Comparing The Accuracy and Efficiency of Algorithms For Converting Cartesian to Geodetic Coordinates

by

Robert W. Voll

Project Report submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING
in
Civil Engineering

APPROVED:

Steven D. Johnson, Ph.D.,Committee Chairman

Tomás Soler, Ph.D.

Gerald M. Elphingstone, Ph.D.

May, 1990

Blacksburg, Virginia

Comparing The Accuracy and Efficiency

of Algorithms

For Converting Cartesian to Geodetic Coordinates

by

Robert W. Voll

Committee Chairman: Steven D. Johnson, Ph.D.

Civil Engineering

(ABSTRACT)

A computer program written in PASCAL code was formulated for testing conversion algorithms for efficiency and accuracy. Seven direct methods and seven iterative methods were chosen for comparison. Representations of the latest algorithms as well as those no longer in popular use were employed in the test.

Of the 14 methods tested, results show Bowring's 1976 iterative procedure to be the most accurate.

Table of Contents

Figure page

1.0 INTRODUCTION

A method is needed that will allow comparisons of speed and accuracy to be made among the conversion algorithms currently available. This will allow the best algorithm for a particular use to be intelligently selected.

Great accuracy is desired in positioning space-based systems since positioning accuracy determines, to a large extent, the accuracy of data received from them.

Algorithm execution speed is also of great importance due to the distance covered by a satellite in a short period of time and the great number of positioning points needed to track its movement accurately. A fast algorithm can provide required data in near real-time, via computer.

Many different algorithms have been derived for converting Cartesian Coordinates into the Geodetic Coordinates of latitude(φ), longitude(λ), and height(h). Each technique proposes an advantage in accuracy and/or time saving computations whether by iteration or by the direct (closed form) method.

The objective of this project is to devise a procedure for testing conversion algorithms for efficiency and accuracy. Several of the latest algorithms as well as some of the older ones no longer in popular use were chosen for comparison.

2.Ø DESCRIPTION OF ALGORITHMS TESTED

Given the geocentric cartesian coordinates X, Y, and Z, and the ellipsoid major and minor semiaxes (a and b), the geodetic latitude and height may be calculated using any of the following methods.

Comparison of error graphs for all methods may be located in Appendix 1.

2.1 Direct Methods

2.1.1 - Borkowski's Method

In 1989 K. M. Borkowski published his closed-form method for transforming Geocentric to Geodetic Coordinates.

The following variables are defined:

a,b = Major and minor semiaxes.

r = Equatorial component of position vector in the cartesian coordinate system.

p_θ,c = Intermediate terms.

ψ = Parametric, or reduced, latitude

φ = Geodetic latitude

X,Y,Z= Geocentric coordinates

h = Geodetic height

The equations are

$r = \sqrt{(X^2+Y^2)}$

$p_0 = \arctan(bZ/ar)$

$c = (a^2-b^2)/\sqrt{((ar)^2+(bZ)^2)}$

$\psi = p_0+0.5c(\sin(2p_0))/(1.0-c(\cos(2p_0)))$;{initial value}

$\psi = \psi-(\sin(\psi-p0)-0.5c(\sin2\psi))/(\cos(\psi-p0)-c(\cos2\psi))$

$\varphi = \arctan(a(\sin\psi)/(b(\cos\psi)))$

$h = (r-a(\cos\psi))\cos\varphi+(Z-b(\sin\psi))\sin\varphi$

Borkowski's method, as translated from the original Lahey FORTRAN code and programmed in PASCAL, may be found in subroutine "GEOD", appendix 3.

2.1.2 - Bowring's 1985 Method

B. R. Bowring, in 1985, published a non-iterative method for calculating latitude from Cartesian space coordinates (X,Y,Z) and a height formula, accurate for outer-space situations.

The following variables are defined:

a,b = major and minor semiaxes

R,φ = polar coordinates

e^2 = the square of the first eccentricity

ε = the second eccentricity

β = the geodetic latitude

η = the normal terminated by the minor axis

h = height above the ellipsoid.

The equations are

$p = \sqrt{(X^2+Y^2)}$

$R = \sqrt{(p^2+Z^2)}$

$e^2 = (a^2-b^2)/a^2$

$u = \arctan(bZ(1+\varepsilon b/R)/(ap))$

$\beta = \arctan((Z+\varepsilon b \sin^2 u)/(p-e^2 a \cos^3 u)$

$\eta = a/\sqrt{(1-e^2 \sin^2 B)}$

$h = p\cos B + Z\sin B - a^2/\eta$

The PASCAL subroutine containing these equations may be found in procedure "bowring85" located in appendix 3.

2.1.3 - Bowring's 1985 Method, Version 2

In 1988 T. Soler and L.D. Hothem presented a modified version of the Bowring method as listed above.

The equations, in program order, follow:

$p = \sqrt{(X^2+Y^2)}$

$r = \sqrt{(p^2+Z^2)}$

$e^2 = (a^2-b^2)/a^2$

$\mu = \arctan((Z/p)((1-f)+(e^2 a/r)))$

$\varphi = \arctan((Z(1-f)+(e^2 a \sin^3 \mu))/((1-f)(p-(e^2 a \cos^3 \mu))))$

$h = p \cos\varphi + Z \sin\varphi - a\sqrt{(1-e^2 \sin^2 \varphi)}$

This method is mathematically identical to Bowring's 1985 method, listed above, and yields identical results. The

programmed algorithm may be found in procedure "bowring88" located in appendix 3.

2.1.4 - Gersten's Method

R. H. Gersten developed A procedure for determining the geodetic sub-latitude and height of a space vehicle from its geocentric position. Developed in 1960, this algorithm is the most dated of all tested for this report. The PASCAL procedure for this algorithm, entitled "gersten60", may be found in appendix 3. The original work published by Gersten did not discuss areas where the algorithm could not be used but the algorithm will not work at the poles. Problems in computation at the poles can be handled in more than one way. A discussion of two methods of overcoming this problem and a description of the equations used in this algorithm are discussed.

The following variables are defined:

ε = an intermediate term.

r = the distance between the geocenter and the space vehicle. When X and Y are zero, Z equals r and the computation lies along the Z axis. e.g. the geocentric and geodetic latitudes are exactly 90°.

φ' = the geocentric latitude

φ = the geodetic latitude

h = the geodetic height

The equations are

$r = \sqrt{(X^2+Y^2+Z^2)}$

$e^2 = (a^2-b^2)/a^2$

$\varepsilon = ae^2/r$

$\varphi' = \arcsin(Z/r)$

$\sin\varphi = \sin\varphi'(1 + \varepsilon \cos^2\varphi')$

$\cos\varphi = \cos\varphi'(1 - \varepsilon \sin^2\varphi')$

$\varphi = \arctan(\sin\varphi/\cos\varphi)$; If error trapping was not done previously it must be done at this point to prevent attempted division by zero when $\varphi = 90°$.

$h = r-a(1-0.5e^2(1+\varepsilon)\sin^2\varphi'+0.5e^2(\varepsilon-0.25e^2)\sin\varphi'^4$

In 1977 D.K. Olson presented a refinement of the Gersten method. Olson's method is not included in this test but may be encoded at a later date.

2.1.5 - Heikkenen's Method

In 1982 M. Heikkinen published, in German, a closed formula for transforming rectangular space coordinates to geodetic latitude and height.

The following variables are defined:

e'^2 = the square of the second eccentricity

E^2 = linear eccentricity squared

The equations are

$e^2 = (a^2-b^2)/a^2$

$e'^2 = (a^2-b^2)/b^2$

$E^2 = a^2-b^2$

$r = \sqrt{(X^2+Y^2)}$

$F = 54b^2Z^2$

$G = r^2+(1-e^2)Z^2-e^2E^2$

$c = e^4Fr^2/G^3$

$s = (1+c+\sqrt{(c^2+2c)})^{1/3}$

$P = F/((3(s+(1/s)+1)^2G^2)$

$Q = \sqrt{(1+2e^4P)}$

$r_0 = (Pe^2r/(1+Q))+$

$\sqrt{((a^2/2)(1+(1/Q))-(P(1-e^2)Z^2)/(Q(1+Q)))-(Pr^2)/2)}$

$U = \sqrt{((r-e^2r_0)^2+Z^2)}$

$V = \sqrt{((r-e^2r_0)^2+(1-e^2)Z^2)}$

$Z_0 = b^2Z/aV$

$h = U(1-b^2/aV)$

$\sin\varphi = (Z+e'^2Z_0)/\sqrt{(r^2+(Z+e'^2Z_0)^2)}$

$\cos\varphi = r/\sqrt{(r^2+(Z+e'^2Z_0)^2)}$

$\varphi = \arctan(\sin\varphi/\cos\varphi)$

The PASCAL programming of Heikkinen's equations may be located in appendix 3 in procedure "heikkinen82".

2.1.6 - Paul's Method

M.K. Paul published, in 1973, a closed formula for obtaining the geodetic coordinates of latitude (φ) and height (h) given the geocentric cartesian coordinates (X,Y,Z).

The following variables are defined:

t_1 = geodetic latitude

The equations are

$$p = \sqrt{(X^2+Y^2)}$$

$$e^2 = (a^2-b^2)/a^2$$

$$\alpha = (p^2+a^2e^2)/(1-e^2)$$

$$\beta = (p^2-a^2e^2)/(1-e^2)$$

$$q = 1 + [27Z^2\ (\alpha^2-\beta^2)]/[2\ (Z^2+\beta)^3]$$

$$t_1 = [(Z^2+\beta)/12][(q+\sqrt{(q^2-1)})^{1/3} + (q+\sqrt{(q^2-1)})^{-1/3}] - \beta/6 + Z^2/12$$

If t_1 is greater than $\emptyset$ then:

$$\varphi = \arctan[(Z/2 + \sqrt{(t_1)} + \sqrt{(-\beta/2 + Z^2/4 - t_1 + \alpha Z/(4\sqrt{(t_1)}))})]$$

If t_1 is less than or equal to $\emptyset$ then:

$$\varphi = \arctan[(\alpha+\beta+\sqrt{(\alpha^2-\beta^2)})Z/(2\beta p) - (\sqrt{(\alpha^2-\beta^2)}(\alpha+ (\alpha^2-\beta^2))^2Z^3)/(4\beta^4 p)]$$

$$N = a/\sqrt{(1-e^2 \sin^2\varphi)}$$

$$h = (p/\cos\varphi) - N$$

Paul's method, programmed in PASCAL, is located in appendix 3 in procedure "paul73".

2.1.7 - Wei's Direct Method

In 1986 Z. Wei developed a direct solution for the conversion of Cartesian coordinates into the geodetic coordinates of latitude and height. His solution consists of eleven equations.

The following variable is defined:

k = the Lagrangian multiplier

The equations are

$e^2 = (a^2-b^2)/a^2$

$\alpha = (X^2+Y^2)/a^2 + (1-e^2)Z^2/a^2$

$\beta = (X^2+Y^2)/a^2 - (1-e^2)Z^2/a^2$; a misprint was found in Wei's published equation for β, the term $(X_2 + Y_2)$, should be $(X^2 + Y^2)$.

$q = 1 + 27e^4(\alpha^2-\beta^2)/2(\alpha-e^4)^3$

$A = -q + \sqrt{(q^2-1)}$

$t_2+t_3 = abs^*[(1/3)(\alpha+e^4/2)-((\alpha-e^4)/12)(A^{1/3} + A^{-1/3})]$

$2\sqrt{(t_2 t_3)} = abs^*[\sqrt{((t_2+t_3)^2 + ((\alpha-e^4)((A^{1/3}- A^{-1/3}))^2/48)}]$

$\sqrt{(t_1)} = -e^2\beta/4(2\sqrt{(t_2 t_3)})$

$k = abs^*[\sqrt{(t_1)} + abs^*[\sqrt{((t_2+t_3) + 2\sqrt{(t_2 t_3)})}] - (1-(e^2/2))]$

$h = k/(1+k)\ \sqrt{(X^2+Y^2+((1+k)/(1-e^2+k))^2Z^2)}$

φ = arctan(((1+k)/(1-e²+k))Z/√(X²+Y²))

* Taking the absolute value of these terms was a modification of the original work. It was required to prevent premature program termination. The complete PASCAL procedure, "weidirect86", is located in appendix 3.

2.2 Iterative Methods

The maximum number of iterations for all methods is set to 20.

2.2.1 - Baird's Method

In 1964 R. W. Baird designed a method for converting position data recorded in rectangular cartesian coordinates into geodetic coordinates. The original program was written in FORTRAN IV code. Baird's equations were used in the PASCAL procedure "baird64" listed in appendix 3. Newton's method of iteration to solve the quartic transcendental equation in Z_1 is used. The number of iterations is determined by the difference in successive values of Z_1.

The following variables are defined:

$f(Z_1)$ = the function

$f'(Z_1)$= the derivative of the function. This term was error checked for zero.

The equations used in order of their appearance in the procedure are

$e^2 = (a^2-b^2)/a^2$

$k_8 = 1-e^2$ $\quad k_1 = -e^4/k_8$ $\quad k_2 = -2e^2$

$k_7 = a$ $\quad k_9 = k_7^2$ $\quad k_3 = k_9 e^2$

$k_4 = -k_8$ $\quad k_5 = 2k_9 e^2 k_8$ $\quad k_6 = k_9 k_8^2$

$R_e = \sqrt{(X^2+Y^2)}$

$Z_1 = k_7 Z\sqrt{k_8}/\sqrt{((k_8(X^2+Y^2+Z^2))}$; the first approximation of Z_1, (called Z_2 in the reference).

Begin iterative procedure:

$f(Z_1) = k_1 Z_1^4 + k_2 Z Z_1^3 + (k_3+k_4 Z^2-R_e^2) Z_1^2 + k_5 Z Z_1 + k_6 Z^2$

$f'(Z_1) = 4k_1 Z_1^3 + 3k_2 Z Z_1^2 + 2(k_3+k_4 Z^2-R_e^2) Z_1 + k_5 Z$

$Y_1 = Z_1 - f(Z_1)/f'(Z_1)$

$Z_{old} = Z_1$

$Z_1 = Y_1$

The iteration continues until the absolute value of $(Z_1 - Z_{old})$ is less than or equal to the accuracy desired or until the maximum number of iterations is reached.

$\varphi = \arctan(Z_1/\sqrt{(k_8(k_9 k_8 - Z_1^2))})$

$h = Z/\sin\varphi - k_7 k_8/\sqrt{(1-e^2 \sin^2\varphi)}$; An error exists at this point in the FORTRAN IV code found in the reference. The k_8 term in this equation is carried as (1-e) instead of ($1-e^2$) in the reference code.

2.2.2 - Bowring's 1976 Method

An iterative algorithm to generate geodetic coordinates; developed by Bowring and explained by Rapp, p123. This method required error trapping when values of X and Y equal zero; when φ is 90°. The computer code may be located in Appendix 3 in PASCAL procedure "bowring76".

The following variables are defined:

i = a counter

f = polar flattening

M = the principal radius of curvature in the Meridian.

N = the principal radius of curvature in the prime vertical.

φ_{old} stores the last value of the geodetic latitude.

The equations are

i = 0; setting the initial value.

$e^2 = (a^2-b^2)/a^2$

$\beta = \arctan((a/b)(Z/\sqrt{(X^2+Y^2)}))$;{eqn.6.10}

$f = (a-b)/a$;{eqn.3.3}

$e'^2 = (a^2-b^2)/b^2$;{eqn.3.5}

$\varphi = \arctan(Z/\sqrt{(X^2+Y^2)})$; using the geocentric latitude for the initial value of φ.

Begin iterative procedure:

i = i+1

$\varphi_{old} = \varphi$

φ = arctan((Z+e'²bsin³β)/($\sqrt{}$(X²+Y²)-ae²cos³β);

{eqn.6.105}

β = arctan((1-f)tanφ);{eqn.6.107}

These last few equations are repeated until the difference between φ and φ_{old} is less than or equal to the accuracy desired or until the maximum allowable number of iterations has been reached.

M = a(1-e²)/(1-e²sin²φ)$^{3/2}$;{eqn.3.87}

N = a/$\sqrt{}$(1-e²sin²φ);{eqn.3.99}

h = $\sqrt{}$(X²+Y²)/cosφ-N;{eqn.6.98} calculating geodetic height for equatorial latitudes.

h = Z/sinφ-N+e²N;{eqn.6.99} calculating geodetic height for polar latitudes.

This completes Bowring's 1976 Method.

2.2.3 - Goad's Method

An iterative method for converting Cartesian coordinates into latitude, and height using Newton's method as taught by Dr. Goad at the University of Ohio. Error trapping for zero values of X and Y, i.e. when geodetic latitude equals 90°, was required. This method may be located in Appendix 3 in PASCAL procedure "goad87".

The following variables are defined:

r = polar distance.

ha = an initial approximation for geodetic height.

count; keeping count of the number of iterations

The equations are

$e^2 = (a^2-b^2)/a^2$

$pe = \sqrt{(X^2+Y^2)}$

φ = arctan(z/pe); calculating the geocentric latitude as an initial approximation for geodetic latitude.

$r = \sqrt{((pe)^2+Z^2)}$

$ha = r-a(1-f)\sin^2\varphi$

Begin iterative procedure

{compute approximate N, p, Δp, and ΔZ}

$N = a/\sqrt{(1-e^2\sin^2\varphi)}$

$pa = (N+ha)\cos\varphi$

$\Delta p = pe-pa$

$za = (N(1-e^2)+ha)\sin\varphi$

$\Delta Z = Z-za$

{using approximate quantities, estimate $\Delta\varphi$ and Δh}

$\Delta\varphi = \cos\varphi\Delta Z - \sin\varphi\Delta p/(N+ha)$

$\Delta h = \cos\varphi\Delta p + \sin\varphi\Delta Z$

$\varphi = \varphi + \Delta\varphi$; updating φ and h for the next iteration.

$ha = ha + \Delta h$

count = count+1

The iterative procedure is repeated until Δp and ΔZ are less than the accuracy desired or until the maximum allowable number of iterations has been reached.

2.2.4 - Hirvonen and Moritz Method

Calculating geodetic coordinates(latitude, and height) given cartesian coordinates(X,Y,Z); an iterative method proposed by Hirvonen and Moritz in 1963 and explained by Rapp in his notes, March 1984. Error trapping for zero values of X and Y, i.e. when geodetic latitude equals 90°, was required. This method may be found in Appendix 3 in PASCAL procedure "hm63".

The following variables are defined:

φ_1 and φ_2 hold the latest value of φ

The equations are

$i = 0;$

$e^2 = (a^2-b^2)/a^2$

$\varphi = \arctan((Z/\sqrt{(X^2+Y^2)})(1/(1-e^2)));$ {eqn. 6.97}

{first approximation for the geodetic latitude, assume height is zero}

$ww = \sqrt{(1-e^2\sin^2\varphi)};$ {eqn. 3.40}

$N = a/ww;$ {eqn. 3.99}

Begin iterative procedure

$\varphi_1 = \varphi$

φ = arctan((Z+e²Nsinφ_1)/√(X²+Y²));{eqn. 6.95}

φ_2 = φ

ww = √(1-e²sin²φ);{eqn. 3.40, recalculating ww using latest value of φ}

N = a/ww;{eqn. 3.99}

i = i+1

End iterative procedure when the difference between φ_1 and φ_2 is less than or equal to the accuracy desired or when the number of iterations reaches the maximum specified.

h = (√(X²+Y²)/cosφ)-N;{eqn. 6.98}

2.2.5 - Nautiyal's Method

An iterative algorithm to generate geodetic coordinates from Earth-centered Earth-fixed coordinates that is free from convergence problems near the poles and near the equator. Developed by A. Nautiyal, Defense Research and Development Laboratory, Hyderabad, India. Error trapping for zero values of X and Y, i.e. when geodetic latitude equals 90°, was required. Nautiyal's method, programmed in PASCAL, may be located in procedure "nautiyal", Appendix 3.

The following variables are defined:

i = the counter

Θ = geocentric latitude

bb = negative value of the second eccentricity squared

temp = stores the last value of t

φ = geodetic latitude

The equations are

$i = \emptyset$

$e^2 = (a^2-b^2)/a^2$

$aa = (Z/b)^2$

$bb = -e^2/(1-e^2)$

$cc = (a/b)^2$

$dd = X/a$

$aa_0 = (bb)^2$

$aa_1 = 2bbccdd$

$aa_2 = aa+cc^2dd^2-bb^2$

$aa_3 = -2bbccdd$

$aa_4 = -cc^2dd^2$

$\tan\Theta = Z/\sqrt{(X^2+Y^2)}$

$t = \sqrt{(1/(1+\tan^2\Theta/(1-e^2)))}$;{initial value}

Begin Newton-Raphson method

$i = i+1$;{incrementing counter}

$temp = t$

$f(t) = aa_0t^4 + aa_1t^3 + aa_2t^2 + aa_3t + aa_4$;{eqn.12}

$f'(t) = 4aa_0t^3 + 3aa_1t^2 + 2aa_2t + aa_3$

$t = t - f(t)/f'(t)$

The last five equations are iterated until (t-temp) is less than or equal to the accuracy desired or i reaches the

The last five equations are iterated until (t-temp) is less than or equal to the accuracy desired or i reaches the set limit.

$\tan^2\varphi = (1-t^2)/(t^2(1-e^2))$;{eqn.17}

$\varphi = \arctan(\sqrt{(\tan^2\varphi)})$

$h = (\sqrt{(X^2+Y^2)}-at)\sqrt{(1+\tan^2\varphi)}$;{eqn.18}

2.2.6 - Torge's Method

In 1975 W. Torge presented, in German, his publication of "Geodäsie", a reference work for graduate students whose areas of study include geodesy, surveying, photogrammetry, and cartography. The following iterative method for computation of geodetic coordinates was taken from the 1980 English translation and revision of Torge's work. Error trapping is required when X and Y are Zero, i.e. when $\varphi = 90°$. Torge's method, programmed in PASCAL, may be located in Appendix 3 in procedure "torge".

The following variables are defined:

i = the counter for the number of iterations

φ_{old} = temporary storage for last value of φ

N = radius of curvature in the prime vertical

The equations are

$i = 0$

$e^2 = (a^2-b^2)/a^2$

φ = arctan(Z/√(X²+Y²));{using the geocentric latitude as a first approximation for geodetic latitude}

N = a/√(1-e²sin²φ);{initial value}

h = (√(X²+Y²)/cosφ)-N;{first approximation}

Repeat the following sequence of equations until the difference between φ and φ_{old} is less then or equal to the value of accuracy desired or until the maximum number of iterations has been reached:

φ_{old} = φ; storing last value of φ.

N = a/√(1-e²sin²φ);{using latest value of φ}

φ = arctan(Z/(√(X²+Y²)(1-e²N/(N+h)))))

h = √(X²+Y²)/cosφ-N

i = i+1

2.2.7 - Wei's Iterative Method

In 1986 Z. Wei developed an iterative solution for the conversion of Cartesian coordinates into the geodetic coordinates of latitude and height. His solution, programmed in PASCAL subroutine "weiiterate86", located in appendix 3, consists of the following equations:

The following variables are defined:

φ = the geodetic latitude.

The equations are

e² = (a²-b²)/a²

$\tan\varphi = Z/\sqrt{(X^2+Y^2)}$; initial value

Begin iterative procedure

$\tan\varphi_{i-1} = \tan\varphi$

eqn. A.4a in the text:

$$\tan\varphi = Z/\sqrt{(X^2+Y^2)} + ae^2\tan\varphi_{i-1}/(\sqrt{(X^2+Y^2)}\sqrt{(1+(1-e^2)\tan^2\varphi)})$$

Iteration is continued until $\tan\varphi - \tan\varphi_{i-1}$ is less than or equal to the accuracy desired or until the maximum number of iterations is reached.

$\varphi = \arctan(\tan\varphi)$

$N = a/\sqrt{(1-e^2\sin^2\varphi)}$; computing the radius of curvature of the prime vertical.

$h = \sqrt{(X^2+Y^2)}/\cos\varphi - N$

With the computation of the geodetic height (h) the process is completed.

In the text an alternate to eqn. A.4a is given in eqn. A.4b. e.g. $\tan\varphi = Z/(\sqrt{(X^2+Y^2)} - ae^2/\sqrt{(1+(1-e^2)\tan^2\varphi)})$ which is said to converge slightly faster than eqn. A.4a.

Equation A.4b is mathematically identical to equation A.4a; proof follows:

From eqns. A.1 in the text:

$X = (N+h)\cos\varphi\cos\lambda$

$Y = (N+h)\cos\varphi\sin\lambda$

$X^2+Y^2 = \{(N+h)\cos\varphi\cos\lambda\}^2 + \{(N+h)\cos\varphi\sin\lambda\}^2$

$$= (N+h)^2\cos^2\varphi\cos^2\lambda + (N+h)^2\cos^2\varphi\sin^2\lambda$$

$$= (N+h)^2\cos^2\varphi(\cos^2\lambda+\sin^2\lambda)$$

$$\sqrt{(X^2+Y^2)} = (N+h)\cos\varphi$$

$$N+h = \sqrt{(X^2+Y^2)}/\cos\varphi \quad \text{(eqn. A.3 derived)}$$

$$N = a/\sqrt{(1-e^2\sin^2\varphi)} \quad \text{(eqn. A.2)}$$

$$Z = (N+h-Ne^2)\sin\varphi \quad \text{(eqn. A.1)}$$

Substituting eqns. A.2 and A.3 into A.1:

$$Z = (\sqrt{(X^2+Y^2)}/\cos\varphi - ae^2/\sqrt{(1-e^2\sin^2\varphi)})\sin\varphi \qquad (1)$$

An aside:

$$\tan\varphi = \sin\varphi/\cos\varphi \;>>>>>>>\; \sin\varphi = \tan\varphi\cos\varphi$$

$$\text{thus: } \sin^2\varphi = \tan^2\varphi\cos^2\varphi \qquad (2)$$

Substitute (2) into (1):

$$Z = \sin\varphi\sqrt{(X^2+Y^2)}/\cos\varphi - ae^2\sin\varphi/\sqrt{(1-e^2\tan^2\varphi\cos^2\varphi)}$$

$$= \tan\varphi\sqrt{(X^2+Y^2)} - ae^2\sin\varphi/\sqrt{(\sin^2\varphi+\cos^2\varphi-e^2\tan^2\varphi\cos^2\varphi)}$$

$$= \tan\varphi\sqrt{(X^2+Y^2)} - ae^2\sin\varphi/\sqrt{(\cos^2\varphi(\sin^2\varphi/\cos^2\varphi + 1 - e^2\tan^2\varphi))}$$

$$= \tan\varphi\sqrt{(X^2+Y^2)} - ae^2\sin\varphi/(\cos\varphi\pm(\tan^2\varphi+1-e^2\tan^2\varphi))$$

$$= \tan\varphi\{\sqrt{(X^2+Y^2)} - ae^2/\pm(1+(1-e^2)\tan^2\varphi)\}$$

$$\tan\varphi = Z/\{\sqrt{(X^2+Y^2)} - ae^2/\pm(1+(1-e^2)\tan^2\varphi)\} \quad \text{(eqn. A.4b)}$$

Deriving eqn. A.4a from eqn. A.4b:

$$Z = \tan\varphi\sqrt{(X^2+Y^2)} - \tan\varphi ae^2/\sqrt{(1+(1-e^2)\tan^2\varphi)}$$

$$Z/\sqrt{(X^2+Y^2)} = \tan\varphi - \tan\varphi ae^2/\{\sqrt{(X^2+Y^2)}\sqrt{(1+(1-e^2)\tan^2\varphi)}\}$$

$$\tan\varphi = Z/\sqrt{(X^2+Y^2)} +$$

$$\tan\varphi a e^2 / \{\sqrt{(X^2+Y^2)}\sqrt{(1+(1-e^2)\tan^2\varphi)}\} \quad \text{(eqn. A.4a)}$$

Therefore A.4a and A.4b are different forms of the same equation and should give the same results.

3.0 IMPLEMENTATION

Hardware consisted of an IBM/PC equipped with a 20 megabyte hard disk drive, floppy disk drive, 640 kb RAM, and an 8087 math coprocessor. Due to the large number of disk I/O operations a hard disk drive was found to cut program operating time significantly, but is not necessary for program operation. It is necessary, however, to have a disk with storage capacity in excess of the usual 360 kb. Likewise, it is possible, with Turbo PASCAL 5.0[1], to emulate 8087 math co-processor operation, but program operation time and test times will be found to increase significantly.

An IBM/PC was chosen for the test to avoid the user charges incurred when using a mainframe computer, to keep hardware requirements simple and low cost, and to preserve program transportability. In fact, much of the programming and debugging was accomplished using a small Toshiba "1000" laptop computer before finally transfering the program to the desk top IBM/PC for the actual testing process.

Turbo Pascal 5.0 was chosen as the programming language for the following features: low cost, transportability, coprocessor emulation capability, uncomplicated modular programming, easy disk access for I/O processing, fast

[1] Borland International

debugging capability, no practical limit on program size, simple access to the DOS clock, and, rapid, one pass, compiler. The Turbo 5.Ø graphics capability was not exploited in order to retain program transportability. One disadvantage was encountered; it was necessary to program a library of math routines before proceeding with the actual comparisons.

4.0 TEST PROCEDURE

The initial data set, data set 1, for the test consisted of 133 values each, of latitude(φ), longitude(λ), and height(h), based on latitudes from 0° to 90° (every 5°) and heights from the surface to 30,000 km (every 5000 km). Height of the Global Positioning System (GPS) Satellite, approximately 26,561 km, was taken into account when setting the maximum height value. These initial values were generated by the test program itself.

4.1 PROGRAM OPERATION

Tracing the program flow, see figure 1, "Test Procedure Flow":
For simplification only the test portion of the program is shown.

Data set 1, Appendix 4, generated by the program and employed as initial input, is read into a disk data file, llh.fil, subsequently converted to Cartesian coordinates (X,Y,Z) through the following equations:

$$X = (N + h)\cos\varphi\cos\lambda$$

$$Y = (N + h)\cos\varphi\sin\lambda$$

$$Z = (N(1 - e^2) + h)\sin\varphi,$$

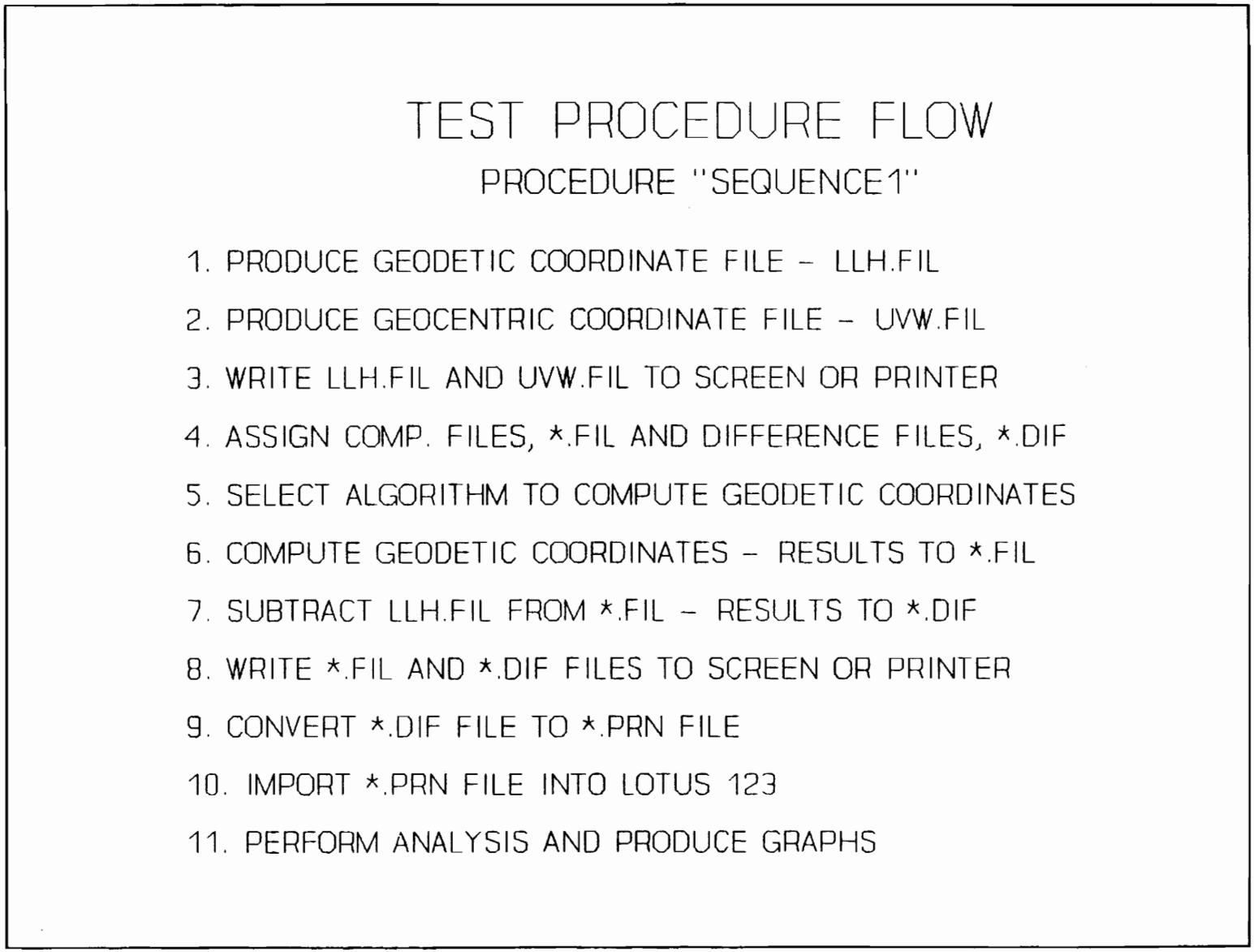


Figure 1 - Test Procedure Flow

and read into uvw.fil, data set 2, Appendix 5. Longitude is held constant at zero degrees since it is not a direct part of the test.

For iterative methods, if the difference between successive iterations of geodetic latitude becomes less than or equal to .00001 seconds of arc or if more than 20 iterations are reached without convergence, the program proceeds to compute geodetic height of the point using the latest value of geodetic latitude.

Values calculated by each algorithm consist of latitude and height which are saved to disk files (*.fil). The given values of latitude and height held in llh.fil are subtracted from these computed values. To convert latitude difference values from radians to meters of error along the meridian use the following equations:

1) Compute the principle radius of curvature in the plane of the meridian; $M = a(1-e^2)/(1-e^2\sin^2\varphi)^{3/2}$

2) Latitude Difference (meters) = (M+h)Latitude Difference (radians). Latitude difference values and height difference values, in meters, are read into *.dif files. These difference values represent the computation error generated by each algorithm.

Error values from the *.dif files are separated into two groups, one consisting of errors in latitude, the other of errors in height. The results are read into *.prn files; i.e. ASCII data files which could be imported into the Lotus 123[2] (version 2.01) application program for analysis. Keeping in mind that this version of Lotus can import a maximum of 240 characters per row and that desired accuracy required that 25 characters be allotted for each difference value, the maximum number of points per row was nine. With a minimum height of zero, a maximum of 30,000 km, and an interval of 5000 km, it is most convenient to import seven points per Lotus input row. The number of rows imported depends on the increment of latitude used.

As calculations are performed by the chosen algorithm using the data set of geocentric cartesian coordinates (X,Y,Z) from uvw.fil, data set 2, Appendix 5, timing is accomplished simultaneously by accessing the IBM/PC system clock. This clock is capable of accuracy to .01 seconds. Although this does not approach the accuracy of many mainframe system clocks, computation of the entire data set is being timed and times on the order of seconds are quite sufficient for these comparisons. By accessing the system clock from within each algorithm, at the beginning and end of each computation, disk

---

[1] Lotus Development Corporation

I/O time is eliminated. This results in timing of algorithm operation only.

Error trapping at singular points increases the accuracy and decreases the computation time of an algorithm. In most cases error trapping was only necessary at the poles and these algorithms can be compared fairly. In some cases, such as Baird's method, Hedman's method, and others, error trapping was also required at the equator. Error trapping may give an algorithm an unfair advantage; the algorithm may show a higher degree of accuracy and decreased execution time because of the error routine not because of the algorithm itself. When the cartesian coordinates "X" and "Y" are zero the geodetic latitude is 90° and the geodetic height equals the cartesian coordinate "Z" minus the value of the semi-minor axis "b"; $h_{90} = Z-b$. On the other hand when "Z" is zero the geodetic latitude is also zero and h_0 (the geodetic height) = $\sqrt{(X^2+Y^2)} - a$ (the semi-major axis). Of course when calculations are performed using these last two equations the error value is zero at the equator and at the pole.

The program also allows user input for minimum and maximum height and latitude, as well as for the increment values for both height and latitude. With these options it is possible to produce a test data set near any latitude and/or height.

5.0 ANALYSIS

Data imported into the Lotus spreadsheet from *.prn files, as mentioned in the Test Procedures section of this paper, was graphed at heights of 0km, 15,000km and 30,000km; see Appendix 1. Graphs were saved as *.pic files while the actual data in each spreadsheet was saved in *.WK1 files for later reference. Graphs (*.pic files) may be printed using the Lotus Printgraph Utility or Wordperfect V5.0.

The self scaling option for Lotus Graphs was used as an aid in plotting error curves. Graphs of error curves for all methods tested may be located in Appendix 1.

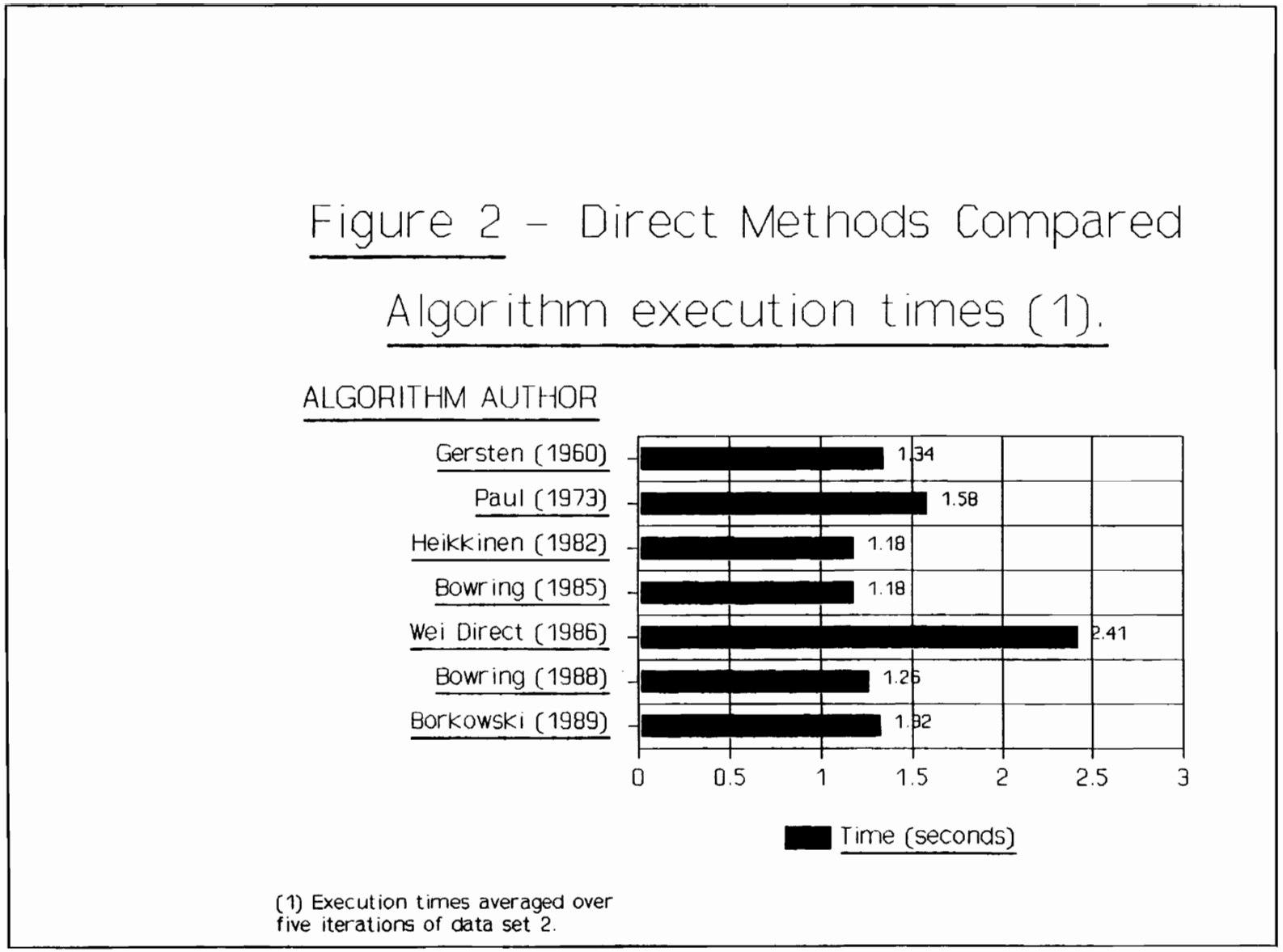


Figure 2 - Direct Methods Compared

Algorithms were compared not only for accuracy but also for the pattern of error. A non-random pattern in the error graph indicates a systematic error in the algorithm that could possibly be modeled out; a random pattern would be more desireable.

Accuracy, timing, and other feature comparisons may be found in figures 2 through 7.

