

数值计算方法大作业

学号: 521030910014 姓名: 刘洺皓

1 摘要

本文提出了一种基于 jax 的优化框架, 通过 Gauss-Newton 或者 Levenberg-Marquardt 方法求解 IRIS 和 NLLS 问题, 代码仓库在 <https://github.com/Learner209/jaxopt>。我们先综述了数值计算方法在优化问题中的应用, 特别是迭代重加权最小二乘法 (IRLS) 和非线性最小二乘法 (NLLS) 在 Lp 范数线性回归和非线性回归中的应用。此外, 本文还介绍了 g2o 优化框架, 该框架广泛应用于同时定位与地图构建 (SLAM) 问题中。通过实验部分, 我们在二维和三维流型数据集上评估了不同求解器 (如 Cholesky 分解和共轭梯度法) 的性能, 并研究了迭代次数对算法收敛速度的影响。分析表明, 稀疏求解器在处理大型稀疏系统时比密集 Cholesky 分解更为高效和鲁棒。最后在附录中, 本文讨论了代码设计与架构, 特别是在利用 JAX 库进行高效数值计算方面的设计选择, 同时也给出了迭代次数在更多数据集上的实验。

2 背景介绍

2.1 迭代重加权最小二乘法 (IRLS)

迭代重加权最小二乘法 (IRLS) 是一种用于求解非线性优化问题的有效方法 [4]。它通过迭代求解加权最小二乘问题来处理具有如下形式的 p-范数目标函数的优化问题 [9]:

$$\arg \min_{\beta} \sum_{i=1}^n w_i^{(t)} |y_i - x_i^T \beta|^p$$

通过迭代方法, 每一步都涉及求解一个加权最小二乘问题 [10]:

$$\beta^{(t+1)} = \arg \min_{\beta} \sum_{i=1}^n w_i^{(t)} |y_i - x_i^T \beta|^2$$

其中, $w_i^{(t)}$ 是第 t 次迭代的权重矩阵 [13]。IRLS 广泛应用于广义线性模型的最大似然估计和稳健回归中, 用于寻找 M 估计量以减轻异常值的影响 [12], 例如通过最小化最小绝对误差而不是最小二乘误差。

IRLS 的一个重要优势是它可以与 Gaussian-Newton 和 Levenberg-Marquardt 等数值算法结合使用 [1]。

2.2 Lp 范数线性回归

为了求解线性回归问题中 Lp 范数的最小化问题 [18]:

$$\arg \min_{\beta} \sum_{i=1}^n |y_i - x_i^T \beta|^p$$

第 t+1 步的 IRLS 算法涉及求解加权最小二乘问题 [20]:

$$\beta^{(t+1)} = \arg \min_{\beta} \sum_{i=1}^n w_i^{(t)} |y_i - x_i^T \beta|^2$$

其中, $W^{(t)}$ 是对角权重矩阵, 通常初始设置为:

$$w_i^{(0)} = 1$$

然后在每次迭代后更新为：

$$w_i^{(t+1)} = \left(|y_i - x_i^T \beta^{(t)}| + \epsilon \right)^{2-p}$$

在 $p = 1$ 的情况下，这对应于最小绝对偏差回归（在这种情况下，结果会更精确，应该使用线性规划方法）。权重函数中使用 ϵ 相当于鲁棒估计中的 Huber 损失函数。

2.3 非线性最小二乘法 (NLLS)

非线性最小二乘法用于拟合一组 m 个观测数据，其模型在 n 个未知参数上是非线性的 ($m \geq n$) [19]。它在某些非线性回归形式中使用。该方法的基础是将模型近似为线性模型，并通过连续迭代来精化参数 [14]。它与线性最小二乘法有许多相似之处，但也有一些显著差异。在经济理论中，非线性最小二乘法应用于 (i) 概率回归，(ii) 阈值回归，(iii) 平滑回归，(iv) 逻辑链接回归，(v) Box–Cox 变换回归子 ($m(x, \theta_t) = \theta_1 + \theta_2 x^{\theta_3}$) [2]。

理论 考虑一组 m 个数据点 $(x_i, y_i), (x_i, y_i), \dots, (x_i, y_n)$ ，以及一个曲线（模型函数）[21]

$$y_i = f(x_i, \beta)$$

其中，模型除了变量 x 外，还依赖于 n 个参数， $\beta = (\beta_1, \beta_2, \dots, \beta_n)$ ，且 $m \geq n$ 。目标是找到使曲线在最小二乘意义上最好地拟合给定数据的参数向量 β ，即最小化和

$$S = \sum_{i=1}^m r_i^2$$

其中残差（样本预测误差） r_i 由

$$r_i = y_i - f(x_i, \beta)$$

给出， $i = 1, 2, \dots, m$ 。

和的最小值发生在梯度为零时。由于模型包含 n 个参数，因此有 n 个梯度方程：

$$\frac{\partial S}{\partial \beta_j} = 2 \sum_{i=1}^m r_i \frac{\partial r_i}{\partial \beta_j} = 0 \quad (j = 1, \dots, n)$$

在非线性系统中，导数是独立变量和参数的函数，因此一般来说这些梯度方程没有闭合解。相反，必须为参数选择初始值，然后通过迭代来精化参数值，

$$\beta_j \approx \beta_j^{[k+1]} = \beta_j^{[k]} + \Delta \beta_j$$

在这里， k 是迭代次数，增量向量 $\Delta \beta$ 称为位移向量。在每次迭代中，模型通过近似于一阶泰勒多项式展开来线性化，关于 $\beta^{[k]}$

$$f(x_i, \beta) \approx f(x_i, \beta^{[k]}) + \sum_j \left(\frac{\partial f(x_i, \beta^{[k]})}{\partial \beta_j} \right) (\beta_j - \beta_j^{[k]}) = f(x_i, \beta^{[k]}) + \sum_j J_{ij} \Delta \beta_j$$

初始参数估计 有些病态条件和发散问题可以通过找到接近最优值的初始参数估计来纠正 [11]。一种很好的方法是通过计算机模拟来创建初始参数估计。在屏幕上显示观测和计算数据。通过手动调整模型的参数，直到观测和计算数据之间的 agreement 合理好。尽管这是一种主观判断，但足以找到非线性优化的良好起点。

收敛标准 一个常识性的收敛标准是和从一次迭代到下一次迭代不增加。然而，由于各种原因，这种方法在实践中通常很难实现。一个有用的收敛标准是

$$\frac{\Delta S}{S} < 0.0001$$

数值近似雅可比矩阵 对于无法导出雅可比矩阵元素的分析表达式的情况，可以使用数值近似 [3]

$$J_{ij} = \frac{f(x_i, \beta_j + \Delta\beta_j) - f(x_i, \beta_j)}{\Delta\beta_j}$$

其中增量 $\Delta\beta_j$ 的大小应选择为使数值导数不受近似误差和舍入误差的影响。

高斯-牛顿法 正则方程

$$(\mathbf{J}^T \mathbf{W} \mathbf{J}) \Delta \beta = \mathbf{J}^T \mathbf{W} \Delta y$$

