理解するための GPS測位計算プログラム入門

（その1）WGS84と座標変換のはなし

独立行政法人　電子航法研究所　福島　荘之介

1. はじめに

衛星関連を勉強しはじめた無線技術者の方から，「GPSの測位原理は習ったんですが，実際にはどう計算するんですか？教科書に数式はあるんですが，自分で計算しても，思うようにいかないんです」というご質問を頂くことがある。確かにGPSを基礎から説明した良書[1.1－1.5]は既にあり，測位計算の方法についても詳細に説明されている。しかし，「この式から実施にどう数値計算するか」を扱った解説はなく，実際に計算するには高い障壁を感じる方も多いかもしれない。また，教科書の式のとおりに計算しても，市販の受信機と同じ結果になるとは限らない。

そこで，本稿ではなるべく簡単に，GPSでよく扱う数値計算の手法を紹介し，市販の受信機との計算結果の比較を試みる。また，自作のC言語のソースプログラムを掲載し，読者がパーソナルコンピュータを使用して，実際に計算の過程を確認したり，改良を試みられるようにする。ただし，紙面に多量のプログラムリストを掲載することを避け，説明のための最小限に留めて残りはインターネットで配布する。また，1回が1つの話題で完結するようにし，以下の順で3回程度の解説を行いたい。

1．WGS84と座標変換のはなし
2．GPS衛星の軌道計算のはなし
3．測位計算のはなし

読者の対象は，GPSを勉強しはじめた若手の技術者，または若手でなくてもGPSの測位原理は，概念的に理解しているが，さらに一歩，二歩進んだ厳密な理解を望んでいる技術者としたい。

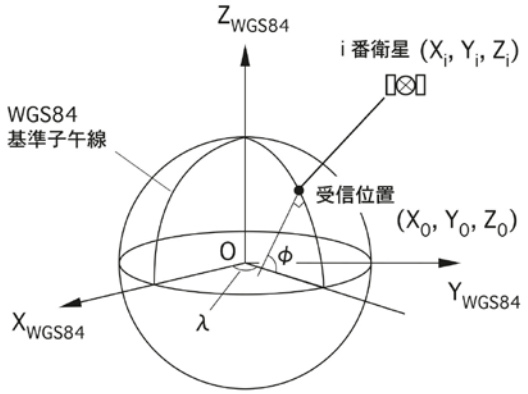

図1．1：WGS84 座標系

2. 「WGS84」って何ですか？

WGS84（World Geodetic System 1984）は，GPSの基準座標系（**図1．1**）であり，重要なのは直交座標（右手系）であるということである。直交座標であれば，例えば*i*番衛星の位置は，（Xi, Yi, Zi）のように3つの数字の組（ベクトル）で表せる。また，通常地表面近くにある受信位置（例えば航空機位置）も同様に（Xo, Yo, Zo）と表すことができる。これを，**地球中心・地球固定直交座標系 ECEF**（earth centered, earth fixed）と呼ぶ。この座標系の定義は以下である。

原点 ＝地球重心
Z軸　＝BIH（Bureau International de l’Heure：国

際報時局 [1]）の定義する極運動のCIO（Conventional International Origin：慣用国際原点）の方向に平行

X軸 ＝WGS84基準子午面（BIHにより定義される経度ゼロ）と平均赤道面の交線

Y軸 ＝Z，X軸と右手直交系をなす

平たく言えば，地球の重心（どうやって測るかは別として）を原点として，地球の自転軸の北極方向をZ軸とする。Z軸に垂直にグリニッジ子午線の方向をX軸として，これらの軸と直交するように右手系でY軸を決めるという意味である。また極運動とは，自転軸が長周期で半径 10m 程の範囲で円を描くように動いていることを意味しており，CIOは 1900～1905 年の北極の平均位置である。

