OrbisNet Sigma - 厳密網平均（Network Adjustment）仕様書

2026年５月　シーラス＆株式会社リットー

概要

厳密網平均（Rigorous Network Adjustment） は、複数のGNSS観測セッションから得られた基線観測値を統合し、観測誤差を最小化しながら一貫性のある座標を決定する手法です。

OrbisNet Sigma では、**最小二乗法（Least Squares Method）** による厳密網平均を実装しており、複数点・複数セッションの観測に対応しています。

厳密網平均の目的

個別点計算（Simple Solution）との違い

個別点計算（Simple Solution）
• 各点を独立に計算
• 点間の幾何学的制約を考慮しない
• 観測誤差が各点に直接影響
• セッション間の不整合が残る可能性

↓ vs ↑

厳密網平均（Rigorous Network Adjustment）
• 全点の観測値を同時に処理
• 点間の距離制約（基線ベクトル）を活用
• 観測誤差を最小化しながら一貫性を維持
• セッション間の矛盾を自動的に解消
• 各点の信頼区間（信頼度）を算出

厳密網平均の理論

1.　観測方程式（Observation Equations）

基線観測値（ΔX, ΔY, ΔZ）と座標の関係を表す：

観測方程式の標準形：

l + v = h(x)

ここで：

l　= 観測値（基線ベクトルΔX, ΔY, ΔZ）
v　= 残差（補正量、求めるべき値）
h(x) = 座標パラメータの関数
x　= 未知パラメータ（点の座標XYZ）

具体的には：

観測点 i と基準点 の基線観測値：

```
ΔX_obs = X_i - X_ref + v_ΔX
ΔY_obs = Y_i - Y_ref + v_ΔY
ΔZ_obs = Z_i - Z_ref + v_ΔZ
```

整理すると：

```
X_i = ΔX_obs + X_ref - v_ΔX
Y_i = ΔY_obs + Y_ref - v_ΔY
Z_i = ΔZ_obs + Z_ref - v_ΔZ
```

2. 重み行列（Weight Matrix）

観測値の精度に基づいて重みを設定：

重みの計算：

$w_i = 1 / (\sigma_i)^2$

ここで：

w_i ＝観測値 i の重み
σ_i ＝観測値 i の標準偏差

具体例：

観測セッション1：

$\sigma_{\Delta X} = 0.002$ m → $w_{\Delta X} = 1 / (0.002)^2 = 250{,}000$
$\sigma_{\Delta Y} = 0.003$ m → $w_{\Delta Y} = 1 / (0.003)^2 = 111{,}111$
$\sigma_{\Delta Z} = 0.005$ m → $w_{\Delta Z} = 1 / (0.005)^2 = 40{,}000$

観測セッション2：

$\sigma_{\Delta X} = 0.001$ m → $w_{\Delta X} = 1 / (0.001)^2 = 1{,}000{,}000$
$\sigma_{\Delta Y} = 0.002$ m → $w_{\Delta Y} = 1 / (0.002)^2 = 250{,}000$
$\sigma_{\Delta Z} = 0.003$ m → $w_{\Delta Z} = 1 / (0.003)^2 = 111{,}111$

結果：セッション2の重みが大きい（より精度が高い）

$$W = \begin{pmatrix} w_1 & 0 & 0 & 0 & 0 & \cdots \\ 0 & w_2 & 0 & 0 & 0 & \cdots \\ 0 & 0 & w_3 & 0 & 0 & \cdots \\ 0 & 0 & 0 & w_4 & 0 & \cdots \\ 0 & 0 & 0 & 0 & w_5 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \cdots \end{pmatrix}$$

3. 正規方程式（Normal Equations）

観測方程式を線形化し、最小二乗法で解く：

線形化：
$l + v = h(x_0) + (\partial h / \partial x)|_{x_0} \cdot \Delta x$

ここで：

$h(x_0)$ ＝初期値での関数値
$(\partial h / \partial x)|_{x_0}$ ＝ヤコビ行列（A行列）
Δx ＝初期値からの補正量
$l + v$ ＝観測値＋残差

正規方程式：

$(A^T \cdot W \cdot A) \cdot \Delta x = A^T \cdot W \cdot (l - h(x_0))$

または簡潔に：

$N \cdot \Delta x = u$

ここで：

$N = A^T \cdot W \cdot A$ （正規行列）
$u = A^T \cdot W \cdot l$ （定数項ベクトル）

解：

$\Delta x = N^{-1} \cdot u = (A^T \cdot W \cdot A)^{-1} \cdot (A^T \cdot W \cdot l)$

