# LZM_HIT第三章作业



## 作业1：仿真imu数据，并用Allan方差进行分析

### 1.1 数据仿真生成

首先熟悉一下`gnss-ins-sim`程序，通过`gnss-ins-sim`软件生成2个小时的静止转台的IMU数据，设置如下，这里设置停留时间为`2h`：

| A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|
| ini lat (de | ini lon (d | ini alt (m | ini vx_bo | ini vy_bo | ini vz_bo | ini yaw (( | ini pitch ( | ini roll (deg) | |
| 32 | 120 | 0 | 0 | 0 | 0 | 0 | 0 | 180 | |
| comman | yaw (deg | pitch (de | roll (deg) | vx_body | vy_body | vz_body | comman | GPS visibility | |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 7200 | 0 | |

### 1.2 标定软件代码更改与运行

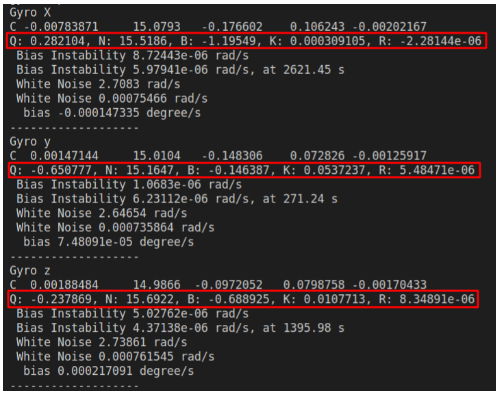
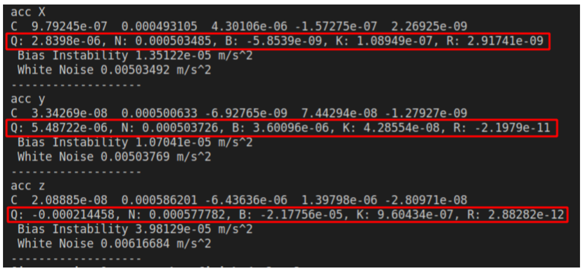
由于`imu_ultis`软件的输入是`bag`包，这里为了方便测试，改写代码，从生成的文件中读取即可，新建`imu_an_sim.cpp`用于标定仿真的数据

**运行方法**

```
## 运行
python demo_no_algo_allan.py

## 进入ros_ws文件夹编译
catkin_make

## 将生成的文件 accel-0.csv gps-0.csv time.csv 拷贝至imu_utils/sim_data中，source当前工作空间
运行：roslaunch imu_utils sim.launch

## 进入imu_utils/scripts文件夹
运行：draw_allan_sim.m
```

## 1.3 结果

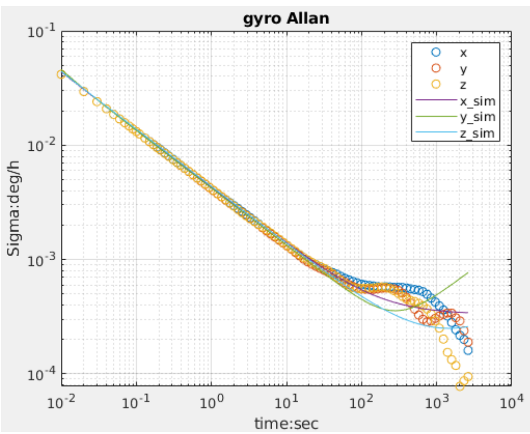
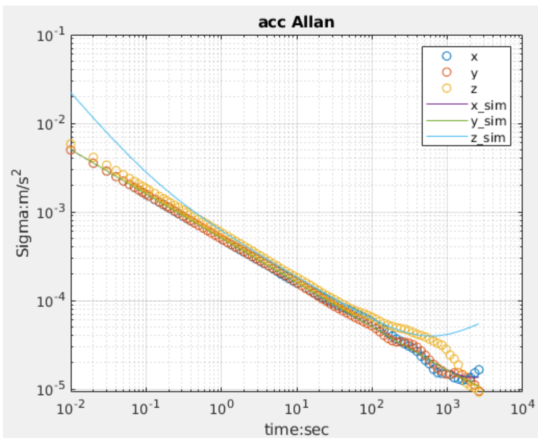
`imu_utils`代码运行结果：

| gyro | acc |
| --- | --- |
|  |  |





`draw_allan_sim.m`运行结果：

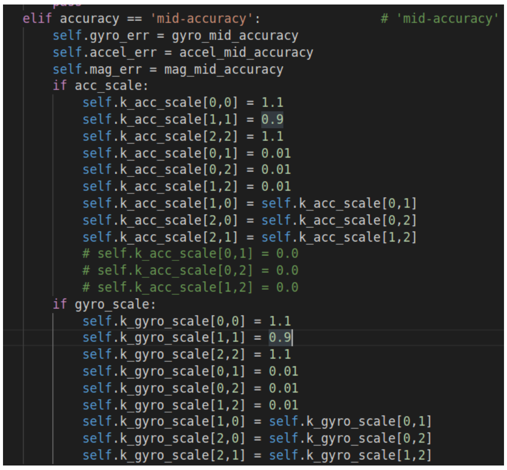
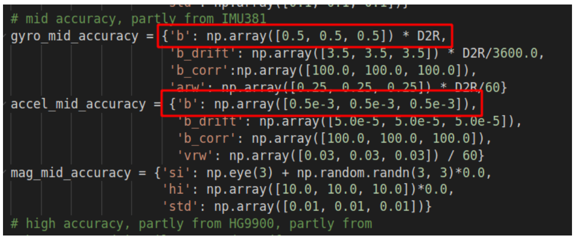
| gyro | acc |
| --- | --- |
|  |  |





