GPS Solut (2005) 9: 51–58
DOI 10.1007/s10291-004-0116-4

Klaus Hehl

C++ and Java code for recursion formulas in mathematical geodesy

Received: 25 June 2004
Accepted: 18 September 2004
Published online: 4 March 2005

The code and PHP forms are available from the GPS Toolbox website at http://www.ngs.noaa.gov/gps-toolbox and the author's site at http://hehl.tfh-berlin.de/GL/.

The GPS Toolbox is a column dedicated to highlighting algorithms and source code utilized by GPS Engineers and scientists. If you have an interesting program or software package you would like to share with our readers, please pass it along; e-mail it to us at gps-toolbox@ngs.noaa.gov. To comment on any of the source code discussed here, or to download source code, visit our website at http://www.ngs.noaa.gov/gps-toolbox. This column is edited by Stephen Hilla, National Geodetic Survey, NOAA, Silver Spring, Maryland, and Mike Craymer, Geodetic Survey Division, Natural Resources Canada, Ottawa, Ontario, Canada.

K. Hehl
University of Applied Sciences (TFH),
Luxemburger Str. 10, 13353 Berlin,
Germany
E-mail: hehl@tfh-berlin.de
Tel.: +49-30-45042611
Fax: +49-30-45042632

Abstract The central part of this paper is the publication of C++ and Java code for the solution of a number of basic tasks related to the geodesic/meridian arc in mathematical geodesy. We provide an introduction to a recursive formulation of the series expansions of the underlying integrals. Recursive formulation has the advantage that simple relationships are identical for all orders and thus can easily be programmed. The code examples shown below, together with the algorithm sections, can be easily transferred to other programming languages. For testing and teaching purposes the development of forms for online computation, using the Perl, and PHP Hypertext Processor scripting languages, is ongoing.

Introduction

In geodesy many computations use the ellipsoid of revolution as the mathematical reference surface. This includes computations using GPS where, e.g., in navigation applications it is often required to determine the coordinates of an unknown point given the azimuth and distance from a known point, or to find the azimuths and shortest distance along the ellipsoid between two given points. These problems are known as the direct and inverse problems in geodesy, respectively. A consequence of expressing the computations on the ellipsoid is that quite a number of relations are series expansions, such as the meridian arc length, UTM-coordinates, and the

direct and inverse problems mentioned above. While preparing a new series of lectures on mathematical geodesy, I took the opportunity to look for the best algorithms available and to code them into C++, Java, and the Perl, PHP Hypertext Preprocessor scripting languages. For theoretical developments and for teaching, I also use MathCad. The theoretical developments of the expressions used in this contribution are available in Klotz (1991, 1993). The subject matter is also addressed, in part, in Thomas (1952) and Snyder (1982).

A lot of integrals in mathematical geodesy can be formulated such that the coefficients are computable *recursively*. This has the following big advantages:

1. Coefficients are calculated from the same relation for every degree
2. We get the highest accuracy by simply changing one parameter—the upper limit of a for-loop
3. Simple arithmetic makes it possible to use the routines for both real number computation and complex number computation, and
4. Algorithms using the geodesic line as a connecting curve on the ellipsoid are no longer limited to a certain range or accuracy.

Computation of meridian arc lengths

Some elements of theory

The meridian arc, i.e. the shortest distance of a given point on the ellipsoid from the equator, can be derived using relations of differential geometry. We start from the parameter representation

$$x(\beta) = \begin{pmatrix} a \cdot \cos(\beta) \\ 0 \\ b \cdot \sin(\beta) \end{pmatrix}$$

with the semimajor axis a, the semiminor axis b, and the reduced latitude β (see Fig. 1). Using the flattening $f = (a - b)/a$ we relate β to the ellipsoidal latitude φ by the relation $\tan\beta = (1 - f) \cdot \tan\varphi$.

Introducing the first eccentricity, $e^2 = (a^2 - b^2)/a^2$, we find the following closed expression for the length of the desired meridian arc G shown in Fig. 2:

$$G(\beta) = a \int_0^{\beta} \sqrt{1 - e^2 \cos^2(x)}\, dx.$$

This relation needs numerical evaluation (e.g. using Matlab, Mathematica, Maple, or MathCad) to compute a function value. It is well known that series expansion is the key to finding a formulation, which will allow one to write efficient subroutines in any programming language. The fundamental idea is outlined below.

We use the following series expansion

$$(1 - x)^{\alpha} = \sum_{n=0}^{\infty} \binom{\alpha}{n} \cdot (-1)^n \cdot x^n$$

which holds for any real number α and converges for any $|x| < 1$.The real numbers

$$\binom{\alpha}{n} = \frac{\alpha \cdot (\alpha - 1) \cdot (\alpha - 2) \cdot \ldots \cdot (\alpha - n + 1)}{1 \cdot 2 \cdot \ldots \cdot n}$$

are called generalized binomial coefficients.

