短，但是求解的目标值较大。		

总结

项目背景

本项目的主要目的是开发一个灵活的优化算法包，用以解决具有复合函数形式的优化问题。复合函数通常表示为 `f(x) + h(x)`，其中 `f(x)` 是一个光滑的可微函数，而 `h(x)` 是一个可能非光滑但易于求解其近邻算子的函数。通过此项目，我们旨在实现并测试几种核心的优化算法，包括基于梯度的方法和基于近邻梯度的方法，同时探索如Armijo步长和Barzilai-Borwein (BB) 步长策

略在实际应用中的效果。

项目结构和实现

项目被组织为以下主要部分：

- **算法实现**：包括最小二乘法、L1范数、L2范数、以及L0范数的具体实现。
- **步长策略**：实现了两种动态调整步长的策略，Armijo步长策略和BB步长策略。
- **逻辑回归与最小二乘问题**：除了传统的最小二乘问题外，还引入了逻辑回归问题作为测试算法性能的另一种方式。

代码结构

代码分为几个主要模块，存放在不同的文件夹中：

- `include/`：包含所有的头文件，定义了函数和算法的接口。
- `src/`：包含所有的源代码文件，实现了具体的算法逻辑。

主要功能

- **Function 类**：用于定义优化问题中的目标函数。
- **ProximalOperator 类**：定义了处理含有非光滑项的优化问题所需的近邻算子。
- **ProximalGradientOptimizer 类**：实现了基于近邻梯度的优化方法，是项目的核心。

测试与验证

项目通过多种数据集进行了广泛测试，包括合成数据和真实数据集。测试结果表明，不同的步长选择策略对算法性能有显著影响。特别是在处理大规模数据集时，BB步长策略因其自适应特性表现出较好的性能。

结果分析

通过对比不同算法和步长策略的性能，我们得出以下结论：

- **Armijo步长**：虽然可以保证稳定的收敛，但在某些情况下会因过于保守而导致收敛

速度较慢。

- **BB步长**：在大多数情况下能够加速收敛，尤其是在梯度具有复杂变化或问题规模较大时。

项目挑战

项目实施过程中遇到的主要挑战包括算法的参数调整、处理大规模数据的计算效率，以及不同算法在不同类型问题上的适用性分析。同时对于BB算法的参数选择存在一些问题，在某些情况下，由于步长策略选择和容忍度参数设置问题，会出现震荡的情况。

总体总结

本项目成功开发了一个功能完备的优化算法软件包，不仅支持多种优化策略，还提供了强大的灵活性来处理各种优化问题。通过实际数据的广泛测试，我们验证了所提算法的有效性和高效性，为未来的研究和实际应用奠定了坚实的基础。