# 作业2：设计一种转台旋转方案，并基于仿真数据进行内参求解的验证

## 2.1 设计尺度偏差和bias偏差

| scale | bias |
| --- | --- |
|  |  |

```python
elif accuracy == 'mid-accuracy':                    # 'mid-accuracy'
    self.gyro_err = gyro_mid_accuracy
    self.accel_err = accel_mid_accuracy
    self.mag_err = mag_mid_accuracy
    if acc_scale:
        self.k_acc_scale[0,0] = 1.1
        self.k_acc_scale[1,1] = 0.9
        self.k_acc_scale[2,2] = 1.1
        self.k_acc_scale[0,1] = 0.01
        self.k_acc_scale[0,2] = 0.01
        self.k_acc_scale[1,2] = 0.01
        self.k_acc_scale[1,0] = self.k_acc_scale[0,1]
        self.k_acc_scale[2,0] = self.k_acc_scale[0,2]
        self.k_acc_scale[2,1] = self.k_acc_scale[1,2]
        # self.k_acc_scale[0,1] = 0.0
        # self.k_acc_scale[0,2] = 0.0
        # self.k_acc_scale[1,2] = 0.0
    if gyro_scale:
        self.k_gyro_scale[0,0] = 1.1
        self.k_gyro_scale[1,1] = 0.9
        self.k_gyro_scale[2,2] = 1.1
        self.k_gyro_scale[0,1] = 0.01
        self.k_gyro_scale[0,2] = 0.01
        self.k_gyro_scale[1,2] = 0.01
        self.k_gyro_scale[1,0] = self.k_gyro_scale[0,1]
        self.k_gyro_scale[2,0] = self.k_gyro_scale[0,2]
        self.k_gyro_scale[2,1] = self.k_gyro_scale[1,2]
```

```python
# mid accuracy, partly from IMU381
gyro_mid_accuracy = {'b': np.array([0.5, 0.5, 0.5]) * D2R,
                     'b_drift': np.array([3.5, 3.5, 3.5]) * D2R/3600.0,
                     'b_corr':np.array([100.0, 100.0, 100.0]),
                     'arw': np.array([0.25, 0.25, 0.25]) * D2R/60}
accel_mid_accuracy = {'b': np.array([0.5e-3, 0.5e-3, 0.5e-3]),
                      'b_drift': np.array([5.0e-5, 5.0e-5, 5.0e-5]),
                      'b_corr': np.array([100.0, 100.0, 100.0]),
                      'vrw': np.array([0.03, 0.03, 0.03]) / 60}
mag_mid_accuracy = {'si': np.eye(3) + np.random.randn(3, 3)*0.0,
                    'hi': np.array([10.0, 10.0, 10.0])*0.0,
                    'std': np.array([0.01, 0.01, 0.01])}
# high accuracy, partly from HG9900, partly from
```

根据课程里的方法，选择6个角度，使得各个轴的正向分别朝上和朝下，采集`IMU`的数据，数据结果如下(数据位于`depart_calib/acc`下)：

zyx_0_0_0　zyx_0_0_90　zyx_0_0_180　zyx_0_0_-90　zyx_0_90_0　zyx_0_-90_0

这里，`zyx`表示旋转顺序，后面的三个表示`yaw,pitch,roll`角度。

陀螺以的角度为(尺度角度位于`depart_calib/gyro/scale`下，bias角度位于`depart_calib/gyro/bias`下)：

**尺度角度**：

zyx_0_0_90　zyx_0_0_-90　zyx_0_90_0　zyx_0_-90_0　zyx_90_0_0　zyx_-90_0_0

**bias角度**：

zyx_0_0_0　zyx_0_0_180　zyx_0_180_0　zyx_180_0_0

## 2.2 代码编写

根据课程的内容，这里使用`Matlab`来进行测试

```
进入depart_calib文件夹，matlab运行least_square.m
```

## 2.3 结果展示

| 只有bias误差项 | 包含bias误差以及噪声项 |
|---|---|
| <pre>>> least_square<br>calibrate accel,please wait ...<br>##### imu accel calibration result is #####<br><br>scal_matrix_acc =<br><br>    1.1000    0.0100    0.0100<br>    0.0100    0.9000    0.0100<br>    0.0100    0.0100    1.1000<br><br><br>bias_acc =<br><br>   1.0e-03 *<br><br>    0.5000    0.5000    0.5000<br><br>calibrate gyro,please wait ...<br>##### imu gyro calibration result is #####<br><br>scal_matrix_gyro =<br><br>    1.1000    0.0100    0.0100<br>    0.0100    0.9000    0.0100<br>    0.0100    0.0100    1.1000<br><br><br>bias_gyro =<br><br>    0.5000<br>    0.5000<br>    0.5000</pre> | <pre>>> least_square<br>calibrate accel,please wait ...<br>##### imu accel calibration result is #####<br><br>scal_matrix_acc =<br><br>    1.1000    0.0100    0.0100<br>    0.0100    0.9000    0.0100<br>    0.0100    0.0100    1.1000<br><br><br>bias_acc =<br><br>   1.0e-03 *<br><br>    0.4598    0.4678    0.4975<br><br>calibrate gyro,please wait ...<br>##### imu gyro calibration result is #####<br><br>scal_matrix_gyro =<br><br>    1.1001    0.0102    0.0100<br>    0.0101    0.9003    0.0101<br>    0.0099    0.0095    1.0999<br><br><br>bias_gyro =<br><br>    0.5003<br>    0.4995<br>    0.5002</pre> |

可以看到，如果只有噪声误差项，可以很好的标定出来内参，但是包含了误差项之后，对内参的标定会稍微有一点影响，但是结果和设置的内参也很接近。

**注意**：需要注意的是，因为保存数据的时候，陀螺仪角速度是(deg/s)，代码里没有对这个进行更改，所以标定出来的bias也是角度的单位，即和设置的参数是一样的。

# 作业3：推导基于LM进行加速度计和陀螺仪内参估计的优化过程。按照课件中的模型，修改参考代码，并基于仿真数据实现

## 3.1 LM推导

### 3.1.1 $L(\boldsymbol{\theta}^{acc})$推导

首先，写出$h()$函数的表达式：

$$
\begin{aligned}
h(a_k^s, \boldsymbol{\theta}^{acc}) &= \begin{bmatrix} K_{ax} & 0 & 0 \\ S_{ayx} & K_{ay} & 0 \\ S_{azx} & S_{azy} & K_{az} \end{bmatrix} \begin{bmatrix} a_{kx}^s \\ a_{ky}^s \\ a_{kz}^s \end{bmatrix} + \begin{bmatrix} \Delta_x \\ \Delta_y \\ \Delta_z \end{bmatrix} \\
&= \begin{bmatrix} K_{ax} a_{kx}^s + \Delta_x \\ S_{ayx} a_{kx}^s + K_{ay} a_{ky}^s + \Delta_y \\ S_{azx} a_{kx}^s + S_{azy} a_{ky}^s + K_{az} a_{kz}^s + \Delta_z \end{bmatrix}
\end{aligned}
$$

