NOAA Technical Report NESS 81

Improved Algorithm for Calculation of UTM and Geodetic Coordinates

Washington, D.C.
September 1980

U.S. DEPARTMENT OF COMMERCE
National Oceanic and Atmospheric Administration
National Environmental Satellite Service

NOAA TECHNICAL REPORTS

National Environmental Satellite Service Series

The National Environmental Satellite Service (NESS) is responsible for the establishment and operation of the environmental satellite systems of NOAA.

Publication of a report in NOAA Technical Report NESS series will not preclude later publication in an expanded or modified form in scientific journals. NESS series of NOAA Technical Reports is a continuation of, and retains the consecutive numbering sequence of, the former series, ESSA Technical Report National Environmental Satellite Center (NESC), and of the earlier series, Weather Bureau Meteorological Satellite Laboratory (MSL) Report. Reports 1 through 39 are listed in publication NESC 56 of this series.

Reports in the series are available from the National Technical Information Service (NTIS), U.S. Department of Commerce, Sills Bldg., 5285 Port Royal Road, Springfield, VA 22161, in paper copy or microfiche form. Order by accession number, when given, in parentheses. Beginning with 64, printed copies of the reports, if available, can be ordered through the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402. Prices given on request from the Superintendent of Documents or NTIS.

ESSA Technical Reports

NESC 46 Monthly and Seasonal Mean Global Charts of Brightness From ESSA 3 and ESSA 5 Digitized Pictures, February 1967-February 1968. V. Ray Taylor and Jay S. Winston, November 1968, 9 pp. plus 17 charts. (PB-180-717)

NESC 47 A Polynomial Representation of Carbon Dioxide and Water Vapor Transmission. William L. Smith, February 1969 (reprinted April 1971), 20 pp. (PB-183-296)

NESC 48 Statistical Estimation of the Atmosphere's Geopotential Height Distribution From Satellite Radiation Measurements. William L. Smith, February 1969, 29 pp. (PB-183-297)

NESC 49 Synoptic/Dynamic Diagnosis of a Developing Low-Level Cyclone and Its Satellite-Viewed Cloud Patterns. Harold J. Brodrick and E. Paul McClain, May 1969, 26 pp. (PB-184-612)

NESC 50 Estimating Maximum Wind Speed of Tropical Storms From High Resolution Infrared Data. L. F. Hubert, A. Timchalk, and S. Fritz, May 1969, 33 pp. (PB-184-611)

NESC 51 Application of Meteorological Satellite Data in Analysis and Forecasting. Ralph K. Anderson, Jerome P. Ashman, Fred Bittner, Golden R. Farr, Edward W. Ferguson, Vincent J. Oliver, Arthur H. Smith, James F. W. Purdom, and Rance W. Skidmore, March 1974 (reprint and revision of NESC 51, September 1969, and inclusion of Supplement, November 1971, and Supplement 2, March 1973), pp. 1--6C-18 plus references.

NESC 52 Data Reduction Processes for Spinning Flat-Plate Satellite-Borne Radiometers. Torrence H. MacDonald, July 1970, 37 pp. (COM-71-00132)

NESC 53 Archiving and Climatological Applications of Meteorological Satellite Data. John A. Leese, Arthur L. Booth, and Frederick A. Godshall, July 1970, pp. 1-1--5-8 plus references and appendixes A through D. (COM-71-00076)

NESC 54 Estimating Cloud Amount and Height From Satellite Infrared Radiation Data. P. Krishna Rao, July 1970, 11 pp. (PB-194-685)

NESC 56 Time-Longitude Sections of Tropical Cloudiness (December 1966-November 1967). J. M. Wallace, July 1970, 37 pp. (COM-71-00131)

NOAA Technical Reports

NESS 55 The Use of Satellite-Observed Cloud Patterns in Northern Hemisphere 500-mb Numerical Analysis. Roland E. Nagle and Christopher M. Hayden, April 1971, 25 pp. plus appendixes A, B, and C. (COM-73-50262)

NESS 57 Table of Scattering Function of Infrared Radiation for Water Clouds. Giichi Yamamoto, Masayuki Tanaka, and Shoji Asano, April 1971, 8 pp. plus tables. (COM-71-50312)

NESS 58 The Airborne ITPR Brassboard Experiment. W. L. Smith, D. T. Hilleary, E. C. Baldwin, W. Jacob, H. Jacobowitz, G. Nelson, S. Soules, and D. Q. Wark, March 1972, 74 pp. (COM-72-10557)

NESS 59 Temperature Sounding From Satellites. S. Fritz, D. Q. Wark, H. E. Fleming, W. L. Smith, H. Jacobowitz, D. T. Hilleary, and J. C. Alishouse, July 1972, 49 pp. (COM-72-50963)

NESS 60 Satellite Measurements of Aerosol Backscattered Radiation From the Nimbus F Earth Radiation Budget Experiment. H. Jacobowitz, W. L. Smith, and A. J. Drummond, August 1972, 9 pp. (COM-72-51031)

(Continued on inside back cover)

NOAA Technical Report NESS 81

Improved Algorithm for Calculation of UTM and Geodetic Coordinates

Jeff Dozier

Washington, D.C.
September 1980

U.S. DEPARTMENT OF COMMERCE
Philip M. Klutznick, Secretary

National Oceanic and Atmospheric Administration
Richard A. Frank, Administrator

National Environmental Satellite Service
David S. Johnson, Director

Contents

IMPROVED ALGORITHM FOR CALCULATION OF UTM AND GEODETIC COORDINATES

JEFF DOZIER

National Environmental Satellite Service
Environmental Sciences Group
(on leave from University of California, Santa Barbara)

ABSTRACT

Expression of the equations for a UTM (Gauss-Krüger) projection in terms of Jacobian elliptic functions, rather than their series expansions, allows the projection to be used over wider zones than the standard 6 deg strips, and thus makes it applicable to satellite data from the NOAA A-G series. An efficient iterative solution method for either UTM or geodetic coordinates is developed using a complex-arithmetic version of Newton's method. The method can be used for longitudes up to 90 deg from the central meridian.

1. Description of UTM projection

The UTM (Universal Transverse Mercator) projection is commonly used world-wide for the production of large-scale (i.e. $\geqslant$ 1:1,000,000) maps. The coordinate system is described in detail in many references (e.g. Richardus and Adler, 1972; Maling, 1973, U.S. Army, 1973). It is a Gauss-Krüger projection with the following standards and modifications:

(1) The world is divided, with minor exceptions, into 60 zones with principal meridians every 6 deg of longitude. The meridian for zone 1 is 177 deg W.

(2) Along the principal meridian the scale is set to 0.9996 instead of 1.0, in order to minimize the average error in the zone.

(3) To avoid the use of negative numbers, the easting coordinate is incremented by 5×10^5. Similarly, in the southern hemisphere, the northing coordinate is incremented by 10^7.

2. UTM equations in closed form

Description of the UTM projection coordinates requires elliptic functions and integrals. Normally these have been expressed as series expansions, and the existent calculations methods were designed for hand computation with extensive use of tables. The equivalent computer programs (e.g. Quinones, 1970a, 1970b) typically have code which is cumbersome and whose accuracy decreases markedly outside the 6 deg longitude zone.

The more formal descriptions of the Gauss-Krüger equations which have appeared in the literature, have presented the results in terms of series expansions (e.g. Redfearn, 1948; Lee, 1962, 1976). Unfortunately these converge slowly and are often not sufficiently accurate. In this paper I adapt Lee's (1976) closed form expressions for these equations to a reasonably efficient computer solution which is accurate to machine precision. The method can be used for all longitudes up to and including 90 deg from the central meridian.

