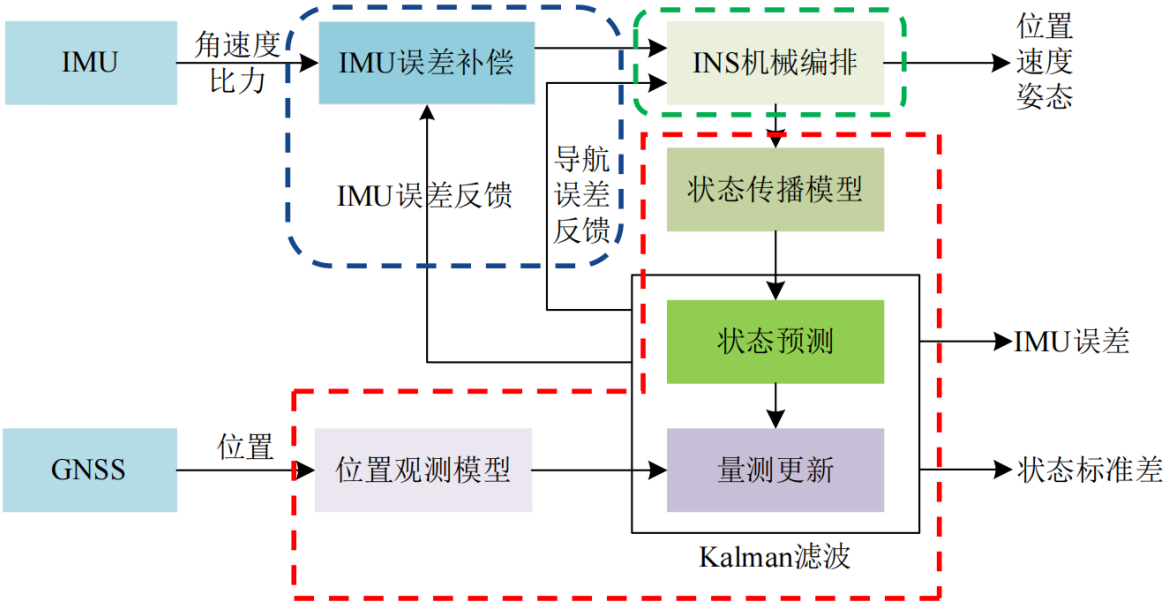
> 原始 Markdown文档、Visio流程图、XMind思维导图见：
> https://github.com/LiZhengXiao99/Navigation-Learning

一、KF-GINS 简介

1、程序概述

KF-GINS 是武大 i2Nav 实验室开源的一套松组合导航程序；可以读取 IMU 数据文件、GNSS 结果文件，进行松组合解算，计算位置、速度、姿态、陀螺仪零偏、加速度计零偏、陀螺仪比例、加速度计比力，共 21 维状态向量。代码量小，有详细的文档、注释和讲解，代码结构很好理解，有一些可以学习的工程技巧。

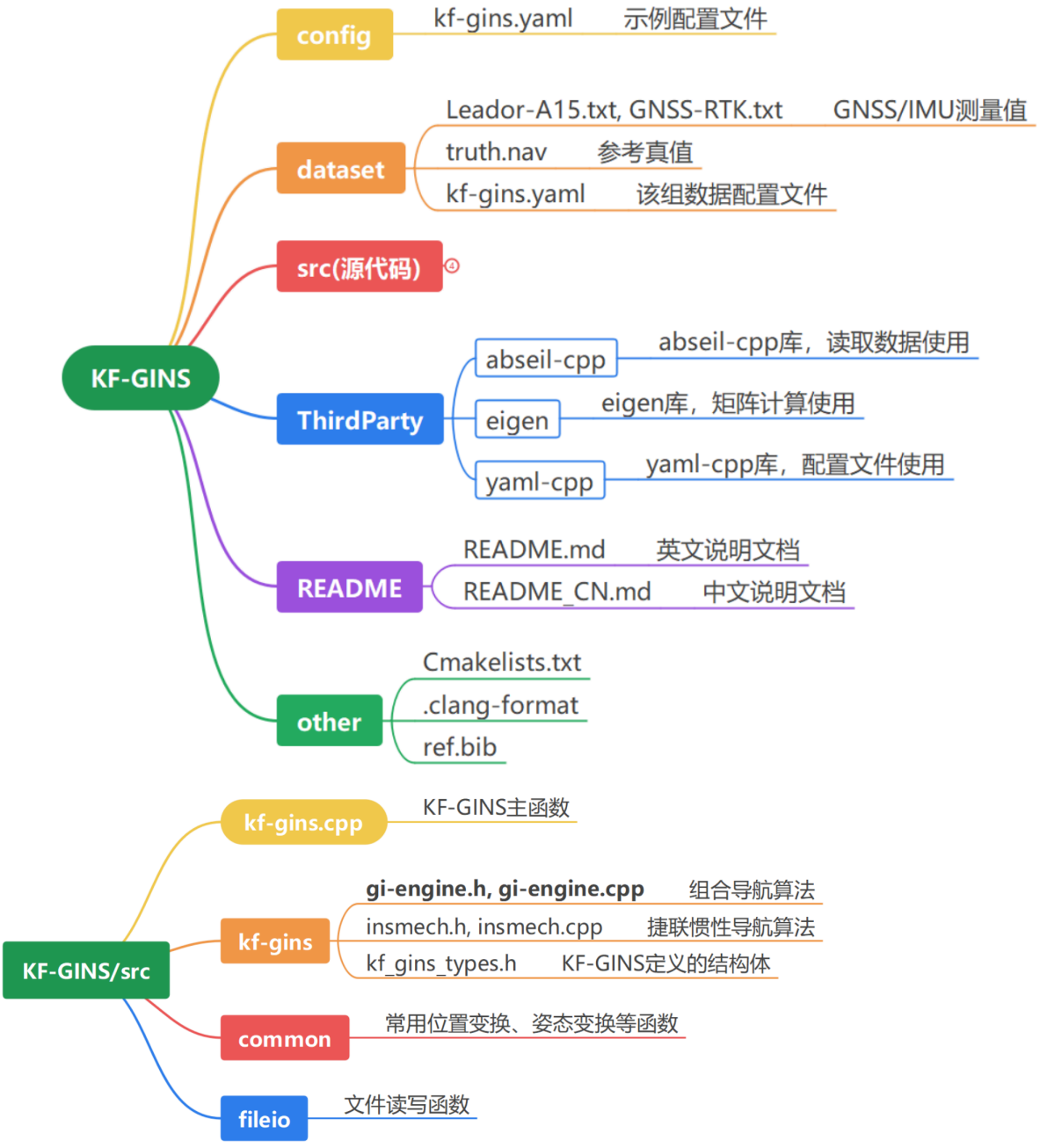


2、相关资料

- **项目开源地址**：https://github.com/i2Nav-WHU

- **i2NAV组合导航讲义、数据集**：http://www.i2nav.cn/index/newList_zw?newskind_id=13a8654e060c40c69e5f3d4c13069078
- **介绍视频**：https://www.bilibili.com/video/BV1Zs4y1B7m2/

3、文件结构



用 cloc 对 src 目录进行统计，结果如下。可以看出代码量很小，只有1412行，注释很详细，足有804行。

```
github.com/AlDanial/cloc v 1.90  T=0.01 s (1254.9 files/s, 184328.2 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C++                              5            182            346            726
C/C++ Header                    13            246            458            686
-------------------------------------------------------------------------------
SUM:                            18            428            804           1412
-------------------------------------------------------------------------------
```

4、第三方库

- **abseil-cpp**：Google的开源C++库，提供了一系列实用的工具和功能，例如字符串处理、时间处理、错误处理、日志记录等。
- **eigen**：用于线性代数、矩阵和向量操作、数值计算和转换。
- **yaml-cpp**：YAML解析器和生成器库。

无需自己配置，作者把它们放到 ThirdParty 文件夹，并在 CMakeLists 文件中引入了：

```
# Eigen3
include_directories(ThirdParty/eigen-3.3.9)

# yaml-cpp-0.7.0
add_subdirectory(ThirdParty/yaml-cpp-0.7.0)
target_link_libraries(${PROJECT_NAME} yaml-cpp)

# abseil
set(ABSL_PROPAGATE_CXX_STD true)
add_subdirectory(ThirdParty/abseil-cpp-20220623.1)
target_link_libraries(${PROJECT_NAME}
        absl::strings
        absl::str_format
        absl::time)
```

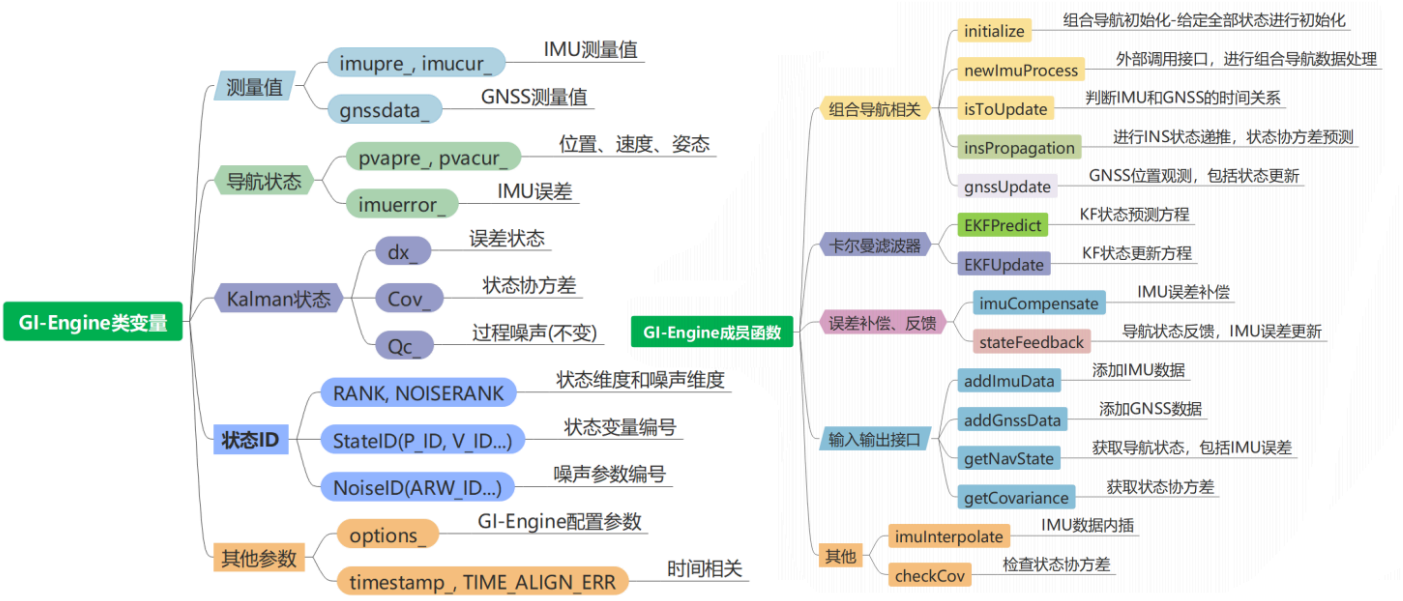
二、编译、调试

基于 WSL + VScode 编译非常容易，用的几个库都直接放到 ThirdParty 文件夹，并在 CMakeLists 文件中引入了，不用我们再配置。如果已经配置好基础的 C++ 环境（cmake、gcc、gdb），把项目 clone 下来之后，选 KF-GINS 目录的 CMakeLists.txt 作为构建目标直接就能构建、编译成功，调试时能停在 main 函数开头设的断点。

launch.json 中作者已经设置命令行参数为配置文件路径，我们只要改好 config 路径下 kf-gins.yaml 配置文件中的几个文件路径（imupath、gnsspath、outputpath）和解算时间（starttime、endtime），就可以跑通示例数据了。

三、类型定义

1、核心类：GIEngine



GIEngine 类

private 字段

更新时间对齐误差，IMU状态和观测信息误差小于它则认为两者对齐：

```
const double TIME_ALIGN_ERR = 0.001;
```

IMU和GNSS原始数据

```
IMU imupre_;
IMU imucur_;
GNSS gnssdata_;
```

IMU状态（位置、速度、姿态和误差）

```
PVA pvacur_;
PVA pvapre_;
ImuError imuerror_;
```

Kalman滤波相关：

```
Eigen::MatrixXd Cov_;
Eigen::MatrixXd Qc_;
Eigen::MatrixXd dx_;
```

状态向量和噪声维数

```
const int RANK      = 21;
const int NOISERANK = 18;
```

状态ID和噪声ID

```
enum StateID { P_ID = 0, V_ID = 3, PHI_ID = 6,
    BG_ID = 9, BA_ID = 12, SG_ID = 15, SA_ID = 18 };
enum NoiseID { VRW_ID = 0, ARW_ID = 3, BGSTD_ID = 6,
    BASTD_ID = 9, SGSTD_ID = 12, SASTD_ID = 15 };
```

