```
android.py class AndroidRawGnss(Navdata)

tx_rx_gnss_ns = self["TimeNanos"] - self["FullBiasNanos",0] +
self["TimeOffsetNanos"] - self["BiasNanos",0]

/*
 tx_rx_gnss_ns 得到的是接收机的校正后接收时间，考虑了接收机的时间偏差和时间偏移
- self["TimeNanos"] : 这是接收机的时间戳，表示接收机在接收到卫星信号时的纳秒时间。
- self["FullBiasNanos",0] : 这是接收机的完整时间偏差（以纳秒为单位），通常用于校正接收机的时间偏差。这里的 ,0 表示从数据集中获取第一个值。
- self["TimeOffsetNanos"] : 这是接收机的时间偏移量（以纳秒为单位），用于进一步校正接收机的时间。
- self["BiasNanos",0] : 这是接收机的时间偏差（以纳秒为单位），用于校正接收机的时间。这里的 ,0 表示从数据集中获取第一个值。
*/

gps_week_nanos = np.floor(-self["FullBiasNanos"]*1e-
9/consts.WEEKSEC)*consts.WEEKSEC*1E9
/*
gps周的概念：GPS时间是从1980年1月6日开始计时的，时间以周和秒为单位进行计量
该式计算出当前周的起始时间gps_week_nanos
*/
# gps constellation
tx_rx_gps_secs = (tx_rx_gnss_ns - gps_week_nanos)*1E-9
t_rx_secs = np.where(self["gnss_id"]=="gps",
                             tx_rx_gps_secs,
                             t_rx_secs)
/*
不同的卫星比如GPS、北斗、伽利略等使用不同的时间基准和信号特性，这导致了接收时间（ t_rx_secs ）的计算方式不同
减去gps_week_nano，可以得到当前时间在本周内的纳秒数t_rx_secs，用于统一计算
*/

t_tx_secs = self["ReceivedSvTimeNanos"]*1E-9
/*
接收到的卫星信号时间t_tx_secs(GPS周定义下的时间)
*/

self["raw_pr_m"] = (t_rx_secs - t_tx_secs)*consts.C
/*
原始伪距raw_pr_m"
*/

clk.py
b_sv_m.append(float(timelist_val[9]) * C)
/*
从星历数据中获得了卫星时钟偏差b_sv_m，以米为单位
*/
android.py class AndroidRawGnss(Navdata)
for _, _, subset in loop_time(self,"gps_millis", delta_t_decimals=-2):
    subset["corr_pr_m"] = subset["raw_pr_m"] + subset["b_sv_m"]
    first_timestamp = solve_wls(subset)
    if not np.isnan(first_timestamp["b_rx_wls_m"]):
                    break
```

```
self["raw_pr_m"] += first_timestamp["b_rx_wls_m"]
/*
subset["corr_pr_m"]对应冯海桐同学的真实（修正）伪距
 first_timestamp操作是移除接收机的初始时钟偏差b_rx_wls_m，对每个时间计算接收机偏差，当不为0
时结束计算，对raw_pr_m进行修正，去除初始时钟偏差
*/

fde_example.py
full_states["corr_pr_m"] = full_states["raw_pr_m"] + full_states["b_sv_m"]
/*
在直接运行的python文件中对corr_pr_m进行了计算 ：原始伪距 + 卫星时钟偏差（星历中得到）
*/



/*
上述内容是解读fde_greedy_residual函数具体流程的基础，下面详细介绍fde_greedy_residual流程
*/

fde.py  fde_greedy_residual
/*
fde的核心是solve_residuals()，_residual_chi_square()，_residual_exclude()这三个函数，
下面详细讲解这三个函数的具体流程
*/

solve_residuals()
pos_sv_m = measurement_subset[["x_sv_m","y_sv_m","z_sv_m"]].T //卫星位置
rx_pos = receiver_state[[rx_idxs["x_rx*_m"][0],
                                  rx_idxs["y_rx*_m"][0],
                                  rx_idxs["z_rx*_m"][0]],
                                  rx_t_idx].reshape(1,-1)
pos_rx_m = np.tile(rx_pos, (num_svs, 1)) //接收机位置
gt_pr_m = np.linalg.norm(pos_rx_m - pos_sv_m, axis = 1,
                                  keepdims = True) \
                + receiver_state[rx_idxs["b_rx*_m"][0],rx_t_idx]
// 实验计算伪距gt_pr_m 接收机和卫星距离加上接收机时钟偏差b_rx*_m
 residuals_epoch = corr_pr_m - gt_pr_m
        residuals += residuals_epoch.reshape(-1).tolist()|
//计算伪距残差residuals  corr_pr_m输入的真实伪距（不考虑星历误差等，可以认为是真实伪距）
gt_pr_m 理论计算出的真实伪距

_residual_chi_square()
geo_matrix =
(receiver_state[["x_rx_wls_m","y_rx_wls_m","z_rx_wls_m"]].reshape(-1,1) \
               -  navdata[["x_sv_m","y_sv_m","z_sv_m"]]).T
    geo_matrix /= np.linalg.norm(geo_matrix,axis=0)

    chi_square = residuals.T @ (weights - weights @ geo_matrix \
               @ np.linalg.pinv(geo_matrix.T @ weights @ geo_matrix) \
               @ geo_matrix.T @ weights ) @ residuals
/*
生成了一个考虑距离，权重等因素的卡方估计值来进行筛选
geo_matrix 表示卫星与接收机间的距离
weights 表示权重矩阵
residuals表示残差矩阵
*/
```

```
normalized_residual = _residual_exclude(navdata_subset,receiver_state) //计算标准化残差
                fault_idx = np.argsort(normalized_residual)[-1]//记录标准化残差最大的卫星编号
    navdata_subset.remove(cols=[fault_idx], inplace=True)//移除该卫星
_residual_exclude()
    geo_matrix /= np.linalg.norm(geo_matrix,axis=0)

    # calculate normalized residual
    x_tilde = np.linalg.pinv(geo_matrix.T @ weights @ geo_matrix) \
            @ geo_matrix.T @ weights @ residuals

    normalized_residual = np.divide(np.multiply(np.diag(weights).reshape(-1,1),
                                                (residuals - geo_matrix @
x_tilde)**2),
                                    np.diag(1 - weights @ geo_matrix @ \
                                     np.linalg.pinv(geo_matrix.T @ weights @
geo_matrix) \
                                     @ geo_matrix.T).reshape(-1,1))
/*
计算标准化残差的过程并没有弄懂，但是其使用的每个元素都是之前介绍过的元素
*/

snapshot.py
```

