Ceres Solver

一、Ceres 简介

1. **Ceres 是什么**
   - Google 开源的 **非线性优化库**
   - 专注于 **非线性最小二乘问题**：

     $\min_x \sum_i \|f_i(x)\|^2$
   - 可扩展到一般优化问题（有约束 / 无约束）。
2. **为什么要学**
   在机器人和计算机视觉中，大量问题都可以建模为最小二乘优化：
   - 相机标定（Camera Calibration）
   - 位姿估计（Pose Estimation, PnP）
   - 捆绑调整（Bundle Adjustment, SLAM 中核心步骤）
   - 动作预测、轨迹拟合（底盘小陀螺转速拟合、能量机关角速度拟合等）
3. **特性**
   - 自动/数值/手动求导支持
   - 支持稀疏矩阵求解器，适合大规模问题

二、安装

1. **安装方式**

```
sudo apt install cmake libeigen3-dev libgoogle-glog-dev libgflags-dev \
                 libatlas-base-dev libsuitesparse-dev
git clone https://github.com/ceres-solver/ceres-solver.git
mkdir ceres-solver/build && cd ceres-solver/build
cmake ..
make -j4
sudo make install
```

   - **验证**

```
#include <ceres/ceres.h>
#include <iostream>
int main() {
    std::cout << "Ceres installed!" << std::endl;
    return 0;
}
```

三、Ceres 基础概念

1. **Residual Block（残差块）**
   - 优化的基本单元
   - 定义 “观测值” 与 “模型预测值” 的差异
   - 形式：

     residual=observed−f(params)
2. **Cost Function（代价函数）**
   - 用户实现的函数，用于计算残差
   - 需要支持 **自动求导（AutoDiff）** 或手写 Jacobian
3. **Problem（问题对象）**
   - 存放优化变量和残差块
   - 通过 `problem.AddResidualBlock()` 添加
4. **Solver（求解器）**
   - 控制优化过程
   - 典型流程：
     - 构造 `Options`
     - 调用 `ceres::Solve()`
     - 输出结果

四、一维优化

目标：让参数 x 收敛到目标值 10。

```
#include "ceres/ceres.h"
#include "glog/logging.h"

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

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

  double x = 0.0;  // 初始值
  ceres::Problem problem;
```

```
  // 使用 AutoDiff 包装残差函数
  problem.AddResidualBlock(
      new ceres::AutoDiffCostFunction<CostFunctor, 1, 1>(new CostFunctor),
      nullptr, &x);

  // 配置求解器
  ceres::Solver::Options options;
  options.linear_solver_type = ceres::DENSE_QR;
  options.minimizer_progress_to_stdout = true;

  ceres::Solver::Summary summary;
  ceres::Solve(options, &problem, &summary);

  std::cout << summary.BriefReport() << "\n";
  std::cout << "Final x = " << x << "\n";
  return 0;
}
```

运行后输出：

```
iter      cost      cost_change   ...
   0  100.000000
   1    0.000000   100.000000
...
Final x = 10
```

五、曲线拟合

任务：拟合模型 y = e^{ax+b}，估计参数 a, b。

```
std::vector<double> x_data, y_data;
double a_true = 0.3, b_true = 0.1;
std::default_random_engine gen;
std::normal_distribution<double> noise(0.0, 0.2);
for (int i = 0; i < 100; ++i) {
    double x = i / 100.0;
    x_data.push_back(x);
    y_data.push_back(exp(a_true * x + b_true) + noise(gen));
}
```

定义残差

```
struct ExponentialResidual {
  ExponentialResidual(double x, double y) : x_(x), y_(y) {}
  template <typename T>
  bool operator()(const T* const ab, T* residual) const {
    residual[0] = y_ - exp(ab[0] * x_ + ab[1]);
    return true;
  }
  double x_, y_;
};
```

构建优化问题

```
double ab[2] = {0.0, 0.0};
ceres::Problem problem;
for (int i = 0; i < x_data.size(); ++i) {
  problem.AddResidualBlock(
      new ceres::AutoDiffCostFunction<ExponentialResidual, 1, 2>(
          new ExponentialResidual(x_data[i], y_data[i])),
      nullptr, ab);
}
```

运行优化

```
ceres::Solver::Options options;
options.linear_solver_type = ceres::DENSE_QR;
options.minimizer_progress_to_stdout = true;

ceres::Solver::Summary summary;
ceres::Solve(options, &problem, &summary);

std::cout << "Estimated a, b = " << ab[0] << ", " << ab[1] << std::endl;
```

结果会接近 `a_true=0.3, b_true=0.1`。

六、技巧

1. **鲁棒核函数**（Robust Loss）
   - 应对异常值

   ```
   ceres::LossFunction* loss = new ceres::HuberLoss(1.0);
   problem.AddResidualBlock(cost_function, loss, params);
   ```

2. **参数约束**
   - 固定参数：`problem.SetParameterBlockConstant()`
   - 设置上下界：`problem.SetParameterLowerBound(g, 9, 11)`

3. **选择求解器**
    - 小规模问题：`DENSE_QR`
    - **大规模稀疏问题：`SPARSE_SCHUR`**

七、TASK

我们提供一个视频，该视频描述了一个弹丸的飞行轨迹，假设弹道模型满足下面的公式（$t_0 = 0$）

$$\Delta t = t - t_0$$
$$x(t) = x_0 + \frac{v_{x0}}{k}(1 - e^{-k\Delta t})$$
$$y(t) = y_0 + \frac{(v_{y0} + \frac{g}{k})}{k}(1 - e^{-k\Delta t}) - \frac{g}{k}\Delta t$$

请根据视频使用ceres库拟合弹丸的初始速度（单位px/s），以及g(px/s^2)和k$(1/s)$），

其中，g和k的范围分别为：100-1000(px/s^2), 0.01-1$(1/s)$，所提供视频FPS=60

要求拟合参数的误差在3%以内，我们会综合程序优雅性和效率给出排名。