次に，**WGS84 準拠楕円体**とは，地球の形状に近似した回転楕円体であり，楕円を Z 軸中心に回転させたものを言う。この楕円の長半径 a と偏平率 f などは定数として与えられている（**表1．1**）。また，短半径 b，離心率 e と偏平率 f の間には，次の関係がある。

$$e = \sqrt{a^2 - b^2}\,/\,a$$

$$f = (a-b)/a$$

この準拠楕円体を用いると，さきほど ECEF で表した受信位置は，緯度（φ），経度（λ），楕円対高（h）で表すことができる。これを**測地座標**という。λは受信位置の子午線と基準子午線が赤道面上でなす角，φは受信位置の子午線の接線からおろした垂線（垂直線）が赤道面と交わる角である（ECEF 座標の原点を通らないことに注意）。また，h は楕円体からの高さ（垂直線方向）であり，平均海面レベルを基準とするジオイド高とは異なる。

以上から地球上または空間の位置は，WGS84 座標系によって，ECEF 直交座標または測地座標で表示できることがわかる。これらの座標上の位置は相互に変換が可能であり，（φ，λ，h）が与えら

表1．1：WGS84 座標の定数

記号（単位）	パラメータ	定数
a (m)	赤道面平均半径	6 378 137
f	扁平率	1/298.257 223 563
Ω−dot (rad/s)	平均自転速度	7.292 115 146 7E−5
GM (m^3/s^2)	重力定数	3.986 005E14
C (m/s)	光速	2.997 924 58E8

れたとき，（x，y，z）は，

$$x = (N+h)\cos\phi\cos\lambda$$
$$y = (N+h)\cos\phi\sin\lambda$$
$$z = \{N(1-e^2)+h\}\sin\phi$$

と計算することができる。ここで，

$$N = a\Big/\sqrt{1-e^2\sin^2\phi}$$

$$e^2 = 2f - f^2$$

である。また逆に，（X，Y，Z）から（φ，λ，h）に変換するには，

$$\phi = \tan^{-1}\{(z+e'^2 b\sin^3\theta)\big/(p-e^2 a\cos^3\theta)\}$$
$$\lambda = \tan^{-1}(y/x)$$
$$h = (p/\cos\phi) - N$$

と近似式を使えばよい。ただし，

$$p = \sqrt{x^2+y^2}$$
$$\theta = \tan^{-1}(za/pb)$$
$$e^2 = (a^2-b^2)/a^2$$
$$e'^2 = (a^2-b^2)/b^2$$

である。これらの式の導出は，測地学の良書[1.6]に詳しい解説がある。

3. 変換プログラムをつくる（ECEF 座標⇔測地座標）

以上の計算をC言語のプログラムで書いた例（**リスト1．1**：blh_ecef.c）を紹介する。このプログラムは短いので全文を載せる。リストは2つの関数を含んでいて，blh2ecef()は測地座標から ECEF 直交座標への変換を，ecef2blh()はその逆変換を行う。引

[1] BIH は 1988 年から IERS（International Earth Rotation Service：国際地球回転事業）の一部として組織を改編した.

数と返値は vector 型の構造体で，これは**リスト1．3**で定義している（後で使うために matrix 型も定義しておく）。また，**リスト1．2**には**表1．1**で示した定数パラメータなどを定義する。GPS の計算では，

$\pi = 3.1415926535898$

とする約束がある[1.7]。main()は，サンプルデータを使いこれらの関数を実行して動作をチェックする。サンプルの位置は，仙台空港内にある電子航法研究所の基準点である。main()では，まずこの測地座標を ECEF 直交座標に変換し，次に得られた ECEF 座標から測地座標に逆変換する。**リスト1．1**の最後の「結果」と同じ値が出力され，初めに与えた値に戻れば成功である。ここで緯経度の桁数に関する注意を1つ。1センチメートルの精度を必要とする場合，緯経度（度の単位）は小数点以下8桁で十分である。また，度分秒で{DD, MM, SS.SSSS}と表示する場合には，SS.SSSS は小数点以下4桁で十分である。これは，「緯度の1分はおよそ1NM（1852m）に相当する」という覚えやすい事実から概算できる。

4. 自分の位置を中心に（地平座標へ変換）

ECEF 直交座標を用いると，地球上または空間の位置を(X, Y, Z)で表すことができる。しかし，我々が実際に必要な量は，自分の立っている場所から見た対象物（例えば飛行機）の距離や方位角，仰角といった値である。このためには，**地平直交座標**を定義し，ECEF 直交座標から変換する必要がある。地平直交座標は，水平線座標，局所座標とも呼ばれる。地平直交座標は，**図1．2**に示すように，地表面付近のある点を原点として，天頂方向（垂直線の上方向）にZ軸の正を，これに直角に東方向に X 軸を，北方向に Y 軸をとる。このため，ENU 座標（East, North, Up）と呼ばれることもある。Z 軸の負の方向は鉛直線方向（重力の働く方向）と厳密には異なる。しかし，その差は大きくても数十秒程度と言われている。

