雅克比矩阵推导

1. 线特征残差

$$d_{\xi} = \frac{|(\tilde{p_i} - p_b) \times (\tilde{p_i} - p_a)|}{|p_a - p_b|}$$

令

$$X = \frac{(\tilde{p_i} - p_b) \times (\tilde{p_i} - p_a)}{|p_a - p_b|}$$

则有

$$J_{\xi} = \frac{\partial d_{\xi}}{\partial T} = \frac{\partial |X|}{\partial T} = \frac{\partial |X|}{\partial X} \frac{\partial X}{\partial T} = \frac{X}{|X|} \frac{\partial X}{\partial T}$$

ppt中已经对 $\frac{\partial X}{\partial T}$ 做了说明，这里整理下最终结果

$$J_{\xi} = \frac{X}{|X|} \frac{(p_a - p_b)^{\wedge}}{|p_a - p_b|} \begin{bmatrix} -(RP_i + t)^{\wedge} \\ I_{3\times 3} \end{bmatrix}_{3\times 6}$$

2. 面特征残差

$$d_H = \left| (\tilde{p_i} - p_j) \cdot \frac{(p_l - p_j) \times (p_m - p_j)}{|(p_l - p_j) \times (p_m - p_j)|} \right|$$

令

$$X = (\tilde{p_i} - p_j) \cdot \frac{(p_l - p_j) \times (p_m - p_j)}{|(p_l - p_j) \times (p_m - p_j)|}$$

则有

$$J_H = \frac{\partial d_H}{\partial T} = \frac{\partial |X|}{\partial X} \frac{\partial X}{\partial T} = \frac{X}{|X|} \frac{\partial X}{\partial \tilde{p_i}} \frac{\partial \tilde{p_i}}{\partial T}$$

ppt中已经对 $\frac{\partial X}{\partial T}$ 做了说明，这里整理下最终结果

$$J_H = \frac{X}{|X|} \frac{(p_l - p_j) \times (p_m - p_j)}{|(p_l - p_j) \times (p_m - p_j)|} \begin{bmatrix} -(RP_i + t)^{\wedge} \\ I_{3\times 3} \end{bmatrix}_{3\times 6}$$

