实验说明文档

1. 前置知识与参数设置

1. 牛顿下降法与实验中参数的设置
2. Cmake是什么，如何在Linux上用
3. Eigen数值计算
4. 基于spdlog的日志
5. 格式化输出

2. 实验要求

```
0.用CMake管理项目，用Eigen库完成向量计算和矩阵求逆
1.所有浮点数都用double，向量和矩阵也使用后缀为d类型
2.同时用spdlog输出日志到终端和optimizer.log，以debug级别记录每一步迭代的结果
3.以info级别记录得到的最小值，所有向量和数值直接输出，不加任何描述。
4.不用其它第三方库
5.输出形式为：以下输出序列可以作为用例来验证程序正确性
[optimizer] [debug] (138, 1620)
[optimizer] [debug] (69, 810)
[optimizer] [debug] (34.5, 405)
[optimizer] [debug] (17.25, 202.5)
[optimizer] [debug] (8.625, 101.25)
[optimizer] [debug] (4.3125, 50.625)
[optimizer] [debug] (2.15625, 25.3125)
[optimizer] [debug] (1.078125, 12.65625)
[optimizer] [debug] (0.5390625, 6.328125)
[optimizer] [debug] (0.26953125, 3.1640625)
[optimizer] [debug] (0.134765625, 1.58203125)
[optimizer] [debug] (0.0673828125, 0.791015625)
[optimizer] [debug] (0.03369140625, 0.3955078125)
[optimizer] [debug] (0.016845703125, 0.19775390625)
[optimizer] [debug] (0.0084228515625, 0.098876953125)
[optimizer] [debug] (0.00421142578125, 0.0494384765625)
[optimizer] [debug] (0.002105712890625, 0.02471923828125)
[optimizer] [debug] (0.0010528564453125, 0.012359619140625)
[optimizer] [info] 0.00015386869199573994
注意一个细节：
[optimizer] [debug] (0.0010528564453125, 0.012359619140625)中
x=0.0010528564453125
y=0.012359619140625
z=x*x+y*y=0.00015386869199573994
就是[optimizer] [info] 0.00015386869199573994
```

3. 实验代码

3.0. 运行前目录

```
./lab0
  |CMakeLists.txt
  |formatter.hpp
  |main.cpp
  |./deps
        |./Eigen
        |./fmt
        |./spdlog
```

3.1. CMakeLists.txt

```
# 设置CMake的最低版本要求为3.5
cmake_minimum_required(VERSION 3.5)

# 定义项目的名字为"optimizer"和版本号为0.1
project(optimizer VERSION 0.1)

# 设置项目使用的C++标准为C++17
set(CMAKE_CXX_STANDARD 17)

# 如果所需的C++标准不可用，则停止配置
set(CMAKE_CXX_STANDARD_REQUIRED TRUE)

# 生成编译命令的JSON文件，此文件可以被一些工具和编辑器使用，例如clang-tidy和Visual Studio Code
set(CMAKE_EXPORT_COMPILE_COMMANDS TRUE)

# 定义要编译的源文件列表
set(SOURCES   main.cpp)

# 定义项目中的头文件列表
set(HEADERS
 deps/spdlog/spdlog.h
 deps/spdlog/sinks/stdout_color_sinks.h
 deps/spdlog/sinks/basic_file_sink.h
)

# 为项目添加一个可执行文件目标，包括源文件和头文件，此处的${PROJECT_NAME}=demo
add_executable(${PROJECT_NAME} ${SOURCES} ${HEADERS})

# 设置该目标的include目录，当编译时，编译器会在这些目录中查找头文件，PRIVATE指示后面的目录或目标仅用于此目标
target_include_directories(${PROJECT_NAME} PRIVATE deps)

# 添加编译定义，这些定义会作为宏在项目中
target_compile_definitions(${PROJECT_NAME}
PRIVATE SPDLOG_FMT_EXTERNAL  # 使用外部的fmt库，而不是spdlog内部的版本
PRIVATE FMT_HEADER_ONLY      # 使用header-only的fmt版本
)
```

