サンプルプログラムの計算式について

細田 真道

[日本語 / 英語 (English)]

連立方程式を解くニュートン法ライブラリ > サンプルプログラムの計算式について

目次

1 はじめに

本ライブラリの三種類のサンプルプログラム

- 最小二乗法を使わないサンプル
  - sample.cc
- 最小二乗法を使ったサンプル（重み無し）
  - sample-non_weighted.cc
- 最小二乗法を使ったサンプル（重み付き）
  - sample-weighted.cc

は、いずれも GPS 衛星の座標と距離から GPS 受信機の位置を計算する、というものになっています。基本的な考え方は文献 [1] に拠っていますが、使用している計算式は異なります。ここでは、サンプルプログラムで使用している計算式と、その計算式を使ったサンプルプログラムの処理について説明します。サンプルプログラムで使用している座標、距離、重み等のデータについてはサンプルプログラムのデータについてをご覧ください。

2　計算式

2.1　衛星の座標と距離

GPS は 4 つ以上の GPS 衛星の位置を既知、それぞれの衛星から GPS 受信機までの距離を既知、として未知である GPS 受信機の位置を求めるものです。

2.1.1　座標

n 個の衛星の座標（既知）を $S_i\left(x_i, y_i, z_i\right), (i = 1, 2, ..., n)$ として、受信機の座標（未知）を $P\left(x_\mathrm{p}, y_\mathrm{p}, z_\mathrm{p}\right)$ とします。

2.1.2　距離

各衛星から受信機までの距離は、誤差を含んだ仮の距離（疑似距離）として計測されます（既知）。これを R_i とします。一方、衛星と受信機の真の距離は

$$\sqrt{\left(x_i - x_\mathrm{p}\right)^2 + \left(y_i - y_\mathrm{p}\right)^2 + \left(z_i - z_\mathrm{p}\right)^2}$$

となります。この、真の距離と疑似距離との差は受信機の時計オフセット（未知）によるものとなります。これを ΔS で表すと、

$$R_i = \sqrt{\left(x_i - x_\mathrm{p}\right)^2 + \left(y_i - y_\mathrm{p}\right)^2 + \left(z_i - z_\mathrm{p}\right)^2} + \Delta S \tag{1}$$

という関係があることになります。

2.1.3　方程式

未知数は $x_\mathrm{p}, y_\mathrm{p}, z_\mathrm{p}, \Delta S$ の 4 個ですので、衛星が 4 個あれば、式 (1) より、

$$\begin{cases} R_1 = \sqrt{\left(x_1 - x_\mathrm{p}\right)^2 + \left(y_1 - y_\mathrm{p}\right)^2 + \left(z_1 - z_\mathrm{p}\right)^2} + \Delta S \\ R_2 = \sqrt{\left(x_2 - x_\mathrm{p}\right)^2 + \left(y_2 - y_\mathrm{p}\right)^2 + \left(z_2 - z_\mathrm{p}\right)^2} + \Delta S \\ R_3 = \sqrt{\left(x_3 - x_\mathrm{p}\right)^2 + \left(y_3 - y_\mathrm{p}\right)^2 + \left(z_3 - z_\mathrm{p}\right)^2} + \Delta S \\ R_4 = \sqrt{\left(x_4 - x_\mathrm{p}\right)^2 + \left(y_4 - y_\mathrm{p}\right)^2 + \left(z_4 - z_\mathrm{p}\right)^2} + \Delta S \end{cases}$$

という連立方程式を解くことによって受信機の座標 $P\left(x_\mathrm{p}, y_\mathrm{p}, z_\mathrm{p}\right)$ および真の距離と疑似距離の誤差 ΔS を求めることができます。しかし、この方程式は非線形であるため、解析的に解を得ることは困難です。そこでニュートン法で近似解を得る方法をとります。

2.2 関数の準備

本ライブラリでは、解きたい連立方程式について、ベクトル関数の形で表した $\boldsymbol{F}$ と、そのヤコビ行列 $\boldsymbol{J}$ を用意する必要があります。

2.2.1 F

まず、式 (1) を以下のように関数の形に書き換えます。

$$f_i\left(x, y, z, \Delta S\right) = \sqrt{\left(x_i - x\right)^2 + \left(y_i - y\right)^2 + \left(z_i - z\right)^2} + \Delta S - R_i \tag{2}$$

