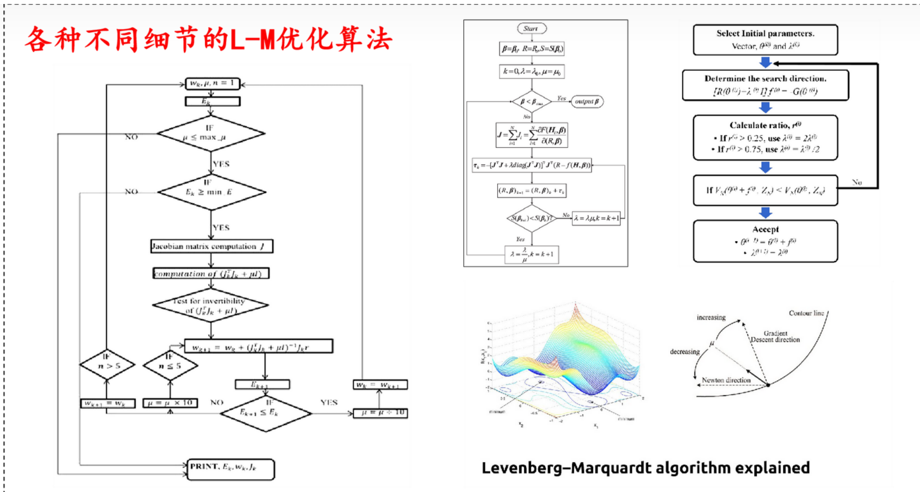

高斯-牛顿优化算法 & L-M优化算法逐行推导

无疆WGH
自动驾驶 | SLAM & 多传感器融合定位

首先post一篇我早期研究Cartographer后端优化的文章，作为**L-M优化算法在SLAM后端中应用**的实例。

马赫WGH：SPA优化算法详解：以Cartographer后端为例
57 赞同 · 16 评论 文章

以下进入正题。

一. 高斯-牛顿优化基础

Levenberg-Marquadt（L-M）优化是高斯-牛顿优化（Gauss-Newton, G-N）的延申，本文先从G-N优化讲起。

G-N优化（L-M优化同此）常用于解决如下形式的优化问题：

$\min_{\mathbf{x}} \frac{1}{2}||f(\mathbf{x})||_2^2$ ------------ 式(1)

其中， $\mathbf{x}$ 是优化变量， $f(\mathbf{x})$ 表示目标函数。

> $f(\mathbf{x})$ 非严格地相当于 $f(\mathbf{x}) \equiv \begin{bmatrix} e_1(\mathbf{x}) \\ e_2(\mathbf{x}) \\ \vdots \\ e_K(\mathbf{x}) \end{bmatrix}$ ，其中 $e_k(\mathbf{x})$ 为列向量。

对于采用位姿图（Pose Graph）形式的SLAM后端来说，上式可以具体化为如下形式：

$\min_{\mathbf{x}} \sum_k^K e_k^T(\mathbf{x})\Omega_k e_k(\mathbf{x})$ ------------ 式(2)

对应的信息矩阵，大小为 $m \times m$ 。我们的优化目标是让所有 K 个误差项的总和最小，也即式(2)。

不难发现，**式(2) 和 式(1) 在形式上是相同的**， $e_k^T(\mathbf{x})\Omega_k e_k(\mathbf{x})$ 不就是 $e_k(\mathbf{x})$ 的2-范数的平方吗？（ Ω_k 只是简单加权而已）由于本文以SLAM后端优化为背景，因此下文将以 式(2) 为背景进行L-M优化的推导。

首先，我们对各个误差项在 $\mathbf{x}$ 附近进行一阶泰勒展开：

$e_k(\mathbf{x}+\Delta\mathbf{x}) \approx e_k(\mathbf{x}) + J_k\Delta\mathbf{x}$ ------------ 式(3)

其中， J_k 是 $e_k(\mathbf{x})$ 关于 $\mathbf{x}$ 的导数，也即**雅可比矩阵**，其大小为 $m \times n$ 。

在G-N优化体系中，我们并不直接求解 $\mathbf{x}$ ，而是寻找一个 $\mathbf{x}$ 的增量 $\Delta\mathbf{x}$ ，使得 $e_k^T(\mathbf{x}+\Delta\mathbf{x})\Omega_k e_k(\mathbf{x}+\Delta\mathbf{x})$ 尽可能的小，也即：