2.1. Notation

a	semi-major axis of the ellipsoid
b	semi-minor axis of the ellipsoid
k	eccentricity of the ellipsoid $[k^2 = (a^2-c^2)/a^2]$
ϕ	geodetic latitude
ψ	isometric latitude $[\psi = \operatorname{arctanh}(\sin\phi) - k \operatorname{arctanh}(k \sin\phi)]$
λ	longitude, from principal meridian of projection

x,y	northing and easting coordinates of Gauss-Krüger projection
u,v	northing and easting coordinates of intermediate projection
ζ	$\psi+i\lambda$
z	$x+iy$
w	$u+iv$
m	parameter of elliptic functions [$m = k^2$]
sn,cn,dn	Jacobian elliptic functions
K,E	elliptic integrals of first and second kinds (complete when no argument is specified)
α,β,γ	parameters used in calculation of complete elliptic integrals

2.2. Gauss-Krüger equations

The general equation for a transverse mercator projection of a ellipsoid, expressed in terms of elliptic functions, is:

$$\zeta = \operatorname{arctanh}(\operatorname{sn} w) - k \operatorname{arctanh}(k \operatorname{sn} w) \tag{1}$$

where $\operatorname{sn} w$ is the Jacobian elliptic function $\operatorname{sn}(w|m)$ (see Abramowitz and Stegun, 1964, ch. 16; Byrd and Friedman, 1971). The Gauss-Krüger projection (one of a general class of transverse mercator projections) is defined in terms of w as:

$$\frac{z}{a} = (1-m) \int_0^w \operatorname{dn}^{-2} t \, dt$$

or

$$\frac{z}{a} = \mathrm{E}(w|m) - m \frac{\operatorname{sn} w \operatorname{cn} w}{\operatorname{dn} w} \tag{2}$$

where $\mathrm{E}(w|m)$ is an elliptic integral of the second kind.

2.3. Some useful approximations and identities

From the wealth of information about elliptic functions and integrals, I have made use of the following:

2.3.1. q-series approximations

Define the nome q as:

$$q = \exp\left[-\frac{\pi \mathrm{K}'}{\mathrm{K}}\right]$$

The quarter periods K, K′ are complete elliptic integrals of the first kind with parameters $m, 1-m$ respectively.

$$\mathrm{K} = \mathrm{K}(m) = \int_0^{\frac{\pi}{2}} (1-m \sin^2\theta)^{-\frac{1}{2}} \, d\theta$$

$$\mathrm{K}' = \mathrm{K}(1-m)$$

From these, a rapidly converging expression for $\operatorname{sn} w$ is:

$$\operatorname{sn} w = \frac{2\pi}{m^{\frac{1}{2}}\mathrm{K}} \sum_{n=0}^{\infty} \frac{q^{n+\frac{1}{2}}}{1-q^{2n+1}} \sin\left[(2n+1)\frac{\pi w}{2\mathrm{K}}\right]$$

Similar approximations exist for $\operatorname{cn} w$, $\operatorname{dn} w$ etc., as well as for an elliptic integral of the second kind:

$$E(w|m) = Z(w|m) + \frac{wE}{K}$$

E is the complete elliptic integral of the second kind, and $Z(w|m)$ is Jacobi's zeta function:

$$E = E(m) = \int_0^{\frac{\pi}{2}} (1-m\sin^2\theta)^{1/2}\, d\theta$$

$$Z(w|m) = \frac{2\pi}{K} \sum_{n=1}^{\infty} \frac{q^n}{1-q^{2n}} \sin\frac{n\pi w}{K}$$

2.3.2. Gauss' arithmetic-geometric mean methods

Start with the triple $(\alpha_0, \beta_0, \gamma_0)$. Then

$$\alpha_n = \frac{\alpha_{n-1}+\beta_{n-1}}{2}$$

$$\beta_n = \sqrt{\alpha_{n-1}\beta_{n-1}}$$

$$\gamma_n = \alpha_n{}^2 - \beta_n{}^2$$

until $\alpha_n = \beta_n$ to the desired accuracy.

To evaluate complete elliptic integrals, we start with (Bulirsch, 1965; Hart, *et al.*, 1968):

$$\alpha_0 = 1 \quad \beta_0 = \sqrt{1-m} \quad \gamma_0 = m$$

Then

$$K = K(m) = \frac{\pi}{2\alpha_n}$$

$$E = E(m) = K\left[1 - \sum_{j=0}^{n} 2^{j-1}\gamma_j\right]$$

The algorithm can be checked by Legendre's relation:

$$EK' + E'K - KK' = \frac{\pi}{2}$$

2.3.3. Identities

$$\mathrm{sn}^2 w + \mathrm{cn}^2 w = 1$$

$$m\,\mathrm{sn}^2 w + \mathrm{dn}^2 w = 1$$

2.3.4. Derivatives

$$\frac{d}{dw}\mathrm{sn}\,w = \mathrm{cn}\,w\,\mathrm{dn}\,w$$

$$\frac{d}{dw}\mathrm{cn}\,w = -\mathrm{sn}\,w\,\mathrm{dn}\,w$$

$$\frac{d}{dw}\mathrm{dn}\,w = -m\,\mathrm{sn}\,w\,\mathrm{cn}\,w$$

$$\frac{d}{dw}E(w|m) = \mathrm{dn}^2 w$$

3. Solution of UTM equations

We define the *forward transform* as solving for (x,y) given (ϕ,λ) and the *inverse transform* as the reverse process. The principal difficulty in the forward transform is solving equation (1) for w, given ζ. Likewise, the principal difficulty in the inverse transform is solving equation (2) for w, given z. Once w is known, neither the solution of (1) for ζ nor (2) for z presents any formidable difficulty. The q-series approximations in 2.3.1 typically converge to 10^{-18} accuracy in 10 or 11 iterations.

The numerical computation for the elliptic integrals, as described in section 2.3.2, need be performed only once for any ellipsoid.

There are two reasonable choices for solving (1) or (2) when w is unknown.

3.1. Infinite series method

The traditional method is to express the appropriate equation in its Taylor series expansion, and then to invert the power series (Van Orstrand, 1910). For example, Lee (1962) combines (1) and (2) to express $z = f(\zeta)$ as:

$$\frac{z}{a} = \zeta - \frac{1}{3!\,(1-m)}\zeta^3 + \frac{(5-m)}{5!\,(1-m)^3}\zeta^5 - \frac{(61+26m+m^2)}{7!\,(1-m)^5}\zeta^7 + \ldots \qquad (3)$$

The difficulties with this approach are that:

(1) No general equations for the coefficients in the series expansions of the Jacobian elliptic functions are known, so one must work them out from the Taylor series. The algebra can get difficult.

(2) Similarly, the formulas for inversion of a series become more and more cumbersome as the order increases.

(3) The series converges slowly; for example the 7th order term in (3) still has a magnitude exeeding 5 km at 45 deg N.

(4) It is therefore difficult to use the series method where our accuracy specifications are arbitrary, or where we might wish to use the Gauss-Krüger projection over a zone wider than 6 deg (Snyder, 1979). Therefore the traditional methods of calculating UTM coordinates are inappropriate for many kinds of satellite data, including those from the NOAA A-G series.

3.2. Iterative method

For these reasons I resort instead to an iterative scheme. This has the disadvantage of increasing the computation time, but gives us the capability to solve either the forward or inverse transform to the limit of our computing machine's precision. The method is accurate for all longitudes up to 90 deg away from the central meridian of the projection, although the projection itself is generally not suitable at more than 10 or 15 deg away. The choice of numerical methods is influenced by two factors:

(1) The equations are well-behaved and have analytic derivatives. Furthermore, the identities between the Jacobian elliptic functions allow the first derivatives to be easily evaluated as a by-product of the evaluations of the functions.

(2) For all cases we can select a very good initial guess by letting $k = 0$ and solving the equation for a spherical rather than ellipsoidal earth.