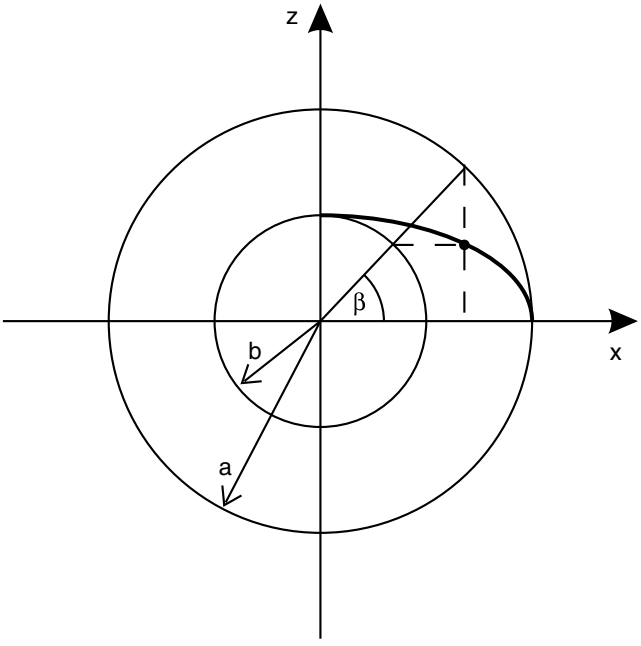

Fig. 1 The ellipse as an affine mapping of a circle

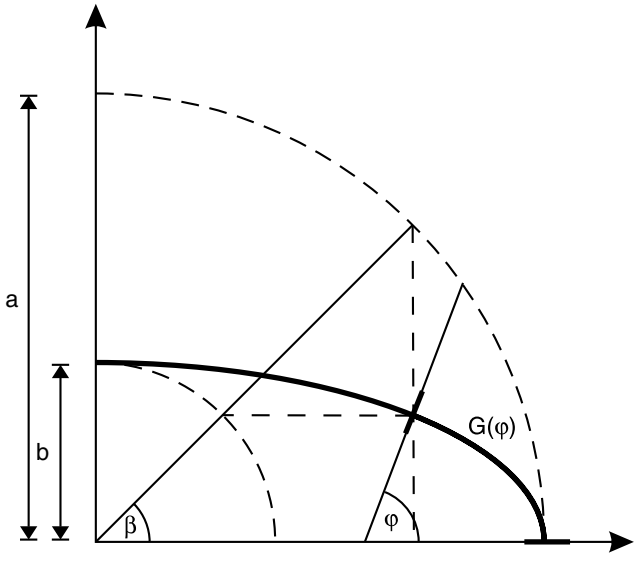

Fig. 2 An example meridian arc G

Applying this series to our problem (for every geodetic ellipsoid, $e^2 \cos^2$ is much smaller than 1, so we can expect rapid convergence) we get

$$\begin{aligned}\sqrt{1-e^2\cos^2 x} &= \left(1-e^2\cos^2 x\right)^{1/2} \\ &= \sum_{n=0}^{\infty} \binom{0.5}{n} \cdot \left(-e^2\cos^2 x\right)^n \\ &= \binom{0.5}{0} - \binom{0.5}{1} \cdot \left(e^2\cos^2 x\right)^1 \\ &\quad + \binom{0.5}{2} \cdot \left(e^2\cos^2 x\right)^2 \pm \cdots\end{aligned}$$

where the first binomial coefficients are:

$$\begin{aligned} n=0: &\quad \binom{0.5}{0} = 1 \\ n=1: &\quad \binom{0.5}{1} = \frac{0.5}{1} = \frac{1}{2} = -\binom{0.5}{0} \cdot \frac{2\cdot 1-3}{2\cdot 1} \\ n=2: &\quad \binom{0.5}{2} = \frac{0.5\cdot(0.5-1)}{1\cdot 2} = -\frac{1}{8} \\ &\quad = -\binom{0.5}{1} \cdot \frac{2\cdot 2-3}{2\cdot 2}\end{aligned}$$

The coefficient of order n follows the rule indicated by the right hand sides

$$\binom{0.5}{n} = -\binom{0.5}{n-1} \cdot \frac{2\cdot n-3}{2\cdot n}$$

and can thus be *recursively* computed with the start value, which is by definition $\binom{0.5}{0} = 1$.