目标函数为

$$
\begin{aligned}
L(\boldsymbol{\theta}^{acc}) &= \Sigma_{k=1}^{M} (||\boldsymbol{g}||^2 - ||h(a_k^s, \boldsymbol{\theta}^{acc})||^2)^2 \\
&= \Sigma_{k=1}^{M} (||\boldsymbol{g}||^2 - ||h(a_k^s, \boldsymbol{\theta}^{acc})||^2)^T (||\boldsymbol{g}||^2 - ||h(a_k^s, \boldsymbol{\theta}^{acc})||^2)
\end{aligned}
$$

令$e_k(\boldsymbol{\theta}^{acc})) = ||\boldsymbol{g}||^2 - ||h(a_k^s, \boldsymbol{\theta}^{acc})||^2$，根据优化的理论基础，需要求取$e_k(\boldsymbol{\theta}^{acc}))$对状态量$\boldsymbol{\theta}^{acc})$的雅克比矩阵$\frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial \boldsymbol{\theta}^{acc}}$，进而构建优化问题，因为$||\boldsymbol{g}||^2$为常量，所以可以忽略不计算，即:

$$
\begin{aligned}
\frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial \boldsymbol{\theta}^{acc}} &= -\frac{\partial ||h(a_k^s, \boldsymbol{\theta}^{acc})||^2}{\partial \boldsymbol{\theta}^{acc}} \\
&= -\frac{\partial ||h(a_k^s, \boldsymbol{\theta}^{acc})||^2}{\partial h(a_k^s, \boldsymbol{\theta}^{acc})} * \frac{\partial h(a_k^s, \boldsymbol{\theta}^{acc})}{\partial \boldsymbol{\theta}^{acc}} \\
&= -2 * h(a_k^s, \boldsymbol{\theta}^{acc})^T * \frac{\partial h(a_k^s, \boldsymbol{\theta}^{acc})}{\partial \boldsymbol{\theta}^{acc}} \\
&= -2 * \begin{bmatrix} K_{ax} a_{kx}^s + \Delta_x \\ S_{ayx} a_{kx}^s + K_{ay} a_{ky}^s + \Delta_y \\ S_{azx} a_{kx}^s + S_{azy} a_{ky}^s + K_{az} a_{kz}^s + \Delta_z \end{bmatrix} \frac{\partial h(a_k^s, \boldsymbol{\theta}^{acc})}{\partial \boldsymbol{\theta}^{acc}}
\end{aligned}
$$

所以只需要对$\boldsymbol{\theta}^{acc}$中的每个参数变量求取偏导就可以得到最终的结果:

(1) $\frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial S_{ayx}}$

$$
\begin{aligned}
\frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial S_{ayx}} &= -2 * \begin{bmatrix} K_{ax} a_{kx}^s + \Delta_x \\ S_{ayx} a_{kx}^s + K_{ay} a_{ky}^s + \Delta_y \\ S_{azx} a_{kx}^s + S_{azy} a_{ky}^s + K_{az} a_{kz}^s + \Delta_z \end{bmatrix}^T \begin{bmatrix} 0 \\ a_{kx}^s \\ 0 \end{bmatrix} \\
&= -2 * (S_{ayx} a_{kx}^s + K_{ay} a_{ky}^s + \Delta_y) * a_{kx}^s
\end{aligned}
$$

(2) $\frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial S_{azx}}$

$$
\begin{aligned}
\frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial S_{azx}} &= -2 * \begin{bmatrix} K_{ax} a_{kx}^s + \Delta_x \\ S_{ayx} a_{kx}^s + K_{ay} a_{ky}^s + \Delta_y \\ S_{azx} a_{kx}^s + S_{azy} a_{ky}^s + K_{az} a_{kz}^s + \Delta_z \end{bmatrix}^T \begin{bmatrix} 0 \\ 0 \\ a_{kx}^s \end{bmatrix} \\
&= -2 * (S_{azx} a_{kx}^s + S_{azy} a_{ky}^s + K_{az} a_{kz}^s + \Delta_z) * a_{kx}^s
\end{aligned}
$$

(3) $\frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial S_{azy}}$

$$
\begin{aligned}
\frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial S_{azy}} &= -2 * \begin{bmatrix} K_{ax} a_{kx}^s + \Delta_x \\ S_{ayx} a_{kx}^s + K_{ay} a_{ky}^s + \Delta_y \\ S_{azx} a_{kx}^s + S_{azy} a_{ky}^s + K_{az} a_{kz}^s + \Delta_z \end{bmatrix}^T \begin{bmatrix} 0 \\ 0 \\ a_{ky}^s \end{bmatrix} \\
&= -2 * (S_{azx} a_{kx}^s + S_{azy} a_{ky}^s + K_{az} a_{kz}^s + \Delta_z) * a_{ky}^s
\end{aligned}
$$

(4) $\frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial K_{ax}}$

$$
\begin{aligned}
\frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial K_{ax}} &= -2 * \begin{bmatrix} K_{ax} a_{kx}^s + \Delta_x \\ S_{ayx} a_{kx}^s + K_{ay} a_{ky}^s + \Delta_y \\ S_{azx} a_{kx}^s + S_{azy} a_{ky}^s + K_{az} a_{kz}^s + \Delta_z \end{bmatrix}^T \begin{bmatrix} a_{kx}^s \\ 0 \\ 0 \end{bmatrix} \\
&= -2 * (K_{ax} a_{kx}^s + \Delta_x) * a_{kx}^s
\end{aligned}
$$