Under these circumstances Newton's method will give us very rapid convergence.

3.2.1. The forward transform

Rewrite (1) as:

$$f(w) = \operatorname{arctanh}(\operatorname{sn} w) - k \operatorname{arctanh}(k \operatorname{sn} w) - \zeta = 0$$

Then, using sections 2.3.3 and 2.3.4:

$$f'(w) = \frac{1-m}{\mathrm{cn}w\,\mathrm{dn}w}$$

By Newton's method (see Lanczos, 1957):

$$w_{n+1} = w_n - \frac{f(w_n)}{f'(w_n)}$$

The starting guess w_0 is found by letting $k = 0$. Then $m = 0$ and $\mathrm{sn}w = \sin w$, and:

$$w_0 = \arcsin(\tanh\zeta)$$

3.2.2. The inverse transform

Rewrite (2) as:

$$f(w) = \mathrm{E}(w|m) - m\,\frac{\mathrm{sn}w\,\mathrm{cn}w}{\mathrm{dn}w} - \frac{z}{a} = 0$$

Then

$$f'(w) = \mathrm{dn}^2 w - m\left[\mathrm{cn}^2 w - (1-m)\,\frac{\mathrm{sn}^2 w}{\mathrm{dn}^2 w}\right]$$

When $k = 0$, solution of (2) leads to:

$$\frac{z}{a} = \frac{w}{2} + \frac{\sin(2w)}{4}$$

To a first order approximation $\sin(2w) = 2w$, so our initial guess w_0 is:

$$w_0 = \frac{z}{a}$$

4. Subprograms

There are only two subprograms with which the user need interface. They are written in the C programming language (Kernighan and Ritchie, 1978) but would be easily translated into other languages, as the C code is relatively algorithmic. The routines are utmf and utminv, to calculate the forward and inverse transforms respectively. They are listed in the Appendix, along with the necessary supporting routines.

Arguments for utmf are:

```
utmf (lc, zone, lat, lon, north, east)
int              lc, *zone;
double  lat, lon, *north, *east;
```

The input variables are:

lc	ellipsoid code (see Table 1)
zone	UTM reference zone (if zone = 0, the routine will calculate the appropriate zone)
lat, lon	geodetic coordinates in radians (positive in northern, eastern hemispheres)

Output variables are:

north	northing
east	easting

For utminv, the arguments are in a different order:

```
utminv (hflag, lc, zone, north, east, lat, lon)
int              hflag, lc, zone;
double  north, east, *lat, *lon;
```

In this case zone must be non-zero, as it is part of the UTM coordinate specification; north and east are input variables and lat and lon are output variables. The variable hflag, when negative, specifies that the northing coordinate has been incremented by 10^7.

Table 1
Ellipsoid Options for UTM Routines

code	ellipsoid	major axis (m)	minor axis (m)	eccentricity
0	sphere of Int'l volume	6370949	6370949	.000000
1	International	6378388	6356912	.081992
2	Clarke 1866	6378206	6356584	.082272
3	Clarke 1880	6378249	6356515	.082483
4	Everest	6377276	6356075	.081473
5	Bessel	6377397	6356079	.081697
6	Modified Everest	6377304	6356103	.081473
7	Australian National	6378160	6356775	.081820
8	Airy	6377563	6356257	.081673
9	Modified Airy	6377342	6356036	.081673
10	Walbeck	6376896	6355835	.081207
11	Southeast Asia	6378155	6356773	.081813
12	Krasovskiy	6378245	6356863	.081813
13	GTDS Model	6378140	6356755	.081820

Acknowledgements

This work was initiated and completed while I was supported by a Senior Postdoctoral Research Associateship from the National Research Council, National Academy of Sciences. John P. Snyder of the U.S. Geological Survey provided several useful comments on the manuscript.

References

Abramowitz, M., and Stegun, I.A., eds., 1964, *Handbook of Mathematical Functions,* New York: Dover, 1046 pp.

Brent, R.P., 1973, *Algorithms for Minimization without Derivatives,* Englewood Cliffs, NJ: Prentice-Hall, 195 pp.

Bulirsch, R., 1965, Numerical calculation of elliptic integrals and elliptic functions, *Numerische Mathematik,* **7,** 78-90.

Byrd, P.F., and Friedman, M.D., 1971, *Handbook of Elliptic Integrals for Engineers and Scientists,* Berlin: Springer-Verlag, 358 pp.

Hart, J.F., *et al.,* 1968, *Computer Approximations,* New York: Wiley, 343 pp.

Kernighan, B.W., and Ritchie, D.M., 1978, *The C Programming Language,* Englewood Cliffs, NJ: Prentice-Hall, 228 pp.

Lanczos, C., 1957, *Applied Analysis,* Englewood Cliffs, NJ: Prentice-Hall, 539 pp.

Lee, L.P., 1962, The transverse mercator projection of the entire spheroid, *Empire Survey Review,* **16,** 208-217.

Lee, L.P., 1976, *Conformal Projections Based on Elliptic Functions,* Canadian Cartographer Monograph 16, Toronto: University of Toronto Press, 128 pp.

Maling, D.H., 1973, *Coordinate Systems and Map Projections,* London: George Philip and Son, 255 pp.

Quinones, J., 1970a, *UTM forward,* U.S. Army Topographic Command, Department of Computer Services, Computer Memorandum 945.

Quinones, J., 1970b, *UTM inverse,* U.S. Army Topographic Command, Department of Computer Services, Computer Memorandum 946.

Redfearn, J.C.B., 1948, Transverse mercator formulae, *Empire Survey Review,* **9,** 318-322.

Richardus, P., and Adler, R.K., 1972, *Map Projections for Geodesists, Cartographers and Geographers,* Amsterdam: North-Holland, 174 pp.

Snyder, J.P., 1979, Calculating map projections for the ellipsoid, *The American Cartographer,* **6,** 67-79.

U.S. Army, 1973, *Universal Transverse Mercator Grid,* TM 5-241-8.

Van Orstrand, C.E., 1910, Reversion of power series, *Philosophical Magazine,* **37,** 366-376.

APPENDIX

Here are listed the source codes for the routines in the UTM programs. The codes are in alphabetical order, but the structure of the routines is as follows:

1. The major subprograms
   - utmf: calculates UTM coordinates from geodetic
   - utminv: calculates geodetic coordinates from UTM
2. Miscellaneous routines related to the UTM projection
   - utmorig: longitude of principal meridian
   - utmzone: zone associated with a given longitude
3. Miscellaneous routines related to earth coordinates
   - geodet: geodetic latitude (ϕ) from isometric (ψ)
   - isomet: isometric latitude (ψ) from geodetic (ϕ)
   - sphcon: axes (a, c) and eccentricity (k) for a given ellipsoid
4. Some transcendental and complex functions
   - complex: elementary operations (place ahead of main)
   - atanh: inverse of hyperbolic tangent (real)
   - casin: complex arc sine
   - catan: complex arc tangent
   - catanh: complex arc tanh
   - cpowr: complex number to real power
   - csin: complex sine
   - ctanh: complex hyperbolic tangent
5. Elliptic integrals and elliptic functions
   - e_comp: complete elliptic integral of 1st or 2nd kind (K or E)
   - e2ci: complex elliptic integral of 2nd kind (with real parameter)
   - jsn: complex Jacobian elliptic function sn
   - agm1: Gauss' arithmetic-geometric mean from initial triple
   - jzeta: Jacobi's zeta function
   - jseries: general routine for q-series approximations
   - qjper: quarter periods K, K′ for Jacobian elliptic functions and other q-series approximations
6. Gauss-Krüger and transverse Mercator coordinates
   - gkfor: Gauss-Krüger coordinates z from isometric coordinates ζ
   - gkinv: ζ from z
   - gkf: Gauss-Krüger coordinates z from intermediate coordinates w
   - gk: Gauss-Krüger coordinates from w, sn$w$
   - tmw: intermediate coordinates w from isometric coordinates ζ
   - tmwi: intermediate coordinates w from Gauss-Krüger coordinates z
   - tmerc: isometric coordinates ζ from sn$w$
7. Miscellaneous routines
   - abend: terminates program with error message
   - cnewton: complex version of Newton's method
   - getpi: returns double precision value of π
   - macheps: returns machine epsilon
   - zerobr: finds a zero of an arbitrary real function (from Brent, 1973)