For the integrals that are encountered we find the following relation

$$\int \cos^{2n}\beta \, d\beta = \frac{1}{2n}\cos^{2n-1}\beta \sin\beta + \frac{2n-1}{2n}\int \cos^{2n-2}\beta \, d\beta$$

Since we would like to only outline the basic idea, we omit steps needed to find the recursion formulas for the remaining integrals. Eventually, we obtain two sets of coefficients, which are recursively computed by

$$\begin{aligned} &c_0 = 1; \quad c_n = c_{n-1} \cdot \frac{2n-1}{2n} \cdot \frac{2n-3}{2n} \cdot e^2 \\ &k_0 = 1; \quad k_n = k_{n-1} \cdot \frac{2n}{2n+1} \cdot \cos^2\beta\end{aligned}$$

The first set of coefficients c_n depends only on the selected ellipsoid, and the second set of coefficients k_n depend only on the latitude of a given point.

The theoretical developments lead to the following simple equation

$$G(\beta) = a \cdot \left(\text{koeff1} \cdot \beta + \frac{1}{2} \cdot \text{koeff2} \cdot \sin(2\beta)\right)$$

with the coefficients

$$\begin{aligned}\text{koeff1} &= \sum_{n=0}^{N} c_n = (c_0 + c_1 + \cdots + c_N) \\ \text{koeff2} &= \sum_{n=1}^{N} c_n \cdot \sum_{m=0}^{n-1} k_m = c_1\cdot k_0 + c_2\cdot(k_0+k_1) + \cdots \\ &\quad + c_N \cdot (k_0 + k_1 + \cdots + k_{N-1}) \\ &= k_0 \cdot (c_1 + c_2 + c_3 + \cdots + c_N) \\ &\quad + k_1 \cdot (c_2 + c_3 + \cdots + c_N) + \cdots + k_{N-1}\cdot c_N\end{aligned}$$

Algorithm

A suitable algorithm to calculate G is given below.

1. Select order N ($N=4$, e.g. guarantees millimeter accuracy) and ellipsoid (semimajor axis a, flattening $f=(a-b)/a$
2. Calculate the first eccentricity $e^2 = f\cdot(2-f)$
3. Calculate quantities depending on ellipsoid
   (a) Recursively calculate coefficients
   $$c_0 = 1; \quad c_n = c_{n-1} \cdot \frac{2n-1}{2n} \cdot \frac{2n-3}{2n} \cdot e^2$$
   (b) Form and store the ($N\times 1$)-vector
   $$\boldsymbol{k}_2 = \begin{pmatrix} c_1 + c_2 + c_3 + \cdots + c_N \\ c_2 + c_3 + \cdots + c_N \\ \vdots \\ c_N \end{pmatrix}$$
   (c) Calculate coefficient koeff1 $= 1 + (c_1 + \ c_2 + c_3 + \cdots + c_N) = 1 + \boldsymbol{k}_2[0$ the symbol $\boldsymbol{k}_2$ [0] denotes the first element of the vector k_2
4. Calculate quantities depending on the ellipsoidal latitude φ
   (a) Calculate reduced latitude β from $\tan\beta = (1-f)\cdot\tan\varphi$, calculate $csquare = \cos^2\beta$
   (b) Recursively calculate
   $$k_0 = 1; \quad k_n = k_{n-1} \cdot \frac{2n}{2n+1} \cdot csquare$$
   (c) Form and store the ($N\times 1$)-vector
   $$\boldsymbol{k}^{\mathrm{T}} = (\,k_0 \quad k_1 \quad \cdots \quad k_{N-1}\,)$$
   (d) Calculate coefficient from scalar product
   $$\text{koeff2} = \boldsymbol{k}_2^{\mathrm{T}} \cdot \boldsymbol{k}$$
5. Calculate length of meridian arc
   $$G(\beta) = a \cdot \left(\text{koeff1} \cdot \beta + \frac{1}{2} \cdot \text{koeff2} \cdot \sin(2\beta)\right)$$

Of course, for a given accuracy, truncated expressions can be given, e.g. for $N=4$

$$\text{koeff1} = 1 - \frac{1}{4}e^2 - \frac{3}{64}e^4 - \frac{15}{768}e^6 - \frac{525}{49,152}e^8$$

For latitude $\beta = 90° = \frac{\pi}{2}$ we get $G(90°) = a \cdot \text{koeff1} \cdot \frac{\pi}{2}$ so that the circumference of the corresponding ellipse is $2\pi{\cdot}a \cdot \text{koeff1}$.