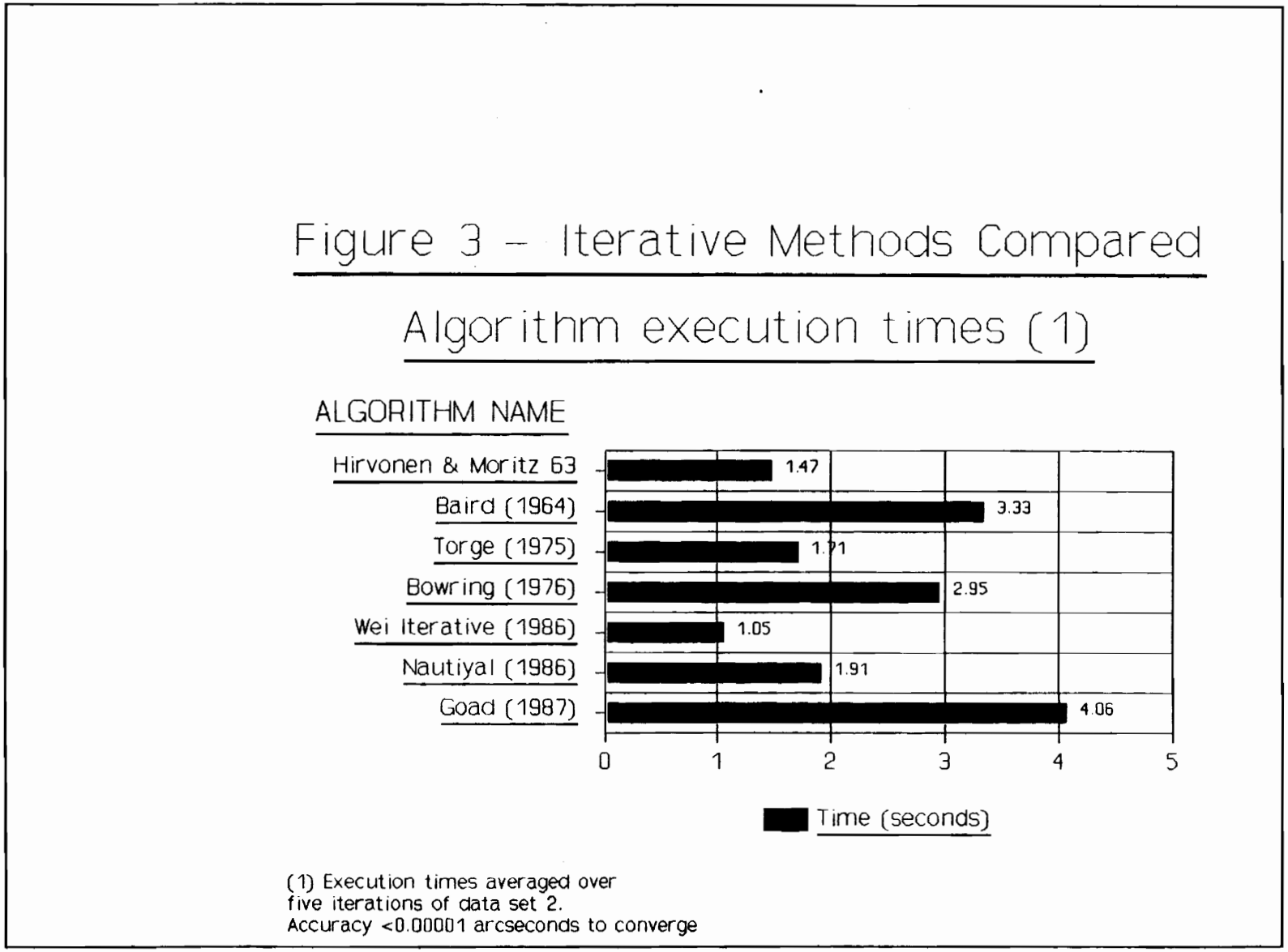


Figure 3 - Iterative Methods Compared

As a rule, Direct Methods usually have fewer equations to program but the equations themselves tend to be longer and more difficult. For the seven direct methods tested the average is about 9 lines of code (loc) and this ranges from 6 loc for Bowring's 1985 Version 2 to 18 loc for Heikkinen's Method. Refer to figure 7 for loc comparisons.

The average lines of code for the seven iterative methods tested was 14 but Nautiyal's Method needed 21 while Wei's Iterative Method required only 7. Usually, the iterative method has simpler, easier to program equations with the iterative procedure adding slightly to programming difficulty.

Easiest methods to program: Bowrings's 1985 Method, Version 2; Wei's iterative method.

In reviewing the test results, patterns in some of the error graphs, Appendix 1, indicate the error is not random.

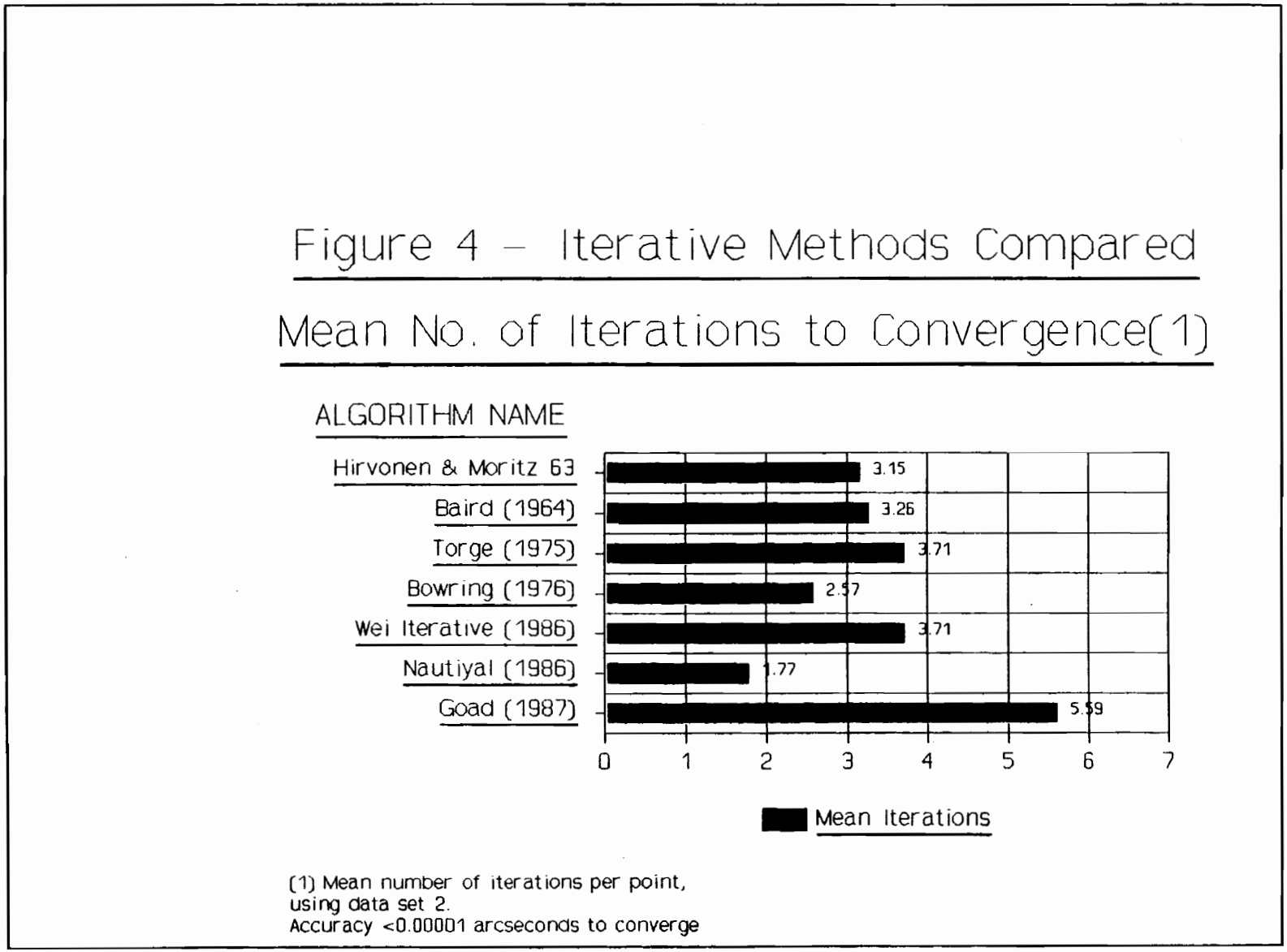


Figure 4 - Iterative Methods Compared

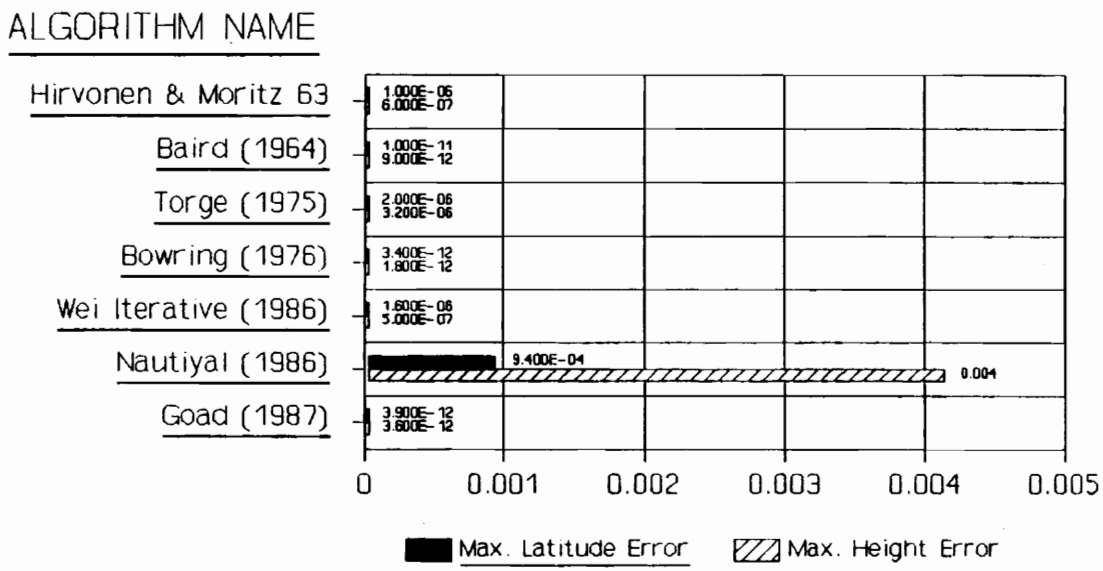
Based on a comparison of maximum error in latitude and height, figure 6, Wei's Direct method is the most accurate of the direct methods while, figure 5, Bowring's 1976 Method has a slight edge over Goad's as the most accurate iterative method. Recalling that an iterative method will be as accurate as the user may want simply by changing to a smaller incremental value between iterations, Bowring's 1976 Method is the most accurate of all tested.

Wei's iterative method using equation A.4a from the reference gave mean execution times of 1.Ø5 seconds. This is Ø.1Ø seconds faster than Heikkinen's or Bowring's 1985 Method which both had times of 1.15 seconds. See Figures 2 and 3 for algorithm execution times.

While comparing iterative algorithms for efficiency, the mean number of iterations per data set was found useful. See figure 4 for the results. Actual timing of algorithms to convergence, however, was still found to be the best method of comparison for speed of execution, see figure 3.

Figure 5 – Iterative Methods Compared

Maximum Error – Latitude/Height (1)



(1) Absolute Values, meters, as calculated using Data Set 2.

Figure 5 - Iterative Methods Compared

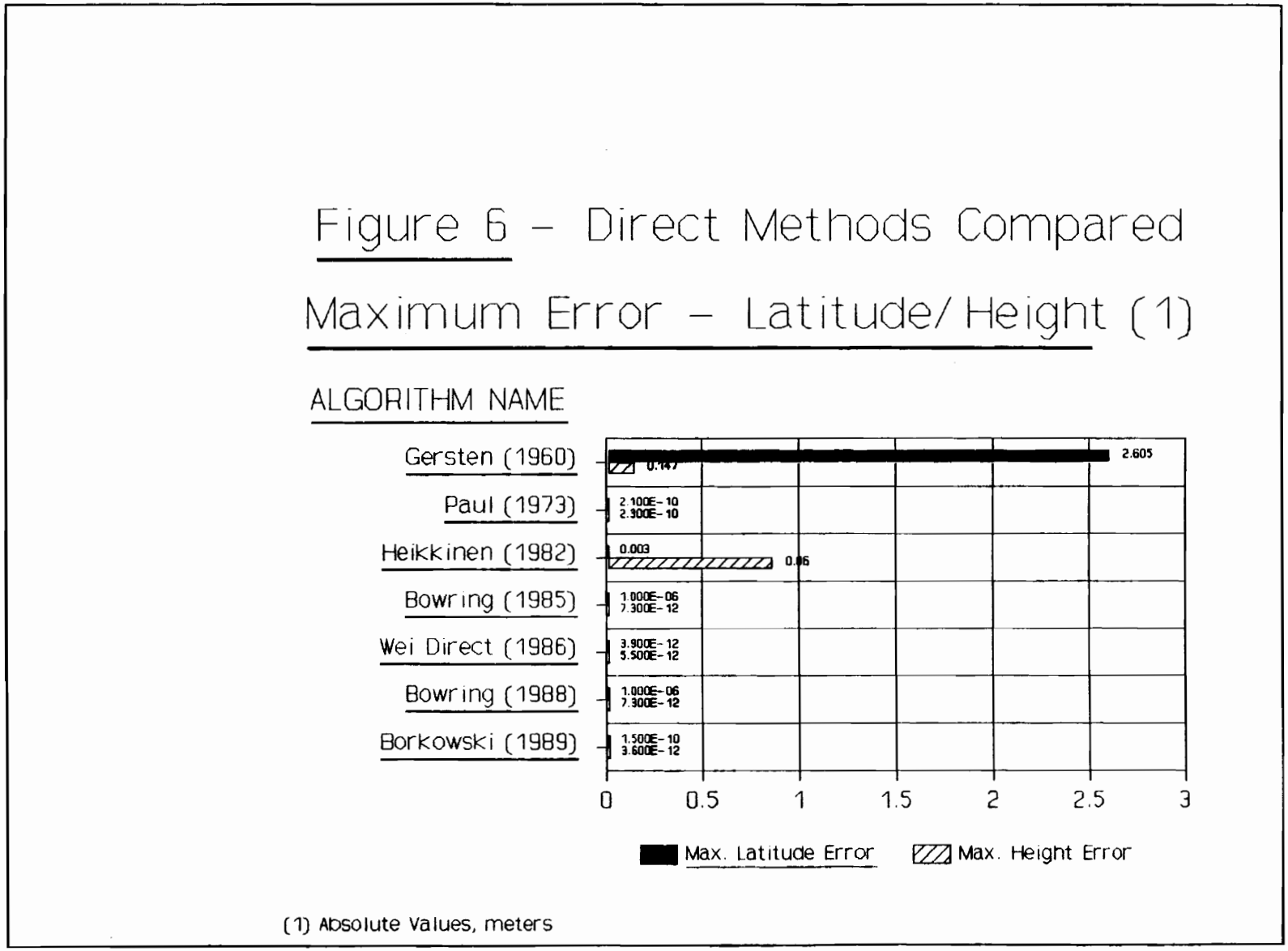


Figure 6 - Direct Methods Compared

Iterative methods appear to be more flexible; they can be as accurate as desired. i.e. By requiring a difference between successive iterations that is very small, a given iterative method can be more precise, and is usually easier to program than any closed method, usually at the cost of execution speed.

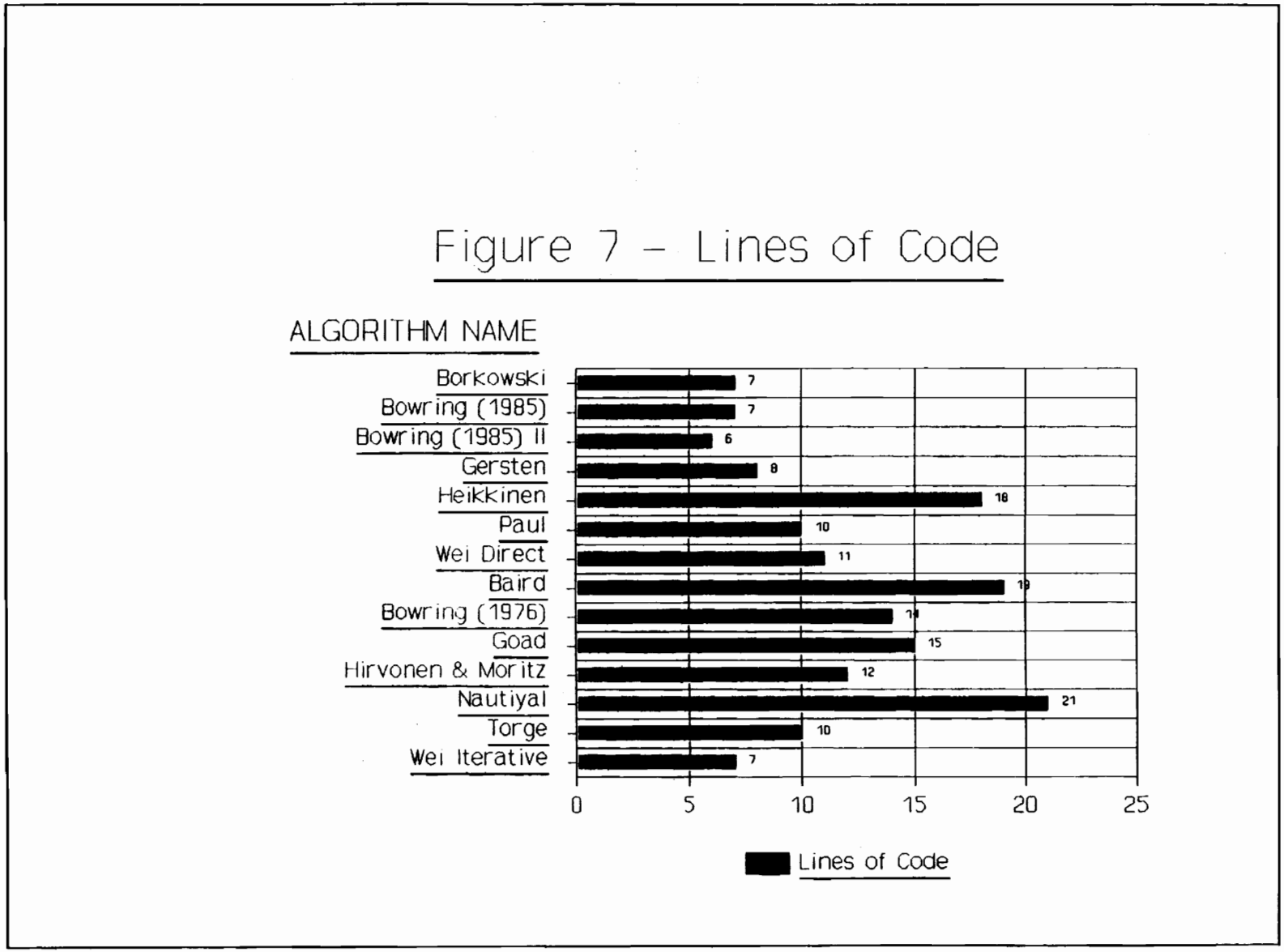


Figure 7 - Lines of Code

The best method is the fastest method that gives acceptable results for a given application; for example: if an iterative method gives acceptable results in one or two iterations it may be considerably faster than a direct method that gives a greater degree of accuracy than required.

6.0 CONCLUSIONS

Based on the foregoing comparisons the best overall algorithm, of those presently tested, was found to be Bowring's 1976 iterative method. While it is not the fastest method tested, the true test, in my opinion, is overall accuracy and lack of a pattern in the error curves. Bowring's 1976 Method was outstanding in both areas. The complete PASCAL procedure for this method may be found in Appendix 2.

In addition to testing of those algorithms not yet considered, which may include contacting the authors for further clarification of their work, I plan to continue development of the program in the area of Datum transformations, program robustness, and improved file I/O capabilities.

7.0 REFERENCES

Baird, R. W., Cartesian Coordinate to Geodetic Coordinate Conversions, White Sands Missile Range Data Reduction Directorate, June 1, 1964.

Berger, W. J. and Ricupito, J. R., Geodetic Latitude and Altitude of a Satellite. ARS Journal, Vol. 30, No. 9, September 1960, pp. 901-902.

Borkowski, K.M., Transformation of Geocentric to Geodetic Coordinates without Approximations, Astrophys. Space Sci., 139,pp. 1-4, and 146 (1988), p.201.

Bowring, B. R., Transformation From Spatial to Geographical Coordinates. Survey Review, 23 (181): 323-327, (1976).

Bowring, B. R., The Accuracy of Geodetic Latitude and Height Equations, Survey Review, 28, (1985) pp. 202-206.

Churchyard, J. N., Three Degree of Freedom Trajectory Program Description, Missile System Div., Atlantic Research Corp., Rept. AR/MSD-127-OOP, July 1968.

Defense Mapping Agency, Department of Defense World Geodetic System 1984 - Its Definition and Relationships With Local Geodetic Systems. DMA Technical Report 8350.2, Defense Mapping Agency, Washington, D.C., 1987.

Eissfeller, B., A Taylor Series Expansion For the Transformation Problem of Cartesian Baseline Components Into Ellipsoidal Coordinate Differences, GPS Research 1985 at the Institute of Astronomical and Physical Geodesy, Universitat der Bundeswehr Munchen, Heft 19, pp. 47-64.

Gersten, R. H., Geodetic Sub-Latitude and Altitude of a Space Vehicle. The Journal of the Astronautical Sciences, Vol. 3, No. 1, Spring 1961, pp. 28--29.

Hedgley, D.R. Jr., An Exact Transformation From Geocentric to Geodetic Coordinates For Nonzero Altitudes, NASA-TR-R--458; H-909, Mar. 1976, National Aeronautics and Space Administration Hugh L. Dryden Flight Research Center, Edwards, Ca.

Hedman, E. L. Jr., A High--Accuracy Relationship Between Geocentric Cartesian Coordinates and Geodetic Latitude and

Altitude. Journal of Spacecraft and Rockets, Engineering Notes, Vol. 7, July-Aug. 1970, pp. 993-995.

Heffron, W. G. Jr. and Watson, S.B., Relationships Between Geographic and Inertial Coordinates. J. Spacecraft, Vol. 4, No. 4, April 1967, pp. 531-532.

Heikkinen, M., Geschlossene Formeln zur Berechnung raumlicher geodätischer Koordinaten aus rechtwinkligen Koordinaten, Zeitschrift Vermess, 107, pp.207-211, 1982.

Heiskanen, W. A., and Moritz, H., Physical Geodesy. W. H. Freeman and Co., San Francisco, Ca. and London, pp. 181-183, (1967).

Hirvonen, R. A., New Theory of the Gravimetric Geodesy, Annales Academiae Scientiarum Fennicae, Series A, III, Geologica-Geographica, 56, Helsinki, Finland, 1960.

Hirvonen, R. A., and Moritz, H., Practical Computations of Gravity at High Altitudes, Report No. 27, Inst. Geod. Phot. Cart. Ohio State Univ. pp. 4-5, (1963).

Long, S.A.T., General-Altitude Transformations Between Geocentric and Geodetic Coordinates, Celestial Mechanics 12, (1975), 225-230, by D. Reidel Publishing Co., Dordrecht-Holland.

Lupash, L. O., A New Algorithm for the Computation of the Geodetic Coordinates as a Function of Earth Centered Earth-Fixed Coordinates. Journal of Guidance, Control, and Dynamics, Vol. 8, Nov.-Dec. 1985, pp. 787-789.

Moritz, H., Geodetic Reference System, Bulletin Géodésique, 54, 395-405, 1980.

Moritz, H., Geodetic Reference System, Bulletin Géodésique, 58, 388-398, 1984.

Morrison, J. and Pines, S., The Reduction from Geocentric to Geodetic Coordinates. The Astronomical Journal, Vol. 66, No. 1, February 1961.

Nautiyal, A., Algorithm to Generate Geodetic Coordinates from Earth-Centered Earth-Fixed Coordinates, Journal of Guidance, Control, and Dynamics, Engineering Notes, Vol. 11, No. 3, May-June 1988, pp. 281-283.

Olson, D. K., Optimal Coordinate Transformation Formulas. Data Processing Division Range Instrumentation Systems Department, Pacific Missile Test Center, Point Mugu, California, Technical Note No. 3450-1-77, (1977).

Paul, M. K., A Note on Computation of Geodetic Coordinates from Geocentric (Cartesian) Coordinates Bulletin Géodésique, No. 108, pp. 135-139, (1973).

Pick, M., Closed Formulae for Transformation of The Cartesian Coordinate System Into a System of Geodetic Coordinates, Studia geoph. et geod. 20 (1985), pp. 112-119.

Purcell, E. W. and Cowan, W. B., Relating Geodetic Latitude and Altitude to Geocentric Latitude and Radius Vector. ARS Journal, July 1961, pp. 932-934.

Rapp, R. H., Geometric Geodesy, Vols. I & II Lecture notes, published by Dept. of Geodetic Sci. and Surveying, Ohio State Univ., Columbus, Ohio, (1984).

Soler, T., On Differential Transformations Between Cartesian and Curvilinear (Geodetic) Coordinates. Report No.

236, Dept. of Geodetic Sce., Ohio State Univ., Columbus, Ohio, 1976.

Soler, T., and Hothem, L. D., Important Parameters Used In Geodetic Transformations, Journal of Surveying Engineering, ASCE, Vol. 115, No. 4, November, 1989, pp. 414-417.

Torge, W., Geodäsie. Slg. Göschen Nr. 2163, W. de Gruyter, Berlin-New York 1975.

Torge, W., Geodesy. Walter de Gruyter Publishers, New York, N.Y., (1980).

Vanicêk, P. and Krakiwsky, E.J. (1986), Geodesy: The Concepts, Second Edition, 1986,North Holland Publishing Co. pp. 323-327.

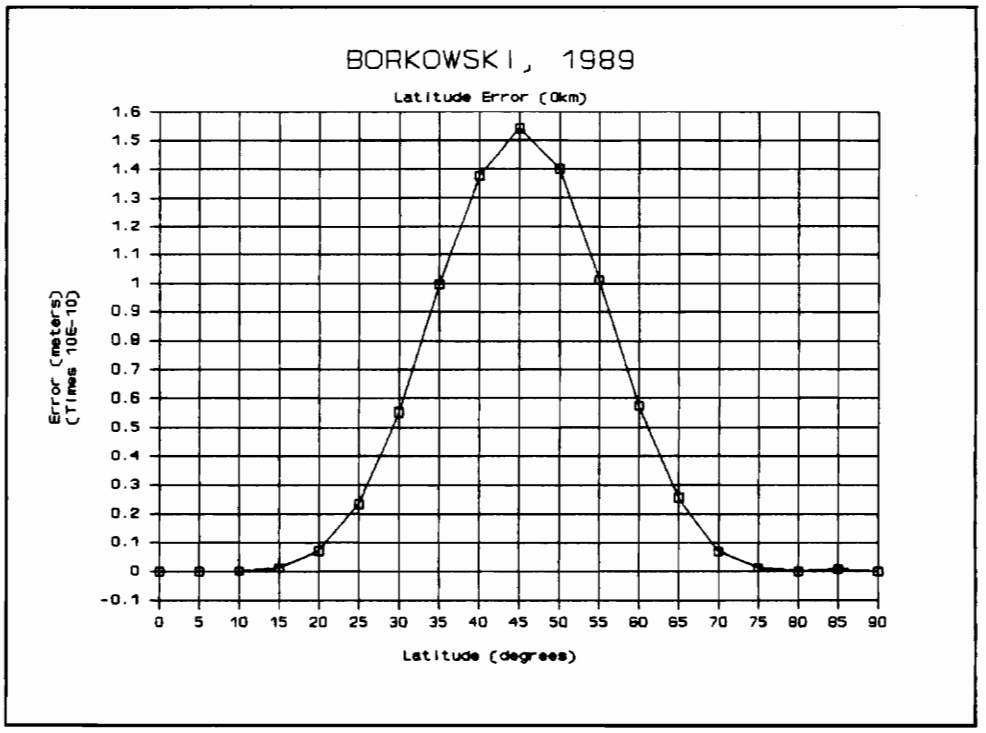
Varão-Romáo, M. Salomé, Transformacao de Coordenadas Cartesianas Tridimensionais em Geográficas por um processo directo, Revista do Instituto Geogràfico e Cadastral, No. 7, Dezembro 1987, Publicacáo Anual, pp. 87-94.

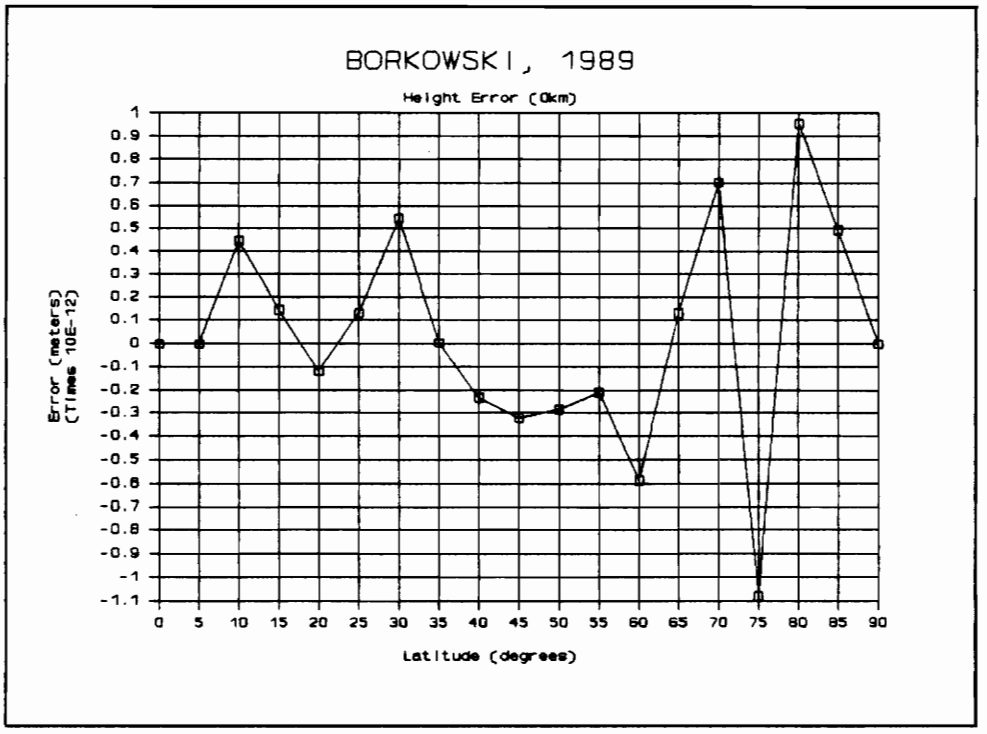
Wei, Z., Positioning With NAVSTAR, The Global Positioning System. Department of Geodetic Science and Surveying, The

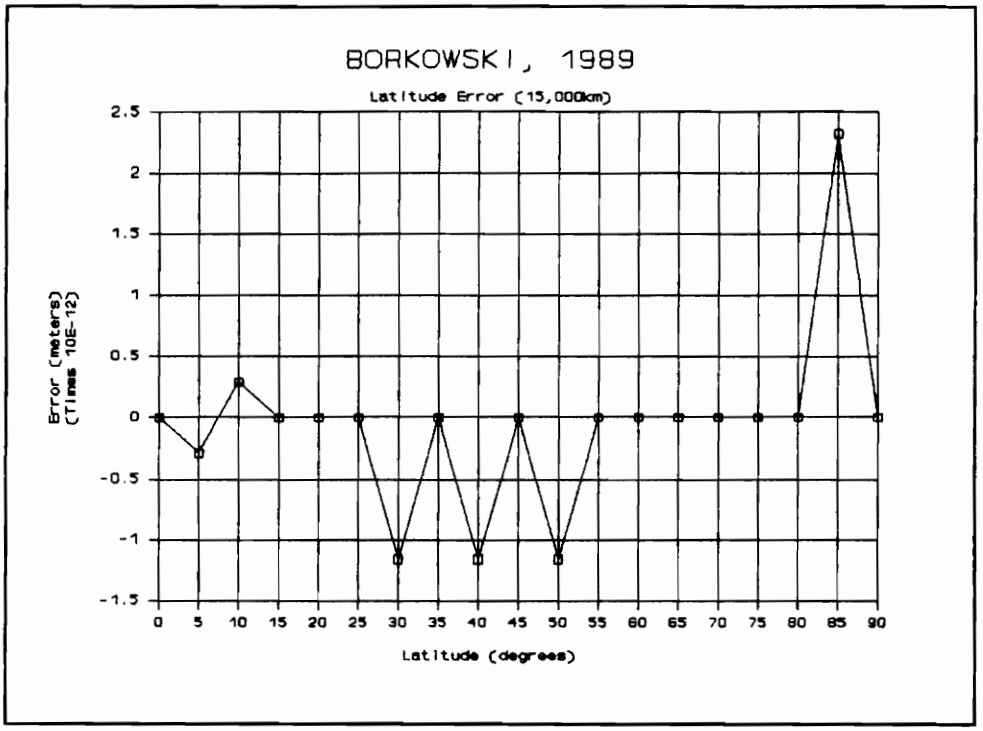
Ohio State University, Report No. 370, Columbus, OH. August, 1986.

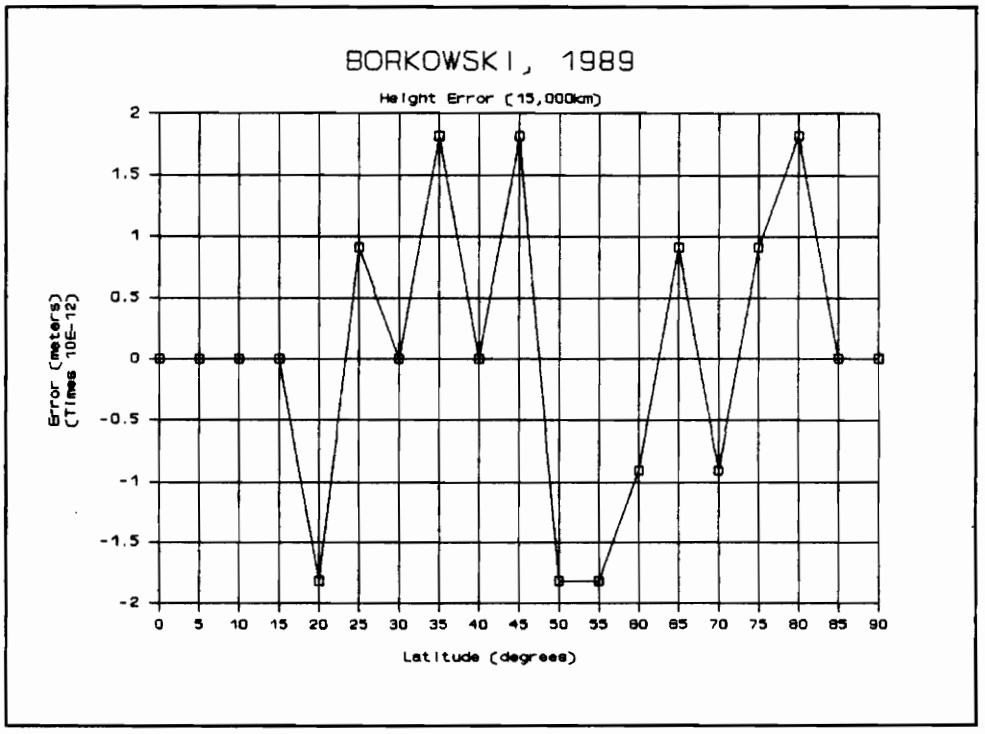
8.0 APPENDICES

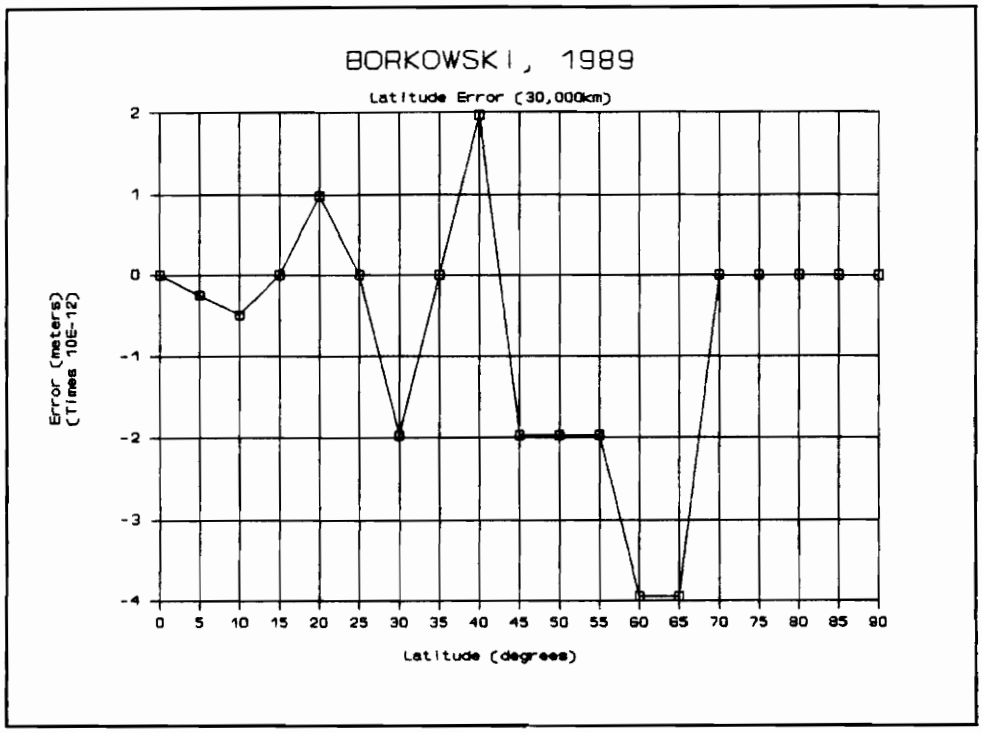
8.1 Appendix A, Graphs of Error Curves

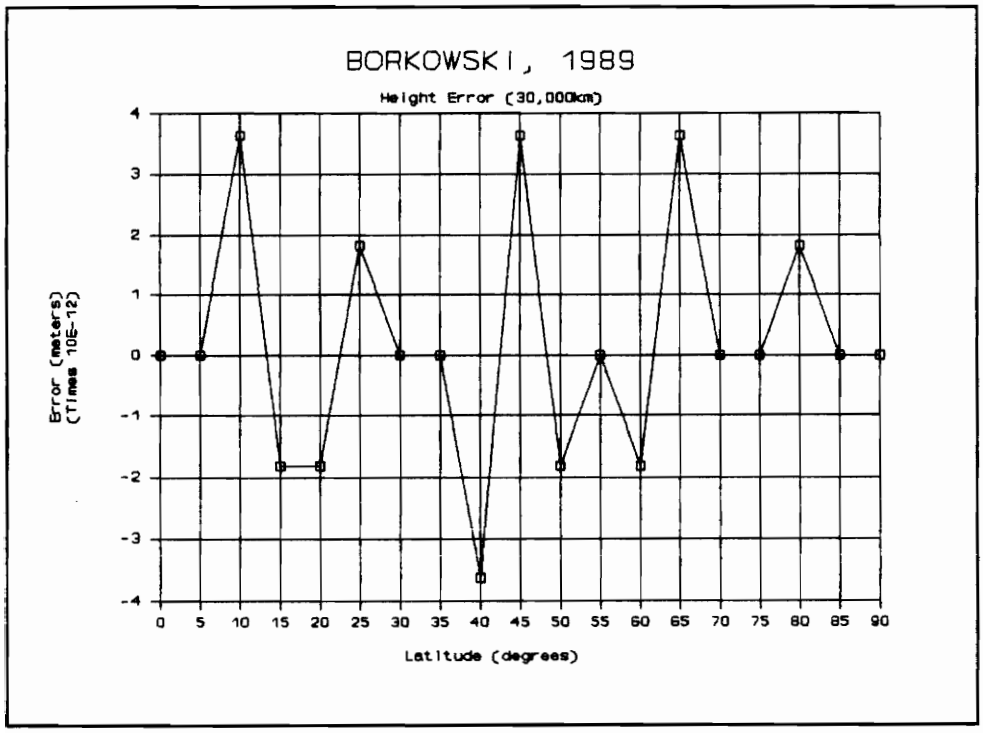

BORKOWSKI, 1989
Latitude Error (0km)
Error (meters)
(Times 10E-10)
1.6
1.5
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

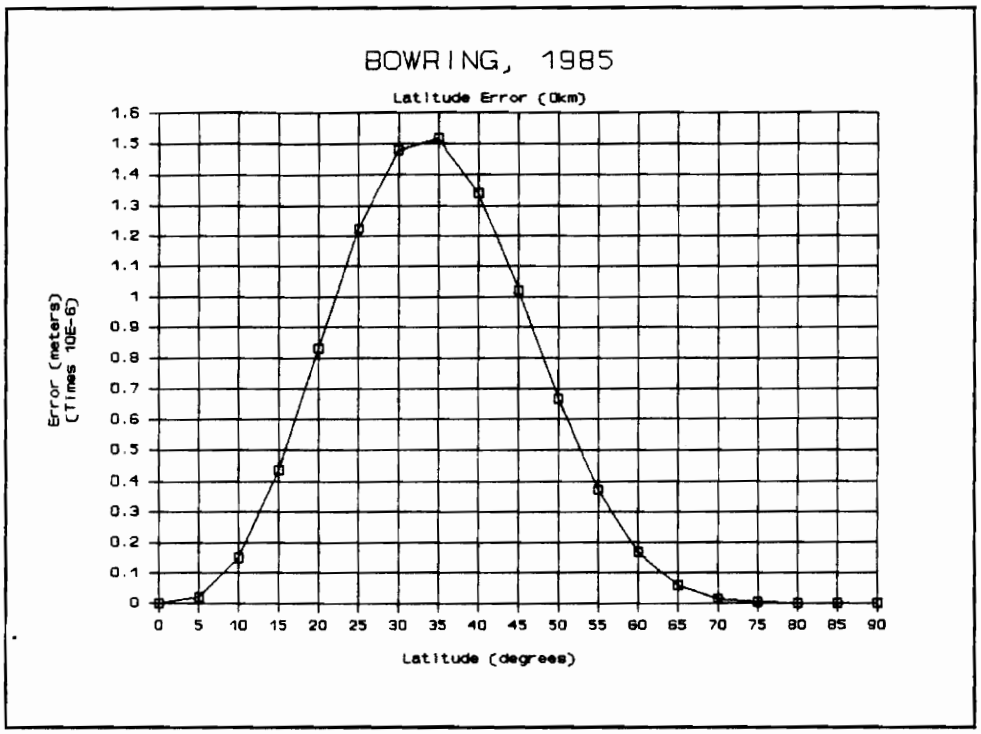

BORKOWSKI, 1989
Height Error (0km)
Error (meters)
(Times 10E-12)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
-0.8
-0.9
-1
-1.1
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