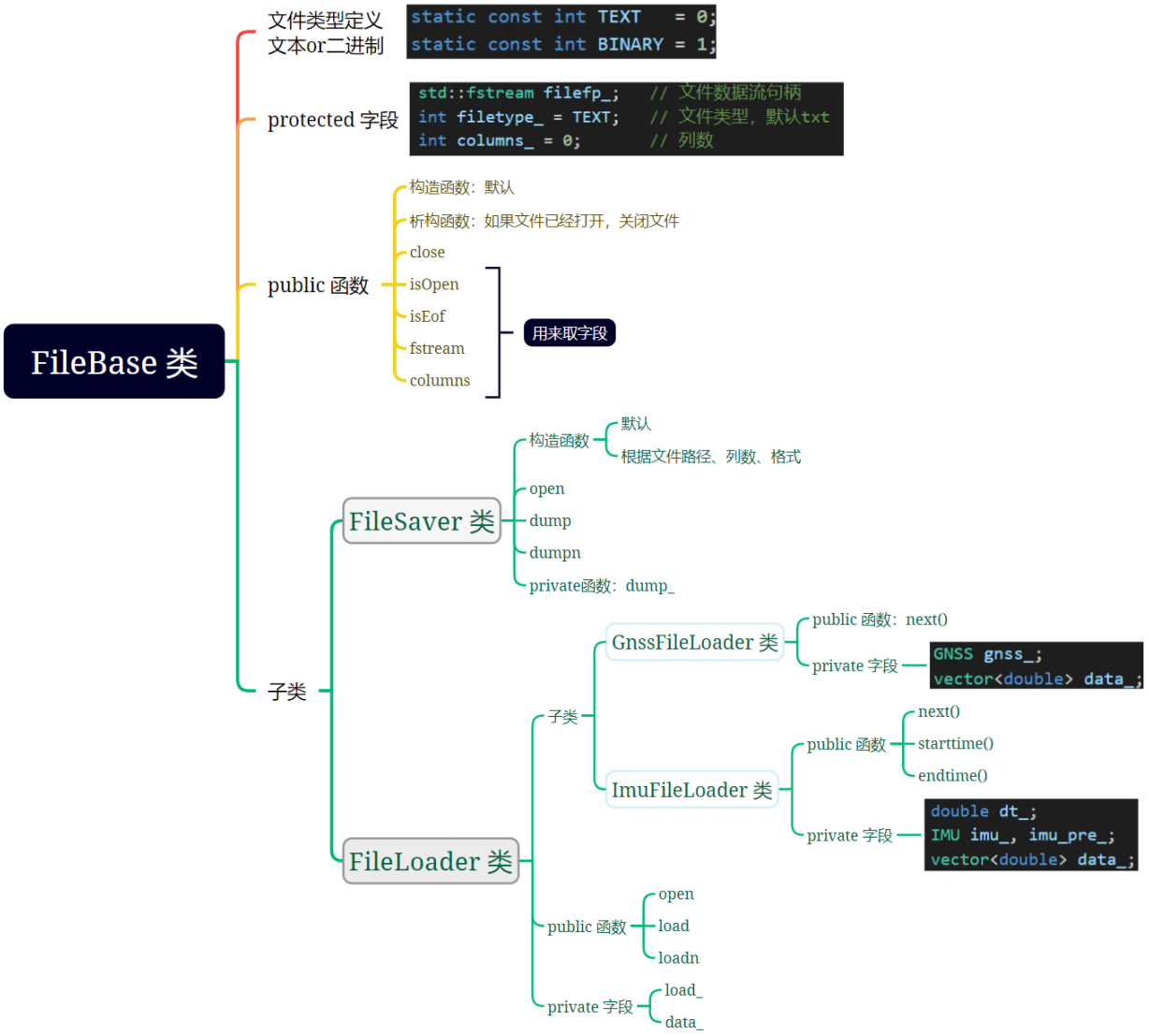
public 函数

- explicit：构造函数
- addImuData：添加新的IMU数据，(不)补偿IMU误差
- addGnssData：添加新的GNSS数据
- newImuProcess：处理新的IMU数据
- imuInterpolate：内插增量形式的IMU数据到指定时刻
- timestamp：获取当前时间
- getNavState：获取当前IMU状态
- getCovariance：获取当前状态协方差

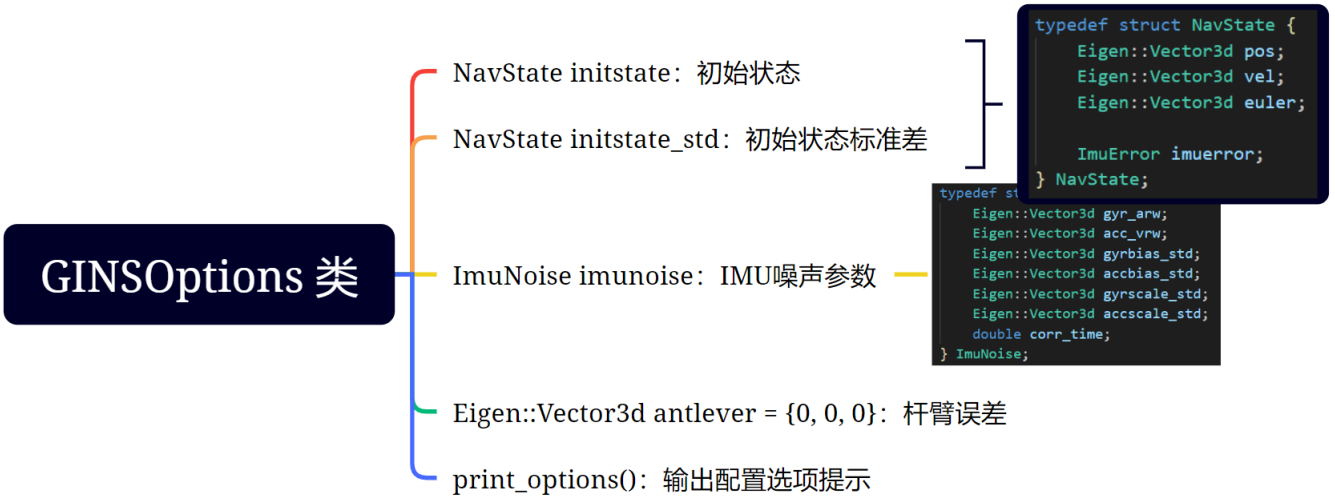
private 函数

- initialize：初始化系统状态和协方差
- imuCompensate：当前IMU误差补偿到IMU数据中
- isToUpdate：判断是否需要更新,以及更新哪一时刻系统状态
- insPropagation：进行INS状态更新(IMU机械编排算法), 并计算IMU状态转移矩阵和噪声阵
- gnssUpdate：使用GNSS位置观测更新系统状态
- EKFPredict：Kalman 预测
- EKFUpdate：Kalman 更新
- stateFeedback：反馈误差状态到当前状态
- checkCov：检查协方差对角线元素是否都为正

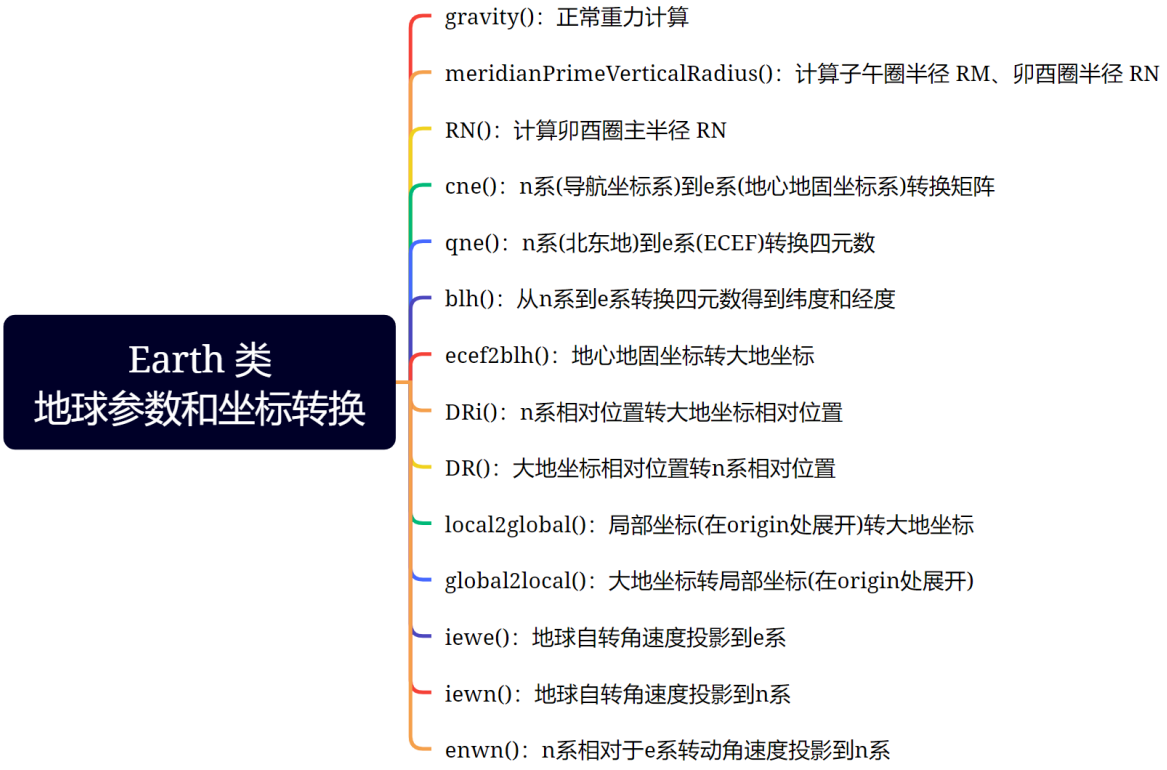
2、文件读写类型



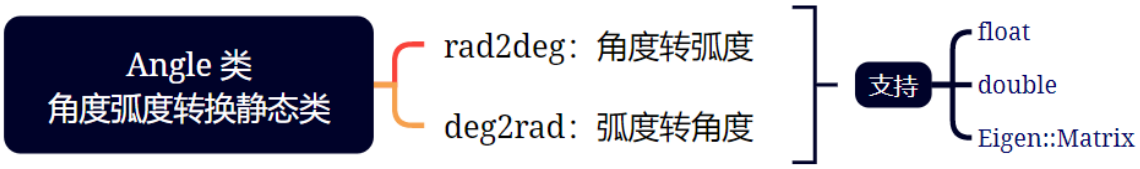
3、配置选项类：GINSOptions



4、大地参数计算静态类：Earth



5、角度弧度转换静态类：Angle

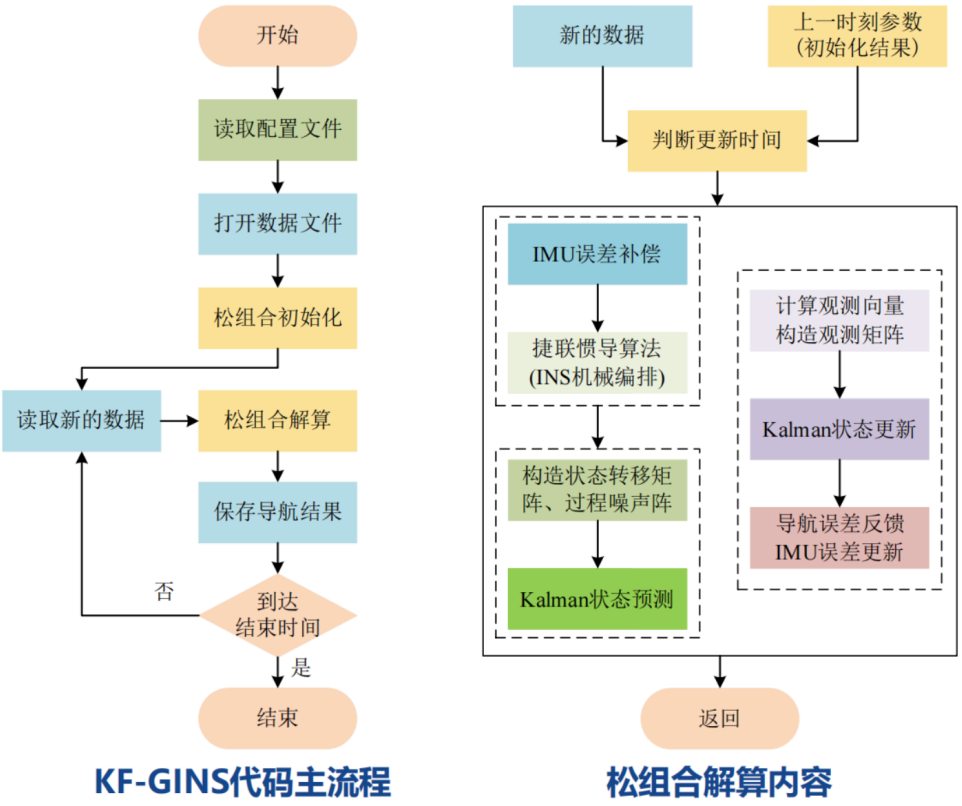


6、姿态转静态类：Rotation

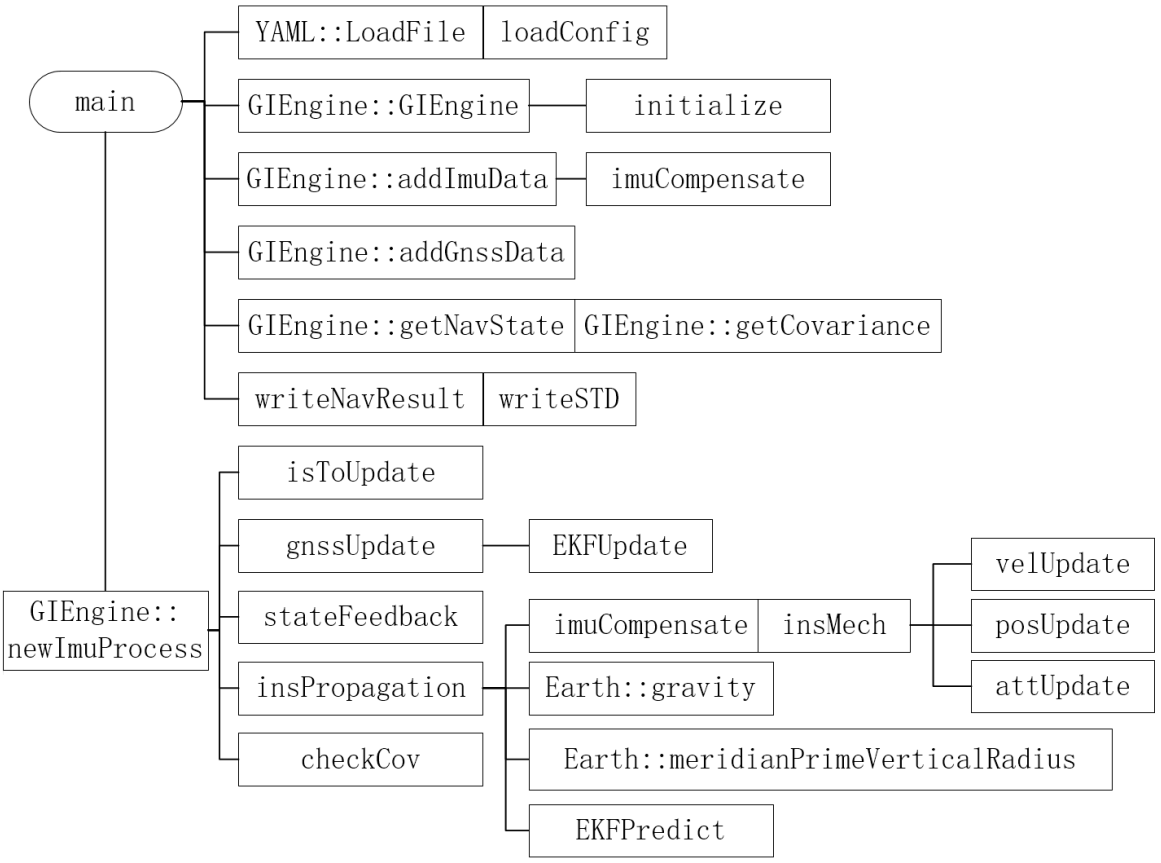
> Rotation、Earth、Angle 里面都只是写了静态函数，没有用到相关的字段，也可以写成命名空间。写命名空间的话，在源文件开头 using 之后，可以省略前面的 `Rotation::`、

