1.设置参数，生成Allan方差标定曲线

imu_utils是港科沈老师组标定IMU的工具，代码编译运行更加方便。而kalibr_allan是rpng黄国权老师组用来标定IMU的工具，需要在ubuntu中安装matlab，操作更为繁琐。在这里，以imu_utils标定imu内参给大家举例。

首先，我们需要在电脑上安装ROS，然后创建一个工作空间catkin_ws，在catkin_ws目录下创建src目录，即

```
mkdir catkin_ws
cd catkin_ws && mkdir src && cd src
```

然后我们将code_utils功能包和imu_utils功能包拷贝到src目录下：

```
git clone https://github.com/gaowenliang/code_utils.git
git clone https://github.com/gaowenliang/imu_utils.git
```

首先编译code_utils功能包，在catkin_ws目录下进行编译：

```
cd .. && catkin_make -DCATKIN_WHITELIST_PACKAGES="code_utils"
```

在编译code_utils时可能会碰到找不到backward.hpp这个头文件的报错，解决方案有以下几种：

```
方案一：
src/code_utils/CMakeLists.txt中，添加路径：include_directories("include/code_utils")
方案二：
src/code_utils/src/sumpixel_test.cpp中的#include "backward.hpp"改为
    #include "code_utils/backward.hpp"
方案三：（不推荐）
src/code_utils/include/backward.hpp文件扔到src/code_utils/src中
```

然后再编译imu_utils功能包:

```
catkin_make -DCATKIN_WHITELIST_PACKAGES="imu_utils"
```

至此，imu_utils编译完成，可以利用课程提供的ROS功能包生成特定的imu数据集（rosbag），然后启动imu_utils工具，播放rosbag，标定imu内参

使用imu_utils标定，分别设置3组imu参数：

第一组：

```
设定值为---
g_w: 5.0e-5
a_w: 5.0e-4
g_n: 0.015
a_n: 0.019
标定值为---
g_n: 2.1399140014220999e-01/sqrt(200) = 0.01513147701
a_n: 2.6874358954838429e-01/sqrt(200) = 0.01900304145
```

第二组：

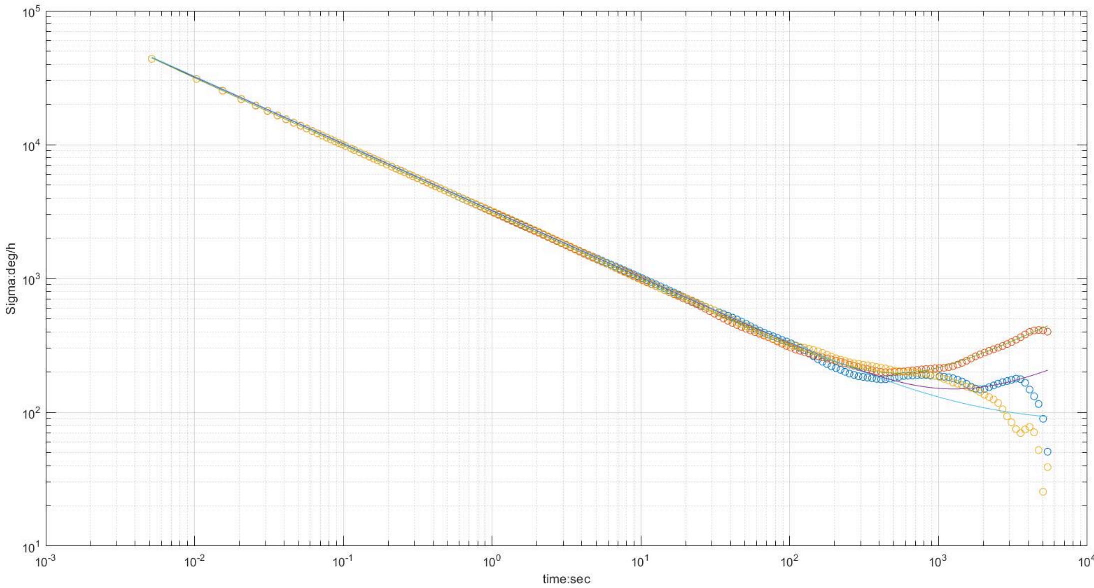
```
设定值为---
g_w: 3.0e-4
a_w: 2.0e-3
g_n: 0.056
a_n: 0.078
标定值为---
g_n: 7.9063566508284433e-01/sqrt(200) = 0.05590638402
a_n: 1.1078258512014039e+00/sqrt(200) = 0.07833511717
```

