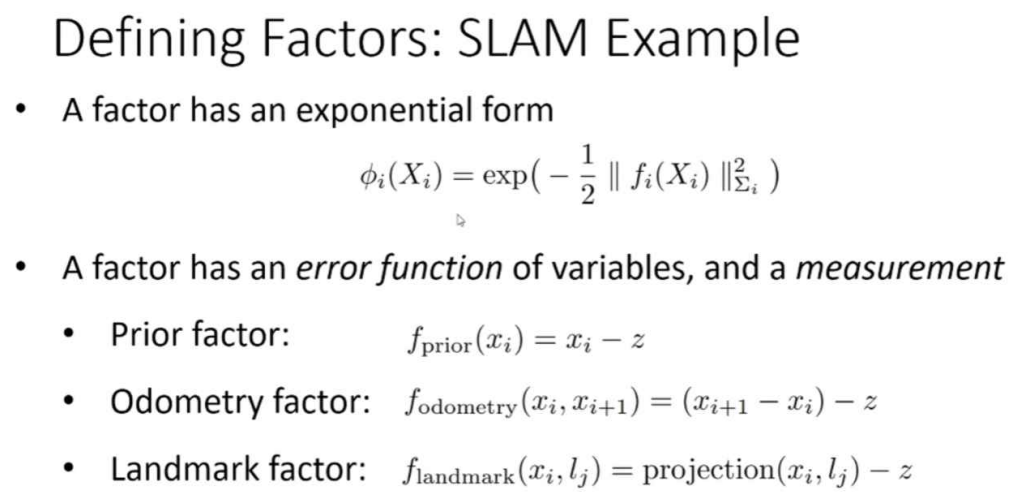
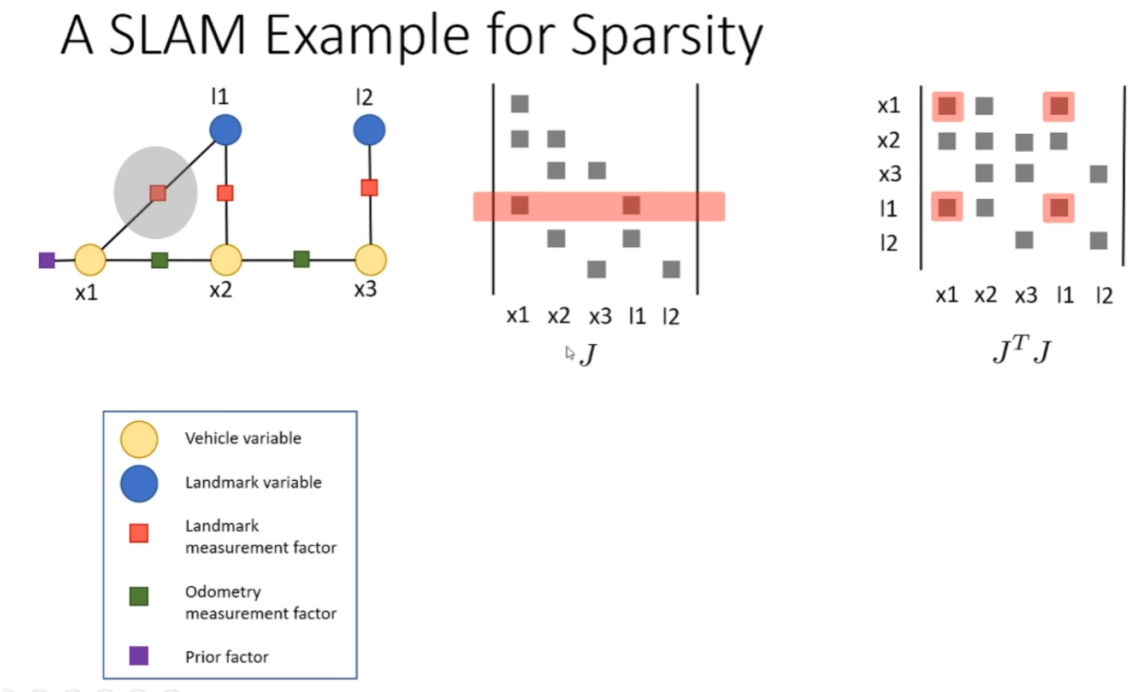
GTSAM基本介绍