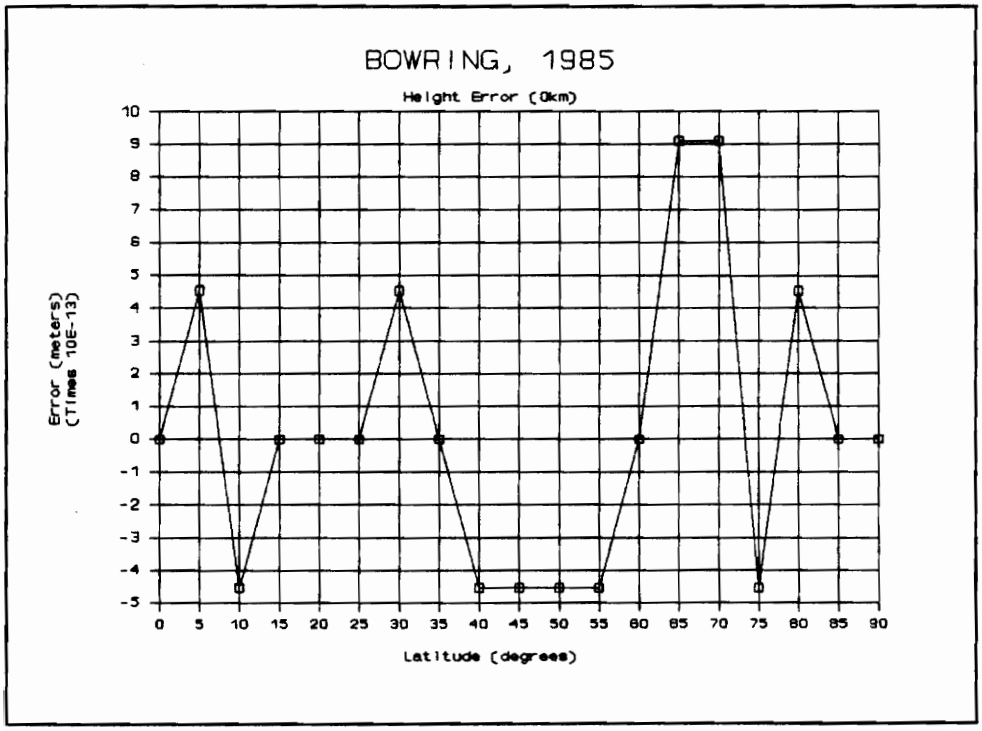

BORKOWSKI, 1989
Latitude Error (15,000km)
Error (meters)
(Times 10E-12)
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

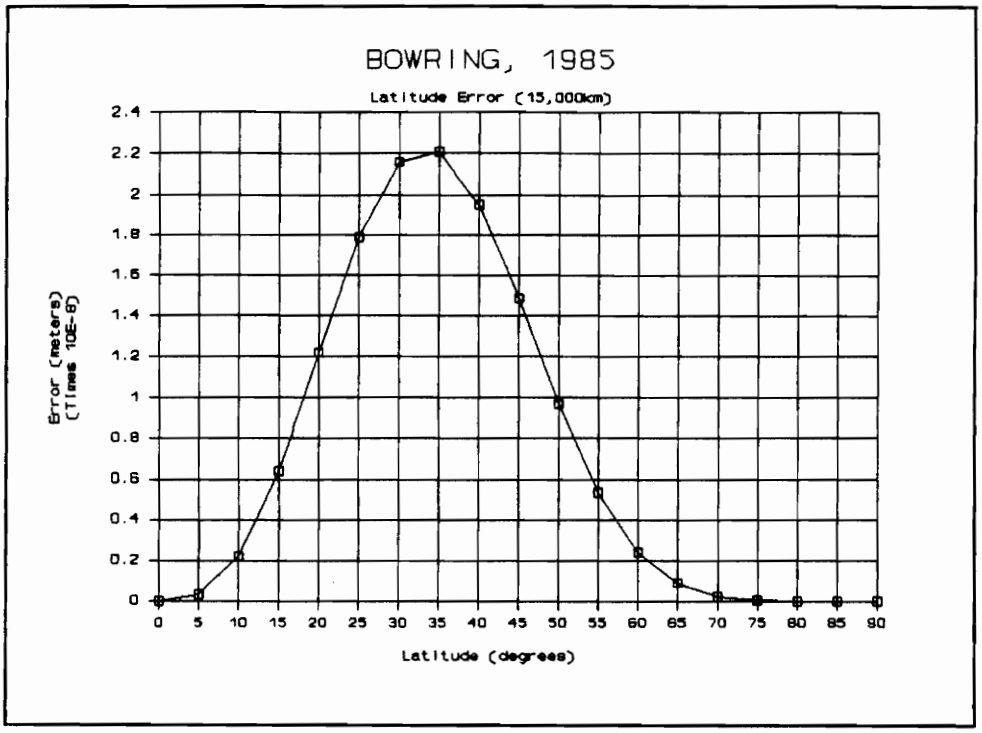

BORKOWSKI, 1989
Height Error (15,000km)
Error (meters)
(Times 10E-12)
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

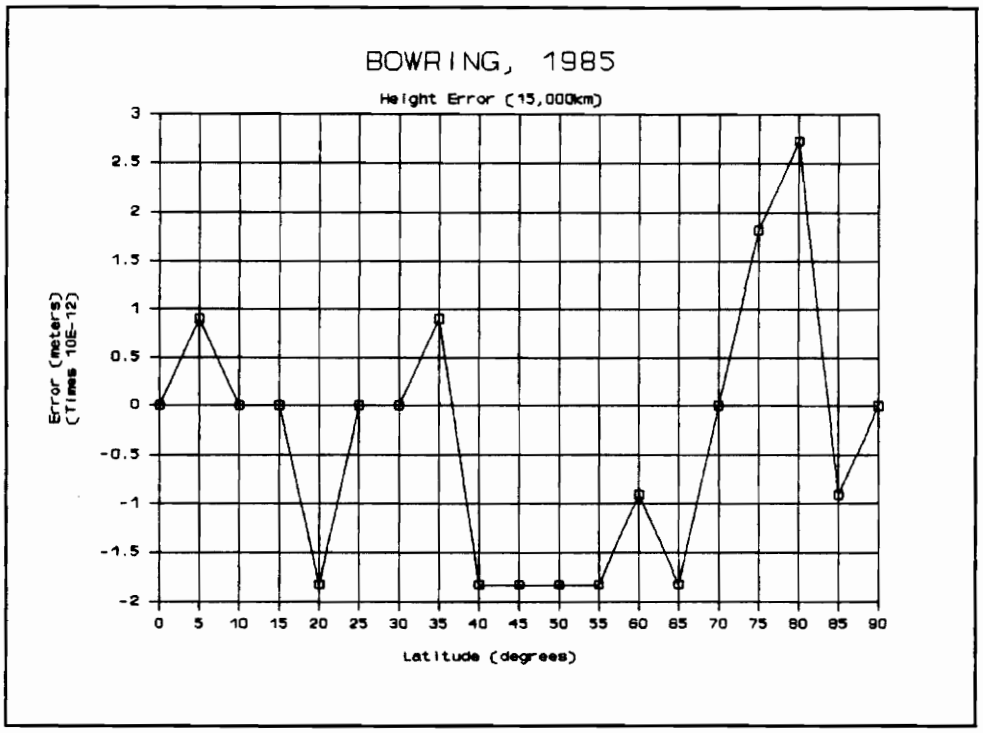

BORKOWSKI, 1989
Latitude Error (30,000km)
Error (meters)
(Times 10E-12)
2
1
0
-1
-2
-3
-4
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

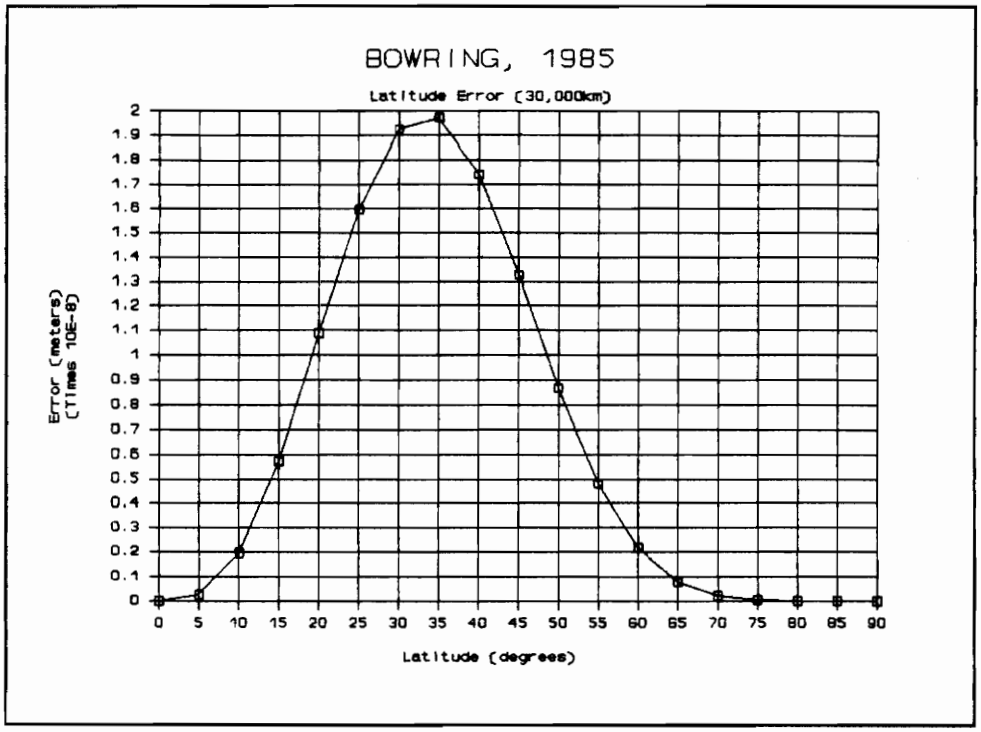

BORKOWSKI, 1989
Height Error (30,000km)
Error (meters)
(Times 10E-12)
4
3
2
1
0
-1
-2
-3
-4
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

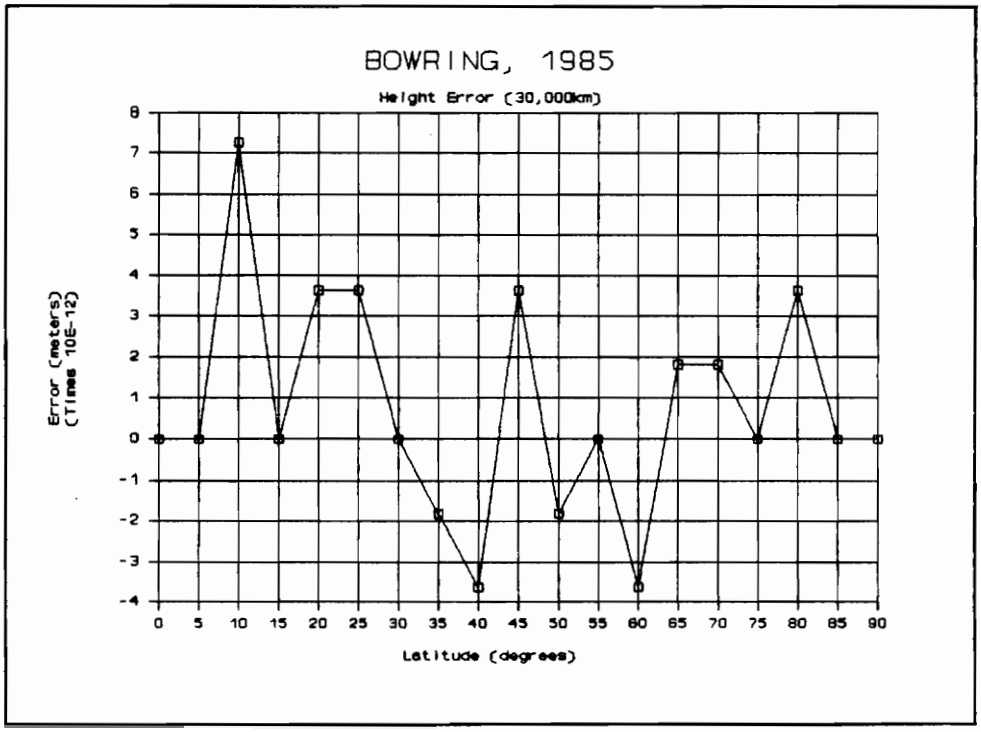

BOWRING, 1985
Latitude Error (0km)
Error (meters)
(Times 10E-6)
1.6
1.5
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

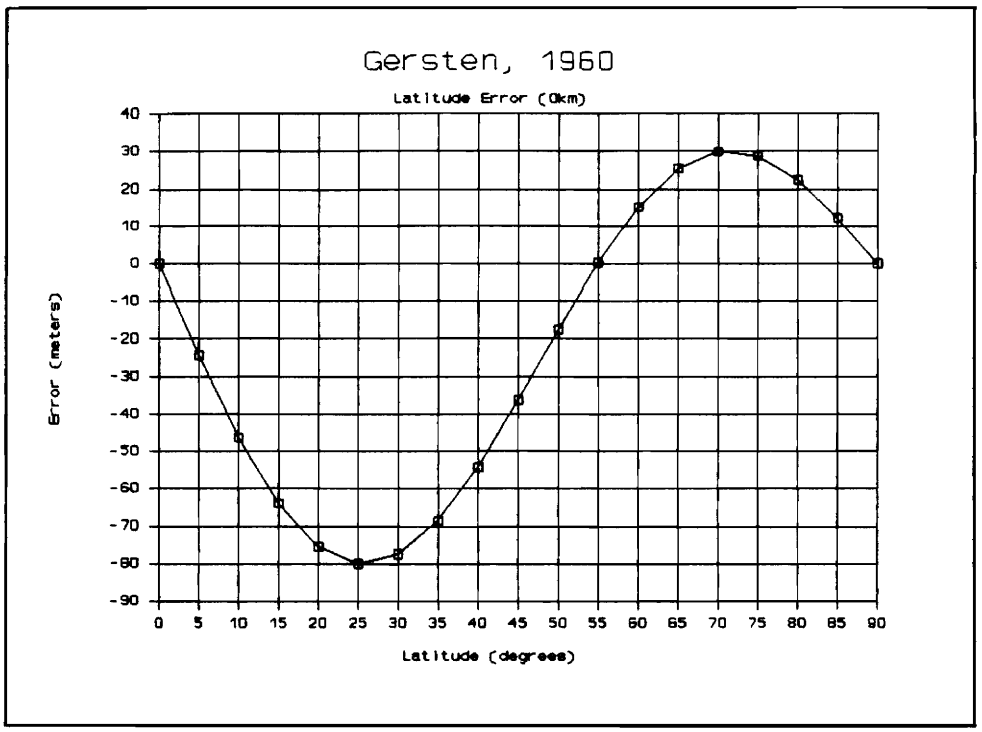

BOWRING, 1985
Height Error (0km)
Error (meters)
(Times 10E-13)
10
9
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-5
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

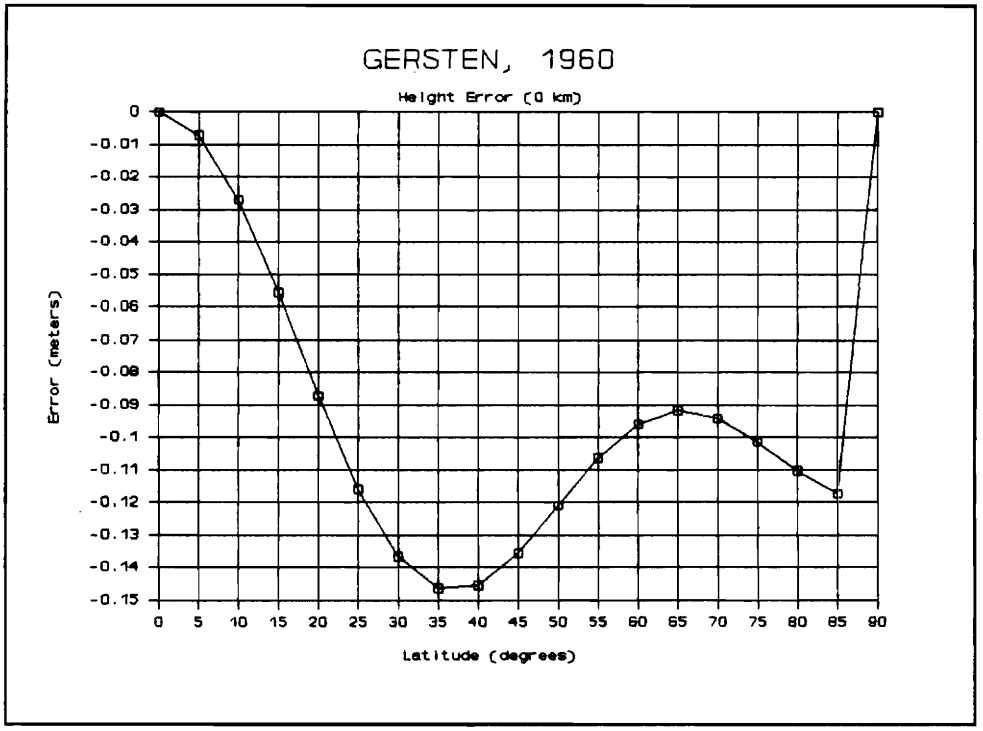

BOWRING, 1985
Latitude Error (15,000km)
Error (meters)
(Times 10E-8)
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

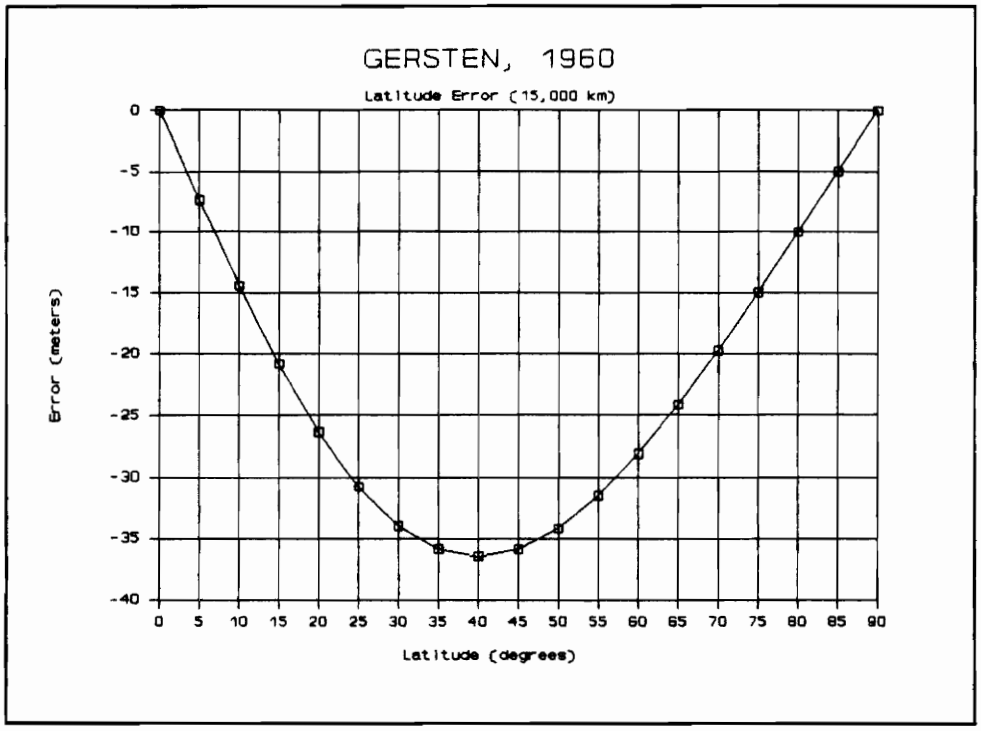

BOWRING, 1985
Height Error (15,000km)
Error (meters)
(Times 10E-12)
3
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

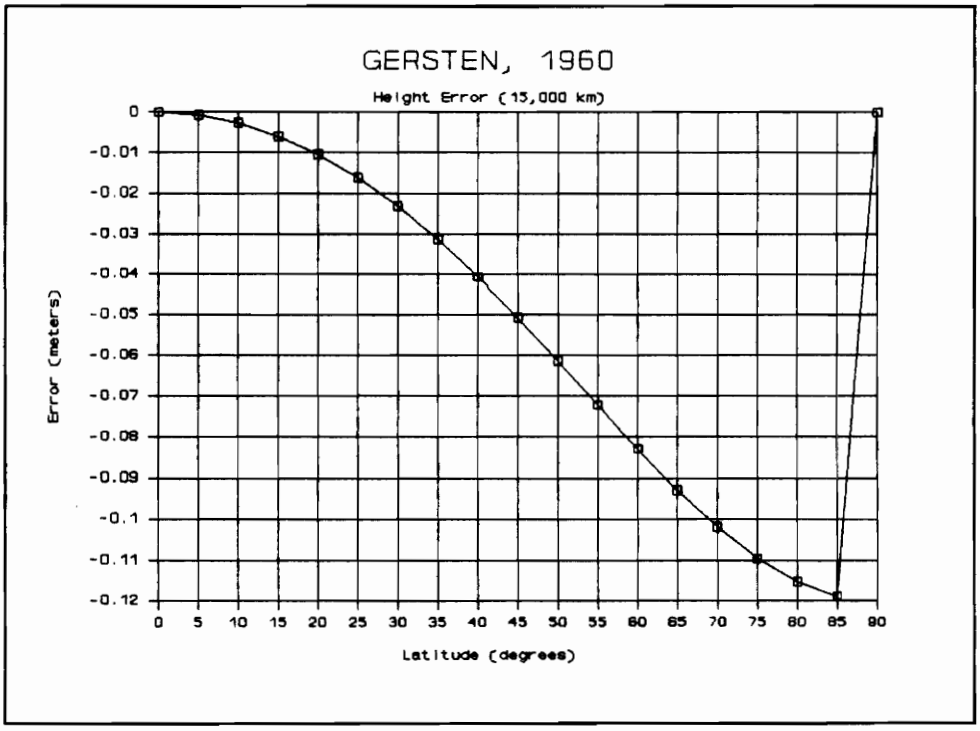

BOWRING, 1985
Latitude Error (30,000km)
Error (meters)
(Times 10E-8)
2
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

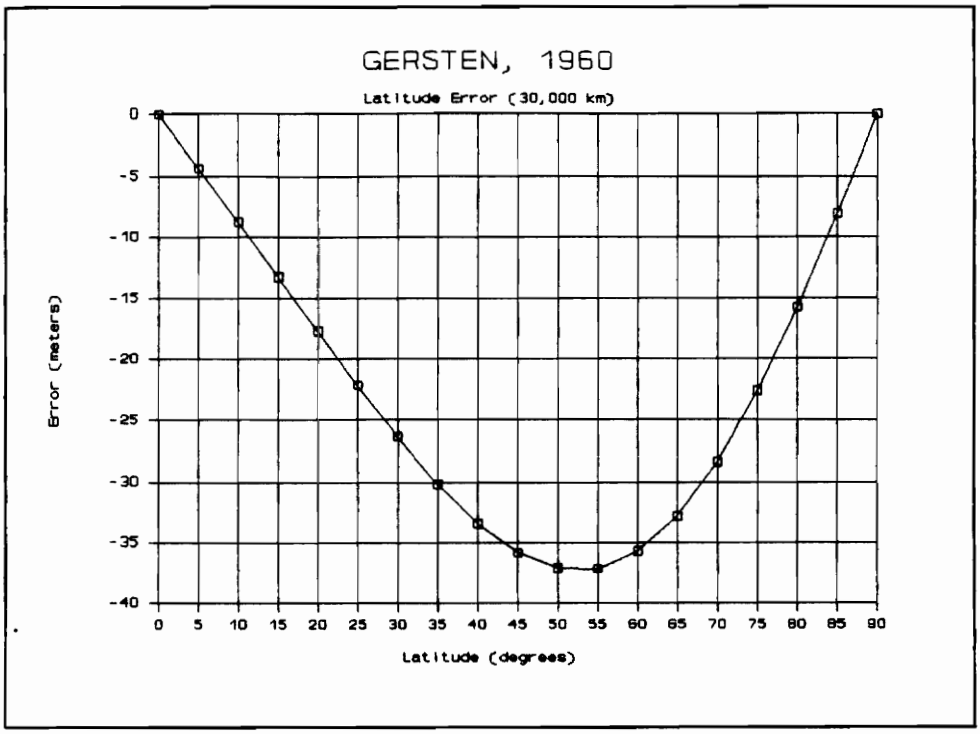

BOWRING, 1985
Height Error (30,000km)
Error (meters)
(Times 10E-12)
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

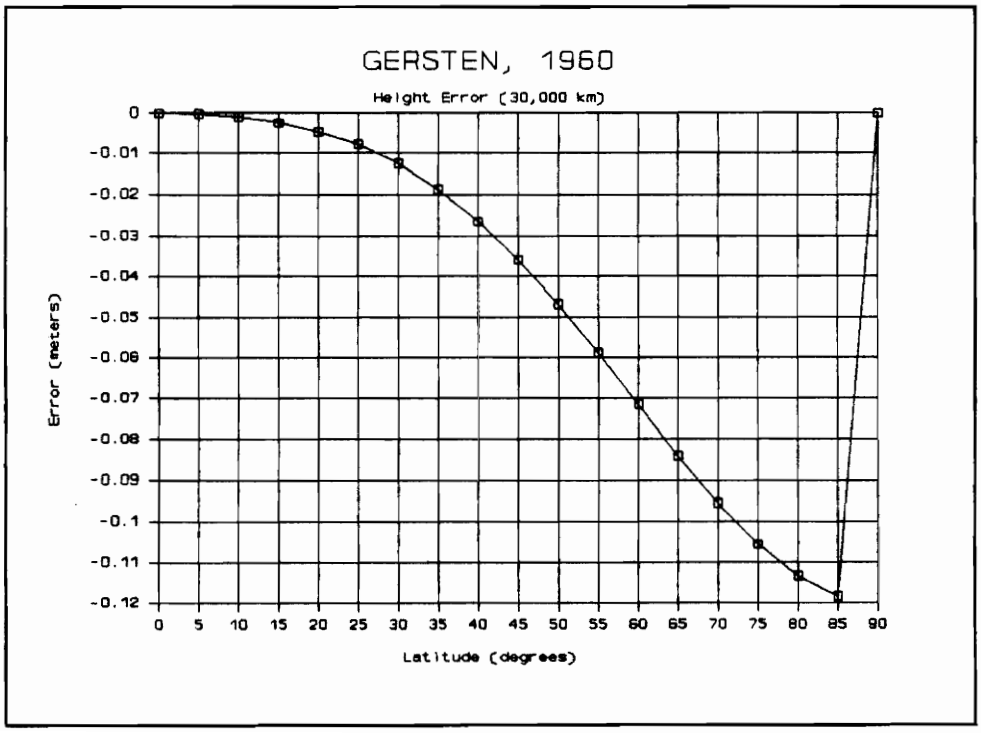

Gersten, 1960
Latitude Error (0km)
Error (meters)
40
30
20
10
0
-10
-20
-30
-40
-50
-60
-70
-80
-90
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

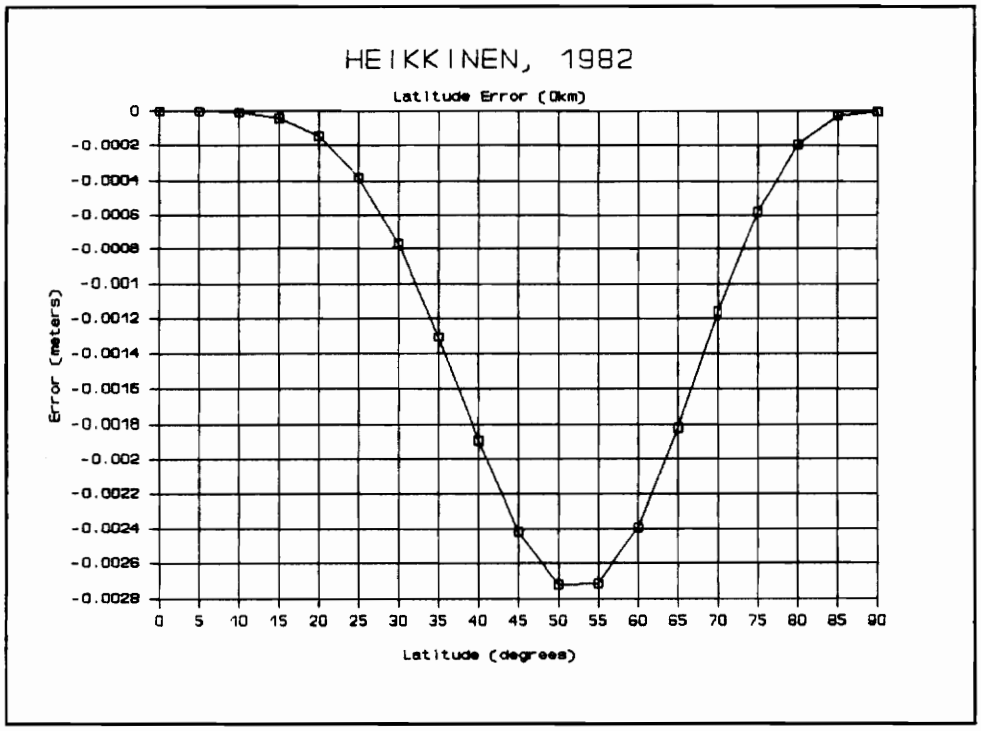

GERSTEN, 1960
Height Error (0 km)
Error (meters)
0
-0.01
-0.02
-0.03
-0.04
-0.05
-0.06
-0.07
-0.08
-0.09
-0.1
-0.11
-0.12
-0.13
-0.14
-0.15
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

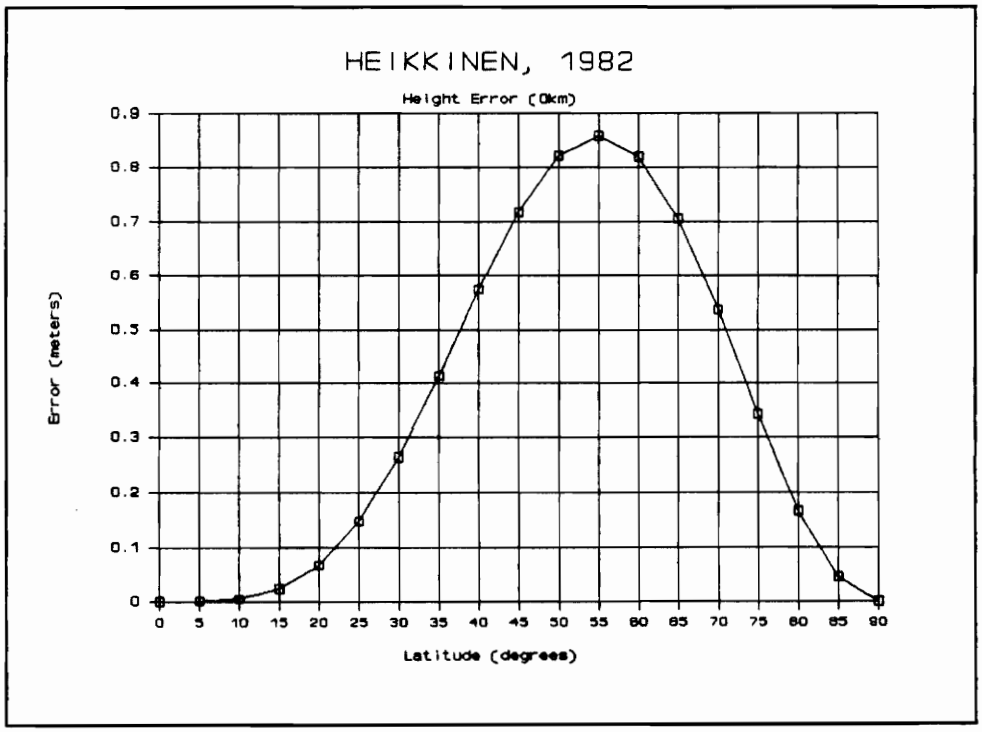

GERSTEN, 1960
Latitude Error (15,000 km)
Error (meters)
0
-5
-10
-15
-20
-25
-30
-35
-40
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

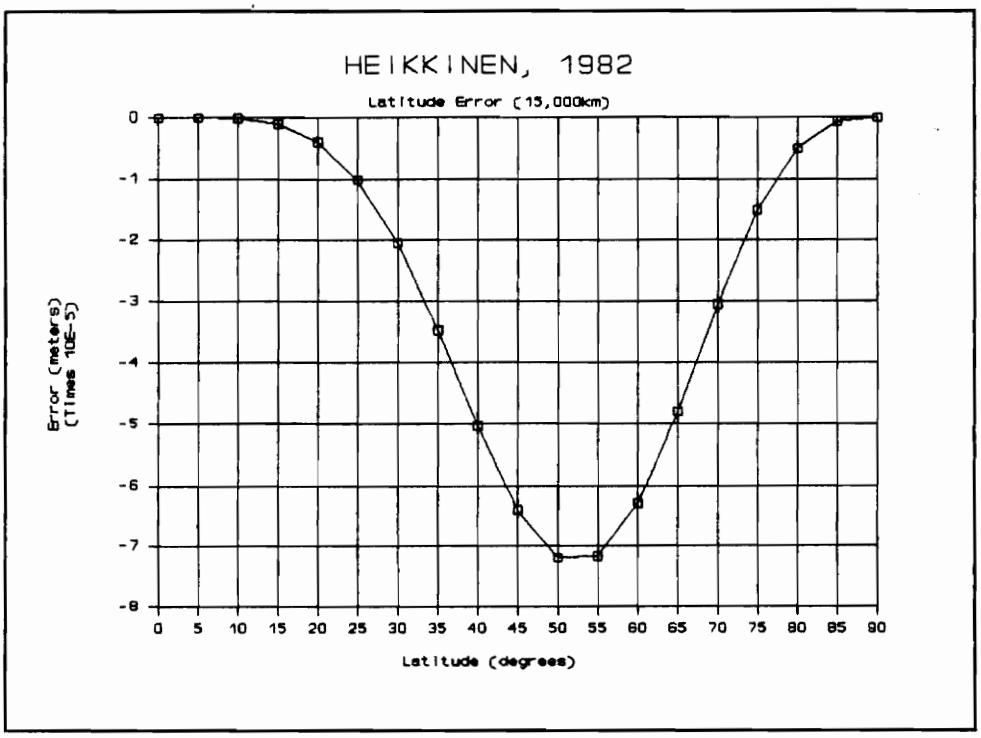

GERSTEN, 1960
Height Error (15,000 km)
Error (meters)
0
-0.01
-0.02
-0.03
-0.04
-0.05
-0.06
-0.07
-0.08
-0.09
-0.1
-0.11
-0.12
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

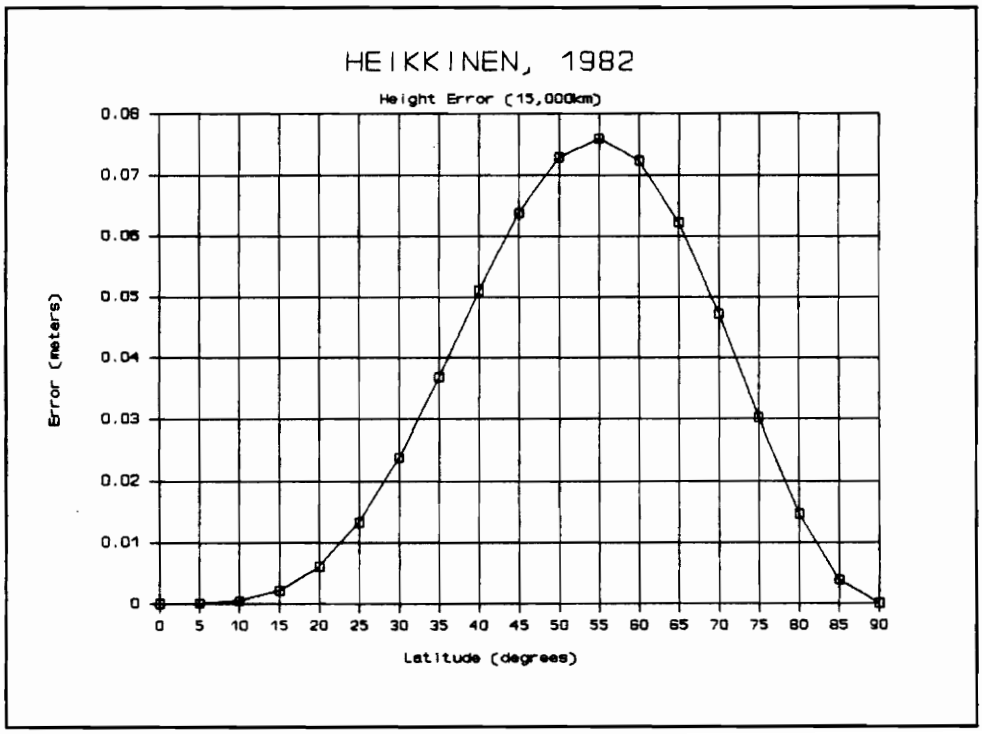

GERSTEN, 1960
Latitude Error (30,000 km)
Error (meters)
0
-5
-10
-15
-20
-25
-30
-35
-40
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

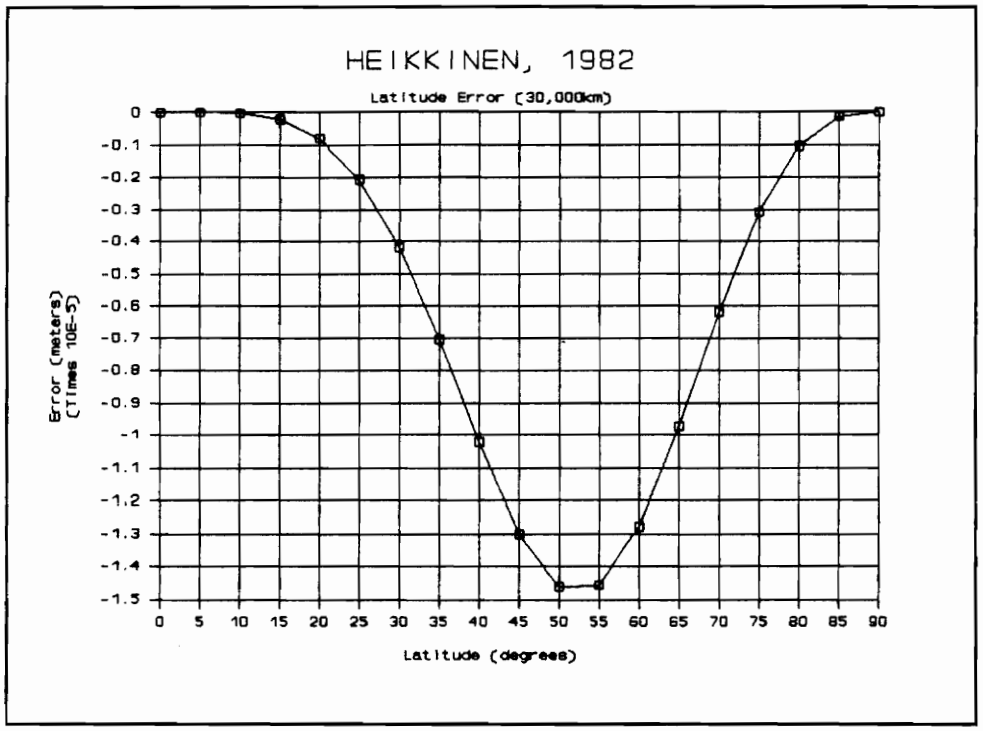

GERSTEN, 1960
Height Error (30,000 km)
Error (meters)
0
-0.01
-0.02
-0.03
-0.04
-0.05
-0.06
-0.07
-0.08
-0.09
-0.1
-0.11
-0.12
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

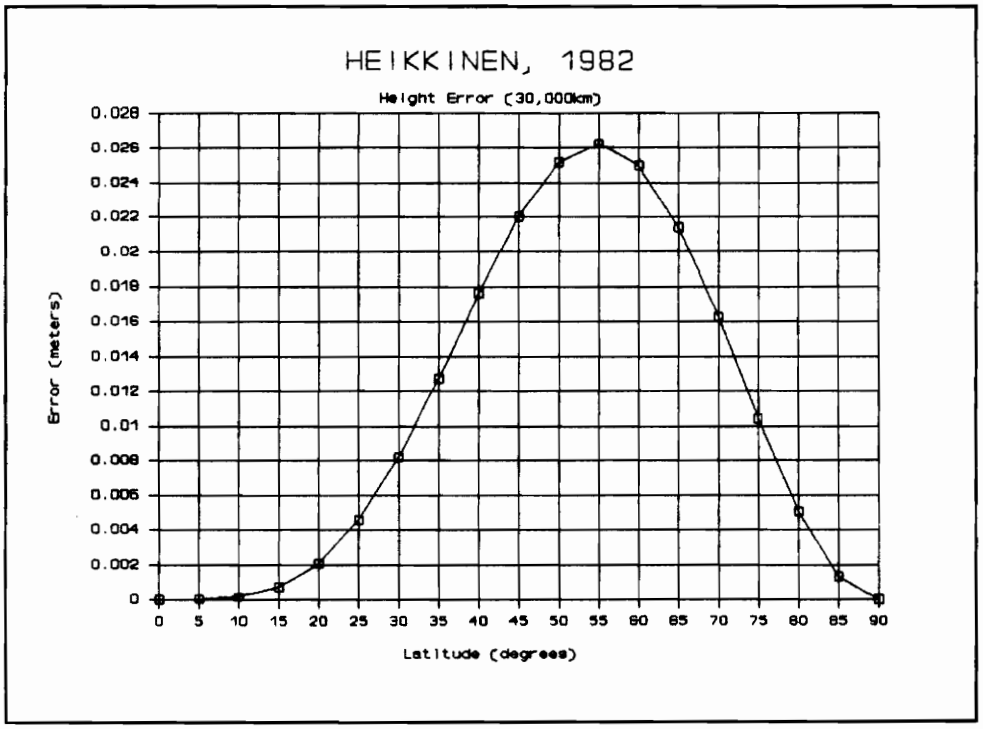

HEIKKINEN, 1982
Latitude Error (0km)
Error (meters)
0
-0.0002
-0.0004
-0.0006
-0.0008
-0.001
-0.0012
-0.0014
-0.0016
-0.0018
-0.002
-0.0022
-0.0024
-0.0026
-0.0028
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

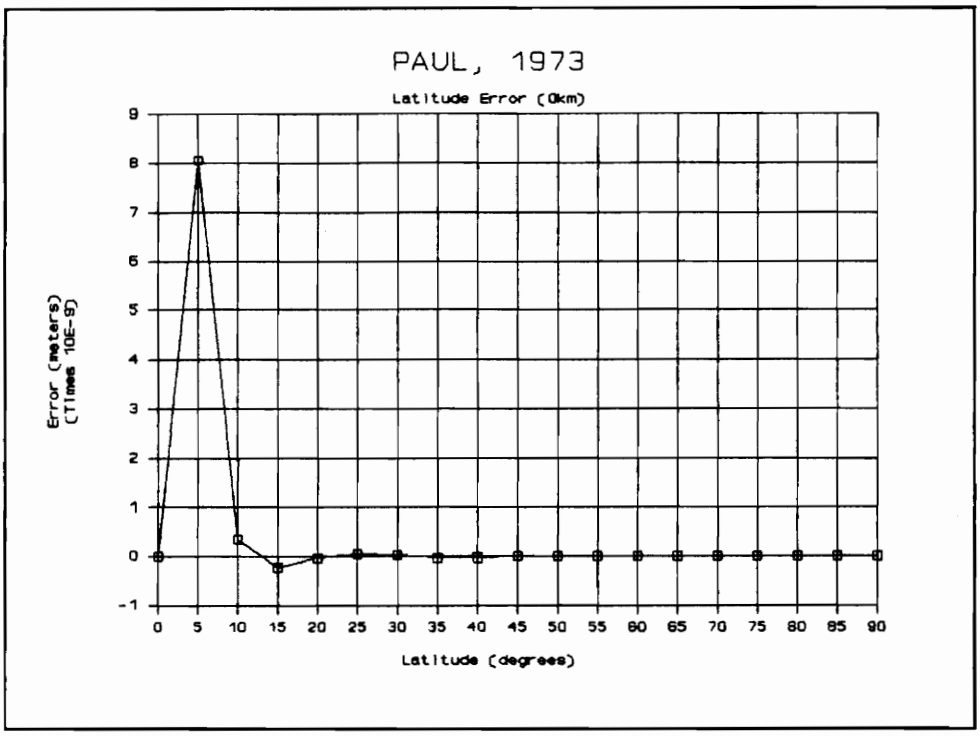

HEIKKINEN, 1982
Height Error (0km)
Error (meters)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

