Ceres 小组学习成果总结

目录

一、 Ceres 官网教程讲解

1. 求解优化问题

Ceres 问题求解主要分成以下三部分：
（1）构建 cost function，即代价函数，也就是寻优的目标式(通过预测结果和测量值求误差的函数)。这个部分需要使用函子（functor）这一技巧来实现；
（2）通过上一步的代价函数构建待求解的优化问题；
（3）配置求解器参数并求解问题，这个步骤就是设置方程怎么求解、求解过程是否输出等，然后调用一下 Solve 方法。a

问题：求解 x 值使得1/2(10−x)2最小
代码为：

```
#include<iostream>
#include<ceres/ceres.h>
using namespace std;

//第一部分：构建代价函数
struct CostFunctor {
   //模板函数，泛化函数类型
   template <typename T>
   //重载符号（），仿函数；传入待优化变量列表和承接残差的变量列表，三个 const 意思，从左到右依次是修饰指针不可改变指向的内容，修饰变量 x，修饰成员函数，防止成员函数修改被调用对象的值
   bool operator()(const T* const x, T* residual) const {
    //残差计算步骤
      residual[0] = T(0.5)*(T(10.0) - x[0])*(T(10.0) - x[0]);//1/2(10−x)^2
      return true;
   }
};

//主函数
int main(int argc, char** argv)//参考 https://blog.csdn.net/dgreh/article/details/80985928
{
   // 寻优参数 x 的初始值，为5
   double initial_x = 5.0;
   double x = initial_x;

   // 第二部分：构建寻优问题
   //实例化 Problem
   ceres::Problem;
   //代价函数赋值
   //使用自动求导，将之前的代价函数结构体传入，第一个1是输出维度，即残差的维度，第二个1是输入维度，即待寻优参数 x 的维度。
//    ceres::CostFunction* cost_function = new AutoDiffCostFunction<CostFunctor, 1, 1>(new CostFunctor);
   //添加误差项， 1 、上一步实例化后的代价函数 2 、核函数 3 、待优化变量
   problem.AddResidualBlock(new ceres::AutoDiffCostFunction<CostFunctor, 1, 1>(new CostFunctor), nullptr, &x);
```

```
    //第三部分： 配置并运行求解器
    ceres::Solver::Options;
    //配置增量方程的解法，此处为 QR 求解
    options.linear_solver_type = ceres::DENSE_QR;// Ax=b 矩阵分解方法，ceres 库中有 cholesky，SVD
    options.minimizer_progress_to_stdout = true;  //是否输出到 cout
    //优化信息
    ceres::Solver::Summary;
    //求解： 1、求解器 2、实例化 problem 3、优化器
    Solve(options, &problem, &summary);
    //输出优化的简要信息,迭代次数和每次的 cost
    std::cout << summary.BriefReport() << "\n";
   //最终结果
    std::cout << "初始值 x : " << initial_x<< " 迭代到-> " << x << "\n";
    return 0;
}
```

---

2. 分析

第一部分：构建代价函数结构体

CostFunction 结构体中，对括号符号重载的函数中，传入参数有两个，一个是待优化的变量 x，另一个是残差 residual，也就是代价函数的输出。

```
struct CostFunctor {
    //模板函数，泛化函数类型
    template <typename T>
    //传入待优化变量列表和承接残差的变量列表
    bool operator()(const T* const x, T* residual) const {
        //残差计算步骤
        residual[0] = T(0.5)*(T(10.0) - x[0])*(T(10.0) - x[0]);//1/2(10-x)^2
        return true;
    }
};
```

第二部分：通过代价函数构建待求解的优化问题

这一步最主要的部分是残差块添加函数：AddResidualBlock 的使用，涉及到的三个参数分别是（1）自动求导代价函数；（2）是否添加核函数//核函数：Ceres 库中提供的核函数主要有：TrivialLoss 、HuberLoss、 SoftLOneLoss 、 CauchyLoss；减少离群点影响（3）待优化变量。

其中自动求导函数 AutoDiffCostFunction 可以单独进行，它内部的三个参数分别为（1）上述定义的代价函数结构体；（2）参差维度；（3）待优化变量的维度。

第三部分：配置优化器执行优化

这一部分实现求解器实例化；选择求解方式（这里用 QR 分解）;是否输出运行信息；优化器实例化；调用 Slove 函数进行问题求解；简要输出执行信息。

其中 Solve 函数很重要，它负责最后的问题求解，涉及到的三个参数分别是（1）求解器实例化；（2）优化问题实例化；(3)优化器实例化。

二、 SLAM 十四讲第二版第9章 ceres 实践部分

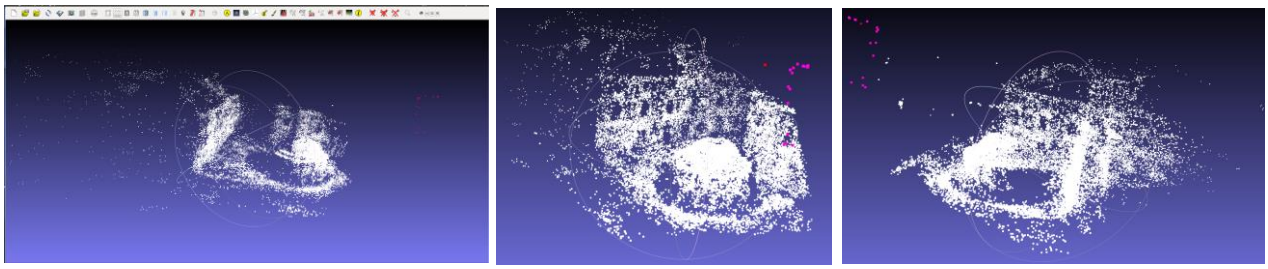
1、程序运行步骤：

cd ch9
mkdir build
cd build
cmake ..
make
./bundle_adjustment_ceres ../problem-16-22106-pre.txt

