

教程

非线性最小二乘

介绍

Ceres 可以解决以下形式的非线性最小二乘问题

$$\begin{aligned} \min_{\mathbf{x}} \quad & \frac{1}{2} \sum_i \rho_i \left(\|f_i(x_{i_1}, \dots, x_{i_k})\|^2 \right) \\ \text{s.t.} \quad & l_j \leq x_j \leq u_j \end{aligned} \tag{1}$$

这种形式的问题广泛出现在科学和工程领域——从统计学中的拟合曲线到计算机视觉中根据照片构建3D模型。

在本章中，我们将学习如何使用Ceres Solver解决公式(1)的问题，本章中描述的素示例以及更多示例的完整代码可以在示例目录中找到。

表达式 $\rho_i \left(\|f_i(x_{i_1}, \dots, x_{i_k})\|^2 \right)$ 称为 `ResidualBlock`，其中 $f_i(\cdot)$ 是一个 `CostFunction` 依赖于参数块 $[x_{i_1}, \dots, x_{i_k}]$ 。在大多数优化问题中，小组标量会一起出现。例如，平移向量的三个分量和定义相机姿势的四元数的四个分量。我们将这样一组小标量称为 `ParameterBlock`。当然，`ParameterBlock` 可以只是一个参数。

l_j 和 u_j 是参数块 x_j 的界限。

ρ_i 是 `LossFunction`。`LossFunction` 是一个标量函数，用于减少异常值对非线性最小二乘问题解的影响。

作为特例，当 $\rho_i(x) = x$ ，即恒等函数，且 $l_j = -\infty$ 和 $u_j = \infty$ 时，我们得到了更为熟悉的非线性最小二乘问题。

$$\frac{1}{2} \sum_i \|f_i(x_{i_1}, \dots, x_{i_k})\|^2. \tag{2}$$

Hello World

首先，考虑寻找函数最小值的问题。

$$\frac{1}{2}(10 - x)^2.$$

这是一个很简单的问题，其最小值位于 $x = 10$ ，但它是一个很好的起点，可以说明使用 Ceres 解决问题的基础知识

第一步是编写一个functor来评估这个函数 $f(x) = 10 - x$

```
struct CostFunctor {
    template <typename T>
    bool operator()(const T* const x, T* residual) const {
        residual[0] = 10.0 - x[0];
        return true;
    }
};
```

这里要注意的重要一点是，`operator()` 是一个模板方法，它假定其所有输入和输出都属于某种类型 `T`。

这里使用模板允许 Ceres 调用 `CostFunction::operator<T>()`，当只需要残差的值时使用

`T=double`，当需要雅可比矩阵时使用特殊类型 `T=Jet`。在导数部分，我们将更详细地讨论向 Ceres 提供导数的各种方式。

一旦我们有了计算残差函数的方法，现在就可以使用它构建非线性最小二乘问题并让 Ceres 解决它。

```
int main(int argc, char** argv) {
    google::InitGoogleLogging(argv[0]);

    // 求解变量初值
    double initial_x = 5.0;
    double x = initial_x;

    // 构建问题
    Problem problem;

    // 设置唯一的cost_function（也称为残差）。使用自动微分来求导数（雅可比矩阵）
    CostFunction* cost_function =
        new AutoDiffCostFunction<CostFunction, 1, 1>();
    problem.AddResidualBlock(cost_function, nullptr, &x);

    // 运行求解器！
    Solver::Options options;
    options.linear_solver_type = ceres::DENSE_QR;
    options.minimizer_progress_to_stdout = true;
    Solver::Summary summary;
    Solve(options, &problem, &summary);

    std::cout << summary.BriefReport() << "\n";
    std::cout << "x : " << initial_x
                << " -> " << x << "\n";
    return 0;
}
```

`AutoDiffCostFunction` 以 `CostFunction` 作为输入，自动区分它并为其提供 `CostFunction` 接口。

编译并运行 `examples/helloworld.cc` 得到

```
iter      cost      cost_change  |gradient|  |step|  tr_ratio  tr_radius
ls_iter  iter_time  total_time
  0  4.512500e+01   0.00e+00   9.50e+00   0.00e+00   0.00e+00   1.00e+04
  0  7.87e-06    2.53e-03
  1  4.511598e-07   4.51e+01   9.50e-04   0.00e+00   1.00e+00   3.00e+04
  1  5.48e-05    2.61e-03
  2  5.012552e-16   4.51e-07   3.17e-08   9.50e-04   1.00e+00   9.00e+04
  1  4.05e-06    2.62e-03
Ceres Solver Report: Iterations: 3, Initial cost: 4.512500e+01, Final cost:
5.012552e-16, Termination: CONVERGENCE
x : 0.5 -> 10
```

从 $x = 5$ 开始，求解器经过两次迭代达到 10。细心的读者会注意到这是一个线性问题，一次线性求解就足以获得最优值。求解器的默认配置针对的是非线性问题，为了简单起见，我们在本例中没有更改它。确实可以在一次迭代中使用 Ceres 获得此问题的解。还请注意，求解器在第一次迭代中确实非常接近最优函数值 0。当我们讨论 Ceres 的收敛和参数设置时，我们将更详细地讨论这些问题。

iter	迭代次数	当前的迭代次数，从0开始
cost	代价函数值	目标函数的当前值，表示优化过程中计算的代价值（目标值）。Ceres Solver的目标是最小化这个值。
cost_change	代价变化量	当前迭代与上一次迭代相比，代价函数的变化量。这个值越小，说明优化正在收敛。
gradient	梯度的范数	目标函数的梯度（或偏导数）的范数（大小），反映了代价函数的变化率。梯度越小，说明距离最优解越近。
step	步长的范数	当前迭代中优化变量的更新步长，表示每次迭代中调整的幅度。步长越小，说明调整的幅度越小，通常与收敛接近相关。
tr_ratio	信赖域比率	信赖域子问题的实际代价减少和预测代价减少的比率，反映了信赖域方法的收敛情况。比率接近1说明模型的预测与实际相符，优化效果较好。
tr_radius	信赖域半径	当前迭代中信赖域的半径，表示优化中信赖域的大小。信赖域半径决定了每次迭代的步长。
ls_iter	线搜索迭代次数	在每次迭代中，为找到合适步长而进行的线搜索迭代次数。
iter_time	每次迭代耗时	每次迭代的时间，单位为秒。可以用来评估每次迭代的效率。
total_time	总耗时	从优化过程开始到当前迭代为止的总耗时。