そうすると、解くべき方程式は

$$f_i\left(x_\mathrm{p}, y_\mathrm{p}, z_\mathrm{p}, \Delta S\right) = 0$$

となります。これをベクトルの形で書き直すことにします。

$$\boldsymbol{F}(\boldsymbol{X}) = \begin{pmatrix} f_1\left(x, y, z, \Delta S\right) \\ f_2\left(x, y, z, \Delta S\right) \\ \vdots \\ f_n\left(x, y, z, \Delta S\right) \end{pmatrix} \tag{3}$$

ただし、

$$\boldsymbol{X} = \begin{pmatrix} x \\ y \\ z \\ \Delta S \end{pmatrix}, \boldsymbol{0} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

とします。 すると、解くべき方程式は、

$$\boldsymbol{F}(\boldsymbol{X}) = \boldsymbol{0} \tag{4}$$

になります。

2.2.2 J

次に、ここからヤコビ行列 $\boldsymbol{J}$ を導きます。 ヤコビ行列は、

$$\boldsymbol{J} = \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{X}}$$

なので、成分で表すと、

$$\boldsymbol{J}(\boldsymbol{X}) = \begin{pmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial z} & \frac{\partial f_1}{\partial \Delta S} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial z} & \frac{\partial f_2}{\partial \Delta S} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_n}{\partial x} & \frac{\partial f_n}{\partial y} & \frac{\partial f_n}{\partial z} & \frac{\partial f_n}{\partial \Delta S} \end{pmatrix} \tag{5}$$

となります。各成分は式 (2) を偏微分して、

$$\frac{\partial f_i}{\partial x} = \frac{-(x_i - x)}{\sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2}}$$

$$\frac{\partial f_i}{\partial y} = \frac{-(y_i - y)}{\sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2}}$$

$$\frac{\partial f_i}{\partial z} = \frac{-(z_i - z)}{\sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2}}$$

$$\frac{\partial f_i}{\partial \Delta S} = 1$$

となります。

2.3 ニュートン法

ここでは、ライブラリ本体 newton.hh / newton.cc のニュートン法の処理を簡単に紹介します。 ニュートン法では、まず、解の候補となる初期値 $\boldsymbol{X}_0$ を適当に決めます。

$$\boldsymbol{X}_0 = \begin{pmatrix} x^{(0)} \\ y^{(0)} \\ z^{(0)} \\ \Delta S^{(0)} \end{pmatrix} \tag{6}$$

そして $\boldsymbol{F}(\boldsymbol{X}_0)$ と $\boldsymbol{J}(\boldsymbol{X}_0)$ から、次の解の候補 $\boldsymbol{X}_1$ への修正値 $\Delta\boldsymbol{X}$ を計算します。

$$\Delta\boldsymbol{X} = \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta s \end{pmatrix}$$

これを一般化して、k 番目の解の候補を $\boldsymbol{X}_k$ とします。

$$\boldsymbol{X}_k = \begin{pmatrix} x^{(k)} \\ y^{(k)} \\ z^{(k)} \\ \Delta S^{(k)} \end{pmatrix}$$

そして、$\boldsymbol{F}(\boldsymbol{X}_k)$ と $\boldsymbol{J}(\boldsymbol{X}_k)$ を使って、次の解の候補への修正値 $\Delta\boldsymbol{X}$ を求め、$k+1$ 番目の解の候補を求める、ということを繰り返します。 具体的には、$\boldsymbol{J}$ が正方行列であれば、つまり、衛星が 4 個であれば、

$$\boldsymbol{J}(\boldsymbol{X}_k)\,\Delta\boldsymbol{X} = -\boldsymbol{F}(\boldsymbol{X}_k) \tag{7}$$

を $\Delta\boldsymbol{X}$ について解くことになります。これは数学的には逆行列を使えば求めることができます。

$$\Delta\boldsymbol{X} = -\left(\boldsymbol{J}(\boldsymbol{X}_k)\right)^{-1}\boldsymbol{F}(\boldsymbol{X}_k)$$