第三组：

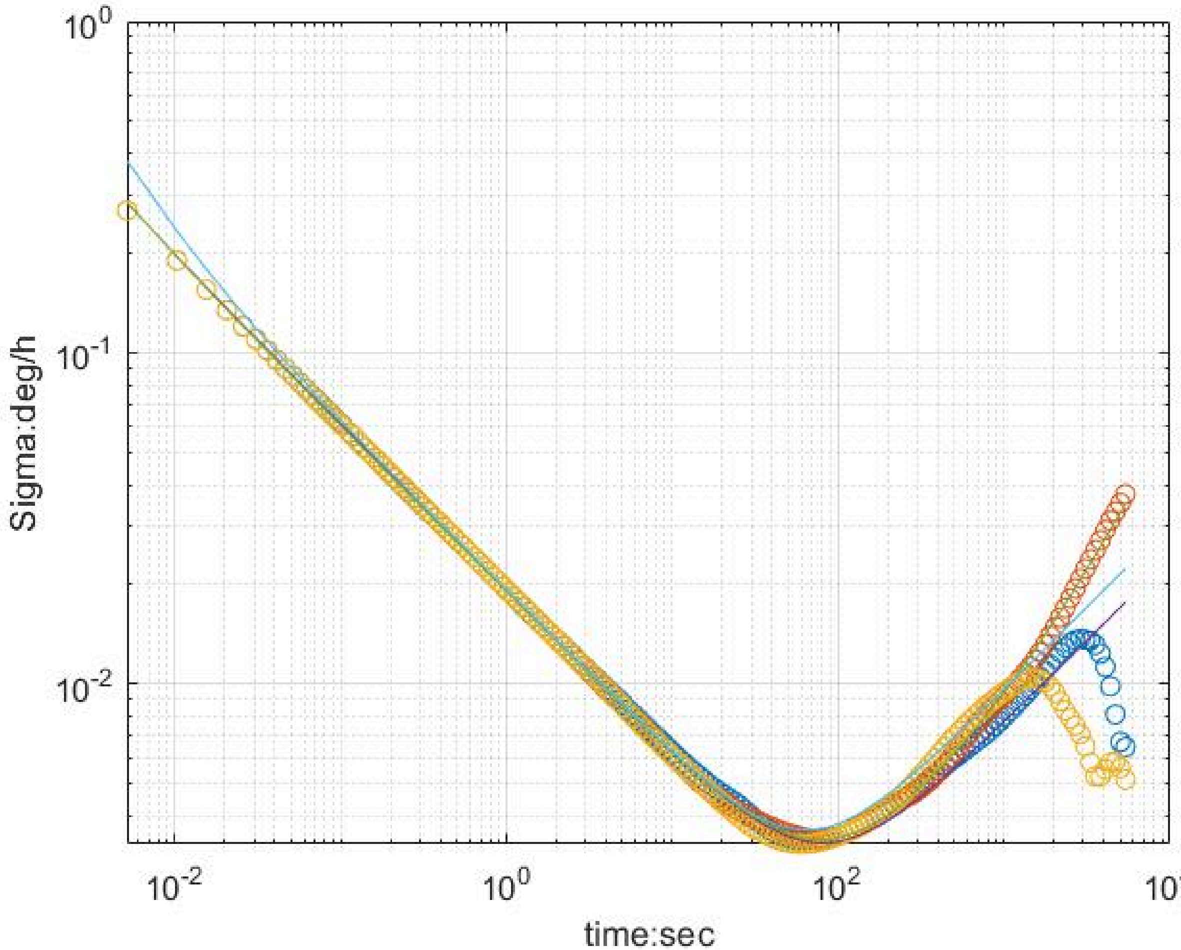
```
设定值为---
g_w: 1.0e-3
a_w: 1.0e-2
g_n: 1.0e-2
a_n: 1.0e-1
标定值为---
g_n: 1.4110400375839960e-01/sqrt(200) = 0.00997755979
a_n: 1.4026838880597403e+00/sqrt(200) = 0.09918472891
```

绘制第三组数据的Allan方差曲线为：

gyroscope:



accelerometer:



2.IMU仿真代码中欧拉积分替换成中值积分

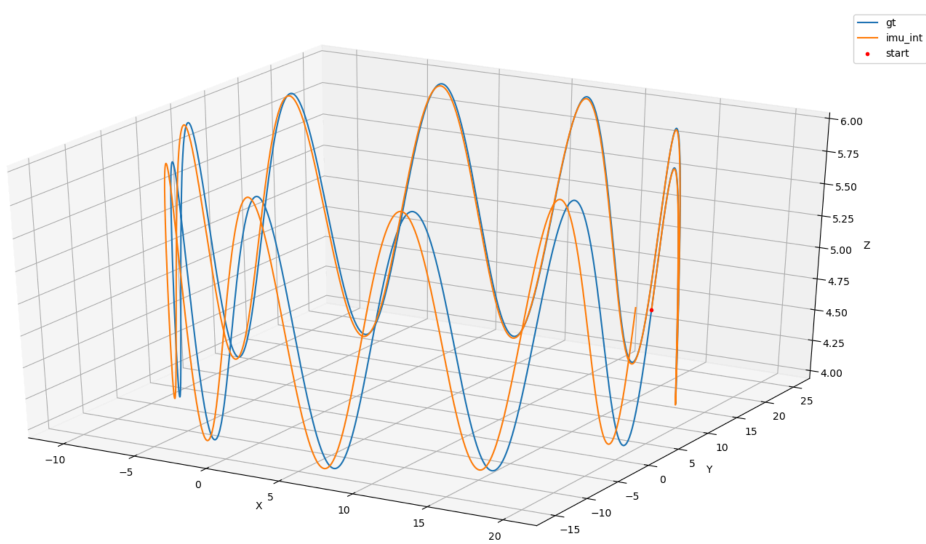
欧拉积分的代码片段为：

```
// 欧拉积分
MotionData imu_data1 = imudata[i];

// delta_q = [1 , 1/2 * thetax , 1/2 * theta_y, 1/2 * theta_z]
Eigen::Quaterniond dq;
Eigen::Vector3d dtheta_half = imu_data1.imu_gyro * dt / 2.0;
dq.w() = 1;
dq.x() = dtheta_half.x();
dq.y() = dtheta_half.y();
dq.z() = dtheta_half.z();
dq.normalize();

// aw = Rwb * ( acc_body - acc_bias ) + gw
Eigen::Vector3d acc_w = Qwb * (imu_data1.imu_acc) + gw;
Qwb = Qwb * dq;
Pwb = Pwb + Vw * dt + 0.5 * dt * dt * acc_w;
Vw = Vw + acc_w * dt;
```

数据仿真效果为：



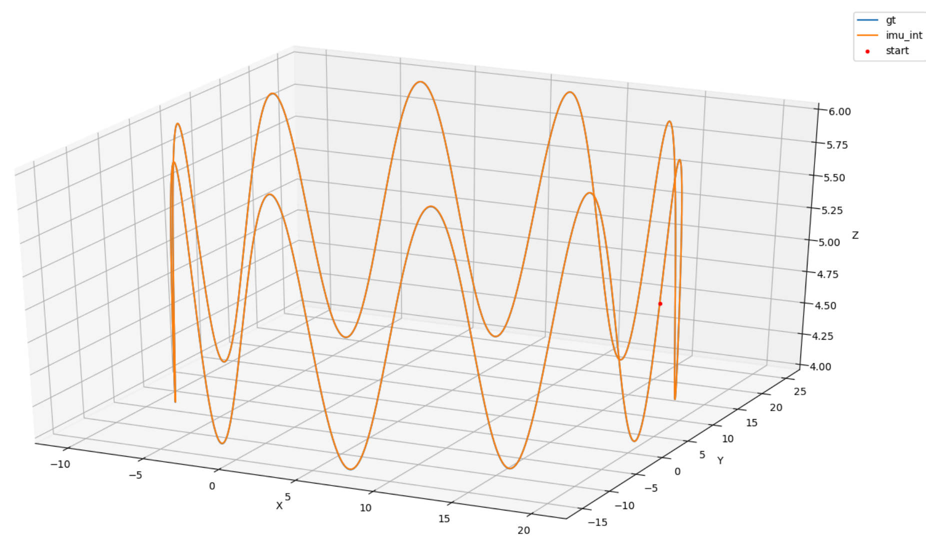
改为中值积分：

```
// 中值积分
MotionData imu_data0 = imudata[i - 1]; // 上一时刻的imu数据
MotionData imu_data1 = imudata[i];     // 当前时刻的imu数据

Eigen::Quaterniond dq;
Eigen::Vector3d dtheta_half = (imu_data0.imu_gyro + imu_data0.imu_gyro)*dt/4.0;
dq.w() = 1;
dq.x() = dtheta_half.x();
dq.y() = dtheta_half.y();
dq.z() = dtheta_half.z();
dq.normalize();

Eigen::Vector3d acc_w = (Qwb * dq * (imu_data1.imu_acc) + gw +
    Qwb * (imu_data0.imu_acc) + gw) / 2;
Qwb = Qwb * dq;
Pwb = Pwb + Vw * dt + 0.5 * dt * dt * acc_w;
Vw = Vw + dt * acc_w;
```