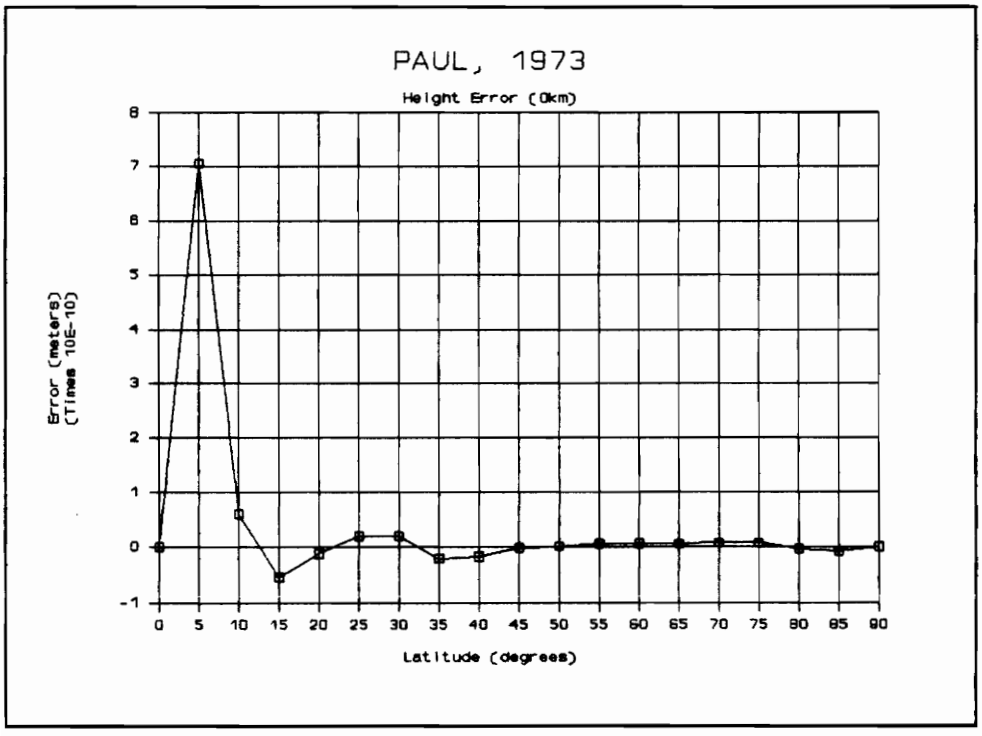

HEIKKINEN, 1982
Latitude Error (15,000km)
Error (meters)
(Times 10E-5)
0
-1
-2
-3
-4
-5
-6
-7
-8
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

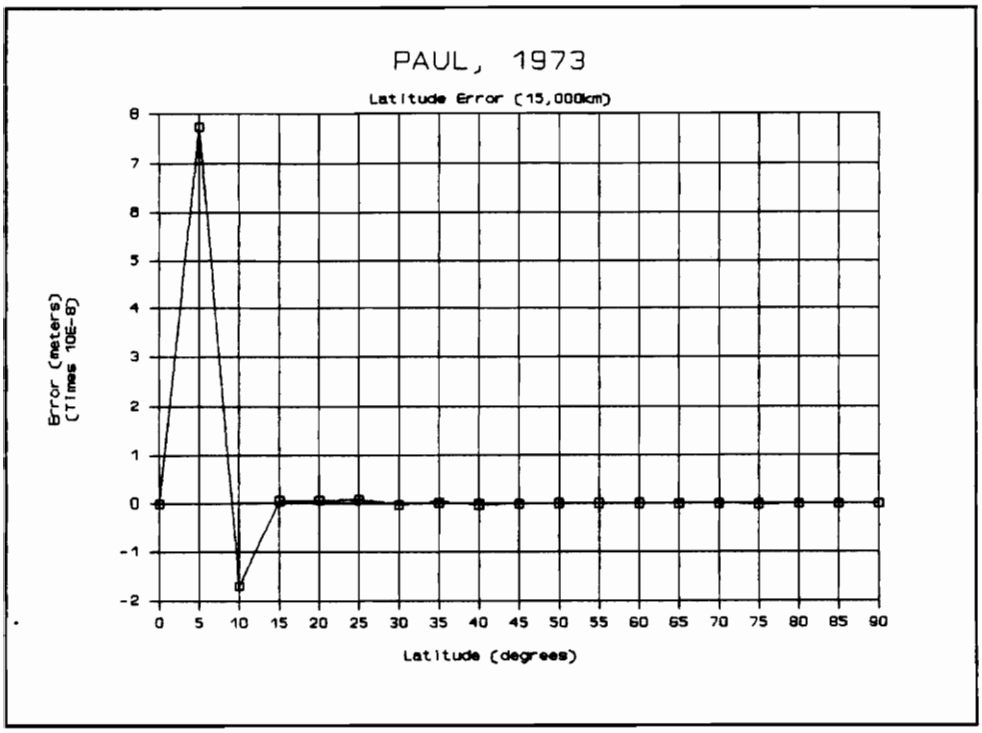

HEIKKINEN, 1982
Height Error (15,000km)
Error (meters)
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

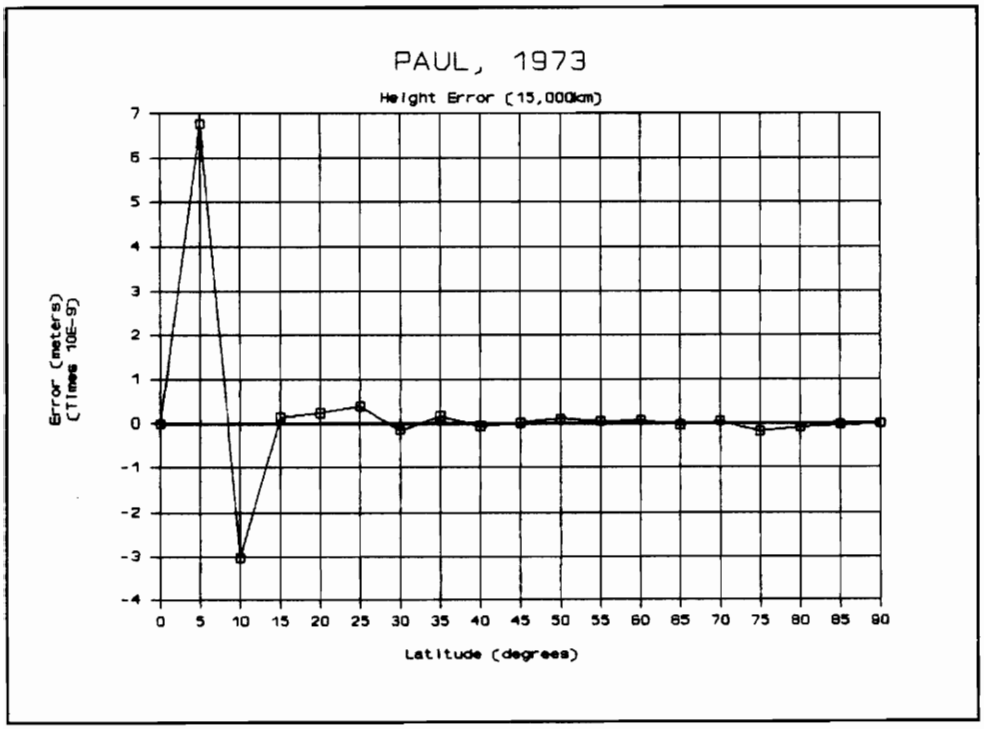

HEIKKINEN, 1982
Latitude Error (30,000km)
Error (meters)
(Times 10E-5)
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
-0.8
-0.9
-1
-1.1
-1.2
-1.3
-1.4
-1.5
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

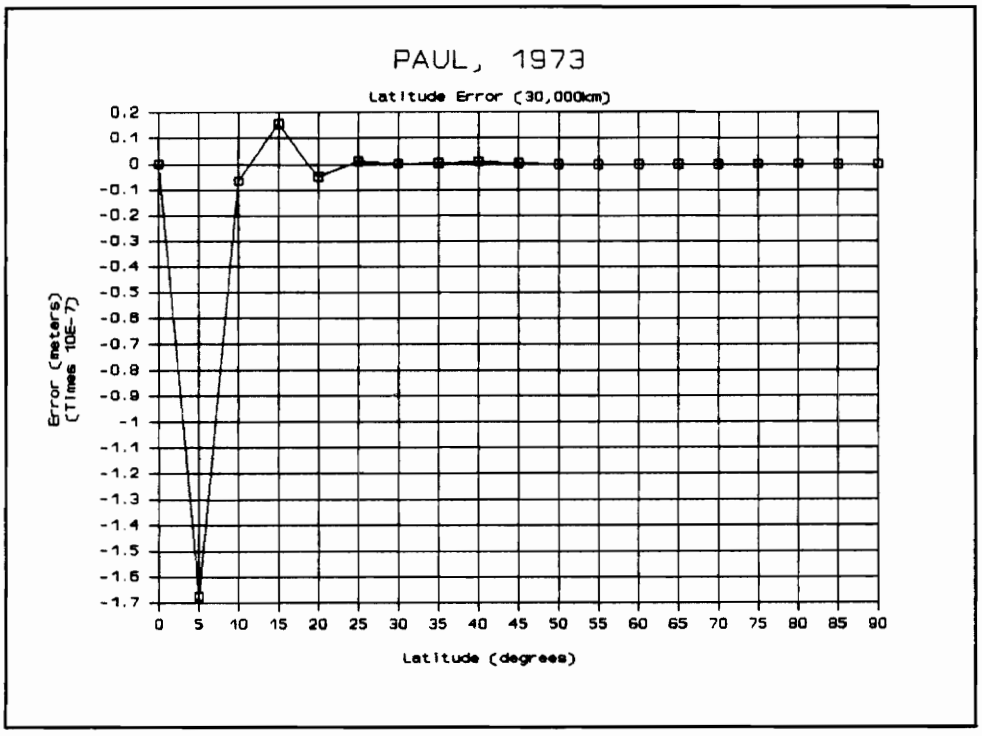

HEIKKINEN, 1982
Height Error (30,000km)
Error (meters)
0.028
0.026
0.024
0.022
0.02
0.018
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

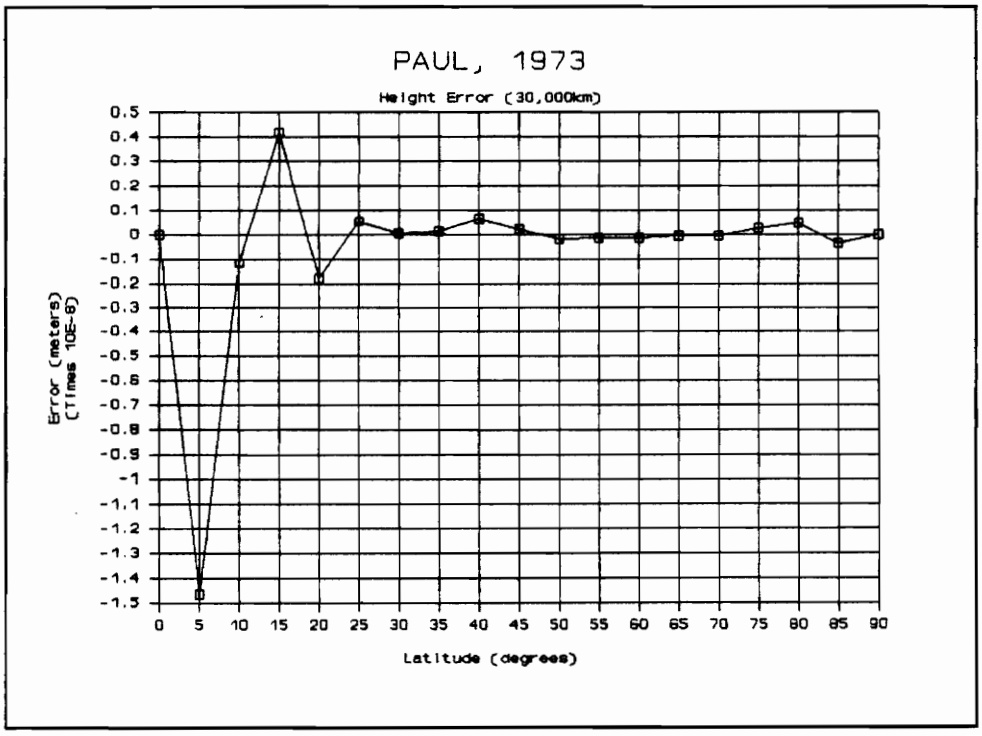

PAUL, 1973
Latitude Error (0km)
Error (meters)
(Times 10E-9)
9
8
7
6
5
4
3
2
1
0
-1
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

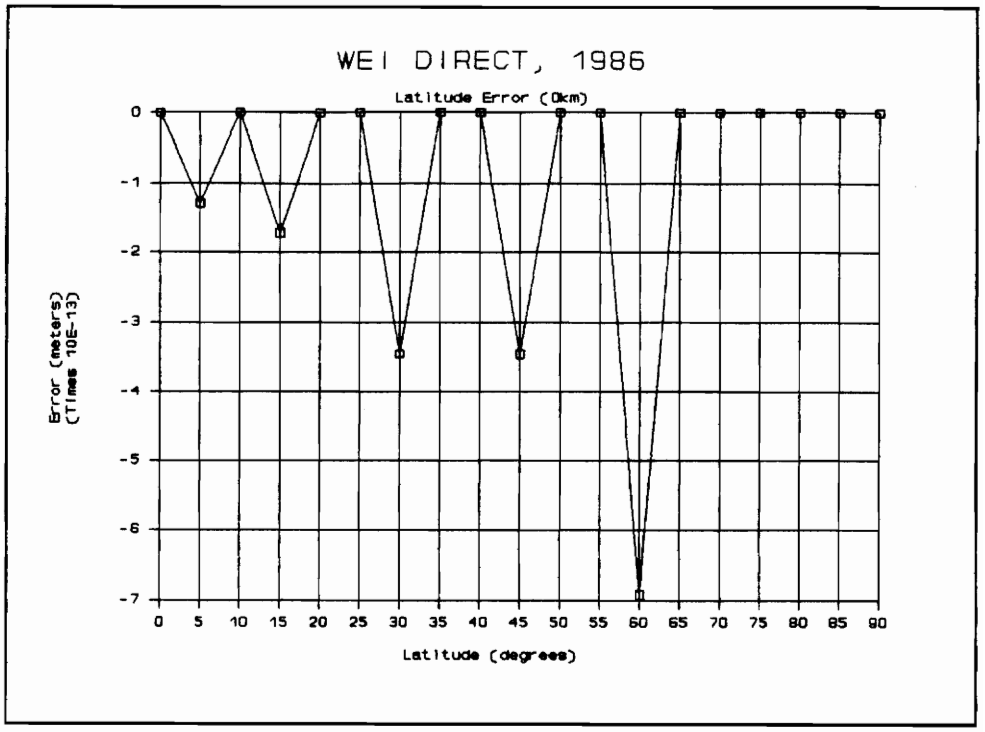

PAUL, 1973
Height Error (0km)
Error (meters)
(Times 10E-10)
8
7
6
5
4
3
2
1
0
-1
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

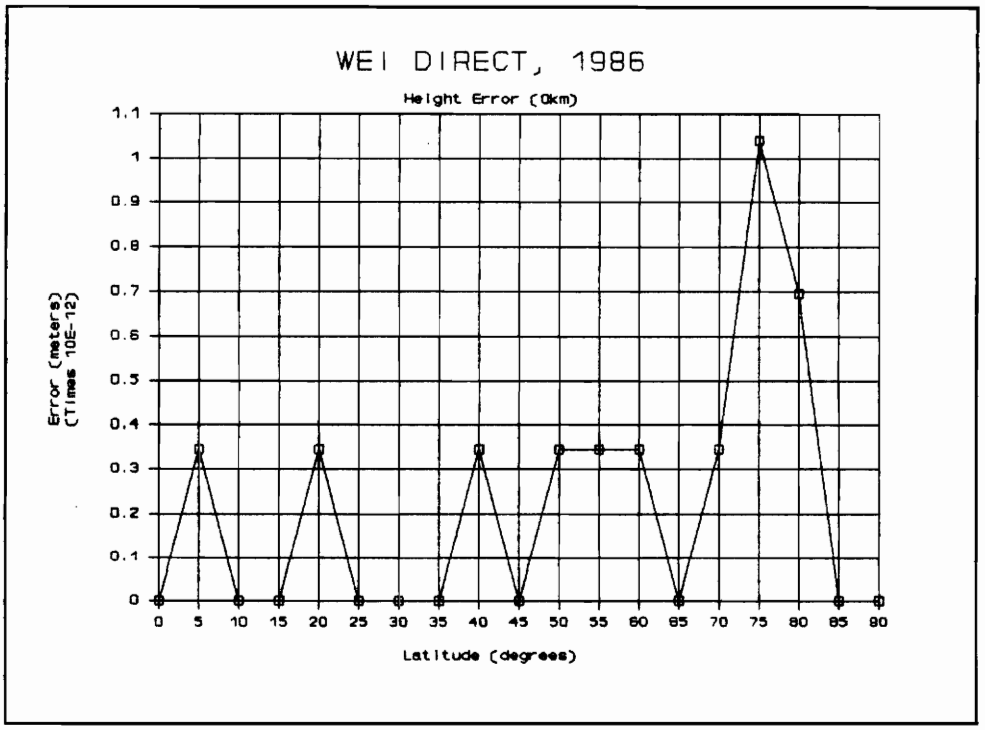

PAUL, 1973
Latitude Error (15,000km)
Error (meters)
(Times 10E-8)
8
7
6
5
4
3
2
1
0
-1
-2
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

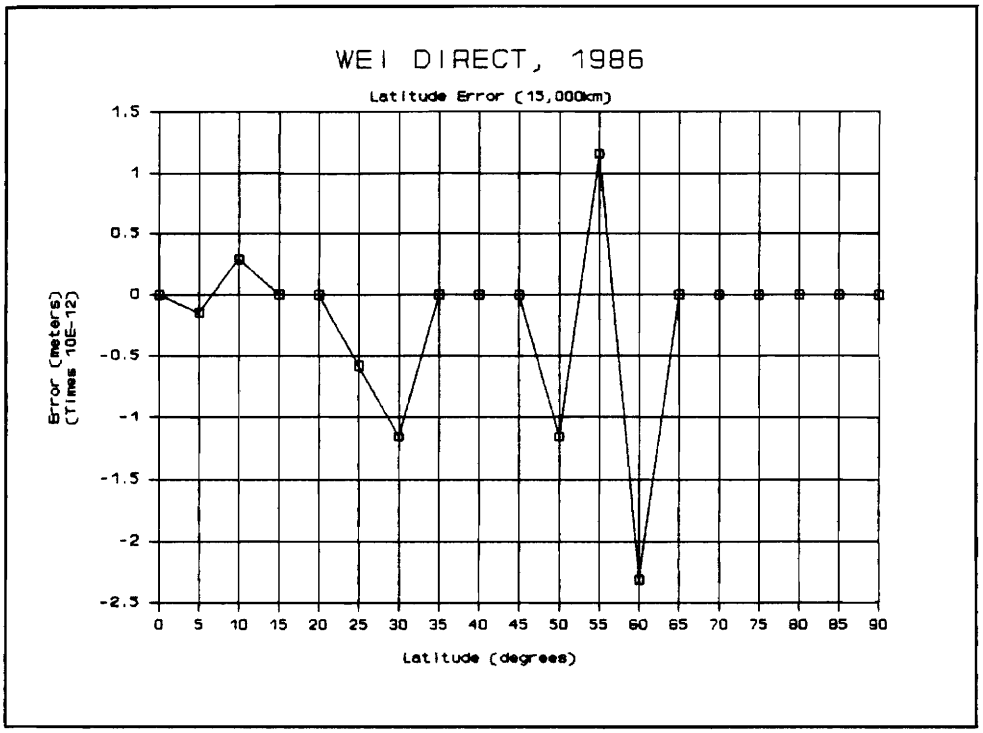

PAUL, 1973
Height Error (15,000km)
Error (meters)
(Times 10E-9)
7
6
5
4
3
2
1
0
-1
-2
-3
-4
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

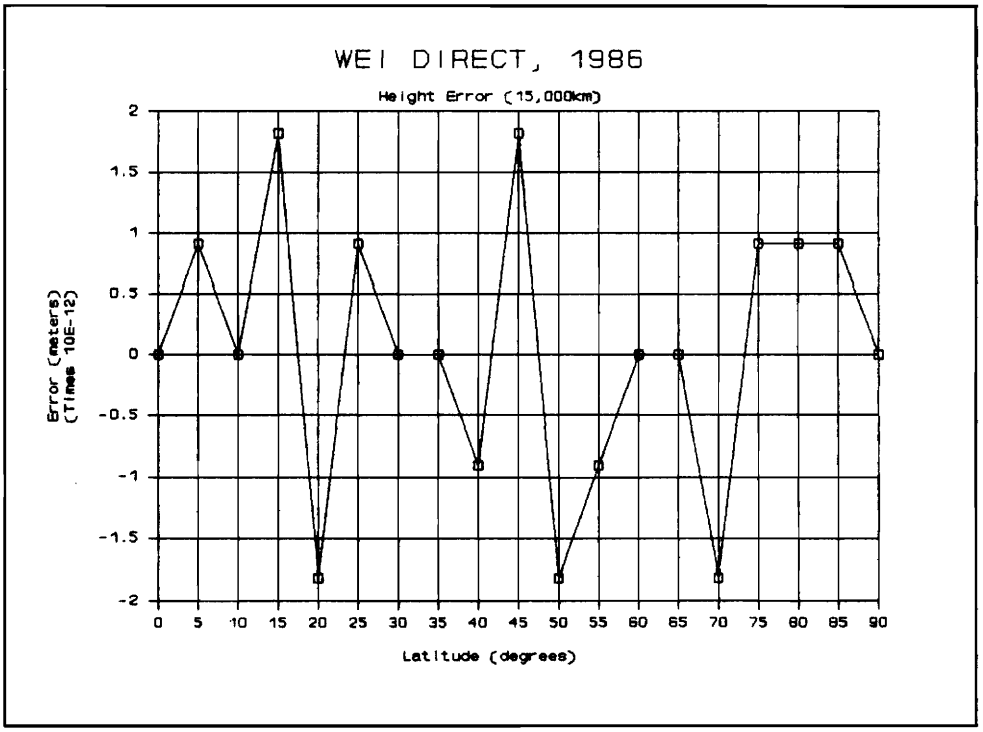

PAUL, 1973
Latitude Error (30,000km)
Error (meters)
(Times 10E-7)
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
-0.8
-0.9
-1
-1.1
-1.2
-1.3
-1.4
-1.5
-1.6
-1.7
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

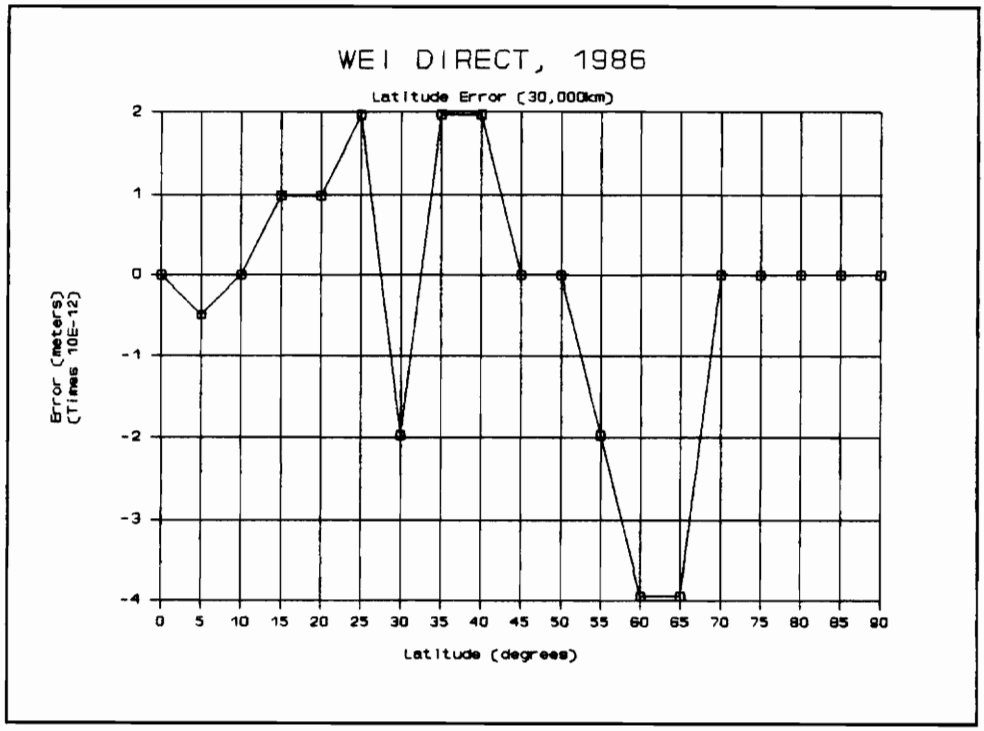

PAUL, 1973
Height Error (30,000km)
Error (meters)
(Times 10E-6)
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
-0.8
-0.9
-1
-1.1
-1.2
-1.3
-1.4
-1.5
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

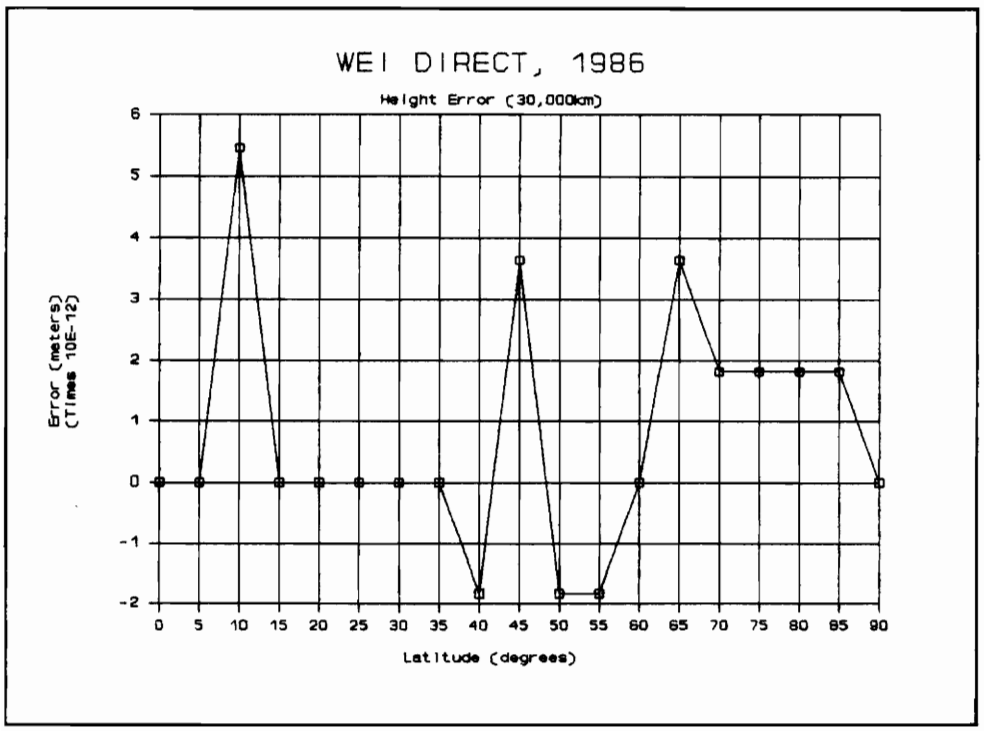

WEI DIRECT, 1986
Latitude Error (0km)
Error (meters)
(Times 10E-13)
0
-1
-2
-3
-4
-5
-6
-7
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

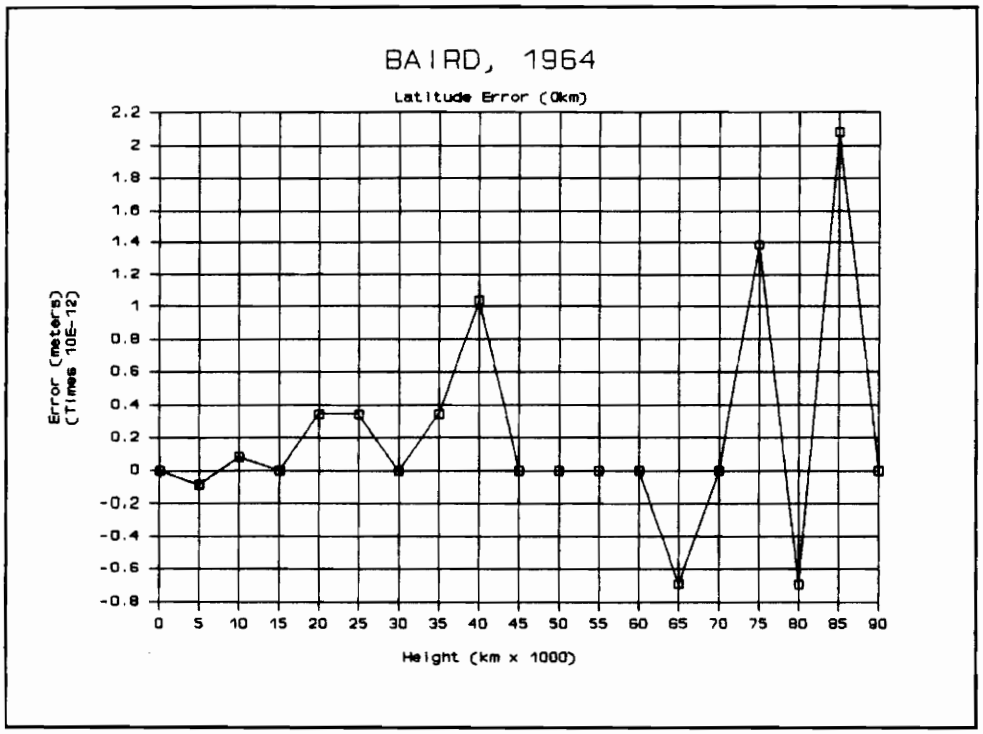

WEI DIRECT, 1986
Height Error (0km)
Error (meters)
(Times 10E-12)
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

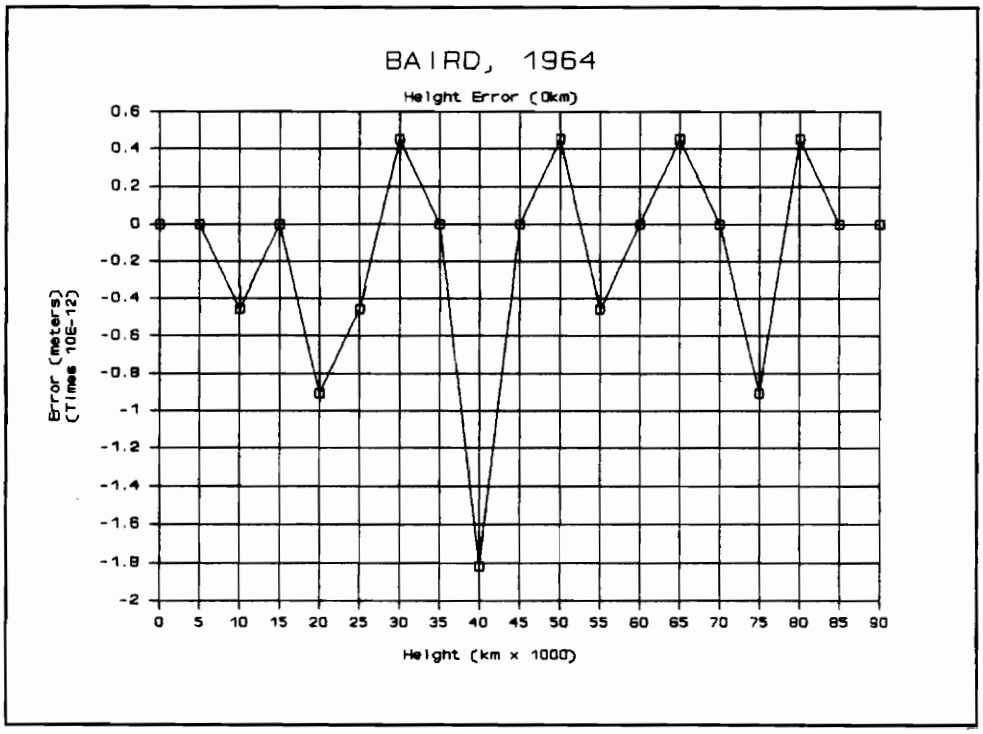

WEI DIRECT, 1986
Latitude Error (15,000km)
Error (meters)
(Times 10E-12)
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

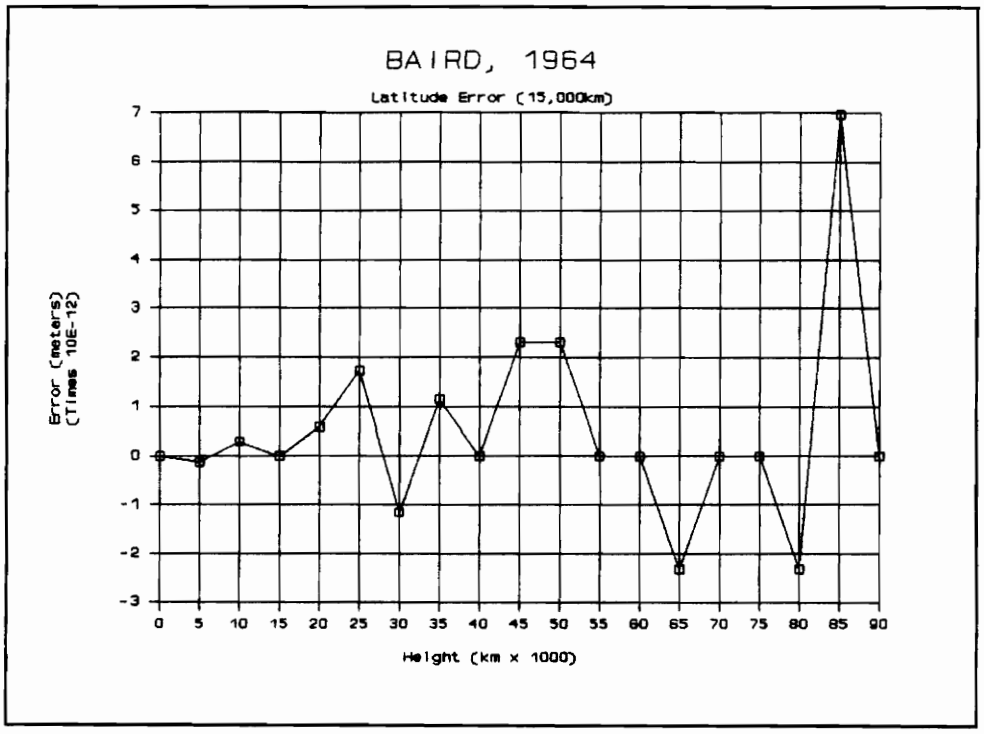

WEI DIRECT, 1986
Height Error (15,000km)
Error (meters)
(Times 10E-12)
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

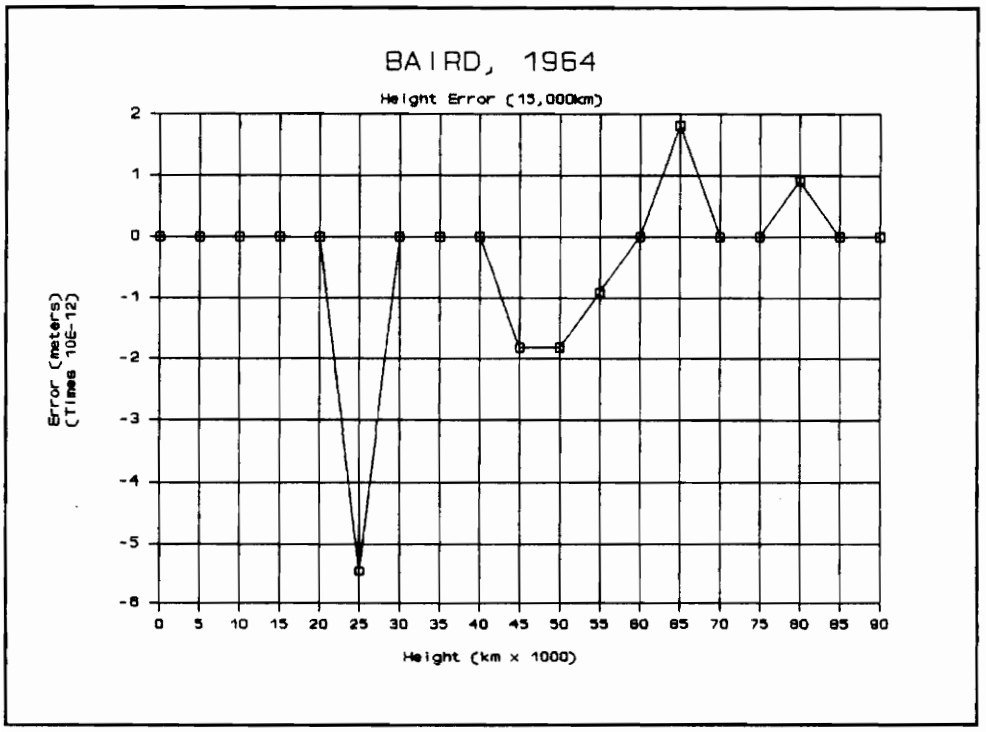

WEI DIRECT, 1986
Latitude Error (30,000km)
Error (meters)
(Times 10E-12)
2
1
0
-1
-2
-3
-4
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

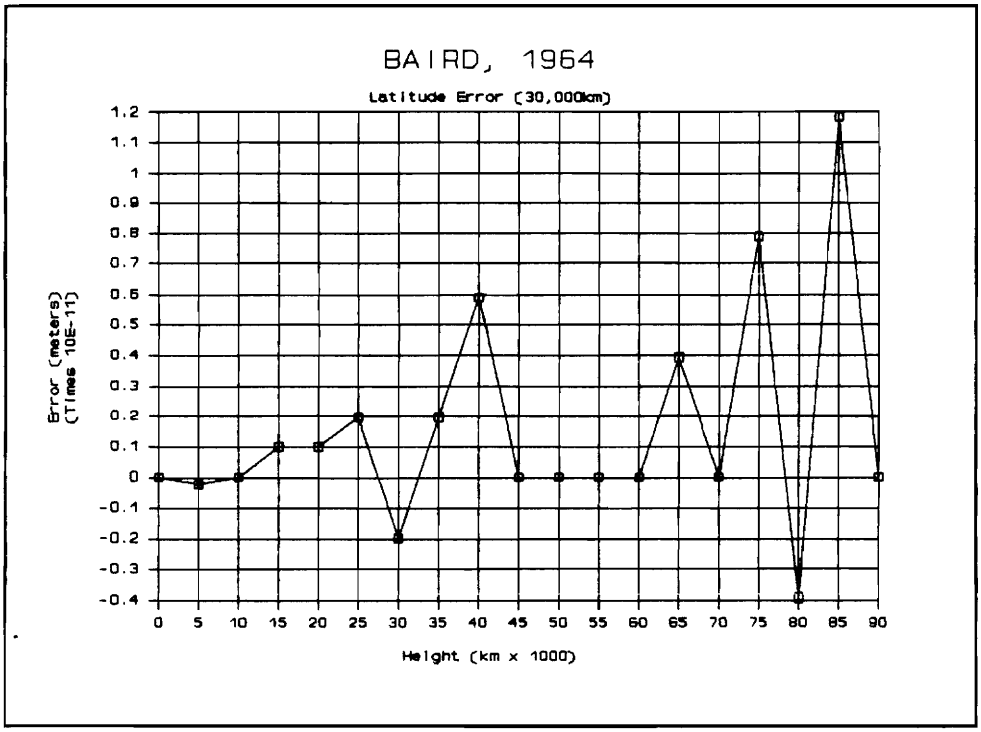

WEI DIRECT, 1986
Height Error (30,000km)
Error (meters)
(Times 10E-12)
6
5
4
3
2
1
0
-1
-2
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

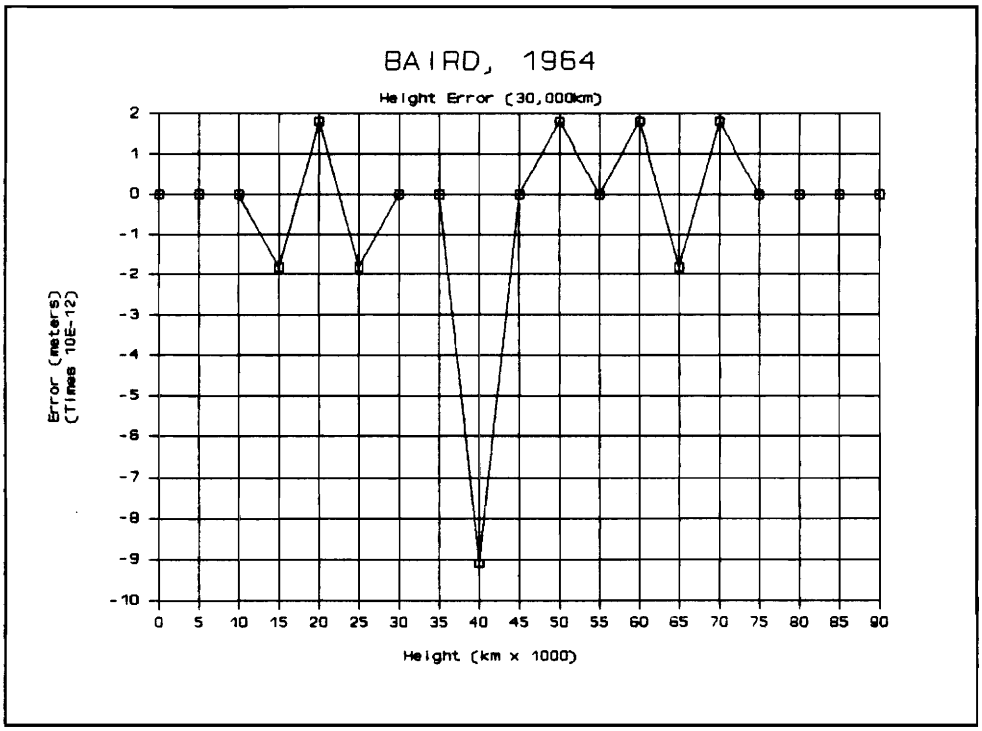

BAIRD, 1964
Latitude Error (0km)
Error (meters)
(Times 10E-12)
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Height (km x 1000)

