# GTSAM基本介绍

## 1. 基本介绍

GTSAM(Georgia Tech Smoothing and Mapping)是一个C++编写的因子图工具箱,由佐治亚理工学院开发,主要用于解决SLAM和SFM问题,并且提供了Matlab使用接口(在可视化或用户交互上可能更方便)。

## Defining Factors: SLAM Example

· A factor has an exponential form

$$\phi_i(X_i) = \exp\left(-\frac{1}{2} \parallel f_i(X_i) \parallel_{\Sigma_i}^2\right)$$

A factor has an error function of variables, and a measurement

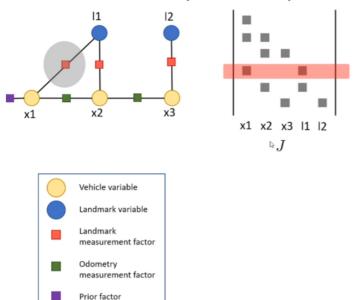
• Prior factor:  $f_{prior}(x_i) = x_i - z$ 

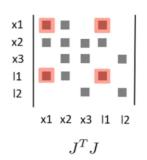
• Odometry factor:  $f_{\text{odometry}}(x_i, x_{i+1}) = (x_{i+1} - x_i) - z$ 

• Landmark factor:  $f_{landmark}(x_i, l_j) = projection(x_i, l_j) - z$ 

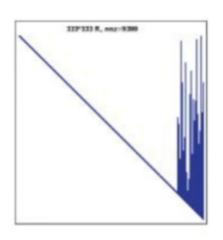
因子就是观测,因子图假设所有观测服从高斯分布。因子图求解就是从全局的角度让状态量接近观测量的过程。最终将这些概率密度函数乘积最大化问题转化为**最小二乘问题**。

# A SLAM Example for Sparsity

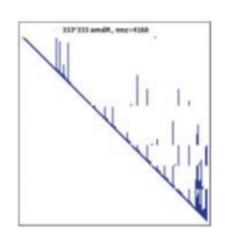




如上图所示,雅可比矩阵行为量测,列为因子,量测只关联了少量的因子使得因子之间是稀疏连接的,Gtsam和Ceres一样,会利用稀疏性提高计算效率,Gtsam会使用COLAMD,近似查找最优的因子排序,让 $J^TJ$ 保持很好的稀疏性。



Original ordering nnz = 9299



COLAMD ordering nnz = 4168

为了求逆,常用的方法是利用Cholesky分解将  $J^TJ$  转化为  $R^TR$  ,R为上三角阵。不同的状态排列顺序会影响最终上三角阵的稀疏性(左边是传统排序-前状态后landmark,右边是COLAMD)

对于常见的SLAM问题,机器人实时运行中,求解规模会逐渐增大,每次增加一个因子,每次求解和上一次求解大部分的因子是完全一样的(Incremental Inference),为了避免重复性的工作,有了iSAM1(增量QR分解)和iSAM2(贝叶斯树)

## 2. 安装

参考: https://gtsam.org/get\_started/

## 2.1 源码安装

### 环境依赖



boost可能出现版本不适配,使用apt install默认安装位置如下,可卸载旧版本后重装

```
1 头文件: /usr/include/boost
2 lib:/usr/lib/x86_64-linux-gnu/libboost*
3 cmake:/usr/lib/x86_64-linux-gnu/cmake/boost*
```

gcc ubuntu20默认gcc9 ubuntu18默认gcc7,若出现版本不适配,可使用**update-alternatives**进行版本管理

### 版本号查看

### **CMakeLists**

```
1 mkdir build
2 cd build
3 cmake ..
4 make check (optional, runs unit tests)
5 sudo make install #这里可能需要sudo权限
```

整个源码都是通过cmake来组织的,默认只会编译C++版本,可以通过配置

GTSAM\_BUILD\_PYTHON=ON或GTSAM\_INSTALL\_MATLAB\_TOOLBOX=ON选项来开启相关工具箱。

默认安装路径

```
1 头文件: /usr/local/include/gtsam
```

2 lib:/usr/local/lib

可通过CMAKE\_INSTALL\_PREFIX来指定目录

```
cmake -DCMAKE_INSTALL_PREFIX:PATH=$HOME ..
```

### ps:我的问题

普通方式安装使用时会出现本地eigen库和gtsam不一致的问题

```
1 cmake -DGTSAM_USE_SYSTEM_EIGEN=ON ..
```

```
-- GTSAM_COMPILE_OPTIONS_PUBLIC_MINSIZEREL :
-- GTSAM_COMPILE_DEFINITIONS_PUBLIC_MINSIZEREL :
-- Use System Eigen : ON (Using version: 3.4.0)
-- Use System Metis : OFF
-- Using Boost version : 1.71.0
-- Use Intel TBB : TBB not found
```

## 2.2 PPA软件源安装

```
1 # Add PPA
2 sudo add-apt-repository ppa:borglab/gtsam-release-4.0
3 sudo apt update # not necessary since Bionic
4 #Install:
5 sudo apt install libgtsam-dev libgtsam-unstable-dev
```

## tips:

当执行 sudo add-apt-repository ppa: 后可在**软件和更新**中会同步配置,当取消选择 某条 ppa,系统将在 /etc/apt/sources.list.d 中的 ppa\_name.list 文件中注释掉仓库条目,如果选择 "删除" 选项,将会删除 /etc/apt/sources.list.d 目录中 ppa\_name.list 文件里的仓库条目。