数据仿真效果为：



可以明显看到，中值积分精度更高，数据仿真效果更好。

提升作业

Spline Fusion: A continuous-time representation for visual-inertial fusion with application to rolling shutter cameras

Cumulative cubic B-Splines 曲线

累积基函数

具有 k 个控制点的 $k-1$ 阶 B-spline 曲线的标准基函数:

$$p(t) = \sum_{i=0}^{n} p_i B_{i,k}(t) \qquad (1)$$

$p_i \in \mathbb{R}^n$ 为时间 t_i, $i=0,1,\cdots,n$ 处的控制点, $B_{i,k}(t)$ 称为 B-Spline 的基函数, 是一个标量值, 表示不同控制点权重, 其满足 De Boor-Cox 回归形式:

$$B_{i,0}(t) := \begin{cases} 1, & t_i \leq t \leq t_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

$$B_{i,k}(t) := \frac{t-t_i}{t_{i+k}-t_i} B_{i,k-1}(t) + \frac{t_{i+(k+1)}-t}{t_{i+(k+1)}-t_{i+1}} B_{i+1,k-1}(t)$$

式(1) 写成累积形式:

$$p(t) = p_0 \tilde{B}_{0,k}(t) + \sum_{i=1}^{n} (p_i - p_{i-1}) \tilde{B}_{i,k}(t) \qquad (2)$$

其中 $\tilde{B}_{i,k}(t) = \sum_{j=i}^{n} B_{j,k}(t)$ 为累积基函数

均匀分布下累积基函数矩阵表示

假设 B-Spline 中控制点是服从均匀分布的, 每个控制点时间间隔为 Δt, 定义时间分数为

$$u(t) = \frac{t-t_i}{t_{i+1}-t_i}, \quad u(t) \in [0,1)$$

(2) 的累积 B-Spline 可写成矩阵形式：

$$p(t) = \widetilde{B}(u(t)) \begin{bmatrix} p_0^T \\ \vdots \\ p_i^T \\ \vdots \\ p_n^T \end{bmatrix} = \begin{bmatrix} 1 & \cdots & u(t)^i & \cdots & u(t)^{k-1} \end{bmatrix} \widetilde{M}_k \begin{bmatrix} p_0^T \\ \vdots \\ p_i^T \\ \vdots \\ p_n^T \end{bmatrix}$$

$$\widetilde{M}_k = \begin{bmatrix} \sum_{s=0}^{n-1} m_{0,s} & \sum_{s=1}^{n-1} m_{0,s} & \cdots & \sum_{s=n-1}^{n-1} m_{0,s} \\ \sum_{s=0}^{n-1} m_{1,s} & \sum_{s=1}^{n-1} m_{1,s} & \cdots & \sum_{s=n-1}^{n-1} m_{1,s} \\ \vdots & & & \\ \sum_{s=0}^{n-1} m_{k-1,s} & \sum_{s=1}^{n-1} m_{k-1,s} & \cdots & \sum_{s=n-1}^{n-1} m_{k-1,s} \end{bmatrix}_{k \times n}$$

$$m_{i,j} = \frac{1}{(k-1)!} C_{k-1}^{k-1-i} \sum_{s=j}^{k-1} \left[(-1)^{s-j} C_k^{s-j} (k-s-1)^{k-1-i} \right]$$