(5) $\frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial K_{ay}}$

$$
\begin{aligned}
\frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial K_{ax}} &= -2 * \begin{bmatrix} K_{ax} a_{kx}^s + \Delta_x \\ S_{ayx} a_{kx}^s + K_{ay} a_{ky}^s + \Delta_y \\ S_{azx} a_{kx}^s + S_{azy} a_{ky}^s + K_{az} a_{kz}^s + \Delta_z \end{bmatrix}^T \begin{bmatrix} 0 \\ a_{ky}^s \\ 0 \end{bmatrix} \\
&= -2 * (S_{ayx} a_{kx}^s + K_{ay} a_{ky}^s + \Delta_y) * a_{ky}^s
\end{aligned}
$$

(6) $\frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial K_{ay}}$

$$\frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial K_{ax}} = -2 * \begin{bmatrix} K_{ax}a_{kx}^s + \Delta_x \\ S_{ayx}a_{kx}^s + K_{ay}a_{ky}^s + \Delta_y \\ S_{azx}a_{kx}^s + S_{azy}a_{ky}^s + K_{az}a_{kz}^s + \Delta_z \end{bmatrix}^T \begin{bmatrix} 0 \\ 0 \\ a_{kz}^s \end{bmatrix}$$
$$= -2 * (S_{azx}a_{kx}^s + S_{azy}a_{ky}^s + K_{az}a_{kz}^s + \Delta_z) * a_{kz}^s$$

(7) $\frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial \Delta_x}$

$$\frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial \Delta_x} = -2 * \begin{bmatrix} K_{ax}a_{kx}^s + \Delta_x \\ S_{ayx}a_{kx}^s + K_{ay}a_{ky}^s + \Delta_y \\ S_{azx}a_{kx}^s + S_{azy}a_{ky}^s + K_{az}a_{kz}^s + \Delta_z \end{bmatrix}^T \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$
$$= -2 * (K_{ax}a_{kx}^s + \Delta_x)$$

(8) $\frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial \Delta_y}$

$$\frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial \Delta_y} = -2 * \begin{bmatrix} K_{ax}a_{kx}^s + \Delta_x \\ S_{ayx}a_{kx}^s + K_{ay}a_{ky}^s + \Delta_y \\ S_{azx}a_{kx}^s + S_{azy}a_{ky}^s + K_{az}a_{kz}^s + \Delta_z \end{bmatrix}^T \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$
$$= -2 * (S_{ayx}a_{kx}^s + K_{ay}a_{ky}^s + \Delta_y)$$

(9) $\frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial \Delta_z}$

$$\frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial \Delta_x} = -2 * \begin{bmatrix} K_{ax}a_{kx}^s + \Delta_x \\ S_{ayx}a_{kx}^s + K_{ay}a_{ky}^s + \Delta_y \\ S_{azx}a_{kx}^s + S_{azy}a_{ky}^s + K_{az}a_{kz}^s + \Delta_z \end{bmatrix}^T \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$
$$= -2 * (S_{azx}a_{kx}^s + S_{azy}a_{ky}^s + K_{az}a_{kz}^s + \Delta_z)$$

综上可得

$$J_k = \frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial \boldsymbol{\theta}^{acc}}$$
$$= \begin{bmatrix} \frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial S_{ayx}} & \frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial S_{azx}} & \frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial S_{azy}} & \frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial K_{ax}} & \frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial K_{ay}} & \frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial K_{az}} & \frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial \Delta_x} & \frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial \Delta_y} & \frac{\partial e_k(\boldsymbol{\theta}^{acc})}{\partial \Delta_z} \end{bmatrix}$$

最终LM方程为：

$$(\Sigma_{k=1}^M(J_k^T * J_k + \lambda)) * \delta\boldsymbol{\theta}^{acc} = \Sigma_{k=1}^M(-J_k^T * e_k)$$

通过不断调整λ因子，即可得到最终的结果

### 3.1.2 $L(\boldsymbol{\theta}^{gyro})$推导

整个推导过程与上面加速度计的类似，只不过角速度需要积分计算出旋转矩阵，进而得到分解的加速度

首先，写出$\boldsymbol{u}_{g,k}$函数的表达式：

$$\boldsymbol{u}_{g,k} = \Psi[\omega_i, \boldsymbol{u}_{a,k-1}]$$

其中$\Psi$表示在$(k-1,k)$区间内的角速度姿态解算方程。

目标函数为

$$L(\boldsymbol{\theta}^{gyro}) = \Sigma_{k=2}^M(\boldsymbol{u}_{a,k} - \boldsymbol{u}_{g,k})^2$$
$$= \Sigma_{k=2}^M(\boldsymbol{u}_{a,k} - \Psi[\omega_i, \boldsymbol{u}_{a,k-1}])^2$$

令$e_k(\boldsymbol{\theta}^{gyro})) = \boldsymbol{u}_{a,k} - \boldsymbol{u}_{g,k}$，根据优化的理论基础，需要求取$e_k(\boldsymbol{\theta}^{gyro}))$对状态量$\boldsymbol{\theta}^{gyro})$的雅克比矩阵$\frac{\partial e_k(\boldsymbol{\theta}^{gyro})}{\partial \boldsymbol{\theta}^{gyro}}$，进而构建优化问题，因为$\boldsymbol{u}_{a,k}$为常量，所以可以忽略不计算，即:

$$
\begin{aligned}
J_k = \frac{\partial e_k(\boldsymbol{\theta}^{gyro})}{\partial \boldsymbol{\theta}^{gyro}} &= -\frac{\partial \Psi[\omega_i, \boldsymbol{u}_{a,k-1}]}{\partial \boldsymbol{\theta}^{gyro}} \\
&= -\frac{\partial \Psi[\omega_i, \boldsymbol{u}_{a,k-1}]}{\partial \omega_i}^T * \frac{\partial \omega_i}{\boldsymbol{\theta}^{gyro}}
\end{aligned}
$$