All of the routines are coded in the C programming language (Kernighan and Ritchie, 1978). This language is close enough to Algol to make translation into other languages relatively simple. The following conventions and notations may be somewhat peculiar:

1. C makes explicit the distinction between the value of a variable and its address. Only a variable whose *address* is passed to a subprogram may be changed by the subprogram. These have an asterisk (*), meaning "pointer to", prepended to the variable name in the subprogram. Variables passed by value may be changed in a subprogram, but these changes will not be passed back to the calling program. The prefix & to a variable in a calling program means "address of".
2. The notation
   i op= j;
   is the same as
   i = i op j;
3. The notation
   i++;
   applies to integers only and means
   i = i + 1;
   The ++ after the variable indicates that the incrementation takes place after the use of the variable in an operation.

1. abend

```
#define BELL   07
abend (msg, dump)
char *msg;
int  dump;
/*
 * function: abend
 *
 * purpose : terminate a program if an irrecoverable
 *         error occurs, and print explanatory message
 *         if (dump == 1)  cause a core dump also
 *
 * usage   : abend ("message to be printed",dump);
 */
{
      putchar (BELL);
      printf ("\n\tABEND: %s\n", msg);

      if (dump == 1) {
            printf ("\t(core dumped)\n");
            fclose (stdout);
            abort ();   /*  dump core for debugging  */
      }
      else
            exit ();
}
```

2. agm1

```
/*
      returns arithmetic-geometric mean of a, b
      also calculates sum 2^(k-1) c sub k
*/
double      agm1 (a, b, c, c2sum)
double      a, b, c, *c2sum;
{
      double      fabs(), macheps(), sqrt();
      double      sum, h, tk, eps;
      char        str[80];

      if (a*b <= 0.) {
            sprintf (str, "agm1:  a %e b %e", a, b);
            abend (str, 0);
      }

      sum = 5.e-1 * c;
      tk = 1;

      eps = macheps();

      while (fabs (a-b) > a*eps) {
            h = a;
            a = (a + b) / 2;
            b = sqrt (h * b);
            c = (a-b) * (a+b);
            sum += tk * c;
            tk *= 2;
      }

      *c2sum = sum;
      return (a);
}
```

3. atanh

```
/*
      inverse hyperbolic tangent
*/

double      atanh (x)
double      x;
{
      double      log(), hold;
      char        str[80];

      /* check range */
      hold = x * x;
      if (hold < 0.  ||  hold >= 1.) {
            sprintf (str, "atanh:  argument = %e", x);
            abend (str, 1);
      }

      return (5.e-1 * log ((1. + x) / (1. - x)));
}
```

4. casin

```
/*
      complex arcsin
*/
complex     casin (z)
complex     z;
{
      extern int        errno;

      complex     result;
```

```
    double      alpha, beta, xp, xm, y2, l, r;
    double      sqrt(), asin(), log();
    char        str[80];

    xp = z.re*z.re + 2.*z.re + 1.;
    xm = z.re*z.re - 2.*z.re + 1.;
    y2 = z.im * z.im;
    l = sqrt (xp + y2);
    if (errno == EDOM)
        abend ("casin: bad sqrt for l", 1);
    r = sqrt (xm + y2);
    if (errno == EDOM)
        abend ("casin: bad sqrt for r", 1);
    alpha = 5.e-1 * (l + r);
    beta = 5.e-1 * (l - r);

    /* correct for slight rounding error */
    if (fabs(beta) > 1. && fabs(beta) < 1.01)
        beta = l > r ? 1 : -1;

    result.re = asin (beta);
    if (errno == EDOM) {
        sprintf (str, "casin: bad asin() arg %.18e", beta);
        abend (str, 0);
    }
    result.im = log (alpha + sqrt ((alpha+1.) * (alpha-1.)));
    if (errno == EDOM)
        abend ("bad log or sqrt in casin", 1);
    if (z.im < 0.)
        result.im = -result.im;

    return (result);
}
```

5. catan

```
/*
    complex arctan
*/
complex   catan (z)
complex   z;
{
    complex   result;
    double    x, y, xsqysq, twoy;
    double    log(), atan2();

    extern int      errno;

    x = z.re;
    y = z.im;
    if (x == 0. && y == 1.)
        abend ("catan:  z^2 = -1", 0);

    xsqysq = x * x + y * y;
    twoy = 2. * y;
    result.re = 5.e-1 * atan2 (2.*x, 1.-xsqysq);
    result.im = 2.5e-1 * log ((xsqysq+twoy+1.) /
        (xsqysq-twoy+1.));
    if (errno == EDOM)
        abend ("catan: bad arg to log()", 0);

    return (result);
```

6. catanh

```
/*
    complex arctanh
    uses relation  arctanh z = -i arctan iz
*/
complex   catanh (z)
complex   z;
{
    complex   i, hold;
    complex   cmult(), catan();

    i.re = 0;
    i.im = 1;
    hold = cmult (i, z);
    i.im = -1;

    return (cmult (i, catan (hold)));
}
```

7. cnewton

```
/*
    complex version of Newton's method

    solves for complex zero of f(z)
    given initial guess z

    the function f (z, ofzp) returns f(z)
    and 1/f'(z) in ofzp
*/
complex   cnewton (z, f, tol)
complex   z, (*f)();
double    tol;
{
    complex   fz, ofzp, h, zold, cr, cmult(), csub();
    double    macheps(), cabs();
    double    eps, me;
    int   maxit;
    char      str[80];

    me = 2. * macheps();
    maxit = 50;
    zold = z;
    fz = (*f) (z, &ofzp);

    while (maxit--) {

        /* check if z is a root */
        if (cabs (fz) <= tol)  return (z);

        /* calculate new approximation */
        h = cmult (fz, ofzp);
        z = csub (z, h);

        /* see how close new approx. is  */
        eps = tol + me * cabs(z);
        cr = csub (z, zold);
        if (5.e-1 * cabs(cr) < eps)  return (z);

        zold = z;
        fz = (*f) (z, &ofzp);
    }

    /* if we get to here, we didn't converge */
```

```
        sprintf (str,
        "cnewton, no convergence: eps %e  .5 cr %e  |f(z)|  %e",
        eps, 5.e-1*cabs(cr), cabs(fz));
        abend (str, 0);
}
```

8. complex

```
typedef struct {
        double      re, im;
}
complex;
complex    cadd (z1, z2)
complex    z1, z2;
{
        complex    result;

        result.re = z1.re + z2.re;
        result.im = z1.im + z2.im;

        return (result);
}
complex    csub (z1, z2)
complex    z1, z2;
{
        complex    result;

        result.re = z1.re - z2.re;
        result.im = z1.im - z2.im;

        return (result);
}
complex    cmult (z1, z2)
complex    z1, z2;
{
        complex    result;

        result.re = z1.re * z2.re - z1.im * z2.im;
        result.im = z1.re * z2.im + z1.im * z2.re;

        return (result);
}
complex    cdiv (z1, z2)
complex    z1, z2;
{
        double     denom;
        complex    result;

        denom = z2.re * z2.re + z2.im * z2.im;
        result.re = (z1.re * z2.re + z1.im * z2.im) / denom;
        result.im = (z2.re * z1.im - z1.re * z2.im) / denom;

        return (result);
}
double     carg (z)
complex    z;
{
        double     atan2();

        return (atan2 (z.im, z.re));
}
complex    conjg (z)
complex    z;
{
        complex    result;

        result.re = z.re;
        result.im = -z.im;

        return (result);
}
```