ECEF 直交座標から地平直交座標への変換は，回転と原点移動のみによって実現される。ある直交座標系をある軸の周りに回転させると，回転後の

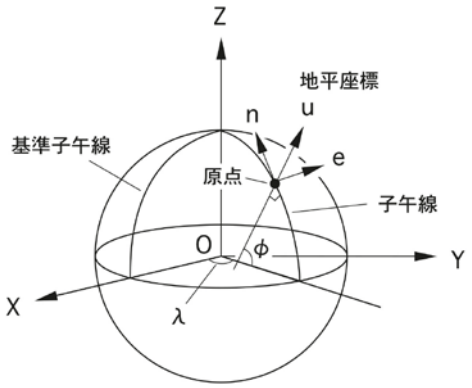

図1．2:地平座標への変換

位置 y は，回転行列 R と回転前の位置 x によって，

$y = R\,x$

と表される。A 軸の周りに右ネジの方向に θ 回転する行列を R（A, θ ）と表せば，

$$R(x,\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{pmatrix}$$

$$R(y,\theta) = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix}$$

$$R(z,\theta) = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

となる。**図1．2**を見ながら，この変換を考える。まず，ECEF 直交座標を Z 軸中心に λ 度回転し，基準子午線を原点の子午線に一致させる（Z 軸の正を親指の方向に右手で持ち，右ねじの方向に回すという意味）。次に，Y 軸中心に（90-ϕ ）度回転し，Z 軸を原点の天頂方向に一致させる。このとき，X 軸は南を向いているので，さらに Z 軸中心に 90 度回転させて，東方向に向ける。最後に原点を移動させる。これを，回転行列の積の順序に注意して書

けば，地平座標での位置 (e, n, u)は，ECEF 直交座標での位置（x，y，z）と原点の位置（x_0，y_0，z_0）で，

$$\begin{pmatrix} e \\ n \\ u \end{pmatrix} = R(z,90)\,R(y,90-\phi)\,R(z,\lambda)\begin{pmatrix} x-x_0 \\ y-y_0 \\ z-z_0 \end{pmatrix}$$

と表せる。

5. 滑走路の長さは？（地平座標への変換プログラム）

このプログラムも，内容を眺めていただく意味で**リスト1．4** に掲載する。関数 rotx(), roty(), rotz()は，それぞれ，X,Y,Z 軸周りの回転行列であり，返値はさきほど定義した構造体の matrix 型を使う。ecef2enu()は，座標変換を行っている部分である。行列とベクトルの積（matvec）や行列どうしの積（matmat）というような関数は一般的なものなので，別プログラム（math_util.c）にまとめた（掲載はせず，インターネットで配布する）。また，動作チェック用の main()では，ecef2blh()を使う。従って，**リスト1．4**を実行するためには，math_util.c と blh_ecef.c を同時に make する必要がある。チェック用の main()で使う変換位置は，仙台空港の B 滑走路 27 側の進入側の末端（滑走路中心線上）の緯経度である（**図1．3**）。これを反対 09 側の末端の位置を原点とした地平直交座標上の位置で示す。実行結果がリスト末にある result= の値になれば成功である。地平座標では，ある位置までの距離や角度を簡単に計算できる。このため**リスト1．4**の最後では，X-Y 平面上の距離を計算している。この結果，滑走路長（length）は 3,000m によく一致していることがわかる。また，滑走路の向き（angle）は，東方向から左回りに約 7.5 度偏位している。この偏位は，磁方位を用いる滑走路の標記と地平座標で用いる真方位との差であり，仙台付近の磁気偏差（magnetic deviation）に一致する。

この**リスト1．4**は，測地座標のある位置（ϕ_1，λ_1，h_1）から，別の位置（ϕ_2，λ_2，h_2）までの距離，方位角，仰角などが計算できる。読者が身近な位置