$-\frac{\partial \Psi[\omega_i, \boldsymbol{u}_{a,k-1}]}{\partial \boldsymbol{\theta}^{gyro}}$取决于$\Psi[\omega_i, \boldsymbol{u}_{a,k-1}]$的形式。

其中：

$$
\begin{aligned}
\omega_i &= \begin{bmatrix} K_{gx} & S_{gxy} & S_{gxz} \\ S_{gyx} & K_{gy} & S_{gyz} \\ S_{gzx} & S_{gzy} & K_{gz} \end{bmatrix} \begin{bmatrix} \omega_x^s \\ \omega_y^s \\ \omega_z^s \end{bmatrix} + \begin{bmatrix} \Delta_x \\ \Delta_y \\ \Delta_z \end{bmatrix} \\
&= \begin{bmatrix} K_{gx}\omega_x^s + S_{gxy}\omega_y^s + S_{gxz}\omega_z^s + \Delta_x \\ S_{gyx}\omega_x^s + K_{gy}\omega_y^s + S_{gyz}\omega_z^s + \Delta_y \\ S_{gzx}\omega_x^s + S_{gzy}\omega_y^s + K_{gz}\omega_z^s + \Delta_z \end{bmatrix}
\end{aligned}
$$

$\omega_i$对12个状态量求偏导即可，具体步骤和上面加速度的一样，最终得到的是一个3x12的雅克比矩阵。

(1) $\frac{\partial \omega_i}{\partial K_{gx}}$

$$
\frac{\partial \omega_i}{\partial K_{gx}} = \begin{bmatrix} \omega_x^s \\ 0 \\ 0 \end{bmatrix}
$$

(2) $\frac{\partial \omega_i}{\partial K_{gy}}$

$$
\frac{\partial \omega_i}{\partial K_{gy}} = \begin{bmatrix} 0 \\ \omega_y^s \\ 0 \end{bmatrix}
$$

(3) $\frac{\partial \omega_i}{\partial K_{gz}}$

$$
\frac{\partial \omega_i}{\partial K_{gz}} = \begin{bmatrix} 0 \\ 0 \\ \omega_z^s \end{bmatrix}
$$

(4) $\frac{\partial \omega_i}{\partial S_{gyx}}$

$$
\frac{\partial \omega_i}{\partial S_{gyx}} = \begin{bmatrix} \omega_x^s \\ 0 \\ 0 \end{bmatrix}
$$

(5) $\frac{\partial \omega_i}{\partial S_{gzx}}$

$$
\frac{\partial \omega_i}{\partial S_{gzx}} = \begin{bmatrix} \omega_x^s \\ 0 \\ 0 \end{bmatrix}
$$

(6) $\frac{\partial \omega_i}{\partial S_{gzy}}$

$$
\frac{\partial \omega_i}{\partial S_{gzx}} = \begin{bmatrix} 0 \\ \omega_y^s \\ 0 \end{bmatrix}
$$

(7) $\frac{\partial\omega_i}{\partial S_{gxy}}$

$$\frac{\partial\omega_i}{\partial S_{gxy}}=\begin{bmatrix}0\\ \omega_y^s\\ 0\end{bmatrix}$$

(8) $\frac{\partial\omega_i}{\partial S_{gxz}}$

$$\frac{\partial\omega_i}{\partial S_{gxz}}=\begin{bmatrix}0\\ 0\\ \omega_z^s\end{bmatrix}$$

(9) $\frac{\partial\omega_i}{\partial S_{gyz}}$

$$\frac{\partial\omega_i}{\partial S_{gyz}}=\begin{bmatrix}0\\ 0\\ \omega_z^s\end{bmatrix}$$

(10) $\frac{\partial\omega_i}{\partial\Delta_x}$

$$\frac{\partial\omega_i}{\partial\Delta_x}=\begin{bmatrix}1\\ 0\\ 0\end{bmatrix}$$

(11) $\frac{\partial\omega_i}{\partial\Delta_y}$

$$\frac{\partial\omega_i}{\partial\Delta_y}=\begin{bmatrix}0\\ 1\\ 0\end{bmatrix}$$

(12) $\frac{\partial\omega_i}{\partial\Delta_z}$

$$\frac{\partial\omega_i}{\partial\Delta_z}=\begin{bmatrix}0\\ 0\\ 1\end{bmatrix}$$

所以：

$$\frac{\partial\omega_i}{\boldsymbol{\theta}^{gyro}}=\begin{bmatrix}\frac{\partial\omega_i}{\partial K_{gx}} & \frac{\partial\omega_i}{\partial K_{gy}} & \frac{\partial\omega_i}{\partial K_{gz}} & \cdots & \frac{\partial\omega_i}{\partial\Delta_x} & \frac{\partial\omega_i}{\partial\Delta_y} & \frac{\partial\omega_i}{\partial\Delta_z}\end{bmatrix}$$

雅克比方程：

$$J_k=-\frac{\partial\Psi[\omega_i,\boldsymbol{u}_{a,k-1}]}{\partial\omega_i}^T*\frac{\partial\omega_i}{\boldsymbol{\theta}^{gyro}}$$