1. 基本介绍

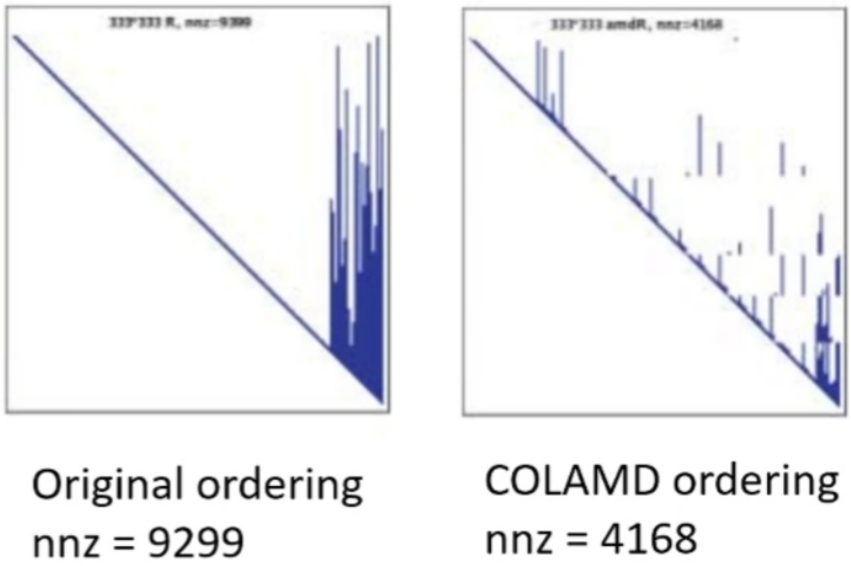
GTSAM（Georgia Tech Smoothing and Mapping）是一个C++编写的因子图工具箱，由佐治亚理工学院开发，主要用于解决SLAM和SFM问题，并且提供了Matlab使用接口（在可视化或用户交互上可能更方便）。



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**。



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 $J^T J$ 保持很好的稀疏性。



为了求逆，常用的方法是利用Cholesky分解将 $J^T J$ 转化为 $R^T R$，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）

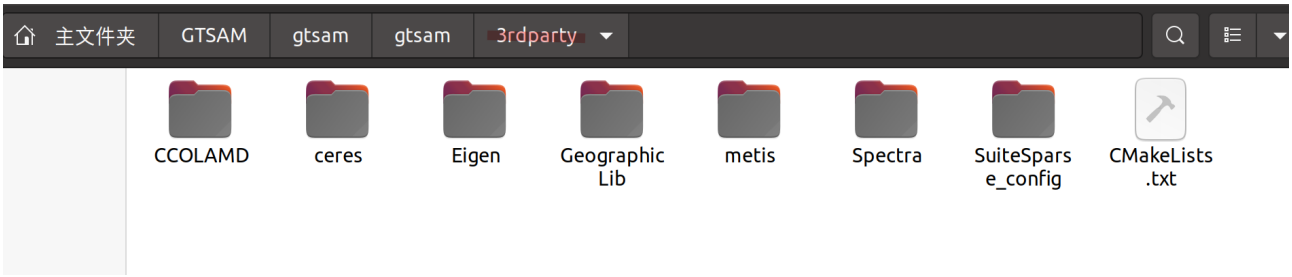
[参考] 深蓝学院:因子图的理论基础与在机器人中的应用

2. 安装

参考：https://gtsam.org/get_started/

2.1 源码安装

环境依赖



Prerequisites:

- Boost >= 1.43 (Ubuntu: `sudo apt-get install libboost-all-dev`)
- CMake >= 3.0 (Ubuntu: `sudo apt-get install cmake`)
- A modern compiler, i.e., at least gcc 4.7.3 on Linux.

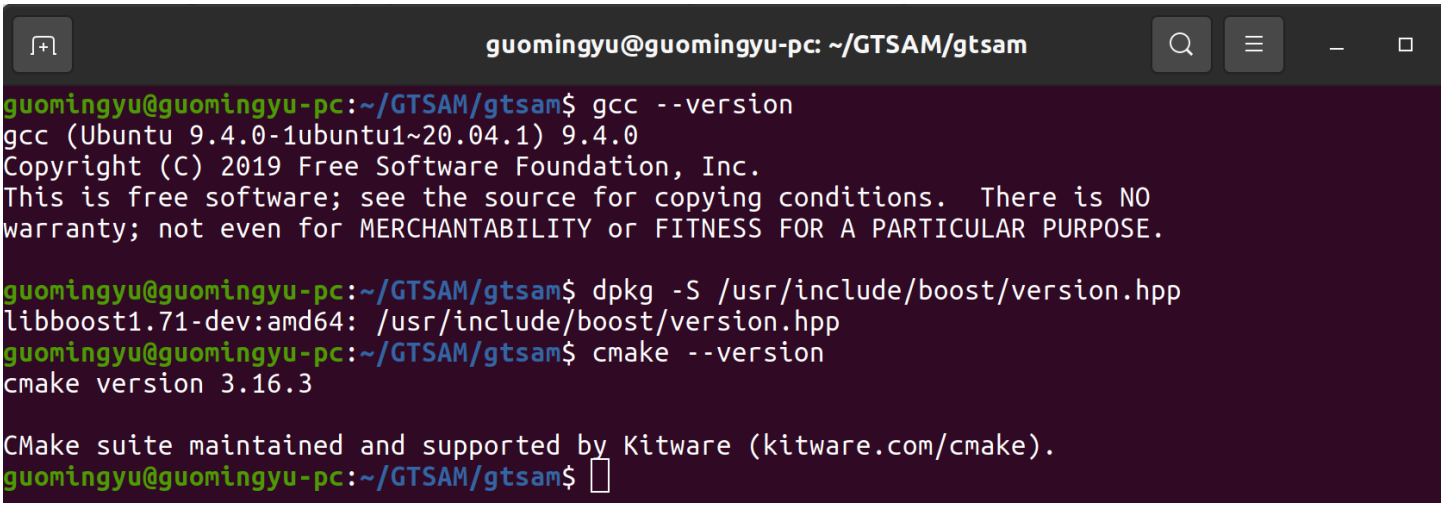
```
git clone https://github.com/borglab/gtsam.git
sudo apt-get install libboost-all-dev
sudo apt-get install cmake
```