最終的な座標：
$x = x_0 + \Delta x$

4. 精度評価（Accuracy Assessment）

① 事後標準偏差（Posterior Standard Deviation）

事後標準偏差（参照単位重み当たりの標準偏差）：

$$\sigma_0^2 = (v^T \cdot W \cdot v) / (m - n)$$

ここで：

- v ＝残差ベクトル
- W ＝重み行列
- m ＝観測数（基線成分数 × セッション数）
- n ＝未知パラメータ数（座標成分数）
- $m - n$ ＝自由度（Degrees of Freedom）

例：

- 観測数 m ＝3点 × 3成分 × 2セッション ＝18
- 未知数 n ＝3点 × 3成分 ＝9（基準点は固定）
- 自由度 ＝18 - 9 ＝9

② 座標の標準偏差（Coordinate Standard Deviation）

座標パラメータの分散共分散行列：

$$\Sigma_x = \sigma_0^2 \cdot (A^T \cdot W \cdot A)^{-1} = \sigma_0^2 \cdot N^{-1}$$

各座標成分の標準偏差：

$$\sigma_{X_i} = \sigma_0 \cdot \sqrt{(N^{-1}_{X_i,X_i})}$$
$$\sigma_{Y_i} = \sigma_0 \cdot \sqrt{(N^{-1}_{Y_i,Y_i})}$$
$$\sigma_{Z_i} = \sigma_0 \cdot \sqrt{(N^{-1}_{Z_i,Z_i})}$$