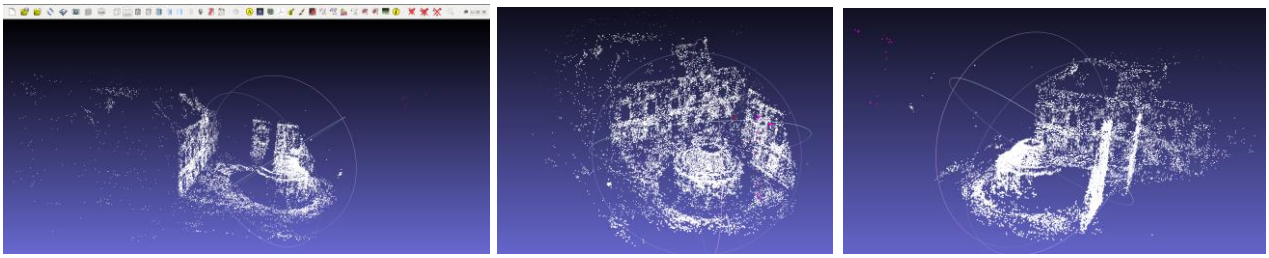
2、运行结果：

```
jack@ubuntu16:~/slambook2/ch9/build$ ./bundle_adjustment_ceres ../problem-16-22106-pre.txt
Header: 16 22106 83718bal problem file loaded...
bal problem have 16 cameras and 22106 points.
Forming 83718 observations.
Solving ceres BA ...
iter      cost      cost_change  |gradient|   |step|    tr_ratio  tr_radius  ls_iter  iter_time  total_time
   0  1.842900e+07    0.00e+00    2.04e+06   0.00e+00   0.00e+00  1.00e+04        0    9.61e-02    2.50e-01
   1  1.449093e+06    1.70e+07    1.75e+06   2.16e+03   1.84e+00  3.00e+04        1    1.69e-01    4.19e-01
   2  5.848543e+04    1.39e+06    1.30e+06   1.55e+03   1.87e+00  9.00e+04        1    1.46e-01    5.64e-01
   3  1.581483e+04    4.27e+04    4.98e+05   4.98e+02   1.29e+00  2.70e+05        1    1.43e-01    7.08e-01
   4  1.251823e+04    3.30e+03    4.64e+04   9.96e+01   1.11e+00  8.10e+05        1    1.44e-01    8.51e-01
   5  1.240936e+04    1.09e+02    9.78e+03   1.33e+01   1.42e+00  2.43e+06        1    1.44e-01    9.95e-01
   6  1.237699e+04    3.24e+01    3.91e+03   5.04e+00   1.70e+00  7.29e+06        1    1.44e-01    1.14e+00
   7  1.236187e+04    1.51e+01    1.96e+03   3.40e+00   1.75e+00  2.19e+07        1    1.43e-01    1.28e+00
   8  1.235405e+04    7.82e+00    1.03e+03   2.40e+00   1.76e+00  6.56e+07        1    1.44e-01    1.43e+00
   9  1.234934e+04    4.71e+00    5.04e+02   1.67e+00   1.87e+00  1.97e+08        1    1.44e-01    1.57e+00
  10  1.234610e+04    3.24e+00    4.31e+02   1.15e+00   1.88e+00  5.90e+08        1    1.44e-01    1.72e+00
  11  1.234386e+04    2.24e+00    3.27e+02   8.44e-01   1.90e+00  1.77e+09        1    1.44e-01    1.86e+00
  12  1.234232e+04    1.54e+00    3.44e+02   6.69e-01   1.82e+00  5.31e+09        1    1.43e-01    2.00e+00
  13  1.234126e+04    1.07e+00    2.21e+02   5.45e-01   1.91e+00  1.59e+10        1    1.43e-01    2.15e+00
  14  1.234047e+04    7.90e-01    1.12e+02   4.84e-01   1.87e+00  4.78e+10        1    1.44e-01    2.29e+00
  15  1.233986e+04    6.07e-01    1.02e+02   4.22e-01   1.95e+00  1.43e+11        1    1.42e-01    2.43e+00
  16  1.233934e+04    5.22e-01    1.03e+02   3.82e-01   1.97e+00  4.30e+11        1    1.42e-01    2.57e+00
  17  1.233891e+04    4.25e-01    1.07e+02   3.46e-01   1.93e+00  1.29e+12        1    1.43e-01    2.72e+00
  18  1.233855e+04    3.59e-01    1.04e+02   3.15e-01   1.96e+00  3.87e+12        1    1.45e-01    2.86e+00
  19  1.233825e+04    3.06e-01    9.27e+01   2.88e-01   1.98e+00  1.16e+13        1    1.44e-01    3.01e+00
  20  1.233799e+04    2.61e-01    1.17e+02   2.16e-01   1.97e+00  3.49e+13        1    1.43e-01    3.15e+00
  21  1.233777e+04    2.18e-01    1.22e+02   1.15e-01   1.97e+00  1.05e+14        1    1.43e-01    3.29e+00
  22  1.233760e+04    1.73e-01    1.10e+02   9.26e-02   1.89e+00  3.14e+14        1    1.42e-01    3.43e+00
  23  1.233746e+04    1.37e-01    1.14e+02   1.01e-01   1.98e+00  9.41e+14        1    1.43e-01    3.58e+00
  24  1.233735e+04    1.13e-01    1.17e+02   3.52e-01   1.96e+00  2.82e+15        1    1.42e-01    3.72e+00
WARNING: Logging before InitGoogleLogging() is written to STDERR
W0505 11:07:15.956624  3283 levenberg_marquardt_strategy.cc:116] Linear solver failure. Failed to compute a step: CHOLMOD warning: Matrix not positive definite.
  25  1.233735e+04    0.00e+00    1.17e+02   0.00e+00   0.00e+00  1.41e+15        1    5.31e-02    3.77e+00
  26  1.233725e+04    9.50e-02    1.21e+02   4.77e-01   1.99e+00  4.24e+15        1    1.38e-01    3.91e+00
W0505 11:07:16.147686  3283 levenberg_marquardt_strategy.cc:116] Linear solver failure. Failed to compute a step: CHOLMOD warning: Matrix not positive definite.
  27  1.233725e+04    0.00e+00    1.21e+02   0.00e+00   0.00e+00  2.12e+15        1    5.28e-02    3.96e+00
W0505 11:07:16.196694  3283 levenberg_marquardt_strategy.cc:116] Linear solver failure. Failed to compute a step: CHOLMOD warning: Matrix not positive definite.
  28  1.233725e+04    0.00e+00    1.21e+02   0.00e+00   0.00e+00  5.30e+14        1    4.90e-02    4.01e+00
  29  1.233718e+04    6.92e-02    5.75e+01   5.74e-02   1.70e+00  1.59e+15        1    1.37e-01    4.15e+00
  30  1.233714e+04    3.65e-02    5.90e+01   3.78e-01   1.93e+00  4.77e+15        1    1.42e-01    4.29e+00
W0505 11:07:16.528515  3283 levenberg_marquardt_strategy.cc:116] Linear solver failure. Failed to compute a step: CHOLMOD warning: Matrix not positive definite.
  31  1.233714e+04    0.00e+00    5.90e+01   0.00e+00   0.00e+00  2.38e+15        1    5.29e-02    4.34e+00
  32  1.233711e+04    3.32e-02    6.11e+01   1.70e+00   2.00e+00  7.15e+15        1    1.38e-01    4.48e+00
W0505 11:07:16.719102  3283 levenberg_marquardt_strategy.cc:116] Linear solver failure. Failed to compute a step: CHOLMOD warning: Matrix not positive definite.
  33  1.233711e+04    0.00e+00    6.11e+01   0.00e+00   0.00e+00  3.57e+15        1    5.30e-02    4.53e+00
W0505 11:07:16.767176  3283 levenberg_marquardt_strategy.cc:116] Linear solver failure. Failed to compute a step: CHOLMOD warning: Matrix not positive definite.
  34  1.233711e+04    0.00e+00    6.11e+01   0.00e+00   0.00e+00  8.94e+14        1    4.80e-02    4.58e+00
  35  1.233708e+04    3.14e-02    6.19e+01   4.45e-01   2.00e+00  2.68e+15        1    1.35e-01    4.72e+00
W0505 11:07:16.955319  3283 levenberg_marquardt_strategy.cc:116] Linear solver failure. Failed to compute a step: CHOLMOD warning: Matrix not positive definite.
  36  1.233708e+04    0.00e+00    6.19e+01   0.00e+00   0.00e+00  1.34e+15        1    5.29e-02    4.77e+00
  37  1.233705e+04    2.50e-02    2.01e+01   3.30e-01   1.68e+00  4.02e+15        1    1.37e-01    4.91e+00
W0505 11:07:17.144937  3283 levenberg_marquardt_strategy.cc:116] Linear solver failure. Failed to compute a step: CHOLMOD warning: Matrix not positive definite.
  38  1.233705e+04    0.00e+00    2.01e+01   0.00e+00   0.00e+00  2.01e+15        1    5.27e-02    4.96e+00
  39  1.233704e+04    1.58e-02    2.08e+01   8.78e-01   1.95e+00  6.03e+15        1    1.38e-01    5.10e+00
  40  1.233702e+04    1.51e-02    2.14e+01   6.36e-01   2.00e+00  1.00e+16        1    1.47e-01    5.24e+00
W0505 11:07:17.482203  3283 levenberg_marquardt_strategy.cc:116] Linear solver failure. Failed to compute a step: CHOLMOD warning: Matrix not positive definite.
  41  1.233702e+04    0.00e+00    2.14e+01   0.00e+00   0.00e+00  5.00e+15        1    5.27e-02    5.30e+00
W0505 11:07:17.530498  3283 levenberg_marquardt_strategy.cc:116] Linear solver failure. Failed to compute a step: CHOLMOD warning: Matrix not positive definite.
  42  1.233702e+04    0.00e+00    2.14e+01   0.00e+00   0.00e+00  1.25e+15        1    4.83e-02    5.35e+00
  43  1.233701e+04    1.48e-02    2.08e+01   3.12e-01   1.99e+00  3.75e+15        1    1.37e-01    5.48e+00
W0505 11:07:17.720423  3283 levenberg_marquardt_strategy.cc:116] Linear solver failure. Failed to compute a step: CHOLMOD warning: Matrix not positive definite.
  44  1.233701e+04    0.00e+00    2.08e+01   0.00e+00   0.00e+00  1.88e+15        1    5.29e-02    5.54e+00
  45  1.233700e+04    1.42e-02    2.10e+01   2.90e-01   1.99e+00  5.62e+15        1    1.37e-01    5.67e+00
W0505 11:07:17.909839  3283 levenberg_marquardt_strategy.cc:116] Linear solver failure. Failed to compute a step: CHOLMOD warning: Matrix not positive definite.
  46  1.233700e+04    0.00e+00    2.10e+01   0.00e+00   0.00e+00  2.81e+15        1    5.28e-02    5.72e+00
W0505 11:07:17.957988  3283 levenberg_marquardt_strategy.cc:116] Linear solver failure. Failed to compute a step: CHOLMOD warning: Matrix not positive definite.
  47  1.233700e+04    0.00e+00    2.10e+01   0.00e+00   0.00e+00  7.03e+14        1    4.81e-02    5.77e+00
  48  1.233698e+04    1.39e-02    2.11e+01   1.20e-01   2.00e+00  2.11e+15        1    1.38e-01    5.91e+00
W0505 11:07:18.148945  3283 levenberg_marquardt_strategy.cc:116] Linear solver failure. Failed to compute a step: CHOLMOD warning: Matrix not positive definite.
  49  1.233698e+04    0.00e+00    2.11e+01   0.00e+00   0.00e+00  1.05e+15        1    5.29e-02    5.96e+00
  50  1.233697e+04    1.36e-02    2.15e+01   2.67e-01   2.00e+00  3.16e+15        1    1.37e-01    6.10e+00

Solver Summary (v 2.0.0-eigen-(3.2.92)-lapack-suitesparse-(4.4.6)-cxsparse-(3.1.4)-eigensparse-no_openmp)

                                     Original                  Reduced
Parameter blocks                        22122                    22122
Parameters                              66462                    66462
Residual blocks                         83718                    83718
Residuals                              167436                   167436

Minimizer                        TRUST_REGION

Sparse linear algebra library    SUITE_SPARSE
Trust region strategy     LEVENBERG_MARQUARDT

                                        Given                     Used
Linear solver                  SPARSE_SCHUR              SPARSE_SCHUR
Threads                                   1                         1
Linear solver ordering            AUTOMATIC             22106,16
Schur structure                       2,3,9                     2,3,9

Cost:
Initial                          1.842900e+07
Final                            1.233697e+04
Change                           1.841667e+07

Minimizer iterations                    51
Successful steps                        37
Unsuccessful steps                      14

Time (in seconds):
Preprocessor                     0.153658

  Residual only evaluation       0.507077 (36)
  Jacobian & residual evaluation 2.270710 (37)
  Linear solver                  2.729041 (50)
Minimizer                        5.948538

Postprocessor                    0.005173
Total                            6.107369

Termination:                   NO_CONVERGENCE (Maximum number of iterations reached. Number of iterations: 50.)
```