> Earth::。

四、程序执行流程



1、函数调用关系



2、重点函数

- `newImuProcess()` 是松组合的核心函数。
- `isToUpdate()` 中根据当前 IMU 和 GNSS 时间戳关系，判断要不要进行 GNSS 量测更新。
- `imuCompensate()` 中进行 IMU 校正，即减去零偏、除以比例。
- `insPropagation()` 中实现捷联惯导 PVA 和噪声递推、构建 F 矩阵。
- `insMech()` 中 IMU 机械编排，依次进行速度更新、位置更新、姿态更新。
- `gnssUpdate()` 中进行 GNSS 量测更新，实现杆臂补偿。
- `stateFeedback()` GNSS 量测更新后，状态向量误差反馈。

3、主函数

首先判断命令行参数，如果不为 2（可执行程序名算第一个参数 `argv[0]`）即没传入配置文件路径，输出提示并退出程序：

```
if (argc != 2) {
    std::cout << "usage: KF-GINS kf-gins.yaml" << std::endl;
    return -1;
}
```

创建 t1、t2、t3 用于计时：

```
long t1,t2,t3; // 用于计时
t1=clock();
std::cout << std::endl << "KF-GINS: An EKF-Based GNSS/INS Integrated Navigation
System" << std::endl << std::endl;
auto ts = absl::Now();
```

读取配置文件：

```
// 加载配置文件
// load configuration file
YAML::Node config;
try {
    config = YAML::LoadFile(argv[1]);
} catch (YAML::Exception &exception) {
    std::cout << "Failed to read configuration file. Please check the path and format
of the configuration file!"
              << std::endl;
    return -1;
}

// 读取配置参数到GINSOptioins中，并构造GIEngine
// load configuration parameters to GINSOptioins
GINSOptions options;
if (!loadConfig(config, options)) {
    std::cout << "Error occurs in the configuration file!" << std::endl;
```

```
    return -1;
}

// 读取文件路径配置
// load filepath configuration
std::string imupath, gnsspath, outputpath;
try {
    imupath    = config["imupath"].as<std::string>();
    gnsspath   = config["gnsspath"].as<std::string>();
    outputpath = config["outputpath"].as<std::string>();
} catch (YAML::Exception &exception) {
    std::cout << "Failed when loading configuration. Please check the file path and
output path!" << std::endl;
    return -1;
}

// imu数据配置，数据处理区间
// imudata configuration， data processing interval
int imudatalen, imudatarate;
double starttime, endtime;
try {
    imudatalen  = config["imudatalen"].as<int>();
    imudatarate = config["imudatarate"].as<int>();
    starttime   = config["starttime"].as<double>();
    endtime     = config["endtime"].as<double>();
} catch (YAML::Exception &exception) {
    std::cout << "Failed when loading configuration. Please check the data length,
data rate, and the process time!"
              << std::endl;
    return -1;
}
```

根据读进来的配置，构造解算用到的几个对象：

- 文件读取对象：`gnssfile`、`imufile`
- 松组合解算核心类：`giengine`
- 构造输出文件对象：`navfile`、`imuerrfile`、`stdfile`

```
// 加载 GNSS 文件和 IMU 文件
// load GNSS file and IMU file
GnssFileLoader gnssfile(gnsspath);
ImuFileLoader imufile(imupath, imudatalen, imudatarate);

t2 =clock();

// 构造GIEngine
// Construct GIEngine
GIEngine giengine(options);

// 构造输出文件
// construct output file
// navfile: gnssweek(1) + time(1) + pos(3) + vel(3) + euler angle(3) = 11
// imuerrfile: time(1) + gyrbias(3) + accbias(3) + gyrscale(3) + accscale(3) = 13
// stdfile: time(1) + pva_std(9) + imubias_std(6) + imuscale_std(6) = 22
int nav_columns = 11, imuerr_columns = 13, std_columns = 22;
FileSaver navfile(outputpath + "/KF_GINS_Navresult.nav", nav_columns,
FileSaver::TEXT);
FileSaver imuerrfile(outputpath + "/KF_GINS_IMU_ERR.txt", imuerr_columns,
FileSaver::TEXT);
```

```
FileSaver stdfile(outputpath + "/KF_GINS_STD.txt", std_columns, FileSaver::TEXT);

// 检查文件是否正确打开
// check if these files are all opened
if (!gnssfile.isOpen() || !imufile.isOpen() || !navfile.isOpen() ||
!imuerrfile.isOpen() || !stdfile.isOpen()) {
    std::cout << "Failed to open data file!" << std::endl;
    return -1;
}
```

检查处理起止时间是否合理，不能小于 0，不能大于周内秒：

```
if (endtime < 0) {
    endtime = imufile.endtime();
}
if (endtime > 604800 || starttime < imufile.starttime() || starttime > endtime) {
    std::cout << "Process time ERROR!" << std::endl;
    return -1;
}
```

循环调用 `imufile.next()`、`gnssfile.next()` 读取 IMU、GNSS 数据，直到时间戳在解算时间范围内。循环结束后 `imu_cur`、`gnss` 分别存解算时间内第一个 IMU、GNSS 量测，且文件指针指向的位置也到达解算时间内数据的开头：

```
IMU imu_cur;
do {
    imu_cur = imufile.next();
} while (imu_cur.time < starttime);

GNSS gnss;
do {
    gnss = gnssfile.next();
} while (gnss.time <= starttime);
```

调用 `addImuData()`、`addGnssData()` 将刚刚读取到解算时间范围内第一个 IMU、GNSS 数据加入 `giengine`，并对 IMU 数据进行误差补偿，减去零偏、除以加上单位阵后的比例：

```
// 添加IMU数据到GIEngine中，补偿IMU误差
// add imudata to GIEngine and compensate IMU error
giengine.addImuData(imu_cur, true);

// 添加GNSS数据到GIEngine
// add gnssdata to GIEngine
giengine.addGnssData(gnss);
```

定义变量，用于保存处理结果、显示处理进度：

```
// 用于保存处理结果
// used to save processing results
```

```
double timestamp;
NavState navstate;
Eigen::MatrixXd cov;

// 用于显示处理进程
// used to display processing progress
int percent = 0, lastpercent = 0;
double interval = endtime - starttime;
```

接下来是一个大 while 死循环，每次循环都会读取一个 IMU 数据，只有当前 IMU 状态时间新于 GNSS 时间时，才会读取 GNSS 数据：

```
// 当前IMU状态时间新于GNSS时间时，读取并添加新的GNSS数据到GIEngine
// load new gnssdata when current state time is newer than GNSS time and add it to
GIEngine
if (gnss.time < imu_cur.time && !gnssfile.isEof()) {
    gnss = gnssfile.next();
    giengine.addGnssData(gnss);
}

// 读取并添加新的IMU数据到GIEngine
// load new imudata and add it to GIEngine
imu_cur = imufile.next();
if (imu_cur.time > endtime || imufile.isEof()) {
    break;
}
giengine.addImuData(imu_cur);
```

调用 `newImuProcess()` 根据当前 IMU、GNSS 数据进行解算，下面会重点介绍：

```
giengine.newImuProcess();
```

解算之后，获取当前时间，IMU状态和协方差、保存并输出处理结果，输出结果的时间戳与 IMU 时间戳一致：

```
// 获取当前时间，IMU状态和协方差
// get current timestamp, navigation state and covariance
timestamp = giengine.timestamp();
navstate  = giengine.getNavState();
cov       = giengine.getCovariance();

// 保存处理结果
// save processing results
writeNavResult(timestamp, navstate, navfile, imuerrfile);
writeSTD(timestamp, cov, stdfile);
```

显示处理进度：

```
percent = int((imu_cur.time - starttime) / interval * 100);
   if (percent - lastpercent >= 1) {
```

```
    std::cout << " - Processing: " << std::setw(3) << percent << "%\r" << std::flush;
    lastpercent = percent;
}
```

循环处理完成之后，关闭打开的文件、输出结束信息、return 0 退出程序：

```
// 关闭打开的文件
// close opened file
imufile.close();
gnssfile.close();
navfile.close();
imuerrfile.close();
stdfile.close();

// 处理完毕
// process finish
auto te = absl::Now();
std::cout << std::endl << std::endl << "KF-GINS Process Finish! ";
std::cout << "From " << starttime << " s to " << endtime << " s, total " << interval
<< " s!" << std::endl;
std::cout << "Cost " << absl::ToDoubleSeconds(te - ts) << " s in total" << std::endl;

return 0;
```

4、配置文件读取

KF-GINS 使用 YMAL 格式的配置文件，通过配置文件可以设置数据文件路径、处理时间段、初始 PVA、初始比例零偏、杆臂等。KF-GINS 的配置都是键值对形式的：`键 :值`，设置的时候改后面的值即可（注意缩进要用空格而不能用 Tab）。程序执行的时候要把配置文件路径作为命令行参数。下面简单介绍读取流程：

在主函数中先调用 yaml-cpp 的接口 `YAML::LoadFile()` 通过 YMAL 配置文件路径，将配置导入为 YMAL 节点 `config`：

```
YAML::Node config;
try {
    config = YAML::LoadFile(argv[1]);
} catch (YAML::Exception &exception) {
    std::cout << "Failed to read configuration file. Please check the path and format
of the configuration file!"
              << std::endl;
    return -1;
}
```

然后调用 `loadConfig()` 从 YMAL 根节点 `config` 读取配置参数到 `GINSOptions` 类型对象 `options` 中：

```
GINSOptions options;
if (!loadConfig(config, options)) {
```

```
    std::cout << "Error occurs in the configuration file!" << std::endl;
    return -1;
}
```

需要注意 `loadConfig()` 并没有把所有配置信息都读进来，它读取的只是取初始位置、IMU零偏、比例和对应的标准差；大部分参数都是三维的，读取的时候先存成 `vector<double>` 然后进行量纲、单位转换，再存到 `options` 对应的 `vector3d` 类型字段中。以初始 PVA 为例：

```
// 读取初始位置(纬度 经度 高程)、(北向速度 东向速度 垂向速度)、姿态(欧拉角，ZYX旋转顺序， 横滚角、俯仰角、航向角)
// load initial position(latitude longitude altitude)
//              velocity(speeds in the directions of north, east and down)
//              attitude(euler angle, ZYX, roll, pitch and yaw)
std::vector<double> vec1, vec2, vec3, vec4, vec5, vec6;
try {
    vec1 = config["initpos"].as<std::vector<double>>();
    vec2 = config["initvel"].as<std::vector<double>>();
    vec3 = config["initatt"].as<std::vector<double>>();
} catch (YAML::Exception &exception) {
    std::cout << "Failed when loading configuration. Please check initial position, velocity, and attitude!"
              << std::endl;
    return false;
}
for (int i = 0; i < 3; i++) {   // 单位转换
    options.initstate.pos[i]   = vec1[i] * D2R;
    options.initstate.vel[i]   = vec2[i];
    options.initstate.euler[i] = vec3[i] * D2R;
}
options.initstate.pos[2] *= R2D;    // 高程不用转
```

文件路径和 IMU 处理配置是在主函数中读取：

```
// 读取文件路径配置
// load filepath configuration
std::string imupath, gnsspath, outputpath;
try {
    imupath    = config["imupath"].as<std::string>();
    gnsspath   = config["gnsspath"].as<std::string>();
    outputpath = config["outputpath"].as<std::string>();
} catch (YAML::Exception &exception) {
    std::cout << "Failed when loading configuration. Please check the file path and output path!" << std::endl;
    return -1;
}

// imu数据配置，数据处理区间
// imudata configuration， data processing interval
int imudatalen, imudatarate;
double starttime, endtime;
try {
    imudatalen  = config["imudatalen"].as<int>();
    imudatarate = config["imudatarate"].as<int>();
    starttime   = config["starttime"].as<double>();
    endtime     = config["endtime"].as<double>();
} catch (YAML::Exception &exception) {
```

```
    std::cout << "Failed when loading configuration. Please check the data length,
data rate, and the process time!"
              << std::endl;
    return -1;
}
```

5、数据文件读取

KF-GINS 中没有一次性把整个文件都读进来；而是先打开文件，获取文件描述符；然后计算一点，读一点。

在主函数中先构造 `GnssFileLoader`、`ImuFileLoader` 类的对象 `gnssfile`、`imufile`：

```
GnssFileLoader gnssfile(gnsspath);
ImuFileLoader imufile(imupath, imudatalen, imudatarate);
```

构造函数中调用 `open()` 函数，将文件打开，获得文件指针，并记录下文件列数和 IMU 采样间隔。

```
explicit GnssFileLoader(const string &filename, int columns = 7) {
    open(filename, columns, FileLoader::TEXT);
}
```

```
ImuFileLoader(const string &filename, int columns, int rate = 200) {
    open(filename, columns, FileLoader::TEXT);

    dt_ = 1.0 / (double) rate;

    imu_.time = 0;
}
```

```
bool FileLoader::open(const string &filename, int columns, int filetype) {
    auto type = filetype == TEXT ? std::ios_base::in : (std::ios_base::in |
std::ios_base::binary);
    filefp_.open(filename, type);

    columns_  = columns;
    filetype_ = filetype;
    return isOpen();
}
```

调用 `imufile.next()`、`gnssfile.next()`，读取一个数据。

在主文件中，先循环调用 `imufile.next()`、`gnssfile.next()` 读取 IMU、GNSS 数据，直到时间戳在解算时间范围内。循环结束后 `imu_cur`、`gnss` 分别存解算时间内第一个 IMU、GNSS

量测，且文件指针指向的位置也到达解算时间内数据的开头：

```
IMU imu_cur;
do {
    imu_cur = imufile.next();
} while (imu_cur.time < starttime);

GNSS gnss;
do {
    gnss = gnssfile.next();
} while (gnss.time <= starttime);
```

之后每次循环解算，都会读取一个 IMU 数据，只有当前 IMU 状态时间新于 GNSS 时间时，才会读取 GNSS 数据：

```
while (true) {
    // 当前IMU状态时间新于GNSS时间时，读取并添加新的GNSS数据到GIEngine
    // load new gnssdata when current state time is newer than GNSS time and add it to
GIEngine
    if (gnss.time < imu_cur.time && !gnssfile.isEof()) {
        gnss = gnssfile.next();
        giengine.addGnssData(gnss);
    }
    // 读取并添加新的IMU数据到GIEngine
    // load new imudata and add it to GIEngine
    imu_cur = imufile.next();
    if (imu_cur.time > endtime || imufile.isEof()) {
        break;
    }
```