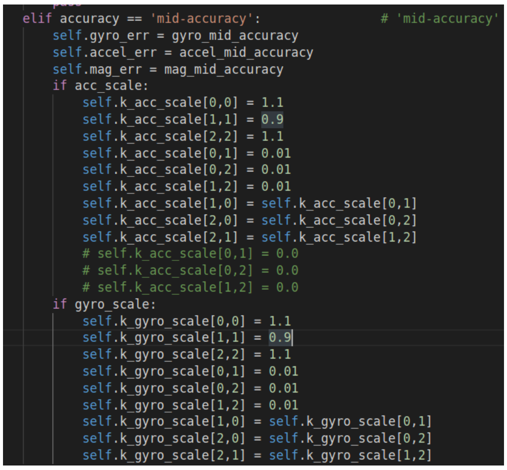
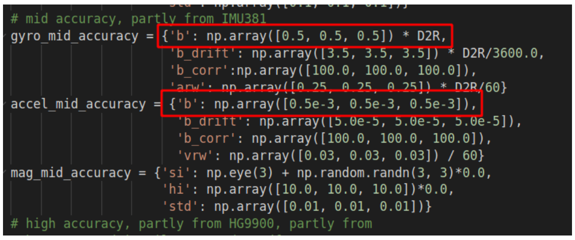
最终LM方程为：

$$(\Sigma_{k=2}^M(J_k^T*J_k+\lambda))*\delta\boldsymbol{\theta}^{gyro}=\Sigma_{k=2}^M(-J_k^T*e_k)$$

通过不断调整λ因子，即可得到最终的结果

## 3.2 修改代码

### 3.2.1 设计尺度偏差和bias偏差

<table>
<tr><th>scale</th><th>bias</th></tr>
<tr><td>
</td>
<td>
</td></tr>
</table>

这里，噪声和之前设置的一样，尺度这里，设置为只有下三角的数据，即解注释上面左图中的三行注释部分：

```
self.k_acc_scale[0,1] = 0.0
self.k_acc_scale[0,2] = 0.0
self.k_acc_scale[1,2] = 0.0
```

### 3.2.2 数据生成文件与编译准备

数据生成文件是`gnss-ins-sim/demo_motion_def_files/motion_def_tk.csv`，修改`demo_no_algo.py`中的文件路径，运行即可，将生成的数据`accel_only_with_bias-0.csv, gyro_only_with_bias-0.csv, time.csv`三个文件拷到`imu_tk/bin/test_data/sim`文件夹下，留给后面测试使用

```
## 编译
cd imu_tk
mkdir build
cd build
cmake ..
make
```

### 3.2.3 数据加载部分代码

同样的，首先要对代码部分进行修改，为了方便，仍然采用读取csv文件的方式，在`imu_tk/apps`的`test_imu_calib.cpp`里面添加关于csv文件读取的功能，为了不改变原来的代码测试，通过输入来判断，如果是仿真的数据，需要输入 时间，加速度，角速度的文件，否则就是之前的代码

```cpp
vector<TriadData> acc_data, gyro_data;

if (!use_sim)
{
  cout << "Importing IMU data from the Matlab matrix file : " << argv[1] << endl;
  importAsciiData(argv[1], acc_data, imu_tk::TIMESTAMP_UNIT_SEC);
  cout << "Importing IMU data from the Matlab matrix file : " << argv[2] << endl;
  importAsciiData(argv[2], gyro_data, imu_tk::TIMESTAMP_UNIT_SEC);
}
else
{
  cout << "Importing Time data from the CSV file : " << argv[1] << endl;
  std::vector<double> time_data;
  loadCsvFile(argv[1], time_data);
  std::cout << "time_data size is: " << time_data.size() << std::endl;
  if (time_data.empty())
    return -1;
  cout << "Importing IMU data from the CSV file : " << argv[1] << endl;
  importCsvData(argv[2], acc_data, time_data, 1.0, imu_tk::TIMESTAMP_UNIT_SEC);
  cout << "Importing IMU data from the CSV file : " << argv[1] << endl;
  importCsvData(argv[3], gyro_data, time_data, DEG2RAD, imu_tk::TIMESTAMP_UNIT_SEC);
}

CalibratedTriad init_acc_calib, init_gyro_calib;
if (!use_sim)
{
  init_acc_calib.setBias(Vector3d(32768, 32768, 32768));
  init_gyro_calib.setScale(Vector3d(1.0 / 6258.0, 1.0 / 6258.0, 1.0 / 6258.0));
}
else
{
  init_acc_calib.setBias(Vector3d(0, 0, 0));
  init_gyro_calib.setScale(Vector3d(0, 0, 0));
}
```

