课后任务

大地电磁一维反演系统

- 设计一个完整的大地电磁（MT）一维反演系统，支持以下功能：
- 使用模块化设计实现正演计算、Jacobian 计算、正则化、优化求解等功能
- 使用 Intel MKL 库进行高性能科学计算（BLAS、LAPACK）
- 使用 Qt 框架构建图形用户界面（GUI）
- 使用面向对象设计实现各功能模块的解耦和可扩展性
- 使用智能指针管理动态分配的内存资源
- 使用多线程技术实现后台计算，避免阻塞 UI
- 包含完整的参数验证和异常处理机制

大地电磁反演问题简介

问题描述

- 大地电磁（MT）反演旨在根据地表观测的电磁响应数据，推断地下介质的电性结构（电阻率分布）。
- 该问题广泛应用于地球物理勘探、矿产资源勘查、地热资源开发等领域，是地球物理反演的基础问题之一。
- 一维反演假设地下介质在水平方向上均匀，仅在垂直方向上变化，适用于层状地质结构。

核心挑战

- **正演问题**：给定地下电性结构，计算地表 MT 响应（视电阻率、相位）
- **反演问题**：根据观测的 MT 响应，反推地下电性结构
- **不适定性**：反演问题通常存在多解性，需要正则化约束
- **计算效率**：需要高效的正演计算和优化算法

MT 正演问题

物理模型

- 一维层状介质模型：地下分为 M 层，每层具有不同的电阻率 ρ_i（或电导率 $\sigma_i = 1/\rho_i$）和不同的层厚度
- 极化模式：TE（横电）极化，电场垂直于传播方向

正演方法

- **解析法**：使用向上递推阻抗的方法，从底层开始逐层向上计算地表阻抗
- **有限差分法/有限元法**：将 Helmholtz 方程离散化，求解三对角线性方程组
- **输出数据**：视电阻率 ρ_a 和相位 ϕ，通常以 $\log_{10}(\rho_a)$ 和 ϕ 的形式存储

MT 一维正演的解析法

基本思想

- 使用向上递推阻抗的方法，从最底层（半空间）开始，逐层向上计算到地表的阻抗
- 基于传输线理论，利用层间阻抗的连续性条件建立递推关系
- 适用于 TE（横电）极化模式，电场垂直于传播方向

关键物理量

- **特征阻抗**：$Z_{0i} = \sqrt{\frac{i\omega\mu_0}{\sigma_i}} = (1+i)\sqrt{\frac{\omega\mu_0}{2\sigma_i}}$
- **波数**：$k_i = \sqrt{i\omega\mu_0\sigma_i} = (1+i)\sqrt{\frac{\omega\mu_0\sigma_i}{2}}$
- 其中 $\mu_0 = 4\pi \times 10^{-7}$ H/m 为真空磁导率，$\sigma_i = 1/\rho_i$ 为第 i 层的电导率

MT 一维正演的解析法：递推公式

最底层（半空间）阻抗

最底层（第 $M-1$ 层）作为半空间，其阻抗为：

$$Z_{M-1} = Z_{0,M-1} = \sqrt{\frac{i\omega\mu_0}{\sigma_{M-1}}} = (1+i)\sqrt{\frac{\omega\mu_0}{2\sigma_{M-1}}}$$

向上递推公式

从第 $i+1$ 层向上递推到第 i 层的阻抗递推公式：

$$Z_i = Z_{0i} \cdot \frac{Z_{i+1} + Z_{0i}\tanh(k_i d_i)}{Z_{0i} + Z_{i+1}\tanh(k_i d_i)}$$

其中 d_i 为第 i 层的厚度，$\tanh(k_i d_i)$ 为复双曲正切函数。

- 递推过程：从 $i = M-2$ 开始，逐层向上计算到 $i = 0$（地表层）

复双曲正切函数

对于复数 $z = x + iy$，双曲正切函数定义为：

$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

- 对于 $z = k_i d_i$，其中 $k_i = (1+i)\alpha_i$，$\alpha_i = \sqrt{\frac{\omega\mu_0\sigma_i}{2}}$：

$$e^{k_i d_i} = e^{\alpha_i d_i}(\cos(\alpha_i d_i) + i\sin(\alpha_i d_i))$$

- 因此：

$$\sinh(k_i d_i) = \frac{1}{2}(e^{k_i d_i} - e^{-k_i d_i}), \quad \cosh(k_i d_i) = \frac{1}{2}(e^{k_i d_i} + e^{-k_i d_i})$$