C++ code

The coefficients c and k are not of interest so, a corresponding code (in C++ notation) looks like

```
// length of meridian arc G using recursion formulas
// (c) hehl@tfh-berlin.de  September, 2004
// given a in [m], e2 = f * (2.0-f), and beta in [rad]
const int N = 8;                                          // 01
// calculate quantities depending on ellipsoid            // 02
double k2[N] = { 0.0 };                                   // 03
double c = 1.0;                                           // 04
for(int n=1;n<=N;n++) {                                   // 05
  double n2 = 2.0 * n;                                    // 06
  c *= (n2-1.0)*(n2-3.0)/n2/n2*e2;                        // 07
  for(int m=0;m<n;m++) k2[m] += c;                        // 08
} // end for(n)                                           // 09
double koeff1 = 1.0 + k2[0];                              // 10
// calculate quantities depending on latitude             // 11
double k=1.0, koeff2=k2[0];                               // 12
double cos2 = cos(beta)*cos(beta);                        // 13
for(int n=1;n<N;n++) {                                    // 14
  double n2 = 2.0 * n;                                    // 15
  k *= n2/(n2+1.0)*cos2;                                  // 16
  koeff2 += k2[n]*k;                                      // 17
} // end for(n)                                           // 18
double G = a*beta*koeff1+a/2.0*sin(2.0*beta)*koeff2;      // 19
```

For future use (i.e. calculations of a great number of points on the same ellipsoid to the same degree of accuracy, or calculation of the latitude from given arc length), the vector $\mathbf{k}_2$ should be stored so that the routine above reduces to lines 12–19. In the spirit of object-oriented programming, it is advised to put the lines 1–9 into an initializing routine called from a constructor and let the vector $\mathbf{k}_2$ be class member data. The accuracy is solely controlled by the upper limit N in the for-loops (see lines 1, 5, and 14).

Symbolically, we call the computation of meridian arc length from latitude arc() and the inverse problem (latitude from arc) lat().

Numerical example

On the Hayford ellipsoid (parameters $a = 6378388.0$, $1/f = 297.0$), the meridian arc length is calculated for a point with reduced latitude $\beta = 45°$. We assume that ultimate accuracy is needed ($N = 8$). Using the code above, we get the values in Table 1 for the vector $\mathbf{k}_2$.

With coefficient koeff1 = 0.9983172080559514, koeff2 = −0.0016835008855936161, we finally obtain $G = G_8 = 4995775.138571393$ m (having in mind that mm-accuracy is sufficient for geodetic practice).

Table 1 Contents of an example $\mathbf{k}_2$ vector

n	$k_2[n]$
0	−0.0016827919440485985
1	−2.1244384652680673E−6
2	−5.956017025297512E−9
3	−2.1909338780441636E−11
4	−9.282823558832374E−14
5	−4.2915868628714474E−16
6	−2.105339525487341E−18
7	−1.0726062700225286E−20

We can also have a closer look at the convergence behavior of the series expansion by examining the values of G for different N given in Table 2.

The second column gives the arc length as a function of N. Column three shows the difference between the 'correct' value and the current value of G. The remaining columns give the log values and their differences. We can draw from this the following conclusions:

1. Developing the series up to $N = 4$ is sufficient for geodetic accuracy requirements
2. Increasing the development order by one improves the accuracy by two to three orders of magnitude.

Table 2 Convergence of the example meridian arc

N	G_N (m)	$\Delta G = G_8 - G_N$ (m)	$\log\lvert\Delta G\rvert$	$\Delta\log$
0	5009574.2206	−13799.0821	4.1	/
1	4995794.8173	−19.6787	1.3	−2.8
2	4995775.1963	−0.0577	−1.2	−2.5
3	4995775.1388	−2.2E−4	−3.7	−2.4
4	4995775.1386	−9.2E−7	−6.0	−2.4
5	4995775.1386	−3.7E−9	−8.4	−2.4

Complex number and real number use of code

Especially powerful is the dual use of code using C++ templates. Above we present source code which can be

- called with a real number (the ellipsoidal latitude φ in radians) as the fourth parameter and delivers a real number—the meridian arc length;
- called with a complex number (to be defined below) and delivers the two components of Gauss-Krueger or (scaling with 0.9996) UTM coordinates as the real and imaginary parts, respectively.

For test reasons the word length of a real number within the entire routine can easily be changed (see line 00) from 'float' to 'double' or even 'long double'

```
typedef REAL double;                                  // 00
template <class TYPE>                                 // 01
TYPE arc(int N, REAL a, REAL f, TYPE b)  {            // 02
// hehl@tfh-berlin.de, September 2004                 // 03
  TYPE beta = atan((1.0-f)*tan(b));                   // 04
  REAL koeff1 = 1.0; TYPE koeff2 = 0.0;               // 05
  if(N>=1) {                                          // 06
    REAL c,e2;                                        // 07
    ETYP k,sumk,cos2;                                 // 08
    c = 1.0; k = sumk = 1.0;                          // 09
    e2 = f*(2.0-f);                                   // 10
    cos2 = cos(beta)*cos(beta);                       // 11
    for(int n=1;n<=N;n++) {                           // 12
      double n2 = 2.0 * n;                            // 13
      c *= (n2-1)*(n2-3)/n2/n2*e2;                    // 14
      k *= n2/(n2+1)*cos2;                            // 15
      koeff1 += c;                                    // 16
      koeff2 += sumk*c;                               // 17
      sumk += k;                                      // 18
    } // end for()                                     // 19
  } // end if()                                        // 20
  return(a*beta*koeff1+a/2.0*sin(2.0*beta)*koeff2);   // 21
} // end template arc()                                // 22
```

