1. Eigen C++ 线性代数库入门

1.1. 头文件

```
// 包含矩阵类型及其基本运算的定义，通常都需要引入
#include <Eigen/Core>
// 包含一些额外的矩阵运算
#include <Eigen/Dense>
```

1.2. Eigen中的矩阵模板类

```
//类定义
/*Scalar为数据类型
RowsAtCompileTime和ColsAtCompileTile为行列数
option默认为ColumnMajor表示这个矩阵是列优先存储的(默认是列向量)*/
Matrix<Scalar, RowsAtCompileTime, ColsAtCompileTile, Options>
```

1.3. 定义矩阵类

```
//一般方式
Matrix<float, 3, 3>//定义3*3的每个元素为float类型的矩阵
Matrix<double, 4, 5>//定义4*5的每个元素为double类型的矩阵
//简化方式：[Matrix/Vector]X[i/f/d]
typedef Matrix<float, 2, 2> Matrix2f;//定义2*2的每个元素为float类型的矩阵
typedef Matrix<float, 3, 1> Vector3f;//定义3*1的每个元素为float类型的向量
//定义完不要忘了这一步
using Eigen::Matrix3f;
using Eigen::Vector3f;
//定义，初始化矩阵和向量
Matrix3f A = Matrix3f::Identity();
/*   1 0 0
  A= 0 1 0
     0 0 1  */
vector3f x(4.0f,5.0f,6.0f);
/*   4
  x= 5
     6      */
```

1.4. 矩阵运算

```
//修改矩阵某个元素的值
A(1,2)=1.0f;
/*   1 0 0
  A= 0 1 2
     0 0 1   */
//矩阵乘法
vector3f p=A*x;
/*   4
  p= 17
     6       */
//修改向量某个元素的值
```

```
p.z()=5.0f;
/*   4
  p= 17
     5       */
```

1.5. 矩阵操作

```
//矩阵转置
Matrix3f A_inv = A.inverse();
Matrix3f A_trans = A.transpose();
//生成单位向量
Vector3f x = Vector3f::UnitX();//[1,0,0]
Vector3f y = Vector3f::UnitY();//[0,1,0]
Vector3f z = Vector3f::UnitZ();//[0,0,1]
//计算两个向量的点积和叉积
float x_dot_y = x.dot(y);//x_dot_y=1*0+0*1+0*0=0
Vector3f z = x.cross(y);//求x,y的法向量，也就是[0,0,1]
```

1.6.更多精彩

https://eigen.tuxfamily.org/dox/

2. spdlog日志基本功能

https://github.com/gabime/spdlog/wiki

2.1. 什么是日志

程序日志好比数的目录和页码，记录程序运行过程中的关键事件(何时开始运行，什么错误，执行完什么),程序出现问题时开发者可迅速定位问题原因，或在程序运行完毕后通过日志来查看程序的执行情况

2.2. spdlog日志库两个核心概念：logger和sink

2.2.1. logger（逻辑层面）

1️⃣ logger好比一个指挥官，是程序员与日志系统交互的接口

2️⃣ logger决定了什么信息需要记录，以及这些信息的记录级别（是否是一个错误or警告or信息）

2.2.2. sink（物理层面）

1️⃣ sinkh好比logger命令的士兵，当logger决定输出某个信息时，sink执行具体的方式来存储/显示

2️⃣ 例如，一个sink要将日志信息写入到一个文件，另一个sink要把日志信息显示在终端上

3️⃣ 一个logger可有多个sink，意味着同一条日志信息可以同时被写入到不同的位置

2.2.3. 二者配合完成操作

> 当要记录一个日志信息：
>
> 1️⃣用logger来指定想要记录的内容
>
> 2️⃣logger随后通知相关的sink
>
> 3️⃣sink将这个信息输出到特定的地方(文件/终端/内存)