6、GIEngine 构造函数

传入配置选项来构造，输出配置选项、时间戳置 0、设置协方差矩阵，系统噪声阵和系统误差状态矩阵大小、初始化系统噪声阵，最后调用 initialize() 赋值初始的状态量、协方差。

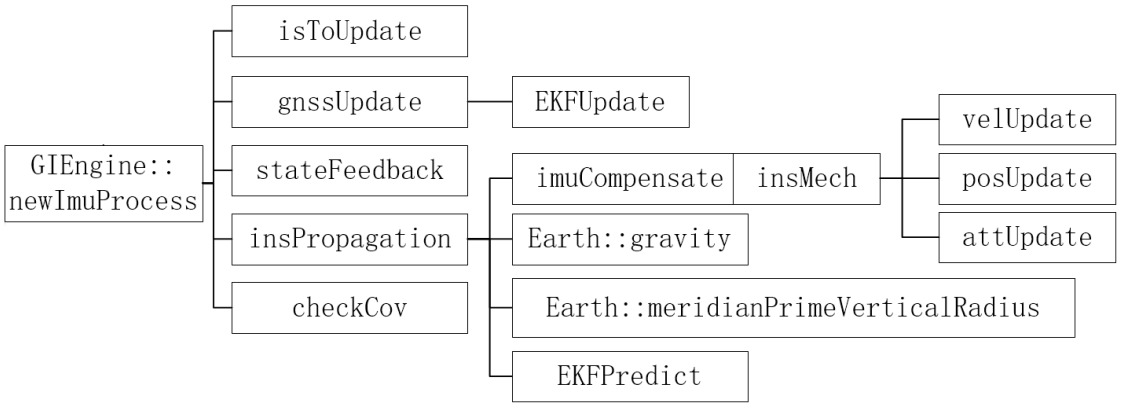
$$\underset{18\times18}{\mathbf{q}}(t)=\begin{bmatrix} \mathrm{VRW}^2\mathbf{I}_{3\times3} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathrm{ARW}^2\mathbf{I}_{3\times3} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \frac{2\sigma_{gb}^2}{T_{gb}}\mathbf{I}_{3\times3} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{2\sigma_{ab}^2}{T_{ab}}\mathbf{I}_{3\times3} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{2\sigma_{gs}^2}{T_{gs}}\mathbf{I}_{3\times3} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{2\sigma_{as}^2}{T_{as}}\mathbf{I}_{3\times3} \end{bmatrix}$$

```
GIEngine::GIEngine(GINSOptions &options) {

    this->options_ = options;
    options_.print_options();
    timestamp_ = 0;
```

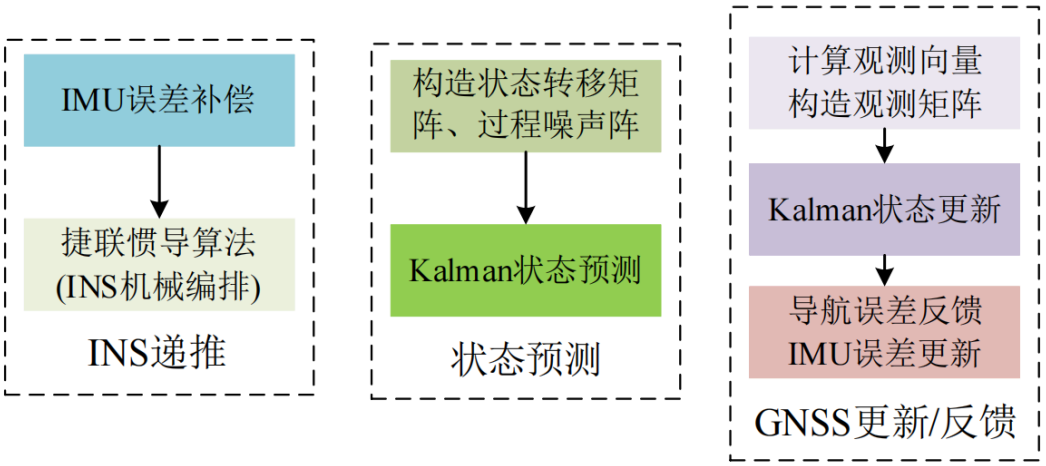
```
    // 设置协方差矩阵，系统噪声阵和系统误差状态矩阵大小
    // resize covariance matrix, system noise matrix, and system error state matrix
    Cov_.resize(RANK, RANK);
    Qc_.resize(NOISERANK, NOISERANK);
    dx_.resize(RANK, 1);
    Cov_.setZero();
    Qc_.setZero();
    dx_.setZero();

    // 初始化系统噪声阵
    // initialize noise matrix
    auto imunoise                   = options_.imunoise;
    Qc_.block(ARW_ID, ARW_ID, 3, 3) =
imunoise.gyr_arw.cwiseProduct(imunoise.gyr_arw).asDiagonal();
    Qc_.block(VRW_ID, VRW_ID, 3, 3) =
imunoise.acc_vrw.cwiseProduct(imunoise.acc_vrw).asDiagonal();
    Qc_.block(BGSTD_ID, BGSTD_ID, 3, 3) =
        2 / imunoise.corr_time *
imunoise.gyrbias_std.cwiseProduct(imunoise.gyrbias_std).asDiagonal();
    Qc_.block(BASTD_ID, BASTD_ID, 3, 3) =
        2 / imunoise.corr_time *
imunoise.accbias_std.cwiseProduct(imunoise.accbias_std).asDiagonal();
    Qc_.block(SGSTD_ID, SGSTD_ID, 3, 3) =
        2 / imunoise.corr_time *
imunoise.gyrscale_std.cwiseProduct(imunoise.gyrscale_std).asDiagonal();
    Qc_.block(SASTD_ID, SASTD_ID, 3, 3) =
        2 / imunoise.corr_time *
imunoise.accscale_std.cwiseProduct(imunoise.accscale_std).asDiagonal();

    // 设置系统状态(位置、速度、姿态和IMU误差)初值和初始协方差
    // set initial state (position, velocity, attitude and IMU error) and covariance
    initialize(options_.initstate, options_.initstate_std);
}

void GIEngine::initialize(const NavState &initstate, const NavState &initstate_std) {

    // 初始化位置、速度、姿态
    // initialize position, velocity and attitude
    pvacur_.pos       = initstate.pos;
    pvacur_.vel       = initstate.vel;
    pvacur_.att.euler = initstate.euler;
    pvacur_.att.cbn   = Rotation::euler2matrix(pvacur_.att.euler);
    pvacur_.att.qbn   = Rotation::euler2quaternion(pvacur_.att.euler);
    // 初始化IMU误差
    // initialize imu error
    imuerror_ = initstate.imuerror;

    // 给上一时刻状态赋同样的初值
    // set the same value to the previous state
    pvapre_ = pvacur_;

    // 初始化协方差
    // initialize covariance
    ImuError imuerror_std           = initstate_std.imuerror;
    Cov_.block(P_ID, P_ID, 3, 3)    =
initstate_std.pos.cwiseProduct(initstate_std.pos).asDiagonal();
    Cov_.block(V_ID, V_ID, 3, 3)    =
initstate_std.vel.cwiseProduct(initstate_std.vel).asDiagonal();
    Cov_.block(PHI_ID, PHI_ID, 3, 3) =
initstate_std.euler.cwiseProduct(initstate_std.euler).asDiagonal();
    Cov_.block(BG_ID, BG_ID, 3, 3)   =
imuerror_std.gyrbias.cwiseProduct(imuerror_std.gyrbias).asDiagonal();
```

```
    Cov_.block(BA_ID, BA_ID, 3, 3)   =
imuerror_std.accbias.cwiseProduct(imuerror_std.accbias).asDiagonal();
    Cov_.block(SG_ID, SG_ID, 3, 3)   =
imuerror_std.gyrscale.cwiseProduct(imuerror_std.gyrscale).asDiagonal();
    Cov_.block(SA_ID, SA_ID, 3, 3)   =
imuerror_std.accscale.cwiseProduct(imuerror_std.accscale).asDiagonal();
}
```

7、newImuProcess()：松组合



这个函数是松组合解算的入口，IMU 量测的频率远远大于 GNSS 量测；所以用 IMU 为基准，得到的系统状态向量和协方差阵是当前 IMU 时间的，每次调用这个函数都会取新 IMU 量测。函数的计算基于当前时刻 IMU 量测和上一时刻 IMU 量测，如果两次量测之间没有 GNSS 数据，就只是进行捷联惯导递推，将系统状态和噪声递推到当前时刻；如果两次量测间有 GNSS 数据，就先捷联惯导递推到 GNSS 时刻，在 GNSS 时刻进行量测更新、误差反馈，最后再捷联惯导递推到当前时刻。



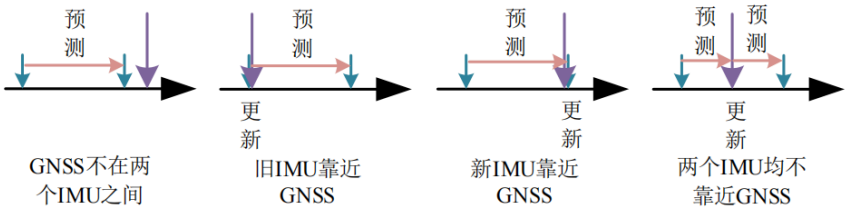
首先将当前 IMU 时间作为系统当前状态时间，也就是说这个函数执行完之后，得到的系统状态向量和协方差阵是当前 IMU 时间的：

```
timestamp_ = imucur_.time;
```

如果GNSS有效，则将量测更新时间设置为 GNSS 时间：

```
double updatetime = gnssdata_.isvalid ? gnssdata_.time : -1;
```

先调用 `isToUpdate()`，根据当前 GNSS 与当前和先前两 IMU 量测的时间关系，判断是否需要进行 GNSS 更新，有四种情况，分别返回不同的值：




```
int GIEngine::isToUpdate(double imutime1, double imutime2, double updatetime) const {
    if (abs(imutime1 - updatetime) < TIME_ALIGN_ERR) {
        // 更新时间靠近imutime1
        // updatetime is near to imutime1
        return 1;
    } else if (abs(imutime2 - updatetime) <= TIME_ALIGN_ERR) {
        // 更新时间靠近imutime2
        // updatetime is near to imutime2
        return 2;
    } else if (imutime1 < updatetime && updatetime < imutime2) {
        // 更新时间在imutime1和imutime2之间, 但不靠近任何一个
        // updatetime is between imutime1 and imutime2, but not near to either
        return 3;
    } else {
        // 更新时间不在imutimt1和imutime2之间，且不靠近任何一个
        // updatetime is not bewteen imutime1 and imutime2, and not near to either.
        return 0;
    }
}
```

> - 根据更新时间对齐误差 `TIME_ALIGN_ERR` 评定是否对齐，默认为 0.001，也就是说时间差距在 1ms 内，认为对齐的。
> - KF-GINS 的数据采集的时候进行了时间对齐，每个有 GNSS 数据的时刻正常都应该有一个 IMU 数据，判断出有 GNSS 数据前后 0.001s 都没有 IMU 数据，就说明，插值一个IMU量测到GNSS时刻。
> - 内插的方法不适合实时导航，想实时得外推。

返回 0，表示 GNSS 不在两个 IMU 之间，在当前 IMU 量测之后，那么只进行捷联惯导递推，调用 `insPropagation()` 根据两帧 IMU 量测将状态递推到当前 IMU 时间戳：

```
if (res == 0) {
    // 只传播导航状态
    // only propagate navigation state
    insPropagation(imupre_, imucur_);
```

返回 1，表示 GNSS 在两个 IMU 之间，更靠近前一个 IMU：

- 先调用 `gnssUpdate()` 进行 GNSS 量测更新；
- 再调用 `stateFeedback()` 进行系统状态反馈；
- 最后调用 `insPropagation()` 根据两帧 IMU 量测将状态递推到当前 IMU 时间戳：

```
} else if (res == 1) {
    // GNSS数据靠近上一历元，先对上一历元进行GNSS更新
    // gnssdata is near to the previous imudata, we should firstly do gnss update
    gnssUpdate(gnssdata_);
    stateFeedback();

    pvapre_ = pvacur_;
    insPropagation(imupre_, imucur_);
```

返回 2，表示 GNSS 在两个 IMU 之间，更靠近后一个 IMU：

- 先调用 `insPropagation()` 根据两帧 IMU 量测将状态递推到当前 IMU 时间戳；
- 再调用 `gnssUpdate()` 进行 GNSS 量测更新；
- 最后调用 `stateFeedback()` 进行系统状态反馈：

```
} else if (res == 2) {
    // GNSS数据靠近当前历元，先对当前IMU进行状态传播
    // gnssdata is near current imudata, we should firstly propagate navigation state
    insPropagation(imupre_, imucur_);
    gnssUpdate(gnssdata_);
    stateFeedback();
```

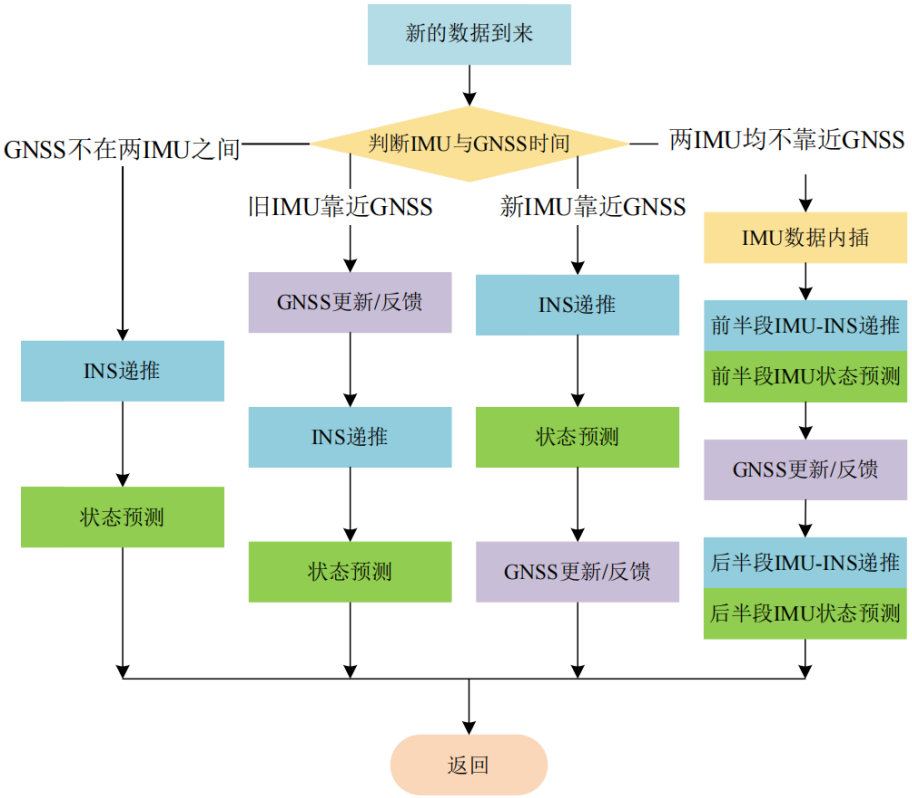
返回 3，表示 GNSS 在两个 IMU 之间，不靠近任何一个：

- 先调用 `imuInterpolate()` 根据两帧 IMU 量测插值到 GNSS 时间戳，得到 GNSS 时刻 IMU 量测值 `midimu`；
- 调用 insPropagation 根据前一个 IMU 和 `midimu` 将状态递推到当前 GNSS 时间戳；
- 再调用 `gnssUpdate()` 进行 GNSS 量测更新，调用 `stateFeedback()` 进行系统状态反馈；
- 最后再调用一次 `insPropagation()` 根据 midimu 和当前时刻 IMU 量测将状态递推到当前 IMU 时间戳：

```
} else {
    // GNSS数据在两个IMU数据之间(不靠近任何一个)，将当前IMU内插到整秒时刻
    // gnssdata is between the two imudata, we interpolate current imudata to gnss
time
    IMU midimu;
    imuInterpolate(imupre_, imucur_, updatetime, midimu);
    // 对前一半IMU进行状态传播
    // propagate navigation state for the first half imudata
    insPropagation(imupre_, midimu);
    // 整秒时刻进行GNSS更新，并反馈系统状态
```

```
    // do GNSS position update at the whole second and feedback system states
    gnssUpdate(gnssdata_);
    stateFeedback();
    // 对后一半IMU进行状态传播
    // propagate navigation state for the second half imudata
    pvapre_ = pvacur_;
    insPropagation(midimu, imucur_);
}
```