梳理一下wls算法的流程和细节

```
snapshot.py
solve_wls()
 pos_sv_m = measurement_subset[["x_sv_m","y_sv_m","z_sv_m"]].T
 position = np.vstack((
                                  receiver_state.where("gps_millis",
                                  timestamp)[[rx_idxs["x_rx*_m"][0],
                                              rx_idxs["y_rx*_m"][0],
                                              rx_idxs["z_rx*_m"][0]]
                                              ,0].reshape(-1,1),
                                             position[3])) # clock bias

 position = wls(position, pos_sv_m, corr_pr_m, weights,
                           only_bias, tol, max_count, sv_rx_time=sv_rx_time)
wls()
 while np.linalg.norm(pos_x_delta) > tol: //直到位置变化量小于阈值
    pos_x_delta = np.linalg.pinv(geometry_matrix.T @ weight_matrix @
geometry_matrix) \
                        @ geometry_matrix.T @ weight_matrix @ pr_delta

   pr_delta = corr_pr_m - gt_r_m - rx_est_m[3,0]

    rx_est_m += pos_x_delta //更新接收机位置和接收机误差
/*
 从星历数据中估计接收机的初始位置x_rx*_m，y_rx*_m，z_rx*_m，接收机时钟偏差初始值为0
 pr_delta对应冯海桐同学解算结果中的L，
```

```
    rx_est_m[3,0] 在代码中表示接收机时钟偏差的估计值。 rx_est_m 是一个包含接收机位置和时钟偏差的数组，其中：
- rx_est_m[0,0] 表示接收机在 ECEF 坐标系中的 x 坐标。
- rx_est_m[1,0] 表示接收机在 ECEF 坐标系中的 y 坐标。
- rx_est_m[2,0] 表示接收机在 ECEF 坐标系中的 z 坐标。
- rx_est_m[3,0] 表示接收机的时钟偏差，以米为单位
问题：代码中显示L = ρ˜i – ρi0 - b(代码中并未考虑电离层等误差)，和冯海桐同学的推导结果不同
解算结果中的A就是简单地代入卫星位置和接收机位置计算
*/
```

x_0, y_0, z_0接收机位置$x_i. y_i, z_i$卫星位置

$\widetilde{\rho}^i$真实伪距ρ_0^i测量伪距V^s卫星时钟偏差

W权重矩阵，默认值为全一样

t_r修正后的接收机时间t_s卫星发射时间

$$\rho = C(t_r - t_s)$$

$$\widetilde{\rho}^i = \rho - cV^s$$

$$\rho_0^i = \sqrt{(x_0 - x^i)^2 + (y_0 - y^i)^2 + (z_0 - z^i)^2}$$

$$y = \widetilde{\rho}^i - \rho_0^i$$

$x_r, y_r. z_r$接收机位置

x^1, y^1, z^1卫星位置

$$X_r = \begin{bmatrix} x_r & y_r & z_r \\ x_r & y_r & z_r \\ \cdots & \cdots & \cdots \end{bmatrix}^T,$$

$$X_s = \begin{bmatrix} x^1 & y^1 & z^1 \\ x^2 & y^2 & z^2 \\ \cdots & \cdots & \cdots \end{bmatrix}$$

$$G = (X_r - X_s)^T$$

$$\chi^2 = y^T(W - WG(G^TWG)G^TW)y$$

- 如果 G 的列线性相关（几何分布不好），则投影矩阵会退化，导致卡方统计量不敏感于某些方向的异常，影响故障检测能力。
- 卫星分布越好（空间发散），G越“满秩”，检测能力越强。

另一种加权最小二乘法的思路

最小化观测值y与模型预测G之间的残差平方和 $\min_x |y - Gx|^2$

加权最小二乘法 $\min_x (y - Gx)^T W (y - Gx)$

对上式对x求导并令导数为零，得到方程$G^TWGx = G^TWy$

解方程得到$x = (G^TWG)^{-1}G^TWy$

w_i, g_i是W, G的第i列

$$r_i = \frac{w_i(y - g_i x)^2}{1 - {g_i}^T w_i (G^T W G)^{-1} g_i}$$