Computation of Meridian Arcs, Gauss-Krueger-, and UTM-Coordinates from Recursive Series

Prof. Dr.-Ing. Klaus Hehl | Forschungsschwerpunkt: Mathematical Geodesy and Satellite Geodesy

Kurzfassung
Zentraler Punkt dieses Beitrags ist die Diskussion von Rekursionsbeziehungen für die Berechnung einiger Standard-Integrale der Mathematischen Geodäsie. Verebnete, konforme Koordinaten erhält man über die Abbildungsgleichungen der Gauss-Krueger- und UTM-Abbildung. Es werden C++ Quellcodes vorgestellt und auf Java-Codes hingewiesen, die entsprechende Reihenentwicklungen umsetzen, deren Genauigkeit nicht mehr limitiert ist und die – wie die Studierenden im Fach „Landesvermessung" immer wieder zeigen – auch per Taschenrechner auszuwerten sind.
Alle Quellcodes (mit Zahlenbeispielen) sind unter den unten angegebenen Adressen verfügbar.

Abstract
This paper describes recursive series expansions of standard integrals in mathematical geodesy. Conformal mapping into the plane results in Gauss-Krueger and UTM coordinates. C++ source code is provided and also a link to Java code is given. Those routines implement the recursion formulas which are no more limited by accuracy and – as demonstrated by my students – can be evaluated even by using pocket calculators.
Corresponding C++ and Java source code, which also includes numerical examples, has been released as central part of [Heh 05] *and is available from* http://www.ngs.noaa.gov/gps-toolbox/Hehl1.htm *or from the author's website, which also provides online computation using PHP, at* http://hehl.tfh-berlin.de

Introduction

Modern Global Navigation Satellite Systems (GNSS: GPS, Glonass, Galileo) provide as primary result highly accurate cartesian coordinates in three-dimensional space. Those are conceptionally simple but not very helpful for practical work. Therefore most users prefer coordinates mapped onto a plane. Because of their properties maps based on Gauss-Krueger (GK) or UTM (Universal Transverse Mercator) projection are demanded in general. The computation of such coordinates requires the use of series expansions, c.f. [Gro76] or [Hec03]. Part of the GK/UTM computation is the determination of the socalled meridian arc length G – also a series expansion. The computation of Soldner coordinates using geodesics, and the computation of the direct and inderect problem on local, regional, and global scale is also implemented in the source codes.

All of the underlying integrals can be formulated as rapid convergent series in such way that the coefficients are computable *recursively*. This procedure bears the following advantages:

1. coefficients are calculated from the same relation for every degree,
2. we get *highest accuracy* by simply changing one parameter – the upper limit of a forloop,
3. *simple arithmetic* makes it possible to use the routines for both real number computation and complex number computation,
4. algorithms using the geodesic line as a connecting curve on the ellipsoid are no longer limited to a certain range or accuracy

More detailed discussions also concerning the geodesic line (GL) as shortest connecting line on a given ellipsoid of revolution is given in [Heh05]. The full theory is originally due to [Kl091, Kl093].

Computation of Meridian Arc Lengths

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 $f=\frac{a-b}{a}$ we relate β to the ellipsoidal latitude φ by the relation $\beta=(1-f)\cdot\tan\varphi$

Introducing the first eccentricity, $e^2=\frac{a^2-b^2}{a^2}$, we find the following closed expression for the length of the desired meridian arc G, as shown in Fig. 2:

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

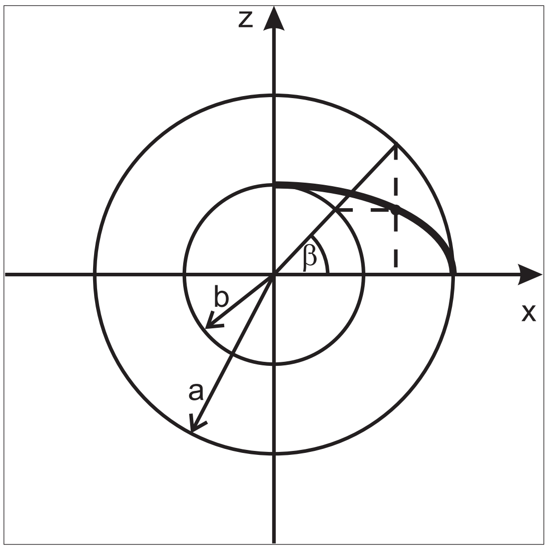

Fig. 1: The ellipse as an affine mapping of a circle

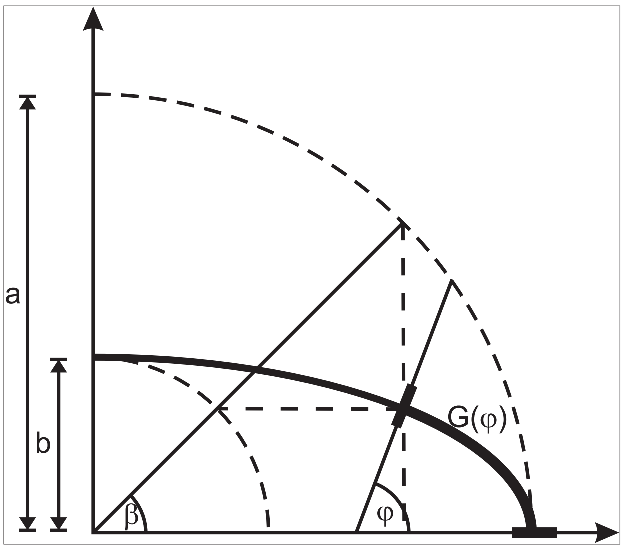

Fig. 1: An example meridian arc G

This relation needs numerical evaluation (e.g. using *computer algebra systems* such as 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 the writing of 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

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

are called *generalized binomial coefficients*.
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

$$\sqrt{1-e^2\cos^2 x}=(1-e^2\cos^2 x)^{\frac{1}{2}}$$

$$\sum_{n=0}^{\infty}\binom{0.5}{n}\cdot(-e^2\cos^2 x)^n=\binom{0.5}{0}-\binom{0.5}{1}\cdot(e^2\cos^2 x)^1+\binom{0.5}{2}\cdot(e^2\cos^2 x)^2\pm\ldots$$

where the first binomial coefficients are:

$$\mathrm{n}=0:\quad \binom{0.5}{0}=1$$

$$\mathrm{n}=1:\quad \binom{0.5}{1}=\frac{0.5}{1}=\frac{1}{2}\qquad\qquad =-\binom{0.5}{0}\cdot\frac{2\cdot 1-3}{2\cdot 1}$$

$$\mathrm{n}=2:\quad \binom{0.5}{2}=\frac{0.5\cdot(0.5-1)}{1\cdot 2}=-\frac{1}{8}\qquad =-\binom{0.5}{1}\cdot\frac{2\cdot 2-3}{2\cdot 2}$$

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 only would like to 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

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

$$k_0 = 1; \quad k_n = k_{n-1} \cdot \frac{2n}{2n+1} \cdot \cos^2 \beta$$

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

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

with the coefficients

$$koeff1 = \sum_{n=0}^{N} c_n = (c_0 + c_1 + \ldots + c_N)$$

$$koeff2 = \sum_{n=1}^{N} c_n \cdot \sum_{m=0}^{n-1} k_m = c_1 \cdot k_0 + c_2 \cdot (k_0 + k_1) + \ldots + c_N \cdot (k_0 + k_1 + \ldots + k_{N-1})$$

$$= k_0 \cdot (c_1 + c_2 + c_3 + \ldots + c_N) + k_1 \cdot (c_2 + c_3 + \ldots + c_N) + \ldots + k_{N-1} \cdot c_N$$

Algorithm

A suitable algorithm to calculate G is given below.