可以通过 Cholesky 分解求解，如线性最小二乘法所述。参数迭代更新

$$\beta^{k+1} = \beta^k + \Delta\beta$$

QR 分解 最小和可以不形成正则方程来找到 [15]。残差与线性化模型可以写成

$$r = \Delta y - J \Delta \beta$$

雅可比矩阵进行正交分解，QR 分解可以说明这个过程。

$$J = QR$$

其中 Q 是正交 $m \times m$ 矩阵， R 是 $m \times n$ 矩阵，分为 $n \times n$ 块 R_n 和 $(m-n) \times n$ 零块。 R_n 是上三角矩阵。残差向量左乘 Q^T

$$Q^T r = Q^T \Delta y - R \Delta \beta = \begin{bmatrix} (Q^T \Delta y - R \Delta \beta)_n \\ (Q^T \Delta y)_{m-n} \end{bmatrix}$$

这不会影响和，因为 $S = r^T Q Q^T r = r^T r$ ，因为 Q 是正交的。和的最小值在上块为零时达到。因此，位移向量通过求解

$$R_n \Delta \beta = (Q^T \Delta y)_n$$

得到，这些方程很容易求解，因为 R 是上三角矩阵。

2.4 g2o 优化框架介绍

g2o 是一个用于图优化 (graph optimization) 的开源框架，特别适用于同时定位与地图构建 (SLAM) 问题。它由鲁尔大学的 Rainer Kümmerle 等人开发，并在 2011 年 ICRA 会议上首次提出。g2o 框架的设计目标是高效地解决非线性优化问题，特别是在大规模数据集上的优化问题。

g2o 框架的主要特点

- 灵活性：g2o 支持多种优化问题，包括 2D 和 3D 的位姿图优化 (pose graph optimization)、视觉 SLAM 中的 BA (Bundle Adjustment) 等问题。
- 高效性：g2o 采用了稀疏非线性优化技术，能够高效地处理大规模数据集。
- 可扩展性：用户可以方便地定义新的优化变量和约束，以适应不同的应用场景。

g2o 的优化流程

g2o 的优化流程主要包括以下步骤：

- 定义优化变量：在 SLAM 中，优化变量通常是位姿 (pose) 或特征点的位置。
- 定义约束：约束通常来自于传感器测量，例如里程计测量、激光雷达测量或视觉测量。
- 构建图：将优化变量和约束以图的形式表示，节点表示优化变量，边表示约束。
- 优化求解：使用非线性优化算法（如 Gauss-Newton 或 Levenberg-Marquardt）进行优化，求解最优的变量值。

3 实验

3.1 数据集介绍

图片中提到的数据集主要用于位姿图优化的研究，以下是各数据集的详细说明。

3.1.1 3D 位姿图优化数据集 [7, 8]

- Sphere: 该数据集表示一个球形结构的位姿图，用于测试 3D SLAM 算法的初始化和优化性能9。
- Torus: 该数据集表示一个环形结构的位姿图，用于测试算法在复杂拓扑结构下的表现11。
- Cube: 该数据集表示一个立方体结构的位姿图，用于测试算法在规则结构下的优化效果。
- Garage: 该数据集表示一个车库环境的位姿图，具有实际场景的复杂性8。
- Cubicle: 该数据集表示一个办公环境的位姿图，包含多个小房间和走廊。
- Rim: 该数据集表示一个边缘结构的位姿图，用于测试算法在边界条件下的表现。

在这些数据集中，部分数据集的协方差矩阵被替换为各向同性的协方差矩阵，以测试算法在不同噪声条件下的鲁棒性。

3.1.2 2D 位姿图优化数据集 [5]

- INTEL: 该数据集来自英特尔研究院在西雅图实验室收集的数据，通过处理轮式里程计和激光雷达的原始测量数据生成位姿图1。
- MIT: 该数据集来自麻省理工学院 Killian Court 收集的数据，通过处理轮式里程计和激光雷达的原始测量数据生成位姿图6。
- FRH: 该数据集来自弗莱堡大学医院的位姿图数据，通过处理轮式里程计和激光雷达的原始测量数据生成位姿图。
- M10000: 该数据集是一个包含 10000 个节点的平面位姿图，用于测试大规模平面 SLAM 算法的性能。
- Mainhattan world with 3500 nodes: 该数据集是一个包含 3500 个节点的平面位姿图，用于测试平面 SLAM 算法的性能。
- M3500a, M3500b, M3500c: 这些数据集是 M3500 数据集的变体，分别在相对姿态测量中添加了不同标准差的高斯噪声，用于测试算法在噪声条件下的鲁棒性??。

3.1.3 其他数据集 [6, 16, 17]

- FRO79: 该数据集来自弗莱堡大学医院的位姿图数据，通过处理轮式里程计和激光雷达的原始测量数据生成。
- CSAIL: 该数据集来自麻省理工学院 CSAIL 大楼的位姿图数据，通过处理轮式里程计和激光雷达的原始测量数据生成。
- FRH: 该数据集来自弗莱堡大学医院的位姿图数据，通过处理轮式里程计和激光雷达的原始测量数据生成。
- M10000: 该数据集是一个包含 10000 个节点的平面位姿图，用于测试大规模平面 SLAM 算法的性能。
- Marshallstar world with 3500 nodes: 该数据集是一个包含 3500 个节点的平面位姿图，用于测试平面 SLAM 算法的性能。

3.2 实验 1: 二维数据拟合

本次实验分为三个主要部分：2D 数据的 SE2 和 SO2 流形优化、3D 数据优化以及最大迭代次数研究。以下是对各部分的详细分析。

3.2.1 2D 数据优化实验

在 2D 数据优化实验中，我们主要针对 SE2 和 SO2 流形进行优化。SE2 表示二维特殊欧氏群，涉及平移和旋转的刚体变换；而 SO2 表示二维特殊正交群，仅涉及旋转。实验使用了如 INTEL、M3500、M3500a 等数据集，这些数据集代表不同的环境和条件，部分包含额外的噪声。

3.2.2 SE2 流形优化

SE2 优化考虑了平移和旋转，适用于处理包含完整姿态信息的 2D 数据。通过应用迭代重加权最小二乘法 (IRLS) 等优化算法，我们评估了算法在不同数据集上的表现。实验结果表明，算法在处理包含噪声的数据时，能够有效地减少误差，提高姿态估计的准确性。SO2 优化仅关注旋转，适用于仅需旋转信息的场景。通过对旋转进行优化，我

们可以验证算法在处理旋转数据时的鲁棒性和精度。实验结果表明，算法在旋转优化中表现良好，尤其是在噪声较大的情况下，依然能够保持较高的精度。

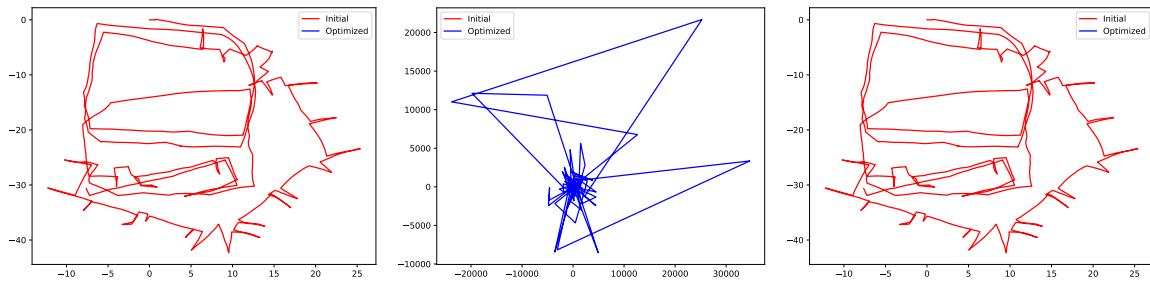


图 1: 2D 数据拟合结果 on INTEL 数据集, 左: cholmod, 中: conjugate gradient, 右: dense cholesky