MT 一维正演的解析法：响应计算

视电阻率和相位

计算得到地表阻抗 Z_0 后，可计算 MT 响应：

- **视电阻率**：

$$\rho_a = \frac{|Z_0|^2}{\omega\mu_0} = \frac{Z_0 \cdot Z_0^*}{\omega\mu_0}$$

其中 Z_0^* 为 Z_0 的共轭复数，$|Z_0|^2 = (\mathrm{Re}(Z_0))^2 + (\mathrm{Im}(Z_0))^2$

- **相位**：

$$\phi = \arctan\left(\frac{\mathrm{Im}(Z_0)}{\mathrm{Re}(Z_0)}\right) \quad \text{（弧度）}$$

或

$$\phi = \frac{180}{\pi}\arctan\left(\frac{\mathrm{Im}(Z_0)}{\mathrm{Re}(Z_0)}\right) \quad \text{（度）}$$

输出格式

通常以 $\log_{10}(\rho_a)$ 和相位 ϕ（度）的形式存储，便于后续处理和可视化

MT 反演问题的数学表述

反演问题的定义

给定观测数据 $d_{obs} \in \mathbb{R}^n$（n 为数据点数，通常 $n = 2n_{freq}$，包含视电阻率和相位），寻找模型参数 $m \in \mathbb{R}^M$（M 为模型层数），使得合成数据 $d_{syn}(m)$ 与观测数据 d_{obs} 的差异最小。

- 正演算子：$d_{syn} = F(m)$，其中 $F : \mathbb{R}^M \to \mathbb{R}^n$ 为非线性正演算子。
- 反演问题通常是不适定的（ill-posed），即解不唯一或不稳定，需要正则化约束。

数据拟合项

数据拟合项衡量合成数据与观测数据的差异：

$$\Phi_d(m) = \|d_{obs} - d_{syn}(m)\|^2 = \sum_{i=1}^{n} [d_{obs,i} - d_{syn,i}(m)]^2$$

- 使用 L2 范数（最小二乘）度量数据拟合程度。

正则化与目标函数

正则化的必要性

反演问题的不适定性导致：

- 解不唯一：多个模型可能产生相同的观测数据
- 解不稳定：观测数据的小扰动可能导致模型的大变化
- 过度拟合：模型可能过度复杂，包含虚假结构

正则化通过引入先验信息（如模型平滑性）来稳定反演过程。

目标函数

阻尼最小二乘（Tikhonov 正则化）目标函数：

$$\Phi(m) = \Phi_d(m) + \lambda \Phi_m(m) = \|d_{obs} - d_{syn}(m)\|^2 + \lambda \|Lm\|^2$$

其中 $\Phi_d(m)$ 为数据拟合项，$\Phi_m(m) = \|Lm\|^2$ 为模型约束项，$\lambda > 0$ 为平衡数据拟合和模型约束的正则化参数，L 为正则化矩阵。

正则化矩阵的类型

平滑度约束（SMOOTHNESS）

二阶差分矩阵，惩罚模型参数的二阶导数，使模型平滑：

$$L_{ij} = \begin{cases} 1, & j = i \\ -2, & j = i+1 \\ 1, & j = i+2 \\ 0, & \text{其他} \end{cases}$$

对应约束：$\sum_{i=1}^{M-2}(m_i - 2m_{i+1} + m_{i+2})^2$，使相邻层之间的变化平滑。

其他正则化类型

- **平坦度约束（FLATNESS）**：一阶差分，$L_{ij} = \delta_{j,i+1} - \delta_{ji}$，惩罚模型梯度
- **最小范数约束（MINIMUM_NORM）**：$L = I$（单位矩阵），最小化模型参数的 L2 范数