9. cpowr

```
/*
        complex number to a real power
*/
complex    cpowr (z, p)
complex    z;
double     p;
{
        extern int         errno;
        complex    result;
        double     pow(), carg(), cabs(), r, nt, cos(), sin(), rn;

        r = cabs (z);
        rn = pow (r, p);
        if (errno == ERANGE)
                abend ("overflow in cpowr", 0);
        if (errno == EDOM)
                abend ("bad arg to pow()", 1);
        nt = carg (z);
        nt *= p;

        result.re = rn * cos (nt);
        result.im = rn * sin (nt);
        return (result);
}
```

10. csin

```
/*
        complex sin
*/
complex    csin (z)
complex    z;
{
        complex    result;
        double     x, y, sin(), cos(), sinh(), cosh();

        x = z.re;
        y = z.im;
        result.re = sin(x) * cosh(y);
        result.im = cos(x) * sinh(y);

        return (result);
}
```

11. ctanh

```
/*
        complex tanh
*/
complex    ctanh (z)
complex    z;
{
        complex    result;
        double     sin(), cos(), sinh(), cosh();
        double     denom, twox, twoy;

        twox = 2. * z.re;
```

```
    twoy = 2. * z.im;
    denom = cosh (twox) + cos (twoy);
    result.re = sinh (twox) / denom;
    result.im = sin (twoy) / denom;

    return (result);
}
```

12. e2ci

```
/*
    complex elliptic integral of 2nd kind
*/
complex   e2ci (m, w)
double    m;
complex   w;
{
    complex   jzeta(), cadd(), z;
    double    k, kp, ek, e_comp(), q;

    static double    mlast, e;

    qjper (m, &k, &kp, &q);
    z = jzeta (k, q, w);

    if (m != mlast || m == 0.) {
        e = e_comp (m, 2);
        mlast = m;
    }

    ek = e / k;
    w.re *= ek;
    w.im *= ek;

    return (cadd (z, w));
}
```

13. e_comp

```
/*
    complete elliptic integral
    parameter m
    kind 1 or 2
*/
double    e_comp (m, kind)
double    m;
int       kind;
{
    /* uses a.g.m. method*/
    double    sqrt(), agm1();
    double    b, c, pi;

    static double    mlast, a, sum;

    getpi (&pi);

    if (kind != 1 && kind != 2)
        abend ("e_comp:  kind must be 1 or 2", 0);

    if (m == 1.) {
        if (kind == 1)
            abend ("e_comp: 1st kind, m = 1", 0);
        return (1.);
    }

    if (m == 0.)  return (pi/2);

    if (m != mlast) {
        mlast = m;
        c = sqrt (m);
        a = 1;
        b = sqrt ((1.+c) * (1.-c));
        c = m;
        a = agm1 (a, b, c, &sum);
    }

    return (kind == 1 ? pi/(2.*a): (pi/(2.*a)) * (1.-sum));
}
```

14. geodet

```
/*
    finds latitude as function of isometric latitude
    and eccentricity
*/

double    psi_trn, k_tran;

double    geodet (psi, k)
double    psi, k;
{
    double    zerobr(), isomd(), tanh(), asin();
    double    phi, pi2, tol;
    char  str[80];

    extern double    psi_trn, k_tran;
    extern int       errno;

    /* first check the special case k = 0  */
    if (k == 0) {
        phi = asin (tanh (psi));
        if (errno == EDOM) {
            sprintf (str, "geodet: asin arg > 1, psi %f", psi);
            abend (str, 1);
        }
        return (phi);
    }

    psi_trn = psi;
    k_tran = k;
    getpi (&pi2);
    pi2 /= 2;
    tol = 0;

    if (psi == 0.)  return (0.);

    else if (psi > 0.)
        phi = zerobr (0., pi2, tol, isomd);

    else
        phi = zerobr (-pi2, 0., tol, isomd);

    return (phi);
}

double    isomd (phi)
double    phi;
{
    extern double    psi_trn, k_tran;
```

```
    double      isomet();

    return (psi_trn − isomet (phi, k_tran));
}
```

15. getpi

```
/* sets double precision value of pi

    Usage: getpi (&pi)
*/
getpi(x)
double    *x;
{
    static int          first;
    static double       holdpi;
    double              atan();

    if (first != 1) {
        holdpi = 4.0 * atan(1.0);
        first = 1;
    }

    *x = holdpi;
}
```

16. gk

```
/*
    calculates G-K coordinates from w, sn w
*/
complex   gk (m, w, snw)
double    m;
complex   w, snw;
{
    complex   cdw, cadd(), cmult(), csub(), e2ci(), cpowr(),
        t1, t2, one, num, denom, ms;

    t1 = e2ci (m, w);

    one.re = 1;
    one.im = 0;
    num = cmult (csub (one, snw), cadd (one, snw));
    ms.re = m * snw.re;
    ms.im = m * snw.im;
    denom = cmult (csub (one, ms), cadd (one, ms));
    cdw = cpowr (cdiv (num, denom), 5.e−1);

    t2 = cmult (snw, cdw);
    t2.re *= m;
    t2.im *= m;

    return (csub (t1, t2));
}
```

17. gkf

```
/*
    calculates G-K coordinates from intermediate coordinates
*/
complex   gkf (a, m, w)
double    a, m;
complex   w;
{
    complex   gk(), jsn(), snw, z;

    snw = jsn (m, w);
    z = gk (m, w, snw);

    z.re *= a;
    z.im *= a;

    return (z);
}
```

18. gkfor

```
/*
    finds Gauss-Krüger coordinates given psi, lon
*/
gkfor (k, a, psi, lon, x, y)
double    k, a, psi, lon, *x, *y;
/*
    k     eccentricity
    a     semi-major axis
    psi   isometric latitude
    lon   longitude (from principal meridian)
    x, y  northing, easting
*/
{
    complex   zeta, z, w, tmw(), gkf();

    zeta.re = psi;
    zeta.im = lon;

    /* find w */
    w = tmw (k, zeta);

    /* now calculate z */
    z = gkf (a, k*k, w);

    *x = z.re;
    *y = z.im;
}
```

19. gkinv

```
/*
    finds psi, lon given Gauss-Krüger coordinates
*/

gkinv (k, a, x, y, psi, lon)
double    k, a, x, y, *psi, *lon;
/*
    see gkfor() for explanation of variables
*/
{
    complex   zeta, z, w, snw;
    complex   jsn(), tmwi(), tmerc();

    z.re = x / a;
    z.im = y / a;

    /* find w */
    w = tmwi (k, z);

    /* now calculate zeta */
    snw = jsn (k*k, w);
    zeta = tmerc (k, snw);

    *psi = zeta.re;
```

```
        *lon = zeta.im;
}
```

20. isomet

```
/*
        finds isometric latitude as a function of latitude and
        eccentricity
*/

double      isomet (lat, eccen)
double      lat, eccen;
{
        double      x, atanh(), sin(), pi2, fabs();
        char        str[80];

        /* check range of arguments */
        if (eccen < 0. || eccen >= 1.) {
                sprintf (str, "isomet: eccentricity = %e",
                        eccen);
                abend (str, 1);
        }

        getpi (&pi2);
        pi2 /= 2;
        if (fabs (lat) > pi2) {
                sprintf (str, "isomet: latitude %.16f", lat);
                abend (str, 1);
        }

        x = sin (lat);
        return (atanh (x) - eccen * atanh (eccen * x));
}
```