$$C_n^i = \frac{n!}{i!(n-i)!}$$

SE3 下的均匀累积 B-Spline

累积 Cubic B-Spline

对式(2)求反对称并运用矩阵的指数与对数映射有：

$$\begin{aligned} \exp(p(t)^\wedge) &= \exp\left(\left(\widetilde{B}_{0,k}(t) p_0 + \sum_{i=1}^{n} \widetilde{B}_{i,k}(t)(p_i - p_{i-1})\right)^\wedge\right) \\ &= \exp\left(\widetilde{B}_{0,k}(t) p_0^\wedge + \sum_{i=1}^{n} \widetilde{B}_{i,k}(t)(p_i - p_{i-1})^\wedge\right) \\ &= \exp\left(\widetilde{B}_{0,k}(t) p_0^\wedge\right) \exp\left(\sum_{i=1}^{n} \widetilde{B}_{i,k}(t)(p_i - p_{i-1})^\wedge\right) \end{aligned}$$

$$T_{ws}(t) = \exp\left(\widetilde{B}_{0,k}(t) \log(T_{w0})\right) \prod_{i=1}^{n} \exp\left(\widetilde{B}_{i,k}(t) \log(T_{wi-1}^{-1} T_{wi})\right) \quad (3)$$

式(3)为 SE3 下描述轨迹位姿的累积 B-spline 曲线方程，

其中 $T_{ws}(t)$ 为 t 时刻 B-Spline 曲线上的位姿，T_{wi} 第 i 个

控制点的位姿

对于 Cubic B-spline 来说，其阶数为3，因此对应控制点数目 $k=4$。对于局部 B-spline 来说，我们定义这四个控制点 $[t_{i-1}, t_i, t_{i+1}, t_{i+2}]$ 且 $t \in [t_i, t_{i+1})$

式(3)表示全局形式的累积 B-spline 方程，对于局部操作来说，可认为 $T_{w,i-1}$ 已知且仅有四个控制点，则式(3)可写为如下展开形式：

$$
\begin{aligned}
T_{w,s}(t) = {} & \exp\left(\tilde{B}_{0,4}(t)\log(T_{w,i-1})\right)\exp\left(\tilde{B}_{1,4}(t)\log(T_{w,i-1}^{-1}T_{w,i})\right) \\
& \exp\left(\tilde{B}_{2,4}(t)\log(T_{w,i}^{-1}T_{w,i+1})\right)\exp\left(\tilde{B}_{3,4}(t)\log(T_{w,i+1}^{-1}T_{w,i+2})\right) \qquad (4)
\end{aligned}
$$

式(4)中系数 $\tilde{B}_{i,4}(t)$ 的求解如果直接利用 De Boor-Cox 求比较繁琐，也不利于编程实现，但所有系数 $\tilde{B}_{i,4}(t)$ 组成的系数矢量 $\tilde{B}(u(t))$ 可表为：

$$
\begin{aligned}
\tilde{B}(u(t)) &= [1 \quad u(t) \quad u^2(t) \quad u^3(t)]\,\tilde{M}^4 \\
&= [1 \quad u(t) \quad u^2(t) \quad u^3(t)]\begin{bmatrix}
\sum_{s=0}^{3} m_{0,s} & \sum_{s=1}^{3} m_{0,s} & \sum_{s=2}^{3} m_{0,s} & \sum_{s=3}^{3} m_{0,s} \\
\sum_{s=0}^{3} m_{1,s} & \sum_{s=1}^{3} m_{1,s} & \sum_{s=2}^{3} m_{1,s} & \sum_{s=3}^{3} m_{1,s} \\
\sum_{s=0}^{3} m_{2,s} & \sum_{s=1}^{3} m_{2,s} & \sum_{s=2}^{3} m_{2,s} & \sum_{s=3}^{3} m_{2,s} \\
\sum_{s=0}^{3} m_{3,s} & \sum_{s=1}^{3} m_{3,s} & \sum_{s=2}^{3} m_{3,s} & \sum_{s=3}^{3} m_{3,s}
\end{bmatrix} \\
&= [1 \quad u(t) \quad u^2(t) \quad u^3(t)]\begin{bmatrix}
6 & 5 & 1 & 0 \\
0 & 3 & 3 & 0 \\
0 & -3 & 3 & 0 \\
0 & 1 & -2 & 1
\end{bmatrix} \\
&= \frac{1}{3!}[6 \quad (5+3u(t)-3u^2(t)+u^3(t)) \quad (1+3u(t)+3u^2(t)-2u^3(t)) \quad u^3(t)]
\end{aligned}
$$

状态的时间微分

IMU 量测模型

$$\omega_m = (R_{w,s}^T \dot{R}_{w,s})^\vee + b_m + n_\omega$$

$$a_m = R_{w,s}^T (\ddot{t}_{w,s} + g_w) + b_a + n_a$$