P1:

1、 $\mathbb{E}[X]=0, \mathbb{E}[XX^T]=\sigma^2 I$

$$\mathbb{E}[RX]=\mathbb{E}[R]\cdot\mathbb{E}[X]=\mathbb{E}[X]$$
$$=0.$$

$$\mathbb{E}[(RX-\mathbb{E}[X])(RX-\mathbb{E}[X])^T]$$
$$=\mathbb{E}[RX\cdot X^TR]$$
$$=\mathbb{E}[R]\mathbb{E}[XX^T]\mathbb{E}[R]$$
$$=\mathbb{E}[XX^T]$$
$$=\sigma^2 I$$

P2:

2. $$\begin{cases} \delta p_{k+1} = \delta p_k + \delta v_k \Delta t \\ \delta v_{k+1} = \delta v_k + (-R_k(\tilde{a} - b_a)^\wedge \delta\theta - R_k \delta b_a + \delta g)\Delta t - \eta_v \\ \delta\theta_{k+1} = \mathrm{Exp}(-(\tilde{\omega} - b_g)\Delta t)\delta\theta_k - \delta b_g \Delta t - \eta_\theta \\ \delta b_{g\,k+1} = \delta b_{g\,k} + \eta_g \\ \delta b_{a\,k+1} = \delta b_{a\,k} + \eta_a \\ \delta g_{k+1} = \delta g \end{cases}$$

基本上就是把矩阵乘出来写：

```
    Vec18T new_dx = Vec18T::Zero();
    new_dx.template block<3,1>(0,0) = new_dx.template block<3,1>(0,0) +
new_dx.template block<3,1>(3,0) * dt;
    new_dx.template block<3,1>(3,0) = new_dx.template block<3,1>(3,0) + (-
R_.matrix() * SO3::hat(imu.acce_ - ba_) * new_dx.template block<3,1>(6,0)
                               - R_.matrix()*new_dx.template block<3,1>(12,0) +
new_dx.template block<3,1>(15,0)) * dt;
    new_dx.template block<3,1>(6,0) = SO3::exp(- (imu.gyro_ - bg_) * dt).matrix()
* new_dx.template block<3,1>(6,0) - new_dx.template block<3,1>(9,0) * dt;
    new_dx.template block<3,1>(9,0) = new_dx.template block<3,1>(9,0);
    new_dx.template block<3,1>(12,0) = new_dx.template block<3,1>(12,0);
    new_dx.template block<3,1>(15,0) = new_dx.template block<3,1>(15,0);
```

P3:

3. $$\begin{cases} \dot{p}_t = v_t \\ \dot{v}_t = R_t(\tilde{a} - b_{at} - \eta_a) + g_t \\ \dot{R}_t = R_t(\tilde{w} - b_{gt} - \eta_g)^\wedge \\ \dot{b}_{gt} = \eta_{bg} \\ \dot{b}_{at} = \eta_{ba} \\ \dot{g}_t = 0 \end{cases}$$

$$\begin{cases} p_t = p + \delta p \\ v_t = v + \delta v \\ R_t = \delta R \cdot R \\ b_{gt} = b_g + \delta b_g \\ b_{at} = b_a + \delta b_a \\ g_t = g + \delta g \end{cases}$$

$$R_t = \delta R \cdot R$$

$$\dot{R}_t = \dot{Exp(\delta\theta)} R + Exp(\delta\theta) \dot{R}$$

$$\cancel{Exp(\delta\theta)}R(\tilde{w} - b_{gt} - \eta_g)^\wedge = \cancel{Exp}(\delta\theta)\dot{\delta\theta}^\wedge R + \cancel{Exp(\delta\theta)} R(\tilde{w} - b_a)^\wedge$$

$$R(\tilde{\omega}-b_{gt}-n_g)^{\wedge}R^TR = \delta\dot{\theta}^{\wedge}R + R(\tilde{\omega}-b_g)^{\wedge}R^TR$$