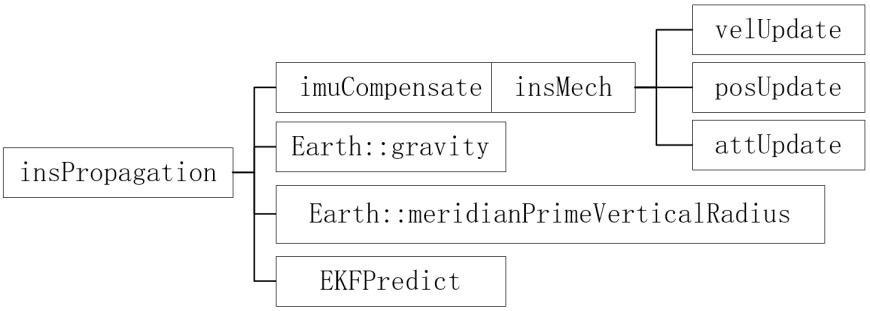
几种情况可总结如下图：



处理完之后调用 `checkCov()` 检查协方差对角线元素是否都为正，更新上一时刻的状态和 IMU 数据：

```
// 检查协方差对角线元素是否都为正
// check diagonal elements of current covariance matrix
checkCov();
// 更新上一时刻的状态和IMU数据
// update system state and imudata at the previous epoch
pvapre_ = pvacur_;
imupre_ = imucur_;
```

五、捷联惯导更新：insPropagation()



1、insPropagation()：捷联惯导递推

根据两帧的 IMU 量测，将系统状态和误差状态从前一个 IMU 时间递推到后一个 IMU 时间；主要有三个步骤：IMU 误差补偿、状态更新（机械编排）、噪声传播。

先调用 `imuCompensate()`，补偿当前时刻 IMU 量测，就是减去零偏、除以加上单位阵后的比例：

```
imuCompensate(imucur);
```

调用 `insMech()` 依次进行速度更新、位置更新、姿态更新：

```
INSMech::insMech(pvapre_, pvacur_, imupre, imucur);
```

之后一大段是误差传播，后面详细介绍。

2、imuCompensate()：IMU数据误差补偿

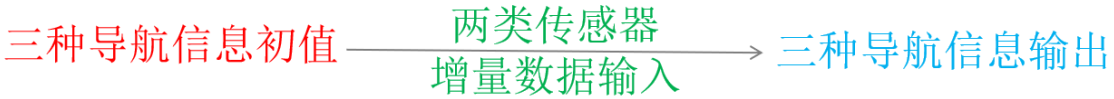
减去零偏、除以加上单位阵后的比例：

$$\mathrm{diag}\,(\boldsymbol{I}+\bar{\boldsymbol{s}}_g)^{-1}\,(\tilde{\boldsymbol{\omega}}_{ib}^{b}-\bar{\boldsymbol{b}}_g)=\boldsymbol{\omega}_{ib}^{b}$$
$$\mathrm{diag}\,(\boldsymbol{I}+\bar{\boldsymbol{s}}_a)^{-1}\,(\tilde{\boldsymbol{f}}_{ib}^{b}-\bar{\boldsymbol{b}}_a)=\boldsymbol{f}_{ib}^{b}$$

```
void GIEngine::imuCompensate(IMU &imu) {
    // 补偿IMU零偏
    // compensate the imu bias
    imu.dtheta -= imuerror_.gyrbias * imu.dt;
    imu.dvel -= imuerror_.accbias * imu.dt;
    // 补偿IMU比例因子
    // compensate the imu scale
    Eigen::Vector3d gyrscale, accscale;
    gyrscale   = Eigen::Vector3d::Ones() + imuerror_.gyrscale;
    accscale   = Eigen::Vector3d::Ones() + imuerror_.accscale;
    imu.dtheta = imu.dtheta.cwiseProduct(gyrscale.cwiseInverse());
```

```
    imu.dvel    = imu.dvel.cwiseProduct(accscale.cwiseInverse());
}
```

3、insMech()：IMU 状态更新（机械编排）



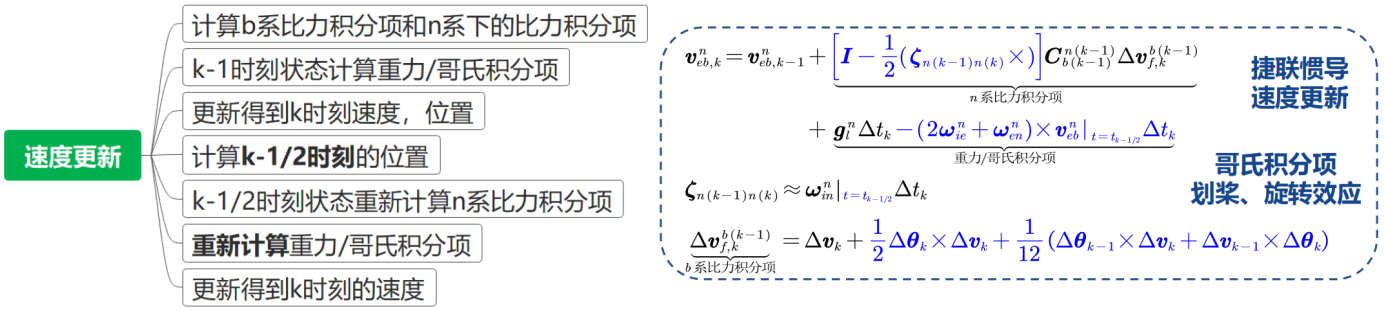
依次进行速度更新、位置更新、姿态更新，不可调换顺序。


```
void INSMech::insMech(const PVA &pvapre, PVA &pvacur, const IMU &imupre, const IMU
&imucur) {
    // perform velocity update, position updata and attitude update in sequence,
irreversible order
    // 依次进行速度更新、位置更新、姿态更新，不可调换顺序
    velUpdate(pvapre, pvacur, imupre, imucur);
    posUpdate(pvapre, pvacur, imupre, imucur);
    attUpdate(pvapre, pvacur, imupre, imucur);
}
```

> - PVA 更新都是先计算中间时刻的速度位置，进而计算中间时刻地球相关参数，再由此计算当前时刻 PVA；我觉得相比直接用上一时刻地球相关参数计算当前 PVA，精度提升不大。
> - 位置更新中：先计算 n 系到 e 系旋转四元数，再调用 `blh()`计算经纬度。
> - 我觉得因为 PVA 写成三个函数，部分计算过程有重复，写在同一个函数实现能更简洁一些。

4、velUpdate()：速度更新

1. 算法



速度更新主要有两部分要算：

- **比力积分项**：其中需要补偿划桨效应，是快变量，需要仔细积分。
- **有害加速度积分项**：包括重力、哥氏加速度、向心力，是慢变量，直接梯形积分，用中间时刻来算。

然后，当前时刻的速度 = 上一时刻速度 + 比力积分项 + 有害加速度积分项：

$$\boldsymbol{v}_{eb,k}^{n} = \boldsymbol{v}_{eb,k-1}^{n} + \underbrace{[\boldsymbol{I} - \frac{1}{2}(\boldsymbol{\zeta}_{n(k-1)n(k)}\times)]\,\boldsymbol{C}_{b(k-1)}^{n(k-1)}\Delta\boldsymbol{v}_{f,k}^{b(k-1)}}_{n\text{ 系比力积分项}} + \underbrace{\boldsymbol{g}_{l}^{n}\Delta t_k - (2\boldsymbol{\omega}_{i}^{n} + \boldsymbol{\omega}_{en}^{n})\times \boldsymbol{v}_{eb}^{n}\mid_{t=t_{k-1/2}}\Delta t_k}_{\text{重力/哥氏积分项}}$$

2. 代码实现

先定义解算过程中涉及的中间变量：

```
Eigen::Vector3d d_vfb, d_vfn, d_vgn, gl, midvel, midpos;
Eigen::Vector3d temp1, temp2, temp3;
Eigen::Matrix3d cnn, I33 = Eigen::Matrix3d::Identity();
Eigen::Quaterniond qne, qee, qnn, qbb, q1, q2;
```

调用 `meridianPrimeVerticalRadius()`，根据上一时刻位置计算**子午圈、卯酉圈半径**：

$$R_M = \frac{R_e(1-e^2)}{(1-e^2\sin^2 L)^{3/2}}、\quad R_N = \frac{R_e}{\sqrt{1-e^2\sin^2 L}}$$

```
Eigen::Vector2d rmrn = Earth::meridianPrimeVerticalRadius(pvapre.pos(0));
```

计算地球自转引起的导航系旋转 `wie_n`：

$$\boldsymbol{\omega}_{ie}^{n} = [\ \omega_e\cos\varphi \quad 0 \quad -\omega_e\sin\varphi\ ]^{\mathrm{T}}$$

```
wie_n << WGS84_WIE * cos(pvapre.pos[0]), 0, -WGS84_WIE * sin(pvapre.pos[0]);
```

计算载体在地球表面移动因地球曲率引起的导航系旋转 wen_n:

$$\boldsymbol{\omega}_{en}^{n} = \begin{bmatrix} v_E/(R_N+h) \\ -v_N/(R_M+h) \\ -v_E \tan\varphi/(R_N+h) \end{bmatrix}$$

```
wen_n << pvapre.vel[1] / (rmrn[1] + pvapre.pos[2]), -pvapre.vel[0] / (rmrn[0] +
pvapre.pos[2]),
    -pvapre.vel[1] * tan(pvapre.pos[0]) / (rmrn[1] + pvapre.pos[2]);
```

调用 gravity()，根据上一时刻位置计算**重力 ** gravity:

$$g_L = 9.7803267715 \times (1 + 0.0052790414 \times \sin^2 L - 0.0000232718 \times \sin^2 2L)$$
$$+h \times (0.0000000043977311 \times \sin^2 L - 0.0000030876910891) + 0.0000000000007211 \times \sin^4 2L$$

```
double gravity = Earth::gravity(pvapre.pos);
```

计算 b 系比力积分项 d_vfb，单子样 + 前一周期补偿划桨效应:

$$\underbrace{\Delta\boldsymbol{v}_{f,k-1)}^{b(k-1)}}_{b\text{ 系比力积分项}} = \Delta\boldsymbol{v}_k + \frac{1}{2}\Delta\boldsymbol{\theta}_k \times \Delta\boldsymbol{v}_k + \frac{1}{12}(\Delta\boldsymbol{\theta}_{k-1} \times \Delta\boldsymbol{v}_k + \Delta\boldsymbol{v}_{k-1} \times \Delta\boldsymbol{\theta}_k)$$

```
// 旋转效应和双子样划桨效应
// rotational and sculling motion
temp1 = imucur.dtheta.cross(imucur.dvel) / 2;
temp2 = imupre.dtheta.cross(imucur.dvel) / 12;
temp3 = imupre.dvel.cross(imucur.dtheta) / 12;
// b系比力积分项
// velocity increment due to the specific force
d_vfb = imucur.dvel + temp1 + temp2 + temp3;
```

比力积分项投影到 n 系，三行代码分别对应的公式为:

$$\boldsymbol{\omega}_{in}^{n} = \boldsymbol{\omega}_{ie}^{n} + \boldsymbol{\omega}_{en}^{n}$$

$$\boldsymbol{\zeta}_{n(k-1)n(k)} \approx \boldsymbol{\omega}_{in}^{n}\Big|_{t=t_{k-1/2}} \Delta t_k$$

$$\underbrace{[\boldsymbol{I} - \frac{1}{2}(\boldsymbol{\zeta}_{n(k-1)n(k)}\times)]\,\boldsymbol{C}_{b(k-1)}^{n(k-1)}\Delta\boldsymbol{v}_{f,k}^{b(k-1)}}_{n\text{ 系比力积分项}}$$

```
// 比力积分项投影到n系
// velocity increment dut to the specfic force projected to the n-frame
temp1 = (wie_n + wen_n) * imucur.dt / 2;
```

```
cnn   = I33 - Rotation::skewSymmetric(temp1);
d_vfn = cnn * pvapre.att.cbn * d_vfb;
```

计算重力/哥式积分项：

$$\underbrace{\boldsymbol{g}_l^n \Delta t_k - (2\boldsymbol{\omega}_i^n + \boldsymbol{\omega}_{en}^n) \times \boldsymbol{v}_{eb}^n \mid_{t=t_{k-1/2}} \Delta t_k}_{\text{重力/哥氏积分项}}$$

```
// 计算重力/哥式积分项
// velocity increment due to the gravity and Coriolis force
gl << 0, 0, gravity;
d_vgn = (gl - (2 * wie_n + wen_n).cross(pvapre.vel)) * imucur.dt;
```

上一时刻速度加上一半比力积分项和比力积分项，得到中间时刻速度：

```
// 得到中间时刻速度
// velocity at k-1/2
midvel = pvapre.vel + (d_vfn + d_vgn) / 2;
```