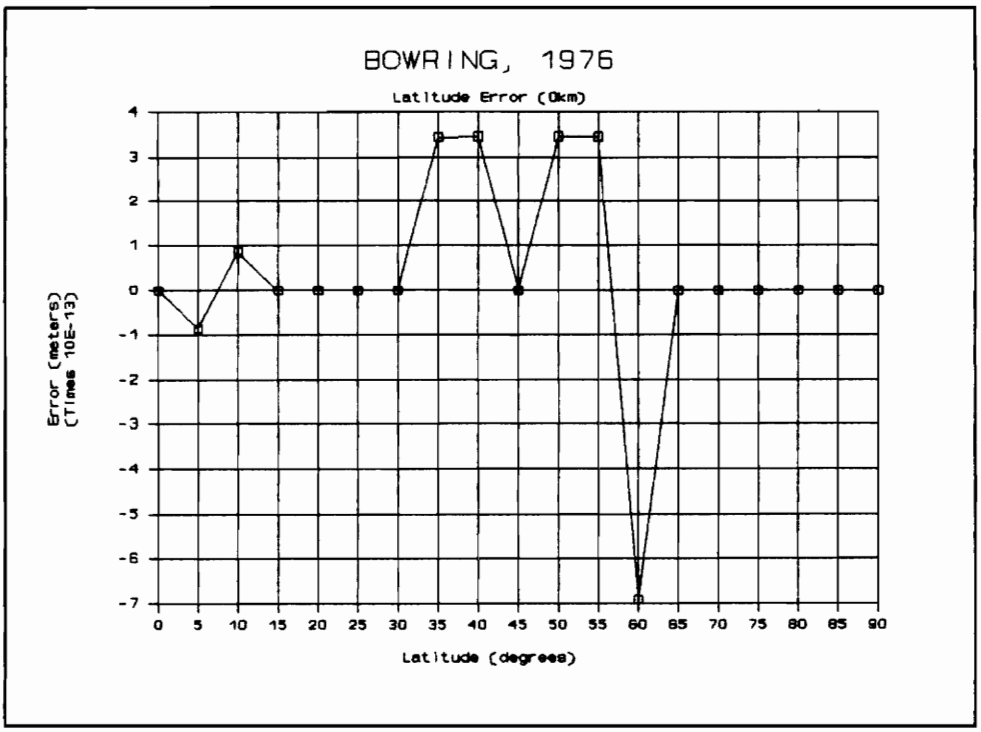

BAIRD, 1964
Height Error (0km)
Error (meters)
(Times 10E-12)
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
-1.4
-1.6
-1.8
-2
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Height (km x 1000)

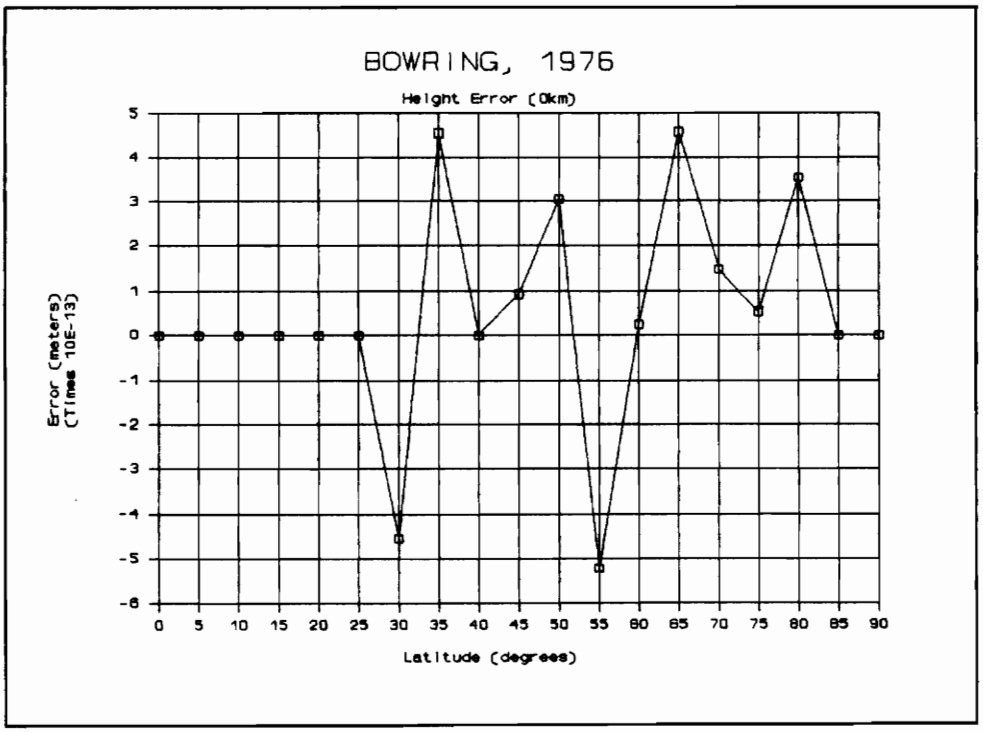

BAIRD, 1964
Latitude Error (15,000km)
Error (meters)
(Times 10E-12)
7
6
5
4
3
2
1
0
-1
-2
-3
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Height (km x 1000)

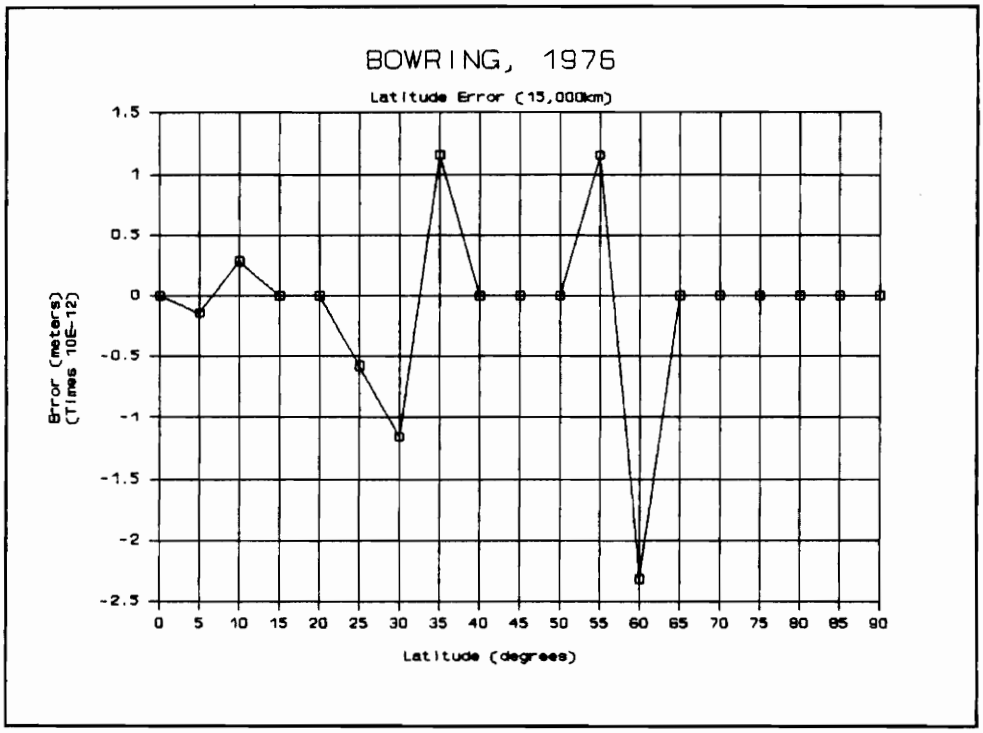

BAIRD, 1964
Height Error (15,000km)
Error (meters)
(Times 10E-12)
2
1
0
-1
-2
-3
-4
-5
-6
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Height (km x 1000)

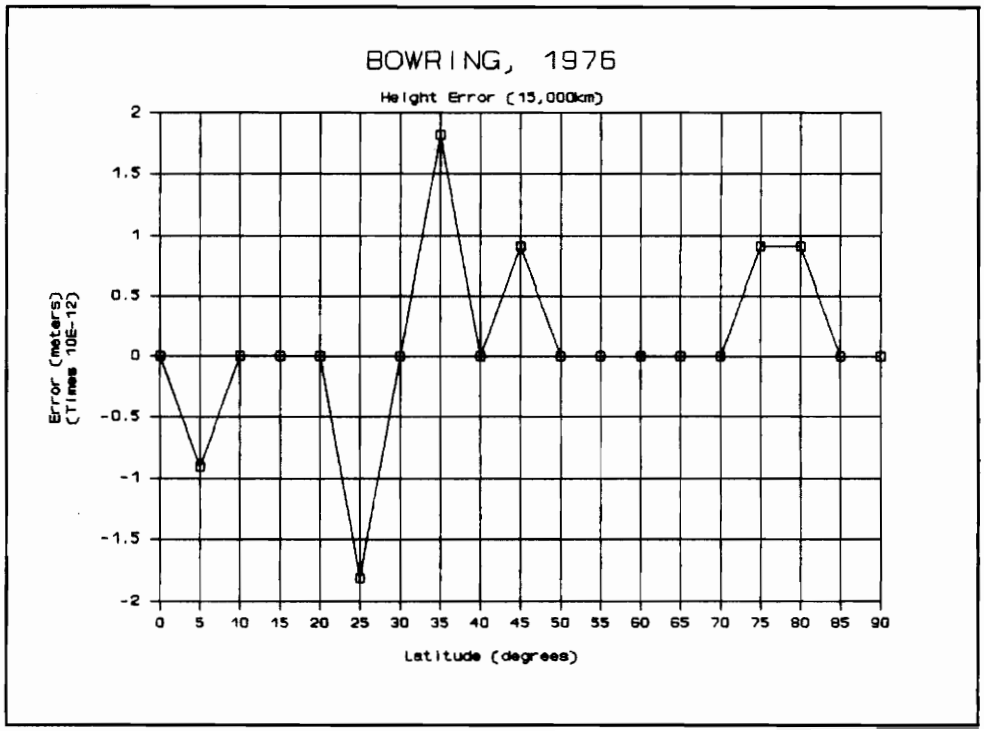

BAIRD, 1964
Latitude Error (30,000km)
Error (meters)
(Times 10E-11)
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Height (km x 1000)

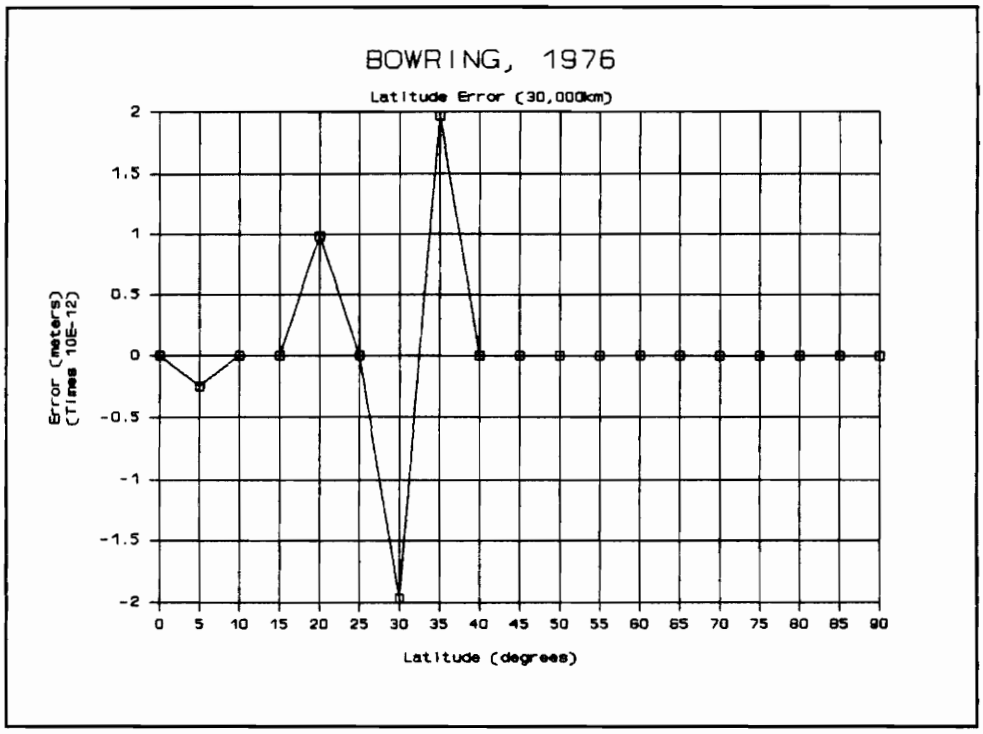

BAIRD, 1964
Height Error (30,000km)
Error (meters)
(Times 10E-12)
2
1
0
-1
-2
-3
-4
-5
-6
-7
-8
-9
-10
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Height (km x 1000)

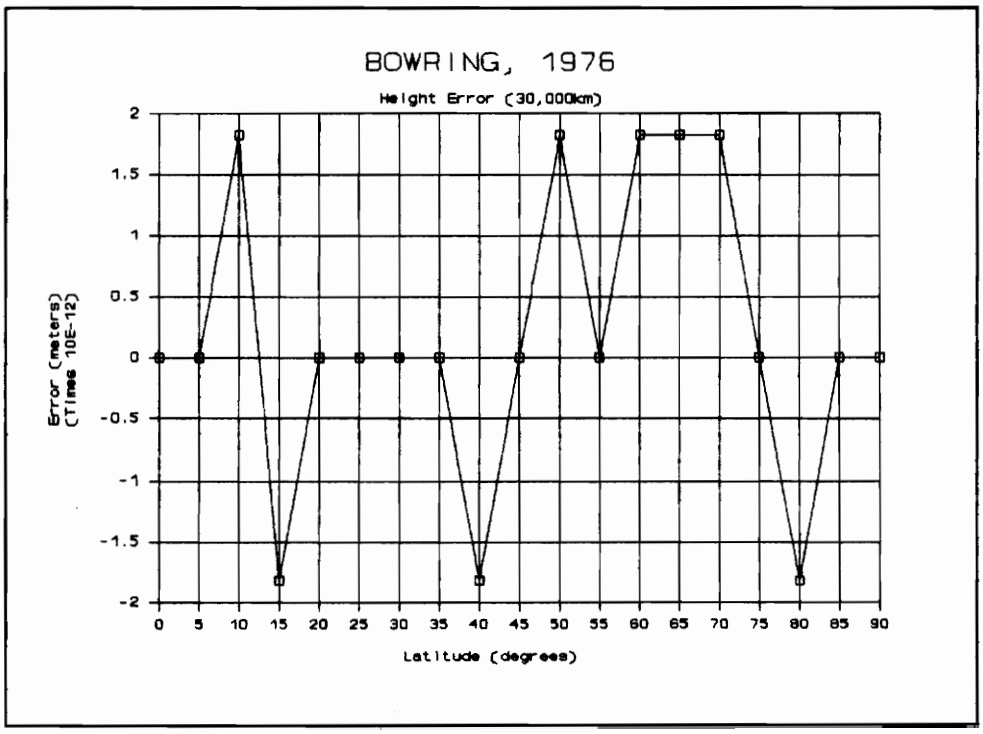

BOWRING, 1976
Latitude Error (0km)
Error (meters)
(Times 10E-13)
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-7
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

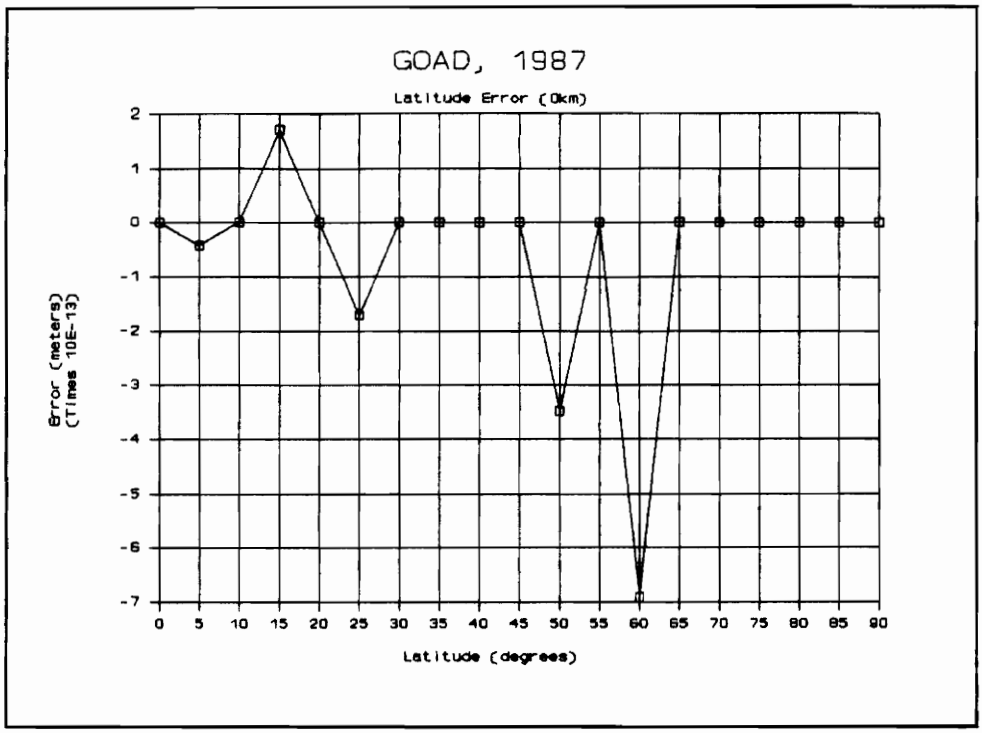

BOWRING, 1976
Height Error (0km)
Error (meters)
(Times 10E-13)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

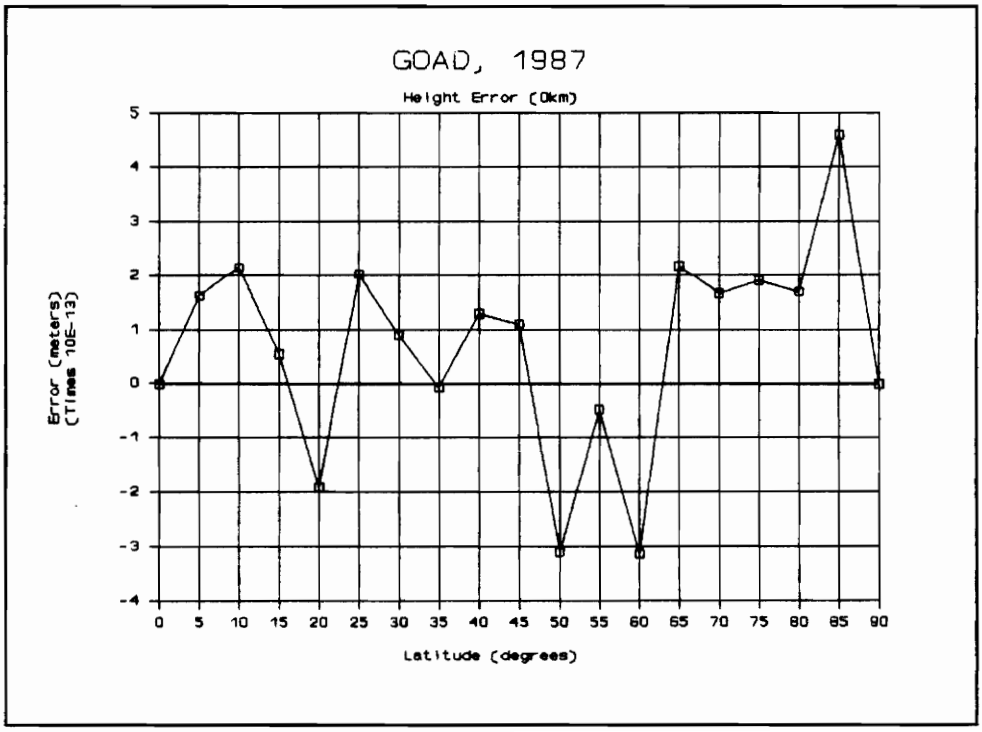

BOWRING, 1976
Latitude Error (15,000km)
Error (meters)
(Times 10E-12)
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

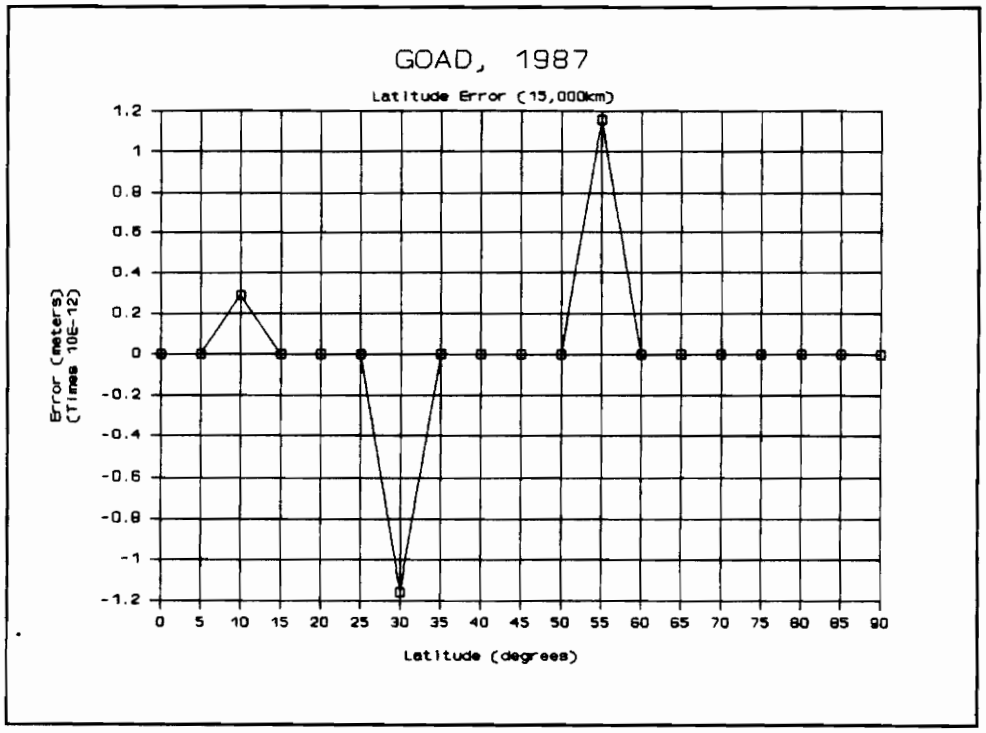

BOWRING, 1976
Height Error (15,000km)
Error (meters)
(Times 10E-12)
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

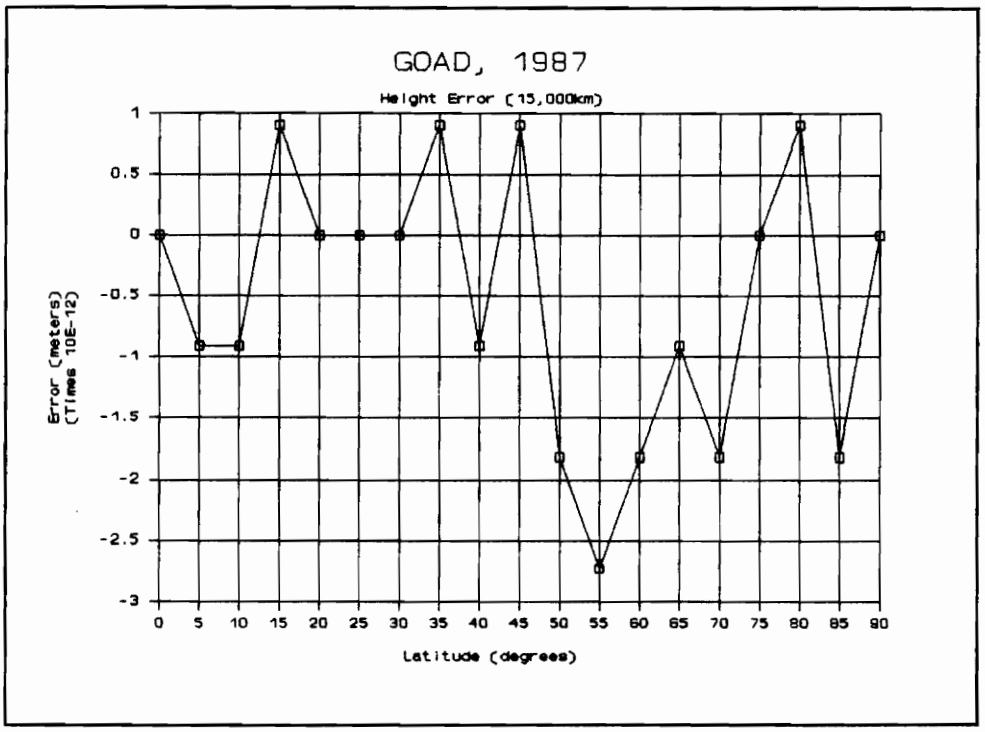

BOWRING, 1976
Latitude Error (30,000km)
Error (meters)
(Times 10E-12)
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

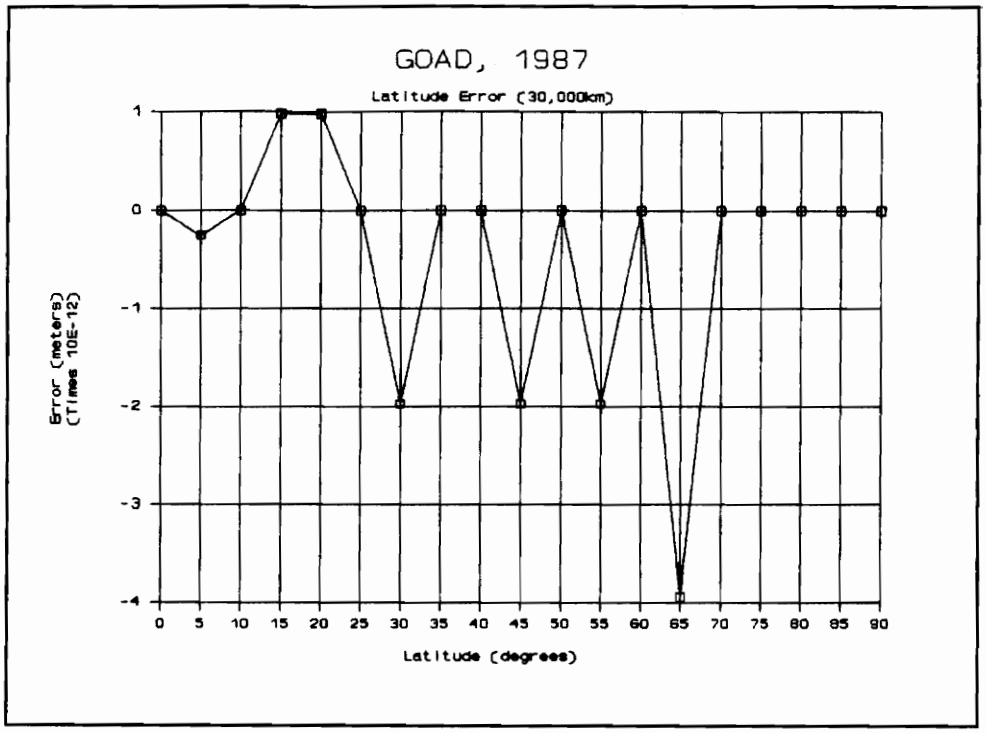

BOWRING, 1976
Height Error (30,000km)
Error (meters)
(Times 10E-12)
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

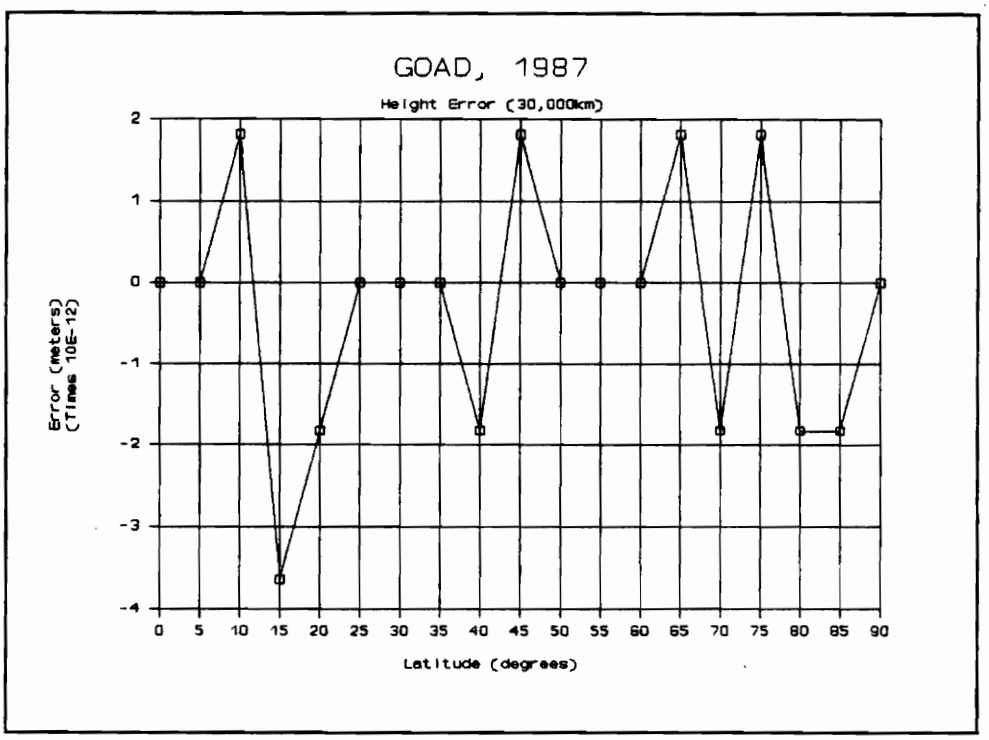

GOAD, 1987
Latitude Error (0km)
Error (meters)
(Times 10E-13)
2
1
0
-1
-2
-3
-4
-5
-6
-7
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

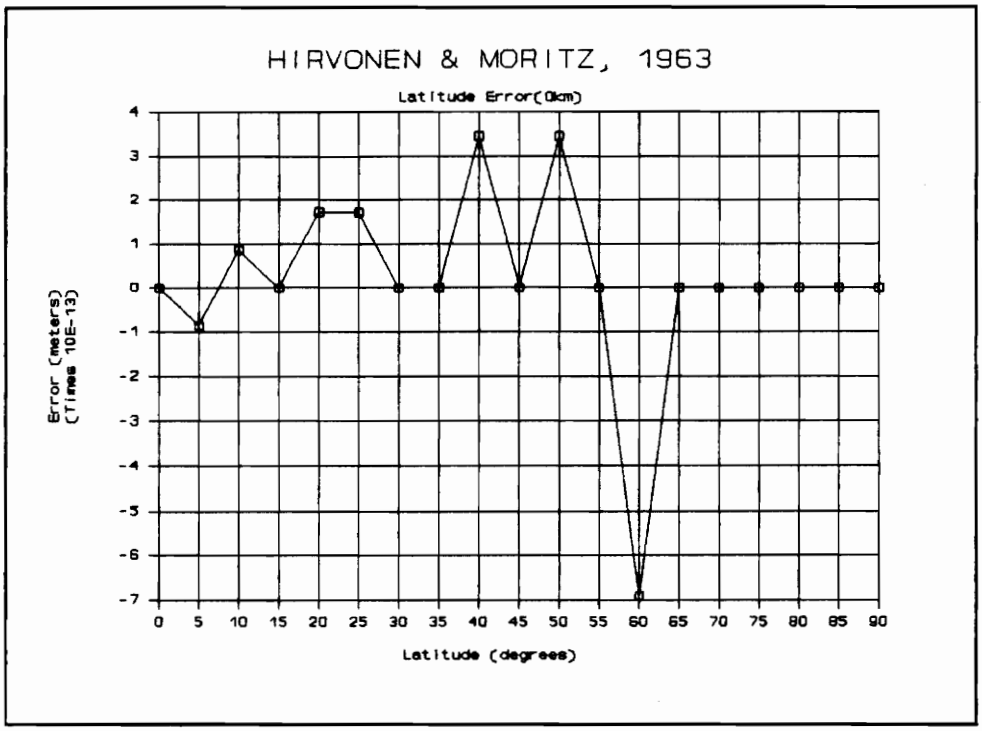

GOAD, 1987
Height Error (0km)
Error (meters)
(Times 10E-13)
5
4
3
2
1
0
-1
-2
-3
-4
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

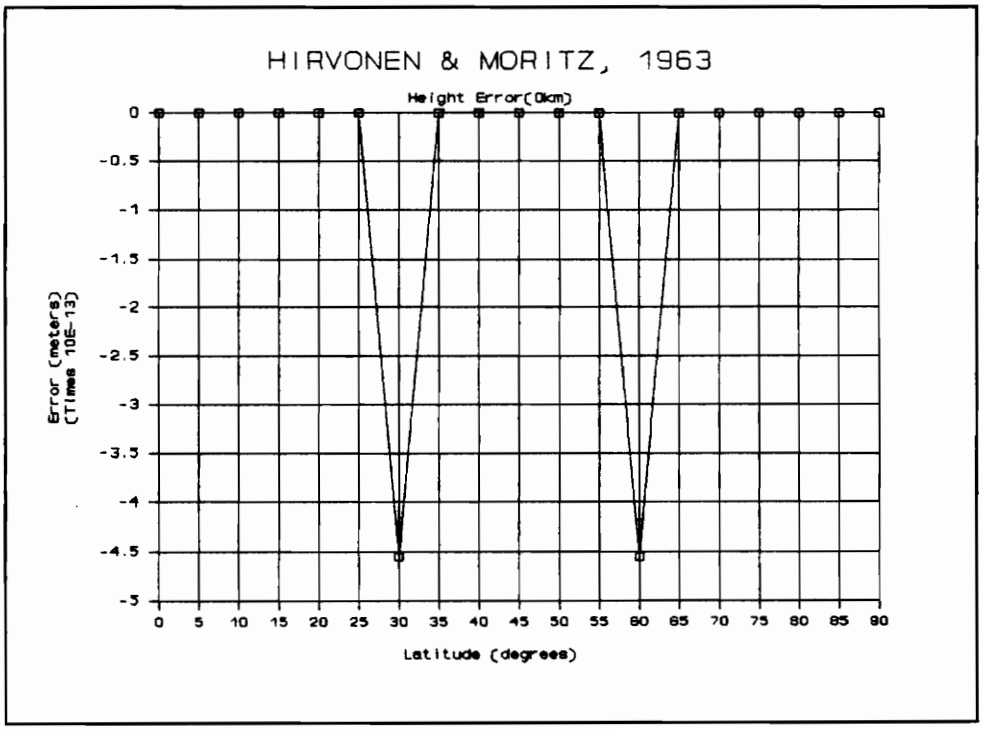

GOAD, 1987
Latitude Error (15,000km)
Error (meters)
(Times 10E-12)
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

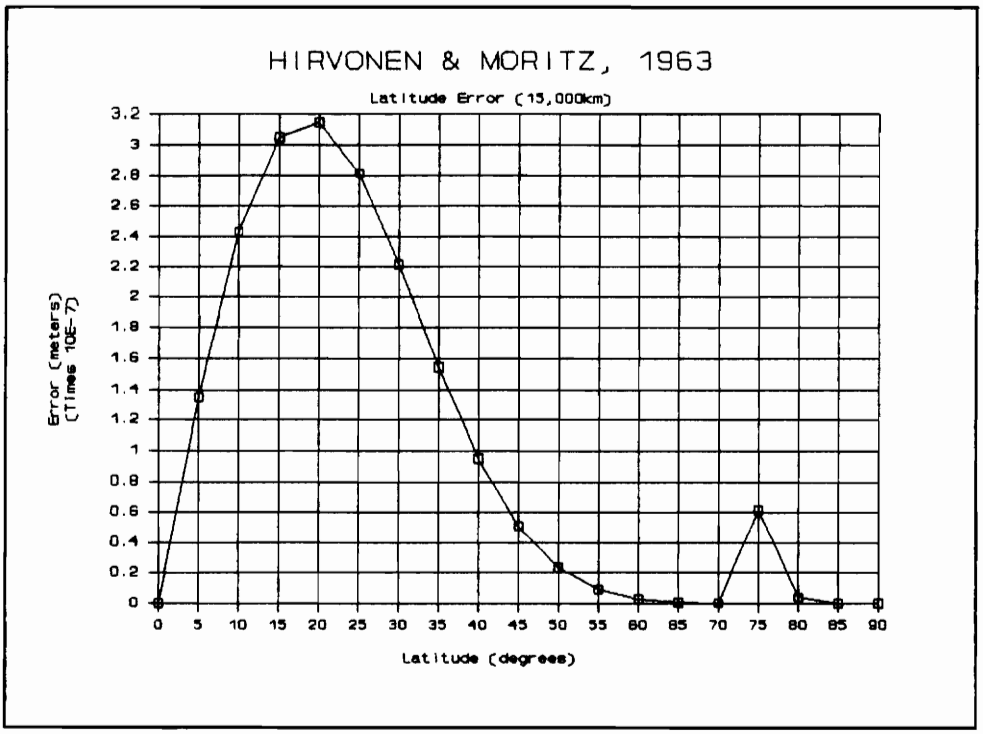

GOAD, 1987
Height Error (15,000km)
Error (meters)
(Times 10E-12)
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

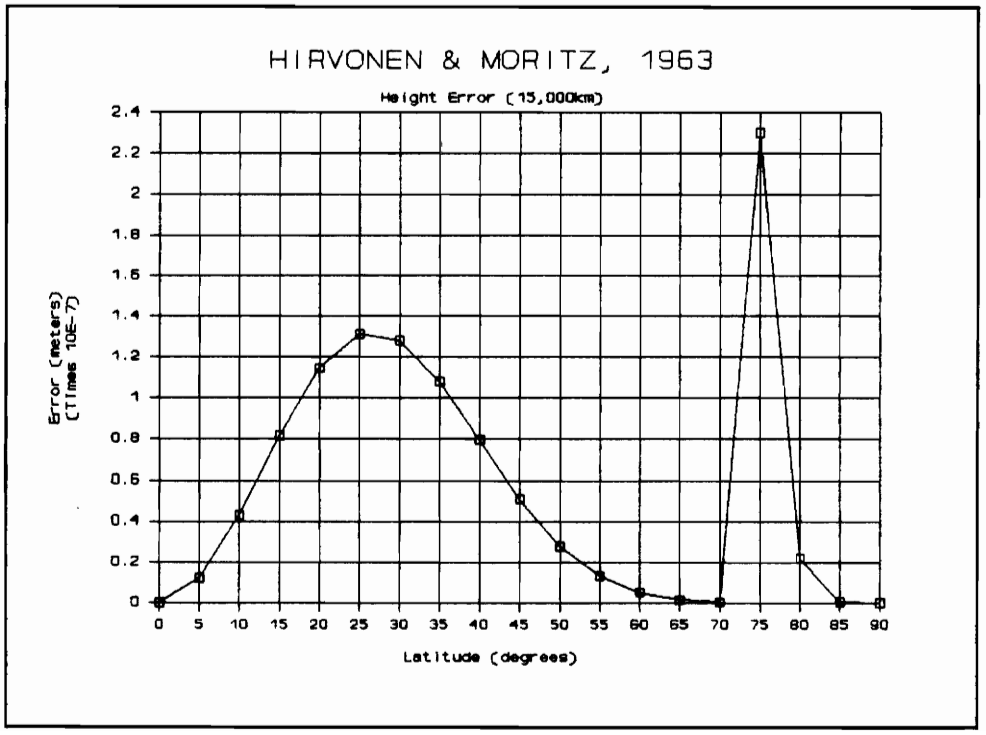

GOAD, 1987
Latitude Error (30,000km)
Error (meters)
(Times 10E-12)
1
0
-1
-2
-3
-4
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

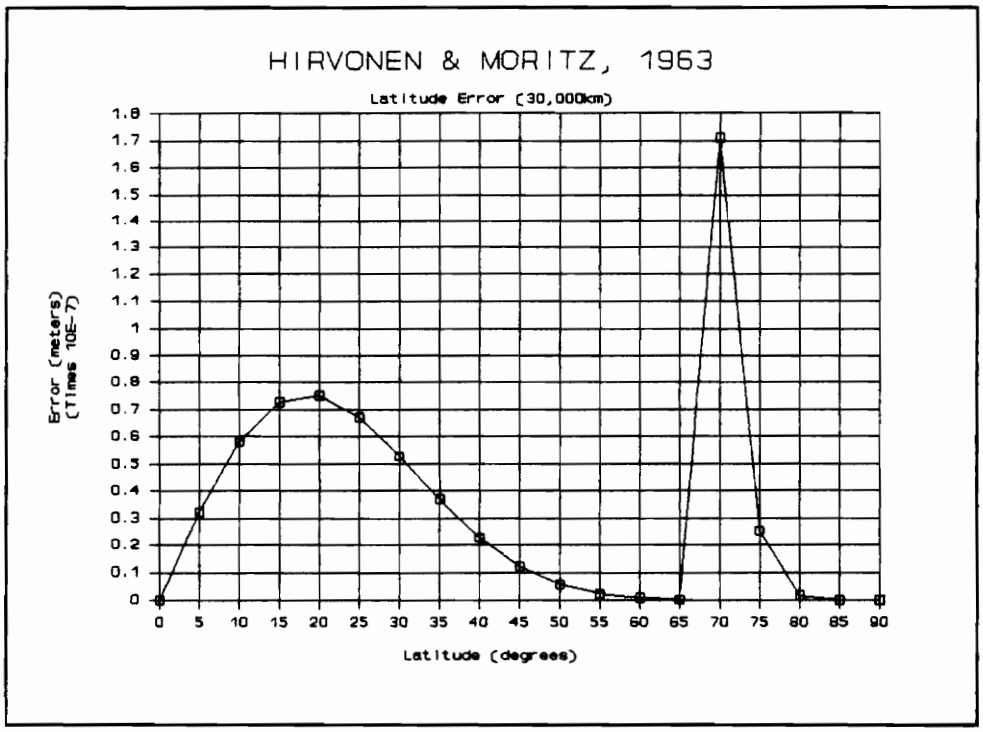

GOAD, 1987
Height Error (30,000km)
Error (meters)
(Times 10E-12)
2
1
0
-1
-2
-3
-4
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