しかし、この方法は数値計算的には効率が悪く、また、演算結果が安定しない等の問題があるとされているため、本ライブラリではここで逆行列によらず連立一次方程式を解くアルゴリズムを使っています。つまり、$\boldsymbol{A} = \boldsymbol{J}(\boldsymbol{X}_k), \boldsymbol{x} = \Delta\boldsymbol{X}, \boldsymbol{b} = -\boldsymbol{F}(\boldsymbol{X}_k)$ と置いて、連立一次方程式

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{8}$$

を $\boldsymbol{x}$ について解くアルゴリズムを使います。そして、これにより求めた $\Delta\boldsymbol{X}$ を使って次の近似解 $\boldsymbol{X}_{k+1}$ を求める、

$$\boldsymbol{X}_{k+1} = \boldsymbol{X}_k + \Delta\boldsymbol{X}$$

という操作を繰り返し行い、 解の候補を真の解に順次近づけていきます。このとき、一定の誤差 ε を決めておき、

$$\|\boldsymbol{F}(\boldsymbol{X}_k)\| < \varepsilon \tag{9}$$

を満たした場合や

$$\left\| \begin{matrix} \frac{\Delta x}{x^{(k)}} \\ \frac{\Delta y}{y^{(k)}} \\ \frac{\Delta z}{z^{(k)}} \\ \frac{\Delta s}{\Delta S^{(k)}} \end{matrix} \right\| < \varepsilon \tag{10}$$

を満たした場合に計算を打ち切り $\boldsymbol{X}_k$ を近似解として出力します。

2.4 最小二乗法

ここでは、ライブラリ本体 newton.hh / newton.cc の最小二乗法の処理を簡単に紹介します。衛星が 4 個よりも多いとき、ヤコビ行列 $\boldsymbol{J}$ は縦長となり、正方行列ではなくなるため、式 (7) が計算できなくなります。こういうときに、測定値には誤差があることを前提にして、最小二乗法を使います。

2.4.1 重み無し

いずれの測定値も誤差が同じである場合や、誤差の情報が無い場合には、すべて平等に誤差があるとして「重み無し」の最小二乗法を使います。 数学の教科書的には、式 (7) の代わりに、正規方程式

$$(\boldsymbol{J}^\top \boldsymbol{J})\,\Delta\boldsymbol{X} = -\boldsymbol{J}^\top \boldsymbol{F} \tag{11}$$

を使えば、重み無しの最小二乗法が適用できます。ただし、この方法は数値計算的には結果が悪くなるので、正規方程式を使わない方がよいとされています [1-4] 。本ライブラリがデフォルトで使用する連立一次方程式を解くアルゴリズムは $\boldsymbol{J}$ が縦長の時、 つまり式 (8) で $\boldsymbol{A}$ が縦長であっても自動的に最小二乗法が適用された解 $\boldsymbol{x}$ が得られるもので、 正規方程式を使わずに $\Delta\boldsymbol{X}$ が得られます。

2.4.2 重み付き

各衛星からの測定値について、誤差が既知である場合に「重み付き」の最小二乗法を使います。 重み付きの正規方程式を使う場合は、各測定値の誤差の分散 ${\sigma_i}^2$ の逆数を使った対角行列

$$\boldsymbol{W} = \begin{pmatrix} \frac{1}{{\sigma_1}^2} & 0 & \cdots & 0 \\ 0 & \frac{1}{{\sigma_2}^2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{{\sigma_n}^2} \end{pmatrix}$$

を重み行列として使い、 式 (11) の代わりに重み付きの正規方程式

$$(\boldsymbol{J}^\top \boldsymbol{W} \boldsymbol{J})\,\Delta\boldsymbol{X} = -\boldsymbol{J}^\top \boldsymbol{W} \boldsymbol{F}$$

を使えばよいことになります。もちろん、数値計算的には正規方程式を使わない方が良いとされます [1-4] 。正規方程式を使わない方法としては、各測定値の誤差の標準偏差 σ_i の逆数を使った対角行列

$$\boldsymbol{W}^{\frac{1}{2}} = \begin{pmatrix} \frac{1}{\sigma_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sigma_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\sigma_n} \end{pmatrix} \tag{12}$$

を重み行列として使い、式 (7) の代わりに

$$\left(\boldsymbol{W}^{\frac{1}{2}}\boldsymbol{J}\right)\Delta\boldsymbol{X} = -\boldsymbol{W}^{\frac{1}{2}}\boldsymbol{F} \tag{13}$$