When called using complex variables, all necessary arithmetic and function calls are performed in the complex domain when using C++. Java templates are meanwhile available and corresponding Java code will be released (Feb. 2004).

Geodetic lines/Geodesics

On an ellipsoid of revolution a geodetic line (GL) is characterized by one simple parameter h, cf. (Klotz 1991). The following relations hold for azimuth α and latitude β of a point on a GL

$$h = \sin(\alpha_i) \cdot \cos(\beta_i) = \sin(\alpha_0) = \cos(\beta_{\mathrm{Max}})$$

A point with index i is any point on the GL, point 0 is the starting point of the GL on the equator, and Max is the northernmost/southernmost point of the line ($\alpha_{\mathrm{Max}} = 90°$); see Fig. 3.

We find the following two extreme values for h:

- The meridian arc is characterized by $h = 0$
- The equator is characterized by $h = \pm 1$.

Therefore, the valid range of h is $-1 \le h \le +1$

The arclength along the GL with starting point 0 is given by

$$s(\beta) = a \cdot \int_0^{\beta} \frac{\cos\beta \cdot \sqrt{1 - e^2\cos^2\beta}}{\sqrt{\cos^2\beta - h^2}}\, d\beta$$

For $h = 0$ this equation reduces to the starting integral for $G(\beta)$ given above. The longitude Λ, which is similar to an ellipsoidal length λ but is zero at point 0,

$$\Lambda(\beta) = h \cdot \int \frac{\sqrt{1 - e^2\cos^2\beta}}{\cos\beta\sqrt{\cos^2\beta - h^2}}\, d\beta$$

For the series expansion we introduce a mapping latitude v by

$$\sin^2 v = \frac{\sin^2\beta}{1 - h^2}$$

and see that, with $h = \cos\beta_{\mathrm{Max}}$, this transformation maps the range $-\beta_{\mathrm{Max}} \le \beta \le \beta_{\mathrm{Max}}$ to

$$-\frac{\pi}{2} \le v \le \frac{\pi}{2}$$

Using series expansion and noting recursion properties leads to the following relations

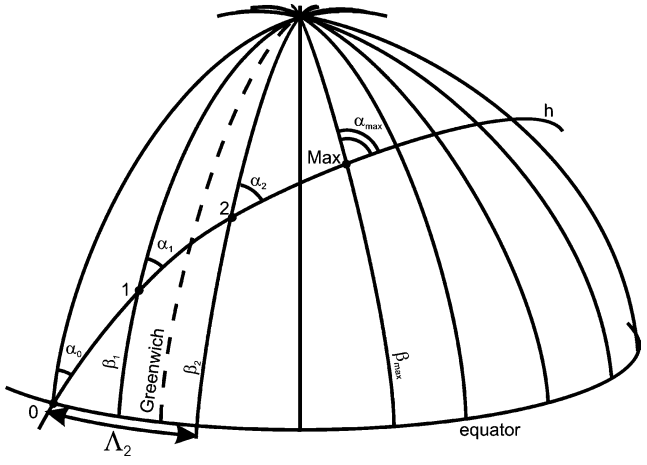

Fig. 3 Parameters associated with a geodesic line

$$s(v) = a \cdot v \cdot \text{koeff1} - \frac{1}{2} \cdot a \cdot \sin 2v \cdot \text{koeff2}$$

$$\Lambda(h, v) = \arcsin(\sin v \cdot \sin \alpha) + h \cdot v \cdot \text{koeff3} - \frac{1}{2} \cdot h \cdot \sin 2v \cdot \text{koeff4}$$

The coefficients will be given next.