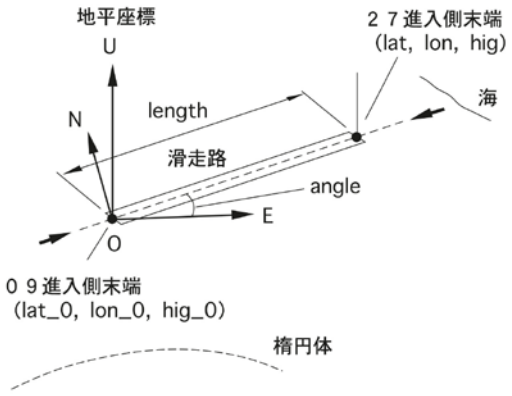

図1．3：地平座標の例（仙台空港B滑走路）

のデータで試されると，様々な応用が可能と思う。例えば，データとして AIP（Aeronautical Information Publication）にある空港標点，航空保安施設の位置 [2]などを想定できる。または，アンテナの位置を原点として，レーダの ρ - θ 座標，VORやILSの方位角，高低角に変換することも可能である。

6. おわりに

今回は，プログラムリストが短かったので大部分を掲載しました。次回からは長くなるので説明に必要な最小限に留めるつもりです。本稿で紹介したプログラムリスト，サンプルデータなどは，インターネットの以下のアドレスから配布します（紙面からタイプする必要はありません）。

http://www.enri.go.jp/~fks442/K_MUSEN/

コンパイルは Linux の gcc で動作確認しておりますが，基本的にはどの環境でも動作可能と思います。Windows 版のCコンパイラでの動作例や簡単な Makefile をつけておきますので，自分の環境に合わせて使ってください。

プログラムは，自作のため冗長な部分も多く，もしかするとバグが潜んでいる可能性もあります。このため，製品などには使用せずに，個人の責任で

[2] ただし，AIP には楕円体高がないので，別に調べる必要があります。

使ってください。プログラムの目的は，本記事のタイトルどおり，**GPS を理解すること**にあります。

今回はプログラムも短く，小手調べといったところですが，次回は衛星の軌道要素から衛星の ECEF 直交座標位置を求める計算を紹介する予定です。ご質問，ご意見，ご指摘などありましたら，電子メールでお願いします。

参考文献

[1.1] 日本測地学会編著,「GPS－人工衛星による精密測位システム－」, 日本測量協会, 1898.

[1.2] 土屋, 辻,「GPS測量の基礎」, 日本測量協会, 1999.

[1.3] 高野, 佐藤, 柏木, 村田,「宇宙における電波計測と電波航法」, コロナ社, 2000.

[1.4] “Global Positioning System: Theory and Applications,” Vol. 1, Vol. 2, AIAA, 1996.

[1.5] P. Misra, P. Enge, “GLOBAL POSITIONING SYSTEM: Signals, Measurements, and Performance,” Ganga-Jamuna Press, 2001.

[1.6] 萩原,「測地学入門」, 東京大学出版, 1982.

[1.7] “ICD-GPS-200C,” Oct. 1993.

表1．2:プログラムの説明

1．プログラム名：blh_ecef.c:
目的：（緯度，経度，楕円体高）と ECEF 直交座標の（X，Y，Z）の相互変換
動作：
main()：メイン：
- ・（緯度φ，経度λ，楕円体高h）を与える
- ・blh2ecef()：（φ，λ，h）を ECEF 直交座標の（X，Y，Z）に変換
- ・ecef2blh()：ECEF 直交座標の（X，Y，Z）を（φ，λ，h）に変換
- ・変換結果を表示

2．プログラム名：ecef_enu.c
目的：ECEF 直交座標から地平線座標への変換
動作：
main()：メイン
- ・変換する位置と原点の（φ，λ，h）を与える
- ・blh2ecef()：変換する点を ECEF に変換
- ・blh2ecef()：原点を ECEF に変換
- ・ecef2enu()：ECEF 座標を地平座標に変換
  - ・回転変換・原点移動（math_util.c の幾何関数を使用）
- ・変換結果を表示

rotx()：回転行列（x 軸を中心に右ねじの方向）
roty()：回転行列（y軸を中心に右ねじの方向）
rotz()：回転行列（z軸を中心に右ねじの方向）

3．プログラム名：math_util.c
目的：幾何変換の関数群
関数：
matvec()：行列とベクトルの乗算
matmat()：行列と行列の乗算
matinv()：逆行列を求める
factor()：matinv()で使用する
subst()：　〃
amax()：　〃
transpose()：行列の転置を求める
abs1()：絶対値を求める
newton()：ニュートン法