$$(R(\tilde{\omega}-b_{gt}-n_g))^{\wedge}\cancel{R} = \delta\dot{\theta}^{\wedge}\cancel{R} + (R(\tilde{\omega}-b_g)^{\wedge})\cancel{R}$$

$$\delta\dot{\theta}^{\wedge} = R(\tilde{\omega}-b_{gt}-n_g)^{\wedge} - R(\tilde{\omega}-b_g)^{\wedge}$$

$$\delta\dot{\theta}^{\wedge} = R(-\delta b_g - n_g)^{\wedge}$$

$$\delta\dot{\theta} = -R\delta b_g - Rn_g.$$

$$\delta\dot{\theta} = -R\delta_{bg} - n_g$$

$$\delta\theta(t+\Delta t) = \delta\theta - R\delta b_g\Delta t - n_\theta$$

$$\dot{v}_t = R_t(\tilde{a}-b_{at}-n_a)+g_t$$

$$= \mathrm{Exp}(\delta\theta)R(\tilde{a}-b_{at}-n_a)+g+\delta g$$

$$\approx (I+\delta\theta^{\wedge})R(\tilde{a}-b_a-\delta b_a-n_a)+g+\delta g$$

$$\approx R\tilde{a} - Rb_a - R\delta b_a - Rn_a + \delta\theta^{\wedge}R\tilde{a} - \delta\theta^{\wedge}Rb_a - \cancel{\delta\theta^{\wedge}Rn_a} + g + \delta g$$

$$= R\tilde{a} - Rb_a - R\delta b_a - Rn_a - (R\tilde{a})^{\wedge}\delta\theta + (Rb_a)^{\wedge}\delta\theta + g + \delta g$$

$$\shortparallel$$

$$\dot{v}+\delta\dot{v} = R(\tilde{a}-b_a)+g+\delta\dot{v}$$

$$\delta\dot{v} = \left((R b_a)^\wedge - (R\tilde{a})^\wedge\right)\delta\theta - R\delta b_a - R\eta_a + \delta g$$

$$\delta\dot{v} = -(R\tilde{a} - R b_a)^\wedge \delta\theta - R\delta b_a - \eta_a + \delta g$$

$$\delta v(t+\Delta t) = \delta v + \left[-(R\tilde{a} - R b_a)^\wedge \delta\theta - R\delta b_a + \delta g\right]\Delta t$$

$$\begin{cases} \delta p(t+\Delta t) = \delta p + \delta v \Delta t \\ \delta v(t+\Delta t) = \delta v + \left[-(R\tilde{a} - R b_a)^\wedge \delta\theta - R\delta b_a + \delta g\right]\Delta t \\ \delta\theta(t+\Delta t) = \delta\theta - R\delta b_g \Delta t - \eta_\theta \\ \delta b_g(t+\Delta t) = \delta b_g + \eta_g \\ \delta b_a(t+\Delta t) = \delta b_a + \eta_a \\ \delta g(t+\Delta t) = \delta g \end{cases}$$

$$\sigma(\eta_v) = \Delta t \, \sigma_a(k), \quad \sigma(\eta_\theta) = \Delta t \, \sigma_g(k)$$

$$\sigma(\eta_g) = \sqrt{\Delta t} \, \sigma_{bg}, \quad \sigma(\eta_a) = \sqrt{\Delta t} \, \sigma_{bg}(k)$$

$$F = \begin{bmatrix} I & \Delta t & 0 & 0 & 0 & 0 \\ 0 & I & -[R(\tilde{a} - b_a)]^{\wedge} \Delta t & 0 & -R\Delta t & \Delta t \\ 0 & 0 & I & -R\Delta t & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & 0 & I \end{bmatrix}$$

观测: $\frac{\partial \mathrm{Log}(\mathrm{Exp}(\delta\theta)R)}{\partial \delta\theta} = J_l^{-1}R$