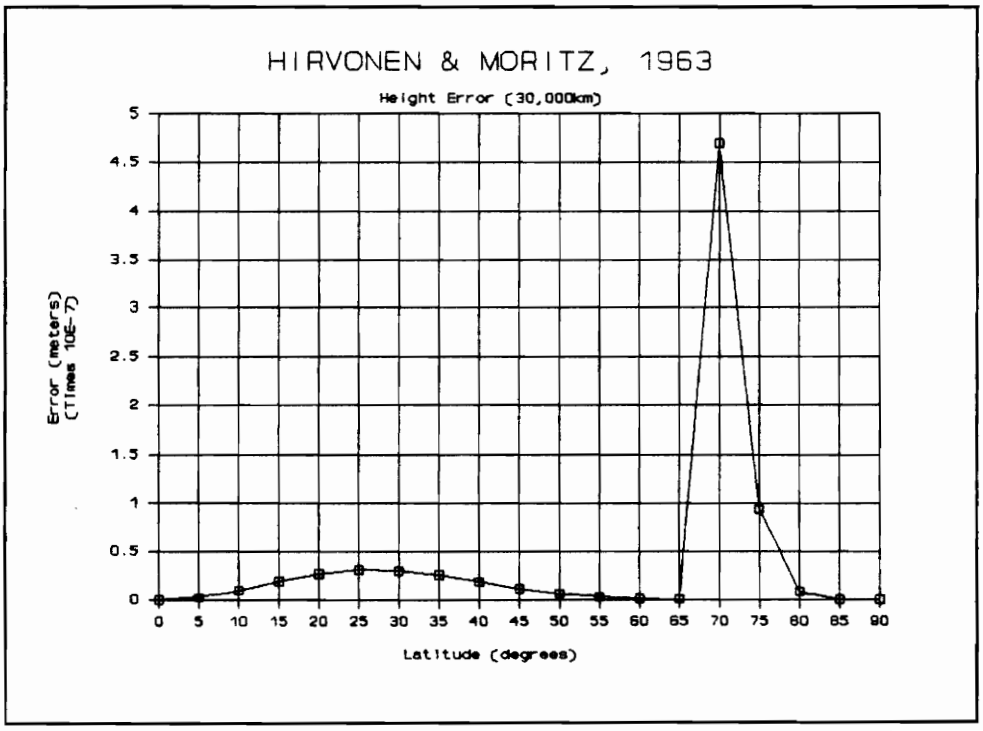

HIRVONEN & MORITZ, 1963
Latitude Error(0km)
Error (meters)
(Times 10E-13)
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-7
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

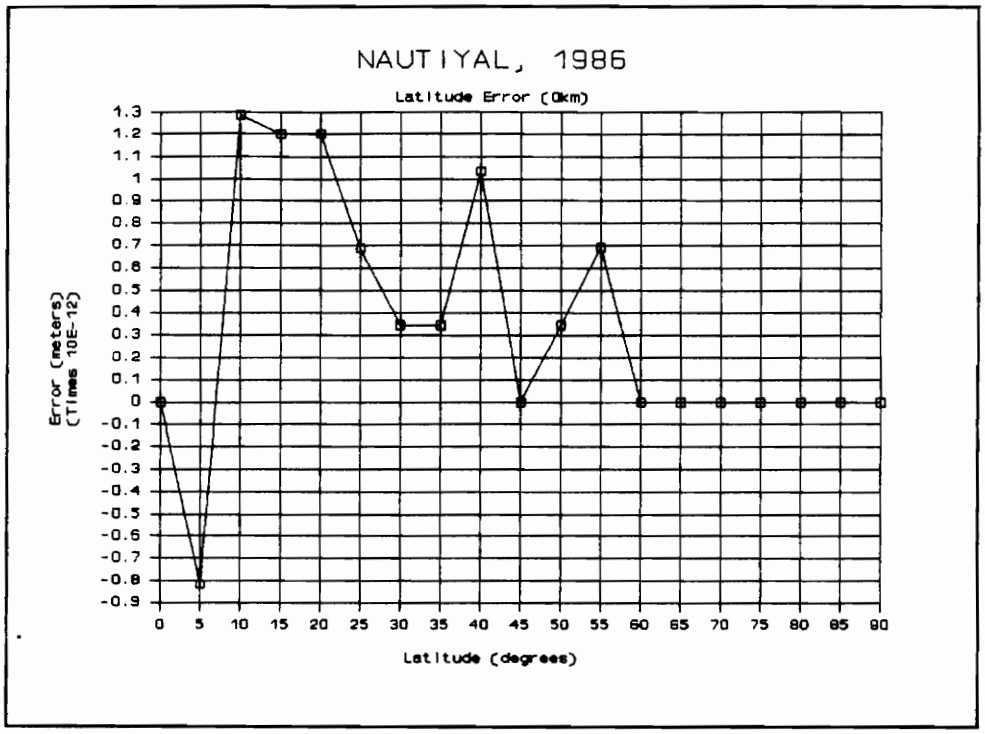

HIRVONEN & MORITZ, 1963
Height Error(0km)
Error (meters)
(Times 10E-13)
0
-0.5
-1
-1.5
-2
-2.5
-3
-3.5
-4
-4.5
-5
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

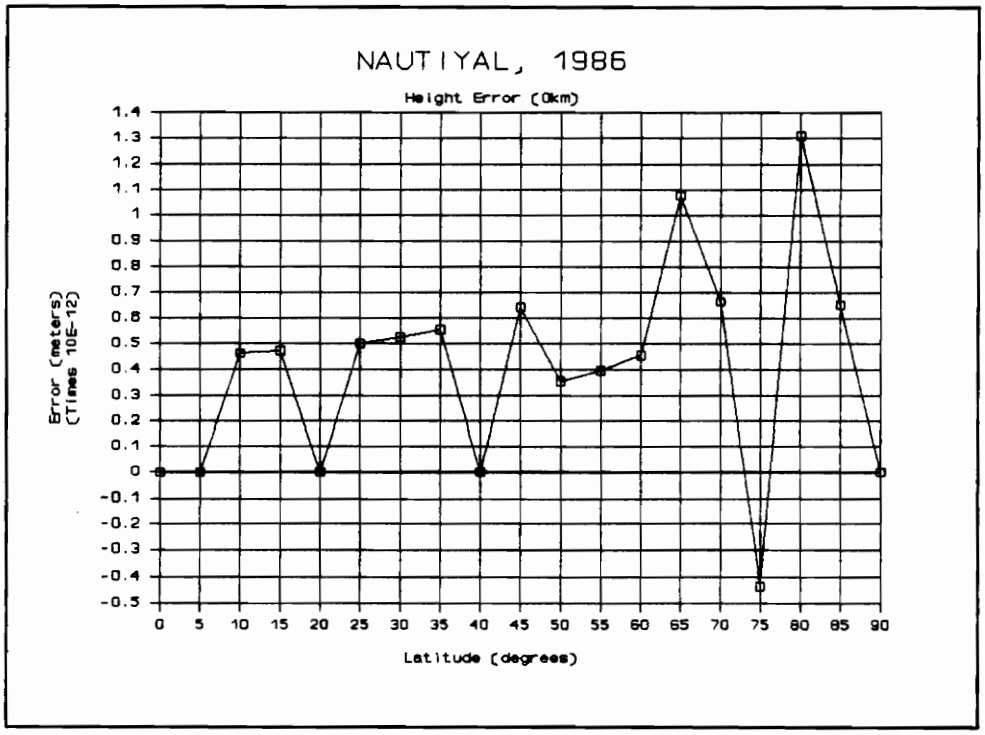

HIRVONEN & MORITZ, 1963
Latitude Error (15,000km)
Error (meters)
(Times 10E-7)
3.2
3
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

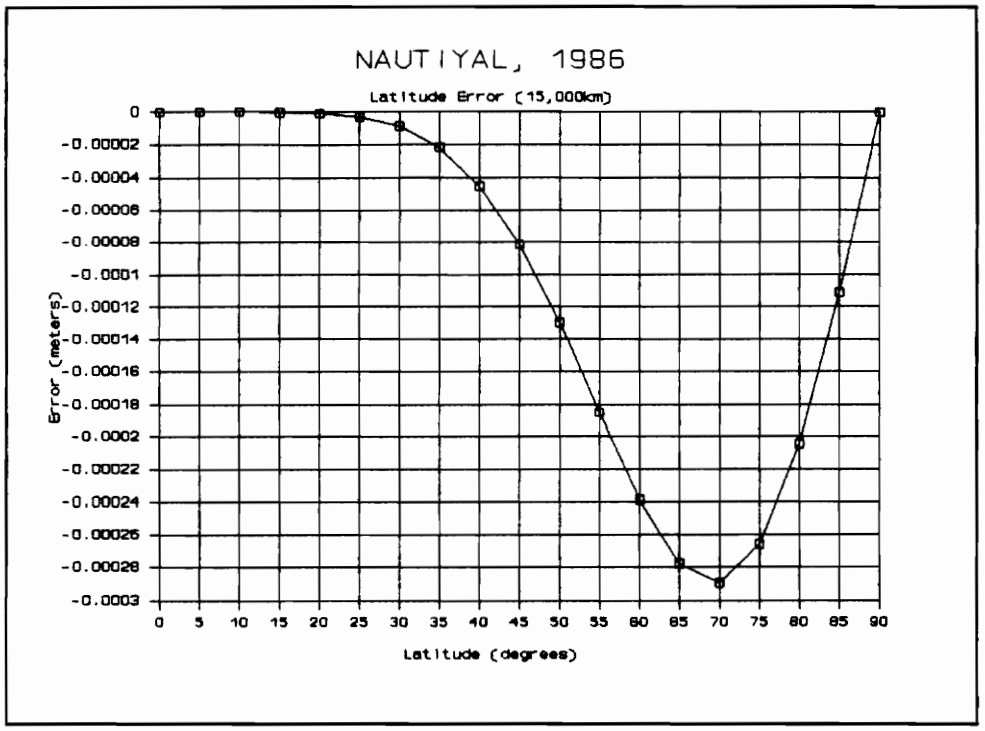

HIRVONEN & MORITZ, 1963
Height Error (15,000km)
Error (meters)
(Times 10E-7)
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

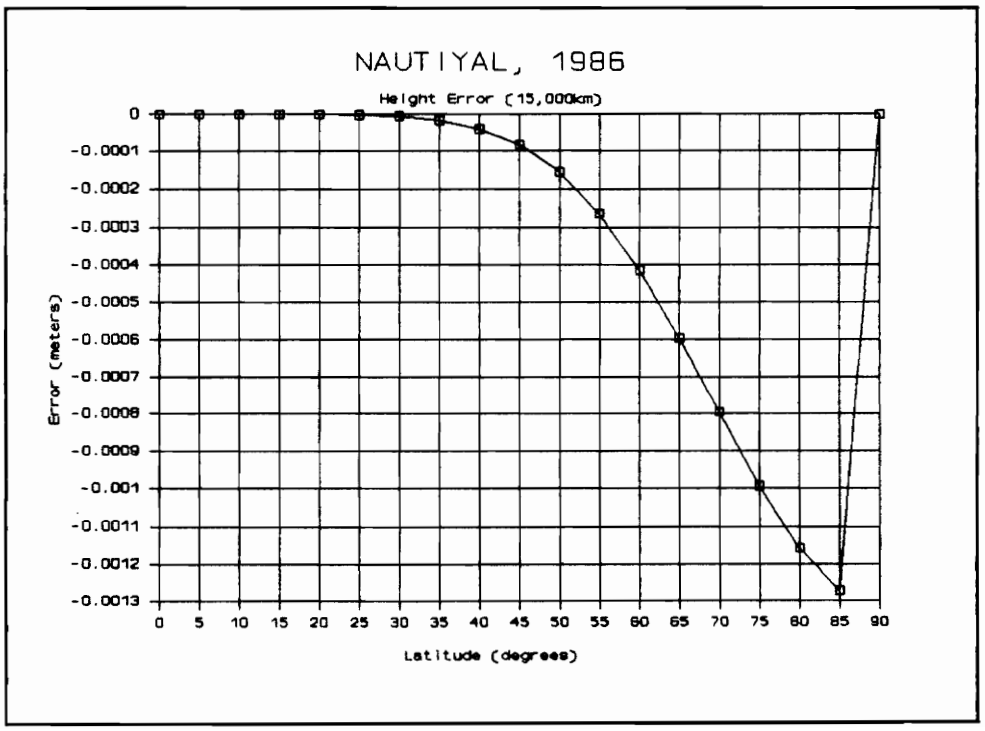

HIRVONEN & MORITZ, 1963
Latitude Error (30,000km)
Error (meters)
(Times 10E-7)
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

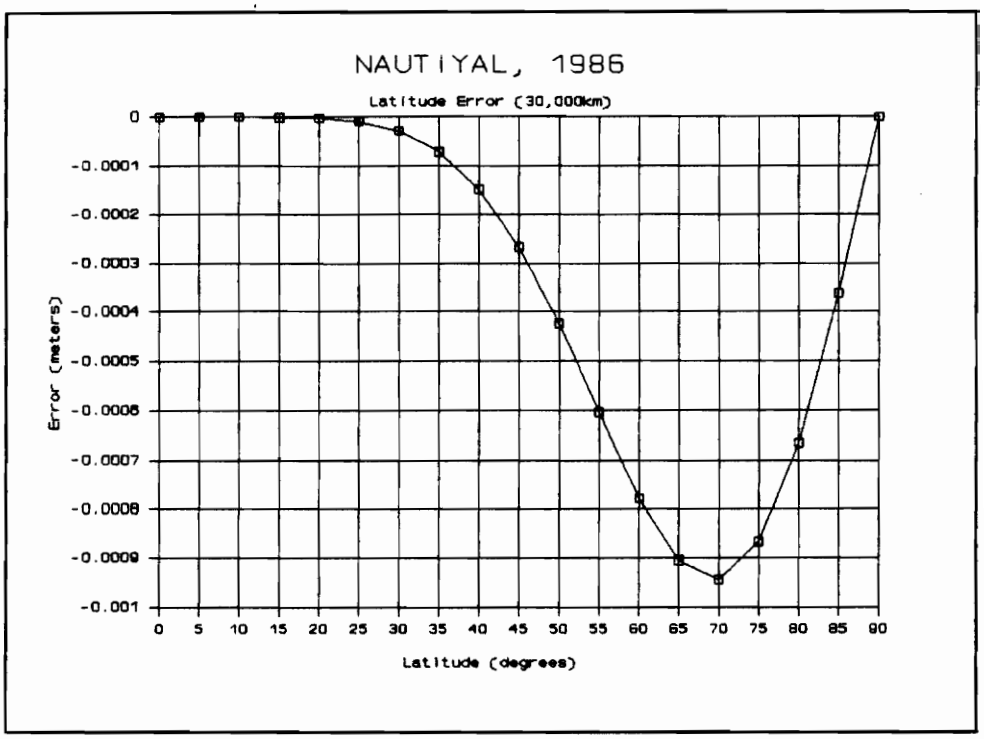

HIRVONEN & MORITZ, 1963
Height Error (30,000km)
Error (meters)
(Times 10E-7)
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

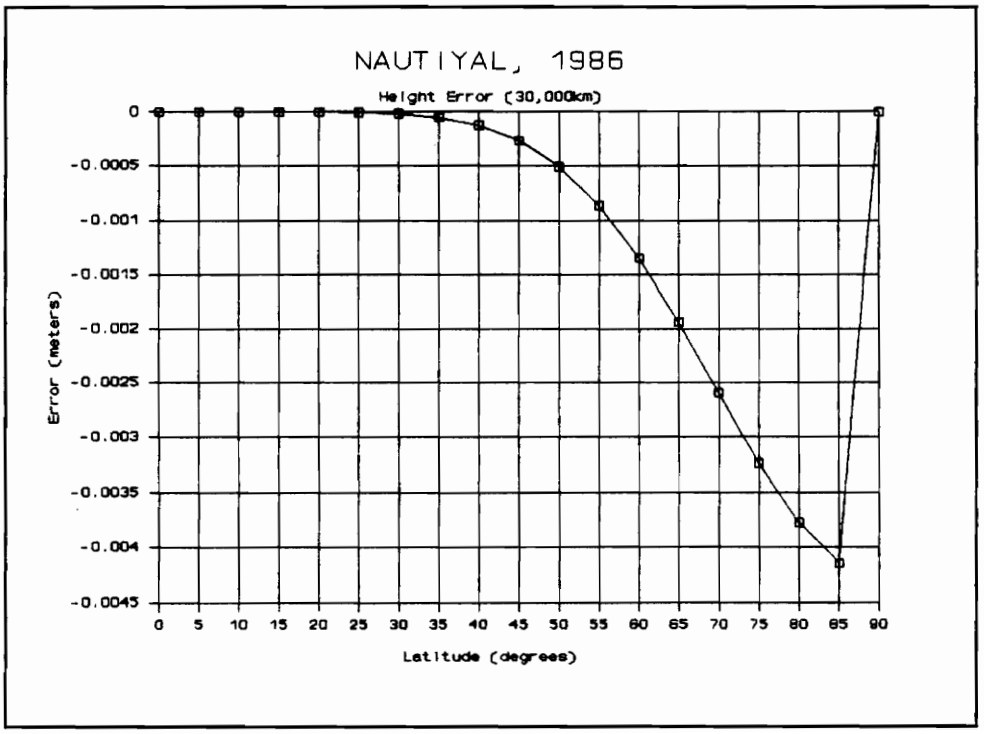

NAUTIYAL, 1986
Latitude Error (0km)
Error (meters)
(Times 10E-12)
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
-0.8
-0.9
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

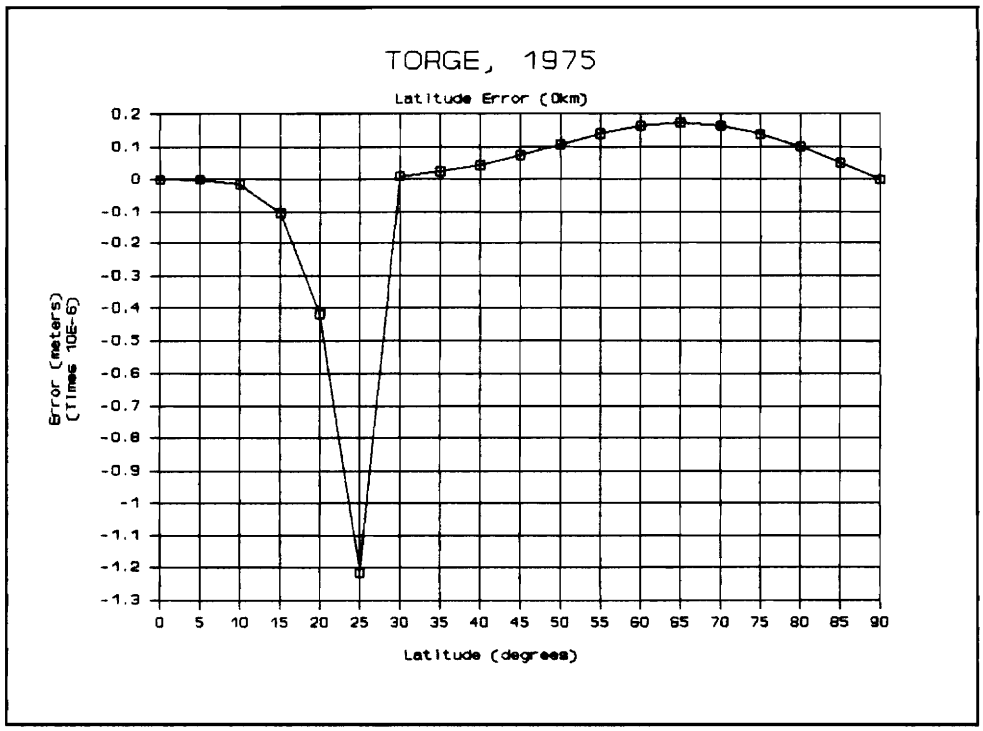

NAUTIYAL, 1986
Height Error (0km)
Error (meters)
(Times 10E-12)
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

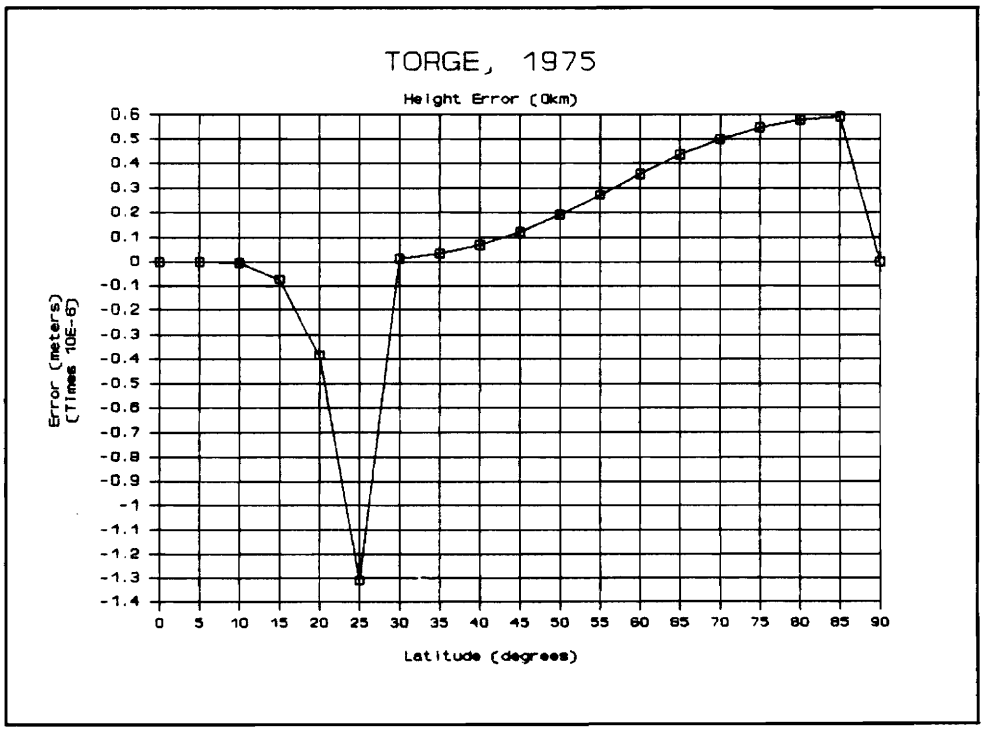

NAUTIYAL, 1986
Latitude Error (15,000km)
Error (meters)
0
-0.00002
-0.00004
-0.00006
-0.00008
-0.0001
-0.00012
-0.00014
-0.00016
-0.00018
-0.0002
-0.00022
-0.00024
-0.00026
-0.00028
-0.0003
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

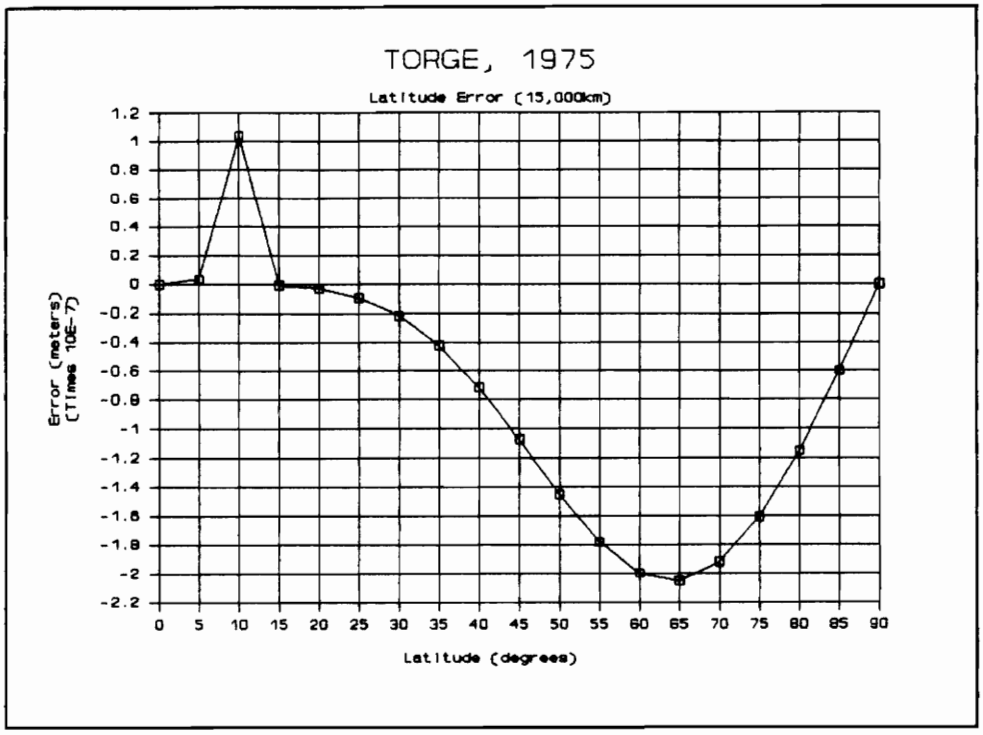

NAUTIYAL, 1986
Height Error (15,000km)
Error (meters)
0
-0.0001
-0.0002
-0.0003
-0.0004
-0.0005
-0.0006
-0.0007
-0.0008
-0.0009
-0.001
-0.0011
-0.0012
-0.0013
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

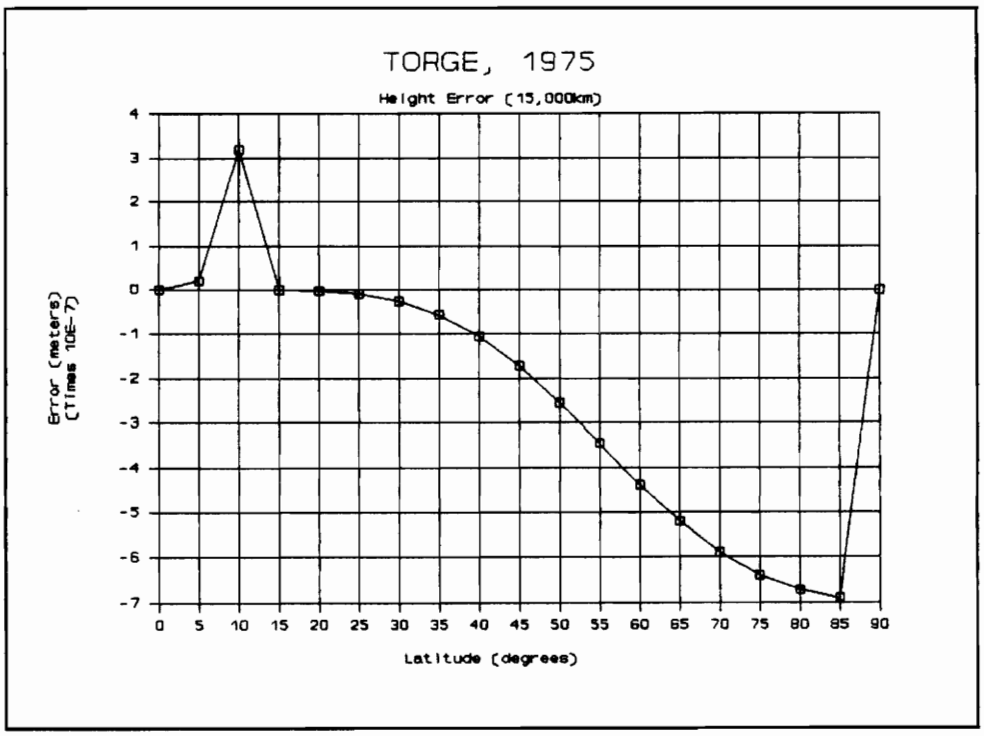

NAUTIYAL, 1986
Latitude Error (30,000km)
Error (meters)
0
-0.0001
-0.0002
-0.0003
-0.0004
-0.0005
-0.0006
-0.0007
-0.0008
-0.0009
-0.001
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

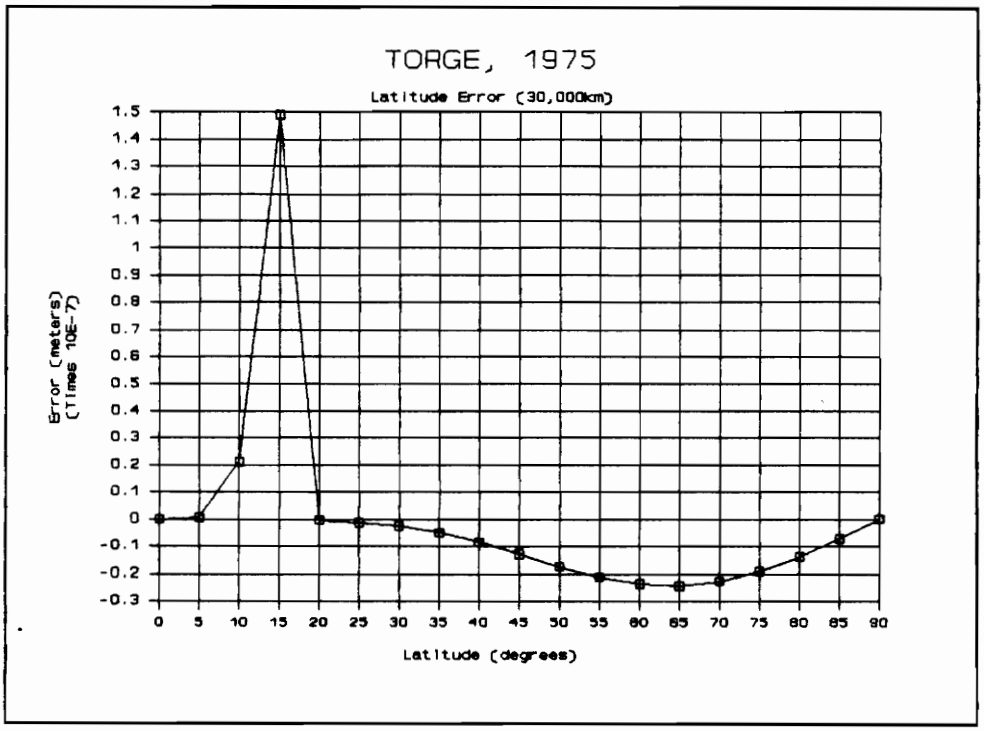

NAUTIYAL, 1986
Height Error (30,000km)
Error (meters)
0
-0.0005
-0.001
-0.0015
-0.002
-0.0025
-0.003
-0.0035
-0.004
-0.0045
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

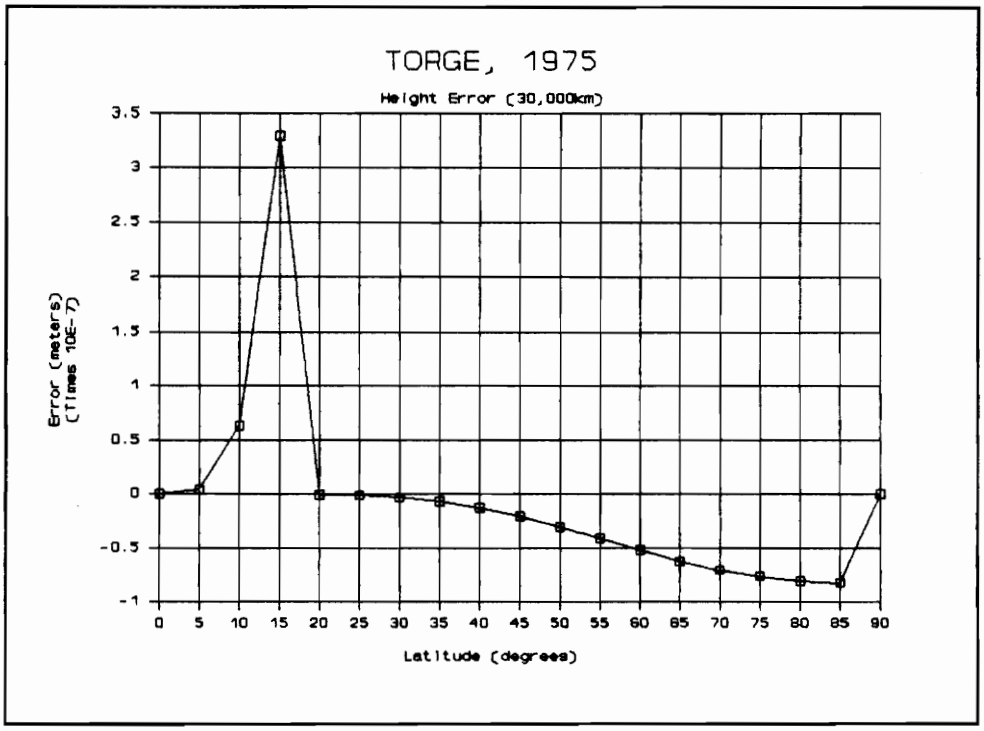

TORGE, 1975
Latitude Error (0km)
Error (meters)
(Times 10E-6)
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
-0.8
-0.9
-1
-1.1
-1.2
-1.3
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

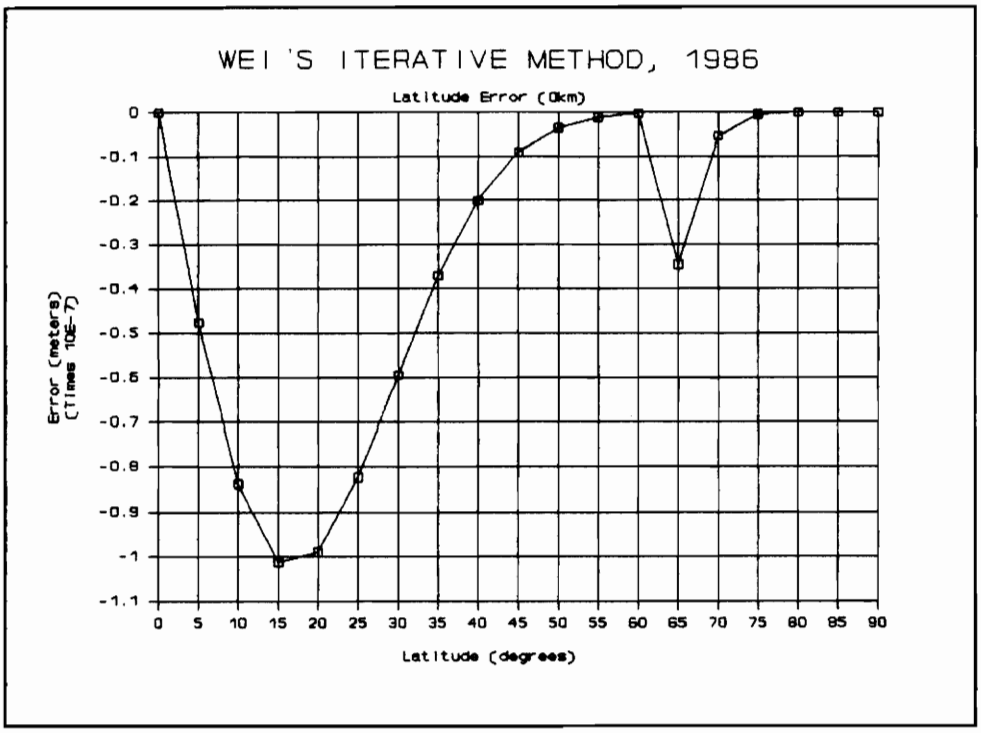

TORGE, 1975
Height Error (0km)
Error (meters)
(Times 10E-6)
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
-0.8
-0.9
-1
-1.1
-1.2
-1.3
-1.4
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

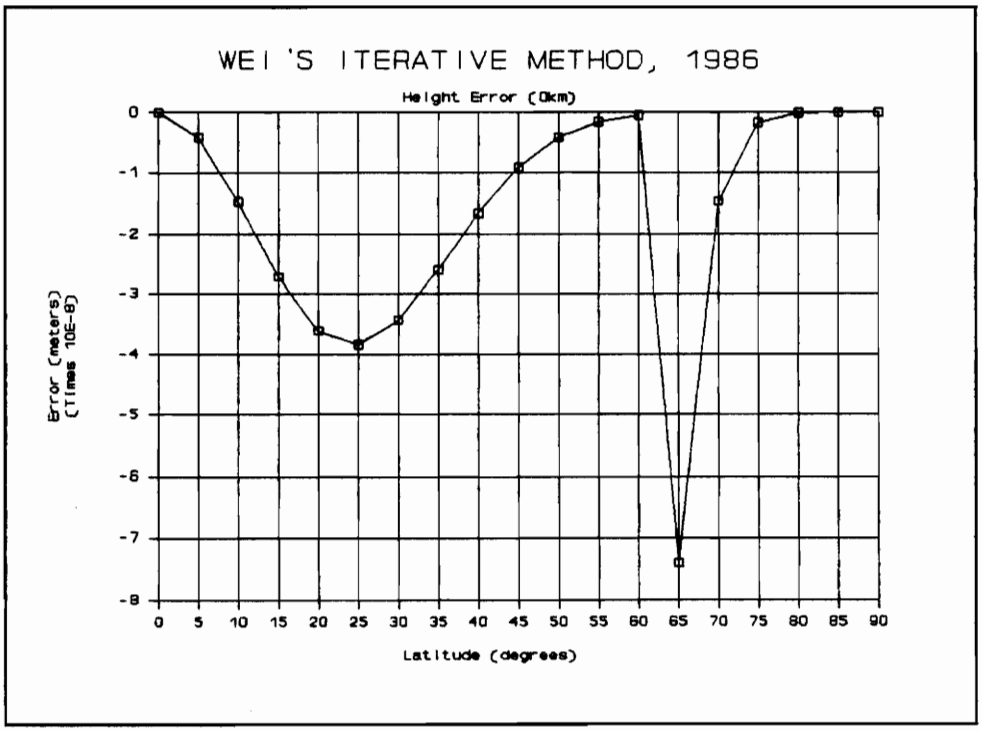

TORGE, 1975
Latitude Error (15,000km)
Error (meters)
(Times 10E-7)
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
-1.4
-1.6
-1.8
-2
-2.2
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

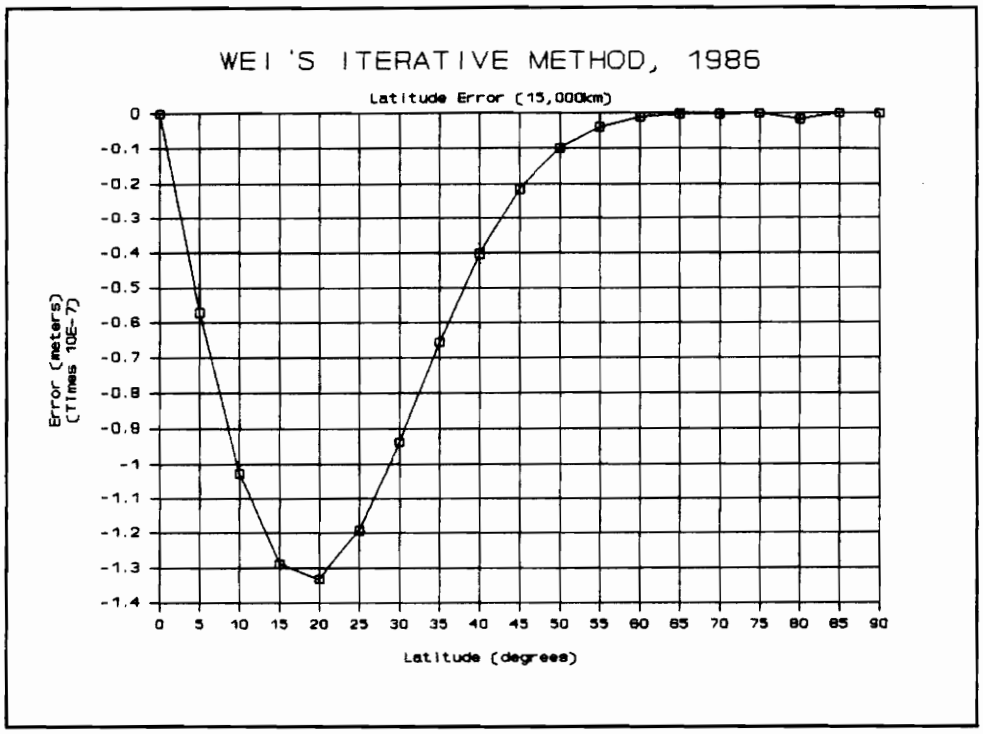

TORGE, 1975
Height Error (15,000km)
Error (meters)
(Times 10E-7)
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-7
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

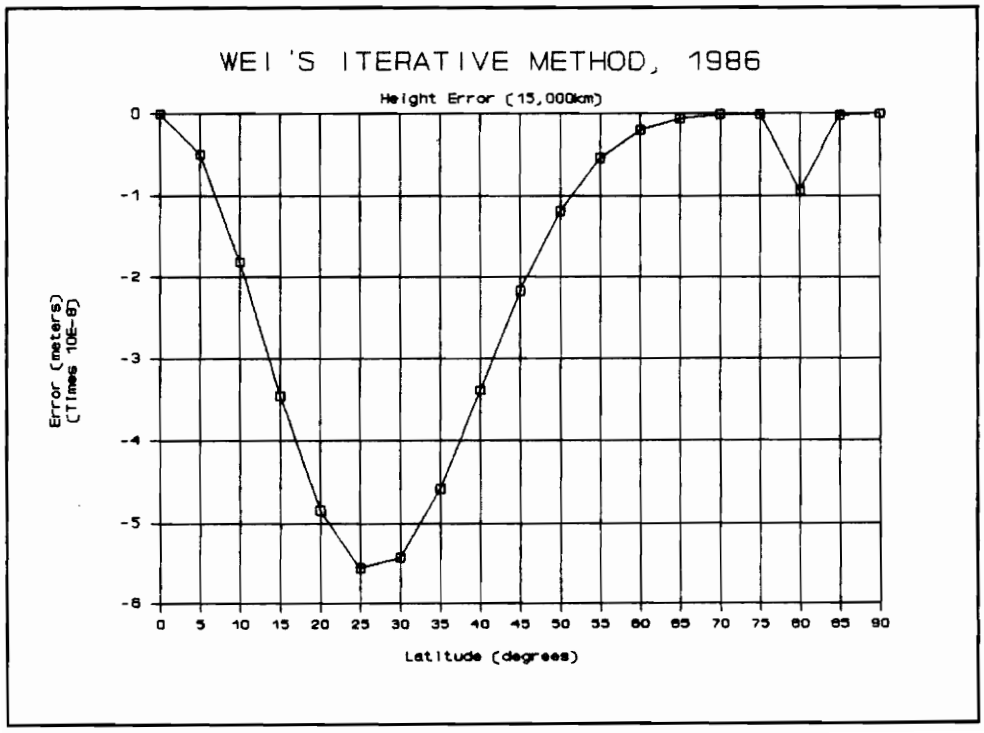

TORGE, 1975
Latitude Error (30,000km)
Error (meters)
(Times 10E-7)
1.5
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