21. jseries

```
/*
        expands an arbitrary complex series (of the type
        found in Jacobian elliptic functions) until con-
        vergence
*/
complex   jseries       (start, q, v, qf, vf)
int   start;
double      q, (*qf)();
complex   v, (*vf)();
/*
        start  = 0 or 1
        q      = real parameter for real qf()
        v      = complex parameter for complex vf()
*/
{
        int   n;
        double      h, aold, anew;
        complex   z, x, cadd();

        n = start;
        anew = z.re = z.im = 0;
        aold = 999;

        while (aold != anew) {
                h = (*qf) (n, q);
                x = (*vf) (n, v);
                n++;
                x.re *= h;
                x.im *= h;
                aold = anew;
                z = cadd (z, x);
                anew = z.re * z.re + z.im * z.im;
        }
        return (z);
}
```

22. jsn

```
/*
        complex Jacobian elliptic function sn u
*/
complex   jsn (m, u)
double      m;
complex   u;
{
        double      pi, k, kp, q, fact, qsnf(), sqrt();
        complex   v, sn;
        complex   vsnf(), jseries(), csin(), ctanh();

        if (m == 0.)
                return (csin (u));

        if (m == 1.0)
                return (ctanh (u));

        qjper (m, &k, &kp, &q);
        getpi (&pi);
        fact = pi / (2.*k);
        v.re = fact * u.re;
        v.im = fact * u.im;
        sn = jseries (0, q, v, qsnf, vsnf);
        fact = 2. * pi / (sqrt(m) * k);

        sn.re *= fact;
        sn.im *= fact;
        return (sn);
}

/*
        coefficients for q-series expansion of sn
*/
double      qsnf (n, q)
int   n;
double      q;
{
        double      np, dp, pow();

        np = n;
        np += 5.e-1;
        dp = 2*n + 1;

        return (pow(q,np) / (1. - pow(q,dp)));
}

/*
        argument for q-series expansion of sn
*/
complex   vsnf (n, v)
int   n;
complex   v;
{
        complex   csin(), arg;
        double      d;
```

```
        d = 2 * n + 1;
        arg.re = v.re * d;
        arg.im = v.im * d;

        return (csin (arg));
}
```

23. jzeta

```
/*
        Jacobi's zeta function
*/
complex   jzeta (k, q, w)
double    k, q;
complex   w;
{
        double      pi, exp(), qfzeta();
        complex     v, h, vfzeta(), jseries();

        if (k == 0.)
                h.re = h.im = 0;

        else {
                getpi (&pi);
                v.re = w.re * pi /k;
                v.im = w.im * pi / k;
                h = jseries (1, q, v, qfzeta, vfzeta);
                h.re *= 2 * pi / k;
                h.im *= 2 * pi / k;

        }

        return (h);
}

/*
        coefficients for sin series for jzeta()
*/
double      qfzeta (n, q)
int     n;
double      q;
{
        double      qp, pow();

        qp = pow (q, (double) n);
        return (qp / (1. - qp*qp));
}

/*
        argument for sin-series for jzeta()
*/
complex   vfzeta (n, v)
int     n;
complex   v;
{
        complex     csin();

        v.re *= n;
        v.im *= n;

        return (csin (v));
}
```

24. macheps

```
double macheps()
/*
        returns machine epsilon
*/
{
        double      s, one;
        static double       x;
        static int      first;

        if (first != 1) {

                first = 1;
                one = 1;

                x = 1;
                s = 2;

                while (s > one) {
                        x *= 5.e-1;
                        s = one + x;
                }
        }

        return (x);
}
```

25. qjper

```
/*
        calculates quarter periods K & K' and nome q
        for Jacobian elliptic functions
*/
qjper (m, k, kp, q)
double      m, *k, *kp, *q;
{
        static double       mlast, klast, kplast, qlast;

        double      sqrt(), e_comp (), exp();
        double      pi, ms;

        getpi (&pi);

        if (m == 0.) {
                *k = pi / 2.;
                *q = 0;
                *kp = 1.e23;
        }

        else {
                if (mlast == m) {
                        *k = klast;
                        *kp = kplast;
                        *q = qlast;
                }

                else {
                        mlast = m;
                        *k = klast = e_comp (m, 1);
                        ms = sqrt (m);
                        *kp = kplast = e_comp ((1-ms) * (1.+ms), 1);
                        *q = qlast = exp (-pi * kplast / klast);
                }
        }
```

```
}
```

26. sphcon

See Table 1.

```
/*
      information about specific ellipsoids
*/

sphcon (l, major, minor, eccen)
int   l;
double      *major, *minor, *eccen;
/*
      code for ellipsoid choices (l -- input)

      0     Sphere of equal volume
      1     International
      2     Clarke 1866
      3     Clarke 1880
      4     Everest
      5     Bessel
      6     Modified Everest
      7     Australian National
      8     Airy
      9     Modified Airy
      10    Walbeck
      11    Southeast Asia
      12    Krasovskiy
      13    GTDS Model

      (output)
      major       semi-major axis
      minor       semi-minor axis
      eccen[defined as sqrt ((a*a - b*b) / (a*a))]
*/
{
      char        str[80];
      double      eccensq, sqrt();
/*
      in some of the values that follow,
      the minor axis is computed
      from the flattening
*/
      switch (l) {

      case 0:
            *major = *minor = 6.370949;
            break;

      case 1:
            *major = 6.378388;
            *minor = 6.3569119461;
            break;

      case 2:
            *major = 6.3782064;
            *minor = 6.3565838;
            break;

      case 3:
            *major = 6.378249145;
            *minor = *major * (1. - 1./2.93465e2);
            break;

      case 4:
            *major = 6.3772763452;
            *minor = 6.3560754133;
            break;

      case 5:
            *major = 6.377397155;
            *minor = 6.3560789628;
            break;

      case 6:
            *major = 6.377304063;
            *minor = *major * (1. - 1./3.008017e2);
            break;

      case 7:
            *major = 6.37816;
            *minor = *major * (1. - 1./2.9825e2);
            break;

      case 8:
            *major = 6.377563396;
            *minor = *major * (1. - 1./2.99324965e2);
            break;

      case 9:
            *major = 6.37734189;
            *minor = *major * (1. - 1./2.99324959e2);
            break;

      case 10:
            *major = 6.376896;
            *minor = 6.3558348467;
            break;

      case 11:
            *major = 6.378155;
            *minor = 6.3567733205;
            break;

      case 12:
            *major = 6.378245;
            *minor = 6.3568630188;
            break;

      case 13:
            *major = 6.37814;
            *minor = 6.356754786253143;
            break;

      default:
            sprintf (str, "sphcon: ellipsoid code %d unimplemented", l);
            abend (str, 0);
      }

      eccensq = (*major - *minor) * (*major + *minor) /
            (*major * *major);
      if (eccensq < 0) {
            sprintf (str, "sphcon - e sq %f, maj %f, min %f",
            eccensq, *major, *minor);
            abend (str, 1);
      }
      *major *= 1.e6;
      *minor *= 1.e6;
      *eccen = sqrt (eccensq);
}
```

27. tmerc

```
/*
      returns transverse mercator coordinates
      as function of sn w
*/
complex   tmerc (k, snw)
double    k;
complex   snw;
{
      complex   ksnw, catanh(), csub(), t1, t2;

      ksnw.re = k * snw.re;
      ksnw.im = k * snw.im;

      t1 = catanh (snw);
      t2 = catanh (ksnw);
      t2.re *= k;
      t2.im *= k;

      return (csub (t1, t2));
}
```