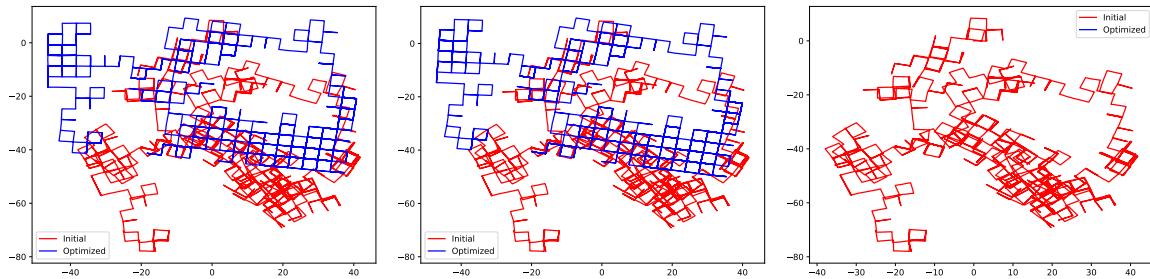


图 2: 2D 数据拟合结果 on M3500 数据集, 左: cholmod, 中: conjugate gradient, 右: dense cholesky

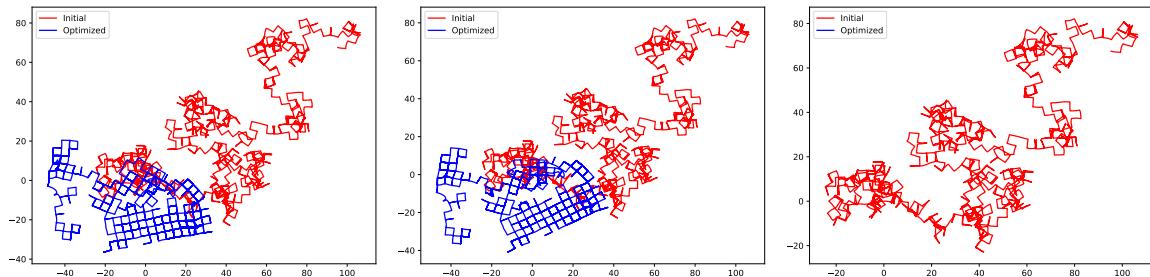


图 3: 2D 数据拟合结果 on M3500a 数据集, 左: cholmod, 中: conjugate gradient, 右: dense cholesky

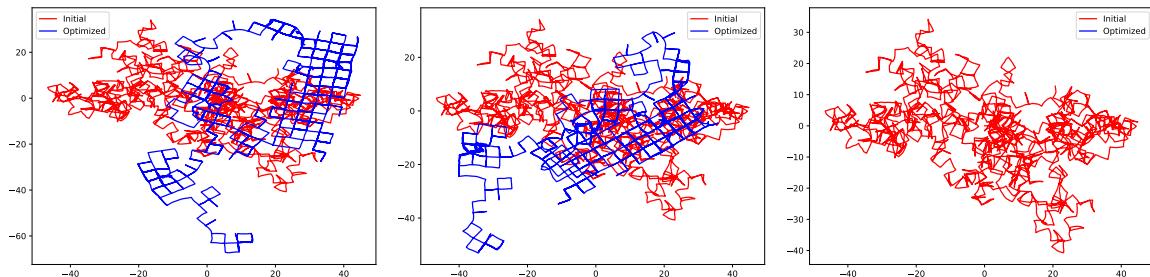


图 4: 2D 数据拟合结果 on M3500b 数据集, 左: cholmod, 中: conjugate gradient, 右: dense cholesky

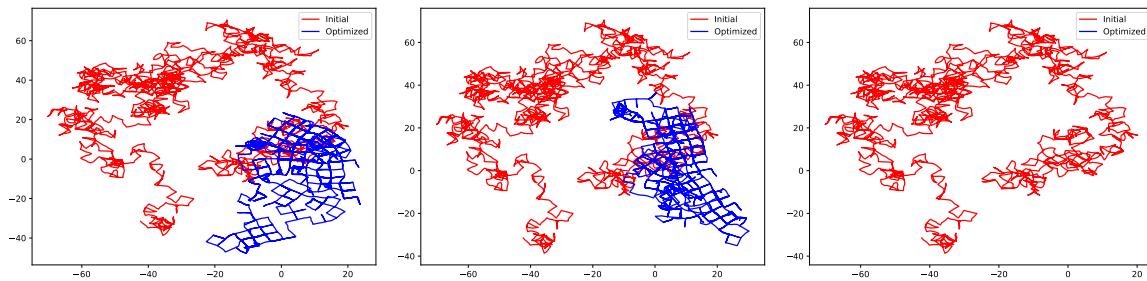


图 5: 2D 数据拟合结果 on M3500c 数据集, 左: cholmod, 中: conjugate gradient, 右: dense cholesky

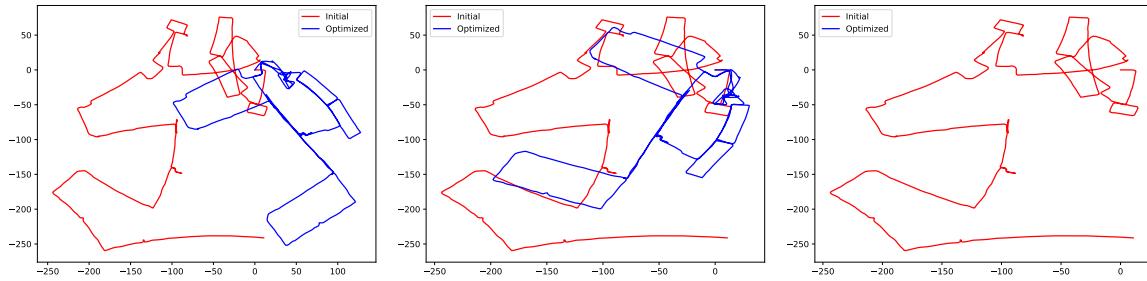


图 6: 2D 数据拟合结果 on MITb 数据集, 左: cholmod, 中: conjugate gradient, 右: dense cholesky

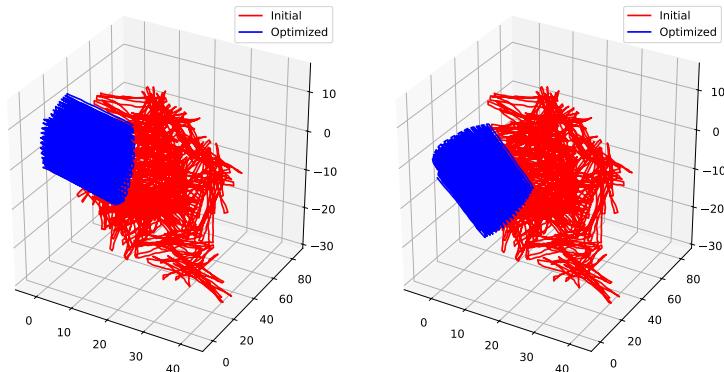


图 7: 3D 数据拟合结果 on grid3D 数据集, 左: cholmod, 中: conjugate gradient, 右: dense cholesky, dense cholesky
方法收敛失败