Algorithm

1. Select N (for example $N=3$) and thus degree of accuracy; the following vectors and matrices have dimension $(N+1)$. Write binomial coefficients ('Pascal Triangle') into matrices $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$, e.g.

$$\boldsymbol{P}_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 1 & 3 & 3 & 1 \end{pmatrix}^{\mathrm{T}} \quad \text{and}$$

$$\boldsymbol{P}_2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \end{pmatrix}^{\mathrm{T}}$$

2. Calculate quantities which are only dependent on selected ellipsoid using the eccentricity $e^2 = f \cdot (2 - f)$

$$c_0 = 1; \quad c_n = c_{n-1} \cdot \frac{2n-3}{2n} \cdot e^2$$

collect the coefficients into an $(N+1)$-vector $c^{\mathrm{T}} = (c_0 \quad c_1 \quad \cdots \quad c_N)$

3. Calculate quantities that are only dependent on the selected GL

$$d_0 = 1; \quad d_n = -d_{n-1} \cdot \frac{2n-1}{2n} \cdot (1 - h^2)$$

form the $(N+1)$-vector $\boldsymbol{d}^{\mathrm{T}} = (d_0 \quad d_1 \quad \cdots \quad d_N)$ and the matrix (example $N=3$)

$$\boldsymbol{D} = \begin{pmatrix} 0 & d_1 & d_2 & d_3 \\ 0 & 0 & d_2 & d_3 \\ 0 & 0 & 0 & d_3 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

4. Calculate vectors $\boldsymbol{k}_2 = \boldsymbol{D} \cdot \boldsymbol{P}_1 \cdot \boldsymbol{c}$ and $\boldsymbol{k}_4 = \boldsymbol{D} \cdot \boldsymbol{P}_2 \cdot \boldsymbol{c}$ and then the point-independent quantities

$$\text{koeff1} = \boldsymbol{d}^{\mathrm{T}} \cdot \boldsymbol{P}_1 \cdot \boldsymbol{c} \quad \text{and} \quad \text{koeff3} = \boldsymbol{d}^{\mathrm{T}} \cdot \boldsymbol{P}_2 \cdot \boldsymbol{c}$$

5. Calculate quantities that are point-dependent

$$k_0 = 1 \quad k_n = k_{n-1} \cdot \frac{2n}{2n+1} \cdot \sin^2 v$$

collect them into the $(N+1)$-vector $\boldsymbol{k}^{\mathrm{T}} = (k_0 \quad k_1 \quad \cdots \quad k_N)$

6. Finally calculate the remaining two coefficients

$$\text{koeff2} = \boldsymbol{k}_2^{\mathrm{T}} \cdot \boldsymbol{k} \quad \text{and} \quad \text{koeff4} = \boldsymbol{k}_4^{\mathrm{T}} \cdot \boldsymbol{k}$$

For $\beta = \beta_{\mathrm{Max}}$ we get $v = \frac{\pi}{2}$ and thus the extreme values of the GL

$$s_{\mathrm{Max}} = a \cdot \frac{\pi}{2} \cdot \text{koeff1} \quad \text{and} \quad \Lambda_{\mathrm{Max}} = \frac{\pi}{2} \cdot (1 + h \cdot \text{koeff3})$$

Basic tasks

The following four basic tasks have been coded into C++ and Java

1. Closed solutions
   (a) Calculate arc length from latitude
   $$s(v) = a \cdot v \cdot \text{koeff1} - \frac{1}{2} \cdot a \cdot \sin 2v \cdot \text{koeff2}$$
   (b) Calculate longitude from latitude
   $$\Lambda(h, v) = \arcsin(\sin v \cdot \sin \alpha) + h \cdot v \cdot \text{koeff3} - \frac{1}{2} \cdot h \cdot \sin 2v \cdot \text{koeff4}$$
2. Iterative solutions
   (a) Calculate latitude from arc length
   (b) Calculate parameter h from latitude and longitude difference of two different points on the same GL.

These relationships have been originally derived by Klotz (1991). They have been re-formulated here and coded into C++ and Java by the author. For simplicity the algorithm is summarized in Table 3.

Numerical example for the direct problem

On the Bessel ellipsoid (parameters $a = 6377397.155$ m, $1/f = 299.15281285$), we have the coordinates of a given point A

$\varphi_A = 53°50'2.8809''$ $\lambda_A = 10°12'4.1772''$

The connecting geodesic to an unknown point 1 starts at point A with an azimuth

$\alpha_{A1} = 25°16'31.96''$

and has a length of

$s_{A1} = 47652.597m$

We would like to compute the coordinates of the unknown point 1 and the azimuth of the GL at point 1.

Utilizing the four basic tasks the procedure is as follows:

- Calculate arc length from 0 to point A by task (1a)
- Add distance s_{A1}, this gives the arc length from point 0 to point 1
- Calculate corresponding latitude from task (2a), this gives φ_1
- Calculate two longitudes Λ_A and Λ_1 from task (1b) add longitude difference to longitude $\lambda_1 = \lambda_A + (\Lambda_1 - \Lambda_A)$
- Azimuth $\alpha_{1A} = 180° + \arcsin(h/\cos\beta)$.

Following our scheme above we calculate (for $N = 8$) the coefficients given in Table 4.