高斯-牛顿法：线性化

非线性问题的线性化

正演算子 $F(m)$ 是非线性的，在模型 $m^{(k)}$ 附近进行一阶泰勒展开：

$$d_{syn}(m) \approx d_{syn}(m^{(k)}) + J^{(k)}(m - m^{(k)})$$

其中 $J^{(k)}$ 为 Jacobian 矩阵（灵敏度矩阵），元素为：

$$J_{ij}^{(k)} = \frac{\partial d_{syn,i}}{\partial m_j}\Big|_{m=m^{(k)}}$$

- Jacobian 矩阵的维度为 $n \times M$，表示每个数据对每个模型参数的灵敏度。

残差向量

定义残差向量：$r^{(k)} = d_{obs} - d_{syn}(m^{(k)})$，则线性化后的数据拟合项为 $\Phi_d(m) \approx \|r^{(k)} - J^{(k)}\delta m\|^2$，其中 $\delta m = m - m^{(k)}$ 为模型更新量。

高斯-牛顿法：正规方程推导

目标函数的线性化

将目标函数在 $m^{(k)}$ 附近线性化：$\Phi(m) \approx \|r^{(k)} - J^{(k)}\delta m\|^2 + \lambda\|L(m^{(k)} + \delta m)\|^2$

最小化条件

对 δm 求导并令其为零，得到正规方程：

$$[(J^{(k)})^T J^{(k)} + \lambda L^T L]\delta m = (J^{(k)})^T r^{(k)} - \lambda L^T L m^{(k)}$$

- 这是对称正定线性方程组，可使用 Cholesky 分解高效求解。

阻尼高斯-牛顿法

简化形式

- 当初始模型 $m^{(0)}$ 接近零或正则化项较小时，可忽略 $\lambda L^T L m^{(k)}$ 项，得到简化形式：

$$[(J^{(k)})^T J^{(k)} + \lambda L^T L]\delta m = (J^{(k)})^T r^{(k)}$$

- 这是对称正定线性方程组，可使用 Cholesky 分解高效求解。

迭代更新

模型更新公式：$m^{(k+1)} = m^{(k)} + \delta m^{(k)}$ 迭代过程：

1. 计算当前模型的合成数据 $d_{syn}(m^{(k)})$
2. 计算残差 $r^{(k)} = d_{obs} - d_{syn}(m^{(k)})$
3. 计算 Jacobian 矩阵 $J^{(k)}$
4. 求解正规方程得到 $\delta m^{(k)}$
5. 更新模型：$m^{(k+1)} = m^{(k)} + \delta m^{(k)}$

Jacobian 矩阵的计算

有限差分法

由于正演算子的解析导数难以计算，使用有限差分法近似：

- **前向差分法**：

$$J_{ij} = \frac{\partial d_{syn,i}}{\partial m_j} \approx \frac{d_{syn,i}(m + \epsilon e_j) - d_{syn,i}(m)}{\epsilon}$$

  其中 e_j 为第 j 个标准基向量，ϵ 为扰动步长（通常 10^{-5}）。

- **中心差分法**（更精确但计算量加倍）：

$$J_{ij} \approx \frac{d_{syn,i}(m + \epsilon e_j) - d_{syn,i}(m - \epsilon e_j)}{2\epsilon}$$

计算复杂度

- 前向差分法需要 M 次正演计算（每列一次）
- 中心差分法需要 $2M$ 次正演计算
- Jacobian 矩阵大小为 $n \times M$，通常 $n \gg M$（如 $n = 122$，$M = 40$）