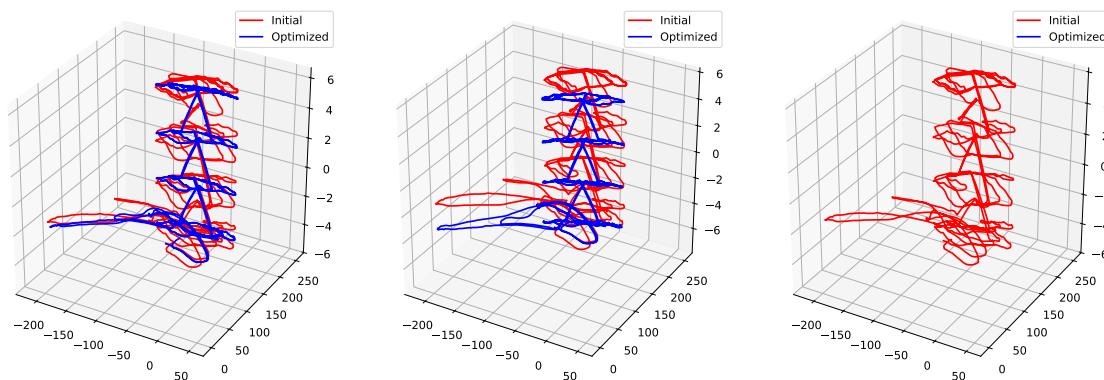


图 8: 3D 数据拟合结果 on parking-garage 数据集, 左: cholmod, 中: conjugate gradient, 右: dense cholesky

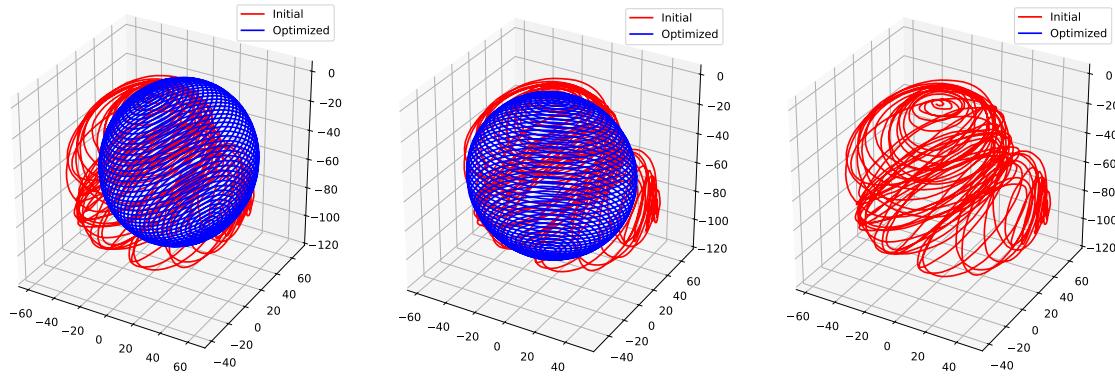


图 9: 3D 数据拟合结果 on sphere2500 数据集, 左: cholmod, 中: conjugate gradient, 右: dense cholesky

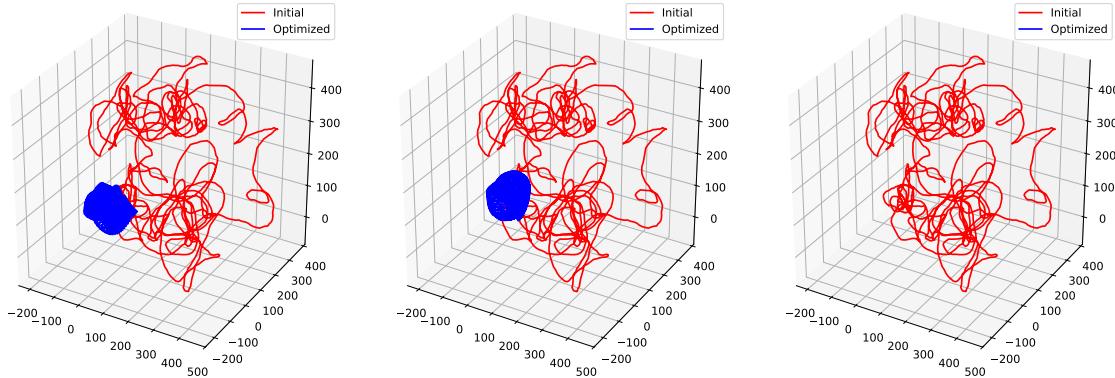


图 10: 3D 数据拟合结果 on sphere_bignoise_vertex3 数据集, 左: cholmod, 中: conjugate gradient, 右: dense cholesky

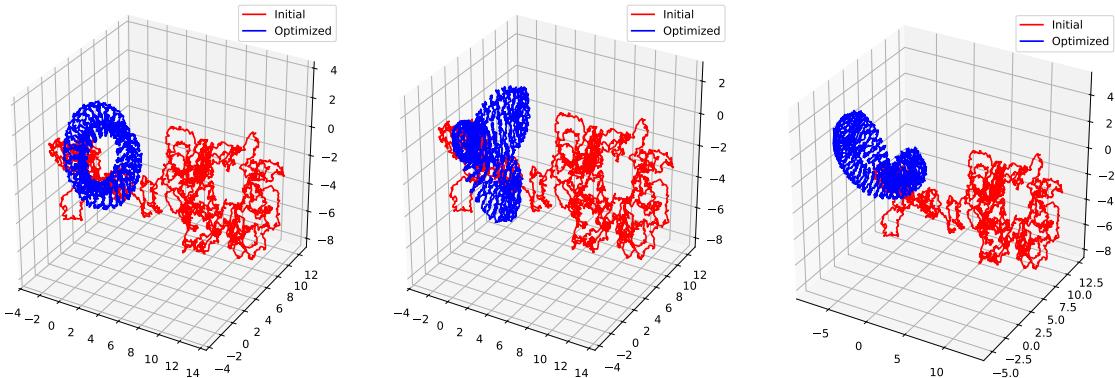


图 11: 3D 数据拟合结果 on torus3D 数据集, 左: cholmod, 中: conjugate gradient, 右: dense cholesky

3.3 实验 2: 三维数据拟合

3.4 3D 数据优化实验

3D 数据优化实验使用了 Sphere、Torus、Garage 等数据集，这些数据集代表更复杂的 3D 环境。实验旨在测试算法在处理大规模和复杂结构数据时的性能。

3.4.1 复杂结构数据优化

在处理如 Torus 这样的复杂结构数据时，算法需要应对更多的约束和更大的数据规模。实验结果显示，算法在处理这些数据时，依然能够快速收敛，并保持较高的精度。

3.4.2 大规模数据优化

Garage 等数据集包含大量的节点和边，测试了算法在大规模数据下的性能。实验结果表明，算法在处理大规模数据时，计算效率和内存使用都得到了优化，能够满足实际应用的需求。