For testing purposes we use the following intermediate quantities:

Table 3 Computation of Geodesics

Series development up to order N requires matrices/vectors of dimension $(N+1)$
Geodetic requirements: $N = 4$, ultimate accuracy, e.g. $N = 8$ (example below: $N = 3$)

Computation of matrices solely dependent on N ('Pascal triangles')

$$\boldsymbol{P}_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 1 & 3 & 3 & 1 \end{pmatrix}^{\mathrm{T}} \qquad \boldsymbol{P}_2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & 1 \end{pmatrix}^{\mathrm{T}}$$

Test: sum of elements in column n must be 2^n

Computation of quantities solely dependent on the ellipsoid
Semimajor axis a, flattening $f = (a - b)/a$, square of 1st numerical eccentricity $e^2 = f \cdot (2 - f)$

$$c_0 = 1 \quad c_n = c_{n-1} \cdot \frac{2n-3}{2n} \cdot e^2$$

$$(\text{Test:} \sqrt{1 - e^2} - \sum_{n=0}^{N} c_n < \varepsilon(N))$$

$$\boldsymbol{c}^{\mathrm{T}} = (c_0 \quad c_1 \quad \dots \quad c_N)$$

$$\boldsymbol{c}^{\mathrm{T}} = \left(1 \quad -\tfrac{1}{2}e^2 \quad -\tfrac{1}{8}e^4 \quad -\tfrac{1}{16}e^6\right)$$

Computation of quantities solely dependent on the geodesic (GL)
GL is described by the 'Clairaut-Constant' $h = \sin(\alpha_i) \cdot \cos(\beta_i) = \sin(\alpha_0) = \cos(\beta_{\mathrm{Max}})$

$$d_0 = 1; \quad d_n = -d_{n-1} \cdot \frac{2n-1}{2n} \cdot (1 - h^2)$$

$$\boldsymbol{d}^{\mathrm{T}} = (d_0 \quad d_1 \quad \dots \quad d_N)$$

$$\boldsymbol{D} = \begin{pmatrix} 0 & d_1 & d_2 & d_3 \\ 0 & 0 & d_2 & d_3 \\ 0 & 0 & 0 & d_3 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Quantities independent of a specific point on the GL

$$\boldsymbol{k}_2 = \boldsymbol{D} \cdot \boldsymbol{P}_1 \cdot \boldsymbol{c} \qquad \boldsymbol{k}_4 = \boldsymbol{D} \cdot \boldsymbol{P}_2 \cdot \boldsymbol{c}$$

$$\text{koeff1} = \boldsymbol{d}^{\mathrm{T}} \cdot \boldsymbol{P}_1 \cdot \boldsymbol{c} \qquad \text{koeff3} = \boldsymbol{d}^{\mathrm{T}} \cdot \boldsymbol{P}_2 \cdot \boldsymbol{c}$$

$$s_{\mathrm{Max}} = a \cdot \frac{\pi}{2} \cdot \text{koeff1} \qquad \Lambda_{\mathrm{Max}} = \frac{\pi}{2} \cdot (1 + h \cdot \text{koeff3})$$

Point-dependent coefficients
Point has ellipsoidal latitude φ; further $\tan\beta = (1 - f) \cdot \tan\varphi$ and $\sin^2 v = \frac{\sin^2\beta}{1 - h^2}$

$$k_0 = 1 \quad k_n = k_{n-1} \cdot \frac{2n}{2n+1} \cdot \sin^2 v$$

$$\boldsymbol{k}^{\mathrm{T}} = (k_0 \quad k_1 \quad \dots \quad k_N)$$

$$\text{koeff2} = \boldsymbol{k}_2^{\mathrm{T}} \cdot \boldsymbol{k} \qquad \text{koeff4} = \boldsymbol{k}_4^{\mathrm{T}} \cdot \boldsymbol{k}$$

The four basic tasks
(1) Calculate arc length from latitude $s(v) = a \cdot v \cdot \text{koeff1} - \frac{1}{2} \cdot a \cdot \sin 2v \cdot \text{koeff2}$

(2) Calculate longitude from latitude

$$\Lambda(h, v) = \arcsin(\sin v \cdot \sin\alpha) + h \cdot v \cdot \text{koeff3} - \tfrac{1}{2} \cdot h \cdot \sin 2v \cdot \text{koeff4}$$

(3) iteratively calculate latitude from arc length

$$\text{start value}: v_0 = \frac{s}{a \cdot \text{koeff1}}$$

$$v_i = v_0 + \frac{\text{koeff2}(v_{i-1})}{2 \cdot \text{koeff1}} \cdot \sin 2v_{i-1}; \quad i = 1, 2, \cdots$$

repeat until, e.g. $|v_i - v_{i-1}| < 10^{-12}\text{rad}$

(4) iteratively calculate parameter h from two points on GL (see Java or C++ code)