收敛判据与停止条件

收敛判据

- **残差判据**：数据拟合残差的 L2 范数小于容差

$$\|r^{(k)}\| = \|d_{obs} - d_{syn}(m^{(k)})\| < \text{tol\_residual}$$

- **模型更新判据**：模型更新量的 L2 范数小于容差

$$\|\delta m^{(k)}\| < \text{tol\_dm}$$

  通常使用 tol_dm $= 10^{-4}$，表示模型已收敛。
- **最大迭代次数**：防止无限迭代，通常设置 max_iter $= 20$。

正则化参数的选择

λ 的选择影响反演结果：λ 过大：模型过度平滑，数据拟合差；λ 过小：模型可能振荡，不稳定；通常通过 L 曲线法或交叉验证选择最优 λ

系统架构设计

模块化设计

- **数据模型模块（mt_model.h）**：定义所有数据结构（模型参数、观测数据、反演结果等）
- **频率生成器（mt_frequency_generator）**：生成 MT 反演所需的频率数组
- **正演求解器（mt_forward_solver）**：封装 1D MT 正演计算，使用 MKL 库进行高性能计算
- **Jacobian 计算器（mt_jacobian_calculator）**：计算反演所需的灵敏度矩阵
- **正则化模块（mt_regularization）**：构建正则化矩阵，实现模型平滑约束
- **优化求解器（mt_optimizer）**：求解反演中的正规方程，使用 MKL BLAS/LAPACK
- **反演核心（mt_inversion_core）**：协调各模块完成反演任务
- **GUI 界面（mt_inversion_gui）**：提供图形用户界面，支持参数设置、实时进度显示、结果可视化

开发步骤

设计阶段

1. 设计系统架构
2. 确定模块划分
3. 定义接口规范

核心模块实现

4. 数据模型模块
5. 频率生成器
6. 正演求解器
7. Jacobian 计算器

优化与界面

8. 正则化模块
9. 优化求解器
10. 反演核心协调器
11. GUI 界面开发

测试与优化

12. 单元测试
13. 集成测试
14. 性能优化

数据模型模块（mt_model.h）

核心数据结构

- 所有数据结构定义在 `MT` 命名空间中
- **InversionParams**：反演参数（层数、频率点数、正则化参数等）
- **InversionResult**：反演结果（真实模型、初始模型、最终模型、残差历史等）
- **ModelParams**：模型参数（电阻率对数、层厚度、层深度）
- **FrequencyParams**：频率参数（周期数组、角频率数组）
- **ObservationData**：观测数据（数据数组、标准差）

设计特点

- 使用 `std::vector` 存储数组数据，支持动态大小
- 提供默认参数值，简化使用
- 结构清晰，便于扩展和维护

频率生成器模块（mt_frequency_generator）

核心功能

- 生成 MT 反演所需的频率数组，通常为对数均匀分布
- 支持自定义周期范围（默认 0.001 秒到 1000 秒）
- 自动计算角频率 $\omega = 2\pi/T$（T 为周期）

主要方法

- **generate()**：生成频率数组
- **generateParams()**：生成频率参数结构
- 支持默认参数和自定义参数

实现特点

- 对数均匀分布：$\log_{10}(T)$ 均匀分布
- 频率点数可配置（默认 61 个）
- 返回周期和角频率两个数组

正演求解器模块（mt_forward_solver）

核心功能

- 封装 1D MT 正演计算，使用向上递推阻抗的解析法
- 输入：模型参数（电阻率对数、层厚度）、角频率数组
- 输出：MT 响应数据（视电阻率对数、相位）

算法流程

- **计算电导率**：$\sigma = 1/(10^{m_{\log \rho}})$
- **向上递推**：从底层开始，逐层向上计算阻抗
- **计算响应**：根据地表阻抗计算视电阻率和相位
- 使用 MKL 复数运算库

MKL 函数使用

- `MKL_Complex16`：复数类型
- 复数运算：乘法、除法、开方
- 高效计算：利用 MKL 优化

Jacobian 计算器模块（mt_jacobian_calculator）

核心功能

- 计算反演所需的 Jacobian 矩阵（灵敏度矩阵）
- Jacobian 矩阵元素：$J_{ij} = \frac{\partial d_i}{\partial m_j}$，表示第 i 个数据对第 j 个模型参数的灵敏度

计算方法

- **前向差分**：$J_{ij} \approx \frac{d_i(m+\epsilon e_j) - d_i(m)}{\epsilon}$
- **中心差分**：$J_{ij} \approx \frac{d_i(m+\epsilon e_j) - d_i(m-\epsilon e_j)}{2\epsilon}$
- 扰动步长 ϵ 可配置（默认 10^{-5}）