$\min_{\Delta\mathbf{x}} \sum_k^K e_k^T(\mathbf{x}+\Delta\mathbf{x})\Omega_k e_k(\mathbf{x}+\Delta\mathbf{x})$ ------------ 式(4)

> G-N优化是一种迭代优化算法，每一次迭代，都会解得一个 $\Delta\mathbf{x}$ ，然后令 $\mathbf{x}=\mathbf{x}+\Delta\mathbf{x}$ ，完成一次迭代。通常认为，当 $\Delta\mathbf{x}$ 足够小时，可以终止迭代，此时的 $\mathbf{x}$ 作为优化结果输出。

将 式(3) 代入 式(4) 有：

$$
\begin{aligned}
&\min_{\Delta\mathbf{x}} \sum_k^K e_k^T(\mathbf{x}+\Delta\mathbf{x})\Omega_k e_k(\mathbf{x}+\Delta\mathbf{x}) \\
&= \min_{\Delta\mathbf{x}} \sum_k^K [e_k(\mathbf{x})+J_k\Delta\mathbf{x}]^T\Omega_k[e_k(\mathbf{x})+J_k\Delta\mathbf{x}] \\
&= \min_{\Delta\mathbf{x}} \sum_k^K [e_k^T\Omega_k e_k + e_k^T\Omega_k J_k\Delta\mathbf{x} + \Delta\mathbf{x}^T J_k^T\Omega_k e_k + \Delta\mathbf{x}^T J_k^T\Omega_k J_k\Delta\mathbf{x}] \\
&= \min_{\Delta\mathbf{x}} \sum_k^K [e_k^T\Omega_k e_k + 2\times e_k^T\Omega_k J_k\Delta\mathbf{x} + \Delta\mathbf{x}^T J_k^T\Omega_k J_k\Delta\mathbf{x}] \\
&----------- \text{式}(5)
\end{aligned}
$$

> #额外补充#
> 式(5) 变换一下形式，又可以写为：
> $\min_{\Delta\mathbf{x}} \{(\sum_k^K e_k^T\Omega_k e_k) + 2\times(\sum_k^K e_k^T\Omega_k J_k)\Delta\mathbf{x} + \Delta\mathbf{x}^T(\sum_k^K J_k^T\Omega_k J_k)\Delta\mathbf{x}\}$
> $= \min_{\Delta\mathbf{x}} \{\mathbf{C} + 2\mathbf{B}\Delta\mathbf{x} + \Delta\mathbf{x}^T\mathbf{A}\Delta\mathbf{x}\},\ in\ which\ \mathbf{A} = \sum_k^K J_k^T\Omega_k J_k$
> 由此我们发现，该问题是关于 $\Delta\mathbf{x}$ 的二次型！这个发现很重要，实际上决定了 $\Delta\mathbf{x}$ 数值求解的稳定性，并且能解释为什么在SLAM非线性优化问题求解中、我们要求海塞矩阵必须为正定矩阵。
> 具体知识可以参见任意一本线性代数教材。

根据极值条件，**求式(5)右侧关于 $\Delta\mathbf{x}$ 的导数并令其为零**，就可以得到 $\Delta\mathbf{x}$ 的解：

$2\times\sum_k^K J_k^T\Omega_k J_k\Delta\mathbf{x} + 2\times\sum_k^K e_k^T\Omega_k J_k = 0$ ------------ 式(6)

将式(6) 简记为：

$\mathbf{H}\cdot\Delta\mathbf{x} = \mathbf{g}$ ------------ ***式(7)，G-N优化中的增量方程***

> 其中：
> 各个矩阵的维度是 $\mathbf{H}_{n\times n}\cdot\Delta\mathbf{x}_{n\times 1} = \mathbf{g}_{n\times 1}$ ；
> $\mathbf{H} = \sum_k^K J_k^T\Omega_k J_k$ ，称为**海塞矩阵**；
> $\mathbf{g} = -\sum_k^K J_k^T\Omega_k e_k$ ；

式(7) 是一个常规形式的**线性代数方程组**，可以用**线性代数的方法**来求解 $\Delta\mathbf{x}$ 。

不难看出， G-N优化通过在 $\mathbf{x}$ 附近进行近似二阶泰勒展开来简化计算，这也决定了 $\Delta\mathbf{x}$ 只在 $\mathbf{x}$ 附近才有较高的置信度，我们很自然地想到应该给 $\Delta\mathbf{x}$ 添加一个信赖区域(Trust Region)，不让 Δx 过大，于是就产生了 **L-M优化算法。**