3.2. main.cpp

```cpp
#include <memory>                                   // 用于智能指针
#include <spdlog/spdlog.h>                          // spdlog的主要头文件
#include <spdlog/sinks/stdout_color_sinks.h> // 用于将带颜色的日志输出到控制台
#include <spdlog/sinks/basic_file_sink.h>     // 用于将日志输出到文件
#include <Eigen/Core>                               // Eigen核心功能
#include <Eigen/Dense>                              // 包括了Eigen的密集矩阵和向量的定义及其操作

int main()
{
 /*设置spdlog日志记录器并使用自定义格式*/
 // 创建一个控制台日志sink
 auto console_sink = std::make_shared<spdlog::sinks::stdout_color_sink_mt>();
 // 创建一个文件日志sink
 auto file_sink = std::make_shared<spdlog::sinks::basic_file_sink_mt>("optimizer.log", true);
 // 创建一个日志记录器并添加sinks
 auto logger = std::make_shared<spdlog::logger>("optimizer", spdlog::sinks_init_list{console_sink, file_sink});
 // 注册日志记录器
 spdlog::register_logger(logger);
 // 设置日志记录器的级别为debug
 logger->set_level(spdlog::level::debug);
 // 设置输出格式
 logger->set_pattern("[%n] [%l] %v");

 using Eigen::Vector2d;    // 使用Eigen的2维向量

 Vector2d x(520, 279);     // 设置初始点
                           //可以拿Vector2d x(138, 1620)验证
 Vector2d x_prev;          // 再一次牛顿下降迭代中，用于存储前一个点
 Vector2d x_next;          // 用于存储下一个点
 Vector2d x_delta;         // 用于存储两个连续点之间的差异

 double lambda = 0.5;      // 步长
 double delta = 0.01;      // 收敛的阈值

 logger->debug("({}, {})", x[0], x[1]);                // 先在日志中输出最开始的点

 do {
     x_prev = x;                                       // 记录下当前点的点
     x = x * 0.5;                                      // 按照牛顿下降法迭代点，公式推导详见说明文档
     x_next = x;                                       // 记录下更新后的点
     x_delta = x_prev - x_next;                        // 计算两个连续点之间的差异
```

```
    if (x_delta.norm() <= delta) {break;}             // 在写入日志之前检查条件是否符合
    logger->debug("({}, {})", x_next[0], x_next[1]); // 在日志中输出迭代后的点
 } while (true);

 logger->info("{}", x_prev.norm()*x_prev.norm());     // 记录最终点的函数值，也就是范数平方

 return 0;
}
```

4. 代码运行

4.1. 控制台输出

```
dhy@dhy-virtual-machine:~/桌面/Lab0$ mkdir build
dhy@dhy-virtual-machine:~/桌面/Lab0$ cd build
dhy@dhy-virtual-machine:~/桌面/Lab0/build$ cmake ..
-- The C compiler identification is GNU 11.4.0
-- The CXX compiler identification is GNU 11.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/dhy/桌面/Lab0/build
dhy@dhy-virtual-machine:~/桌面/Lab0/build$ make
[ 50%] Building CXX object CMakeFiles/optimizer.dir/main.cpp.o
[100%] Linking CXX executable optimizer
[100%] Built target optimizer
dhy@dhy-virtual-machine:~/桌面/Lab0/build$ ./optimizer
[optimizer] [debug] (520, 279)
[optimizer] [debug] (260, 139.5)
[optimizer] [debug] (130, 69.75)
[optimizer] [debug] (65, 34.875)
[optimizer] [debug] (32.5, 17.4375)
[optimizer] [debug] (16.25, 8.71875)
[optimizer] [debug] (8.125, 4.359375)
[optimizer] [debug] (4.0625, 2.1796875)
[optimizer] [debug] (2.03125, 1.08984375)
[optimizer] [debug] (1.015625, 0.544921875)
[optimizer] [debug] (0.5078125, 0.2724609375)
[optimizer] [debug] (0.25390625, 0.13623046875)
[optimizer] [debug] (0.126953125, 0.068115234375)
[optimizer] [debug] (0.0634765625, 0.0340576171875)
[optimizer] [debug] (0.03173828125, 0.01702880859375)
[optimizer] [debug] (0.015869140625, 0.008514404296875)
```

```
40 [optimizer] [info] 0.00032432470470666885
```

4.2. 日志文件输出

4.2.1. 文件目录

```
1 ./Lab0/build/optimizer.log
```

4.2.2. 文件内容

```
1  [optimizer] [debug] (520, 279)
2  [optimizer] [debug] (260, 139.5)
3  [optimizer] [debug] (130, 69.75)
4  [optimizer] [debug] (65, 34.875)
5  [optimizer] [debug] (32.5, 17.4375)
6  [optimizer] [debug] (16.25, 8.71875)
7  [optimizer] [debug] (8.125, 4.359375)
8  [optimizer] [debug] (4.0625, 2.1796875)
9  [optimizer] [debug] (2.03125, 1.08984375)
10 [optimizer] [debug] (1.015625, 0.544921875)
11 [optimizer] [debug] (0.5078125, 0.2724609375)
12 [optimizer] [debug] (0.25390625, 0.13623046875)
13 [optimizer] [debug] (0.126953125, 0.068115234375)
14 [optimizer] [debug] (0.0634765625, 0.0340576171875)
15 [optimizer] [debug] (0.03173828125, 0.01702880859375)
16 [optimizer] [debug] (0.015869140625, 0.008514404296875)
17 [optimizer] [info] 0.00032432470470666885
```

5. 文件

Cmake前
Cmake后
提交的文件