外推得到中间时刻位置：

- 计算两时刻n系旋转四元数 `qnn`
- 根据地球自转角速率，计算两时刻e系旋转四元数 `qee`
- 调用 `qne` 根据先前时刻位置，计算先前时刻n系到e系旋转四元数 `qne`
- 当前时刻 n 系到 e 系旋转四元数 `qne` = 两时刻e系旋转四元数 * 先前n系到e系旋转四元数 * 两时刻 n 系旋转四元数
- 中间时刻高程 = 先前高程 - 高程方向速度 * 一半采样周期（因为北东地，计算出的速度时地向的，所以减）
- 调用 `blh()` 根据 n系到e系旋转四元数计算经纬度

```
// 外推得到中间时刻位置
// position extrapolation to k-1/2
qnn = Rotation::rotvec2quaternion(temp1);
temp2 << 0, 0, -WGS84_WIE * imucur.dt / 2;
qee = Rotation::rotvec2quaternion(temp2);
qne = Earth::qne(pvapre.pos);
qne = qee * qne * qnn;
midpos[2] = pvapre.pos[2] - midvel[2] * imucur.dt / 2;
midpos    = Earth::blh(qne, midpos[2]);
```

基于用中间时刻的位置，重新做一遍之前的操作：

- 重新计算中间时刻的地理参数：`rmrn`、`wie_n`、`wen_n`（重力没重算）
- 重新计算 n 系下平均比力积分项：`d_vfn`
- 重新计算重力、哥式积分项：`d_vgn`

```
// 重新计算中间时刻的 rmrn, wie_e, wen_n
// recompute rmrn, wie_n, and wen_n at k-1/2
rmrn = Earth::meridianPrimeVerticalRadius(midpos[0]);
wie_n << WGS84_WIE * cos(midpos[0]), 0, -WGS84_WIE * sin(midpos[0]);
wen_n << midvel[1] / (rmrn[1] + midpos[2]), -midvel[0] / (rmrn[0] + midpos[2]),
    -midvel[1] * tan(midpos[0]) / (rmrn[1] + midpos[2]);

// 重新计算n系下平均比力积分项
// recompute d_vfn
temp3 = (wie_n + wen_n) * imucur.dt / 2;
cnn   = I33 - Rotation::skewSymmetric(temp3);
d_vfn = cnn * pvapre.att.cbn * d_vfb;

// 重新计算重力、哥式积分项
// recompute d_vgn
gl << 0, 0, Earth::gravity(midpos);
d_vgn = (gl - (2 * wie_n + wen_n).cross(midvel)) * imucur.dt;
```

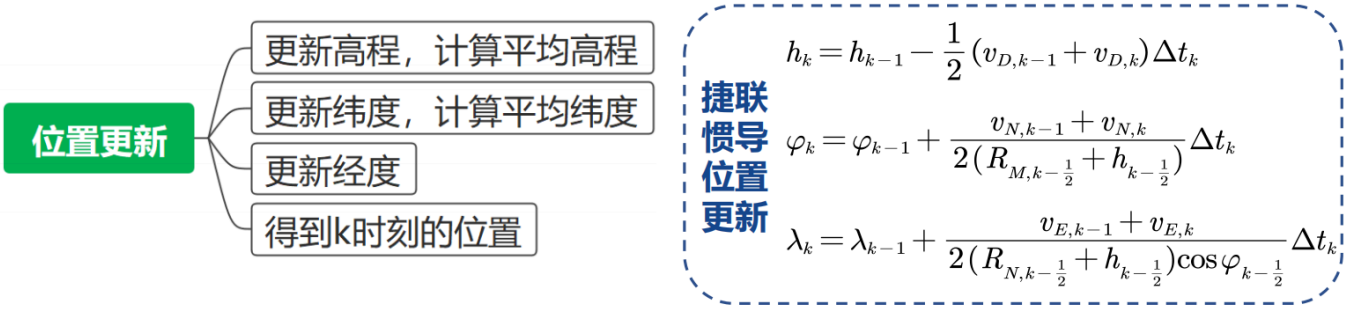
最后，用上一时刻的速度，加上n系下平均比力积分项、重力/哥式积分项，得到当前时刻速度：

$$\boldsymbol{v}_{eb,k}^{n} = \boldsymbol{v}_{eb,k-1}^{n} + \underbrace{[\boldsymbol{I} - \frac{1}{2}(\boldsymbol{\zeta}_{n(k-1)n(k)}\times)]\,\boldsymbol{C}_{b(k-1)}^{n(k-1)}\Delta\boldsymbol{v}_{f,k}^{b(k-1)}}_{n\text{ 系比力积分项}} + \underbrace{\boldsymbol{g}_{l}^{n}\Delta t_k - (2\boldsymbol{\omega}_{i}^{n} + \boldsymbol{\omega}_{en}^{n}) \times \boldsymbol{v}_{eb}^{n} \mid_{t=t_{k-1/2}} \Delta t_k}_{\text{重力/哥氏积分项}}$$

```
pvacur.vel = pvapre.vel + d_vfn + d_vgn;
```

5、posUpdate()：位置更新

1. 算法



先计算当前时刻 n 系到 e 系旋转四元数 qne，再以此算经纬度。其中，当前时刻 qne 的计算分为三部分：

- 先前时刻 qne：通过上一时刻经纬度计算。
- 两时刻 n 系变化 qnn：由地球自转角速度、牵连角速度两部分引起。

- 两时刻 e 系变化 qee：地球自转角速度乘以时间差。

然后，当前时刻n系到e系旋转四元数 = 两时刻e系旋转四元数 * 先前n系到e系旋转四元数 * 两时刻n系旋转四元数。

2. 代码实现

先定义解算过程中涉及的中间变量：

```
Eigen::Vector3d temp1, temp2, midvel, midpos;
Eigen::Quaterniond qne, qee, qnn;
```

重新计算中间时刻的速度 `midvel` 和位置 `midpos`：

- 中间时刻速度：取两时刻的平均。
- 中间时刻位置：上一时刻位置 + 平均速度 * 一半采样间隔。

```
// 重新计算中间时刻的速度和位置
// recompute velocity and position at k-1/2
midvel = (pvacur.vel + pvapre.vel) / 2;
midpos = pvapre.pos + Earth::DRi(pvapre.pos) * midvel * imucur.dt / 2;
```

根据中间时刻位置， 重新计算中间时刻地理参数（除了重力）：

```
// 重新计算中间时刻地理参数
// recompute rmrn, wie_n, wen_n at k-1/2
Eigen::Vector2d rmrn;
Eigen::Vector3d wie_n, wen_n;
rmrn = Earth::meridianPrimeVerticalRadius(midpos[0]);
wie_n << WGS84_WIE * cos(midpos[0]), 0, -WGS84_WIE * sin(midpos[0]);
wen_n << midvel[1] / (rmrn[1] + midpos[2]), -midvel[0] / (rmrn[0] + midpos[2]),
    -midvel[1] * tan(midpos[0]) / (rmrn[1] + midpos[2]);
```

重新计算 k 时刻到 k-1 时刻 n 系旋转矢量：

$$\omega_{in}^{n} = \omega_{ie}^{n} + \omega_{en}^{n}$$

$$\zeta_{n(k-1)n(k)} \approx \boldsymbol{\omega}_{in}^{n}\Big|_{t=t_{k-1/2}} \Delta t_k$$

```
// 重新计算 k 时刻到 k-1 时刻 n系旋转矢量
// recompute n-frame rotation vector (n(k) with respect to n(k-1)-frame)
temp1 = (wie_n + wen_n) * imucur.dt;
qnn   = Rotation::rotvec2quaternion(temp1);
```

e 系转动等效旋转矢量 (k-1时刻k时刻，所以取负号)，直接就是地球自转角速率乘以时间差：

```
// e系转动等效旋转矢量 (k-1时刻k时刻，所以取负号)
// e-frame rotation vector (e(k-1) with respect to e(k)-frame)
temp2 << 0, 0, -WGS84_WIE * imucur.dt;
qee = Rotation::rotvec2quaternion(temp2);
```

由先前时刻位置，调用 `qne()`，得到先前n系到e系旋转四元数：

```
qne = Earth::qne(pvapre.pos);
```

当前时刻n系到e系旋转四元数 = 两时刻e系旋转四元数 * 先前n系到e系旋转四元数 * 两时刻n系旋转四元数：

```
qne = qee * qne * qnn;
```

当前时刻高程 = 先前高程 - 高程方向速度 * 采样间隔（因为北东地，计算出的速度时地向的，所以减）：

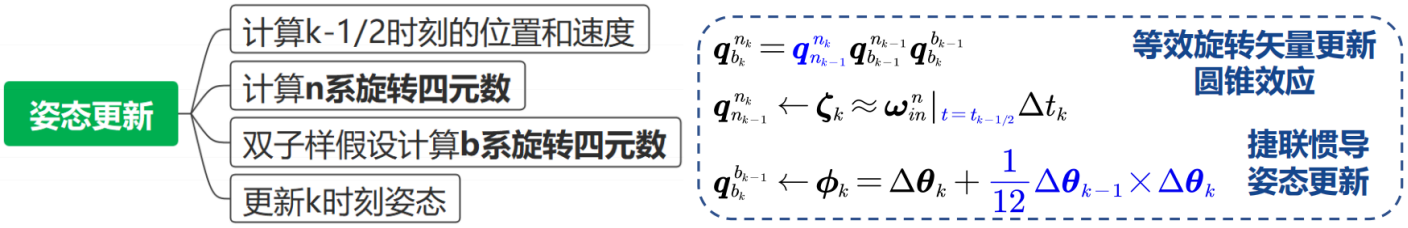
```
pvacur.pos[2] = pvapre.pos[2] - midvel[2] * imucur.dt;
```

调用 `blh()` 根据 n 系到 e 系旋转四元数计算经纬度：

```
pvacur.pos = Earth::blh(qne, pvacur.pos[2]);
```

6、attUpdate()：姿态更新

1. 算法



姿态更新主要也算两部分：

- **两时刻 n 系的变化**：由地球自转角速度、牵连角速度两部分引起。
- **两时刻 b 系的变化**：通过 IMU 角增量量测值计算，需补偿圆锥运动。

然后，两时刻 n 系的旋转四元数 * 上一时刻姿态四元数 * 两时刻 b 系旋转四元数得到当前姿态：

$$\boldsymbol{q}_{b_k}^{n_k} = \boldsymbol{q}_{n_{k-1}}^{n_k} \boldsymbol{q}_{b_{k-1}}^{n_{k-1}} \boldsymbol{q}_{b_k}^{b_{k-1}}$$

2. 代码实现

先定义解算过程中涉及的中间变量：

```
Eigen::Quaterniond qne_pre, qne_cur, qne_mid, qnn, qbb;
Eigen::Vector3d temp1, midpos, midvel;
```

重新计算中间时刻的速度和位置，中间速度是两时刻平均、中间位置相对于是作了一次位置更新：

- 根据两时刻速度，计算平均速度 `midvel`
- 根据上一时刻位置计算n系到e系转换四元数 `qne_pre`
- 根据当前时刻位置计算n系到e系转换四元数 `qne_cur`
- 根据两时刻转换四元数，计算n系到e系平均转换四元数 `qne_mid`（注意得通过等效旋转矢量，并非直接插值）
- 计算当前中间时刻位置 `midpos`

```
// 重新计算中间时刻的速度和位置
// recompute velocity and position at k-1/2
midvel = (pvapre.vel + pvacur.vel) / 2;
qne_pre   = Earth::qne(pvapre.pos);
qne_cur   = Earth::qne(pvacur.pos);
temp1     = Rotation::quaternion2vector(qne_cur.inverse() * qne_pre);
qne_mid   = qne_pre * Rotation::rotvec2quaternion(temp1 / 2).inverse();
midpos[2] = (pvacur.pos[2] + pvapre.pos[2]) / 2;
midpos    = Earth::blh(qne_mid, midpos[2]);
```

重新计算中间时刻地理参数：

```
// 重新计算中间时刻地理参数
// recompute rmrn, wie_n, wen_n at k-1/2
Eigen::Vector2d rmrn;
Eigen::Vector3d wie_n, wen_n;
rmrn = Earth::meridianPrimeVerticalRadius(midpos[0]);
wie_n << WGS84_WIE * cos(midpos[0]), 0, -WGS84_WIE * sin(midpos[0]);
wen_n << midvel[1] / (rmrn[1] + midpos[2]), -midvel[0] / (rmrn[0] + midpos[2]),
    -midvel[1] * tan(midpos[0]) / (rmrn[1] + midpos[2]);
```

计算 n 系的旋转四元数 k-1 时刻到 k 时刻系旋转：

$$\boldsymbol{\omega}_{in}^{n} = \boldsymbol{\omega}_{ie}^{n} + \boldsymbol{\omega}_{en}^{n}$$

$$\boldsymbol{\zeta}_{n(k-1)n(k)} \approx \boldsymbol{\omega}_{in}^{n}\Big|_{t=t_{k-1/2}} \Delta t_k$$

```
// 重新计算 k时刻到k-1时刻 n系旋转矢量
// recompute n-frame rotation vector (n(k) with respect to n(k-1)-frame)
temp1 = (wie_n + wen_n) * imucur.dt;
qnn   = Rotation::rotvec2quaternion(temp1);
```

计算 b 系旋转四元数补偿二阶圆锥误差:

$$\boldsymbol{q}_{b_k}^{b_{k-1}} \leftarrow \boldsymbol{\phi}_k = \Delta\boldsymbol{\theta}_k + \frac{1}{12}\Delta\boldsymbol{\theta}_{k-1} \times \Delta\boldsymbol{\theta}_k$$

```
// 计算b系旋转四元数 补偿二阶圆锥误差
// b-frame rotation vector (b(k) with respect to b(k-1)-frame)
// compensate the second-order coning correction term.
temp1 = imucur.dtheta + imupre.dtheta.cross(imucur.dtheta) / 12;
qbb   = Rotation::rotvec2quaternion(temp1);
```

两时刻 n 系的旋转四元数 * 上一时刻姿态四元数 * 两时刻 b 系旋转四元数得到当前姿态:

$$\boldsymbol{q}_{b_k}^{n_k} = \boldsymbol{q}_{n_{k-1}}^{n_k}\boldsymbol{q}_{b_{k-1}}^{n_{k-1}}\boldsymbol{q}_{b_k}^{b_{k-1}}$$

```
// 姿态更新完成
// attitude update finish
pvacur.att.qbn   = qnn * pvapre.att.qbn * qbb;
pvacur.att.cbn   = Rotation::quaternion2matrix(pvacur.att.qbn);
pvacur.att.euler = Rotation::matrix2euler(pvacur.att.cbn);
```

7、误差传播

就是协方差的更新:

- 先构造连续时间的 F 矩阵，离散化得到状态转移矩阵 $\boldsymbol{\Phi}_{k/k-1} = \boldsymbol{I} + \boldsymbol{F}_{k-1}\Delta t_k$ ，
- `gi_engine` 初始化的用角速度随机游走 arw、加速度随机游走 vrw，角速度零偏白噪声、加速度零偏白噪声、角速度比例、加速度零偏比例构造了恒定的 18 维噪声阵 `Qc_`。然后每次惯导更新的时候计算一个 21×18 维 噪声驱动阵 `G` ，计算得到噪声阵。

$$\boldsymbol{Q}_k = \begin{pmatrix} \boldsymbol{\Phi}_{k/k-1}\boldsymbol{G}_{k-1}\boldsymbol{q}_{k-1}\boldsymbol{G}_{k-1}^T\boldsymbol{\Phi}_{k/k-1}^T \\ +\boldsymbol{G}_k\boldsymbol{q}_k\boldsymbol{G}_k^T \end{pmatrix} \Delta t_k/2$$

- 用状态转移矩阵和噪声阵卡尔曼滤波时间更新协方差阵 `cov_` 和状态向量 `dx_`，如果误差反馈状态向量是 0，无需更新。

$$\boldsymbol{x}_{k/k-1} = \boldsymbol{\Phi}_{k/k-1}\boldsymbol{x}_{k-1}$$
$$\boldsymbol{P}_{k/k-1} = \boldsymbol{\Phi}_{k/k-1}\boldsymbol{P}_{k-1}\boldsymbol{\Phi}_{k/k-1}^T + \boldsymbol{Q}_k$$

下面介绍具体公式和代码：

在 `insPropagation()` 函数中，IMU状态更新之后进行。

先要构建 F 矩阵：

$$\mathbf{F} = \begin{bmatrix} \mathbf{F}_{rr} & \mathbf{I}_{3\times3} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{F}_{vr} & \mathbf{F}_{vv} & [(\mathbf{C}_b^n \boldsymbol{f}^b)\times] & \mathbf{0} & \mathbf{C}_b^n & \mathbf{0} & \mathbf{C}_b^n \operatorname{diag}(\boldsymbol{f}^b) \\ \mathbf{F}_{\phi r} & \mathbf{F}_{\phi v} & -(\boldsymbol{\omega}_{in}^n \times) & -\mathbf{C}_b^n & \mathbf{0} & -\mathbf{C}_b^n \operatorname{diag}(\boldsymbol{\omega}_{ib}^b) & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{-1}{T_{gb}}\mathbf{I}_{3\times3} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{-1}{T_{ab}}\mathbf{I}_{3\times3} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{-1}{T_{gs}}\mathbf{I}_{3\times3} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{-1}{T_{as}}\mathbf{I}_{3\times3} \end{bmatrix}$$

上面 F 矩阵每一块都是一个 3×3 矩阵，分别表示位置、速度、姿态、陀螺仪零偏、加速度计零偏、陀螺仪比例、加速度计比力，共21维。可以很明显的看出左上角几乎是满的，因为位置、速度、姿态之间误差耦合很深；右下的器件误差很稀释，因为我们认为器件与器件之间的量测是相互独立的，处理在对角线上有元素之外，只在与器件相关的位置有元素（陀螺仪和姿态相关、加速度计和速度相关）。

定义了枚举值来索引左上角元素的下标：

```
enum StateID { P_ID = 0, V_ID = 3, PHI_ID = 6,
    BG_ID = 9, BA_ID = 12, SG_ID = 15, SA_ID = 18 };
```

代码排的很整齐，遵循从左往右、从上往下的顺序计算：

```
// 位置误差
// position error
temp.setZero();
temp(0, 0)                = -pvapre_.vel[2] / rmh;
temp(0, 2)                = pvapre_.vel[0] / rmh;
temp(1, 0)                = pvapre_.vel[1] * tan(pvapre_.pos[0]) / rnh;
temp(1, 1)                = -(pvapre_.vel[2] + pvapre_.vel[0] * tan(pvapre_.pos[0])) /
rnh;
temp(1, 2)                = pvapre_.vel[1] / rnh;
F.block(P_ID, P_ID, 3, 3) = temp;
F.block(P_ID, V_ID, 3, 3) = Eigen::Matrix3d::Identity();

// 速度误差
// velocity error
temp.setZero();
temp(0, 0) = -2 * pvapre_.vel[1] * WGS84_WIE * cos(pvapre_.pos[0]) / rmh -
             pow(pvapre_.vel[1], 2) / rmh / rnh / pow(cos(pvapre_.pos[0]), 2);
temp(0, 2) = pvapre_.vel[0] * pvapre_.vel[2] / rmh / rmh - pow(pvapre_.vel[1], 2) *
tan(pvapre_.pos[0]) / rnh / rnh;
temp(1, 0) = 2 * WGS84_WIE * (pvapre_.vel[0] * cos(pvapre_.pos[0]) - pvapre_.vel[2] *
sin(pvapre_.pos[0])) / rmh +
             pvapre_.vel[0] * pvapre_.vel[1] / rmh / rnh / pow(cos(pvapre_.pos[0]),
2);
temp(1, 2) = (pvapre_.vel[1] * pvapre_.vel[2] + pvapre_.vel[0] * pvapre_.vel[1] *
```

```
tan(pvapre_.pos[0])) / rnh / rnh;
temp(2, 0) = 2 * WGS84_WIE * pvapre_.vel[1] * sin(pvapre_.pos[0]) / rmh;
temp(2, 2) = -pow(pvapre_.vel[1], 2) / rnh / rnh - pow(pvapre_.vel[0], 2) / rmh / rmh
+
             2 * gravity / (sqrt(rmrn[0] * rmrn[1]) + pvapre_.pos[2]);
F.block(V_ID, P_ID, 3, 3) = temp;
temp.setZero();
temp(0, 0)                  = pvapre_.vel[2] / rmh;
temp(0, 1)                  = -2 * (WGS84_WIE * sin(pvapre_.pos[0]) + pvapre_.vel[1] *
tan(pvapre_.pos[0]) / rnh);
temp(0, 2)                  = pvapre_.vel[0] / rmh;
temp(1, 0)                  = 2 * WGS84_WIE * sin(pvapre_.pos[0]) + pvapre_.vel[1] *
tan(pvapre_.pos[0]) / rnh;
temp(1, 1)                  = (pvapre_.vel[2] + pvapre_.vel[0] * tan(pvapre_.pos[0]))
/ rnh;
temp(1, 2)                  = 2 * WGS84_WIE * cos(pvapre_.pos[0]) + pvapre_.vel[1] /
rnh;
temp(2, 0)                  = -2 * pvapre_.vel[0] / rmh;
temp(2, 1)                  = -2 * (WGS84_WIE * cos(pvapre_.pos(0)) + pvapre_.vel[1] /
rnh);
F.block(V_ID, V_ID, 3, 3)   = temp;
F.block(V_ID, PHI_ID, 3, 3) = Rotation::skewSymmetric(pvapre_.att.cbn * accel);
F.block(V_ID, BA_ID, 3, 3)  = pvapre_.att.cbn;
F.block(V_ID, SA_ID, 3, 3)  = pvapre_.att.cbn * (accel.asDiagonal());

// 姿态误差
// attitude error
temp.setZero();
temp(0, 0) = -WGS84_WIE * sin(pvapre_.pos[0]) / rmh;
temp(0, 2) = pvapre_.vel[1] / rnh / rnh;
temp(1, 2) = -pvapre_.vel[0] / rmh / rmh;
temp(2, 0) = -WGS84_WIE * cos(pvapre_.pos[0]) / rmh - pvapre_.vel[1] / rmh / rnh /
pow(cos(pvapre_.pos[0]), 2);
temp(2, 2) = -pvapre_.vel[1] * tan(pvapre_.pos[0]) / rnh / rnh;
F.block(PHI_ID, P_ID, 3, 3) = temp;
temp.setZero();
temp(0, 1)                    = 1 / rnh;
temp(1, 0)                    = -1 / rmh;
temp(2, 1)                    = -tan(pvapre_.pos[0]) / rnh;
F.block(PHI_ID, V_ID, 3, 3)   = temp;
F.block(PHI_ID, PHI_ID, 3, 3) = -Rotation::skewSymmetric(wie_n + wen_n);
F.block(PHI_ID, BG_ID, 3, 3)  = -pvapre_.att.cbn;
F.block(PHI_ID, SG_ID, 3, 3)  = -pvapre_.att.cbn * (omega.asDiagonal());

// IMU零偏误差和比例因子误差，建模成一阶高斯-马尔科夫过程
// imu bias error and scale error, modeled as the first-order Gauss-Markov process
F.block(BG_ID, BG_ID, 3, 3) = -1 / options_.imunoise.corr_time *
Eigen::Matrix3d::Identity();
F.block(BA_ID, BA_ID, 3, 3) = -1 / options_.imunoise.corr_time *
Eigen::Matrix3d::Identity();
F.block(SG_ID, SG_ID, 3, 3) = -1 / options_.imunoise.corr_time *
Eigen::Matrix3d::Identity();
F.block(SA_ID, SA_ID, 3, 3) = -1 / options_.imunoise.corr_time *
Eigen::Matrix3d::Identity();
```

其中：

$$\mathbf{F}_{rr} = \begin{bmatrix} -\frac{v_D}{R_M+h} & 0 & \frac{v_N}{R_M+h} \\ \frac{v_E \tan\varphi}{R_N+h} & -\frac{v_D+v_N \tan\varphi}{R_N+h} & \frac{v_E}{R_N+h} \\ 0 & 0 & 0 \end{bmatrix}$$

```
temp.setZero();
temp(0, 0) = -pvapre_.vel[2] / rmh;
temp(0, 2) = pvapre_.vel[0] / rmh;
temp(1, 0) = pvapre_.vel[1] * tan(pvapre_.pos[0]) / rnh;
temp(1, 1) = -(pvapre_.vel[2] + pvapre_.vel[0] * tan(pvapre_.pos[0])) / rnh;
temp(1, 2) = pvapre_.vel[1] / rnh;
F.block(P_ID, P_ID, 3, 3) = temp;
```

$$\mathbf{F}_{vr} = \begin{bmatrix} \frac{-2v_E\omega_e\cos\varphi}{R_M+h} - \frac{v_E^2\sec^2\varphi}{(R_M+h)(R_N+h)} & 0 & \frac{v_Nv_D}{(R_M+h)^2} - \frac{v_E^2\tan\varphi}{(R_N+h)^2} \\ \frac{2\omega_e(v_N\cos\varphi - v_D\sin\varphi)}{R_M+h} + \frac{v_Nv_E\sec^2\varphi}{(R_M+h)(R_N+h)} & 0 & \frac{v_Ev_D+v_Nv_E\tan\varphi}{(R_N+h)^2} \\ \frac{2\omega_ev_E\sin\varphi}{R_M+h} & 0 & -\frac{v_E^2}{(R_N+h)^2} - \frac{v_N^2}{(R_M+h)^2} + \frac{2g_p}{\sqrt{R_MR_N+h}} \end{bmatrix}$$

```
temp.setZero();
temp(0, 0) = -2 * pvapre_.vel[1] * WGS84_WIE * cos(pvapre_.pos[0]) / rmh -
             pow(pvapre_.vel[1], 2) / rmh / rnh / pow(cos(pvapre_.pos[0]), 2);
temp(0, 2) = pvapre_.vel[0] * pvapre_.vel[2] / rmh / rmh - pow(pvapre_.vel[1], 2) *
tan(pvapre_.pos[0]) / rnh / rnh;
temp(1, 0) = 2 * WGS84_WIE * (pvapre_.vel[0] * cos(pvapre_.pos[0]) - pvapre_.vel[2] *
sin(pvapre_.pos[0])) / rmh +
             pvapre_.vel[0] * pvapre_.vel[1] / rmh / rnh / pow(cos(pvapre_.pos[0]),
2);
temp(1, 2) = (pvapre_.vel[1] * pvapre_.vel[2] + pvapre_.vel[0] * pvapre_.vel[1] *
tan(pvapre_.pos[0])) / rnh / rnh;
temp(2, 0) = 2 * WGS84_WIE * pvapre_.vel[1] * sin(pvapre_.pos[0]) / rmh;
temp(2, 2) = -pow(pvapre_.vel[1], 2) / rnh / rnh - pow(pvapre_.vel[0], 2) / rmh / rmh
+
             2 * gravity / (sqrt(rmrn[0] * rmrn[1]) + pvapre_.pos[2]);
F.block(V_ID, P_ID, 3, 3) = temp;
```

$$\mathbf{F}_{vv} = \begin{bmatrix} \frac{v_D}{R_M+h} & -2\left(\omega_e\sin\varphi + \frac{v_E\tan\varphi}{R_N+h}\right) & \frac{v_N}{R_M+h} \\ 2\omega_e\sin\varphi + \frac{v_E\tan\varphi}{R_N+h} & \frac{v_D+v_N\tan\varphi}{R_N+h} & 2\omega_e\cos\varphi + \frac{v_E}{R_N+h} \\ -\frac{2v_N}{R_M+h} & -2\left(\omega_e\cos\varphi + \frac{v_E}{R_N+h}\right) & 0 \end{bmatrix}$$

```
temp.setZero();
temp(0, 0) = pvapre_.vel[2] / rmh;
temp(0, 1) = -2 * (WGS84_WIE * sin(pvapre_.pos[0]) + pvapre_.vel[1] *
tan(pvapre_.pos[0]) / rnh);
temp(0, 2) = pvapre_.vel[0] / rmh;
temp(1, 0) = 2 * WGS84_WIE * sin(pvapre_.pos[0]) + pvapre_.vel[1] *
tan(pvapre_.pos[0]) / rnh;
temp(1, 1) = (pvapre_.vel[2] + pvapre_.vel[0] * tan(pvapre_.pos[0])) / rnh;
temp(1, 2) = 2 * WGS84_WIE * cos(pvapre_.pos[0]) + pvapre_.vel[1] / rnh;
temp(2, 0) = -2 * pvapre_.vel[0] / rmh;
temp(2, 1) = -2 * (WGS84_WIE * cos(pvapre_.pos(0)) + pvapre_.vel[1] / rnh);
F.block(V_ID, V_ID, 3, 3)   = temp;
```

$$\mathbf{F}_{\phi r} = \begin{bmatrix} -\frac{\omega_e \sin\varphi}{R_M+h} & 0 & \frac{v_E}{(R_N+h)^2} \\ 0 & 0 & -\frac{v_N}{(R_M+h)^2} \\ -\frac{\omega_e \cos\varphi}{R_M+h} - \frac{v_E \sec^2\varphi}{(R_M+h)(R_N+h)} & 0 & -\frac{v_E \tan\varphi}{(R_N+h)^2} \end{bmatrix}$$

```
temp.setZero();
temp(0, 0) = -WGS84_WIE * sin(pvapre_.pos[0]) / rmh;
temp(0, 2) = pvapre_.vel[1] / rnh / rnh;
temp(1, 2) = -pvapre_.vel[0] / rmh / rmh;
temp(2, 0) = -WGS84_WIE * cos(pvapre_.pos[0]) / rmh - pvapre_.vel[1] / rmh / rnh /
pow(cos(pvapre_.pos[0]), 2);
temp(2, 2) = -pvapre_.vel[1] * tan(pvapre_.pos[0]) / rnh / rnh;
F.block(PHI_ID, P_ID, 3, 3) = temp;
```

$$\mathbf{F}_{\phi v} = \begin{bmatrix} 0 & \frac{1}{R_N+h} & 0 \\ -\frac{1}{R_M+h} & 0 & 0 \\ 0 & -\frac{\tan\varphi}{R_N+h} & 0 \end{bmatrix}$$

```
temp.setZero();
temp(0, 1) = 1 / rnh;
temp(1, 0) = -1 / rmh;
temp(2, 1) = -tan(pvapre_.pos[0]) / rnh;
F.block(PHI_ID, V_ID, 3, 3)   = temp;
```