28. tmw

```
/*
      iteratively finds w as function of k, zeta
*/
complex   z_tran;
double    k_tran;

complex   tmw (k, zeta)
double    k;
complex   zeta;
{
      extern complex z_tran;
      extern double  k_tran;

      complex   cnewton(), tmfd(), w, casin(), ctanh();
      double    tol, pi2;
      double    sqrt(), macheps();
      double    fabs();

      tol = sqrt (macheps());
      getpi (&pi2);
      pi2 /= 2;

      /* initial guess */
      w = casin (ctanh (zeta));

      /* check for quadrant compatibility */
      if (fabs (zeta.im) > pi2)
            abend ("tmw: long. diff > pi/2", 0);

      k_tran = k;
      z_tran = zeta;

      w = cnewton (w, tmfd, tol);

      return (w);
}

/*
      returns difference between transverse mercator
      intermediate coordinates and zeta
      returns (in ofpw) 1/f'
*/
complex   tmfd (w, ofpw)
complex   w, *ofpw;
{
      extern complex z_tran;
      extern double  k_tran;

      complex   snw, fw, cnw, dnw, ms, num, denom,
            one;

      complex   jsn(), cdiv(), cmult(), cadd(), cpowr(),
            tmerc();

      double    m;

      one.re = 1;
      one.im = 0;
      m = k_tran * k_tran;
      snw = jsn (m, w);
      fw = csub (tmerc (k_tran, snw), z_tran);

      cnw = cpowr (cmult (csub (one, snw), cadd (one, snw)), 5.e-1);
      ms = snw;
      ms.re *= m;
      ms.im *= m;
      dnw = cpowr (cmult (csub (one, ms), cadd (one, ms)), 5.e-1);

      num = cmult (cnw, dnw);
      denom.re = 1. - m;
      denom.im = 0;
      *ofpw = cdiv (num, denom);

      return (fw);
}
```

29. tmwi

```
/*
      iteratively finds w as function of k, z
*/

complex   z_tran;
double    k_tran;

complex   tmwi (k, z)
double    k;
complex   z;
{
      extern complex z_tran;
      extern double  k_tran;

      complex   cnewton(), casin(), tmid();
      complex   w;
      double    pi2, tol;
      double    sqrt(), macheps();

      tol = sqrt (macheps());
      getpi (&pi2);
      pi2 /= 2;

      k_tran = k;
      z_tran = z;

      /* initial guess */
```

```
    w = z;

    w = cnewton (w, tmid, tol);

    return (w);
}

/*
    retturns difference between transverse mercator
    intermediate coordinates and z

    returns (in ofpw) reciprocal of first derivative
*/
complex   tmid (w, ofpw)
complex   w, *ofpw;
{
    extern complex  z_tran;
    extern double   k_tran;

    complex   snw, cn2w, dn2w, hold , ms, fw, one;
    complex   jsn(), csub(), gk(), cmult();
    double    m, mp;

    m = k_tran * k_tran;
    mp = (1.+k_tran) * (1.-k_tran);
    snw = jsn (m, w);

    fw = csub (gk (m, w, snw), z_tran);

    /* calculate derivative  */
    snw = cmult (snw, snw);
    one.re = 1;
    one.im = 0;
    cn2w = csub (one, snw);
    ms = snw;
    ms.re *= m;
    ms.im *= m;
    dn2w = csub (one, ms);
    hold = cdiv (snw, dn2w);
    hold.re *= mp;
    hold.im *= mp;
    hold = csub (cn2w, hold);
    hold.re *= m;
    hold.im *= m;
    hold = csub (dn2w, hold);
    *ofpw = cdiv (one, hold);

    return (fw);
}
```

30. utmf

```
/*
    computes the UTM coordinates (major zone)
    for a point whose lat-long coordinates are known
*/

#define N_FALSE    1.e7
#define E_FALSE    5.e5
#define SCALE_F    9.996e-1

utmf (l, lat, lon, zone, north, east)
int    l, *zone;
double     lat, lon, *north, *east;
/*
    variables:

    (input)
    l     ellipsoid code [see sphcon() for tabulation]
    zone (if != 0)  major zone
    lat   latitude in radians (+ in north hem)
    lon   longitude in radians (+ in east hem)

    (output)
    zone major zone (if == 0 on input)
    northnorthing
    east  easting
*/
{
    double     eccen,          /* eccentricity of ellipsoid */
               major,          /* semi-major axis (m)       */
               orig,           /* longitude origin of zone */
               minor,          /* minor axis, not used      */
               isomet(),
               utmorig();

    int    utmzone(), abs();

    sphcon (l, &major, &minor, &eccen);
    if (*zone == 0)  *zone = utmzone (lon);
    orig = utmorig (*zone);

    gkfor (eccen, major, isomet (lat, eccen), lon-orig, north, east);

    *north *= SCALE_F;
    *east  *= SCALE_F;

    if (lat < 0.)  *north += N_FALSE;
    *east += E_FALSE;
}
```

31. utminv

```
/*
    computes geodetic coordinates for a point
    whose UTM coordinates are known
*/

#define N_FALSE    1.e7
#define E_FALSE    5.e5
#define SCALE_F    9.996e-1

utminv (hflag, l, zone, north, east, lat, lon)
int    hflag, l, zone;
double     north, east, *lat, *lon;
/*
    variables:

    (input)
    hflag (if negative) in so. hem., add N_FALSE to northing
    l     ellipsoid code [see sphcon() for tabulation]
    zone (if != 0)  major zone
    lat   latitude in radians (+ in north hem)
    lon   longitude in radians (+ in east hem)

    (output)
    zone major zone (if == 0 on input)
    northnorthing
    east  easting
*/
```

```
        double      eccen,          /* eccentricity of ellipsoid */
                major,          /* semi-major axis (m)      */
                orig,       /* longitude origin of zone */
                minor,          /* minor axis, not used     */
                psi,        /* isometric latitude   */
                dlon,       /* longitude difference     */
                geodet(),
                utmorig();

        int     utmzone(), abs();

        sphcon (l, &major, &minor, &eccen);
        orig = utmorig (zone);
        east -= E_FALSE;
        if (hflag < 0)
            north -= N_FALSE;
        east /= SCALE_F;
        north /= SCALE_F;

        gkinv (eccen, major, north, east, &psi, &dlon);

        *lon = dlon + orig;
        *lat = geodet (psi, eccen);
}
```

32. utmorig

```
/*
        returns origin (in radians) of designated UTM zone
*/

double      utmorig (zone)
int     zone;
{
        double      d, pi;

        /* check to see if zone reasonable */
        if (zone < 0 || zone > 60)
            abend ("utmorig - zone value unreasonable", 1);

        getpi (&pi);
        d = zone * 6 - 183;
        return (d * pi / 1.8e2);
}
```

33. utmzone

```
/*
        returned value is UTM zone
        corresponding to lon (radians)

*/

int     utmzone (lon)
double      lon;
{
        int     geod[3], z, i;
        double      pi, fabs();

        getpi (&pi);
        /* check range */
        if (fabs (lon) > pi)
            abend ("utmzone -- abs (lon) > pi", 0);
        rdms (lon, geod);
        z = geod[0] / 6;

        /* set i = 0 if on zone boundary */
        i =  geod[0]%6 + geod[1] + geod[2] == 0 ?  0 : 1;

        if (lon < 0)  z += 31 - i;
        else z += 30 + i;
        return (z);
}
```