以上推出是1X6的矩阵，在floam ceres中是1X7,因为旋转的3个自由度用四元数表示，就是4个自由度

编码实现

需要求出 $\frac{\partial \tilde{p_i}}{\partial q}$ 的矩阵，老师说可以使用对李代数求导结果 $-(R_{pi})^{\wedge}$，尝试之后不能正常建图。使用对四元素求导(https://www.cnblogs.com/JingeTU/p/11707557.html)公式，代码如下
点线：

```
class LidarEdgeFactor : public ceres::SizedCostFunction<1, 4, 3> {
  LidarEdgeFactor(Eigen::Vector3d curr_point_, Eigen::Vector3d last_point_a_,
                  Eigen::Vector3d last_point_b_, double s_)
      : curr_point(curr_point_), last_point_a(last_point_a_),
        last_point_b(last_point_b_), s(s_) {}

  virtual bool Evaluate(double const *const *parameters, double *residuals,
                        double **jacobians) const {
    Eigen::Map<const Eigen::Quaterniond> q_last_curr(parameters[0]);
    // Eigen::Quaterniond q_identity(1, 0, 0, 0);
    // q_last_curr = q_identity.slerp(s, q_last_curr);
    Eigen::Map<const Eigen::Vector3d> t_last_curr(parameters[1]);
    Eigen::Vector3d lp = q_last_curr * curr_point + t_last_curr; //   new point

    Eigen::Vector3d nu = (lp - last_point_a).cross(lp - last_point_b);
    Eigen::Vector3d de = last_point_a - last_point_b;
    residuals[0] = nu.norm() / de.norm();

    Eigen::Matrix3d J_xi_p = skew(de) / de.norm();
    Eigen::Vector4d q = q_last_curr.coeffs();

    if (jacobians != NULL) {
      if (jacobians[0] != NULL) {
        Eigen::Matrix<double, 3, 4> J_x_q;
        double qx = q(0), qy = q(1), qz = q(2), qw = q(3);
        double X = curr_point(0), Y = curr_point(1), Z = curr_point(2);
        // https://www.cnblogs.com/JingeTU/p/11707557.html
        J_x_q << qy * Y + qz * Z, qw * Z + qx * Y - 2 * qy * X,
            -qw * Y + qx * Z - 2 * qz * X, qy * Z - qz * Y,
            -qw * Z - 2 * qx * Y + qy * X, qx * X + qz * Z,
            qw * X + qy * Z - 2 * qz * Y, -qx * Z + qz * X,
            qw * Y - 2 * qx * Z + qz * X, -qw * X - 2 * qy * Z + qz * Y,
            qx * X + qy * Y, qx * Y - qy * X;
        // https://zhuanlan.zhihu.com/p/131342530
        // J_x_q << 2 * X * qx + 2 * Y * qy + 2 * Z * qz,
        //     -2 * X * qy + 2 * Y * qx + 2 * Z * qw,
        //     -2 * X * qz - 2 * Y * qw + 2 * Z * qx,
        //     2 * X * qw - 2 * Y * qz + 2 * Z * qy,
        //     2 * X * qy - 2 * Y * qx - 2 * Z * qw,
        //     2 * X * qx + 2 * Y * qy + 2 * Z * qz,
        //     2 * X * qw - 2 * Y * qz + 2 * Z * qy,
        //     2 * X * qz + 2 * Y * qw - 2 * Z * qx,
        //     2 * X * qz + 2 * Y * qw - 2 * Z * qx,
        //     -2 * X * qw + 2 * Y * qz - 2 * Z * qy,
        //     2 * X * qx + 2 * Y * qy + 2 * Z * qz,
        //     -2 * X * qy + 2 * Y * qx + 2 * Z * qw;
        Eigen::Map<Eigen::Matrix<double, 1, 4, Eigen::RowMajor>> J_so3_q(
            jacobians[0]);
        J_so3_q.setZero();
        J_so3_q.block<1, 4>(0, 0) =
```

```
            -nu.transpose() / nu.norm() * J_xi_p * J_x_q * 2;
      }
      if (jacobians[1] != NULL) {
        Eigen::Map<Eigen::Matrix<double, 1, 3, Eigen::RowMajor>> J_so3_t(
            jacobians[1]);
        J_so3_t.setZero();
        J_so3_t.block<1, 3>(0, 0) = -nu.transpose() / nu.norm() * J_xi_p;
      }
    }

    return true;
  };

public:
  static ceres::CostFunction *Create(const Eigen::Vector3d curr_point_,
                                     const Eigen::Vector3d last_point_a_,
                                     const Eigen::Vector3d last_point_b_,
                                     const double s_) {
    return new LidarEdgeFactor(curr_point_, last_point_a_, last_point_b_, s_);
  }

  Eigen::Vector3d curr_point, last_point_a, last_point_b;
  double s;
}
```