原始点云：



优化后点云：



3、基础知识介绍

1)、BAL 数据格式：

官网：https://grail.cs.washington.edu/projects/bal/

其它地址：https://blog.csdn.net/qq_36170626/article/details/100547230

a) 官网解释：

Data Format

Each problem is provided as a bzip2 compressed text file in the following format.

```
<num_cameras> <num_points> <num_observations>
<camera_index_1> <point_index_1> <x_1> <y_1>
...
<camera_index_num_observations> <point_index_num_observations> <x_num_observations> <y_num_observations>
<camera_1>
...
<camera_num_cameras>
<point_1>
...
<point_num_points>
```

Where, there camera and point indices start from 0. Each camera is a set of 9 parameters - R,t,f,k1 and k2. The rotation R is specified as a Rodrigues' vector.

b) 数据详细解释：

```
16 22106 83718
0 0     -3.859900e+02 3.871200e+02
1 0     -3.844000e+01 4.921200e+02
2 0     -6.679200e+02 1.231100e+02
7 0     -5.991800e+02 4.079300e+02
12 0     -7.204300e+02 3.143400e+02
13 0     -1.151300e+02 5.548999e+01
0 1     3.838800e+02 -1.529999e+01
1 1     5.597500e+02 -1.061500e+02
```

第一行：16个相机，22106个点，共进行83718次相机对点的观测

第2行到83719行：

例如： 6 18595 3.775000e+01 4.703003e+01

第6个相机观测18595个点，得到的相机的观测数据为3.775000e+01 4.703003e+01

第83720行到83720 + 16*9 = 83864

共16个相机的9纬参数：-R(轴角3维)，t(3维)，f(焦距 fx=fy)，k1,k2畸变参数

第83864到83864+3*22106=150182 // 由第一行知道一共 22106个点的三维坐标

c) BAL 数据相机投影计算

相机模型：

我们使用了针孔相机模型，我们为每个相机估计了一些参数，旋转矩阵R，平移矩阵t,焦距f,径向失真参数K1,K2,将3D点投影到相机中的公式为：

```
P  =  R * X + t       (conversion from world to camera coordinates)//把世界坐标转换为相机坐标
p  = -P / P.z         (perspective division)//相机坐标归一化处理
p' =  f * r(p) * p    (conversion to pixel coordinates)//转换得到像素坐标
r(p) = 1.0 + k1 * ||p||^2 + k2 * ||p||^4.
```

这给出了像素投影，其中图像的原点是图像的中心，正x轴指向右，正y轴指向上（此外，在相机坐标系中，正z- 轴向后指向，因此相机正在向下看负z轴，如在OpenGL中那样。

2)、 轴角转四元素公式：

轴-角(Axis-Angle)顾名思义就是绕某条单位轴旋转一定角度，从这个意义上看，它构造四元数是非常舒服的，毕竟直观的几何意义有一点点类似，绕单位轴 $\mathbf{u}$ 旋转 θ 的四元数是：

$$\mathbf{q}(w,\mathbf{v}) = (cos\frac{\theta}{2}, \mathbf{u}sin\frac{\theta}{2})$$

3)、std:nth_element()函数：

http://c.biancheng.net/view/566.html

nth_element() 算法和 partial_sort() 不同。应用的范围由它的第一个和第三个参数指定。第二个参数是一个指向第 n 个元素的迭代器。如果这个范围内的元素是完全有序的，nth_dement() 的执行会导致第 n 个元素被放置在适当的位置。这个范围内，在第 n 个元素之前的元素都小于第 n 个元素，而且它后面的每个元素都会比它大。算法默认用 < 运算符来生成这个结果。