34. zerobr

```
/*
        finds zero of a function by Brent's algorithm
        (from Brent, 1973)
*/
double      zerobr (a, b, t, f)
double      a, b, t, (*f)();
{
        double      c, d, e, fa, fb, fc, tol, m, p, q, r, s, eps;
        double      macheps();
        double      fabs(), log();
        int     maxfun;
        char        str[80];

        eps = 2 * macheps();

        /* compute max number of function evaluations */
        if (a == b)
            abend ("zerobr():  a = b", 0);
        fb = fabs (b) >= fabs (a) ? fabs (b) : fabs (a);
        tol = 5.e-1 * t + 2 * eps * fb;
        s = log (fabs (b - a) / tol) / log (2.);
        maxfun = s * s + 1;

        fa = (*f)(a);
        fc = fb = (*f)(b);
        if (fb == 0)  return (b);
        if (fa == 0)  return (a);
        if (fb*fa > 0)
            abend ("zerobr(): root not spanned", 1);

        while (maxfun--) {

            if ((fb > 0 && fc > 0)  ||  (fb <= 0  && fc <= 0))  {
                c = a;
                fc = fa;
                d = e = b - a;
            }

            if (fabs(fc) < fabs(fb)) {
                a = b;
                b = c;
                c = a;
                fa = fb;
                fb = fc;
                fc = fa;
            }

            tol = eps * fabs(b) + t;
            m = (c - b) / 2;

            if (fabs(m) < tol  ||  fb == 0)  return (b);
```

```
        /* see if bisection is forced */
        if (fabs(e) < tol || fabs(fa) <= fabs(fb))  d = e = m;

        else {
                s = fb/fa;

                if (a == c) {    /* linear interpolation */
                        p = 2 * m * s;
                        q = 1 - s;
                }

                else {          /* inverse quadratic interpolation */
                        q = fa/fc;
                        r = fb/fc;
                        p = s * (2 * m * q * (q-r) - (b-a) * (r-1));
                        q -= 1;
                        q *= (r-1) * (s-1);
                }

                if (p > 0)  q = -q;
                else        p = -p;

                s = e;
                e = d;

                if (2*p < 3*m*q - fabs(tol*q) && p < fabs(s*q/2))
                        d = p/q;
                else  d = e = m;
        }

        a = b;
        fa = fb;

        if (fabs(d) > tol)      b += d;
        else if (m > 0)         b += tol;
        else                    b -= tol;

        fb = (*f)(b);
}
abend ("zerobr(): did not converge",1);
```

(Continued from inside front cover)

NESS 61 The Measurement of Atmospheric Transmittance From Sun and Sky With an Infrared Vertical Sounder. W. L. Smith and H. B. Howell, September 1972, 16 pp. (COM-73-50020)

NESS 62 Proposed Calibration Target for the Visible Channel of a Satellite Radiometer. K. L. Coulson and H. Jacobowitz, October 1972, 27 pp. (COM-73-10143)

NESS 63 Verification of Operational SIRS B Temperature Retrievals. Harold J. Brodrick and Christopher M. Hayden, December 1972, 26 pp. (COM-73-50279)

NESS 64 Radiometric Techniques for Observing the Atmosphere From Aircraft. William L. Smith and Warren J. Jacob, January 1973, 12 pp. (COM-73-50376)

NESS 65 Satellite Infrared Soundings From NOAA Spacecraft. L. M. McMillin, D. Q. Wark, J.M. Siomkajlo, P. G. Abel, A. Werbowetzki, L. A. Lauritson, J. A. Pritchard, D. S. Crosby, H. M. Woolf, R. C. Luebbe, M. P. Weinreb, H. E. Fleming, F. E. Bittner, and C. M. Hayden, September 1973, 112 pp. (COM-73-50936/6AS)

NESS 66 Effects of Aerosols on the Determination of the Temperature of the Earth's Surface From Radiance Measurements at 11.2 m. H. Jacobowitz and K. L. Coulson, September 1973, 18 pp. (COM-74-50013)

NESS 67 Vertical Resolution of Temperature Profiles for High Resolution Infrared Radiation Sounder (HIRS). Y. M. Chen, H. M. Woolf, and W. L. Smith, January 1974, 14 pp. (COM-74-50230)

NESS 68 Dependence of Antenna Temperature on the Polarization of Emitted Radiation for a Scanning Microwave Radiometer. Norman C. Grody, January 1974, 11 pp. (COM-74-50431/AS)

NESS 69 An Evaluation of May 1971 Satellite-Derived Sea Surface Temperatures for the Southern Hemisphere. P. Krishna Rao, April 1974, 13 pp. (COM-74-50643/AS)

NESS 70 Compatibility of Low-Cloud Vectors and Rawins for Synoptic Scale Analysis. L. F. Hubert and L. F. Whitney, Jr., October 1974, 26 pp. (COM-75-50065/AS)

NESS 71 An Intercomparison of Meteorological Parameters Derived From Radiosonde and Satellite Vertical Temperature Cross Sections. W. L. Smith and H. M. Woolf, November 1974, 13 pp. (COM-75-10432)

NESS 72 An Intercomparison of Radiosonde and Satellite-Derived Cross Sections During the AMTEX. W. C. Shen, W. L. Smith, and H. M. Woolf, February 1975, 18 pp. (COM-75-10439/AS)

NESS 73 Evaluation of a Balanced 300-mb Height Analysis as a Reference Level for Satellite-Derived soundings. Albert Thomasell, Jr., December 1975, 25 pp. (PB-253-058)

NESS 74 On the Estimation of Areal Windspeed Distribution in Tropical Cyclones With the Use of Satellite Data. Andrew Timchalk, August 1976, 41 pp. (PB-261-971)

NESS 75 Guide for Designing RF Ground Receiving Stations for TIROS-N. John R. Schneider, December 1976, 126 pp. (PB-262-931)

NESS 76 Determination of the Earth-Atmosphere Radiation Budget from NOAA Satellite Data. Arnold Gruber, November 1977, 31 pp. (PB-279-633)

NESS 77 Wind Analysis by Conditional Relaxation. Albert Thomasell, Jr., January 1979.

NESS 78 Geostationary Operational Environmental Satellite/Data Collection System. July 1979, 86 pp. (PB-301-276)

NESS 79 Error Characteristics of Satellite-Derived Winds. Lester F. Hubert and Albert Thomasell, Jr. June 1979, 44 pp. (PB-300-754)

NESS 80 Calculation of Atmospheric Radiances and Brightness Temperatures in Infrared Window Channels of Satellite Radiometers. Michael P. Weinreb and Michael L. Hill, March 1980, 43 pp.

NOAA SCIENTIFIC AND TECHNICAL PUBLICATIONS

The National Oceanic and Atmospheric Administration was established as part of the Department of Commerce on October 3, 1970. The mission responsibilities of NOAA are to assess the socioeconomic impact of natural and technological changes in the environment and to monitor and predict the state of the solid Earth, the oceans and their living resources, the atmosphere, and the space environment of the Earth.

The major components of NOAA regularly produce various types of scientific and technical information in the following kinds of publications:

PROFESSIONAL PAPERS — Important definitive research results, major techniques, and special investigations.

CONTRACT AND GRANT REPORTS — Reports prepared by contractors or grantees under NOAA sponsorship.

ATLAS — Presentation of analyzed data generally in the form of maps showing distribution of rainfall, chemical and physical conditions of oceans and atmosphere, distribution of fishes and marine mammals, ionospheric conditions, etc.

TECHNICAL SERVICE PUBLICATIONS — Reports containing data, observations, instructions, etc. A partial listing includes data serials; prediction and outlook periodicals; technical manuals, training papers, planning reports, and information serials; and miscellaneous technical publications.

TECHNICAL REPORTS — Journal quality with extensive details, mathematical developments, or data listings.

TECHNICAL MEMORANDUMS — Reports of preliminary, partial, or negative research or technology results, interim instructions, and the like.

Information on availability of NOAA publications can be obtained from:

ENVIRONMENTAL SCIENCE INFORMATION CENTER (D822)
ENVIRONMENTAL DATA AND INFORMATION SERVICE
NATIONAL OCEANIC AND ATMOSPHERIC ADMINISTRATION
U.S. DEPARTMENT OF COMMERCE

6009 Executive Boulevard
Rockville, MD 20852

NOAA--S/T 80-281