### 3.2.4 模型更新的代码

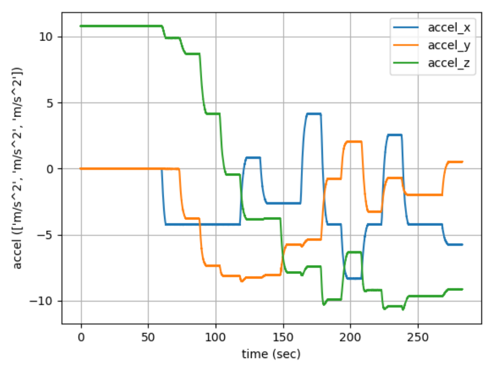
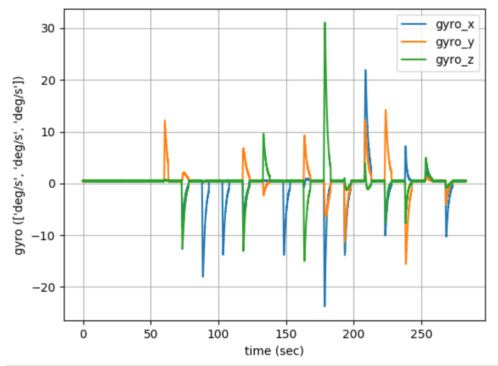
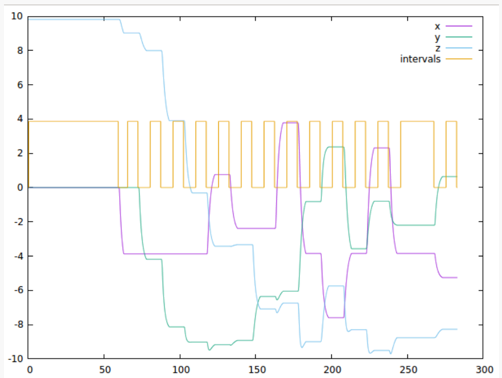
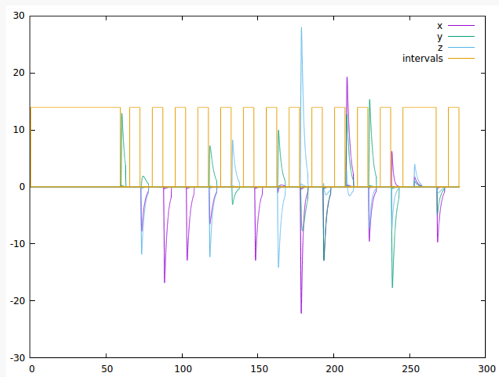
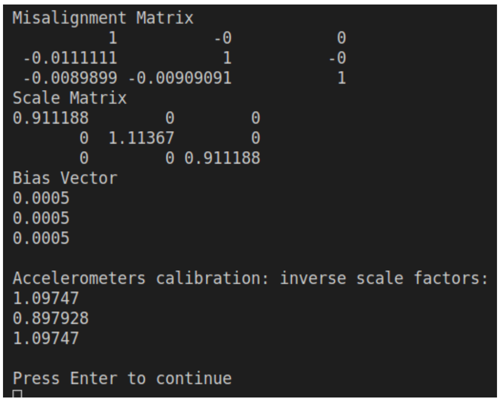
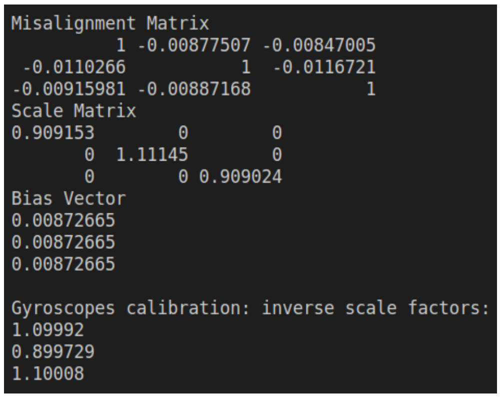
因为替换了模型，现在加速度的尺度矩阵是下三角有元素，所以需要对优化的模型进行更新：

```cpp
template <typename _T1>
struct MultiPosAccResidual
{
  MultiPosAccResidual(const _T1& g_mag, const Eigen::Matrix<_T1, 3, 1>& sample) : g_mag_(g_mag), sample_(sample)
  {
  }

  template <typename _T2>
  bool operator()(const _T2* const params, _T2* residuals) const
  {
    Eigen::Matrix<_T2, 3, 1> raw_samp(_T2(sample_(0)), _T2(sample_(1)), _T2(sample_(2)));
    /* Assume body frame same as accelerometer frame,
     * so bottom left params in the misalignment matris are set to zero */
    // CalibratedTriad_<_T2> calib_triad(params[0], params[1], params[2], _T2(0), _T2(0), _T2(0), params[3], params[4],
    //                                   params[5], params[6], params[7], params[8]);
    CalibratedTriad_<_T2> calib_triad(_T2(0), _T2(0), _T2(0), params[0], params[1], params[2], params[3], params[4],
                                      params[5], params[6], params[7], params[8]);

    Eigen::Matrix<_T2, 3, 1> calib_samp = calib_triad.unbiasNormalize(raw_samp);
    residuals[0] = _T2(g_mag_) - calib_samp.norm();
    return true;
  }

  static ceres::CostFunction* Create(const _T1& g_mag, const Eigen::Matrix<_T1, 3, 1>& sample)
  {
    return (new ceres::AutoDiffCostFunction<MultiPosAccResidual, 1, 9>(new MultiPosAccResidual<_T1>(g_mag, sample)));
  }

  const _T1 g_mag_;
  const Eigen::Matrix<_T1, 3, 1> sample_;
};
```

## 3.3 运行与结果展示

```
## 运行
cd imu_tk/bin
./test_imu_calib test_data/sim/time.csv test_data/sim/accel_only_with_bias-0.csv
test_data/sim/gyro_only_with_bias-0.csv
```

**结果如下：**

<table>
<tr><th></th><th>accel</th><th>gyro</th></tr>
<tr><td>标定前</td><td></td><td></td></tr>
<tr><td>标定后</td><td></td><td></td></tr>
<tr><td>标定结果</td><td>Misalignment Matrix<br>1 -0 0<br>-0.0111111 1 -0<br>-0.0089899 -0.00909091 1<br>Scale Matrix<br>0.911188 0 0<br>0 1.11367 0<br>0 0 0.911188<br>Bias Vector<br>0.0005<br>0.0005<br>0.0005<br><br>Accelerometers calibration: inverse scale factors:<br>1.09747<br>0.897928<br>1.09747<br><br>Press Enter to continue</td><td>Misalignment Matrix<br>1 -0.00877507 -0.00847005<br>-0.0110266 1 -0.0116721<br>-0.00915981 -0.00887168 1<br>Scale Matrix<br>0.909153 0 0<br>0 1.11145 0<br>0 0 0.909024<br>Bias Vector<br>0.00872665<br>0.00872665<br>0.00872665<br><br>Gyroscopes calibration: inverse scale factors:<br>1.09992<br>0.899729<br>1.10008</td></tr>
</table>

可以看到，区间划分很正确，且标定后尺度得到恢复，而且标定出来的bias和尺度因子跟设计的参数几乎完全一样，其中需要注意的是，这里在代码中读取csv的时候将陀螺仪的数据进行了单位转换，所以标定出来的bias是0.00872665(rad) = 0.5(deg)，所以结果是正确的。

# 作业4：对一组数据进行惯性导航解算验证

## 4.1 数据生成

这里直接使用`gnss-ins-sim`里面的两个例子进行验证，首先是`motion_def-90deg_turn.csv`，然后是`motion_def-3d.csv`，`ref_frame`设置为1；

```
1. 修改demo_free_integration.py文件中的运动模型文件
2. 运行demo_free_integration.py
3. 将生成的文件夹拷贝至Ins_Navigation/data/路径下
```