1. select order (N = 4, e.g., guarantees sub-millimeter accuracy), and ellipsoid (semimajor axis a, flattening $f = \frac{a-b}{a}$)
2. calculate the first numerical eccentricity $e^2 = f \cdot (2-f)$
3. calculate quantities *depending on ellipsoid*
   i. recursively calculate coefficients

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

   ii. form and store the $(N \times 1)$-vector

   $$\mathrm{k}_2 = \begin{pmatrix} c_1 + c_2 + c_3 + \ldots + c_N \\ c_2 + c_3 + \ldots + c_N \\ \vdots \\ c_N \end{pmatrix}$$

   iii. calculate coefficient
   $koeff1 = 1 + (c_1 + c_2 + c_3 + \ldots + c_n) = 1 + \mathbf{k}_2[0]$
   the symbol $\mathbf{k}_2[0]$ denotes the first element of the vector $\mathbf{k}_2$
4. calculate quantities *depending on the ellipsoidal latitude* φ
   i. calculate reduced latitude β from $\tan\beta = (1-f) \cdot \tan\varphi$,
   calculate $cc = \cos^2 \beta$
   ii. recursively calculate

   $$k_0 = 1; \quad k_n = k_{n-1} \cdot \frac{2n}{2n+1} \cdot cc$$

   iii. form and store the $(N \times 1)$-vector
   $\mathbf{k}^T = (k_0 \quad k_1 \quad \ldots \quad k_{N-1})$
   iv. calculate coefficient from scalar product
   $koeff2 = \mathbf{k}_2^T \cdot \mathbf{k}$
5. calculate length of meridian arc

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

Similar to the *Horner* scheme for the evaluation of polynomials simplified versions for *koeff1* and *koeff2* can be derived. Of course, for a given accuracy, truncated expressions can be given, e.g. for N = 4

$$koeff1 = 1 - \frac{1}{4} e^2 - \frac{3}{64} e^4 - \frac{15}{768} e^6 - \frac{525}{49152} e^8$$

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

$$2\pi \cdot a \cdot koeff1$$

C++ Code

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

```
//length of meridian arc G using recursion formulas
//(c) hehl@tfh-berlin.de  December 2005
//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 to 19. In the spirit of *object-oriented programming* it is advised to put the lines 1 to 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).

Computing the latitude β or φ, resp., from a given arc length G requires a simple iteration which is also implemented in the given code. Again: note that *koeff2* needs to be calculated repeatedly, *koeff1* and $\mathbf{k}_2$ only once.

Numerical Example to Verify the Algorithm

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

With coefficient *koeff1* = 0.9983172080559514, *koeff2* = -0.0016835008855936161, we finally obtain $G = G_8 = 4995775.138_{571393}$ m (having in mind that mm-accuracy is sufficient for geodetic practice).

We can also have a closer look at the convergence behaviour of the series expansion by examining the values of G for different orders N, see Tab. 2.

n	k2[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

Tab. 1: Contents of an example $\mathbf{k}_2$ vector

N	G_N [m]	$\Delta G = G_8 - G_N$ [m]	log | ΔG |	Δ 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

Tab. 2: Convergence of the example meridian arc

The second column gives the arc length as a function of N. Column three shows the difference between the 'correct' value ($N=8$) 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.

Complex Number and Real Number Use of Code

Especially powerful is the *dual use* of code using *C++ templates*. Below we present a source code (with rudimentary error checking in line 6; a, f, and b must be 'valid') 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 G,
- called with a complex number $\boldsymbol{b}$ (to be defined below, also in radians) and returns the two components of Gauss-Krueger- or (scaling with 0.9996) UTM coordinates as the real and imaginary parts, resp. of a complex number z.

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' (Linux/GNU)