切空间:

$$\mathrm{Exp}(\delta\theta^+)R^+ = \mathrm{Exp}(\delta\theta^+)\mathrm{Exp}(\delta\theta_k)R_k$$

$$= \mathrm{Exp}(\delta\theta)R_k$$

$$\mathrm{Exp}(\delta\theta^+) = \mathrm{Exp}(\delta\theta)\mathrm{Exp}(-\delta\theta_k)$$

$$\delta\theta^+ \approx \delta\theta - \delta\theta_k - \frac{1}{2}\delta\theta^\wedge\delta\theta_k$$

$$= \delta\theta - \delta\theta_k + \frac{1}{2}\delta\theta_k^\wedge\delta\theta$$

$$\frac{\partial\delta\theta^+}{\partial\theta} = I + \frac{1}{2}\delta\theta_k^\wedge$$

$$\therefore J_k = \mathrm{diag}(I_3, I_3, I_3 + \frac{1}{2}\delta\theta_k^\wedge, I_3, I_3, I_3)$$

更新名义变量

```
/// 更新名义状态变量，重置error state
    void UpdateAndReset() {
        p_ += dx_.template block<3, 1>(0, 0);
        v_ += dx_.template block<3, 1>(3, 0);
        // R_ = R_ * SO3::exp(dx_.template block<3, 1>(6, 0));
        //    左绕动
        R_ = SO3::exp(dx_.template block<3, 1>(6, 0)) * R_;

        if (options_.update_bias_gyro_) {
            bg_ += dx_.template block<3, 1>(9, 0);
```

```
        }

        if (options_.update_bias_acce_) {
            ba_ += dx_.template block<3, 1>(12, 0);
        }

        g_ += dx_.template block<3, 1>(15, 0);

        ProjectCov();
        dx_.setZero();
    }
```

IMU的递推更新用左绕动，还有F矩阵用左绕动推导出来的形式

```
template <typename S>
bool ESKF<S>::Predict(const IMU& imu) {
    assert(imu.timestamp_ >= current_time_);

    double dt = imu.timestamp_ - current_time_;
    if (dt > (5 * options_.imu_dt_) || dt < 0) {
        // 时间间隔不对，可能是第一个IMU数据，没有历史信息
        LOG(INFO) << "skip this imu because dt_ = " << dt;
        current_time_ = imu.timestamp_;
        return false;
    }

    Vec18T new_dx = Vec18T::Zero();
    new_dx.template block<3,1>(0,0) = new_dx.template block<3,1>(0,0) +
new_dx.template block<3,1>(3,0) * dt;
    new_dx.template block<3,1>(3,0) = new_dx.template block<3,1>(3,0) + (-
R_.matrix() * SO3::hat(imu.acce_ - ba_) * new_dx.template block<3,1>(6,0)
                               - R_.matrix()*new_dx.template block<3,1>(12,0) +
new_dx.template block<3,1>(15,0)) * dt;
    new_dx.template block<3,1>(6,0) = SO3::exp(- (imu.gyro_ - bg_) * dt).matrix()
* new_dx.template block<3,1>(6,0) - new_dx.template block<3,1>(9,0) * dt;
    new_dx.template block<3,1>(9,0) = new_dx.template block<3,1>(9,0);
    new_dx.template block<3,1>(12,0) = new_dx.template block<3,1>(12,0);
    new_dx.template block<3,1>(15,0) = new_dx.template block<3,1>(15,0);

    // nominal state 递推
    VecT new_p = p_ + v_ * dt + 0.5 * (R_ * (imu.acce_ - ba_)) * dt * dt + 0.5 *
g_ * dt * dt;
    VecT new_v = v_ + R_ * (imu.acce_ - ba_) * dt + g_ * dt;
    // SO3 new_R = R_ * SO3::exp((imu.gyro_ - bg_) * dt);
    //   左绕动
    SO3 new_R = SO3::exp((imu.gyro_ - bg_) * dt) * R_;

    R_ = new_R;
    v_ = new_v;
    p_ = new_p;
    // 其余状态维度不变

    // error state 递推
    // 计算运动过程雅可比矩阵 F，见(3.47)
```

```
    // F实际上是稀疏矩阵，也可以不用矩阵形式进行相乘而是写成散装形式，这里为了教学方便，使用矩阵
形式
    Mat18T F = Mat18T::Identity();
  // 主对角线
    // F.template block<3, 3>(0, 3) = Mat3T::Identity() * dt;
    // p 对 v
    // F.template block<3, 3>(3, 6) = -R_.matrix() * SO3::hat(imu.acce_ - ba_) *
dt;  // v对theta
    // F.template block<3, 3>(3, 12) = -R_.matrix() * dt;
    // v 对 ba
    // F.template block<3, 3>(3, 15) = Mat3T::Identity() * dt;
     // v 对 g
    // F.template block<3, 3>(6, 6) = SO3::exp(-(imu.gyro_ - bg_) * dt).matrix();
    // theta 对 theta
    // F.template block<3, 3>(6, 9) = -Mat3T::Identity() * dt;
     // theta 对 bg

    //   左绕动
    F.template block<3, 3>(0, 3) = Mat3T::Identity() * dt;
  // p 对 v
    F.template block<3, 3>(3, 6) = -SO3::hat(R_.matrix() * (imu.acce_ - ba_)) *
dt;  // v对theta
    F.template block<3, 3>(3, 12) = -R_.matrix() * dt;
  // v 对 ba
    F.template block<3, 3>(3, 15) = Mat3T::Identity() * dt;
// v 对 g
    F.template block<3, 3>(6, 6) = Mat3T::Identity();     // theta 对 theta
    F.template block<3, 3>(6, 9) = -R_.matrix() * dt;                        //
theta 对 bg



    // mean and cov prediction
    dx_ = F * dx_;  // 这行其实没必要算，dx_在重置之后应该为零，因此这步可以跳过，但F需要参
与Cov部分计算，所以保留
    cov_ = F * cov_.eval() * F.transpose() + Q_;
    current_time_ = imu.timestamp_;
    return true;
}
```