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

```
头文件：/usr/include/boost
lib:/usr/lib/x86_64-linux-gnu/libboost*
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

```
mkdir build
cd build
cmake ..
make check (optional, runs unit tests)
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

```
头文件：/usr/local/include/gtsam
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

```
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

```
# Add PPA
sudo add-apt-repository ppa:borglab/gtsam-release-4.0
sudo apt update  # not necessary since Bionic
#Install:
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

```
cmake_minimum_required(VERSION 3.0)
project(curve_fitting)
set(CMAKE_CXX_STANDARD 11)

find_package(GTSAM REQUIRED)
include_directories(${GTSAM_INCLUDE_DIR})

include_directories("/usr/local/include/eigen3")

add_executable(${PROJECT_NAME} curve_fitting.cpp)
target_link_libraries(${PROJECT_NAME} gtsam)
```

注意链接库名是gtsam，不是${GTSAM_LIBS}

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

```cpp
#include <gtsam/nonlinear/DoglegOptimizer.h>// dogleg
#include <gtsam/nonlinear/GaussNewtonOptimizer.h> // GN
#include <gtsam/nonlinear/LevenbergMarquardtOptimizer.h>
#include <gtsam/inference/Symbol.h>// symbol_shorthand

#include <random>

// y = ax^2 + bx
double funct(const gtsam::Vector2& param, const double x) {
    return (param(0)*x*x + param(1)*x);
}

// NoiseModelFactor1表示一元边, NoiseModelFactor2表示2元边,一次类推一直到6
class curvfitFactor : public gtsam::NoiseModelFactor1<gtsam::Vector2> {
public:
    curvfitFactor(gtsam::Key key, const gtsam::SharedNoiseModel& noise, double x
        : gtsam::NoiseModelFactor1<gtsam::Vector2>(noise, key), x_(x), y_(y){
    }
    gtsam::Vector evaluateError(const gtsam::Vector2& param, boost::optional<gts
        auto val = funct(param, x_);
        if (H) {
            gtsam::Matrix Jac = gtsam::Matrix::Zero(1, 2);
            Jac << x_ * x_, x_;
            (*H) = Jac;
        }
        return gtsam::Vector1(val - y_);
    }
private:
    double x_, y_;
};

int main() {
    using  gtsam::symbol_shorthand::X;// symbol_shorthand仅用于表示状态变量(char +
    const double sig = 0.05;
    std::random_device rd;// 非确定性种子
```

```
    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,方
    gtsam::NonlinearFactorGraph graph;
    const gtsam::Vector2 para(2, 3);// 曲线参数a b

    for (int i = 0; i < 100; ++i) {
        // 构造观测值mx my
        double mx = i;
        auto my = funct(para, mx) + noise(generator);

        // Isotropic: 各项同性噪声模型,相当于在协方差对角线作放缩
        // 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
        auto noiseM = gtsam::noiseModel::Isotropic::Sigma(1, sig);
        // 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(0)
        graph.emplace_shared<curvfitFactor>(X(0), noiseM, mx, my);
    }

    // 设置初始值
    gtsam::Values intial;
    intial.insert<gtsam::Vector2>(X(0), gtsam::Vector2(1.5, 2.2));

    // 选择优化方法
    // gtsam::GaussNewtonOptimizer opt(graph, intial);
    gtsam::LevenbergMarquardtOptimizer opt(graph, intial);
    // gtsam::DoglegOptimizer opt(graph, intial);
    std::cout << "\ninitial error=" << graph.error(intial) << std::endl;
    auto res = opt.optimize();
    std::cout << "\nfinal error=" << graph.error(res) << std::endl;
    gtsam::Vector2 matX0 = res.at<gtsam::Vector2>(X(0));
    std::cout << "a = " << matX0[0] << ", b = " << matX0[1] << "\n";

    return 0;
}
```

4. 对比

A Comparison of Graph Optimization.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