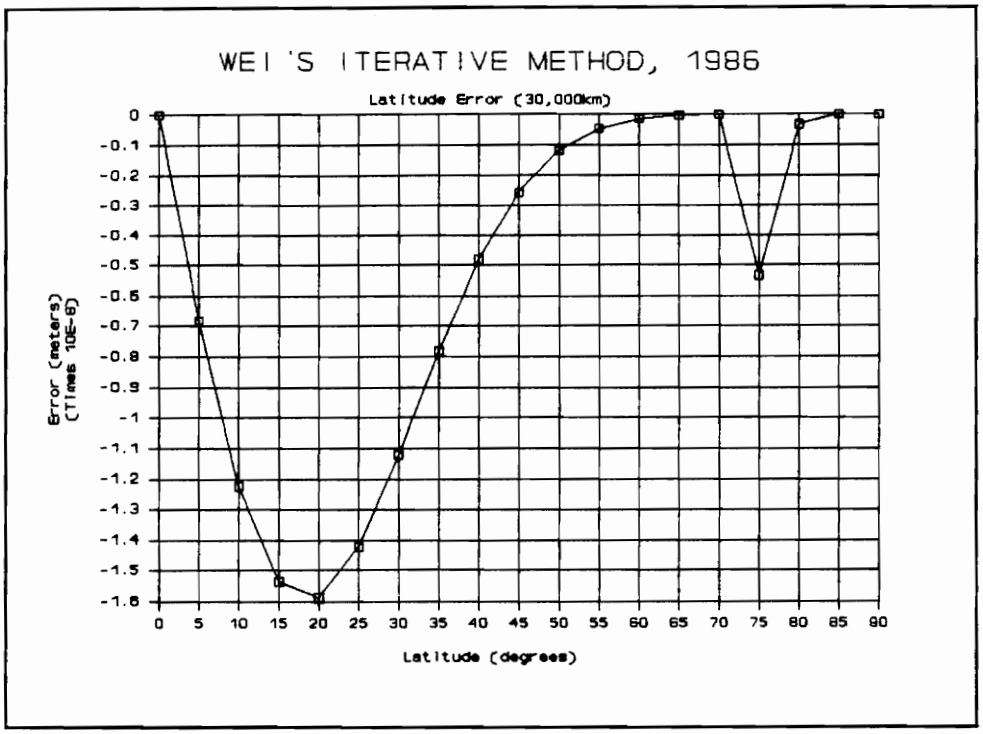

TORGE, 1975
Height Error (30,000km)
Error (meters)
(Times 10E-7)
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
-1
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

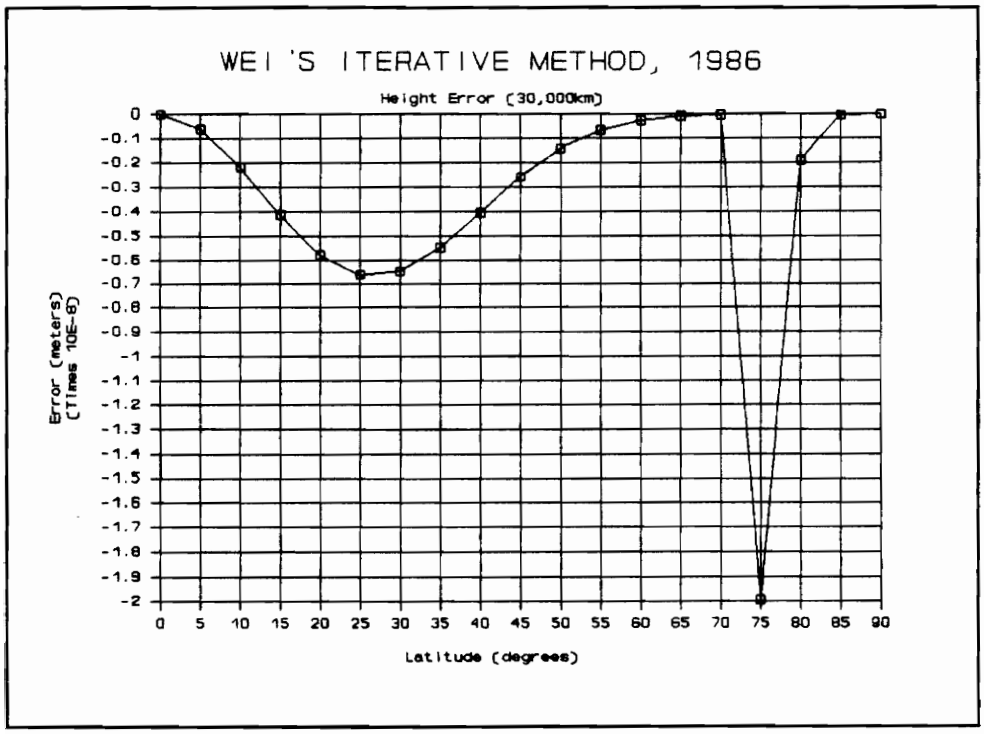

WEI 'S ITERATIVE METHOD, 1986
Latitude Error (0km)
Error (meters)
(Times 10E-7)
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
-0.8
-0.9
-1
-1.1
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)


WEI 'S ITERATIVE METHOD, 1986
Height Error (0km)
Error (meters)
(Times 10E-8)
0
-1
-2
-3
-4
-5
-6
-7
-8
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)


WEI 'S ITERATIVE METHOD, 1986
Latitude Error (15,000km)
Error (meters)
(Times 10E-7)
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
-0.8
-0.9
-1
-1.1
-1.2
-1.3
-1.4
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)


WEI 'S ITERATIVE METHOD, 1986
Height Error (15,000km)
Error (meters)
(Times 10E-8)
0
-1
-2
-3
-4
-5
-6
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)


WEI 'S ITERATIVE METHOD, 1986
Latitude Error (30,000km)
Error (meters)
(Times 10E-6)
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
-0.8
-0.9
-1
-1.1
-1.2
-1.3
-1.4
-1.5
-1.6
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)


WEI 'S ITERATIVE METHOD, 1986
Height Error (30,000km)
Error (meters)
(Times 10E-6)
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
-0.8
-0.9
-1
-1.1
-1.2
-1.3
-1.4
-1.5
-1.6
-1.7
-1.8
-1.9
-2
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
Latitude (degrees)

8.2 Appendix B, Recommended Algorithm

```
PROCEDURE bowring76(x,y,z:extended;VAR phi,h,time:extended);
VAR
  oldphi,beta,ff,capn,capm,eprimesqr,ee:extended;
  i:INTEGER;
BEGIN
  i:=0;
  IF((x=0.0)and(y=0.0))THEN{at the pole}
    begin
      i:=i+1;
      phi:=90*pi/180;
      h:=z-b
    end
  else
    begin
      ee:=(sqr(a)-sqr(b))/sqr(a);
      beta:=arctan((a/b)*(z/sqrt(sqr(x)+sqr(y))));{eqn.6.106}
      ff:=(a-b)/a;{eqn.3.3}
      eprimesqr:=(sqr(a)-sqr(b))/sqr(b);{eqn.3.5}
      phi:=arctan(z/sqrt(sqr(x)+sqr(y)));{initial value}
      REPEAT{Newton-Raphson method}
        i:=i+1;
        oldphi:=phi;
        phi:=arctan((z+eprimesqr*b*sqr(sin(beta))*sin(beta))/
(sqrt(sqr(x)+sqr(y))-a*ee*sqr(cos(beta))*cos(beta)));{eqn.6.
105}
        beta:=arctan((1-ff)*tan(phi));{eqn.6.107}
      UNTIL((abs(phi-oldphi)<=accuracy)OR(i=maxcount));

capm:=a*(1-ee)/rpower((1-ee*sqr(sin(phi))),3,2);{eqn.3.87}
      capn:=a/sqrt(1-ee*sqr(sin(phi)));{eqn.3.99}
      if(phi<=0.78)then
            h:=sqrt(sqr(x)+sqr(y))/cos(phi)-capn;{eqn.6.98}
      if(phi>0.78)then h:=z/sin(phi)-capn+ee*capn;{eqn.6.99}
                                                  (              *
h:=sqrt(sqr(sqrt(sqr(x)+sqr(y))-a*cos(beta))+sqr(z-b*sin(bet
a)));
      {eqn.6.108} *)
    end;
END;
```