を使い、 $\boldsymbol{A} = \boldsymbol{W}^{\frac{1}{2}}\boldsymbol{J}, \boldsymbol{x} = \Delta\boldsymbol{X}, \boldsymbol{b} = -\boldsymbol{W}^{\frac{1}{2}}\boldsymbol{F}$ と置いて、重み無しと同様の $\boldsymbol{A}$ が縦長でも式 (8) の連立一次方程式が解けるアルゴリズムを使い、解 $\boldsymbol{x}$ 、 つまり $\Delta\boldsymbol{X}$ を得ます [1, 3] 。本ライブラリではどちらの方法も使えますが、サンプルプログラムは後者の正規方程式を使わない方法を使っています。

3 サンプルプログラム

サンプルプログラムで使用している座標、距離、重み等のデータについては、ここでは中身を示すだけにとどめ詳しい説明をしません。 これらの詳細はサンプルプログラムのデータについてをご覧ください。

3.1 最小二乗法を使わないサンプル

以下、最小二乗法を使わないサンプル sample.cc からの抜粋です。

3.1.1 衛星座標

```
// Satellite position (known, given from received data)
std::vector<std::vector<double>> Si
{
  {   -11327938.990000,     9886884.330000,    21895433.227000 },
  {     4755496.711000,    19362623.328000,    18112665.323000 },
  {    -7506201.243000,    24076860.073000,     7092793.940000 },
  {   -23085789.286000,    12409399.010000,     4602891.246000 },
};
```

4 衛星の座標 $S_i, (i = 1, 2, 3, 4)$ を設定しています。 $\boldsymbol{F}$ を求める式 (3) や $\boldsymbol{J}$ を求める式 (5) の計算に使います。

3.1.2 距離

```
// Observed distance (known, measurement result)
std::vector<double> Ri
{
     20690632.972000,
     23225588.018000,
     21288081.687000,
     21187099.471000,
};
```

それぞれの衛星から受信機までの距離 R_i, $(i = 1, 2, 3, 4)$ を設定しています。 $\boldsymbol{F}$ を求める式 (3) や $\boldsymbol{J}$ を求める式 (5) の計算に使います。

3.1.3 受信機座標（正解データ）

```
// Receiver position (actually unknown)
std::vector<double> unknown_P
{
   -3947719.36876915,    3364403.46661849,    3699487.64248845
};
```

本ライブラリの計算結果が正しいか確かめるための正解データとする目的で、文献 [1]「(その 3)」のプログラムを重み無しに設定して計算した座標です。本ライブラリのニュートン法の計算には使いません。

3.1.4 計算打ち切り条件

```
// Error epsilon (0.1 mm)
double epsilon {0.0001};
```

本ライブラリでニュートン法の計算を打ち切る基準として使用する値です。 式 (9) や式 (10) の ε として使います。

3.1.5 補助関数

```
inline double square (double x)
{
  return x * x;
}
```

二乗を計算する関数です。

```
inline double distance (const std::vector<double> &p1,
                        const std::vector<double> &p2)
{
  return sqrt ( square ( p1[0] - p2[0])
                + square ( p1[1] - p2[1])
                + square ( p1[2] - p2[2]) );
}
```

距離を計算する関数です。

3.1.6 F

```
// Calculate F (to be solved as equations)
std::vector<double> calc_f (const std::vector<double> &Xarg)
{
  std::vector<double> fv;

  for (int i = 0; i < Si.size (); ++i)
    fv.push_back (distance (Si[i], Xarg) + Xarg[3] - Ri[i]);

  return fv;
}
```

$\boldsymbol{F}$ を求める式 (3) を計算する関数です。ライブラリからコールバックされます。

3.1.7 J

```
// Calculate J (Jacobian matrix of F)
std::vector<std::vector<double>> calc_j (const std::vector<double> &Xarg)
{
  std::vector<std::vector<double>> jv;

  for (int i = 0; i < Si.size (); ++i)
    {
      std::vector<double> j_row;
      double d = distance (Si[i], Xarg);

      for (int n = 0; n < 3; ++n)
```

```
        j_row.push_back (- (Si[i][n] - Xarg[n]) / d);
      j_row.push_back (1.0);

      jv.push_back (j_row);
    }

  return jv;
}
```