③ 水平精度・鉛直精度

水平精度（Horizontal Accuracy）：

$\sigma\_H = \sqrt{(\sigma\_X^2 + \sigma\_Y^2)}$

鉛直精度（Vertical Accuracy）：

$\sigma\_V = \sigma\_Z$

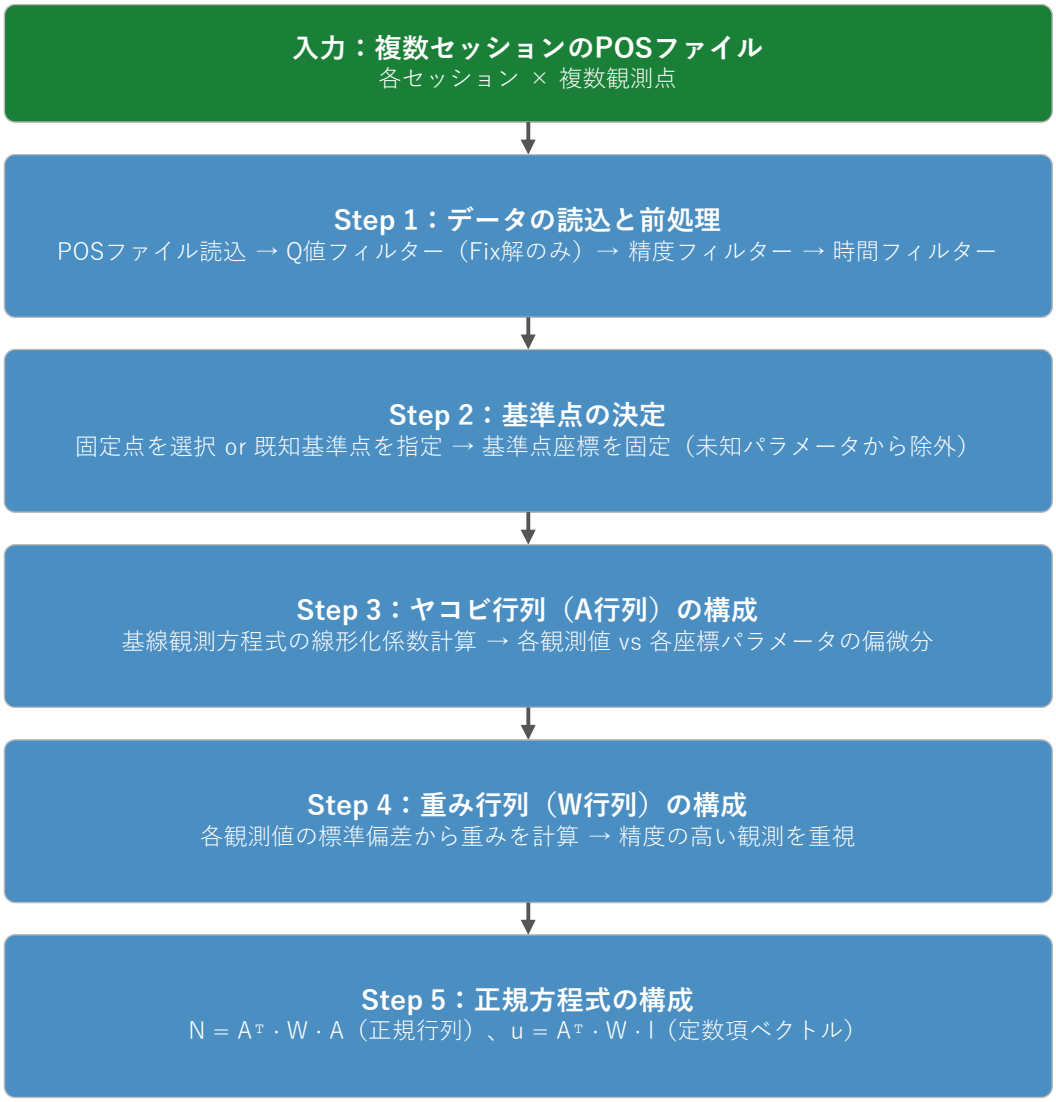
④ 95% 信頼区間（95% Confidence Interval）

95%信頼区間（2σ）：

$CI\_95\% = 1.96 \times \sigma$　（正規分布仮定）

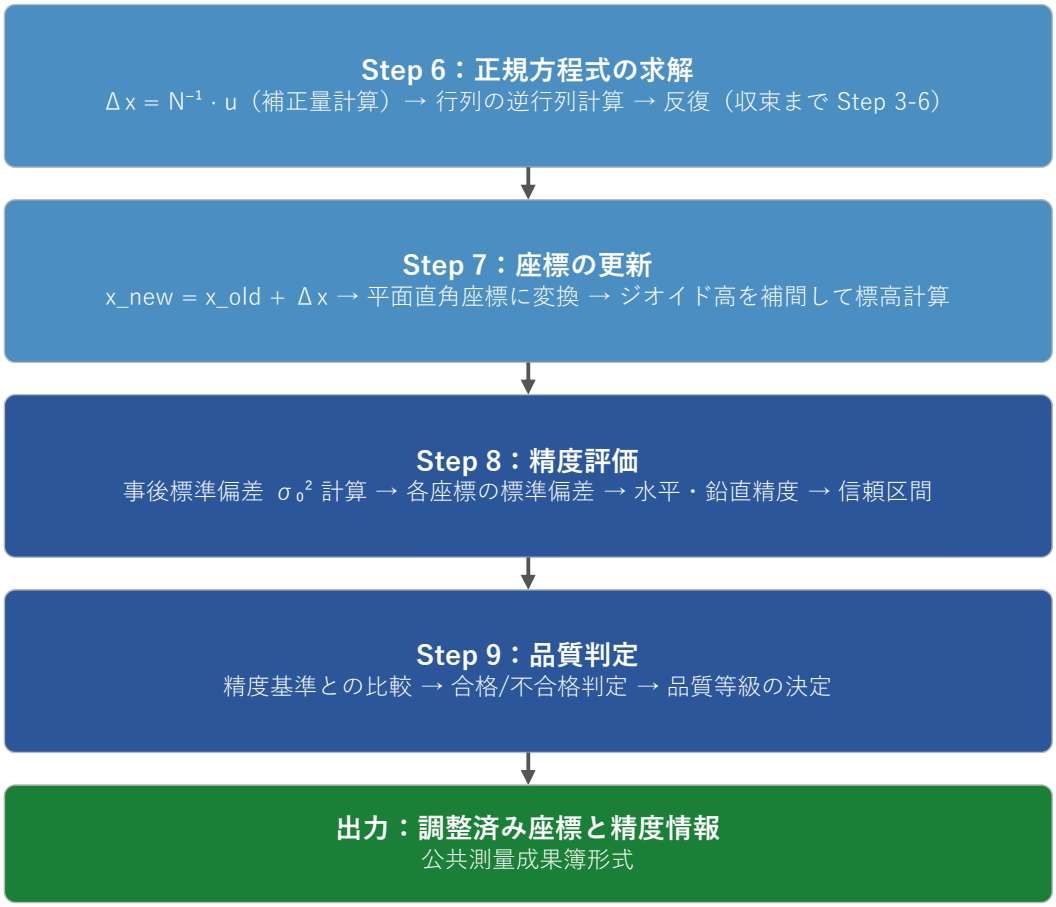
例：
$\sigma\_B = 0.0008''$　（緯度の標準偏差）
$CI\_95\% = 1.96 \times 0.0008'' = 0.001568''$

厳密網平均の処理フロー



▼ 次スライドへ続く

厳密網平均の処理フロー（続き）



反復計算の収束フロー


反復ループ（Step 3 → Step 6）
ヤコビ行列 A の構成
偏微分 ∂h/∂x を計算
重み行列 W の構成
W = diag(1/σ²)
正規方程式の構成と求解
N = AᵀWA, u = AᵀWl, Δx = N⁻¹u
座標の更新
x_new = x_old + Δx
収束判定: max(|Δx|) < ε ？
No → Step 3 に戻る / Yes → 次へ
精度評価（Step 8）
σ₀², σ_X, σ_Y, σ_Z, σ_H, CI_95%
品質判定（Step 9）
合格/不合格 → 品質等級の決定