下面是一个使用 nth_elemen() 的示例：

```
01. std::vector<int> numbers {22, 7, 93, 45, 19, 56, 88, 12, 8, 7, 15, 10};
02. size_t count {5}; // Index of nth element
03. std::nth_element(std::begin(numbers), std::begin(numbers) + count, std::end(numbers));
```

这里的第 n 个元素是 numbers 容器的第 16 个元素，对应于 numbers[5]，图 1 展示了它的工作方式。

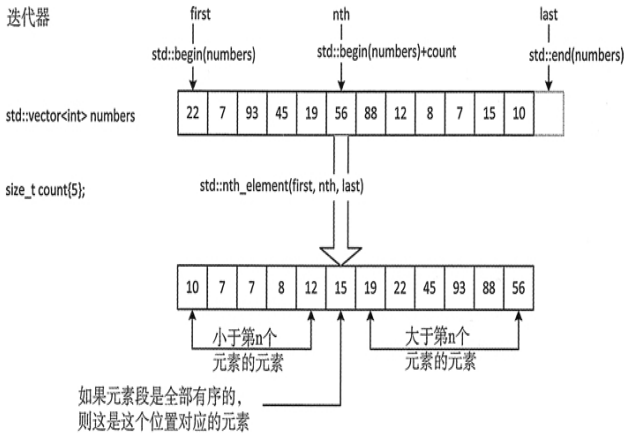


图 1 nth_element() 算法的操作

第 n 个元素之前的元素都小于它，但不必是有序的。同样，第 n 个元素后的元素都大于它，但也不必是有序的。如果第二个参数和第三个参数相同（元素段的末尾），这时这个算法是无效的。

正如本章前面的算法一样，可以自己定义比较函数作为函数的第 4 个参数：

```
01. std::nth_element(std::begin(numbers), std::begin(numbers) + count, std::end(numbers) , std::greater<>());
```

这里使用 > 运算符来比较函数，所以第 n 个元素将是元素按降序排列后的第 n 个元素。第 n 个元素之前的元素都大于它，之后的元素都小于它。如果 number 容器中的初始值和之前的一样，那么结果为：

```
45 56 93 88 22 19 10 12 15 7 8 7
```

在你的系统上，第 n 个元素两边元素的顺序可能会不同，但它左边的元素都应该比它大，而右边的元素都应该比它小。

4）、范数：

https://blog.csdn.net/a493823882/article/details/80569888

3、L1范数

L1范数是我们经常见到的一种范数，它的定义如下：

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$$

表示向量 X 中非零元素的绝对值之和。

L1范数有很多的名字，例如我们熟悉的曼哈顿距离、最小绝对误差等。使用L1范数可以度量两个向量间的差异，如绝对误差和（Sum of Absolute Difference）：

$$SAD(x_1, x_2) = \sum_{i}^{n} |x_{1i} - x_{2i}|$$

对于L1范数，它的优化问题如下：

$$min \|\mathbf{x}\|_1$$
$$s.t.\ Ax = b$$

由于L1范数的天然性质，对L1优化的解是一个稀疏解，因此L1范数也被叫做稀疏规则算子。通过L1可以实现特征的稀疏，去掉一些没有信息的特征，例如在对用户的电影爱好做分类的时候，用户有100个特征，可能只有十几个特征是对分类有用的，大部分特征如身高体重等可能都是无用的，利用L1范数就可以过滤掉。

5）、atan（）函数：

atan2的公式

$$\operatorname{atan2}(y,x) = \begin{cases} \arctan\left(\frac{y}{x}\right) & x > 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & y \geq 0, x < 0 \\ \arctan\left(\frac{y}{x}\right) - \pi & y < 0, x < 0 \\ +\frac{\pi}{2} & y > 0, x = 0 \\ -\frac{\pi}{2} & y < 0, x = 0 \\ \text{undefined} & y = 0, x = 0 \end{cases}$$

atan2的公式

6）、归一化：

https://www.jianshu.com/p/95a8f035c86c

归一化：1）把数据变成(0，1)或者（1,1）之间的小数。主要是为了数据处理方便提出来的，把数据映射到0～1范围之内处理，更加便捷快速。2）把有量纲表达式变成无量纲表达式，便于不同单位或量级的指标能够进行比较和加权。归一化是一种简化计算的方式，即将有量纲的表达式，经过变换，化为无量纲的表达式，成为纯量。

7）、 ceres 线性求解器类型：

```
enum LinearSolverType {
  // These solvers are for general rectangular systems formed from the
  // normal equations A'A x = A'b. They are direct solvers and do not
  // assume any special problem structure.

  // Solve the normal equations using a dense Cholesky solver; based
  // on Eigen.
  DENSE_NORMAL_CHOLESKY,

  // Solve the normal equations using a dense QR solver; based on
  // Eigen.
  DENSE_QR,

  // Solve the normal equations using a sparse cholesky solver; requires
  // SuiteSparse or CXSparse.
  SPARSE_NORMAL_CHOLESKY,

  // Specialized solvers, specific to problems with a generalized
  // bi-partitite structure.

  // Solves the reduced linear system using a dense Cholesky solver;
  // based on Eigen.
  DENSE_SCHUR,

  // Solves the reduced linear system using a sparse Cholesky solver;
  // based on CHOLMOD.
  SPARSE_SCHUR,

  // Solves the reduced linear system using Conjugate Gradients, based
  // on a new Ceres implementation.  Suitable for large scale
  // problems.
  ITERATIVE_SCHUR,

  // Conjugate gradients on the normal equations.
  CGNR
};
```

三、Ceres 官网 BA 程序中的单应矩阵

两个像素点$\boldsymbol{p}_1,\boldsymbol{p}_2$具有如下关系。

$$s_1\boldsymbol{p}_1=\boldsymbol{K}\boldsymbol{P},\quad s_2\boldsymbol{p}_2=\boldsymbol{K}(\boldsymbol{R}\boldsymbol{P}+\boldsymbol{t})$$

s_1,s_2代表尺度意义下相等，记作$sp\simeq p$

所以有$\boldsymbol{p}_1\simeq\boldsymbol{K}\boldsymbol{P},\quad \boldsymbol{p}_2\simeq\boldsymbol{K}(\boldsymbol{R}\boldsymbol{P}+\boldsymbol{t})$

同时定义点$\boldsymbol{p}_1,\boldsymbol{p}_2$在三维空间所在的平面为$\boldsymbol{P}$，则满足方程

$$\boldsymbol{n}^{\mathrm{T}}\boldsymbol{P}+d=0$$

整理有

$$-\frac{\boldsymbol{n}^{\mathrm{T}}\boldsymbol{P}}{d}=1$$

所以

$$\begin{aligned}p_2&\simeq\boldsymbol{K}(\boldsymbol{R}\boldsymbol{P}+\boldsymbol{t})\\&\simeq\boldsymbol{K}\left(\boldsymbol{R}\boldsymbol{P}+\boldsymbol{t}\cdot\left(-\frac{\boldsymbol{n}^{\mathrm{T}}\boldsymbol{P}}{d}\right)\right)\\&\simeq\boldsymbol{K}\left(\boldsymbol{R}-\frac{\boldsymbol{t}\boldsymbol{n}^{\mathrm{T}}}{d}\right)\boldsymbol{P}\\&\simeq\boldsymbol{K}\left(\boldsymbol{R}-\frac{\boldsymbol{t}\boldsymbol{n}^{\mathrm{T}}}{d}\right)\boldsymbol{K}^{-1}\boldsymbol{p}_1\end{aligned}$$

$$\boldsymbol{p}_2\simeq\boldsymbol{H}\boldsymbol{p}_1$$

称$\boldsymbol{H}$为单应矩阵，他是一个直接描述像素坐标$\boldsymbol{p}_1,\boldsymbol{p}_2$变换的矩阵。

四、位姿图优化（一）

Pose Graph 3D (位姿图优化)