IMU零偏误差和比例因子误差，建模成一阶高斯-马尔科夫过程,：

$$\frac{-1}{T_{gs}}\mathbf{I}_{3\times3}$$

```
// IMU零偏误差和比例因子误差，建模成一阶高斯-马尔科夫过程
// imu bias error and scale error, modeled as the first-order Gauss-Markov process
F.block(BG_ID, BG_ID, 3, 3) = -1 / options_.imunoise.corr_time *
Eigen::Matrix3d::Identity();
F.block(BA_ID, BA_ID, 3, 3) = -1 / options_.imunoise.corr_time *
Eigen::Matrix3d::Identity();
F.block(SG_ID, SG_ID, 3, 3) = -1 / options_.imunoise.corr_time *
Eigen::Matrix3d::Identity();
F.block(SA_ID, SA_ID, 3, 3) = -1 / options_.imunoise.corr_time *
Eigen::Matrix3d::Identity();
```

然后构建系统噪声驱动矩阵：

$$\underset{21\times18}{\mathbf{G}_{18}} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{C}_b^n & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_b^n & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_{3\times3} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}_{3\times3} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}_{3\times3} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}_{3\times3} \end{bmatrix}$$

```
// 系统噪声驱动矩阵
// system noise driven matrix
G.block(V_ID, VRW_ID, 3, 3)    = pvapre_.att.cbn;
G.block(PHI_ID, ARW_ID, 3, 3)  = pvapre_.att.cbn;
G.block(BG_ID, BGSTD_ID, 3, 3) = Eigen::Matrix3d::Identity();
G.block(BA_ID, BASTD_ID, 3, 3) = Eigen::Matrix3d::Identity();
G.block(SG_ID, SGSTD_ID, 3, 3) = Eigen::Matrix3d::Identity();
G.block(SA_ID, SASTD_ID, 3, 3) = Eigen::Matrix3d::Identity();
```

系统传播噪声的状态转移矩阵：

$$\boldsymbol{\Phi}_{k/k-1} = \boldsymbol{I} + \boldsymbol{F}_{k-1}\Delta t_k$$

```
// 状态转移矩阵
// compute the state transition matrix
Phi.setIdentity();
Phi = Phi + F * imucur.dt;
```

系统传播噪声：

$$\boldsymbol{Q}_k = \left(\begin{matrix} \boldsymbol{\Phi}_{k/k-1}\boldsymbol{G}_{k-1}\boldsymbol{q}_{k-1}\boldsymbol{G}_{k-1}^T\boldsymbol{\Phi}_{k/k-1}^T \\ +\boldsymbol{G}_k\boldsymbol{q}_k\boldsymbol{G}_k^T \end{matrix} \right) \Delta t_k/2$$

```
// 计算系统传播噪声
// compute system propagation noise
Qd = G * Qc_ * G.transpose() * imucur.dt;
Qd = (Phi * Qd * Phi.transpose() + Qd) / 2;
```

EKF 预测传播系统协方差和系统误差状态：

$$\boldsymbol{x}_{k/k-1} = \boldsymbol{\Phi}_{k/k-1}\boldsymbol{x}_{k-1}$$
$$\boldsymbol{P}_{k/k-1} = \boldsymbol{\Phi}_{k/k-1}\boldsymbol{P}_{k-1}\boldsymbol{\Phi}_{k/k-1}^T + \boldsymbol{Q}_k$$

如果误差反馈了，那 x 应该是 0 ，无须再计算。

```
EKFPredict(Phi, Qd);
```

```
void GIEngine::EKFPredict(Eigen::MatrixXd &Phi, Eigen::MatrixXd &Qd) {

    assert(Phi.rows() == Cov_.rows());
    assert(Qd.rows() == Cov_.rows());

    // 传播系统协方差和误差状态
    // propagate system covariance and error state
    Cov_ = Phi * Cov_ * Phi.transpose() + Qd;
```

```
    dx_  = Phi * dx_;
}
```

六、GNSS 量测更新、系统状态反馈

1、gnssUpdate()：GNSS 量测更新

先将 IMU 位置 `pvacur_.pos` 转到 GNSS 天线相位中心位置 `antenna_pos`：

$$\hat{\boldsymbol{r}}_G = \hat{\boldsymbol{r}}_I + \boldsymbol{D}_R^{-1}\hat{\boldsymbol{C}}_b^n \boldsymbol{l}^b$$

```
// IMU位置转到GNSS天线相位中心位置
// convert IMU position to GNSS antenna phase center position
Eigen::Vector3d antenna_pos;
Eigen::Matrix3d Dr, Dr_inv;
Dr_inv      = Earth::DRi(pvacur_.pos);
Dr          = Earth::DR(pvacur_.pos);
antenna_pos = pvacur_.pos + Dr_inv * pvacur_.att.cbn * options_.antlever;
```

计算位置观测向量：IMU 预测天线位置减去 GNSS 观测位置，得到经纬高的差值，乘以 D_R 转为 NED 差值：

$$\boldsymbol{z}_r = \boldsymbol{D}_R\left(\hat{\boldsymbol{r}}_G - \tilde{\boldsymbol{r}}_G\right)$$

```
// GNSS位置测量新息
// compute GNSS position innovation
Eigen::MatrixXd dz;
dz = Dr * (antenna_pos - gnssdata.blh);
```

构造 GNSS 位置观测矩阵，姿态处是姿态乘以杆臂误差，位置处是单位阵：

$$\boldsymbol{H}_r = \begin{bmatrix} \boldsymbol{I}_3 & 0_3 & (\boldsymbol{C}_b^n \boldsymbol{l}^b)\times & 0_3 & 0_3 & 0_3 & 0_3 \end{bmatrix}$$

```
// 构造GNSS位置观测矩阵
// construct GNSS position measurement matrix
Eigen::MatrixXd H_gnsspos;
H_gnsspos.resize(3, Cov_.rows());
H_gnsspos.setZero();
H_gnsspos.block(0, P_ID, 3, 3)   = Eigen::Matrix3d::Identity();
H_gnsspos.block(0, PHI_ID, 3, 3) = Rotation::skewSymmetric(pvacur_.att.cbn *
options_.antlever);
```

位置观测噪声阵，就是用数据文件中读取到的 GNSS 位置标准差平方得到协方差，组成成协方差阵：

```
// 位置观测噪声阵
// construct measurement noise matrix
Eigen::MatrixXd R_gnsspos;
R_gnsspos = gnssdata.std.cwiseProduct(gnssdata.std).asDiagonal();
```

得到观测向量 **z**，观测矩阵 **H**, 观测噪声矩阵 **R** 后，调用 `EKFUpdate()`，量测更新：

```
EKFUpdate(dz, H_gnsspos, R_gnsspos);
```

最后，GNSS更新之后设置为不可用：

```
gnssdata.isvalid = false;
```

2、EKFUpdate()：EKF 更新协方差和误差状态

判断矩阵维度是否合理，不合理直接退出程序：

> 这几行 assert 可能是为了调试方便，能显示出哪两个矩阵维数不对。

```
assert(H.cols() == Cov_.rows());
assert(dz.rows() == H.rows());
assert(dz.rows() == R.rows());
assert(dz.cols() == 1);
```

计算 Kalman 滤波增益系数 K：

$$\boldsymbol{K}_k = \boldsymbol{P}_{k/k-1}\boldsymbol{H}_k^T\left(\boldsymbol{H}_k\boldsymbol{P}_{k/k-1}\boldsymbol{H}_k^T + \boldsymbol{R}_k\right)^{-1}$$

```
// 计算Kalman增益
// Compute Kalman Gain
auto temp = H * Cov_ * H.transpose() + R;
Eigen::MatrixXd K = Cov_ * H.transpose() * temp.inverse();
```

更新系统误差状态和协方差：

$$\boldsymbol{P}_k = \left(\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k\right)\boldsymbol{P}_{k/k-1}\left(\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k\right)^T + \boldsymbol{K}_k\boldsymbol{R}_k\boldsymbol{K}_k^T$$
$$\boldsymbol{K}_k = \boldsymbol{P}_{k/k-1}\boldsymbol{H}_k^T\left(\boldsymbol{H}_k\boldsymbol{P}_{k/k-1}\boldsymbol{H}_k^T + \boldsymbol{R}_k\right)^{-1}$$

```
// 更新系统误差状态和协方差
// update system error state and covariance
```

```
Eigen::MatrixXd I;
I.resizeLike(Cov_);
I.setIdentity();
I = I - K * H;
// 如果每次更新后都进行状态反馈，则更新前dx_一直为0，下式可以简化为：dx_ = K * dz;
// if state feedback is performed after every update, dx_ is always zero before the
update
// the following formula can be simplified as : dx_ = K * dz;
dx_  = dx_ + K * (dz - H * dx_);
Cov_ = I * Cov_ * I.transpose() + K * R * K.transpose();
```

3、stateFeedback()：状态反馈

想清楚卡尔曼滤波到底算的是什么，考虑到底是加还是减。零偏、比例因子残差是加、速度位置残差是减，反馈之后误差状态置 0：

- 位置反馈要乘以 `DRi()`，因为估计的位置增量是 ENU，要转为 LLH 增量。
- 姿态反馈首先要把算出的等效旋转矢量增量转为四元数，然后左乘这个四元数。

```
void GIEngine::stateFeedback() {

    Eigen::Vector3d vectemp;

    // 位置误差反馈
    // posisiton error feedback
    Eigen::Vector3d delta_r = dx_.block(P_ID, 0, 3, 1);
    Eigen::Matrix3d Dr_inv  = Earth::DRi(pvacur_.pos);
    pvacur_.pos -= Dr_inv * delta_r;

    // 速度误差反馈
    // velocity error feedback
    vectemp = dx_.block(V_ID, 0, 3, 1);
    pvacur_.vel -= vectemp;

    // 姿态误差反馈
    // attitude error feedback
    vectemp               = dx_.block(PHI_ID, 0, 3, 1);
    Eigen::Quaterniond qpn = Rotation::rotvec2quaternion(vectemp);
    pvacur_.att.qbn       = qpn * pvacur_.att.qbn;
    pvacur_.att.cbn       = Rotation::quaternion2matrix(pvacur_.att.qbn);
    pvacur_.att.euler     = Rotation::matrix2euler(pvacur_.att.cbn);

    // IMU零偏误差反馈
    // IMU bias error feedback
    vectemp = dx_.block(BG_ID, 0, 3, 1);
    imuerror_.gyrbias += vectemp;
    vectemp = dx_.block(BA_ID, 0, 3, 1);
    imuerror_.accbias += vectemp;

    // IMU比例因子误差反馈
    // IMU sacle error feedback
    vectemp = dx_.block(SG_ID, 0, 3, 1);
    imuerror_.gyrscale += vectemp;
    vectemp = dx_.block(SA_ID, 0, 3, 1);
    imuerror_.accscale += vectemp;
```

```
    // 误差状态反馈到系统状态后,将误差状态清零
    // set 'dx' to zero after feedback error state to system state
    dx_.setZero();
}
```

七、KF-GINS常见问题

> 复制自 PPT

KF-GINS能够达到怎么样的定位精度?

组合导航算法精度**更受IMU等级、以及测试时GNSS定位质量**影响。组合导航算法相对成熟，对于同样的设备只要算法正确实现，算法几乎不会对定位精度产生较大影响。

初始导航状态和初始导航状态标准差如何给定?

- 初始位置(和速度)可由**GNSS给定**，初始姿态(和速度)可从**参考结果中获取**;
- 位置速度标准差可由GNSS给定，姿态标准差可给经验值，车载领域一般横滚俯仰小，航向大一些

IMU数据输入到程序之前，需要扣除重力加速度吗?

不需要，**惯性导航算法中补偿了重力加速度**

INS机械编排中旋转效应等补偿项，对于低端IMU是否需要补偿?

低端IMU测量噪声更大，**简化的IMU积分算法对低端IMU精度产生的影响较小**

组合导航中GNSS信号丢失期间进行纯惯导解算，这时IMU误差项可以补偿吗?

GNSS丢失期间IMU误差项不更新，但是可以**利用之前估计的IMU误差项继续补偿**

IMU数据，如何从速率形式转到增量形式?

一般采用更高频率速率数据积分得到增量数据，参考：新手入门系列4——MEMS IMU**原始数据采集和时间同步**的那些坑(i2Nav网站)

IMU零偏和比例因子建模时相关时间如何给定?

建模为一阶高斯-马尔可夫过程，实际中一般**根据经验设置相关时间，MEMS设置1h**

GNSS/INS组合导航中是否需要考虑惯性系和车体系的转换?

不需要，GNSS/INS组合导航不受载体限制，不需要考虑车体系

初始化拓展

- 动态初始对准，GNSS位置差分速度(或者GNSS直接输出速度)，位置差分计算初始航向
- 快速航向初始化 ，轨迹匹配方法快速获取准确初始航向
- 静态粗对准，适用于高精度惯导，双矢量法找到初始姿态

观测信息拓展

- 直接构建观测向量、观测模型和噪声矩阵，调用EKFUpdate更新和stateFeedback反馈
- GNSS速度观测信息、NHC约束信息(对于车载)、零速信息修正

状态信息拓展

- 如果需要增广系统状态(如里程计增广比例因子[2])，则修改RANK(NOISERANK)，添加StateID, NoiseID
- 协方差、状态转移矩阵、观测信息对应修改；添加观测信息，进行更新反馈