実装の詳細

1.　ヤコビ行列（A行列）の構成

基線観測方程式：

ΔX = X_移動点 - X_基準点
ΔY = Y_移動点 - Y_基準点
ΔZ = Z_移動点 - Z_基準点

偏微分（ヤコビ行列）：

∂(ΔX)/∂X_移動点 = +1,　∂(ΔX)/∂X_基準点 = -1,　その他 = 0
∂(ΔX)/∂Y_移動点 = 0,　∂(ΔX)/∂Y_基準点 = 0
∂(ΔX)/∂Z_移動点 = 0,　∂(ΔX)/∂Z_基準点 = 0

同様に ΔY, ΔZ についても計算

具体例（観測点1, 2 と基準点の場合）：

観測値の順序：
1. ΔX_{1→ref}
2. ΔY_{1→ref}
3. ΔZ_{1→ref}
4. ΔX_{2→ref}
5. ΔY_{2→ref}
6. ΔZ_{2→ref}

パラメータの順序：
1. X_1, Y_1, Z_1, X_2, Y_2, Z_2 （基準点は固定）

ヤコビ行列 A（6×6）：

	X_1	Y_1	Z_1	X_2	Y_2	Z_2	
1:	+1	0	0	-1	0	0	（ΔX_{1→ref}）
2:	0	+1	0	0	-1	0	（ΔY_{1→ref}）
3:	0	0	+1	0	0	-1	（ΔZ_{1→ref}）
4:	+1	0	0	-1	0	0	（ΔX_{2→ref}）
5:	0	+1	0	0	-1	0	（ΔY_{2→ref}）
6:	0	0	+1	0	0	-1	（ΔZ_{2→ref}）

2. 正規行列（N行列）の構成

N = A^T ・ W ・ A

例（単純な3×3の場合）：

A^T ・ W ・ A =

$$\begin{pmatrix} w\_1 & 0 & 0 \\ 0 & w\_2 & 0 \\ 0 & 0 & w\_3 \end{pmatrix} \times \begin{pmatrix} a\_{11} & a\_{12} & a\_{13} \\ a\_{21} & a\_{22} & a\_{23} \\ a\_{31} & a\_{32} & a\_{33} \end{pmatrix}$$

結果：対称正定値行列（Symmetric Positive Definite Matrix）

3. 正規方程式の求解

Cholesky分解：

N = L · L^T

ここで L は下三角行列

メリット：
- 計算効率が良い（ガウス消去法の約1/3）
- 数値的に安定
- 公共測量の標準手法

計算手順：

1. Cholesky分解：N = L · L^T を計算
2. 前進消去：L · y = u を解く → y を求める
3. 後退代入：L^T · Δx = y を解く → Δx を求める

4. 収束判定

補正量の大きさで収束判定：

$$\max(|\Delta x_i|) < \varepsilon$$

ここで：
ε ＝収束条件（通常 1mm 以下）

例：
反復1回目：$\max(|\Delta x|) = 0.05$ m → 収束せず
反復2回目：$\max(|\Delta x|) = 0.002$ m → 収束せず
反復3回目：$\max(|\Delta x|) = 0.0001$ m → 収束！