## 4.2 程序编译运行

```
## 编译
cd Ins_Navigation
mkdir build
cd build
cmake ..
make -j4

## 修改config/config.yaml参数
data_folder: 2020-11-07-20-45-00 #2020-11-07-19-01-44
solution_mode: 0  # 0 is offline, 1 is online，目前暂时不支持在线
use_fixed_model: false  # 0: use fixe angle rotation model,1: use rotation vector model

## 运行
打开终端，先运行roscore
cd build
./ins_navigation
```

参数说明：

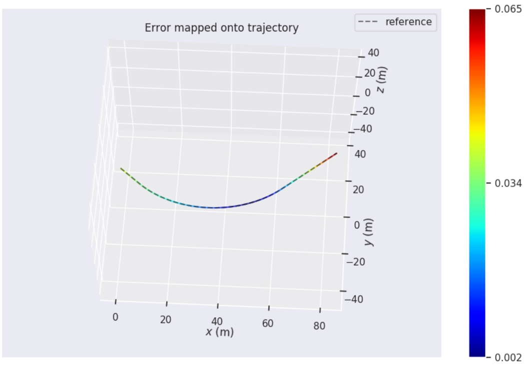
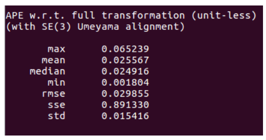
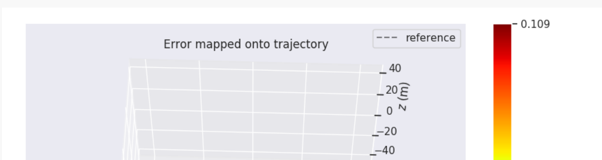
1.数据文件data_folder: 即`gnss-ins-sim`运行之后生成的文件夹

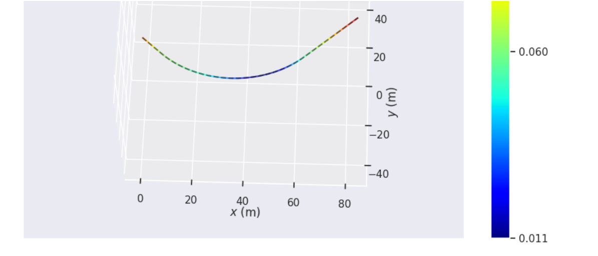
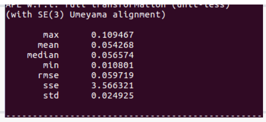
2.运行模式solution_mode: 目前只支持离线的，即使用仿真数据

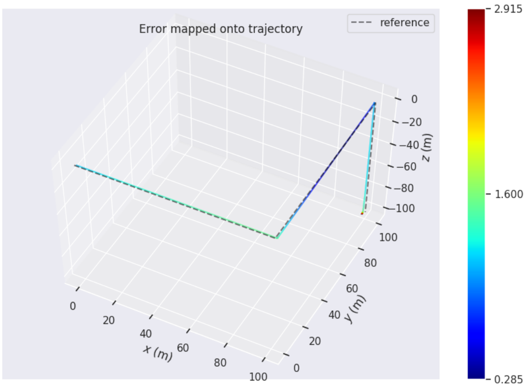
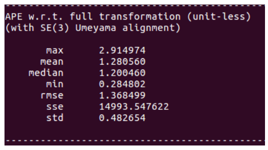
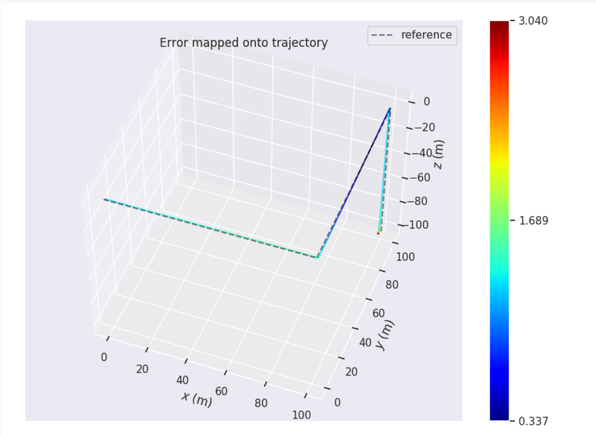
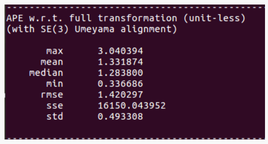
3.是否使用角增量use_fixed_model: 如果是true，则认为姿态更新是固定轴转动，即角增量的方法，否则，则认为是旋转矢量方法。

## 4.3 结果

程序运行完成后，会在对应的数据文件夹下，生成`result`文件夹，用来存储`tum`格式的惯导解算值，以及参考真实值，通过使用`evo`测评工具来进行测试：

<table>
<tr><th>motion_def-90deg_turn</th><th>map</th><th>error</th></tr>
<tr><td>角增量</td><td>
</td><td>
</td></tr>
<tr><td></td><td>
</td><td></td></tr>
</table>

| motion_def-90deg_turn | | |
|---|---|---|
| 旋转矢量 |  | (with SE(3) Umeyama alignment)<br>max 0.109467<br>mean 0.054268<br>median 0.056574<br>min 0.010801<br>rmse 0.059719<br>sse 3.566321<br>std 0.024925 |

| motion_def-3d | map | error |
|---|---|---|
| 角增量 |  | APE w.r.t. full transformation (unit-less)<br>(with SE(3) Umeyama alignment)<br>max 2.914974<br>mean 1.280560<br>median 1.200460<br>min 0.284802<br>rmse 1.368499<br>sse 14993.547622<br>std 0.482654 |
| 旋转矢量 |  | APE w.r.t. full transformation (unit-less)<br>(with SE(3) Umeyama alignment)<br>max 3.040394<br>mean 1.331874<br>median 1.283800<br>min 0.336686<br>rmse 1.420297<br>sse 16150.043952<br>std 0.493308 |

可以看到的是，采用两种方法均可以通过惯导解算得到与真实值接近的结果，但是结果显示，通过角增量的方法得到的结果更加接近真实值。