リスト1．1　**blh_ecef.c**

```
/*---
  WGS84 の緯度,経度,高さと直交座標(ECEF)の変換
  2002.4.25: S.Fukushima(ENRI)
---*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "def01.h"
#include "str01.h"

vector blh2ecef(double phi, double ramda, double height)
/* 緯度,経度,高さから ECEF 座標に変換 */
{
 vector ecef;

 ecef.n = 3;
 ecef.a[0] = (NN(phi)+height)*cos(phi*PI/180)*cos(ramda*PI/180);
 ecef.a[1] = (NN(phi)+height)*cos(phi*PI/180)*sin(ramda*PI/180);
 ecef.a[2] = (NN(phi)*(1-E2)+height)*sin(phi*PI/180);

 return ecef;
}

vector ecef2blh(vector ec)
/* ECEF 座標から WGS84 の{緯度,経度,楕円体高}へ変換 */
{
 vector blh;
 int i = 0;
 double phi, ramda, height, p;
 double x, y, z;
```

```
 double sita;

 ec.n = 3; blh.n = 3;
 x = ec.a[0], y = ec.a[1], z = ec.a[2];

 p = sqrt(x*x + y*y);
 sita = (180/PI) * atan2(z*A, p*B);

/*--- 緯度 */
 phi = (180/PI) * atan2(z+ED2*B*(CUB(sin(sita*PI/180))),(p-
 E2*A*(CUB(cos(sita*PI/180)))));

/*--- 経度 */
 ramda = (180/PI) * atan2(y,x);

/*--- 高さ */
 height = (p / cos(phi*PI/180)) - NN(phi);

 blh.a[0] = phi; blh.a[1] = ramda; blh.a[2] = height;
 return blh;
}

/*--- チェック用 */
int main()
{
  double lat, lon, hig;
  vector ecef, blh;

  lat =  38.13579617;
  lon = 140.91581617;
  hig =  41.940;

  ecef = blh2ecef(lat, lon, hig);
  printf("%.3f %.3f %.3f\n", ecef.a[0], ecef.a[1], ecef.a[2]);

  blh = ecef2blh(ecef);
  printf("%.8f %.8f %.3f\n", blh.a[0], blh.a[1], blh.a[2]);
}

/*---
  結果: -3899086.094 3166914.545 3917336.601
        38.13579617 140.91581617 41.940
---*/
```

リスト1．２ def01.h

```
/*--- 定数 */
#define      MAXN     12

/*--- 2 乗, 3 乗 */
#define SQR(x)          ((x)*(x))
#define CUB(y)          ((y)*(y)*(y))

/*--- WGS84 座標パラメータ */
#define PI              3.1415926535898
#define A               6378137.0           /* Semi-major axis */
#define ONE_F           298.257223563       /* 1/F */
#define B               (A*(1.0 - 1.0/ONE_F))
#define E2              ((1.0/ONE_F)*(2-(1.0/ONE_F)))
#define ED2             (E2*A*A/(B*B))
#define NN(p)           (A/sqrt(1.0 - (E2)*SQR(sin(p*PI/180.0))))

#define C               2.99792458E+08      /* Speed of light */
#define MU              3.986005E+14        /* Earth's universal gravity
  */
#define OMEGADOTE 7.2921151467E-05  /*  Earth's  rotation  rate
  (rad/s) */
#define F               4.442807633E-10     /* F sec/m^(1/2) */
```

リスト1．３ str01.h

```
/*--- ベクトルの定義 */
typedef struct {
  int n;                    /* size of vector */
  double a[MAXN];           /* elements of vector */
  int err;                  /* err=1: error */
} vector;

/*--- 行列の定義 */
typedef struct {
  int n;                    /* size of raw */
  int m;                    /* size of column */
  double a[MAXN][MAXN];     /* elements of matrix */
  char message[80];         /* error report */
  int err;                  /* err=1: error */
} matrix;
```