2.3. 程序示例

> 创建了一个输出到终端(stdout)的sink和一个输出到文件的sink，并创建一个与它们关联的logger

2.3.1. 关键代码事先讲解

```
auto console_sink = std::make_shared<spdlog::sinks::stdout_color_sink_st>();
/*解析：
1.auto:在不确定返回变量类型时候，auto会让编译器自动推断并确认变量的类型
2.console_sink:变量名，保存创建的sink的引用
3.std::make_shared:是一个模板函数，用于创建一个std::shared_ptr智能指针(空就销毁)
4.spdlog::sinks::stdout_color_sink_st:这是spdlog库中一个类，它表示一个sink，该sink的作用是将日志输出到控制台，并为不同级别的日志添加不同的颜色
执行操作：
1.使用std::make_shared创建一个spdlog::sinks::stdout_color_sink_st类型的对象，并返回该对象的一个智能指针。
2.将这个智能指针保存在console_sink变量中。*/

auto file_sink = std::make_shared<spdlog::sinks::basic_file_sink_st>("my_program.log", true);
/*解析：
1.spdlog::sinks::basic_file_sink_st:是spdlog库中的一个类,它是一个sink，专门用于将日志消息输出到文件,_st表明它是单线程版本，对于线程安全版本使用_mt
2.("my_program.log", true):这是传递给spdlog::sinks::basic_file_sink_st构造函数的两个参数。"my_program.log"为日志写入的文件名，true表示若文件存在，新的日志应该追加进去
执行操作：
1.使用std::make_shared创建一个类型为spdlog::sinks::basic_file_sink_st的对象，并传递给它两个参数：文件名和一个标志，表示是否追加到现有文件。
2.返回的智能指针(指向创建的sink对象)被存储在file_sink变量中。*/
```

2.3.2.程序

```
#include <memory>  //使可以使用智能指针，如std::shared_pt
#include <spdlog/spdlog.h> //spdlog的主要头文件，这提供了spdlog的核心功能
#include <spdlog/sinks/stdout_color_sinks.h> //引入spdlog一特定sink，将日志信息带颜色输出终端
```

```cpp
#include <spdlog/sinks/basic_file_sink.h>//引入spdlog的另一个sink，将日志信息输出到文件

int main()
{
auto console_sink = std::make_shared<spdlog::sinks::stdout_color_sink_st>();
auto file_sink = std::make_shared<spdlog::sinks::basic_file_sink_st>("my_program.log", true);

/*创建一个名为"my logger"的logger对象，并与上面创建的两个sink关联*/
spdlog::logger my_logger("my logger", {console_sink, file_sink});

/*使用logger向日志中记录不同级别的消息*/
/*以my_logger.debug为例，当debug时就会调用my_logger.debug然后向日志输出字符串"This is a debug message"，其它的同理*/
/*执行效果为：将日志信息同时输出到两个sink对应的文件(缓冲)*/
my_logger.debug("This is a debug message");            // 记录调试消息
my_logger.info("Some information during processing"); // 记录信息消息
my_logger.warn("Warning, something is going wrong");  // 记录警告消息
my_logger.error("An error occurred");                 // 记录错误消息
my_logger.critical("Critical error, emergency stop"); // 记录关键错误消息

return 0;
}
```

2.3.3. 程序运行：环境Ubuntu20

0️⃣目录结构

```
./根目录
|CMakeLists.txt
|main.cpp
|./deps
|  |./spdlog
```

1️⃣CMakeLists.txt

```cmake
# 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"demo"和版本号为0.1
project(demo 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
)
```

2️⃣运行过程与结果

```
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/demo.dir/main.cpp.o
[100%] Linking CXX executable demo
[100%] Built target demo
dhy@dhy-virtual-machine:~/桌面/Lab0/build$ ./demo
```

```
[2023-09-15 17:09:43.345] [my logger] [info] Some information during processing
[2023-09-15 17:09:43.346] [my logger] [warning] Warning, something is going wrong
[2023-09-15 17:09:43.346] [my logger] [error] An error occurred
[2023-09-15 17:09:43.346] [my logger] [critical] Critical error, emergency stop
```

每一条日志里依次是时间、logger 名字、日志级别和日志信息

同时./demo指令输出的日志同样被存入./根目录/build/my_Program.log中

2.3.4. 运行完程序后的思考：日志级别

1️⃣第一条日志(比如trace或debug)没有出现，原因在于spdlog从低到高有 trace, debug, info, warn, error, critical 六个日志级别，默认只输出info及以上级别