GNSS更新也用左绕动

```
template <typename S>
bool ESKF<S>::ObserveSE3(const SE3& pose, double trans_noise, double ang_noise) {
    /// 既有旋转，也有平移
    /// 观测状态变量中的p, R, H为6x18，其余为零
    Eigen::Matrix<S, 6, 18> H = Eigen::Matrix<S, 6, 18>::Zero();
    H.template block<3, 3>(0, 0) = Mat3T::Identity();  // P部分
    H.template block<3, 3>(3, 6) = Mat3T::Identity();  // R部分（3.66)

    // 卡尔曼增益和更新过程
    Vec6d noise_vec;
    noise_vec << trans_noise, trans_noise, trans_noise, ang_noise, ang_noise,
ang_noise;

    Mat6d V = noise_vec.asDiagonal();
```

```
    Eigen::Matrix<S, 18, 6> K = cov_ * H.transpose() * (H * cov_ * H.transpose()
+ V).inverse();

    // 更新x和cov
    Vec6d innov = Vec6d::Zero();
    innov.template head<3>() = (pose.translation() - p_);          // 平移部分
    // innov.template tail<3>() = (R_.inverse() * pose.so3()).log();  // 旋转部分
(3.67)
    // 左绕动
    innov.template tail<3>() = (pose.so3() * R_.inverse()).log();  // 旋转部分
(3.67)

    dx_ = K * innov;
    cov_ = (Mat18T::Identity() - K * H) * cov_;

    UpdateAndReset();
    return true;
}
```

方差矩阵的投影

```
/// 对P阵进行投影，参考式(3.63)
    void ProjectCov() {
        Mat18T J = Mat18T::Identity();
        // J.template block<3, 3>(6, 6) = Mat3T::Identity() - 0.5 *
SO3::hat(dx_.template block<3, 1>(6, 0));
        //   左绕动
        J.template block<3, 3>(6, 6) = Mat3T::Identity() + 0.5 *
SO3::hat(dx_.template block<3, 1>(6, 0));
        cov_ = J * cov_ * J.transpose();
    }
```

结果和原来一样：