実装コード（C#）

MultiSessionAnalyzer クラス

```
public class MultiSessionAnalyzer
{
    /// <summary>複数セッション・複数点の厳密網平均</summary>
    public static void PerformRigorousNetworkAdjustment(
        List<SessionResult> sessionResults,
        out List<AdjustedPoint> adjustedPoints,
        out AdjustmentStatistics stats)
    {
        // ★ Step 1：観測点の集約
        var observationPoints = CollectObservationPoints(sessionResults);
        int numPoints = observationPoints.Count;

        // ★ Step 2：基準点の決定
        var referencePoint = SelectReferencePoint(observationPoints);
        int numFreePoints = numPoints - 1;  // 基準点は固定


       // ★ Step 3：観測値の集約
            var observations = CollectObservations(
                sessionResults, observationPoints, referencePoint);
            int numObservations = observations.Count;

            // ★ Step 4-6：反復計算
            var adjustedCoords = PerformIterativeAdjustment(
                observationPoints, referencePoint, observations,
                out var normalMatrix, out var covarianceMatrix);

            // ★ Step 7-8：精度評価
            ComputeAccuracy(observations, adjustedCoords, referencePoint,
                            normalMatrix, out var standardDeviations);

            // ★ Step 9：品質判定
            JudgeQuality(adjustedCoords, standardDeviations,
                         out adjustedPoints, out stats);
       }



       /// <summary>Step 3：ヤコビ行列（A行列）の構成</summary>
          private static Matrix ComputeJacobianMatrix(
              List<Point> points, Point refPoint, List<Observation> observations)
          {
              int m = observations.Count;  // 観測数
              int n = (points.Count - 1) * 3;  // 未知数（基準点を除く）

              var A = new Matrix(m, n);

              int rowIdx = 0;
              foreach (var obs in observations)
              {
                  int pointIdx = points.FindIndex(p => p.Id == obs.PointId);
                  int colIdx = (pointIdx - 1) * 3;  // 基準点は0なので開始は-3（スキップ）
```

```
// 基線観測方程式の偏微分
            if (obs.Type == ObservationType.DeltaX)
            {
                A[rowIdx, colIdx + 0] = +1.0;  // ∂(ΔX)/∂X_移動点
                // ∂(ΔX)/∂X_基準点 = -1.0 は基準点なので計算しない
            }
            else if (obs.Type == ObservationType.DeltaY)
            {
                A[rowIdx, colIdx + 1] = +1.0;  // ∂(ΔY)/∂Y_移動点
            }
            else if (obs.Type == ObservationType.DeltaZ)
            {
                A[rowIdx, colIdx + 2] = +1.0;  // ∂(ΔZ)/∂Z_移動点
            }

            rowIdx++;
        }

        return A;
    }



    /// <summary>Step 4：重み行列（W行列）の構成</summary>
        private static Matrix ComputeWeightMatrix(
            List<Observation> observations)
        {
            int m = observations.Count;
            var W = Matrix.Zero(m, m);

            for (int i = 0; i < m; i++)
            {
                double sigma = observations[i].StandardDeviation;
                W[i, i] = 1.0 / (sigma * sigma);  // w_i = 1 / σ_i²
            }

            return W;
        }


    /// <summary>Step 5-6：正規方程式の構成と求解</summary>
        private static Vector PerformIterativeAdjustment(
            List<Point> points, Point refPoint, List<Observation> observations,
            out Matrix normalMatrix, out Matrix covarianceMatrix)
        {
            var currentCoords = InitializeCoordinates(points, refPoint);
            double convergenceTolerance = 0.001;  // 1 mm
            int maxIterations = 10;

            for (int iter = 0; iter < maxIterations; iter++)
            {
                // ヤコビ行列の構成
                var A = ComputeJacobianMatrix(points, refPoint, observations);

                // 重み行列の構成
                var W = ComputeWeightMatrix(observations);

                // 正規行列：N = A^T · W · A
                normalMatrix = A.Transpose() * W * A;

                // 観測方程式の矛盾項計算
                var residuals = ComputeResiduals(
                    observations, currentCoords, refPoint);
```

```
// 定数項：u = A^T · W · l
            var constantVector = A.Transpose() * W * residuals;