## 3. 使用GTSAM拟合曲线

### CMakeLists编写

```
1 cmake_minimum_required(VERSION 3.0)
2 project(curve_fitting)
3 set(CMAKE_CXX_STANDARD 11)
4
5 find_package(GTSAM REQUIRED)
6 include_directories(${GTSAM_INCLUDE_DIR})
7
8 include_directories("/usr/local/include/eigen3")
9
10 add_executable(${PROJECT_NAME} curve_fitting.cpp)
11 target_link_libraries(${PROJECT_NAME} gtsam)
```

## 最小系统

```
待优化状态量: gtsam::Values
```

因子图: gtsam::NonlinearFactorGraph

添加因子(量测): graphFactors.add push\_back emplace\_shared

优化器: gtsam::ISAM2

### 主程序

```
1 #include <gtsam/nonlinear/DoglegOptimizer.h>// dogleg
 2 #include <gtsam/nonlinear/GaussNewtonOptimizer.h> // GN
 3 #include <gtsam/nonlinear/LevenbergMarquardtOptimizer.h>
 4 #include <gtsam/inference/Symbol.h>// symbol_shorthand
 5
 6 #include <random>
 7
8 // y = ax^2 + bx
 9 double funct(const gtsam::Vector2& param, const double x) {
       return (param(0)*x*x + param(1)*x);
10
11 }
12
13 // NoiseModelFactor1表示一元边, NoiseModelFactor2表示2元边,一次类推一直到6
14 class curvfitFactor: public gtsam::NoiseModelFactor1<gtsam::Vector2> {
15 public:
       curvfitFactor(gtsam::Key key, const gtsam::SharedNoiseModel& noise, double x
16
           : gtsam::NoiseModelFactor1<gtsam::Vector2>(noise, key), x_(x), y_(y){
17
18
       gtsam::Vector evaluateError(const gtsam::Vector2& param, boost::optional<gts</pre>
19
20
           auto val = funct(param, x_);
           if (H) {
21
22
               gtsam::Matrix Jac = gtsam::Matrix::Zero(1, 2);
23
               Jac << x_* * x_*, x_*;
               (*H) = Jac;
24
25
           }
           return gtsam::Vector1(val - y_);
26
27
       }
28 private:
       double x_, y_;
29
30 };
31
32 int main() {
33
       using gtsam::symbol_shorthand::X;// symbol_shorthand仅用于表示状态变量(char +
       const double sig = 0.05;
34
       std::random_device rd;// 非确定性种子
35
```

```
36
       std::default_random_engine generator(rd());// 构建随即生成器
       std::normal distribution<double> noise(0, sig);// 生成服从正态分布的噪声(均值, ?
37
       gtsam::NonlinearFactorGraph graph;
38
       const gtsam::Vector2 para(2, 3);// 曲线参数a b
39
40
       for (int i = 0; i < 100; ++i) {
41
           // 构造观测值mx my
42
          double mx = i;
43
44
          auto my = funct(para, mx) + noise(generator);
45
          // Isotropic: 各项同性噪声模型,相当于在协方差对角线作放缩
46
          // 对比ceres,无需设置噪声模型,但构建残差时可以设置信息矩阵
47
          auto noiseM = gtsam::noiseModel::Isotropic::Sigma(1, sig);
48
           // 添加因子: 曲线拟合问题是一堆观测,一个状态量,因此只需要X(O)
49
          graph.emplace_shared<curvfitFactor>(X(0), noiseM, mx, my);
50
51
       }
52
53
       // 设置初始值
       gtsam::Values intial;
54
       intial.insert<gtsam::Vector2>(X(0), gtsam::Vector2(1.5, 2.2));
55
56
       // 选择优化方法
57
       // gtsam::GaussNewtonOptimizer opt(graph, intial);
58
       gtsam::LevenbergMarquardtOptimizer opt(graph, intial);
59
       // gtsam::DoglegOptimizer opt(graph, intial);
60
       std::cout << "\ninitial error=" << graph.error(intial) << std::endl;</pre>
61
       auto res = opt.optimize();
62
       std::cout << "\nfinal error=" << graph.error(res) << std::endl;</pre>
63
       gtsam::Vector2 matX0 = res.at<gtsam::Vector2>(X(0));
64
       std::cout << "a = " << matX0[0] << ", b = " << matX0[1] << "\n";
65
66
67
       return 0;
68 }
```

## 4. 对比



A Comparison of Graph Optimizati on.pdf 2.40MB



论文针对非线性位姿图优化问题,将四种框架求解问题方法均设定为LM,针对不同数据集进行对比。

**结论:**数据噪声大的前端,后端中GTSAM、SE-Sync效果好;前端效果良好的情况下,差别不大;时间上,g2o普遍最长

## 对于batch求解而言:

不论Ceres、G2O还是GTSAM,都支持稀疏矩阵求解功能,因此都可以实现最小二乘问题的快速求解。 个人认为效率上的区别:数值求导/解析求导/自动求导 > 框架差别。

### 使用体感差别会大些:

- 1. G2O文档支持较差,不支持自动求导,但具备Vetex/Edge的概念,方便构建图优化问题,官方提供了不少顶点和边
- 2. Ceres教程完善,支持三种求导方式,使用灵活性较大,属于偏底层的优化问题求解工具
- 3. GTSAM文档较多,不支持自动求导,接口丰富,适合直接上手使用,上层功能完善

ps: 网上有一些gtsam嵌套ceres自动求导的教程,感兴趣可以了解一下