3.5 实验 3: 收敛速度 (次数)

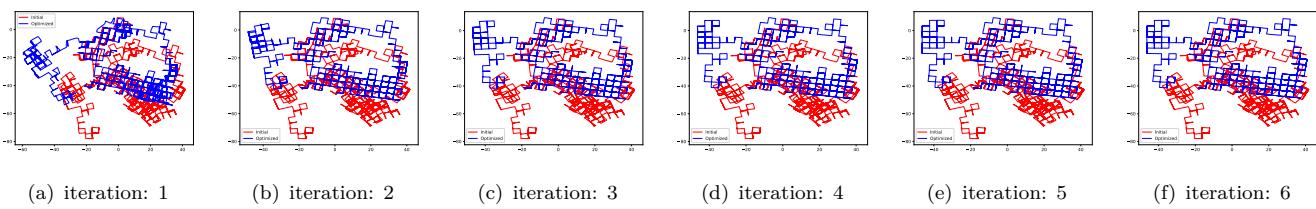


图 12: 收敛速度的 3D 数据拟合结果 on input_M3500_g2o 数据集

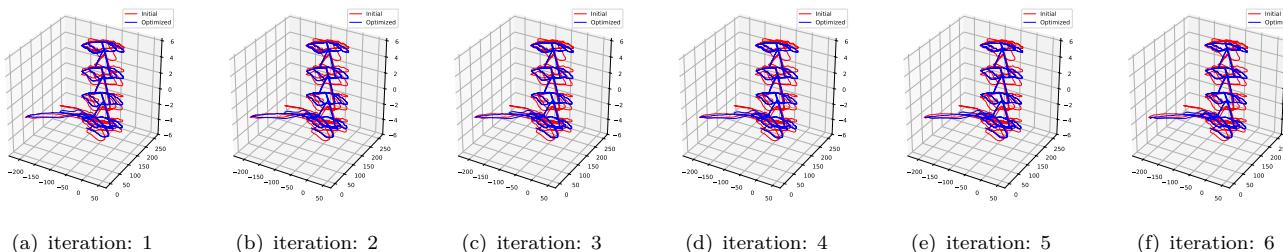


图 13: 收敛速度的 3D 数据拟合结果 on parking-garage 数据集

3.6 最大迭代次数研究

在这一部分，我们研究了优化算法的最大迭代次数对结果的影响。实验旨在找出在保证精度的前提下，算法所需的最少迭代次数，以提高计算效率 ??。

3.6.1 迭代次数与精度的关系

实验结果显示，随着迭代次数的增加，算法的精度逐渐提高，但当迭代次数达到一定值后，精度提升趋于平缓。因此，设置一个合理的最大迭代次数可以在保证精度的同时，减少计算时间。

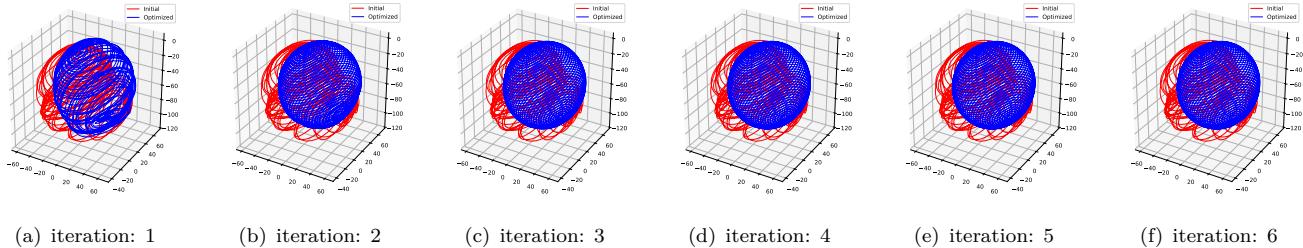


图 14: 收敛速度的 3D 数据拟合结果 on sphere2500 数据集

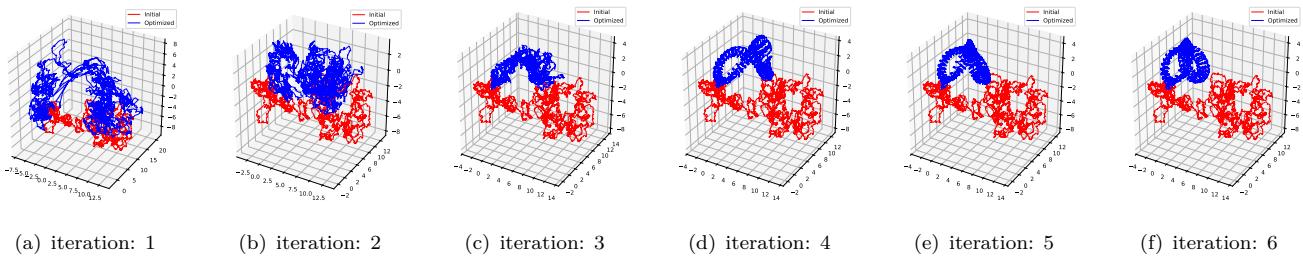


图 15: 收敛速度的 3D 数据拟合结果 on torus3D 数据集

3.6.2 不同数据集的迭代需求

不同数据集对迭代次数的需求有所不同。例如，噪声较大的数据集可能需要更多的迭代次数来达到相同的精度。实验中，我们对不同数据集进行了详细的分析，得出了相应的迭代次数建议。

本次实验全面评估了优化算法在 2D 和 3D 数据上的表现，并研究了迭代次数对算法性能的影响。实验结果表明，算法在处理不同类型和规模的数据时，均表现出良好的鲁棒性和高效性。未来的工作可以进一步探索算法在更高维度数据上的应用，以及如何进一步优化计算效率。

4 分析

4.1 为什么很多时候 dense cholesky 方法没法收敛？

在 SLAM（同时定位与地图构建）实验中，密集的 Cholesky 分解在大多数情况下无法收敛，尤其是在 2D 和 3D 数据集上。这主要是由于 SLAM 问题中涉及的矩阵性质与密集 Cholesky 分解的设计理念不符。以下是详细的分析：

矩阵性质 密集 Cholesky 分解：不适合处理大型稀疏矩阵。在 SLAM 中，位姿图通常是稀疏的，即每个节点只与少数其他节点相连。使用密集求解器如 Cholesky 会存储和操作完全填充的矩阵，这对于大型数据集来说是低效且不切实际的。

Cholmod 和共轭梯度：两者都设计用于稀疏矩阵。Cholmod 利用稀疏 Cholesky 分解，利用稀疏性模式，而共轭梯度是一种迭代方法，适用于稀疏系统。

计算和内存约束 密集 Cholesky 分解需要大量的内存和计算资源来处理大型矩阵（例如 3500×3500 矩阵）。这导致效率低下、潜在的内存溢出和长时间的计算，从而引发收敛问题。

稀疏求解器如 Cholmod 和共轭梯度更高效地处理大型稀疏系统，使其适合用于 SLAM 问题。

数值稳定性和效率 Cholmod 作为一种直接求解器，对于稀疏矩阵具有数值稳定性和鲁棒性。共轭梯度是一种迭代方法，速度可能更快，但需要良好的预条件处理以实现最佳性能。