$\boldsymbol{J}$ を求める式 (5) を計算する関数です。ライブラリからコールバックされます。未知数 4 で衛星の数も 4 なので 4 行 4 列の正方行列となります。

3.1.8 計算

```
  std::vector<double> solution;
  try
    {
      newton_method::newton_method nm;

      nm.set_function (calc_f, calc_j);
      nm.set_epsilon_F (epsilon);
      nm.set_epsilon_deltaX (epsilon);
      std::vector<double> initial_value {0.0, 0.0, 0.0, 0.0};
      solution = nm.solve (initial_value);
    }
```

ニュートン法の計算をしています。

- インスタンス nm を作成
- nm.set_function () で $\boldsymbol{F}, \boldsymbol{J}$ を計算する関数をセット
  - それぞれ式 (3)、式 (5) を計算する関数です
- nm.set_epsilon_F () および nm.set_epsilon_deltaX () で 計算打ち切り条件を設定
  - それぞれ式 (9)、式 (10) の条件を設定します
- initial_value に計算の初期値を設定
  - 式 (6) に相当します
- nm.solve () でニュートン法の計算を実施し、solution に解を得る
  - この中のライブラリの処理は、 nm.set_function () でセットされた $\boldsymbol{F}, \boldsymbol{J}$ を計算する関数を繰り返しコールバックすることで近似解を真の解に近づけていき、 式 (9) または式 (10) の条件を満たした近似解を返します

という流れになっています。

3.1.9 結果表示

```
  std::cout << std::endl << "Solution: P (x y z) and delta_S:" <<
      std::endl;
  for (auto i: solution)
    std::cout << " " << i;
  std::cout << std::endl;

  std::cout << "Error distance:" << std::endl;
  double error_distance = distance (solution, unknown_P);
  std::cout << " " << error_distance << std::endl;
```

最終的に得られた解として受信機の座標 $P\left(x_{\mathrm{p}}, y_{\mathrm{p}}, z_{\mathrm{p}}\right)$ と時計オフセットによる距離の誤差 ΔS を表示します。次いで、受信機座標の正解データとの距離を計算して表示します。

3.1.10 実行結果

サンプルプログラムの実行結果は以下のようになりました（抜粋）。

```
Solution: P (x y z) and delta_S:
 -3947717.825152 3364407.721345 3699485.385124 -14.272990
Error distance:
 5.057781
```

文献 [1] のプログラムを重み無しに設定したものと比べて約 5 m 強の誤差がある、ということになります。文献 [1] のプログラムは 8 衛星を使って最小二乗法で計算したものに対して、 このサンプルプログラムは 4 衛星しか使っていませんから、 この程度の誤差は出るものと思います。

3.2 最小二乗法を使ったサンプル（重み無し）

以下、最小二乗法を使ったサンプル（重み無し）sample-non_weighted.cc からの抜粋です。

3.2.1 衛星座標

```
// Satellite position (known, given from received data)
std::vector<std::vector<double>> Si
{
  {   -11327938.990000,     9886884.330000,    21895433.227000 },
```

```
  {      4755496.711000,     19362623.328000,     18112665.323000 },
  {     -7506201.243000,     24076860.073000,      7092793.940000 },
  {    -23085789.286000,     12409399.010000,      4602891.246000 },
  {    -21893190.888000,     -2248546.668000,     14796664.928000 },
  {    -24893247.395000,      3827508.606000,     -8794926.751000 },
  {    -12971740.598000,    -10587013.898000,     21061849.442000 },
  {      7069732.127000,     22267387.067000,     12627670.276000 },
};
```

8 衛星の座標 S_i, $(i = 1, 2, ..., 8)$ を設定しています。 $\boldsymbol{F}$ を求める式 (3) や $\boldsymbol{J}$ を求める式 (5) の計算に使います。

3.2.2 距離

```
// Observed distance (known, measurement result)
std::vector<double> Ri
{
      20690632.972000,
      23225588.018000,
      21288081.687000,
      21187099.471000,
      21833271.739000,
      24393427.283000,
      24031767.538000,
      23630886.925000,
};
```

それぞれの衛星から受信機までの距離 R_i, $(i = 1, 2, ..., 8)$ を設定しています。 $\boldsymbol{F}$ を求める式 (3) や $\boldsymbol{J}$ を求める式 (5) の計算に使います。