实现特点

- 依赖正演求解器进行计算
- 支持两种差分方法
- 矩阵大小：$n_{data} \times M$（M 为模型参数个数）

正则化模块（mt_regularization）

核心功能

- 构建正则化矩阵 L，用于反演中的模型平滑约束
- 目标：防止反演结果过度振荡，提高稳定性

正则化类型

- **SMOOTHNESS**：平滑度约束（二阶差分）
- **FLATNESS**：平坦度约束（一阶差分）
- **MINIMUM_NORM**：最小范数约束（单位矩阵）

主要方法

- **buildLMatrix()**：构建正则化矩阵 L
- **computeLTL()**：计算 $L^T L$
- **setType()**：设置正则化类型

优化求解器模块（mt_optimizer）

核心功能

- 求解反演中的正规方程：$(J^T J + \lambda L^T L)\delta m = J^T r$
- 使用 MKL BLAS/LAPACK 进行高性能矩阵运算

MKL 函数使用

- `cblas_dsyrk`：计算 $J^T J$（对称矩阵）
- `cblas_dgemv`：计算 $J^T r$
- `LAPACKE_dposv`：求解对称正定线性方程组（Cholesky 分解）

主要方法

- **solve()**：求解正规方程
- **computeJTJ()**：计算 $J^T J$
- **computeJTr()**：计算 $J^T r$
- 支持 Cholesky 和 LU 分解

反演核心模块（mt_inversion_core）

协调器设计

- 作为协调器，使用各个模块化组件完成反演任务
- 保持向后兼容的接口，同时提供模块访问接口

核心功能

- **invert()**：执行反演主流程
- **generateRandomModel()**：生成随机模型
- **computeLayerThicknesses()**：计算层厚度
- 进度回调支持

模块访问

- **getForwardSolver()**：获取正演求解器
- **getJacobianCalculator()**：获取 Jacobian 计算器
- **getRegularization()**：获取正则化模块
- **getOptimizer()**：获取优化求解器

GUI 界面模块（mt_inversion_gui）

核心功能

- 使用 Qt 框架构建图形用户界面
- 支持参数设置、实时进度显示、结果可视化
- 使用多线程技术实现后台计算，避免阻塞 UI

界面组件

- **参数面板**：层数、频率点数、正则化参数等
- **结果表格**：显示各层的真实模型、初始模型、反演结果
- **图表显示**：模型对比图、视电阻率曲线、相位曲线、残差下降曲线
- **日志输出**：显示反演过程的详细信息

技术特点

- **多线程**：使用 `QThread` 实现后台计算
- **信号槽**：使用 Qt 信号槽机制实现线程间通信
- **Qt Charts**：使用 Qt Charts 进行数据可视化
- **智能指针**：使用 `QPointer` 管理线程指针

知识点：命名空间（namespace）

命名空间的作用

- 命名空间用于组织代码，避免命名冲突
- 所有 MT 相关模块定义在 `MT` 命名空间中
- 使用方式：`MT::ForwardSolver`、`MT::ModelParams` 等

知识点：Intel MKL 库

MKL 简介

- Intel Math Kernel Library（MKL）是高性能数学库
- 提供优化的 BLAS、LAPACK、FFT 等函数
- 支持多线程并行计算，充分利用 CPU 性能

本项目使用的 MKL 函数

- **BLAS**：`cblas_dsyrk`（对称矩阵乘法）、`cblas_dgemv`（矩阵向量乘法）
- **LAPACK**：`LAPACKE_dposv`（对称正定线性方程组求解）
- **复数运算**：`MKL_Complex16` 类型及相关运算

知识点：Qt 框架

Qt 简介

- Qt 是跨平台的 C++ 图形用户界面框架
- 提供丰富的 GUI 组件、信号槽机制、多线程支持等
- 支持 Windows、Linux、macOS 等多个平台

本项目使用的 Qt 特性

- **QWidget**：窗口和控件基类
- **QThread**：多线程支持，实现后台计算
- **Qt Charts**：数据可视化（折线图、散点图等）
- **信号槽**：实现对象间通信和线程间通信
- **智能指针**：`QPointer` 自动管理对象生命周期