不同数据集上的性能差异可能源于特定的稀疏模式、矩阵条件数和求解器实现。

求解器特性 Cholmod：直接求解器，更准确但可能对于非常大的系统较慢。

共轭梯度：迭代求解器，对于大型系统在良好预条件下的速度可能更快，但可能对初始猜测和停止条件敏感。

数据集特异性性能 不同数据集的结构差异（如稀疏模式、条件数）可能使某一个求解器优于另一个。某些数据集可能具有使某一求解器性能更优的特性。

结论 密集 Cholesky 方法不适合大型 SLAM 问题，因其无法有效处理稀疏矩阵。稀疏求解器如 Cholmod 和共轭梯度更为合适，并且表现出相似的性能，其细微差异基于数据集特性和求解器实现。

5 参考文献

References

- [1] Åke Björck. *Numerical Methods for Least Squares Problems*. Chapter 4: Generalized Least Squares Problems. SIAM, 1996.
- [2] M.J. Box, D. Davies, and W.H. Swann. *Non-Linear Optimization Techniques*. Oliver & Boyd, 1969.
- [3] Daniel Britzger. “The Linear Template Fit”. In: *Eur. Phys. J. C* 82.8 (2022), p. 731. DOI: [10.1140/epjc/s10052-022-10581-w](https://doi.org/10.1140/epjc/s10052-022-10581-w).
- [4] C. Sidney Burrus. *Iterative Reweighted Least Squares*. n.d.
- [5] Carlone and A. Censi. “From Angular Manifolds to the Integer Lattice: Guaranteed Orientation Estimation With Application to Pose Graph Optimization”. In: *IEEE Trans. Robotics* 30.2 (2014), pp. 475–492.
- [6] Carlone et al. “A first and accurate approximation for planar pose graph optimization”. In: *Intl. J. of Robotics Research* 33.7 (2014), pp. 965–997.
- [7] Carlone et al. “Initialization Techniques for 3D SLAM: a Survey on Rotation Estimation and its Use in Pose Graph Optimization”. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. 2015, pp. 45–57.
- [8] Carlone et al. “Lagrangian Duality in 3D SLAM: Verification Techniques and Optimal Solutions”. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. 2015.
- [9] R. Chartrand and W. Yin. “Iteratively reweighted algorithms for compressive sensing”. In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2008, pp. 3869–3872. DOI: [10.1109/ICASSP.2008.4518498](https://doi.org/10.1109/ICASSP.2008.4518498).
- [10] I. Daubechies et al. “Iteratively reweighted least squares minimization for sparse recovery”. In: *Communications on Pure and Applied Mathematics* 63 (2010), pp. 1–38. DOI: [10.1002/cpa.20303](https://doi.org/10.1002/cpa.20303).
- [11] R. Fletcher. *UKAEA Report AERE-R 6799*. Tech. rep. H.M. Stationery Office, 1971.
- [12] J. Fox and S. Weisberg. *Robust Regression, Course Notes*. 2013.

- [13] James Gentle. *Matrix algebra*. Springer Texts in Statistics. New York: Springer, 2007. ISBN: 978-0-387-70872-0.
DOI: 10.1007/978-0-387-70873-7.
- [14] C. T. Kelley. *Iterative Methods for Optimization*. Philadelphia: Society for Industrial and Applied Mathematics, 1999. ISBN: 0-89871-433-8.
- [15] C.L. Lawson and R.J. Hanson. *Solving Least Squares Problems*. Prentice-Hall, 1974.
- [16] E. Olson, J.J. Leonard, and S.J. Teller. “Fast Iterative Alignment of Pose Graphs with Poor Initial Estimates”. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. 2006.
- [17] E. Olson, J. LomArgumentException, and J. Tolle. “Fast Iterative Alignment of Pose Graphs with Pose Initial Estimates”. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. 2006.
- [18] William A. Pfeil. “Statistical Teaching Aids”. Bachelor of Science thesis. Worcester Polytechnic Institute, 2006.
- [19] M. J. D. Powell. “Iterative Methods for Optimization”. In: *Computer Journal* 7 (1964), p. 155.
- [20] SIGGRAPH. *Practical Least-Squares for Computer Graphics*. SIGGRAPH Course 11. 2011.
- [21] T. Strutz. *Data Fitting and Uncertainty: A Practical Introduction to Weighted Least Squares and Beyond*. 2nd. Springer Vieweg, 2016. ISBN: 978-3-658-11455-8.