2️⃣设置日志级别

```cpp
// 设置全局日志级别。这会影响所有的logger，除非它们有自己的特定级别设置。
// 在这里，我们设置全局日志级别为'debug'，这意味着所有级别为'trace'及以上的日志都会被输出。
spdlog::set_level(spdlog::level::trace);

// 为特定的logger设置日志级别。这会覆盖全局的日志级别设置，只影响该特定logger。
// 假设我们已经创建了一个名为'logger'的logger实例（它是一个shared_ptr），
// 这行代码会设置它的日志级别为'debug'，所以该logger会输出'debug'及以上级别的日志。
logger->set_level(spdlog::level::debug);

// 为特定的sink设置日志级别。sink是日志消息的输出目的地，比如一个文件或控制台窗口。
// 通过设置sink的级别，你可以控制输出到该sink的日志消息的级别。
// 假设我们有一个名为'sink'的sink实例（它也是一个shared_ptr），
// 这行代码会设置其日志级别为'debug'，所以通过该sink输出的日志会是'debug'及以上级别的。
sink->set_level(spdlog::level::debug);
```

3️⃣以刚才的程序为例：

可以在创建 sink 和 logger 之前设置全局日志级别

可以设置 my_logger 的日级别

可以设置 console_sink 或者 file_sink 的日志级别

例如：

```
int main()
{
// 设置全局日志级别为 'debug'
spdlog::set_level(spdlog::level::debug);
auto console_sink =
std::make_shared<spdlog::sinks::stdout_color_sink_st>();
auto file_sink =
std::make_shared<spdlog::sinks::basic_file_sink_st>
("my_program.log", true);
....
}
```

3. 格式化输出

3.0. 日志中输出字符串

就是我们在spdlog日志基本功能中看到的

```
my_logger.debug("This is a debug message");
my_logger.info("Some information during processing");
my_logger.warn("Warning, something is going wrong");
```

3.1. 基于fmtlibs的pdlog格式化字符串

fmtlib提供对C++内置类型/大多STL容器/C++时间类型的格式化输出，想要用日志记录这些 变量时，只要在格式串里放个{}占位即可

3.1.1. 程序(CMakeLists.txt不变)

```
#include <memory>
#include <spdlog/spdlog.h>
#include <spdlog/sinks/stdout_color_sinks.h>
#include <spdlog/sinks/basic_file_sink.h>

int main()
{
/*使用spdlog的helper函数创建一个仅有一个sink的logger,这是一个简便的方法，使我们不必显式创建sink和logger,st指示logger是为单线程设计的*/
auto logger = spdlog::stdout_color_st("my_logger");

/*定义一些变量*/
int a = 10;
float pi = 3.1415926;
std::string message = "Hello, world!";

/*使用格式化字符串来记录日志，并在其中插入变量a的值,"{}"是一个占位符，它将被变量a的值所替换*/
logger->info("a integer: {}", a);

/*同样地，使用格式化字符串记录日志，并在其中插入变量message的值*/
logger->info("my program says: {}", message);

```

```
/*使用两个占位符记录日志。第一个占位符将被pi的值替换，第二个将被pi的值替换但格式化为固定点数并只显示两位小数,"{:.2f}"的意思是输出一个浮点数并且只显示两位小数*/
logger->info("pi is {},  approximation: {:.2f}", pi, pi);

/*看到logger->info三次调用时，它们都会在日志中产生输出，每次调用logger->info都会产生一个新的日志条目*/

return 0;
}

```

3.1.2. 运行结果

```
dhy@dhy-virtual-machine:~/桌面/Lab0$ mkdir a
dhy@dhy-virtual-machine:~/桌面/Lab0$ cd a
dhy@dhy-virtual-machine:~/桌面/Lab0/a$ 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/a
dhy@dhy-virtual-machine:~/桌面/Lab0/a$ make
[ 50%] Building CXX object CMakeFiles/demo.dir/main.cpp.o
[100%] Linking CXX executable demo
[100%] Built target demo
dhy@dhy-virtual-machine:~/桌面/Lab0/a$ ./demo
[2023-09-15 18:17:36.325] [my_logger] [info] a integer: 10
[2023-09-15 18:17:36.325] [my_logger] [info] my program says: Hello, world!
[2023-09-15 18:17:36.325] [my_logger] [info] pi is 3.1415925, approximation: 3.14
```