L-M优化通过在增量方程中增加一个动态拉格朗日乘子 λ 来改善G-N方法：

$(\mathbf{H}+\lambda\cdot diag\mathbf{H})\Delta\mathbf{x}=\mathbf{g}$ --------- ***式(8)，L-M优化中的增量方程***

> 拉格朗日乘子 λ 设定为正数；由于 $\mathbf{H}$ 是半正定矩阵，所以其对角阵 $diag\mathbf{H}$ 是正定矩阵。这样， $\lambda\uparrow$， $\Delta\mathbf{x}\downarrow$； $\lambda\downarrow$， $\Delta\mathbf{x}\uparrow$。
> 同样地，*式(8)* 用线性代数来求解。

在一次迭代中， $\mathbf{H}$ 和 $\mathbf{g}$ 是不变的，如果我们发觉解出的 $\Delta\mathbf{x}$ 过大，就适当调大 λ ，并重新计算增量方程，以获得相对小一些的 $\Delta\mathbf{x}$ ；

反过来，如果发觉解出的 $\Delta\mathbf{x}$ 在合理的范围内，则适当减小 λ ，减小后的 λ 将用于下次迭代，相当于允许下次迭代时 $\Delta\mathbf{x}$ 有更大的取值，以尽可能地加快收敛速度。

通过这样的调控，L-M算法即保证了收敛的稳定性，又加快了收敛速度。一切都很顺利成章。

But!! Hold on Hold on，似乎还有一个问题：我们如何判定 $\Delta\mathbf{x}$ 是否处在一个合理的范围内？有什么指标吗？

答案：我们可以通过加上 $\Delta\mathbf{x}$ 后，系统的总体误差是变大还是变小，来评价 $\Delta\mathbf{x}$ 的好坏！ （总体误差的计算见下方流程图）

我们优化的目标当然是让总体误差最小，如果加上 $\Delta\mathbf{x}$ 后总体误差反而增大了，就证明 $\Delta\mathbf{x}$ 超出了置信区间，我们需要调大 λ ，重新计算增量；如果总体误差变小，证明 $\Delta\mathbf{x}$ 在合理区间内，调小 λ ，算法继续。

综上，L-M优化算法的执行流程如下：

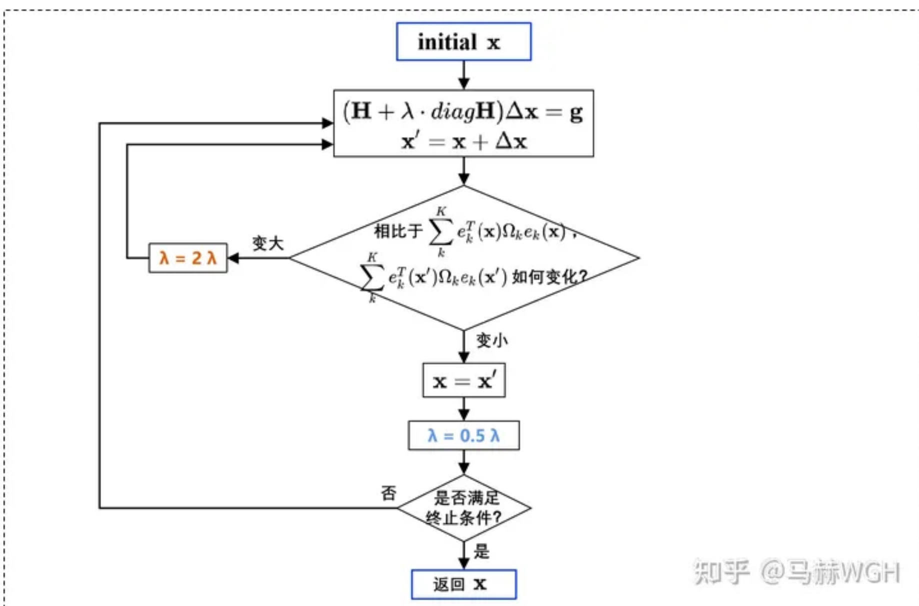

图. L-M优化算法流程图

完毕。

编辑于 2022-09-07 00:33