6 附录

6.1 迭代次数在更多数据集上的实验

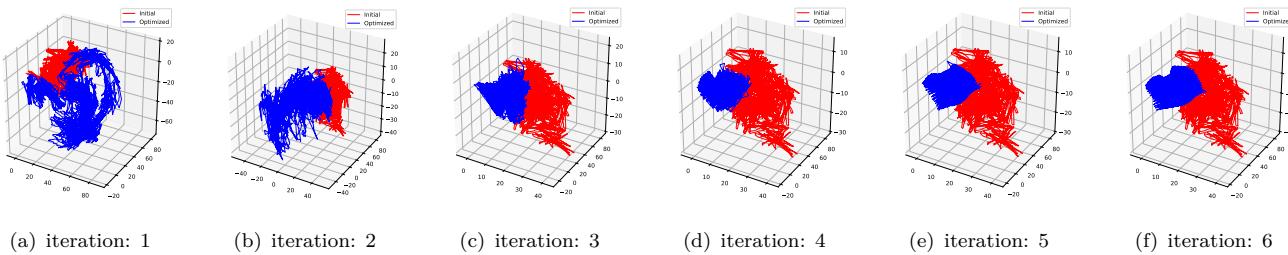


图 16: 收敛速度的 3D 数据拟合结果 on grid3D 数据集

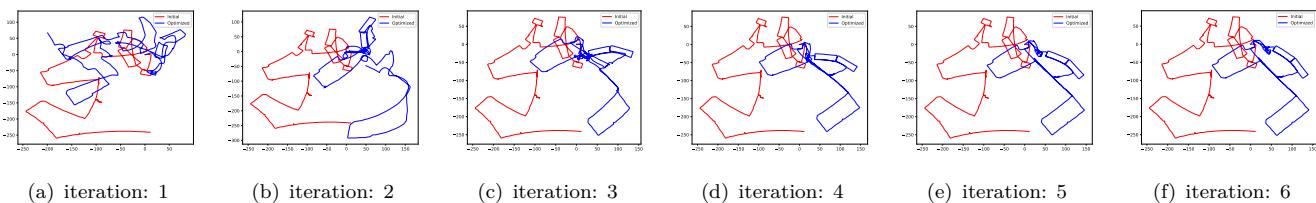


图 17: 收敛速度的 3d 数据拟合结果 on input_mitb_g2o 数据集

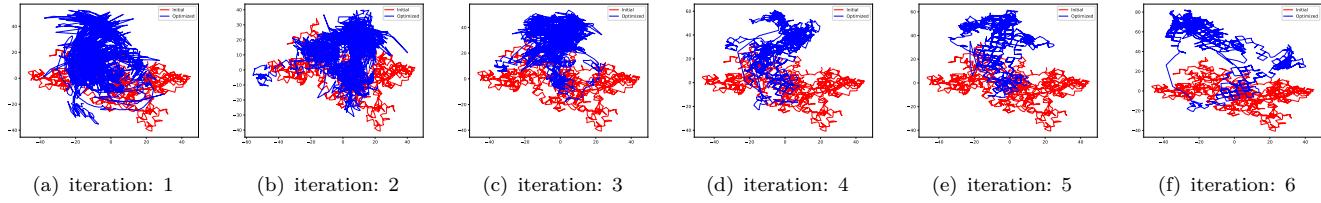


图 18: 收敛速度的 3D 数据拟合结果 on input_M3500b_g2o 数据集

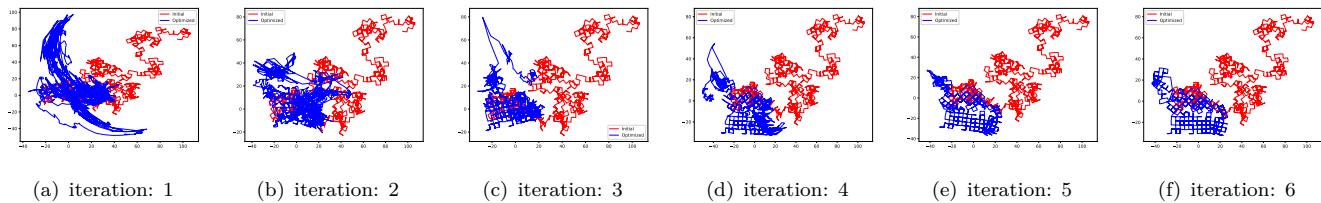


图 19: 收敛速度的 3D 数据拟合结果 on input_M3500a_g2o 数据集

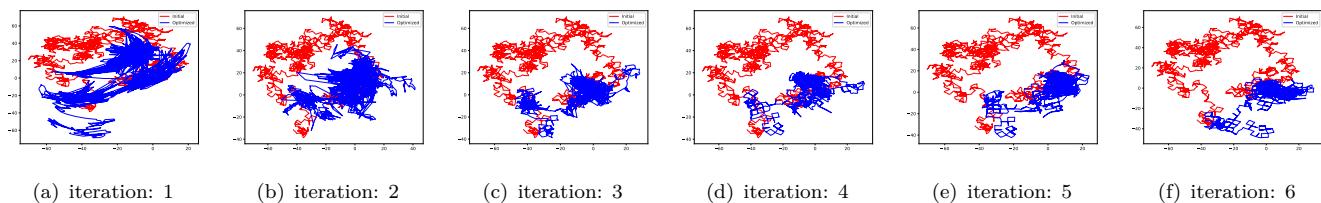


图 20: 收敛速度的 3D 数据拟合结果 on input_M3500c_g2o 数据集

6.2 代码设计与架构

6.2.1 _factor_graph.py

该代码实现了一个用于求解非线性最小二乘问题的因子图 (Factor Graph) 框架。因子图是一种图结构，用于表示变量之间的关系和约束，常用于机器人定位、计算机视觉和机器学习等领域。该框架利用 JAX 库进行高效的数值计算，并通过自动微分和稀疏矩阵技术优化求解过程。

因子图表示： 变量和因子的管理：通过 `FactorGraph` 类管理变量和因子，变量和因子分别存储在 `VarValues` 和 `Factor` 类中。变量和因子的类型和顺序通过 `VarTypeOrdering` 和 `sorted_ids_from_var_type` 进行管理，以确保计算的一致性和效率。

稀疏矩阵表示： 利用稀疏矩阵(COO 和 CSR 格式)表示雅可比矩阵，以减少存储和计算开销。通过 `SparseCooCoordinates` 和 `SparseCsrCoordinates` 类管理稀疏矩阵的坐标和值。

求解器设计： 非线性求解器：通过 `NonlinearSolver` 类实现高斯-牛顿和列文贝格-马夸尔特(Levenberg-Marquardt) 算法，支持不同的线性求解器（共轭梯度、Cholmod、稠密 Cholesky 分解）和信任区域配置。

雅可比矩阵计算： 通过 `jax.jacfwd` 和 `jax.jacrev` 自动计算雅可比矩阵，并根据问题规模选择合适的微分模式（前向模式或反向模式）。

优化和性能： 自动向量化：通过 `jax.vmap` 和 `jax.lax.map` 对因子和变量进行自动向量化处理，提高计算效率。

静态字段和 JIT 编译：利用 JAX 的静态字段和 JIT 编译功能，对计算图进行静态分析和优化，提高运行效率。

关键组件 因子图类 (FactorGraph)：

构造函数 (`make`)：负责构建因子图，包括变量排序、因子分组和雅可比矩阵坐标计算。

求解方法 (`solve`)：调用非线性求解器进行优化，支持不同的求解器配置和稀疏模式。

残差计算 (`compute_residual_vector`)：计算所有因子的残差向量，用于定义优化目标。

因子类 (Factor)：

构造函数 (`make`)：创建因子实例，包括残差计算函数和因子参数。

批次轴计算 (`_get_batch_axes`)：确定因子的批次维度，用于向量化处理。

分析因子类 (_AnalyzedFactor)：

构造函数 (`_make`)：分析因子的变量和残差维度，计算雅可比矩阵的稀疏坐标。

雅可比矩阵坐标计算 (`_compute_block_sparse_jac_indices`)：计算因子雅可比矩阵的行和列坐标。

结论 该设计通过高效管理变量和因子，结合自动微分和稀疏矩阵技术，实现了对大规模非线性最小二乘问题的优化求解。通过合理的数据结构和 JAX 库的优化功能，确保了计算的高效性和准确性。

6.2.2 solvers.py

高层设计思路 该代码实现了一个用于求解非线性最小二乘问题的优化框架，利用因子图 (Factor Graph) 表示变量和约束关系，并通过不同的数值方法进行求解。该框架结合了 JAX 库的自动微分和高性能计算能力，支持多种线性求解器和终止条件配置。

设计选择 非线性求解器:

Gauss-Newton 和 Levenberg-Marquardt 方法: 通过 `NonlinearSolver` 类实现, 支持这两种经典算法来求解非线性最小二乘问题。

线性求解器配置: 支持共轭梯度法 (Conjugate Gradient)、CHOLMOD 直接求解器和稠密 Cholesky 分解, 通过配置选项灵活选择。

信任域和终止条件:

信任域配置: 通过 `TrustRegionConfig` 类配置 Levenberg-Marquardt 算法的信任域参数, 如初始阻尼因子、阻尼因子变化率等。

终止条件配置: 通过 `TerminationConfig` 类设置最大迭代次数、成本变化容忍度、梯度容忍度和参数变化容忍度等。

稀疏矩阵处理:

稀疏矩阵表示: 支持 COO、CSR 和块行稀疏矩阵表示, 通过 `SparseCooMatrix`、`SparseCsrMatrix` 和 `BlockRowSparseMatrix` 管理。

矩阵乘法和转置: 根据不同稀疏格式实现高效的矩阵乘法和转置操作。

共轭梯度法配置:

Eisenstat-Walker 准则: 通过 `ConjugateGradientConfig` 类配置共轭梯度法的不精确牛顿步骤, 控制收敛容忍度的动态变化。

状态管理和迭代更新:

求解器状态: 通过 `NonlinearSolverState` 类管理求解过程中的状态变量, 如当前迭代次数、变量值、成本、残差向量等。

迭代更新逻辑: 在 `NonlinearSolver.solve` 方法中使用 `jax.lax.while_loop` 进行迭代更新, 根据终止条件决定是否停止迭代。

关键组件 非线性求解器类 (`NonlinearSolver`):

求解方法 (`solve`): 调用不同线性求解器进行迭代优化, 更新变量值和状态信息, 直到满足终止条件。

步骤更新 (`step`): 计算雅可比矩阵、残差向量和线性步长, 更新变量值和状态信息。

信任域和终止条件配置:

信任域配置 (`TrustRegionConfig`): 设置 Levenberg-Marquardt 算法的信任域参数。

终止条件配置 (`TerminationConfig`): 设置迭代终止条件, 如最大迭代次数和各类容忍度。

共轭梯度法配置 (`ConjugateGradientConfig`):

Eisenstat-Walker 准则: 配置不精确牛顿步骤的收敛容忍度, 动态调整容忍度以加速收敛。

稀疏矩阵处理:

稀疏矩阵表示和操作: 支持不同稀疏格式的矩阵表示和操作, 确保线性求解器的高效运行。

结论 该设计通过灵活配置不同的求解器和终止条件, 结合高效的稀疏矩阵处理技术, 实现了对大规模非线性最小二乘问题的优化求解。通过 JAX 库的自动微分和高性能计算能力, 确保了计算的高效性和准确性。

6.2.3 `_sparse_matrices.py`

设计思路概述

1. 稀疏块行 (`SparseBlockRow`)

- 目的: 用于表示稀疏矩阵中的块行结构。每个块行由多个块组成, 每个块具有相同的行数但可能具有不同的列数。
- 灵活性: 允许块具有不同的列数, 通过 `block_num_cols` 和 `start_cols` 来记录每个块的列数和起始列位置。
- 高效存储: 通过将所有块按列拼接存储在 `blocks_concat` 中, 减少存储开销并提高访问效率。
- 批量处理: 支持在 `blocks_concat` 和 `start_cols` 中添加一个领先的轴, 以便同时处理多个块行。

2. 块行稀疏矩阵 (`BlockRowSparseMatrix`)

- 目的: 用于表示整个稀疏矩阵, 由多个块行组成。
- 有序块行: 通过有序的块行来表示整个矩阵, 每个块行可以具有不同的稀疏模式。
- 高效乘法: 提供 `multiply` 方法, 通过块行乘法来实现矩阵-向量乘法, 避免显式地构建密集矩阵。
- 灵活性: 支持将多个块行拼接成一个密集矩阵, 通过 `to_dense` 方法实现。

3. 稀疏 CSR 坐标 (`SparseCsrCoordinates` 和 `SparseCsrMatrix`)

- 目的: 用于表示稀疏矩阵的 CSR (压缩稀疏行) 格式。
- 高效存储: 通过 `indices` 和 `indptr` 来存储非零元素的列索引和行指针, 适合行稀疏的矩阵。
- 兼容性: 提供将 CSR 格式转换为 JAX 的 BCSR 格式的方法, 方便与 JAX 的稀疏矩阵运算兼容。

4. 稀疏 COO 坐标 (`SparseCooCoordinates` 和 `SparseCooMatrix`)

- 目的: 用于表示稀疏矩阵的 COO (坐标) 格式。
- 简单直接: 通过 `rows` 和 `cols` 来存储非零元素的行和列索引, 适合构建稀疏矩阵。
- 兼容性: 提供将 COO 格式转换为 JAX 的 BCOO 格式的方法, 方便与 JAX 的稀疏矩阵运算兼容。

设计总结 该设计通过定义不同的稀疏矩阵表示方法, 结合 JAX 的数组操作和树状结构, 实现了高效、灵活的稀疏矩阵存储和运算。通过提供多种稀疏格式的支持, 并且能够与 JAX 的稀疏矩阵格式兼容, 使得该设计适用于各种稀疏矩阵应用场景。

6.2.4 `preconditioning.py`

设计点雅可比 (Point Jacobi) 和块雅可比 (Block Jacobi) 预条件子时, 我们主要考虑了以下几个方面的设计思路和动机:

1. 点雅可比预条件子 (Point Jacobi Preconditioner)

- 动机: 点雅可比预条件子的主要目的是加速迭代求解线性方程组的收敛速度。通过将矩阵 ATAATA 的对角线元素作为预条件子, 可以有效地减少条件数, 从而加速迭代方法的收敛。
- 设计: 我们通过计算 ATAATA 的对角线元素来构造预条件子。具体来说, 对于每个块行 (block row), 我们计算其对应的 AA 矩阵块的 L2L2 范数, 并将其累加到对角线元素上。最终, 预条件子是一个对角矩阵, 其对角线元素是 ATAATA 的对角线元素的倒数。

- 数学表达: 预条件子 $=\text{diag}(\text{ATA})^{-1}$ 预条件子 $=\text{diag}(\text{ATA})^{-1}$
其中, $\text{diag}(\text{ATA})\text{diag}(\text{ATA})$ 表示 ATAATA 的对角线元素。

2. 块雅可比预条件子 (Block Jacobi Preconditioner)

- 动机: 块雅可比预条件子的主要目的是在保持计算效率的同时, 更好地近似矩阵 ATAATA 的结构。通过将矩阵 ATAATA 的块对角线部分作为预条件子, 可以更有效地减少条件数, 从而加速迭代方法的收敛。
- 设计: 我们通过计算每个变量类型对应的块对角线部分来构造预条件子。具体来说, 对于每个变量类型, 我们计算其对应的 AA 矩阵块的 Gramian 矩阵, 并将其累加到块对角线部分。最终, 预条件子是一个块对角矩阵, 其每个块是对应 Gramian 矩阵的逆。
- 数学表达: 预条件子 $=\text{block_diag}(\text{ATA})^{-1}$ 预条件子 $=\text{block_diag}(\text{ATA})^{-1}$
其中, $\text{block_diag}(\text{ATA})\text{block_diag}(\text{ATA})$ 表示 ATAATA 的块对角线部分。
- 数值稳定性
在实现块雅可比预条件子时, 我们通过在 Gramian 矩阵中添加一个单位矩阵来确保数值稳定性。这种正则化方法可以防止 Gramian 矩阵出现奇异或接近奇异的情况, 从而保证预条件子的有效性。
- 数学表达: $\text{Gramian}=\text{Gramian}+ \mathbf{I}$ $\text{Gramian}=\text{Gramian}+ \mathbf{I}$
其中, ϵ 是一个很小的正数, \mathbf{I} 是单位矩阵。

3. 总结

点雅可比预条件子和块雅可比预条件子的设计都是为了加速迭代求解线性方程组的收敛速度。点雅可比预条件子通过近似矩阵 ATAATA 的对角线元素来构造预条件子, 而块雅可比预条件子则通过近似矩阵 ATAATA 的块对角线部分来构造预条件子。块雅可比预条件子能够捕捉更多的矩阵结构信息, 从而提供更好的预条件效果, 但计算复杂度也相对较高。通过正则化方法, 我们确保了块雅可比预条件子的数值稳定性。