            // 正規方程式を求解：Δx = N^{-1} · u
            var corrections = normalMatrix.Inverse() * constantVector;

            // 収束判定
            double maxCorrection = corrections.MaxAbsoluteValue();
            if (maxCorrection < convergenceTolerance)
            {
                break;  // 収束
            }

            // 座標を更新
            currentCoords = currentCoords + corrections;
        }

        // 分散共分散行列：Σ_x = σ₀² · N^{-1}
        covarianceMatrix = normalMatrix.Inverse();

        return currentCoords;
    }


/// <summary>Step 8：精度評価（標準偏差の計算）</summary>
    private static void ComputeAccuracy(
        List<Observation> observations,
        Vector adjustedCoords, Point refPoint,
        Matrix covarianceMatrix,
        out List<CoordinateAccuracy> standardDeviations)
    {
        // ★ 残差の計算
        var residuals = ComputeResiduals(observations, adjustedCoords, refPoint);

       // ★ 事後標準偏差の計算
              double vTWv = 0;
              for (int i = 0; i < residuals.Length; i++)
              {
                  vTWv += residuals[i] * residuals[i] / (observations[i].StandardDeviation *
      observations[i].StandardDeviation);
              }

              int m = observations.Count;  // 観測数
              int n = adjustedCoords.Length;  // 未知数
              int degreesOfFreedom = m - n;

              double sigma0Squared = vTWv / degreesOfFreedom;  // 事後標準偏差の平方
```

```
// ★ 各座標の標準偏差
        standardDeviations = new List<CoordinateAccuracy>();
        for (int i = 0; i < adjustedCoords.Length; i += 3)
        {
            double sigmaX = Math.Sqrt(sigma0Squared * covarianceMatrix[i, i]);
            double sigmaY = Math.Sqrt(sigma0Squared * covarianceMatrix[i + 1, i + 1]);
            double sigmaZ = Math.Sqrt(sigma0Squared * covarianceMatrix[i + 2, i + 2]);

            double sigmaH = Math.Sqrt(sigmaX * sigmaX + sigmaY * sigmaY);  // 水平精度

            standardDeviations.Add(new CoordinateAccuracy
            {
                SigmaX = sigmaX,
                SigmaY = sigmaY,
                SigmaZ = sigmaZ,
                SigmaHorizontal = sigmaH,
                Sigma0Squared = sigma0Squared
            });
        }
    }

/// <summary>Step 9：品質判定</summary>
    private static void JudgeQuality(
        Vector adjustedCoords,
        List<CoordinateAccuracy> accuracies,
        out List<AdjustedPoint> adjustedPoints,
        out AdjustmentStatistics stats)
    {
        adjustedPoints = new List<AdjustedPoint>();

        // 精度基準（例：3級基準点測量）
        double threshold_Horizontal = 0.100;  // 100 mm
        double threshold_Vertical = 0.150;    // 150 mm

        int passCount = 0;
        double sumHorizontal = 0;
        double sumVertical = 0;
```

```
        for (int i = 0; i < adjustedCoords.Length; i += 3)
        {
            var acc = accuracies[i / 3];

            // 品質判定
            string qualityGrade = DetermineQualityGrade(acc);
            bool isPassed = acc.SigmaHorizontal <= threshold_Horizontal &&
                            acc.SigmaZ <= threshold_Vertical;

            if (isPassed)
                passCount++;

            sumHorizontal += acc.SigmaHorizontal;
            sumVertical += acc.SigmaZ;


             adjustedPoints.Add(new AdjustedPoint
             {
                 Coordinates = new Vector3(
                     adjustedCoords[i],
                     adjustedCoords[i + 1],
                     adjustedCoords[i + 2]),
                 Accuracy = acc,
                 QualityGrade = qualityGrade,
                 IsPassed = isPassed
             });
         }


         // 統計情報
         stats = new AdjustmentStatistics
         {
             TotalPoints = adjustedCoords.Length / 3,
             PassedPoints = passCount,
             AverageHorizontalAccuracy = sumHorizontal / (adjustedCoords.Length / 3),
             AverageVerticalAccuracy = sumVertical / (adjustedCoords.Length / 3),
             DegreesOfFreedom = /* m - n */,
             AdjustmentMethod = "Rigorous Network Adjustment (Least Squares)"
         };
     }
}
```