```
typedef REAL double;                                    //00
template <class TYPE>                                   //01
TYPE arc(int N, REAL a, REAL f, TYPE b) {               //02
//hehl@tfh-berlin.de, December 2005                     //03
TYPE beta = atan((1.0-f)*tan(b));                       //04
REAL koeff1 = 1.0; TYPE koeff2 = 0.0;                   //05
if(N>=1 && N<=8) {                                      //06
 REAL c,e2;                                             //07
 TYPE 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 *= (n2-1.0)*(n2-3.0)/n2/n2*e2;                      //14
  k *= n2/(n2+1.0)*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++. For the computation of a great number of points, the previous – and more effective – code (using the $\mathbf{k}_2$-vector) can be re-written as a template in the same way as above.
Java in its most recent version 1.5 has also 'templates' available – they call it *generic functions*; the published Java codes will also be re-written accordingly.

Complex Computations

Looking closer at the classical series expansions for Gauss-Krueger/UTM coordinate computation we find that the meridian arc length G is needed, i.e. *arc(.)* has to be called anyway. It is wellknown that any algorithm for the computation of G (not only the recursive ones) can be applied to a complex latitude $\boldsymbol{b}$. This complex number $\boldsymbol{b}$ is derived from φ and $\Delta\lambda = \lambda - \lambda_0$ and vice versa according to the algorithms below. They have been published by, e.g. [Kl093]. Below, we symbolically denote the computation of the arc length from latitude by *arc(.)*, and the inverse problem, computation of latitude from arc length by *lat(.)*. Ultimate accuracy is assumed, e.g. N=8.

Gauss-Krueger or UTM Coordinates From Latitude and 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 the *isometric latitude* $q = atanh(\sin\varphi) - e \cdot atanh(e \cdot \sin\varphi)$
3. form complex variable $w = q + i \cdot (\lambda - \lambda_0)$, with the imaginary unit i
4. start complex number iteration (converges after 4-5 steps) 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 with the complex number $\boldsymbol{b}$ as third argument
   $z = scale \cdot arc(a, f, b)$
6. separate real and imaginary parts of complex number z
   Easting = imag(z) + zone ID +(500km- shift) *Northing = real(z)*

Latitude and Longitude from Gauss-Krueger or 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 $z = Northing + i \cdot Easting$ (without zone ID and 500km-shift)
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 $w = q + i \cdot (\lambda - \lambda_0)$
   $q = real(\boldsymbol{w})$ and $\Delta\lambda = \lambda - \lambda_0 = imag(w)$
6. start real number iteration with $\varphi_0 = 0$ and
   $\varphi_i = asin(\tanh q + e \cdot atan(e \cdot \sin\varphi_{i-1}))$

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

References

[Gro76] Grossmann, W. (1976): Geodätische Rechnungen und Abbildungen in der Landesvermessung. Wittwer-Verlag, Stuttgart.

[Hec03] Heck, B. (2003): Rechenverfahren und Auswertemodelle in der Landesvermessung. Wichmann-Verlag, Karlsruhe.

[Heh05] Hehl, K. (2005): C++ and Java Code for Recursion Formulas in Mathematical Geodesy. GPS Solutions, Springer-Verlag, Heidelberg, Vol. 9 (1), pp. 51-58

[Klo91] Klotz, J. (1991): Eine analytische Lösung kanonischer Gleichungen der geodätischen Linie zur Transformation ellipsoidischer Flächenkoordinaten. DGK Series C, Nr. 385, München.

[Klo93] Klotz, J. (1993): Eine analytische Lösung der Gauss-Krueger-Abbildung. Zeitschrift für Vermessungswesen (ZfV), pp. 106-116.

Kontakt

Prof. Dr.-Ing. Klaus Hehl
Technische Fachhochschule Berlin
FB III
Luxemburger Straße 10
13353 Berlin
Haus Bauwesen, Raum D 444
Tel.: 030-4504-2611
E-Mail: hehl@tfh-berlin.de
Web: http://hehl.tfh-berlin.de