点面

```
class LidarPlaneFactor : public ceres::SizedCostFunction<1, 4, 3> {
  LidarPlaneFactor(Eigen::Vector3d curr_point_, Eigen::Vector3d last_point_j_,
                   Eigen::Vector3d last_point_l_, Eigen::Vector3d last_point_m_,
                   double s_)
      : curr_point(curr_point_), last_point_j(last_point_j_),
        last_point_l(last_point_l_), last_point_m(last_point_m_), s(s_) {
    ljm_norm = (last_point_j - last_point_l).cross(last_point_j - last_point_m);
    ljm_norm.normalize();
  }

  virtual bool Evaluate(double const *const *parameters, double *residuals,
                        double **jacobians) const {
    Eigen::Map<const Eigen::Quaterniond> q_last_curr(parameters[0]);
    Eigen::Map<const Eigen::Vector3d> t_last_curr(parameters[1]);
    Eigen::Vector3d lp = q_last_curr * curr_point + t_last_curr; //   new point

    residuals[0] = (lp - last_point_j).dot(ljm_norm);

    Eigen::Vector4d q = q_last_curr.coeffs();

    if (jacobians != NULL) {
      if (jacobians[0] != NULL) {
        Eigen::Matrix<double, 3, 4> J_x_q;
        double qx = q(0), qy = q(1), qz = q(2), qw = q(3);
        double X = curr_point(0), Y = curr_point(1), Z = curr_point(2);

        J_x_q << qy * Y + qz * Z, qw * Z + qx * Y - 2 * qy * X,
            -qw * Y + qx * Z - 2 * qz * X, qy * Z - qz * Y,
            -qw * Z - 2 * qx * Y + qy * X, qx * X + qz * Z,
            qw * X + qy * Z - 2 * qz * Y, -qx * Z + qz * X,
            qw * Y - 2 * qx * Z + qz * X, -qw * X - 2 * qy * Z + qz * Y,
            qx * X + qy * Y, qx * Y - qy * X;

        // J_x_q << 2 * X * qx + 2 * Y * qy + 2 * Z * qz,
        //     -2 * X * qy + 2 * Y * qx + 2 * Z * qw,
        //     -2 * X * qz - 2 * Y * qw + 2 * Z * qx,
        //     2 * X * qw - 2 * Y * qz + 2 * Z * qy,
        //     2 * X * qy - 2 * Y * qx - 2 * Z * qw,
        //     2 * X * qx + 2 * Y * qy + 2 * Z * qz,
        //     2 * X * qw - 2 * Y * qz + 2 * Z * qy,
        //     2 * X * qz + 2 * Y * qw - 2 * Z * qx,
        //     2 * X * qz + 2 * Y * qw - 2 * Z * qx,
        //     -2 * X * qw + 2 * Y * qz - 2 * Z * qy,
        //     2 * X * qx + 2 * Y * qy + 2 * Z * qz,
        //     -2 * X * qy + 2 * Y * qx + 2 * Z * qw;
        Eigen::Map<Eigen::Matrix<double, 1, 4, Eigen::RowMajor>> J_so3_q(
            jacobians[0]);
        J_so3_q.setZero();
        J_so3_q.block<1, 4>(0, 0) = ljm_norm.transpose() * J_x_q * 2;
      }
```

```
      if (jacobians[1] != NULL) {
        Eigen::Map<Eigen::Matrix<double, 1, 3, Eigen::RowMajor>> J_so3_t(
            jacobians[1]);
        J_so3_t.setZero();
        J_so3_t.block<1, 3>(0, 0) = ljm_norm.transpose();
      }
    }
    return true;
  }

public:
  static ceres::CostFunction *Create(const Eigen::Vector3d curr_point_,
                                     const Eigen::Vector3d last_point_j_,
                                     const Eigen::Vector3d last_point_l_,
                                     const Eigen::Vector3d last_point_m_,
                                     const double s_) {
    // return (new ceres::AutoDiffCostFunction<LidarPlaneFactor, 1, 4, 3>(
    //     new LidarPlaneFactor(curr_point_, last_point_j_, last_point_l_,
    //                          last_point_m_, s_)));

    return new LidarPlaneFactor(curr_point_, last_point_j_, last_point_l_,
                                last_point_m_, s_);
  }

  Eigen::Vector3d curr_point, last_point_j, last_point_l, last_point_m;
  Eigen::Vector3d ljm_norm;
  double s;
}
```

EVO验证

evo_ape kitti ground_truth.txt laser_odom.txt -r full --plot --plot_mode xyz
自动求导的RMSE为 18.860694
解析求导的RMSE为 30.719905
怀疑求导公式有误,使用四元数转旋转矩阵(https://zhuanlan.zhihu.com/p/131342530),再对四元数偏导,结果RMSE还是30多,应该还有问题，如果有同学做出来，希望能参考下。