3.2. fmtlib如何格式化矩阵？→自定义 formatter

https://dandelion-docs.readthedocs.io/zh_CN/latest/d3/d31/formatter_8hpp.html

在本次作业中，只需直接从 Dandelion 源代码库中复制 utils/formatter.hpp 这个文件到CMake项目的根目录中，然后在引入 spdlog 头文件之前先引入这个头文件，就可以像输出内置类 型那样输出向量和矩阵了

示例见下：

3.2.1. 程序

```cpp
#include "formatter.hpp"
#include <spdlog/spdlog.h>
#include <Eigen/Core>
using Eigen::Vector3f;
using Eigen::Matrix3f;
int main()
{
Vector3f v(1.0f, 2.0f, 3.0f);          //向量v=[1.0 2.0 3.0]
spdlog::info("vector: {:.2f}", v);     //输出向量，用向量v填充{:.2f}保留两位数字
Matrix3f I = Matrix3f::Identity();     //三阶单位向量
spdlog::info("matrix: {:>5.1f}", I); /* >：输出向右对其
                                         5：输出字符串最小宽度为5
                                        .1：精度为1(小数点后有1位)
                                         f：表示要格式化一个浮点数。*/
return 0;
}
```

3.2.2. 运行情况

```
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/demo.dir/main.cpp.o
[100%] Linking CXX executable demo
[100%] Built target demo
dhy@dhy-virtual-machine:~/桌面/Lab0/build$ ./demo
[2023-09-15 19:52:19.830] [info] vector: (1.00, 2.00, 3.00)
[2023-09-15 19:52:19.830] [info] matrix:
1.0   0.0   0.0
0.0   1.0   0.0
0.0   0.0   1.0
```

4. 牛顿下降法

4.1.选择初始点和步长

1️⃣ $x_0 = (x_0, y_0)$ λ 为步长

在本题中取值为：

$x_0 = (2213611582 mod 827, 2213611582 mod 1709) = (520, 279)$ 和$\lambda = 0.5$

2️⃣ 要分析的函数为$f(x, y) = x^2 + y^2$

4.2. 对于第 k步的点 $x_k = (x_k, y_k)$，计算梯度和Hessian矩阵的逆:

1️⃣ 梯度，对于一个标量函数 $f(x, y)$，其梯度是一个向量，给出了函数在该点上升得最快的方向。梯度的公式为：

$$\nabla f(x_k) = \begin{bmatrix} \frac{\partial f}{\partial x}(x_k, y_k) \\ \frac{\partial f}{\partial y}(x_k, y_k) \end{bmatrix} \xrightarrow{\text{在本题中为}} \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

2️⃣ Hessian矩阵是一个二阶导数矩阵，它描述了一个函数的局部曲率。对于函数 $f(x, y)$，Hessian矩阵定义为：

$$Hf(x_k) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2}(x_k, y_k) & \frac{\partial^2 f}{\partial x \partial y}(x_k, y_k) \\ \frac{\partial^2 f}{\partial y \partial x}(x_k, y_k) & \frac{\partial^2 f}{\partial y^2}(x_k, y_k) \end{bmatrix} \xrightarrow{\text{在本题中为}} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$ 其中，$Hf^{-1}(x_k)$ 是Hessian矩阵的逆

所以$Hf^{-1}(x_k) \xrightarrow{\text{在本题中为}} \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}$

4.3. 进行迭代更新

$$x_{k+1} = x_k - \lambda Hf^{-1}(x_k)\nabla f(x_k) \xrightarrow{\text{在本题中为}} x_{k+1} = x_k - \frac{1}{2}\begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}\begin{bmatrix} 2x \\ 2y \end{bmatrix} = x_k - \begin{bmatrix} \frac{1}{2}x \\ \frac{1}{2}y \end{bmatrix}$$

然后惊奇的发现$x_{k+1} = \frac{1}{2}x_k$

4.4. 检查收敛性

若满足$||x_{k+1} - x_k|| < \delta = 0.01$，则停止迭代，认为找到了极小值点 x^{small}(注意这个x^{small}是要在收敛域之内的)