3.2.3 その他

その他の部分は最小二乗法を使わないサンプルと同じです。この場合、衛星の数が 8 個に増えているため、ヤコビ行列 $\boldsymbol{J}$ が正方行列ではなくなり、8 行 4 列の縦長となります。縦長の場合、本ライブラリはデフォルトで最小二乗法が適用された結果を返します。

3.2.4 実行結果

サンプルプログラムの実行結果は以下のようになりました（抜粋）。

```
Solution: P (x y z) and delta_S:
```

```
 -3947719.368769 3364403.466618 3699487.642488 -15.392384
Error distance:
 0.000000
```

文献 [1] のプログラムを重み無しに設定したものと比べ（表示桁数の範囲で）誤差ゼロとなりました。本ライブラリやサンプルプログラムは、文献 [1] とは計算式、処理等が異なりますが、同じデータ、同じ条件で計算した場合の結果は一致した、ということになります。

3.3 最小二乗法を使ったサンプル（重み付き）

以下、最小二乗法を使ったサンプル（重み付き）sample-weighted.cc からの抜粋です。

3.3.1 重み行列

```
// Observed weight (known, measurement result)
std::vector<double> W
{
          0.34246575,
          0.41806020,
          0.44662796,
          0.33967391,
          0.33411293,
          0.29682398,
          0.30759766,
          0.31046259,
};
```

式 (12) の重み行列 $W^{\frac{1}{2}}$ です。実際には行列なのですが、対角行列なので、その成分を並べた std::vector<double> を用意します。

3.3.2 受信機座標（正解データ）

```
// Receiver position (actually unknown)
std::vector<double> unknown_P
{
   -3947719.26542369,   3364403.97164603,   3699487.31861822
};
```

本ライブラリの計算結果が正しいか確かめるための正解データとする目的で、文献 [1]「(その 3)」のプログラムを重み付きに設定して計算した座標です。先のサンプルプログラムでは、重み無し設定で計算した座標を使っていたため数字が異なります。本ライブラリの計算結果が正しいか確かめるための正解データとする目的で使い、本ライブラリのニュートン法の計算には使いません。

3.3.3 計算

```
  std::vector<double> solution;
  try
    {
      newton_method::newton_method nm;

      nm.set_function (calc_f, calc_j);
      nm.set_epsilon_F (epsilon);
      nm.set_epsilon_deltaX (epsilon);
      nm.set_weight (W);
      nm.set_least_square (newton_method::least_square::weighted);
      std::vector<double> initial_value {0.0, 0.0, 0.0, 0.0};
      solution = nm.solve (initial_value);
    }
```

重みをつけた計算をしています。 追加された処理は、

- nm.set_weight () で重み行列を設定
- nm.set_least_square () で重みをつけた計算をするように設定
  - 設定した重み行列を式 (12) とみなして、 式 (13) の方法で計算する設定になります

の 2 つだけです。

3.3.4 その他

その他の部分は重み付きのサンプルと同じです。

3.3.5 実行結果

サンプルプログラムの実行結果は以下のようになりました（抜粋)。

```
Solution: P (x y z) and delta_S:
 -3947719.265424 3364403.971646 3699487.318618 -15.633489
Error distance:
 0.000000
```

重み付きの場合も、文献 [1] のプログラムをと比べて（表示桁数の範囲で）誤差ゼロとなりました。

もう一度書きますが、本ライブラリやサンプルプログラムは、文献 [1] とは計算式、処理等が異なります。それでも、同じデータ、同じ条件で計算した場合の結果は一致した、ということになります。

4　参考文献

[1] 福島荘之介. 理解するための GPS 測位計算プログラム入門. http://www.enri.go.jp/~fks442/K_MUSEN/.

[2] 小柳義夫.「最小二乗法」事始め. 応用数理, Vol. 26 (2016) No. 1, pp. 39-42. http://doi.org/10.11540/bjsiam.26.1_39

[3] 小柳義夫. 最小 2 乗法における新しい手法. 応用物理, Vol. 46 (1977) No. 1, pp. 55-60. http://doi.org/10.11470/oubutsu1932.46.55

[4] 小柳義夫. 最小二乗法の新しいアルゴリズム. 情報処理, Vol. 23 (1982) No. 2, pp. 99-108. http://id.nii.ac.jp/1001/00006446/

5　License