Table 4 Coefficients $\mathbf{k}_2$ and $\mathbf{k}_4$ for N up to 8

N	$k_2[n]$	$k_4[n]$
0	0.0015655914535296317	2.618011423244789E−6
1	−1.8439911546160652E−6	−6.1651745351570805E−9
2	4.829438218657822E−9	2.0178087300013128E−11
3	−1.660588773055953E−11	−7.769163062743946E−14
4	6.57884852191390 8E−14	3.2972532046668064E−16
5	−2.8445128520543706E−16	−1.4923923687814785E−18
6	1.304180912100624E−18	6.809472097997561E−21
7	−5.976814434681885E−21	(0.9966572268183839)

– The parameter of the GL: $h = 0.25251656410048773$
– Starting azimuth at 0: $\alpha_0 = 14°\ 37'\ 35.32655903''$
– Northernmost point of GL: $\beta_{\mathrm{Max}} = 75°\ 25'\ 13.17616373''$
– Distance of point 1 from 0: $s_1 = 6314833.70194304$ m

We finally obtain

$$\varphi_1 = 54°13'15.2891670'' \quad \lambda_1 = 10°30'47.2427967''$$

and

$$\alpha_{1A} = 205°31'40.8621182''$$

confirming the example given by Grossmann (1976).

Complex computations

If we check the series expansions for Gauss-Krueger/UTM coordinate computation, we find that the meridian arc length is needed. Because we have to call the routine arc() anyway for a real number, we also call it for an appropriate complex number and thus use identical code for two different tasks.

We present here a method which uses the formulas given above for complex numbers. The algorithms below have been published by Klotz (1993).

Gauss-Krueger/UTM coordinates from latitude/longitude

Given the ellipsoidal latitude and longitude (φ, λ) of a point, we look for its Gauss-Krueger (GK) or UTM mapping plane coordinates (*Easting*, *Northing*) with the central meridian λ_0.

1. For ellipsoid with flattening f compute $e^2 = f\cdot(2 - f)$
2. From φ compute isometric latitude $q = ATANH(\sin\varphi) - e\cdot ATANH(e\cdot\sin\varphi)$
3. Form complex variable $w = q + i\cdot(\lambda - \lambda_0)$
4. Start complex number iteration with $b_0 = \arcsin(\tanh\ w)$

$$b_i = \arcsin(\tanh(w + e\cdot ATANH(e\cdot\sin b_{i-1})))$$

5. With scale factor for central meridian, scale = 1 or scale = 0.9996 compute

$$z = scale\cdot \mathrm{arc}(a, f, b)$$

6. Separate real and imaginary parts of complex number z

$$Easting = imag(z) \quad Northing = real(z)$$

Latitude/longitude from Gauss-Krueger/UTM coordinates

Given the GK or UTM mapping plane coordinates of a point with the central meridian λ_0, we look for its ellipsoidal latitude and longitude (φ, λ).

1. For ellipsoid with flattening f compute $e^2 = f\cdot(2 - f)$
2. Form complex variable $z = Northing + i\cdot Easting$
3. With appropriate scale factor compute

$$b = lat(a, f, z/scale)$$

4. Compute $w = ATANH(\sin b) - e\cdot ATANH(e\cdot\sin b)$
5. Split real and imaginary number of $w = q + i\cdot(\lambda - \lambda_0)$

$$q = real(w) \quad \text{and} \quad \Delta\lambda = \lambda - \lambda_0 = imag(w)$$

6. Start real number iteration with $\varphi_0 = 0$ and

$$\varphi_i = \arcsin(\tanh(q + e\cdot a\tanh(e\cdot\sin\varphi_{i-1})))$$

Examples have been built into the test routines of the published Java and C++ codes.

Acknowledgements The author acknowledges the valuable hints of his students in the testing phase of the algorithms in his lectures. The students showed that the major parts of the formulas even work on pocket calculators. Finally, the author wishes to acknowledge the valuable hints of the reviewers.

References

Grossmann W (1976) Geodätische Rechnungen und Abbildungen in der Landesvermessung. Wittwer, Stuttgart

Klotz J (1991) Eine analytische Lösung kanonischer Gleichungen der geodätischen Linie zur Transformation ellipsoidischer Flächenkoordinaten. DGK Series C, Nr. 385, Munich

Klotz J (1993) Eine analytische Lösung der Gauss-Krüger-Abbildung. Zeitschrift für Vermessungswesen (ZfV), pp 106–116

Snyder JP (1982) Map projections used by the U. S. Geological survey. Geological survey bulletin 1532. United states printing office, Washington

Thomas PD (1952) Conformal projections in geodesy and cartography. Special publication no. 251, US department of commerce