リスト1．４ ecef_enu.c

```
/*---
  WGS84 の直交座標(ECEF)から水平線座標(ENU)の変換
  2002.8.27: S.Fukushima(ENRI)
---*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "def01.h"
#include "str01.h"

extern vector blh2ecef(double, double, double);
extern vector ecef2blh(vector);
extern matrix matmat(matrix *, matrix *);
extern vector matvec(matrix, vector);

matrix rotx(double sita)
/* x 軸回りの sita 度の回転変換:右ねじの方向 */
{
  matrix rota;
  rota.n = rota.m = 3;

  rota.a[0][0] = 1;
  rota.a[0][1] = 0;
  rota.a[0][2] = 0;

  rota.a[1][0] = 0;
  rota.a[1][1] = cos(sita*PI/180.0);
  rota.a[1][2] = sin(sita*PI/180.0);

  rota.a[2][0] = 0;
  rota.a[2][1] = -sin(sita*PI/180.0);
  rota.a[2][2] = cos(sita*PI/180.0);

  return rota;
}

matrix roty(double sita)
{
  matrix rota;
  rota.n = rota.m = 3;

  rota.a[0][0] = cos(sita*PI/180.0);
  rota.a[0][1] = 0;
  rota.a[0][2] = -sin(sita*PI/180.0);

  rota.a[1][0] = 0;
  rota.a[1][1] = 1;
  rota.a[1][2] = 0;
```

```
  rota.a[2][0] = sin(sita*PI/180.0);
  rota.a[2][1] = 0;
  rota.a[2][2] = cos(sita*PI/180.0);

  return rota;
}

matrix rotz(double sita)
{
  matrix rota;
  rota.n = rota.m = 3;

  rota.a[0][0] = cos(sita*PI/180.0);
  rota.a[0][1] = sin(sita*PI/180.0);
  rota.a[0][2] = 0;

  rota.a[1][0] = -sin(sita*PI/180.0);
  rota.a[1][1] = cos(sita*PI/180.0);
  rota.a[1][2] = 0;

  rota.a[2][0] = 0;
  rota.a[2][1] = 0;
  rota.a[2][2] = 1;

  return rota;
}
vector ecef2enu(vector dest, vector origin)
/*--- ECEF 座標を水平線座標(ENU)へ変換する */
{
  int i, j;
  vector mov, ret, blh;
  matrix rotyp, rotzp1, rotzp2;
  matrix mat_conv1, mat_conv2;

  origin.n       = 3;          origin.err   = 0;
  mov.n          = 3;          mov.err      = 0;
  ret.n  =  3;    ret.err  =  0;

  blh = ecef2blh(origin);

  rotzp1 = rotz(90.0);
  rotyp  = roty(90.0 - blh.a[0]);
  rotzp2 = rotz(blh.a[1]);

  mat_conv1 = matmat(&rotzp1, &rotyp);
  mat_conv2 = matmat(&mat_conv1, &rotzp2);

  for(i=0;i<3;i++) mov.a[i] = dest.a[i] - origin.a[i];
  ret = matvec(mat_conv2, mov);
  return ret;
}

/*--- チェック用 */
int main()
{
   double lat, lon, hig, lat_o, lon_o, hig_o;
   vector ecef, ecef_o, enu;

   lat  =  38.14227288;    /*--- 変換する位置座標(B27 海側)*/
   lon  = 140.93265738;
   hig  =  45.664;

   lat_o  =  38.13877338;                /*--- 原点の座標(B09 山側) */
   lon_o = 140.89872429;
   hig_o =  44.512;

   ecef = blh2ecef(lat, lon, hig);
   ecef_o = blh2ecef(lat_o, lon_o, hig_o);
   enu = ecef2enu(ecef, ecef_o);

   printf("result= %.3f %.3f %.3f\n", enu.a[0], enu.a[1], enu.a[2]);
   printf("length= %.3f, angle(deg)= %.3f\n",
                  sqrt(SQR(enu.a[0])+SQR(enu.a[1])),atand(enu.a[1]/
  enu.a[0]));
}

/*---
   結果: result= 2974.681 388.988 0.447
         length= 3000.006, angle(deg)= 7.450
---*/
```