有关的 Ceres 的 `pose_graph_3d.cc` 的代码在[Ceres 官方][1]的 GitHub 库里下面会一行行地介绍官方的代码,

相关的有两个文件, 分别是 `pose_graph_3d.cc` 和 `pose_graph_3d_error_term.h`

1. pose_graph_3d.cc

这是主文件用来做整个位姿图优化的, 接下来我们从 `main` 这个函数开始说起

a) main

```
  google::InitGoogleLogging(argv[0]);
  GFLAGS_NAMESPACE::ParseCommandLineFlags(&argc, &argv, true);

  CHECK(FLAGS_input != "") << "Need to specify the filename to read.";
```

这里主要是确认有没有输入一个文件名, 这个文件是用来存着这个位姿图优化的问题的

```
ceres::examples::MapOfPoses poses;
ceres::examples::VectorOfConstraints constraints;
```

位姿图优化需要加载一串的位姿和相关位姿之间的 constraint, 仔细看一下位姿和 constraint 的定义

```
typedef std::map<int,
                 Pose3d,
                 std::less<int>,
                 Eigen::aligned_allocator<std::pair<const int, Pose3d>>>
    MapOfPoses;
```

位姿的定义需要这个位姿的 ID, 3D 的实际位姿

```
typedef std::vector<Constraint3d, Eigen::aligned_allocator<Constraint3d>>
    VectorOfConstraints;
```

constrain的定义需要再深入看一下这个 `Constraint3d` 的定义

```
struct Constraint3d {
  int id_begin;
```

```
  int id_end;

  // The transformation that represents the pose of the end frame E w.r.t. the
  // begin frame B. In other words, it transforms a vector in the E frame to
  // the B frame.
  Pose3d t_be;

  // The inverse of the covariance matrix for the measurement. The order of the
  // entries are x, y, z, delta orientation.
  Eigen::Matrix<double, 6, 6> information;

  // The name of the data type in the g2o file format.
  static std::string name() { return "EDGE_SE3:QUAT"; }

  EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};
```

这里可以看出, 定义一个 constraint, 需要知道这个 constraint 连接的是哪两个位姿, 这里用位姿的 ID 来表示, 另外还需要知道这两个位姿之间的相对位姿. 除此之外, 还需要用到有关这个相对位子的 information matrix, 也就是 inverse covariance matrix

```
CHECK(ceres::examples::ReadG2oFile(FLAGS_input, &poses, &constraints))
    << "Error reading the file: " << FLAGS_input;
```

这里回到 main 的函数, 接下来会把文件里的位姿图问题给加载到之前的两个容器里这里举两个例子

```
VERTEX_SE3:QUAT 7 29.583 0.104272 -0.194878 -0.00678458 0.0141998 0.012983
0.999792
```

在一个 g2o文件里, 这一行可以用来表示一个节点, 也就是之前所说的一个位姿

```
EDGE_SE3:QUAT 879 880 4.1973 0.687599 0.00927741 0.000133331 -0.00104311
0.20361 0.979052 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 4.00002 4.69166e-06 -0.00814815
4 -0.000822411 3.83419
```

这一行可以用来表示两个节点之间的相对位姿

```
ceres::Problem problem;
ceres::examples::BuildOptimizationProblem(constraints, &poses, &problem);
```

b) BuildOptimizationProblem

下面这里可以开始定义我们要解决的位姿图优化的问题, 我们可以进入`BuildOptimizationProblem`来看一下细节

```
ceres::LossFunction* loss_function = NULL;
ceres::LocalParameterization* quaternion_local_parameterization =
    new EigenQuaternionParameterization;
```

这部分是应对quaternion常用的设置

```
for (VectorOfConstraints::const_iterator constraints_iter =
         constraints.begin();
     constraints_iter != constraints.end();
     ++constraints_iter) {
```

接下来要循环每一个加载的constraint

```
const Eigen::Matrix<double, 6, 6> sqrt_information =
    constraint.information.llt().matrixL();
```

对每一个constraint提前做Cholesky Decomposition并且把得到的LL^T的L算出来留待之后所用

```
ceres::CostFunction* cost_function =
    PoseGraph3dErrorTerm::Create(constraint.t_be, sqrt_information);
```

对每一个 constraint, 现在开始创造一个残差函数, 这样会引入到下一个文件

2. pose_graph_3d_error_term.h

```
static ceres::CostFunction* Create(
    const Pose3d& t_ab_measured,
    const Eigen::Matrix<double, 6, 6>& sqrt_information) {
  return new ceres::AutoDiffCostFunction<PoseGraph3dErrorTerm, 6, 3, 4, 3, 4>(
      new PoseGraph3dErrorTerm(t_ab_measured, sqrt_information));
}
```

这里可以看到, 对每一个 constraint 所创造出来的残差函数, 会输入一个 constraint 的测量值, 也就是从之前 g2o 文件的 `EDGE_SE3:QUAT` 提取出来的, 这里的 `sqrt_information` 则是刚才提到的 Cholesky Decomposition里的

这里会发现, 每一个残差函数是输出维度为6的残差

```
bool operator()(const T* const p_a_ptr,
                const T* const q_a_ptr,
                const T* const p_b_ptr,
                const T* const q_b_ptr,
                T* residuals_ptr) const {
```

进入到具体的残差函数的定义, 会发现之前的`<PoseGraph3dErrorTerm, 6, 3, 4, 3, 4>`分别定义了残差, 节点 A 的位置, 节点 A 的旋转, 节点 B 的位置, 节点 B 的旋转的维度

```
// Compute the relative transformation between the two frames.
Eigen::Quaternion<T> q_a_inverse = q_a.conjugate();
Eigen::Quaternion<T> q_ab_estimated = q_a_inverse * q_b;
```

这里通过对 quaternion 的运算, 可以用已知的节点 A 和节点 B 的旋转计算出这两个节点的相对旋转

```
// Represent the displacement between the two frames in the A frame.
Eigen::Matrix<T, 3, 1> p_ab_estimated = q_a_inverse * (p_b - p_a);
```

同理, 可以计算出节点 A 和节点 B 的相对位移

```
// Compute the error between the two orientation estimates.
Eigen::Quaternion<T> delta_q =
    t_ab_measured_.q.template cast<T>() * q_ab_estimated.conjugate();
```

把计算出的相对旋转和之前的测量值相比, 可以得出旋转的残差

```
// Compute the residuals.
// [ position         ]   [ delta_p          ]
// [ orientation (3x1)] = [ 2 * delta_q(0:2) ]
Eigen::Map<Eigen::Matrix<T, 6, 1>> residuals(residuals_ptr); residuals.template
block<3, 1>(0, 0) =
    p_ab_estimated - t_ab_measured_.p.template cast<T>(); residuals.template
block<3, 1>(3, 0) = T(2.0) * delta_q.vec();
```

同理, 可以得出相对位移的残差

```
// Scale the residuals by the measurement uncertainty.
residuals.applyOnTheLeft(sqrt_information_.template cast<T>());
```