出力例（公共測量成果簿）

【精度管理】

セッション数	2
観測点数	4
平均水平精度	286.81 mm
平均垂直精度	491.69 mm
最大水平精度	327.09 mm
最大垂直精度	538.24 mm
合格点数	1 / 4

詳細成果表（測地座標）

点名	緯度(度)	経度(度)	楕円体高(m)	水平精度(mm)	鉛直精度(mm)	判定
1	-56.6976	74.6586	-262.047	0.00	0.00	合格
2	-57.6727	64.4165	-261.084	263.52	509.94	不合格
3	-47.3355	64.0985	-325.493	269.81	426.90	不合格
4	-46.8526	73.6385	54.528	327.09	538.24	不合格

標準偏差・信頼区間（95%）

点名	緯度 σ(″)	経度 σ(″)	高さ σ(mm)	95%CI_B(″)	95%CI_L(″)	95%CI_h(mm)
1	0.0000	0.0000	0.00	0.0000	0.0000	0.00
2	7.2528	8.3682	509941	14.2155	16.4016	999485
3	7.9075	5.4433	426901	15.4986	10.6689	836726
4	9.4988	6.8059	538240	18.6176	13.3395	1054951

品質判定基準

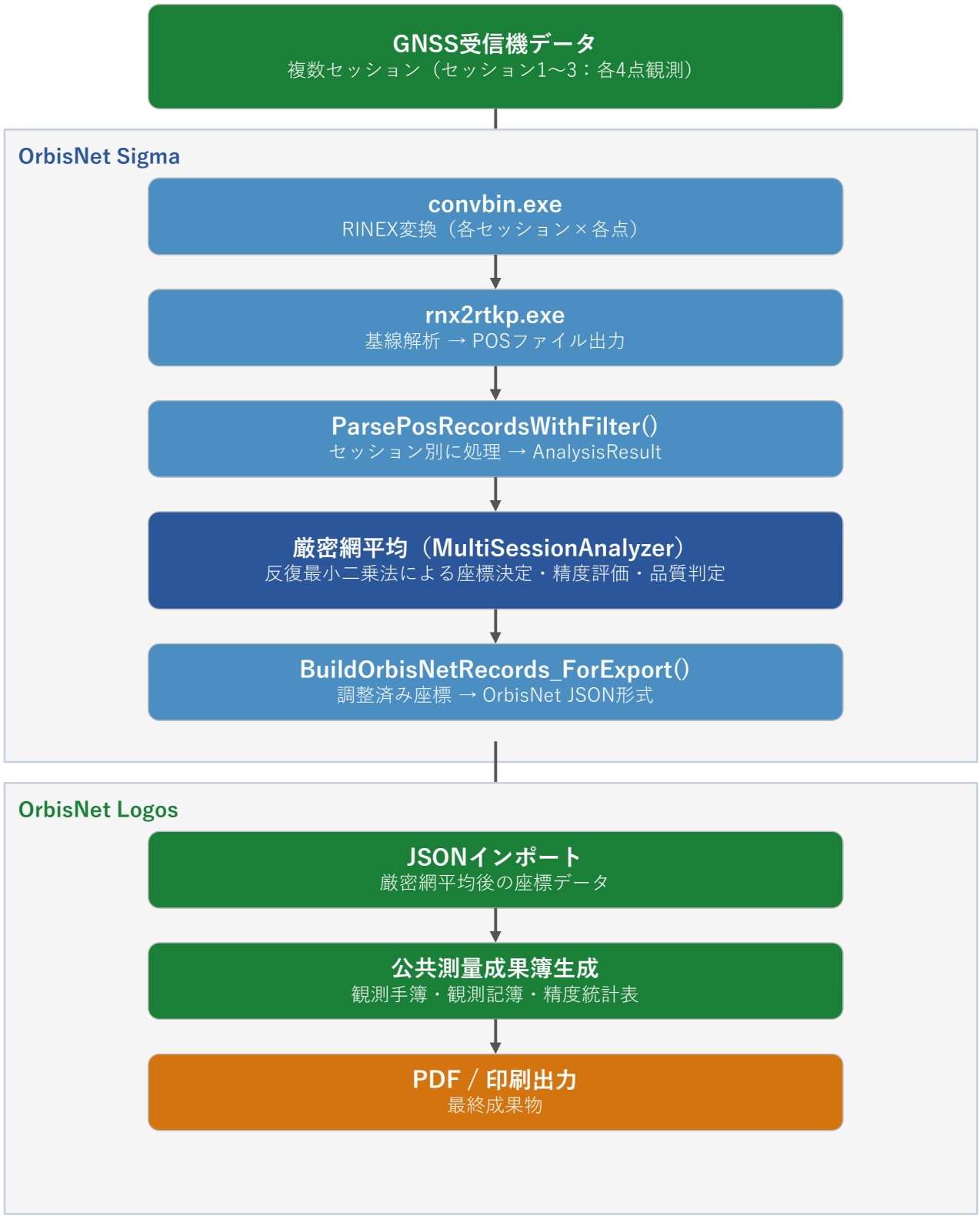
品質等級の例（3級基準点測量）

品質等級	水平精度基準	鉛直精度基準
一級基準点	＜ 5 mm	＜ 10 mm
二級基準点	＜ 50 mm	＜ 100 mm
三級基準点	＜ 100 mm	＜ 150 mm
要再観測	＞ 100 mm	＞ 150 mm