将这两种残差(旋转和相对位移)使用之前的L来scale到同一个level

这样就是基本完成了对整个残差函数的定义

```
problem->AddResidualBlock(cost_function,
                          loss_function,
                          pose_begin_iter->second.p.data(),
                          pose_begin_iter->second.q.coeffs().data(),
                          pose_end_iter->second.p.data(),
                          pose_end_iter->second.q.coeffs().data());

problem->SetParameterization(pose_begin_iter->second.q.coeffs().data(),
                             quaternion_local_parameterization);
problem->SetParameterization(pose_end_iter->second.q.coeffs().data(),
                             quaternion_local_parameterization);
```

还剩下的这些步骤就是将残差函数添加到之前定义的`problem`里去

```
// Returns true if the solve was successful.
bool SolveOptimizationProblem(ceres::Problem* problem) {
  CHECK(problem != NULL);

  ceres::Solver::Options options;
  options.max_num_iterations = 200;
  options.linear_solver_type = ceres::SPARSE_NORMAL_CHOLESKY;

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

  std::cout << summary.FullReport() << '\n';

  return summary.IsSolutionUsable();
}
```

接下来可以回到最开始的main函数里并且通过`ceres::SPARSE_NORMAL_CHOLESKY`来解这个问题

[1] https://github.com/ceres-solver/ceres-solver/blob/master/examples/slam/pose_graph_3d/pose_graph_3d.cc

五、位姿图优化（二）

Robot Pose MLE (位姿图优化)

有关的 Ceres 的 `robot_pose_mle.cc` 的代码在[Ceres 官方][1]的 GitHub 库里

下面会一行行地介绍官方的代码

1. robot_pose_mle.cc

这是主文件用来做这个 MLE 的, 接下来来介绍一下这个解决的问题啊

假设有一个机器人, 目前只能在1D 的一个走廊环境里前进, 机器人上有两个读数, 一个是里程计, 一个是对于走廊尽头的 range 读数, 这俩都是有 Gaussian 噪声的, 现在需要通过 MLE 的方法, 利用这两个读数, 解出机器人最优的位姿

Math

Odometry reading: $u = \{u_0, u_1, \ldots, u_n\}$

Range reading: $y = \{y_0, y_1, \ldots, y_n\}$

MLE:

$$
\begin{aligned}
P(u^*_{0:n}|u_{0:n-1}, y_{0:n-1}) &= \frac{P(y_{0:n-1}|u^*_{0:n}, u_{0:n-1})P(u^*_{0:n}|u_{0:n-1})}{\underbrace{P(y_{0:n-1}|u_{0:n-1})}_{ignore}} \\
&\propto P(y_{0:n-1}|u^*_{0:n}, u_{0:n-1})P(u^*_{0:n}|u_{0:n-1}) \\
&= P(y_{0:n-1}|u_{0:n-1})P(u^*_{0:n}|u_{0:n-1}) \\
&= \prod_i P(y_i|u_{0:i})\underbrace{P(u^*_i|u_i)}_{markov}
\end{aligned}
$$

Gaussian probability:

$$P(x) \propto e^{-(\frac{x-\mu}{\sigma})^2}$$

优化u^* 等同于

$$\arg\min_{u^*} \prod_i P(y_i|u_{0:i})P(u^*_i|u_i)$$

a) **main**

```
  google::InitGoogleLogging(argv[0]);
  GFLAGS_NAMESPACE::ParseCommandLineFlags(&argc, &argv, true);
  // Make sure that the arguments parsed are all positive.
  CHECK_GT(FLAGS_corridor_length, 0.0);
  CHECK_GT(FLAGS_pose_separation, 0.0);
  CHECK_GT(FLAGS_odometry_stddev, 0.0);
  CHECK_GT(FLAGS_range_stddev, 0.0);

  vector<double> odometry_values;
  vector<double> range_readings;
  SimulateRobot(&odometry_values, &range_readings);
```

常规操作, 初始化一下这个优化问题, 然后读取一下参数, 这里的`SimulateRobot`其实就是simulate了一下加上噪声的两个reading

```
  for (int i = 0; i < odometry_values.size(); ++i) {
    // Create and add a DynamicAutoDiffCostFunction for the RangeConstraint
from
    // pose i.
    vector<double*> parameter_blocks;
    RangeConstraint::RangeCostFunction* range_cost_function =
        RangeConstraint::Create(
            i, range_readings[i], &odometry_values, &parameter_blocks);
    problem.AddResidualBlock(range_cost_function, NULL, parameter_blocks);

    // Create and add an AutoDiffCostFunction for the OdometryConstraint for
    // pose i.
    problem.AddResidualBlock(OdometryConstraint::Create(odometry_values[i]),
                             NULL,
                             &(odometry_values[i]));
  }
```

对每一个里程计的读数, 需要往 Ceres 的优化问题里加上有关 range 的残差和有关里程计自己的残差

```
  ceres::Solver::Options solver_options;
  solver_options.minimizer_progress_to_stdout = true;

  Solver::Summary summary;
  printf("Solving...\n");
  Solve(solver_options, &problem, &summary);
  printf("Done.\n");
  std::cout << summary.FullReport() << "\n";
  printf("Final values:\n");
  PrintState(odometry_values, range_readings);
  return 0;
```

和一般情况差不多的来用Ceres 解

b) **RangeCostFunction**

下面这里是定义有关 range 的残差

```
struct RangeConstraint {
  typedef DynamicAutoDiffCostFunction<RangeConstraint, kStride>
      RangeCostFunction;

  RangeConstraint(int pose_index,
                  double range_reading,
                  double range_stddev,
                  double corridor_length)
      : pose_index(pose_index),
        range_reading(range_reading),
        range_stddev(range_stddev),
        corridor_length(corridor_length) {}

  template <typename T>
  bool operator()(T const* const* relative_poses, T* residuals) const {
    T global_pose(0);
    for (int i = 0; i <= pose_index; ++i) {
      global_pose += relative_poses[i][0];
    }
    residuals[0] =
        (global_pose + range_reading - corridor_length) / range_stddev;
    return true;
  }

  // Factory method to create a CostFunction from a RangeConstraint to
  // conveniently add to a ceres problem.
  static RangeCostFunction* Create(const int pose_index,
                                   const double range_reading,
                                   vector<double>* odometry_values,
                                   vector<double*>* parameter_blocks) {
    RangeConstraint* constraint = new RangeConstraint(
        pose_index, range_reading, FLAGS_range_stddev, FLAGS_corridor_length);
    RangeCostFunction* cost_function = new RangeCostFunction(constraint);
    // Add all the parameter blocks that affect this constraint.
    parameter_blocks->clear();
    for (int i = 0; i <= pose_index; ++i) {
      parameter_blocks->push_back(&((*odometry_values)[i]));
      cost_function->AddParameterBlock(1);
    }
    cost_function->SetNumResiduals(1);
    return (cost_function);
  }

  const int pose_index;
  const double range_reading;
  const double range_stddev;
  const double corridor_length;
};
```

```
DynamicAutoDiffCostFunction
```

有关 range 残差的所有代码

因为range的残差需要未知数量的里程计读数, 所以这里和之前所有见到过的 `AutoDiffCostFunction` 都不一样, 需要一个可以变的

```
  RangeConstraint(int pose_index,
                  double range_reading,
                  double range_stddev,
                  double corridor_length)
      : pose_index(pose_index),
        range_reading(range_reading),
        range_stddev(range_stddev),
        corridor_length(corridor_length) {}
```

从这里可以看出, 构建有关 range 的残差需要知道当前我所在的位姿 index, 和所有的 range 读数及 standard deviation, 包括走廊的总长

```
  template <typename T>
  bool operator()(T const* const* relative_poses, T* residuals) const {
    T global_pose(0);
    for (int i = 0; i <= pose_index; ++i) {
      global_pose += relative_poses[i][0];
    }
    residuals[0] =
        (global_pose + range_reading - corridor_length) / range_stddev;
    return true;
  }
```

这里看出, `relative_poses` 是里程计的所有读数, 就像之前数学里的一样, 我们对当前位姿的估计需要用到到目前为止所有的里程计读数, 然后 compound 在一起, 然后用从里程计里得到的估计值和通过 range 读数得到的测量值做残差

```
// Factory method to create a CostFunction from a RangeConstraint to //
conveniently add to a ceres problem. static RangeCostFunction* Create(const
int pose_index,                                    const double range_reading,
vector<double>*                                              odometry_values,
vector<double*>* parameter_blocks) {    RangeConstraint* constraint = new
RangeConstraint(
      pose_index, range_reading, FLAGS_range_stddev, FLAGS_corridor_length);
  RangeCostFunction* cost_function = new RangeCostFunction(constraint);    //
Add all the parameter blocks that affect this constraint.
  parameter_blocks->clear();   for (int i = 0; i <= pose_index;
++i) {     parameter_blocks->push_back(&((*odometry_values)[i]));
cost_function->AddParameterBlock(1);
  }      cost_function->SetNumResiduals(1);
return (cost_function);
}
```

这里发现, 给加到 `parameter_blocks`里的就是我们刚才说的里程计的所有读数这样到目前为止是构建了有关

range 的残差

c) OdometryCostFunction

下面是有关里程计的残差

```
struct OdometryConstraint {
  typedef AutoDiffCostFunction<OdometryConstraint, 1, 1> OdometryCostFunction;

  OdometryConstraint(double odometry_mean, double odometry_stddev)
      : odometry_mean(odometry_mean), odometry_stddev(odometry_stddev) {}

  template <typename T>
  bool operator()(const T* const odometry, T* residual) const {
    *residual = (*odometry - odometry_mean) / odometry_stddev;
    return true;
  }

  static OdometryCostFunction* Create(const double odometry_value) {
    return new OdometryCostFunction(
        new OdometryConstraint(odometry_value, FLAGS_odometry_stddev));
  }

  const double odometry_mean;
  const double odometry_stddev;
};
```

这里可以发现，有关里程计，因为是同时优化里程计，所以把里程计的 reading 当做平均值，用优化的里程计值和里程计的 reading 做残差，如果没有 range 的残差的话，最优解肯定就是里程计值等于 reading.

[1] https://github.com/ceres-solver/ceres-solver/blob/master/examples/robot_pose_mle.cc

六．Ceres 自动求导

1. Analytic Derivatives

Ceres Workshop::Automatic Derivatives

On Derivatives:

Notation: Spivak notation

first derivative

$$Df(a) = \frac{d}{dx} f(x)\Big|_{x=a}$$

evaluate at a

k-th derivative at a

$$D^k f(a)$$

bi-variate function

$$D_1 g = \frac{\partial}{\partial x} g(x,y) \quad . \quad D_2 g = \frac{\partial}{\partial y} g(x,y)$$

partial derivative

$$Dg = [D_1 g \;\; D_2 g]^T$$

Jacobian of function $g(x,y)$

Analytic Derivatives: User defines derivatives in the costfunction

Example: fit Rat43 Curve

$$y = \frac{b_1}{(1 + e^{b_2 - b_3 x})^{1/b_4}}$$ → 4 variables to optimize

$$E = \sum_i \left(\frac{b_1}{(1 + e^{b_2 - b_3 x_i})^{1/b_4}} - y_i \right)^2$$

$$D_1 f(b_1, b_2, b_3, b_4; x, y) = \frac{1}{(1 + e^{b_2 - b_3 x})^{1/b_4}}$$

$$D_2 f(b_1, b_2, b_3, b_4; x, y) = \frac{-b_1 e^{b_2 - b_3 x}}{b_4 (1 + e^{b_2 - b_3 x})^{1/b_4 + 1}}$$

$$D_3 f(b_1, b_2, b_3, b_4; x, y) = \frac{b_1 x e^{b_2 - b_3 x}}{b_4 (1 + e^{b_2 - b_3 x})^{1/b_4 + 1}}$$

$$D_4 f(b_1, b_2, b_3, b_4; x, y) = \frac{b_1 \log(1 + e^{b_2 - b_3 x})}{b_4^2 (1 + e^{b_2 - b_3 x})^{1/b_4}}$$

⇒ 可使用技巧提升计算效率！

2. Numeric Derivatives

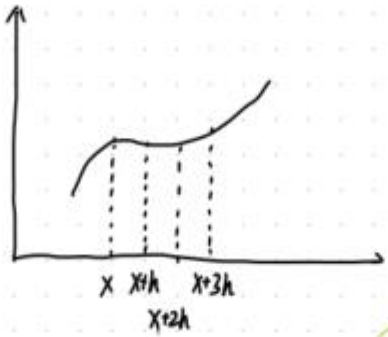
Numeric Derivatives: Approximate derivative with $Df(x) \approx \frac{f(x+h) - f(x)}{h}$

process: define a functor

use Numeric Diff CostFunction to wrap an instance of defined functor

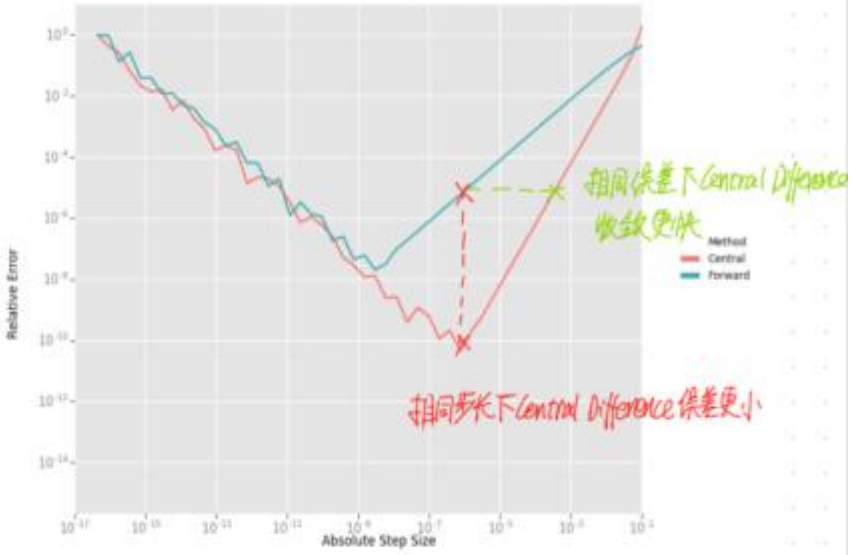
Taylor expansion: $f(x+h) = f(x) + h \cdot Df(x) + \frac{h^2}{2!} D^2 f(x) + \frac{h^3}{3!} D^3 f(x) + \cdots$

$$Df(x) = \frac{f(x+h) - f(x)}{h} - \underbrace{\left[\frac{h^2}{2!} D^2 f(x) + \frac{h^3}{3!} D^3 f(x)\right]}_{O(h),\ \text{error in forward difference}}$$



Central Difference: $Df(x) \approx \frac{f(x+h) - f(x-h)}{2h}$

Need to evaluate twice, more expensive than forward difference!