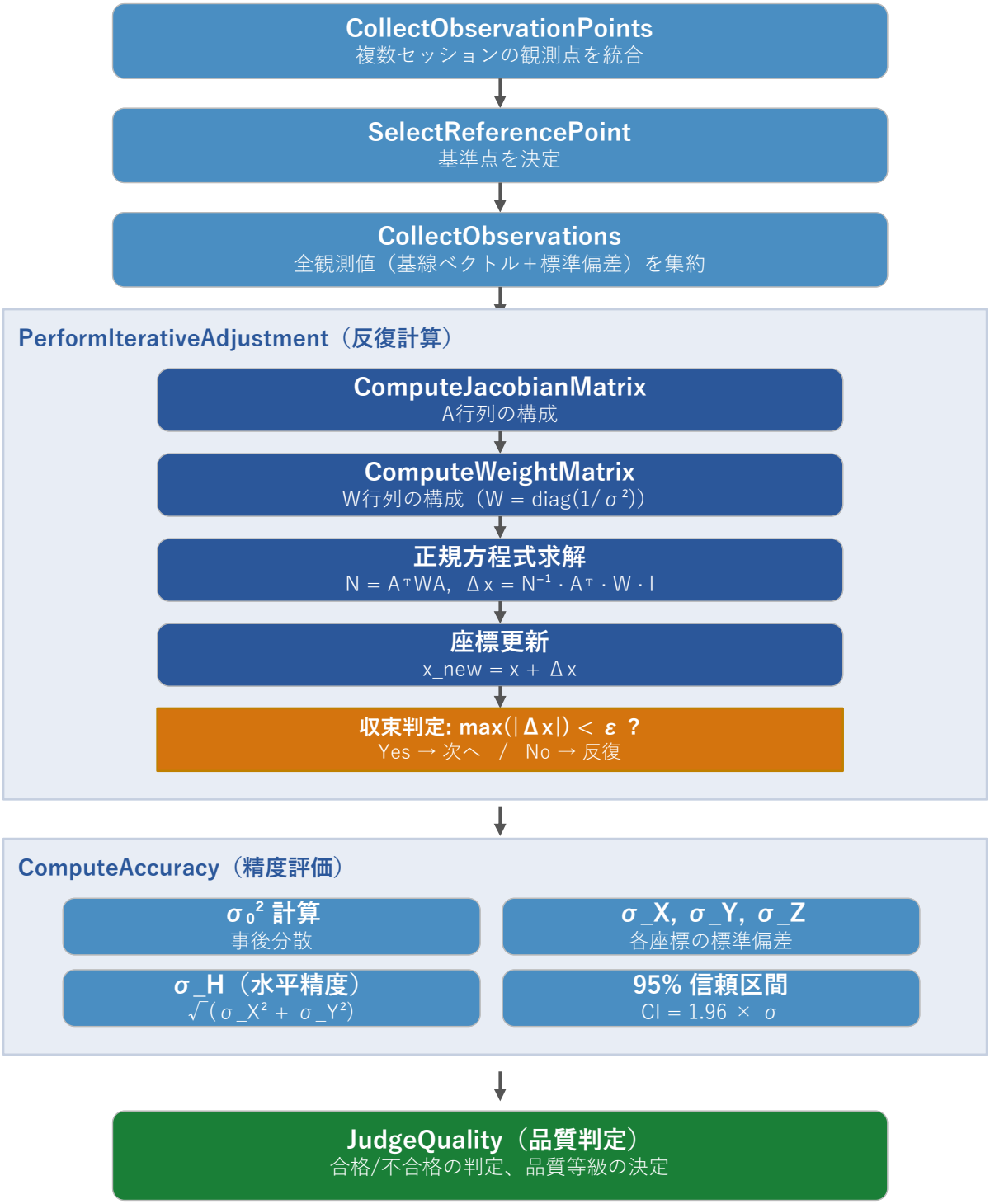
判定ロジック

```
if (σ_H <= 5 mm AND σ_V <= 10 mm) {
    qualityGrade = "一級基準点";
    isPassed = true;
} else if (σ_H <= 50 mm AND σ_V <= 100 mm) {
    qualityGrade = "二級基準点";
    isPassed = true;
} else if (σ_H <= 100 mm AND σ_V <= 150 mm) {
    qualityGrade = "三級基準点";
    isPassed = true;
} else {
    qualityGrade = "要再観測";
    isPassed = false;
}
```

ワークフロー


GNSS受信機データ
複数セッション（セッション1〜3：各4点観測）
OrbisNet Sigma
convbin.exe
RINEX変換（各セッション×各点）
rnx2rtkp.exe
基線解析 → POSファイル出力
ParsePosRecordsWithFilter()
セッション別に処理 → AnalysisResult
厳密網平均（MultiSessionAnalyzer）
反復最小二乗法による座標決定・精度評価・品質判定
BuildOrbisNetRecords_ForExport()
調整済み座標 → OrbisNet JSON形式
OrbisNet Logos
JSONインポート
厳密網平均後の座標データ
公共測量成果簿生成
観測手簿・観測記簿・精度統計表
PDF / 印刷出力
最終成果物

厳密網平均 詳細フロー


CollectObservationPoints
複数セッションの観測点を統合
SelectReferencePoint
基準点を決定
CollectObservations
全観測値（基線ベクトル＋標準偏差）を集約
PerformIterativeAdjustment（反復計算）
ComputeJacobianMatrix
A行列の構成
ComputeWeightMatrix
W行列の構成（W = diag(1/σ²)）
正規方程式求解
N = AᵀWA, Δx = N⁻¹ · Aᵀ · W · l
座標更新
x_new = x + Δx
収束判定: max(|Δx|) < ε ?
Yes → 次へ / No → 反復
ComputeAccuracy（精度評価）
σ₀² 計算
事後分散
σ_X, σ_Y, σ_Z
各座標の標準偏差
σ_H（水平精度）
√(σ_X² + σ_Y²)
95% 信頼区間
CI = 1.96 × σ
JudgeQuality（品質判定）
合格/不合格の判定、品質等級の決定

チェックリスト（厳密網平均の検証）

- [ ] セッション数が2以上あるか
- [ ] 各セッションの観測点が3点以上か
- [ ] 基準点の座標が正確に設定されているか
- [ ] 各観測値の標準偏差が妥当か（0.1 mm〜10 mm程度）
- [ ] 反復計算が収束しているか（通常3〜5回で収束）
- [ ] 事後標準偏差 σ_0 が1付近か（0.5〜2.0の範囲が正常）
- [ ] 残差 $|v|$ が異常に大きくないか
- [ ] 精度の悪い点が明確に特定されているか
- [ ] 品質判定基準が公共測量規程に準拠しているか