Ridder's Method: 更大的 Step size h 来避免浮点近似误差

$$Df(x) = \frac{f(x+h) - f(x-h)}{2h} + \underbrace{\frac{h^2}{3!} D^3 f(x) + \frac{h^4}{5!} D^5 f(x) + \cdots}_{O(h^2)}$$

$$= \frac{f(x+h) - f(x-h)}{2h} + K_2 \cdot h^2 + K_4 \cdot h^4 + \cdots$$

only depend on x

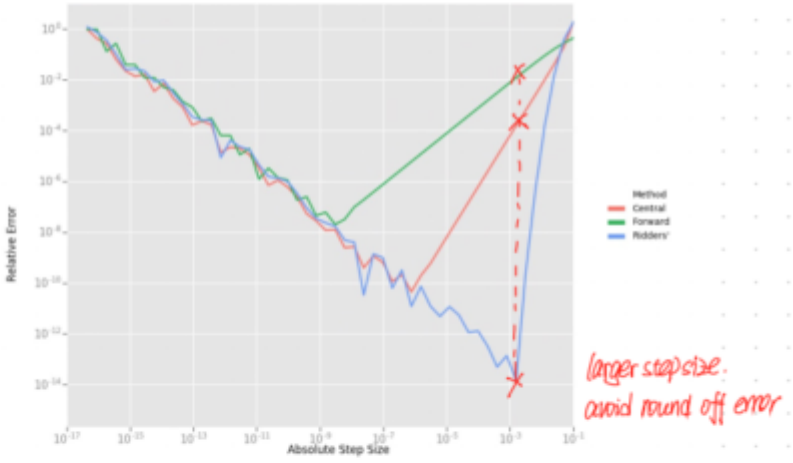
$$A(1,m) = \frac{f(x + h/2^{m-1}) - f(x - h/2^{m-1})}{2h/2^{m-1}}$$

$$Df(x) = A(1,1) + K_2 \cdot h^2 + K_4 \cdot h^4$$

$$Df(x) = A(1,2) + K_2 \cdot \left(\frac{h}{2}\right)^2 + K_4 \cdot \left(\frac{h}{2}\right)^4$$ → 步长减半

$$\hookrightarrow Df(x) = \frac{4 \cdot A(1,2) - A(1,1)}{4-1} + O(h^4)$$

less error than Central Difference!



larger step size: avoid round off error

CostFunction	Time (ns)
Rat43Analytic	255
Rat43AnalyticOptimized	92
Rat43NumericDiffForward	262
Rat43NumericDiffCentral	517
Rat43NumericDiffRidders	3760

→ Very expensive

3. Automatic Derivatives

Automatic Derivatives: Numeric Diff Cost function → Auto Diff Cost function

process: define a costfunctor

Use Auto Diff Costfunction to wrap an instance of defined functor

comparsion:

CostFunction	Time (ns)
Rat43Analytic	255
Rat43AnalyticOptimized	92
Rat43NumericDiffForward	262
Rat43NumericDiffCentral	517
Rat43NumericDiffRidders	3760
Rat43AutomaticDiff	129

Derivation: Dual number and Jets (解决)

对偶数：引入无穷小的量 ε^2 以致 $\varepsilon^2=0$

$f(x) = x^2$

$f(10+x) = (10+\varepsilon)^2$

$= 10^2 + 20\varepsilon + \varepsilon^2$

$= 10^2 + 20\varepsilon$ → $Df(x)$

Taylor expansion: $f(x+\varepsilon) = f(x) + Df(x)\varepsilon + \underbrace{D^2 f(x)\frac{\varepsilon^2}{2} + \cdots}_{0}$

Jets: n-dimension dual number

$X = a + \sum_j v_j \varepsilon_j$

real part　n-dim infinitesimal part (Jacobian)

$f(a+v) = f(a) + Df(a)v$

$f(x_1, \cdots, x_n) = f(a_1, \cdots, a_n) + \underbrace{\sum_i D_i f(a_1, \cdots, a_n) v_i}_{\text{Jacobian}}$

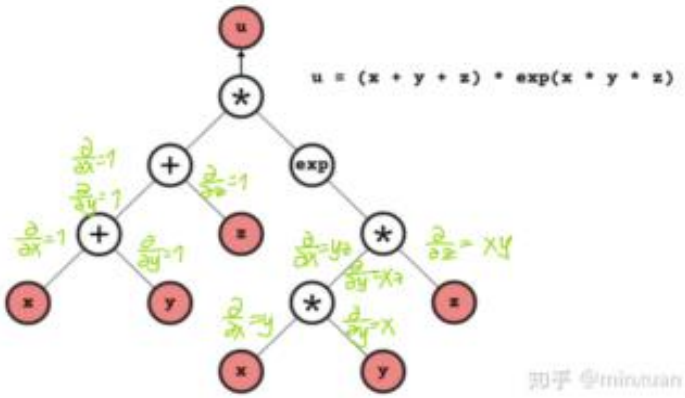
如果 v_i 为标准基向量

$= f(a_1, \cdots, a_n) + \sum_i D_i f(a_1, \cdots, a_n)\varepsilon_i$

↓

通过查找 ε_i 的系数来提取 Jacobian 矩阵

Ceres 使用前向求导，DL 框架使用逆向求导



前向求导：同数值和同数次值对所有变量的导数同时计算

$u=f(x,y,z)$，$\frac{\partial u}{\partial x}$，$\frac{\partial u}{\partial y}$，$\frac{\partial u}{\partial z}$

逆向求导：首先计算函数值，输入函数的变量之间的数学运算存在表达式树中，从树的顶层到底层遍历即可输出对输入变量的导数

$f(x,y)=x^2+xy$ 求 $(1,3)$

$f(1+e_x,3+e_y)=(1+e_x)^2+(1+e_x)\cdot(3+e_y)$

$=1+2e_x+\underbrace{e_x^2}_{>0}+3+e_y+3e_x+\underbrace{e_xe_y}_{=0}$

$=4+5e_x+e_y$

$a \quad \frac{\partial}{\partial x} \quad \frac{\partial}{\partial y}$

In ceres: $e_x=[1,0]$ $e_y=[0,1]$

$x_{dual}=1+e_x$，$y_{dual}=3+e_y$

$f(x_{dual},y_{dual})=x_{dual}^2+x_{dual}*y_{dual}$

$x_{dual}^2=1^2+2e_x=1+[2,0]$

$x_{dual}*y_{dual}=1*3+e_x*3+e_y*1=3+[3,1]$

$f(x_{dual}, y_{dual}) = (1+3) + (2,0] + [3,1]) = 4 + [5,1]$

$a \quad \frac{\partial f}{\partial x} \quad \frac{\partial f}{\partial y}$

Jet.h!

Dynamic Auto Diff Cost function

DynamicAutoDiffCostFunction

class **DynamicAutoDiffCostFunction**

参数的数量/大小在编译时未知

`AutoDiffCostFunction` requires that the number of parameter blocks and their sizes be known at compile time. In a number of applications, this is not enough e.g., Bezier curve fitting, Neural Network training etc.

```
template <typename CostFunctor, int Stride = 4>
class DynamicAutoDiffCostFunction : public CostFunction {
};
```

In such cases `DynamicAutoDiffCostFunction` can be used. Like `AutoDiffCostFunction` the user must define a templated functor, but the signature of the functor differs slightly. The expected interface for the cost functors is:

```
struct MyCostFunctor {
  template<typename T>
  bool operator()(T const* const* parameters, T* residuals) const {
  }
}
```

Since the sizing of the parameters is done at runtime, you must also specify the sizes after creating the dynamic autodiff cost function. For example:

```
DynamicAutoDiffCostFunction<MyCostFunctor, 4>* cost_function =
  new DynamicAutoDiffCostFunction<MyCostFunctor, 4>(
    new MyCostFunctor());
cost_function->AddParameterBlock(5);
cost_function->AddParameterBlock(10);
cost_function->SetNumResiduals(21);
```

在创建动态导数问题时确定参数的尺寸