8.3 Appendix 3, Program and Procedures

The Program is written in three parts:
1) Unit "Globals" which contains global variables and procedures that may be accessed by the other two program parts.
2) Unit "Algoritm" which contains the Algorithms.
3) The Main Program.

```
(***************BEGIN UNIT "GLOBALS"****************)
unit globals;
interface
uses
crt,dos;
PROCEDURE checkinput(VAR valid:BOOLEAN;VAR number:extended);
PROCEDURE fillarray;
PROCEDURE whichone;
PROCEDURE asign(VAR a,b,finv,f,esqr,eprimesq:extended);
PROCEDURE choose;
PROCEDURE latradtodeg(radians:extended;VAR degrees,minutes:INTEGER;
VAR seconds:extended);
PROCEDURE latquadrant(var quadrant:integer);
PROCEDURE longoctant(var octant:integer);
PROCEDURE longradtodeg(radians:extended;VAR degrees,minutes:INTEGER;
VAR seconds:extended);
FUNCTION tan(x:extended):extended;
FUNCTION arcsin(x:extended):extended;
FUNCTION arccos(x:extended):extended;
FUNCTION rpower(mantissa:EXTENDED;en{numerator}:EXTENDED;
ed{denominator}:EXTENDED):EXTENDED;
FUNCTION timediff(hr,min,sec,hun,hr2,min2,sec2,hun2:word):extended;
PROCEDURE inuvw{(VAR u,v,w:extended)};
FUNCTION lambda:extended;
PROCEDURE outllh;
PROCEDURE outlh;
PROCEDURE outl;
PROCEDURE degtorad(degrees,minutes:INTEGER;seconds:extended;VAR radians:extended);
PROCEDURE inerror;
PROCEDURE inllh{(VAR longitude,latitude,height:extended)};
PROCEDURE uvwcalc;
PROCEDURE outuvw;
PROCEDURE uvw;
PROCEDURE uvwdat;
PROCEDURE subtract(l1,h1,l2,h2:extended;VAR ldif,hdif:extended);

TYPE
  name=STRING[45]; {holds the name of an ellipsoid}

CONST{constants declared}
```

```
  {the semi-major axis, in meters                inverse polar flattening}
  {1.}     a1=6378137.0;                           finv1=298.257223563;
  {2.}     a2=6378206.4;                           finv2=294.978698;
  {3.}     a3=6378563.396;                         finv3=299.324964;
  {4.}     a4=6377397.155;                         finv4=299.152813;
  {5.}     a5=6378249.145;                         finv5=293.4663;
  {6.}     a6=6378249.145;                         finv6=293.465;
  {7.}     a7=6377276.345;                         finv7=300.8017;
  {8.}     a8=6378388.0;                           finv8=297;
  {9.}     a9=6378245.0;                           finv9=298.3;
  {10.}    a10=6378166.0;                          finv10=298.3;
  {11.}    a11=6378150.0;                          finv11=298.3;
  {12.}    a12=6378160.0;                          finv12=298.25;
  {13.}    a13=6378160.0;                          finv13=298.25;
  {14.}    a14=6378160.0;                          finv14=298.2471674273;
  {15.}    a15=6378135.0;                          finv15=298.26;
  {16.}    a16=6378140.0;{+/- 5.0}                 finv16=298.257;{+/- .0015}
  {17.}    a17=6378136.0;{+/- 1.0}                 finv17=298.257;
  {18.}    a18=6378137.0;                          finv18=298.257222101;

  accuracy=(1.0E-5*3.14159265358979)/(3600*180);
  {Accuracy desired between successive iterations, radians. Equivalent to 0.00001 seconds
of arc.}
  maxcount=20;          {maximum # of iterations allowed}

VAR{global variables declared}
  file1a,file1b:text;
  infile,uvwfile,llhfile:file of extended;
  fname,fname1,fname2:STRING[12];
  geocentric_file:BOOLEAN;{file choice variable}
  ellipsoid:ARRAY[1..2,1..18]OF extended;{holds the ellipsoid parameters listed}
  nm:ARRAY[1..18]OF name;{holds names of ellipsoids}
  b:extended;{the semi-minor axis, in meters}
  a:extended;{the semi-major axis, in meters}
  esqr:extended;{the first eccentricity squared; unitless}
  jreal:extended;{choice variable}
  j,k:INTEGER;{choice variables}
  choice:CHAR;{choice variable}
  f:extended;{polar flattening, unitless}
  eprimesq:extended;{the second eccentricity squared; unitless}
  finv:extended;{inverse polar flattening}
  hr,min,sec,hun,hr2,min2,sec2,hun2:word;{used to obtain system time}
  time,times:extended;
  u,v,w,longitude,latitude,height:extended;{geocentric and geodetic coordinates}
  seconds:extended;
  valid:BOOLEAN;{used to determine input validity}
  print:boolean;{used to determine if printer output or screen output}
  charactr:char;{dummy variable read after keypressed}
  totalit:integer;
```

```
  {holds total # of iterations per data set for iterative methods}
  datacount:real;{number of datapoints}
  iii:integer;{flag to determine screen or printer output}

implementation
PROCEDURE inerror;
  BEGIN
    textcolor(14);textbackground(4);
    writeln;
    WRITELN('Input error!');
    write('Press any key to continue..');
    charactr:=readkey;
  END{PROCEDURE in error};

FUNCTION tan(x:extended):extended;
{computes the tangent of x}
BEGIN
  if(cos(x)=0.0)then
  begin
    writeln('Attempted division by zero! See function tan.');
    x:=0.0;{to avoid division by zero}
  end;
  tan:=(SIN(x)/COS(x))
END;{function tan}

FUNCTION arcsin(x:extended):extended;
{computes the arcsine of an angle in radians.}
BEGIN
  if(abs(x)=1.0)then
    begin
      writeln('Attempted division by zero! Function arcsin.');
      x:=0.0
    end;
  arcsin:=ARCTAN(x/SQRT(1-SQR(x)))
END;{function arcsin}

FUNCTION arccos(x:extended):extended;
{computes the arccosine of an angle in radians.}
BEGIN
  arccos:=PI/2.0-arctan(x/sqrt(1-sqr(x)));
END;{function arccos}

FUNCTION rpower(mantissa:EXTENDED;en{numerator}:EXTENDED;
ed{denominator}:EXTENDED):EXTENDED;
{provides a means of raising a number to an extended power given the mantissa, exponent
numerator, and exponent denominator.}

FUNCTION ipower(mantissa:EXTENDED;exponent:INTEGER):EXTENDED;
{provides a means of raising a number to an integer power.}
```

```
VAR
  i:INTEGER;{counter}
  temp:EXTENDED;{temporary results}
BEGIN
  temp:=1.0;
  IF(exponent>0)THEN
    BEGIN{if}
      FOR i:=1 TO exponent DO
      temp:=temp*mantissa;
      ipower:=temp
    END{if}
  ELSE{recursive call}
    IF(exponent<0)THEN ipower:=ipower(1/mantissa,ABS(exponent))
  ELSE{exponent=0}
    ipower:=1.0
END;{function ipower}

BEGIN{rpower}
(*********************************************************************
writeln('in rpower');
write('mantissa: ',mantissa);write('  en: ',en);writeln('  ed: ',ed);
*********************************************************************)
  IF(ed=0.0)THEN
    BEGIN
      WRITE('Division by zero attempted! Check your input!');
      WRITELN(' See Function rpower.');
      ed{set}:=1.0{to avoid abnormal end to program}
    END;
  IF(mantissa=0.0)THEN rpower:=0.0
  ELSE
  IF(mantissa>0.0){the usual case}THEN
    BEGIN
      IF(en=0.0)THEN rpower:=1.0
      ELSE
      rpower:=EXP((en/ed)*LN(mantissa))
    END
  ELSE
  {mantissa<0.0}
    BEGIN
      IF{exponent numerator and denominator are whole #'s}
      ((FRAC(en)=0.0)AND(FRAC(ed)=0.0))THEN
      BEGIN
       mantissa:=ipower(mantissa,TRUNC(en));
       en:=1.0{exponent numerator set to 1.0 after utilizing function ipower}
      END;
      IF{en whole even #}(mantissa>0.0)THEN rpower:=EXP((en/ed)*LN(mantissa))
      ELSE
       IF{both numerator and denominator of exponent are whole odd #'s}
       ((mantissa<0.0)AND(ODD(TRUNC(ed))))THEN
```

```
          rpower:=-EXP((1.0/ed)*LN(ABS(mantissa)))
        ELSE
          BEGIN
            WRITE('Imaginary root involved! Check your input! ');
            WRITELN('See function rpower.');
            rpower[set]:=0.0;{to avoid abnormal end to program}
          END
      END{else}
  END{function rpower};

PROCEDURE checkinput(VAR valid:BOOLEAN;VAR number:extended);
{checks keyboard input for validity}
CONST
  eol=30;
VAR
  numberset:SET OF CHAR;{valid input}
  ch:CHAR;{the input}
  cr:CHAR;{carriage return}
  ii,jj,kk,ll,exponent:INTEGER;
  store:ARRAY[1..eol]OF CHAR;{used to store input characters}
BEGIN{PROCEDURE checkinput}
  ii:=1;valid:=true;
  cr:=CHR(13);{carriage return}
  numberset:=['0'..'9','.',cr,'-'];
  ch:=CHR(0);ii:=0;jj:=0;kk:=0;ll:=0;number:=0.0;{initializing variables}
  WHILE NOT(ch=cr)AND NOT(ii=eol)DO
    BEGIN{while}
      ch:=readkey;{taking input directly from keyboard}
      WRITE(ch);{echoing to screen}
      IF NOT(ch='-')THEN
        BEGIN{if}
          ii:=ii+1;
          store[ii]:=ch;{filling array with keyboard input}
        END;{if}
      IF((ch='.')AND(jj>0))THEN valid:=FALSE;
      IF(ch='.')THEN jj:=ii;
      IF NOT(ch IN numberset)THEN valid:=FALSE;
      IF(ch='-')THEN ll:=ll+1;
      IF((ch='-')AND(ii>0))THEN valid:=FALSE;
    END;{while}
  IF(store[1]=cr)THEN valid:=FALSE;
  IF(jj=0)THEN jj:=ii;
  IF(valid)THEN
    BEGIN{if}
      WHILE(kk<jj-1)DO
        BEGIN{while}
          kk:=kk+1;
          exponent:=jj-(kk+1);
          number:=(ORD(store[kk])-ORD('0'))*(rpower(10.0,exponent,1.0))+number;
```

```
        END;{while}
      kk:=jj;
      WHILE(kk<ii-1)DO
        BEGIN{while}
          kk:=kk+1;
                                e x p o n e n t : = j j - k k ;
number:=(ORD(store[kk])-ORD('0'))*(rpower(10.0,exponent,1.0))+
number;
        END;{while}
    END;{if}
  IF(ll=1)THEN number:=-number;
END;{PROCEDURE checkinput}

FUNCTION timediff(hr,min,sec,hun,hr2,min2,sec2,hun2:word):extended; {computes the difference,
in seconds, between two times}
BEGIN{function timediff}
  timediff:=(3600.0*hr2+60.0*min2+sec2+0.01*hun2)-
            (3600.0*hr+60.0*min+sec+0.01*hun)
END;{function timediff}

PROCEDURE fillarray;
{fills array 'ellipsoid' with values for a and finv, fills array 'nm' with names of
ellipsoids}
BEGIN
  ellipsoid[1,1]:=a1;                    ellipsoid[2,1]:=finv1;
  ellipsoid[1,2]:=a2;                    ellipsoid[2,2]:=finv2;
  ellipsoid[1,3]:=a3;                    ellipsoid[2,3]:=finv3;
  ellipsoid[1,4]:=a4;                    ellipsoid[2,4]:=finv4;
  ellipsoid[1,5]:=a5;                    ellipsoid[2,5]:=finv5;
  ellipsoid[1,6]:=a6;                    ellipsoid[2,6]:=finv6;
  ellipsoid[1,7]:=a7;                    ellipsoid[2,7]:=finv7;
  ellipsoid[1,8]:=a8;                    ellipsoid[2,8]:=finv8;
  ellipsoid[1,9]:=a9;                    ellipsoid[2,9]:=finv9;
  ellipsoid[1,10]:=a10;                  ellipsoid[2,10]:=finv10;
  ellipsoid[1,11]:=a11;                  ellipsoid[2,11]:=finv11;
  ellipsoid[1,12]:=a12;                  ellipsoid[2,12]:=finv12;
  ellipsoid[1,13]:=a13;                  ellipsoid[2,13]:=finv13;
  ellipsoid[1,14]:=a14;                  ellipsoid[2,14]:=finv14;
  ellipsoid[1,15]:=a15;                  ellipsoid[2,15]:=finv15;
  ellipsoid[1,16]:=a16;                  ellipsoid[2,16]:=finv16;
  ellipsoid[1,17]:=a17;                  ellipsoid[2,17]:=finv17;
  ellipsoid[1,18]:=a18;                  ellipsoid[2,18]:=finv18;
  nm[1]:='"WGS84"';
  nm[2]:='"NAD27" - based on the Clarke 1866 Ellipsoid';
  nm[3]:='"Airy 1830"';nm[4]:='"Bessel 1841"';
  nm[5]:='"Clarke 1880" (modified)';nm[6]:='"Clarke 1880"';
  nm[7]:='"Everest 1830"';nm[8]:='"International 1924"';
  nm[9]:='"Krassovski 1940"';nm[10]:='"Mercury 1960"';
  nm[11]:='"Modified Mercury 1968"';
```

```
  nm[12]:='"Australian National"';
  nm[13]:='"South America 1969"';
  nm[14]:='"GRS67"';nm[15]:='"WGS72"';
  nm[16]:='"International Association of Geodesy 1975"';
  nm[17]:='"International Association of Geodesy 1983"';
  nm[18]:='"GRS80"';
END;{fillarray}

PROCEDURE whichone;
{this procedure allows selection of a particular ellipsoid for assignment of values to "a"
(the semi-major axis) and "f" (the inverse flattening)}
BEGIN{PROCEDURE whichone}
  REPEAT{until valid and j in [1..18]}
    valid:=TRUE;jreal:=0.0;j:=0;
    CLRSCR;WRITELN;WRITELN;WRITELN;WRITELN;
    WRITELN('                1) ',nm[1]);
    WRITELN('                2) ',nm[2]);
    WRITELN('                3) ',nm[3]);
    WRITELN('                4) ',nm[4]);
    WRITELN('                5) ',nm[5]);
    WRITELN('                6) ',nm[6]);
    WRITELN('                7) ',nm[7]);
    WRITELN('                8) ',nm[8]);
    WRITELN('                9) ',nm[9]);
    WRITELN('               10) ',nm[10]);
    WRITELN('               11) ',nm[11]);
    WRITELN('               12) ',nm[12]);
    WRITELN('               13) ',nm[13]);
    WRITELN('               14) ',nm[14]);
    WRITELN('               15) ',nm[15]);
    WRITELN('               16) ',nm[16]);
    WRITELN('               17) ',nm[17]);
    WRITELN('               18) ',nm[18]);WRITELN;
    WRITE('Please enter a number between 1 and 18..');
    checkinput(valid,jreal);
    IF(jreal>MAXINT)THEN jreal:=maxint;
    j:=TRUNC(jreal);
    CLRSCR;
    IF ((18<j) or (j<0) OR (NOT valid))THEN
      BEGIN
        WRITELN;WRITELN;
        WRITELN('Only numbers between 1 and 18 are valid responses');
        write('Press any key to continue..');
        charactr:=readkey;
        CLRSCR;WRITELN
      END;{if}
  UNTIL((j IN[1..18])AND(valid))
END;{PROCEDURE whichone}
```

```
PROCEDURE asign(VAR a,b,finv,f,esqr,eprimesq:extended);
{this procedure makes assignment of variables based on the ellipsoid selected.}
  BEGIN{PROCEDURE asign}
  a:=ellipsoid[1,j];                {the semi-major axis of the ellipsoid}
  finv:=ellipsoid[2,j];             {inverse flattening}
  f:=1.0/finv;                      {the flattening}
  b:=a*(1.0-f);                     {the semi-minor axis of the ellipsoid}
  esqr:=(SQR(a)-SQR(b))/SQR(a);     {the first eccentricity squared}
  eprimesq:=(SQR(a)-SQR(b))/SQR(b);{the second eccentricity squared}
  WRITELN;WRITELN;WRITELN;WRITELN;WRITELN;WRITELN;WRITELN;
  WRITELN('Constants associated with the ',nn[j],':');WRITELN;
  WRITE('  ');WRITELN('The semi-major axis:              ',a:20:16,' meters');
  WRITE('  ');WRITELN('The semi-minor axis:              ',b:20:16,' meters');
  WRITE('  ');WRITELN('Inverse flattening:                  ',finv:20:16);
  WRITE('  ');WRITELN('The flattening:                      ',f:20:16);
  WRITE('  ');WRITELN('The first eccentricity squared:      ',esqr:20:16);
  WRITE('  ');WRITELN('The second eccentricity squared:     ',eprimesq:20:16);
  WRITELN;
  END;{PROCEDURE asign}

PROCEDURE choose;                   {loop:choosing an ellipsoid}
BEGIN                               {PROCEDURE choose}
  textcolor(0);textbackground(11);
  CLRSCR;
  REPEAT                            {until choice='n'}
    REPEAT                          {until input is correct}
      whichone;                     {choosing the ellipsoid}
    UNTIL((j>0)and(j<19)AND(valid));
  asign(a,b,finv,f,esqr,eprimesq);  {assigning values to variables}
  WRITELN('These parameters will be used for future computations.');
  WRITE('Choose another ellipsoid? y or n: ');READLN(choice);
  CLRSCR;WRITELN;
  UNTIL(choice IN['n','N'])
END;{PROCEDURE choose}

PROCEDURE latquadrant(var quadrant:integer);
{given the LSR coordinate w, returns the quadrant of the angle of latitude}
BEGIN{PROCEDURE latquadrant}
  IF(w>0)THEN
    quadrant:=1;{north latitude}
  IF(w<0)THEN
    quadrant:=2;{south latitude}
END;{PROCEDURE latquadrant}

PROCEDURE latradtodeg(radians:extended;VAR degrees,minutes:INTEGER;
VAR seconds:extended);
{converts radians to degrees, minutes, and seconds of latitude}
VAR
  angle:extended;
```

```
  quadrant:integer;
BEGIN{PROCEDURE latradtodeg}
  latquadrant(quadrant);
  angle:=(radians*180)/PI;
  degrees:=trunc(angle);
  minutes:=trunc((angle-degrees)*60);
  seconds:=((angle-degrees)*60-minutes)*60.0;
  if(round(seconds)>=60)then
    begin minutes:=minutes+1;seconds:=seconds-60.0;end;
  if(minutes>=60)then begin degrees:=degrees+1;minutes:=minutes-60;end;
  if(quadrant=2)then degrees:=-degrees;{South latitude}
END;{PROCEDURE latradtodeg}

PROCEDURE longoctant(var octant:integer);
{given the LSR coordinates u and v, returns the quadrant of the angle of longitude.}
BEGIN{function longoctant}
  IF((u>0)AND(v>0))THEN octant:=1;
  IF((u<0)AND(v>0))THEN octant:=2;
  IF((u<0)AND(v<0))THEN octant:=3;
  IF((u>0)AND(v<0))THEN octant:=4;
  if((u>0)and(v=0))then octant:=5;{0 degrees}
  if((u=0)and(v>0))then octant:=6;{90 degrees}
  if((u<0)and(v=0))then octant:=7;{180 degrees}
  if((u=0)and(v<0))then octant:=8 {270 degrees}
END;{function longoctant}

PROCEDURE longradtodeg(radians:extended;VAR degrees,minutes:INTEGER;
VAR seconds:extended);
{this procedure converts radians to degrees, minutes, and seconds of longitude}
VAR
  angle:extended;
  octant:integer;
BEGIN{PROCEDURE longradtodeg}
(*  longoctant(octant);
  case octant of
    2:radians:=radians+pi/2;
    3:radians:=radians+pi;
    4:radians:=radians+3*pi/2;
    5:angle:=0.0;
    6:angle:=90.0;
    7:angle:=180.0;
    8:angle:=270.0;
  end;{case}
  if(octant in[1..4])then angle:=radians*180/PI;        *)
  angle:=radians*180/PI;
  degrees:=TRUNC(angle);
  minutes:=TRUNC((angle-degrees)*60);
  seconds:=((angle-degrees)*60-minutes)*60;
  IF(round(seconds)>=60)THEN
```

```
    BEGIN minutes:=minutes+1;seconds:=seconds-60 END;
  IF(minutes>=60)THEN BEGIN degrees:=degrees+1;minutes:=minutes-60 END;
END;{PROCEDURE longradtodeg}

PROCEDURE inuvw{(VAR u,v,w:extended)};
{receives geocentric coordinates as input from keyboard}
BEGIN{PROCEDURE inuvw}
  REPEAT{until input is accepted by user}
   REPEAT{until input is valid}
     u:=0.0;v:=0.0;w:=0.0;valid:=TRUE;{initializing}
     CLRSCR;WRITELN;WRITELN;WRITELN;WRITELN;
     WRITE('               Input u in meters: ');(*checkinput(valid,u);*)
     readln(u);WRITELN;
     WRITE('               Input v in meters: ');(*checkinput(valid,v);*)
     readln(v);WRITELN;
     WRITE('               Input w in meters: ');(*checkinput(valid,w);*)
     readln(w);CLRSCR;
     WRITELN;WRITELN;WRITELN;WRITELN;
     WRITELN('    Input data: ');
     WRITELN('                u: ',u:20:15);
     WRITELN('                v: ',v:20:15);
     WRITELN('                w: ',w:20:15);
     WRITELN;WRITELN;
     IF(NOT valid)THEN inerror
   UNTIL(valid);
   WRITE('Is input correct? y or n: ');READLN(choice);WRITELN
  UNTIL(choice IN['y','Y']);
END;{PROCEDURE inuvw}

FUNCTION lambda:extended;
{computes longitude in radians}
BEGIN
  if((u=0.0)and(v<0.0))then lambda:=270*PI/180;
  if((u=0.0)and(v>0.0))then lambda:=90*PI/180;
  if((u=0.0)and(v=0.0))then lambda:=0.0;
  if(u<>0.0)then lambda:=arctan(abs(v)/abs(u))
END;{function lambda}

PROCEDURE outllh;
{receives longitude, latitude and height and prints results in degrees,
minutes, and seconds using PROCEDURE radtodeg.}
VAR
  degrees,minutes:INTEGER;
  seconds:extended;
BEGIN{PROCEDURE outllh}
  WRITELN;WRITELN;
  longradtodeg(longitude,degrees,minutes,seconds);
  WRITELN('Longitude is: ',degrees:3,' degrees ',minutes:2,' minutes ',
  seconds:13:10,' seconds');
```

```
  latradtodeg(latitude,degrees,minutes,seconds);
  WRITELN('Latitude is:  ',degrees:3,' degrees ',minutes:2,' minutes ',
  seconds:13:10,' seconds');
  WRITELN('Height is:     ',height:13:10,' meters');WRITELN;
END;{PROCEDURE outllh}

PROCEDURE outlh;
{receives latitude and height and prints results in degrees, minutes, and
seconds using PROCEDURE radtodeg.}
VAR
  degrees,minutes:INTEGER;
  seconds:extended;
BEGIN{PROCEDURE outlh}
  latradtodeg(latitude,degrees,minutes,seconds);
  WRITELN(degrees:4,' ',minutes:3,' ',seconds:20:5,'   ',height:20:5)
END;{PROCEDURE outlh}

PROCEDURE outl;
{receives longitude  and prints results in degrees, minutes, and seconds using PROCEDURE
radtodeg.}
VAR
  degrees,minutes:INTEGER;
  seconds:extended;
BEGIN{PROCEDURE outl}
  WRITELN;longradtodeg(longitude,degrees,minutes,seconds);
  WRITELN('Longitude:',degrees:3,' degrees ',minutes:2,' minutes ',
  seconds:7:5,' seconds; ');WRITELN
END;{PROCEDURE outl}

PROCEDURE degtorad(degrees,minutes:INTEGER;seconds:extended;VAR radians:extended);
{this procedure converts degrees,minutes, and seconds to radians.} VAR
  number:extended;
  BEGIN{PROCEDURE degtorad}
    number:=degrees+minutes/60+seconds/3600;
    radians:=number*(PI/180.0);
  END;{PROCEDURE degtorad}

PROCEDURE inllh;
{input longitude, latitude, and height. called by procedures uvw, datumconvert}
VAR
  degrees,minutes:INTEGER;
  seconds:extended;
BEGIN{PROCEDURE inllh}
 REPEAT{until user satisfied with input}
  REPEAT{until all longitude input valid}
   REPEAT{until longitude degrees input valid}
    CLRSCR;valid:=TRUE;WRITELN;
    WRITE('Enter whole degrees of longitude: ');
(*    checkinput(valid,jreal);  *)
```

```
readln(jreal);
    IF(jreal>360)THEN valid:=FALSE;
    IF(NOT valid)THEN inerror;
   UNTIL(valid);
   degrees:=TRUNC(jreal);
   REPEAT{until longitude minutes input valid}
    CLRSCR;WRITELN;
    WRITE('Enter whole minutes of longitude: ');
(*    checkinput(valid,jreal); *)
readln(jreal);
    IF(jreal>60)THEN valid:=FALSE;
    IF(NOT valid)THEN inerror;
   UNTIL(valid);
   minutes:=TRUNC(jreal);
   REPEAT{until longitude seconds input valid}
    CLRSCR;WRITELN;
    WRITE('Enter seconds of longitude: ');
(*    checkinput(valid,seconds);*)
readln(jreal);
    seconds:=jreal;
    IF(seconds>=60)THEN valid:=FALSE;
    IF(NOT valid)THEN inerror;
   until(valid);
   UNTIL(valid);
   IF(degrees+minutes+seconds>360)THEN valid:=FALSE;
  UNTIL(valid);
   WRITELN;
   degtorad(degrees,minutes,seconds,longitude);
  REPEAT{until all latitude input valid}
   REPEAT{until latitude degrees input valid}
    CLRSCR;WRITELN;
    WRITE('Enter whole degrees of latitude: ');
(*    checkinput(valid,jreal); *)
readln(jreal);
    IF(jreal>90)THEN valid:=FALSE;
    IF(NOT valid)THEN inerror;
   UNTIL(valid);
   degrees:=TRUNC(jreal);
   REPEAT{until latitude minutes input valid}
    CLRSCR;WRITELN;
    WRITE('Enter whole minutes of latitude: ');
(*    checkinput(valid,jreal); *)
readln(jreal);
    IF(jreal>=60)THEN valid:=FALSE;
    IF(NOT valid)THEN inerror;
   UNTIL(valid);
   minutes:=TRUNC(jreal);
   REPEAT{until latitude seconds input valid}
    CLRSCR;WRITELN;
```

```
    WRITE('Enter seconds of latitude: ');
(*    checkinput(valid,seconds); *)
readln(seconds);
    IF(seconds>=60)THEN valid:=FALSE;
    IF(NOT valid)THEN inerror;
   UNTIL(valid);
  UNTIL(valid);
   degtorad(degrees,minutes,seconds,latitude);
   REPEAT{until height input valid}
    CLRSCR;WRITELN;
    WRITE('Enter the height in meters: ');
(*    checkinput(valid,height);   *)
readln(height);
    IF(NOT valid)THEN inerror;
   UNTIL(valid);
END;{PROCEDURE inllh}

PROCEDURE uvwcalc;
{computes geocentric cartesian coordinates given longitude,latitude,
and height.}
VAR
  n:extended;{the length of the normal line from the surface of the ellipsoid
  to the intersection of this line with the minor axis.  p35 'rapp'.}
BEGIN
  n:=a/(SQRT(1-esqr*SQR(SIN(latitude))));
  u:=(n+height)*COS(latitude)*COS(longitude);
  v:=(n+height)*COS(latitude)*SIN(longitude);
  w:=(n*(1-esqr)+height)*SIN(latitude);
END;{PROCEDURE uvwcalc}

PROCEDURE outuvw;
{prints out result of PROCEDURE uvw.}
BEGIN{PROCEDURE outuvw}
  WRITELN;
  WRITELN('U= ',u:20:15,' meters');
  WRITELN('V= ',v:20:15,' meters');
  WRITELN('W= ',w:20:15,' meters');
END;{PROCEDURE outuvw}

PROCEDURE uvw;
{calculates geocentric cartesian coordinates(u,v,w), given longitude,
latitude, and height.}
BEGIN{PROCEDURE uvw}
  choose;{selecting the ellipsoid}
  textcolor(14);textbackground(5);
  CLRSCR;inllh;uvwcalc;clrscr;outllh;outuvw;
  write('Press any key to continue..');
  charactr:=readkey;
END;{PROCEDURE uvw}
```

```
PROCEDURE uvwdat;
{produces a data file of geocentric coordinates (u,v,w) over a specific range of latitude
and height. longitude is held constant at zero since a profile of the ellipsoid is being
considered. data saved to uvw.fil will consist of rigorous values of geocentric coordinates
to be used for comparison of geodetic algorithms.}
VAR
  phi:longint;{latitude, minutes}
  phimin:longint;{minimum value for phi}
  phimax:longint;{maximum value for phi}
  phistep:longint;{increment for phi}
  heightmin:longint;{minimum value for height}
  heightmax:longint;{maximum value for height}
  heightstep:longint;{increment for height}
  degrees:longint;
  minutes:longint;
  seconds:extended;
  n:extended;{the length of the normal line from the surface of the ellipsoid to the
intersection of this line with the minor axis.  p35 'rapp'.}
  iiii:integer;{flag}
BEGIN{uvwdat}
 degrees:=0; minutes:=0;seconds:=0.0;longitude:=0.0;
 writeln('      1) Use canned values; PHI 0-90 degrees');
 writeln('      2) User inputs parameters');
 write('      3) Use canned values; PHI 89 degrees 59 minutes ');  writeln('45 seconds to
90 degrees');
 writeln('      4) Use canned values; PHI 0-15 seconds');
 readln(iiii);
 if(iiii=1)then
   begin
     phistep:=300;{increment for phi;minutes of arc}
     phimax:=5400;{maximum value for phi, minutes of arc}
     phimin:=0; {minimum value for phi}
     heightmin:=0;
     heightstep:=5000000;{increment for height;meters;5000 km}
     heightmax:=30000000;
     {maximum value for height;meters;30,000 km}
     datacount:=((phimax-phimin)/phistep+1)*
     ((heightmax-heightmin)/heightstep+1);
     clrscr;writeln('These values give ',trunc(datacount),' datapoints.');
     write('Press any key to continue..');
     charactr:=readkey;
     clrscr;
     {these limits and increments give 133 pairs of difference points per algorithm}
   end;{if}
 if(iiii=2)then
 begin
(* repeat{until input data correct}
  writeln('Object is to have <=133 data points.');*)writeln;
```

```
  repeat writeln('enter phimin (integer minutes, try 0): ');readln(phimin);
  until(phimin>=0);
  repeat writeln('enter phimax(integer minutes, try 5400): ');readln(phimax);
  until(phimax>=phimin);
  writeln('enter phistep(integer minutes, try 300): ');readln(phistep);
  repeat writeln('enter heightmin (kilometers, try 0): ');
  readln(heightmin);until(heightmin>=0);
  heightmin:=heightmin*1000;
  repeat writeln('enter heightmax (kilometers, try 30000): ');
  readln(heightmax);
  until(heightmax>=heightmin);
  heightmax:=heightmax*1000;
  writeln('enter heightstep(kilometers, try 5000): ');readln(heightstep);
  heightstep:=heightstep*1000;
datacount:=((phimax-phimin)/phistep+1)*((heightmax-heightmin)/heightstep+1);
  clrscr;writeln('These values give ',trunc(datacount),' datapoints.');
  write('Press any key to continue..');
  charactr:=readkey;
  clrscr;
(* until(trunc(datacount)<=133);          *)
 end;{if}
 if(iiii=3)then
   begin
     phistep:=1;{increment for phi;seconds of arc}
     phimax:=324000;{maximum value for phi, seconds of arc; 90 degrees}
      phimin:=323985; {minimum value for phi, seconds of arc; 89 degrees 59 minutes 45
seconds}
     heightmin:=0;
     heightstep:=5000000;{increment for height;meters;5000 km}
     heightmax:=30000000;
     {maximum value for height;meters;30,000 km}
     datacount:=((phimax-phimin)/phistep+1)*
     ((heightmax-heightmin)/heightstep+1);
     clrscr;writeln('These values give ',trunc(datacount),' datapoints.');
     write('Press any key to continue..');
     charactr:=readkey;
     clrscr;
   end;{if}
 if(iiii=4)then
   begin
     phistep:=1;{increment for phi;seconds of arc}
     phimax:=15;{maximum value for phi, seconds of arc}
     phimin:=0; {minimum value for phi, seconds of arc}
     heightmin:=0;
     heightstep:=5000000;{increment for height;meters;5000 km}
     heightmax:=30000000;
     {maximum value for height;meters;30,000 km}
     datacount:=((phimax-phimin)/phistep+1)*
     ((heightmax-heightmin)/heightstep+1);
```

```
    clrscr;writeln('These values give ',trunc(datacount),' datapoints.');
    write('Press any key to continue..');
    charactr:=readkey;
    clrscr;
   end;{if}
  phi:=phimin;
  if((iiii=1)or(iiii=2))then minutes:=phi;
  if((iiii=3)or(iiii=4))then seconds:=phi;
  ASSIGN(uvwfile,'uvw.fil');REWRITE(uvwfile);
  ASSIGN(llhfile,'llh.fil');REWRITE(llhfile);
  WRITE('filling llh.fil(geodetic coordinates)');
  write(' and uvw.fil(geocentric coordinates)');
  REPEAT{until phi > phimax}
    height:=heightmin;
    REPEAT{until height > heightmax}
      degtorad(degrees,minutes,seconds,latitude);{converting to radians}
(*****************a modified version of uvwcalc follows*********************)
(****calculates geocentric coordinates given longitude,latitude, and height***)
      n:=a/(SQRT(1-esqr*SQR(SIN(latitude))));
      u:=(n+height)*COS(latitude)*COS(longitude);
      v:=(n+height)*COS(latitude)*SIN(longitude);{zero when longitude=0}
      w:=(n*(1-esqr)+height)*SIN(latitude);
      {storing geocentric coordinates in uvw.fil}
      WRITE(uvwfile,u);write(uvwfile,v);WRITE(uvwfile,w);
      {storing geodetic coordinates(radians) in llh.fil}
      WRITE(llhfile,latitude);write(llhfile,longitude);
WRITE(llhfile,height);
      height:=height+heightstep
    UNTIL(height>heightmax);
    phi:=phi+phistep;
    if((iiii=1)or(iiii=2))then minutes:=phi;
    if((iiii=3)or(iiii=4))then seconds:=phi;
    write('.')
  UNTIL(phi>phimax);
  CLOSE(uvwfile);CLOSE(llhfile)
END{PROCEDURE uvwdat};

PROCEDURE subtract(l1,h1,l2,h2:extended;VAR ldif,hdif:extended); {computes the difference
between actual and computed latitudes and heights by subtracting actual from computed values}

BEGIN{PROCEDURE subtract}
  ldif:=l1-l2;hdif:=h1-h2
END{PROCEDURE subtract};
end{unit globals}.
(******************END OF UNIT "GLOBALS"*********************)

(******************BEGIN UNIT "ALGORITM"*********************)
unit algoritm;
```

```
INTERFACE
uses
    globals;
PROCEDURE gersten60(x,y,z:extended;VAR phi,h,time:extended);
PROCEDURE baird64(x,y,z:extended;VAR phi,h,time:extended);
PROCEDURE paul73(x,y,z:extended;VAR phi,h,time:extended);
PROCEDURE heikkinen82(x,y,z:extended;VAR phi,h,time:extended);
PROCEDURE lpash85A(x,y,z:extended;VAR phi,h,time:extended);
PROCEDURE bowring85(x,y,z:extended;VAR capb,h,time:extended);
PROCEDURE bowring88(x,y,z:extended;VAR phi,h,time:extended);
PROCEDURE weidirect86(x,y,z:extended;VAR psi,h,time:extended);
PROCEDURE weiiterate86(x,y,z:extended;VAR psi,h,time:extended);
PROCEDURE eissfeller86(x,y,z:extended;VAR psi,h,time:extended);
PROCEDURE goad87(x,y,z:extended;var phi,ha,time:extended);
PROCEDURE morrison(x,y,z:extended;VAR phi,h,time:extended);
PROCEDURE torge(x,y,z:extended;VAR phi,h,time:extended);
PROCEDURE geod(x,y,z:extended;VAR phi,h,time:extended);
PROCEDURE hm63(x,y,z:extended;var phi,h,time:extended);
PROCEDURE nautiyal(x,y,z:extended;VAR phi,h,time:extended);
PROCEDURE hedman(x,y,z:extended;var phi,h,time:extended);
PROCEDURE romao87(x,y,z:extended;VAR phi,he,time:extended);
PROCEDURE bowring76(x,y,z:extended;VAR phi,h,time:extended);

IMPLEMENTATION
uses
    dos;

PROCEDURE gersten60(x,y,z:extended;VAR phi,h,time:extended);
{ref:The Journal of the Astronautical Sciences,Technical Notes,"Geodetic Sublatitude and
Altitude of a Space Vehicle",1961, by Robert H. Gersten.}
VAR
  ee,r,epsilon,phiprime,sinphi,cosphi:extended;
BEGIN{procedure gersten60}
  gettime(hr,min,sec,hun);
  if((x=0)and(y=0))then{at the pole}
    begin
      phi:=90*pi/180;
      h:=z-b
    end
  else
  begin
    ee:=(sqr(a)-sqr(b))/sqr(a);
    r:=SQRT(SQR(x)+SQR(y)+SQR(z));
    epsilon:=a*ee/r;
    phiprime:=arcsin(z/r);{geocentric latitude}
    sinphi:=SIN(phiprime)*(1+epsilon*SQR(COS(phiprime)));{eqn.7a}
    cosphi:=COS(phiprime)*(1-epsilon*SQR(SIN(phiprime)));{eqn.7b}
    phi:=ARCTAN(sinphi/cosphi);
    h:=r-a*(1-0.5*ee*(1+epsilon)*SQR(SIN(phiprime))+0.5*ee*(epsilon-0.25*
```

```
    ee)*rpower(SIN(phiprime),4.0,1.0));
  end;
  gettime(hr2,min2,sec2,hun2);
  time:=timediff(hr,min,sec,hun,hr2,min2,sec2,hun2);
END;{procedure gersten60}

PROCEDURE baird64(x,y,z:extended;VAR phi,h,time:extended);
{ref:Cartesian Coordinate to Geodetic Coordinate Conversions by Robert W. Baird 1 June 1964.
The original program was written in FORTRAN IV and was designed to convert position data
recorded in rectangular cartesian coordinates into geodetic coordinates. Baird's equations
have been used in this subroutine.
Newton's method of iteration to solve the quartic transcendental equation in z1 is used.
The number of iterations is determined by the difference in successive values of z1 with a
maximum limit on # of iterations set to 20.}
VAR
  k1,k2,k3,k4,k5,k6,k7,k8,k9:extended;   {the coefficients of powers of z1;p20}
  ee,yy,z1,rzero,fz1,fprimez1,zold,term1:extended;
  i:INTEGER;                             {counter}
BEGIN{procedure baird64}
  gettime(hr,min,sec,hun);
  i:=0;
  if((x=0)and(y=0))then{at the pole}
    begin
      i:=i+1;
      phi:=90*pi/180;
      h:=z-b
    end
  else
  if(z=0)then{at the equator}
    begin
      i:=i+1;
      phi:=0;
      h:=SQRT(SQR(x)+SQR(y))-a
    end
  else
    begin
      ee:=(sqr(a)-sqr(b))/sqr(a);
      rzero:=SQRT(SQR(x)+SQR(y));
      k8:=1-ee;
      k1:=-SQR(ee)/k8;k2:=-2.0*ee;k7:=a;k9:=SQR(k7);
      k3:=k9*SQR(ee);k4:=-k8;k5:=2.0*k9*ee*k8;k6:=k9*SQR(k8);
      z1:=k7*z*SQRT(k8)/SQRT((k8)*(SQR(x)+SQR(y))+SQR(z));
      {first approx.;p20,z2}
      REPEAT{Newton's Iterative Procedure}
        fz1:=k1*rpower(z1,4.0,1.0)+k2*z*rpower(z1,3.0,1.0)+(k3+k4*SQR(z)-
        SQR(rzero))*SQR(z1)+k5*z*z1+k6*SQR(z);{the function,p20}
        fprimez1:=4*k1*rpower(z1,3.0,1.0)+3.0*k2*z*SQR(z1)+2*(k3+k4*SQR(z)-
        SQR(rzero))*z1+k5*z;{derivative of the function,p20}
        yy:=z1-fz1/fprimez1;
```

```
        zold:=z1;
        z1:=yy;
        i:=i+1;
      UNTIL((ABS(z1-zold)<=accuracy)OR(i=maxcount));
      term1:=((k8)*(k9*k8-SQR(z1)));
      phi:=arctan(z1/sqrt(term1));
      h:=z/SIN(phi)-(k7*k8)/(SQRT(1-ee*SQR(SIN(phi))));
    end;
  gettime(hr2,min2,sec2,hun2);
  time:=timediff(hr,min,sec,hun,hr2,min2,sec2,hun2);
  WRITELN('baird64 # of iterations: ',i);
  totalit:=totalit+i;
END;{procedure baird64}

PROCEDURE paul73(x,y,z:extended;VAR phi,h,time:extended);
{eqns 7&8 required writing function rpower}
VAR
  ee,p,alpha,beta,q,t1,capn,term1,term2:EXTENDED;
BEGIN
  gettime(hr,min,sec,hun);
  if((x=0)and(y=0))then{at the pole}
    begin
      phi:=90*pi/180;
      h:=z-b
    end
  else
  if(z=0)then{at the equator}
    begin
      phi:=0;
      h:=SQRT(SQR(x)+SQR(y))-a
    end
  else
    begin
      ee:=(sqr(a)-sqr(b))/sqr(a);
      p:=SQRT(SQR(x)+SQR(y));{eqn.4}
      alpha:=(SQR(p)+SQR(a)*SQR(ee))/(1-ee);{eqn.5}
      beta:=(SQR(p)-SQR(a)*SQR(ee))/(1-ee);{eqn.6}
      term1:=(2*rpower((SQR(z)+beta),3.0,1.0));{eqn.7}
      term2:=((27*SQR(z)*(SQR(alpha)-SQR(beta))));
      (*q:=1+((27*SQR(z)*(SQR(alpha)-SQR(beta)))/
      (2*rpower((SQR(z)+beta),3.0,1.0)));*)
      q:=1+term2/term1;
      t1:=((SQR(z)+beta)/12)*(rpower((q+SQRT(SQR(q)-1)),1.0,3.0)+
      rpower((q+SQRT(SQR(q)-1)),-1.0,3.0))-beta/6+SQR(z)/12;{eqn.8}
      IF(t1>0.0)THEN
        begin
          phi:=ARCTAN((z/2+SQRT(t1)+SQRT(-beta/2+SQR(z)/4-t1+alpha*z/
          (4*SQRT(t1))))/p)
        end
```

```
      ELSE
        begin
          phi:=ARCTAN(((alpha+beta+SQRT(SQR(alpha)-SQR(beta))*z)/(2*beta*p))-
          (SQRT(SQR(alpha)-SQR(beta))*SQR(alpha+SQRT(SQR(alpha)-SQR(beta)))*
          rpower(z,3.0,1.0)/(4*rpower(beta,4.0,1.0)*p)));{eqn.9}
        end;
      capn:=a/SQRT(1-ee*SQR(SIN(phi)));{eqn.3}
      h:=p/COS(phi)-capn;{eqn.1d solved for h}
    end;
  gettime(hr2,min2,sec2,hun2);
  time:=timediff(hr,min,sec,hun,hr2,min2,sec2,hun2);
END{procedure paul73};

PROCEDURE heikkinen82(x,y,z:extended;VAR phi,h,time:extended);
VAR
ee,eesqr,capf,r,cape,capg,c,s,capp,capq,r0,capu,capv,z0,sinphi,cosphi:extended;
BEGIN
  gettime(hr,min,sec,hun);
  if((x=0)and(y=0))then{at the pole}
    begin
      phi:=90*pi/180;
      h:=z-b
    end
  else
    begin
      ee:=(sqr(a)-sqr(b))/sqr(a);
      r:=SQRT(SQR(x)+SQR(y));
      cape:=SQRT(sqr(a)-sqr(b));
      capg:=SQR(r)+(1-ee)*SQR(z)-ee*SQR(cape);
      capf:=54.0*SQR(b)*SQR(z);
      c:=SQR(ee)*capf*SQR(r)/rpower(capg,3.0,1.0);
      s:=rpower((SQRT(1+c+SQRT(SQR(c)+2.0*c))),1.0,3.0);
      capp:=capf/(3.0*SQR(s+1/s+1.0)*SQR(capg));
      capq:=SQRT(1.0+2.0*SQR(ee)*capp);
      r0:=-(capp*ee*r/(1+capq))+
      SQRT(((SQR(a)/2)*(1+1/capq)-(capp*(1-ee)*SQR(z))/(capq*(1+capq))-
      capp*SQR(r)/2));
      capu:=SQRT(SQR(r-ee*r0)+SQR(z));
      capv:=SQRT(SQR(r-ee*r0)+(1-ee)*SQR(z));
      z0:=SQR(b)*z/(a*capv);
      h:=capu*(1-SQR(b)/(a*capv));
      eesqr:=(sqr(a)-sqr(b))/sqr(b);
      sinphi:=(z+eesqr*z0)/SQRT(SQR(r)+SQR(z+eesqr*z0));
      cosphi:=r/SQRT(SQR(r)+SQR(z+eesqr*z0));
      phi:=ARCTAN(sinphi/cosphi);
    end;
  gettime(hr2,min2,sec2,hun2);
  time:=timediff(hr,min,sec,hun,hr2,min2,sec2,hun2);
END{procedure heikkinen82};
```

```
PROCEDURE lpash85A(x,y,z:extended;VAR phi,h,time:extended);
{ref: A New Algorithm for the Computation of the Geodetic Coordinates as a Function of Earth
Centered Earth-Fixed Coordinates (algorithm A), by L.O. Lupash, Journal of Guidance, Control,
and Dynamics, Vol.8,#6,Nov-Dec, 1985, Pages 787-789 and Correction Sheet.}
VAR
  azero,aone,atwo,athree,afour:extended;{coefficients of powers of mu}
  mu,fmu,fprimmu,r,temp:extended;
  bb,cc,dd,ff:extended;
  kk:INTEGER;
  ee:extended;
BEGIN{procedure lpash85A}
  gettime(hr,min,sec,hun);
  kk:=0;
  if(z=0)then{at the equator}
    begin
      kk:=kk+1;
      h:=sqrt(sqr(x)+sqr(y))-a;
      phi:=0.0
    end
  else
  if((x=0)and(y=0))then{at the pole}
    begin
      phi:=90*pi/180;
      h:=z-b
    end
  else
    begin
      ee:=(sqr(a)-sqr(b))/sqr(a);
      bb:=(SQR(x)+SQR(y))/SQR(a);cc:=SQR(z)/SQR(a);                    {eqns.10}
      ff:=bb+cc+ee;dd:=ff-cc*ee-ee*(ee-1);              {eqns 15}
      azero:=SQR(ee)*SQR(ee-1);                         {eqn. 14}
      aone:=2.0*ee*(ee-1)*(dd-2.0*bb);                  { "   " }
      atwo:=SQR(dd)-2.0*ee*(ee-1)*ff-4.0*ee*(3.0-2.0*ee)*bb;{ "   " }
      athree:=-2.0*dd*ff-4.0*bb*ee*(ee-3.0);            { "   " }
      afour:=SQR(ff)-4.0*bb*ee;                         { "   " }
      mu:=1.0/(1-((ee*cc)/(SQR(1-ee)*bb+cc)));{zeroth iteration: eqn.18}
      REPEAT{solving by the Newton-Raphson technique}
        fmu:=azero*rpower(mu,4.0,1.0)+aone*rpower(mu,3.0,1.0)+atwo*SQR(mu)+
        athree*mu+afour;                                {eqn.13}
        fprimmu:=4.0*azero*rpower(mu,3.0,1.0)+3.0*aone*SQR(mu)+2.0*
        atwo*mu+athree;                                 {eqn.17}
        temp:=mu;
        mu:=mu-fmu/fprimmu;                             {eqn.16}
        kk:=kk+1;
        if((mu<1.0)or(mu>=(1.0/(1.0-ee))))then
          begin
            writeln('In Lupash85A, mu out of range! See equation 24.');
            writeln('mu: ',mu);
```

```
        end;
      if(mu=0.0)then
        begin
         mu:=accuracy;{to avoid division by zero}
         writeln('Attempted division by zero! See Lupash85A, mu.');
        end;
    UNTIL((mu-temp)<=accuracy)OR(kk>maxcount);
    (*UNTIL(abs((mu-temp)/mu)<=accuracy)OR(kk>maxcount);
    modification of Lupash method *)
    IF((mu<temp)OR(kk>maxcount))THEN
          BEGIN
            WRITEln('Solution does not converge!');
            WRITE(' See procedure lpash85aA! ');
            writeln('maximum iteration or mu(kk)<mu(kk-1)');
          END;
    r:=a*SQRT(mu);                                              {eqn.19}
    h:=r*(-1.0+SQRT(abs(bb*esqr/((esqr-1.0)*mu+1.0))));{eqn.21}
    {absolute value taken in eqn.21}
    phi:=ARCTAN(SQRT((mu-1.0)/(esqr-1.0)*mu+1.0));     {eqn.22}
  end;
 gettime(hr2,min2,sec2,hun2);
 time:=timediff(hr,min,sec,hun,hr2,min2,sec2,hun2);
 WRITELN('lpash85a # of iterations: ',kk);
 totalit:=totalit+kk;
END;{procedure lpash85A}

PROCEDURE bowring85(x,y,z:extended;VAR capb,h,time:extended);
{ref:survey review vol.28,218,october 1985. Computes height using an improved height formula
which is insensitive to latitude to the first order of differentials. Also computes a
non-iterative solution for latitude from given Cartesian space coordinates. Called by
procedure sequence1.}
VAR
 p:extended;      {distance in two dimensions}
 capr:extended;   {polar distance}
 uu:extended;     {an intermediate,eqn.(17)}
 nu:extended;     {length of the normal terminated by the minor axis}
 eesqr:extended;  {the second eccentricity squared}
 ee:extended;     {the first eccentricity squared}
BEGIN{procedure bowring85}
 gettime(hr,min,sec,hun);  {time of beginning of procedure}
 if((x=0)and(y=0))then{at the pole}
   begin
     capb:=90*pi/180;
     h:=z-b
   end
 else
   begin
     ee:=(sqr(a)-sqr(b))/sqr(a);
     eesqr:=(sqr(a)-sqr(b))/sqr(b);
```

```
      p:=SQRT(SQR(x)+SQR(y));
      capr:=SQRT(SQR(p)+SQR(z));
      uu:=ARCTAN(b*z*(1+eesqr*b/capr)/(a*p));{eqn.17}
      capb:=ARCTAN((z+eesqr*b*rpower(SIN(uu),3.0,1.0))/
      (p-ee*a*rpower(COS(uu),3.0,1.0)));{eqn.18, geodetic latitude}
      mu:=a/(SQRT(1-ee*SQR(SIN(capb))));{eqn.11}
      h:=p*COS(capb)+z*SIN(capb)-(SQR(a)/mu);{eqn.7, geodetic height}
    end;
  gettime(hr2,min2,sec2,hun2);
  time:=timediff(hr,min,sec,hun,hr2,min2,sec2,hun2);
END;{procedure bowring85}

PROCEDURE bowring88(x,y,z:extended;VAR phi,h,time:extended);
{ref:"Coordinate Systems Used In Geodesy: Basic Definitions and Concepts" By Tomas Soler
and Larry D. Hothem, Journal of Surveying Engineering, Vol. 114, No. 2, May, 1988. Computes
height using a modification of the Bowring, 1985 method; an improved height formula which
is insensitive to latitude to the first order of differentials. Also computes a non-iterative
solution for latitude
Called by procedure sequence1.}
VAR
  p:extended;      {distance in two dimensions, eqn. (6)}
  rr:extended;     {polar distance, eqn. (7)}
  mu:extended;     {eqn. (8)}
  ee:extended;     {the first eccentricity squared}
  flat:extended;   {polar flattening}
BEGIN{procedure bowring88}
  gettime(hr,min,sec,hun);  {time of beginning of procedure}
  if((x=0)and(y=0))then{at the pole}
    begin
      phi:=90*pi/180;
      h:=z-b
    end
  else
    begin
      ee:=(sqr(a)-sqr(b))/sqr(a);
      flat:=(a-b)/a;
      p:=SQRT(SQR(x)+SQR(y));              {eqn. 6}
      rr:=SQRT(SQR(p)+SQR(z));             {eqn. 7}
      mu:=ARCTAN((z/p)*((1-flat)+(ee*a/rr)));{eqn.8}
      phi:=ARCTAN((z*(1.0-flat)+(ee*a*rpower(SIN(mu),3.0,1.0)))/
      ((1.0-flat)*(p-(ee*a*rpower(COS(mu),3.0,1.0)))));
      {eqn.3, geodetic latitude}
      h:=p*COS(phi)+z*SIN(phi)-a*sqrt(1.0-ee*sqr(sin(phi)));
      {eqn.4, geodetic height}
    end;
  gettime(hr2,min2,sec2,hun2);
  time:=timediff(hr,min,sec,hun,hr2,min2,sec2,hun2);
END;{procedure bowring88}
```

```
PROCEDURE weidirect86(x,y,z:extended;VAR psi,h,time:extended);
{ref:Report No. 370, Dept. of Geodetic Science and Surveying, The Ohio State University, by
Ziqing Wei. A direct solution for the conversion of Cartesian coordinates into latitude(psi)
and height(h).}
VAR
  alpha,beta,ee,q,capA,t2plust3,twosqrtt2t3,sqrtt1,kk:extended;
BEGIN{procedure weidirect86}
  gettime(hr,min,sec,hun);
  if((x=0.0)and(y=0.0))then{at the pole}
    begin
      psi:=90*pi/180;
      h:=z-b
    end
  else
    begin
      ee:=(sqr(a)-sqr(b))/sqr(a);
      alpha:=(SQR(x)+SQR(y))/SQR(a)+(1-ee)*SQR(z)/SQR(a);
      beta:=(SQR(x)+SQR(y))/SQR(a)-(1-ee)*SQR(z)/SQR(a);
      q:=1.0+(27.0*SQR(ee)*(SQR(alpha)-SQR(beta))/
      (2.0*rpower((alpha-SQR(ee)),3.0,1.0)));
                c a p A : = - q + S Q R T ( S Q R ( q ) - 1 ) ;
t2plust3:=ABS((1/3)*(alpha+SQR(ee)/2.0)-(((alpha-SQR(ee))/12.0)*
      (rpower(capA,1.0,3.0)+rpower(capA,-1.0,3.0))));
      twosqrtt2t3:=ABS(SQRT(SQR(t2plust3)+
      (SQR((alpha-SQR(ee))*(rpower(capA,1.0,3.0)-
      rpower(capA,-1.0,3.0)))/48)));
      sqrtt1:=-ee*beta/(4*twosqrtt2t3);
        kk:=ABS(sqrtt1+ABS(SQRT(t2plust3+twosqrtt2t3))-(1-ee/2));
h:=(kk/(1+kk))*SQRT(SQR(x)+SQR(y)+SQR((1+kk)/(1-ee+kk))*SQR(z));
      psi:=ARCTAN(((1+kk)/(1-ee+kk))*(z/(SQRT(SQR(x)+SQR(y)))));
    end;
  gettime(hr2,min2,sec2,hun2);
  time:=timediff(hr,min,sec,hun,hr2,min2,sec2,hun2);
END;{procedure weidirect86}

PROCEDURE weiiterate86(x,y,z:extended;VAR psi,h,time:extended);
{ref:Positioning With NAVSTAR, The Global Positioning System, by Ziqing Wei. An iterative
solution for the conversion of Cartesian coordinates into latitude and height.}
  VAR
    capN:extended;       {radius of curvature in the prime vertical}
    tanpsim1:extended;   {previous value of tangent of psi}
    tanpsi:extended;     {tangent of psi}
    count:INTEGER;
    ee:extended;         {the first eccentricity squared}
  BEGIN{procedure weiiterate86}
    gettime(hr,min,sec,hun);
    count:=0;
    if((x=0.0)and(y=0.0))then{at the pole}
      begin
```

```
        count:=count+1;
        psi:=90*pi/180;
        h:=z-b
      end
    else
      begin
        ee:=(sqr(a)-sqr(b))/sqr(a);
        tanpsi:=z/SQRT(SQR(x)+SQR(y));{zeroth iteration,p132}
        REPEAT{until desired accuracy is obtained}
          tanpsim1:=tanpsi;
(*************************************************************************
tanpsi:=z/((sqrt(sqr(x)+sqr(y))-(a*ee/SQRT(1+(1-ee)*
SQR(tanpsim1))));{eqn. A.4b, p132; fewer iterations = faster convergence}
*************************************************************************)
          tanpsi:=z/sqrt(sqr(x)+sqr(y))+(a*ee*tanpsim1/(sqrt(sqr(x)+sqr(y))*
          sqrt(1+(1-ee)*sqr(tanpsim1))));          {eqn. A.4a}
(*************************************************************************)
          count:=count+1;
        UNTIL(((tanpsi-tanpsim1)<=accuracy)OR(count>maxcount));
        psi:=arctan(tanpsi);
        capN:=a/SQRT(1-ee*SQR(SIN(psi)));
        h:=SQRT(SQR(x)+SQR(y))/COS(psi)-capN;
      end;
    gettime(hr2,min2,sec2,hun2);
    time:=timediff(hr,min,sec,hun,hr2,min2,sec2,hun2);
    WRITELN('weiiterate86 # of iterations: ',count);
    totalit:=totalit+count;
  END;{procedure weiiterate86}

procedure eissfeller86(x,y,z:extended;VAR psi,h,time:extended);
begin
(*************************************************************************
  gettime(hr,min,sec,hun);
  if((x=0.0)and(y=0.0))then{at the pole}
    begin
      count:=count+1;
      psi:=90*pi/180;
      h:=z-b
    end
  else
  begin
  eesqr:=(sqr(a)-sqr(b))/sqr(b);{the second eccentricity squared}
  p:=sqrt(sqr(x)+sqr(y));{2-5}
  tanphi:=z/p;{2-9b, tangent of spherical latitude}
  c:=sqr(a)/b;{2-2b}
  alphasqr:=sqr(a)/sqr(b);{2-8a}
  betasqr:=sqr(b)*sqr(eesqr);{2-8b}
  phip:=2*tanphi*(p*tanphi-z(*(alphasqr+sqr(tanphi));{3-6a}
  phit:=-2*(betasqr*tanphi-p*(p*tanphi-z)*(alphasqr+sqr(tanphi))-
```

```
  tanphi*sqr(p*tanphi-z));{3-6b}
  phiz:=-2*(p*tanphi-z)*(alphasqr+sqr(tanphi));{3-6c}
  phipp:=2*sqr(tanphi)*(alphasqr+sqr(tanphi));{3-6d}
  phipt:=2*(2*alphasqr*p*tanphi-alphasqr*z+4*p*power(tanphi,3,1)-3*
  sqr(tanphi)*z);{3-6e}
  phipz:=-2*tanphi*(alphasqr+sqr(tanphi));{3-6f}
  {3-6g}   phitt:=2*(alphasqr*sqr(p)-betasqr+6*sqr(p)*sqr(tanphi)-6*p*tanphi*z+sqr(z));
  phitz:=-2*(p*(alphasqr+sqr(tanphi))+2*tanphi*(p*tanphi-z));{3-6h}
  phizz:=2*(alphasqr+sqr(tanphi));{3-6i}
  phippp:=0;{3-6j}
  phippz:=0;{3-6k}
  phippt:=4*tanphi(alphasqr+2*sqr(tanphi));{3-6l}
  phiptz:=-2*(alphasqr+3*sqr(tanphi));{3-6m}
  phizzp:=0;{3-6n}
  phizzt:=4*tanphi;{3-6o}
  phizzz:=0;{3-6p}
  phittp:=4*(alphasqr*p+6*p*sqr(tanphi)-3*tanphi*z);{3-6q}
  phittz:=-4*(3*p*tanphi-z);{3-6r}
  phittt:=12*p*(2*p*tanphi-z);{3-6s}
  tp:=-phip/phit;{3-5a}
  tz:=-phiz/phit;{3-5b}
  tpp:=-(phipp+2*phipt*tp+phitt*sqr(tp));{3-5c???????????}
  tpz:=-(phipz+phipt*tz+phitt*tp*tz+phitz*tp);{3-5d????????}
  tzz:=-(phizz+2*phitz+phitt*sqr(tz);{3-5e}
  tppp:=-(phippp+3*phippt*tp+3*phiptt*sqr(tp)+3*phipt*tpp+phittt*
    sqr(tp)*tp+3*phittt*tp*tpp);{3-5f}
tppz:=-(phippz+phippt*tz+2*phiptz*tp+2*phiptt*tp*tz+phittz*sqr(tp)+
phittt*tz*sqr(tp)+2*phipt*tpz+phitt*(2*tp*tpz+tz*tpp)+phitz*tpp);{3-5g}
  end;
  gettime(hr2,min2,sec2,hun2);
   time:=timediff(hr,min,sec,hun,hr2,min2,sec2,hun2);
*************************************************************************)
end{eissfeller86};

procedure goad87(x,y,z:extended;var phi,ha,time:extended);
{method of computing phi and h using Newton's iteration method and setup as taught by Dr.
Goad at Ohio State University.}
var
  ee,pe,r,n,pa,za,delta_p,delta_z,delta_phi,delta_h:extended;
  count:integer;
begin{procedure goad}
  gettime(hr,min,sec,hun);
  count:=0;
  if((x=0.0)and(y=0.0))then{at the pole}
    begin
      count:=count+1;
      phi:=90*pi/180;
      ha:=z-b
    end
```

```
  else
    begin
      ee:=(sqr(a)-sqr(b))/sqr(a);
      pe:=sqrt(sqr(x)+sqr(y));
      phi:=arctan(z/pe);{initial approximation of phi}
      r:=sqrt(sqr(pe)+sqr(z));
      ha:=r-a*(1-f*sqr(sin(phi)));{initial approximation of height}
      repeat
        {compute approximate n, p, delta_p, and delta_z}
        n:=a/sqrt((1-ee*sqr(sin(phi))));
        pa:=(n+ha)*cos(phi);
        delta_p:=pe-pa;
        za:=(n*(1-ee)+ha)*sin(phi);
        delta_z:=z-za;
        {using approximate quantities, approximate delta_phi and delta_h}
        delta_phi:=((cos(phi)*delta_z)-(sin(phi)*delta_p))/(n+ha);
        delta_h:=(cos(phi)*delta_p)+(sin(phi)*delta_z);
        phi:=phi+delta_phi;{update phi and h for next iteration}
        ha:=ha+delta_h;
                              c o u n t : = c o u n t + 1
until(((delta_p<accuracy)and(delta_z<accuracy))or(count>maxcount));
    end;
  gettime(hr2,min2,sec2,hun2);
  time:=timediff(hr,min,sec,hun,hr2,min2,sec2,hun2);
  writeln('goad87 # of iterations: ',count);
  totalit:=totalit+count;
end{procedure goad87};

PROCEDURE morrison(x,y,z:extended;VAR phi,h,time:extended);
{ref:The Astronomical Journal,Volume 66, Number 1, February, 1961.
A direct method form obtaining geodetic coordinates of a point given the geocentric
equatorial polar coordinates.}
VAR
  aa2,aa4,aa6,aa8:extended;{coefficients}
  p:extended;              {distance in two dimensions}
  rho:extended;            {polar distance}
  phiprime:extended;       {geocentric latitude}
  ee:extended;             {the first eccentricity squared}
BEGIN{procedure morrison}
  gettime(hr,min,sec,hun);          {time of beginning of procedure}
  if((x=0.0)and(y=0.0))then{at the pole}
    begin
      phi:=90*pi/180;
      h:=z-b
    end
  else
  begin
    p:=SQRT(SQR(x)+SQR(y));
    rho:=SQRT(SQR(p)+SQR(z));
```

```
    ee:=(sqr(a)-sqr(b))/sqr(a);

    aa2:=(1/(1024*rho))*
    (512*ee+128*sqr(ee)+60*rpower(ee,3,1)+35*rpower(ee,4,1))+
    (1/(32*sqr(rho)))*(rpower(ee,3,1)+(rpower(ee,4,1)))-
    (3/(256*rpower(rho,3,1))*(4*rpower(ee,3,1)+3*rpower(ee,4,1)));

aa4:=(-1/(1024*rho))*(64*sqr(ee)+48*rpower(ee,3,1)+35*rpower(ee,4,1))+(1/(16*sqr(rho)))*(4
*sqr(ee)+2*rpower(ee,3,1)+rpower(ee,4,1))+
(15*rpower(ee,4,1)/(256*ee*sqrt(ee)))-rpower(ee,4,1)/(16*sqr(ee));

    aa6:=(3/(1024*rho))*(4*rpower(ee,3,1)+5*rpower(ee,4,1))-
    (3/(32*sqr(rho)))*(rpower(ee,3,1)+rpower(ee,4,1))+
    (35/(768*rpower(rho,3,1)))*(4*rpower(ee,3,1)+3*rpower(ee,4,1));

    aa8:=(rpower(ee,4,1)/2048)*
    (-5/rho+64/sqr(rho)-252/rpower(rho,3,1)+320/rpower(rho,4,1));

    phiprime:=arctan(z/p);{computing geocentric latitude}


phi:=phiprime+aa2*sin(2*phiprime)+aa4*sin(4*phiprime)+aa6*sin(6*phiprime)+aa8*sin(8*phipri
me);{eqn. 6,computing geodetic latitude}

    h:=rho*cos(phi-phiprime)-sqrt(1-ee*sqr(sin(phi)));
    {eqn. 9, geodetic height}
  end;
  gettime(hr2,min2,sec2,hun2);
  time:=timediff(hr,min,sec,hun,hr2,min2,sec2,hun2);
END;{procedure morrison}

PROCEDURE torge(x,y,z:extended;VAR phi,h,time:extended);
{ref:TBD. An iterative method for computation of geodetic coordinates.}
VAR
  i:integer;       {counter}
  phiold:extended;{temporary storage}
  n:extended;
  ee:extended;     {the first eccentricity squared}
  zz:extended;
BEGIN{procedure torge}
  gettime(hr,min,sec,hun);   {time of beginning of procedure}
  i:=0;
  IF((x=0.0)and(y=0.0))THEN{at the pole}
    begin
      i:=i+1;
      phi:=90*pi/180;
      h:=z-b
    end
  else
```

```
  begin
    ee:=(sqr(a)-sqr(b))/sqr(a);{3.12, p48}
    phi:=arctan(z/(sqrt(sqr(x)+sqr(y))));
    {first approximation for geodetic latitude, geocentric latitude}
    n:=a/sqrt((1-ee*sqr(sin(phi))));{3.26, p51}
    h:=((sqrt(sqr(x)+sqr(y)))/cos(phi))-n;{first approximation, 3.35, p52}
    repeat
      phiold:=phi;
      n:=a/sqrt((1-ee*sqr(sin(phi))));{3.26, p51}
      phi:=arctan(z/(sqrt(sqr(x)+sqr(y))*(1-ee*n/(n+h))));{3.35, p52}
      h:=sqrt(sqr(x)+sqr(y))/cos(phi)-n;{3.35, p52}
      i:=i+1
    UNTIL((abs(phi-phiold)<=accuracy)OR(i>maxcount));
  end;
  gettime(hr2,min2,sec2,hun2);
  time:=timediff(hr,min,sec,hun,hr2,min2,sec2,hun2);
  WRITELN('torge # of iterations: ',i);
  totalit:=totalit+i;
END;{procedure torge}

PROCEDURE geod(x,y,z:extended;VAR phi,h,time:extended);
{ref:Bull. Geod. 63(1989) pp. 50-56; "Accurate Algorithms To Transform Geocentric To Geodetic
Coordinates; Closed form method developed by Kazimierz M. Borkowski and translated from the
original Lahey FORTRAN code.}
VAR
  r:extended;     {equatorial component of position vector in cartesian coordinate system}

  p0:extended;    {intermediate equation}
  c:extended;     {     "           "   }
  psi:extended;   {parametric, or reduced, latitude}
  i:integer;      {counter}
BEGIN{procedure geod}
  gettime(hr,min,sec,hun);          {time beginning of procedure}   i:=0;
  IF((x=0.0)and(y=0.0))THEN{at the pole}
  begin
    i:=i+1;
    phi:=90*pi/180;
    h:=z-b
  end
  else
    begin
      r:=sqrt(sqr(x)+sqr(y));
      p0:=arctan(b*z/(a*r));
      c:=(sqr(a)-sqr(b))/sqrt(sqr(a*r)+sqr(b*z));
       psi:=p0+0.5*c*sin(2.0*p0)/(1.0-c*cos(2.0*p0));{calculating initial value}
psi:=psi-(sin(psi-p0)-0.5*c*sin(2.0*psi))/(cos(psi-p0)-c*cos(2.0*psi));
      phi:=ARCTAN(sin(psi)*a/(cos(psi)*b));
      h:=(r-a*cos(psi))*cos(phi)+(z-b*sin(psi))*sin(phi);
    end;
```

```
  gettime(hr2,min2,sec2,hun2);        {time end of procedure}
  time:=timediff(hr,min,sec,hun,hr2,min2,sec2,hun2);
END;{procedure geod}

procedure hm63(x,y,z:extended;var phi,h,time:extended);
{calculates geodetic coordinates(latitude, and height) given
cartesian coordinates(x,y,z), iterative method proposed by Hirvonen
and Moritz in 1963 and explained by Rapp in his notes, March 1984.}
var
  ww:extended;
  n:extended;{maximum radius of curvature for the reference ellipsoid, meters}
  phi1,phi2:extended;{successive iterations of latitude}
  i:integer;
  ee:extended;                {the first eccentricity squared}
begin
  gettime(hr,min,sec,hun);  {time beginning of procedure}
  i:=0;
  IF((x=0.0)and(y=0.0))THEN {at the pole}
  begin
    i:=i+1;
    phi:=90*pi/180;
    h:=z-b
  end
  else
    begin
      ee:=(sqr(a)-sqr(b))/sqr(a);
      phi:=arctan((z/(sqrt(sqr(x)+sqr(y))))*(1/(1-ee)));{eqn. 6.97}
      {first approximation for phi, assume height is zero}
      ww:=sqrt(1-ee*(sqr(sin(phi))));{eqn. 3.40}
      n:=a/ww;{eqn. 3.99}
      repeat{until difference in phi iteration less than accuracy constant}
        phi1:=phi;
        phi:=arctan((z+ee*n*sin(phi1))/sqrt(sqr(x)+sqr(y)));{eqn. 6.95}
        phi2:=phi;
        ww:=sqrt(1-ee*(sqr(sin(phi))));{eqn. 3.40}
        n:=a/ww;{eqn. 3.99}
        i:=i+1;
      UNTIL((ABS(phi1-phi2)<=accuracy)OR(i=maxcount));
      h:=((sqrt(sqr(x)+sqr(y)))/cos(phi))-n;{eqn. 6.98}
    end;
  gettime(hr2,min2,sec2,hun2);        {time end of procedure}
  time:=timediff(hr,min,sec,hun,hr2,min2,sec2,hun2);
  WRITELN('Hirvonen & Moritz # of iterations: ',i);
  totalit:=totalit+i;
end;{procedure hm63}

PROCEDURE nautiyal(x,y,z:extended;VAR phi,h,time:extended);
```

```
{An iterative algorithm to generate geodetic coordinates from Earth-centered Earth-fixed
coordinates that is free from convergence problems near the poles and near the equator.
Developed by Atul Nautiyal, Defence Research and Development Laboratory, Hyderabad, India.}
VAR
  aa,bb,cc,dd,ee,aa0,aa1,aa2,aa3,aa4,t,temp,ft,fprimet,tantheta,tansqrphi:
  extended;
  i:INTEGER;                                {counter}
BEGIN{procedure nautiyal}
  gettime(hr,min,sec,hun);
  i:=0;
  IF((x=0.0)and(y=0.0))THEN{at the pole}
    begin
      i:=i+1;
      phi:=90*pi/180;
      h:=z-b
    end
  else
    begin
      ee:=(sqr(a)-sqr(b))/sqr(a);{the first eccentricity squared}
      aa:=sqr(z/b);bb:=-ee/(1-ee);cc:=sqr(a/b);dd:=x/a;
      aa0:=sqr(bb);aa1:=2*bb*cc*dd;aa2:=aa+sqr(cc)*sqr(dd)-sqr(bb);
      aa3:=-2*bb*cc*dd;aa4:=-sqr(cc)*sqr(dd);
      tantheta:=z/sqrt(SQR(x)+SQR(y));
      t:=sqrt(1/(1+sqr(tantheta)/(1-ee)));{initial value of t}
      REPEAT{Newton-Raphson method}
        i:=i+1;temp:=t;
        ft:=aa0*rpower(t,4,1)+aa1*rpower(t,3,1)+aa2*sqr(t)+aa3*t+aa4;{eqn.12}
        fprimet:=4*aa0*rpower(t,3,1)+3*aa1*sqr(t)+2*aa2*t+aa3;
        t:=t-ft/fprimet;
      UNTIL(((t-temp)<=accuracy)OR(i=maxcount));
      tansqrphi:=(1-sqr(t))/((sqr(t)*(1-ee)));{eqn.17}
      phi:=arctan(sqrt(tansqrphi));
      h:=(sqrt(sqr(x)+sqr(y))-a*t)*sqrt(1+tansqrphi);{eqn.18}
    end;
  gettime(hr2,min2,sec2,hun2);
  time:=timediff(hr,min,sec,hun,hr2,min2,sec2,hun2);
  WRITELN('Nautiyal # of iterations: ',i);
  totalit:=totalit+i;
END;{procedure nautiyal}

procedure hedman(x,y,z:extended;var phi,h,time:extended);
{calculates geodetic coordinates(latitude, and height) given
cartesian coordinates(x,y,z), iterative method.}

(***************needs work**************)
var
  eee:extended;
  ee:extended;              {the first eccentricity squared}
  ff:extended;
```

```
  fphi:extended;
  fprimephi:extended;
  aa:extended;
  bb:extended;
  n:extended;
  i:integer;
  temp:extended;{added}
begin
  h:=0.0;i:=0;                              {setting initial values}
  gettime(hr,min,sec,hun);                  {time beginning of procedure}
  IF((x=0.0)and(y=0.0))THEN                 {at the pole}
    begin
      i:=i+1;
      phi:=90*pi/180;
      h:=z-b
    end
  else
  if(z=0)then                               {at the equator}
    begin
      i:=i+1;
      phi:=0;
      h:=sqrt(sqr(x)+sqr(y))-a
    end
  else
    begin
      ee:=(sqr(a)-sqr(b))/sqr(a);
{  phi:=arctan(sqr(a)*y/(sqr(b)*x));    first approximation for phi,eqn.2}
      phi:=arctan(z/sqrt(sqr(x)+sqr(y)));   {geocentric latitude, added}
      n:=a/sqrt((1-ee*sqr(sin(phi))));      {first approximation,eqn.10a}
      repeat{until fphi/fprimephi less than accuracy constant}
        i:=i+1;
        temp:=phi;{added}
        eee:=(n+h)*cos(phi);                {eqn.11}
        ff:=(n*(1-ee)+h)*sin(phi);          {eqn.12}
        phi:=arctan(sqr(a)*ff/(sqr(b)*eee));{eqn.14}
        n:=a/sqrt(1-ee*(sqr(sin(phi))));    {using latest value of phi}
        fphi:=n*ee-eee/cos(phi)+ff/sin(phi);{eqn.13a}
            fprimephi:=a*sqr(ee)*sin(phi)*cos(phi)/
(rpower((1-ee*sqr(sin(phi))),3,2))-eee*sin(phi)/sqr(cos(phi))-
        ff*cos(phi)/sqr(sin(phi));          {eqn.13b}
        phi:=phi-(fphi/fprimephi);          {eqn.15}
 (*     UNTIL(((fphi/fprimephi)<=accuracy)OR(i=maxcount));      *)
        until((abs(phi-temp)<=accuracy)or(i=maxcount));(*added*)
      eee:=(n+h)*cos(phi);                  {eqn.11}
      ff:=(n*(1-ee)+h)*sin(phi);            {eqn.12}
      bb:=x*(eee-x)+y*(ff-y);               {eqn.16b}
      h:=sqrt(sqr(eee-x)+sqr(ff-y));        {eqn.16a}
    end;
(*  if(bb<0.0)then h:=-h;  *)                  {see eqn 16c}
```

```
  gettime(hr2,min2,sec2,hun2);                {time end of procedure}
  time:=timediff(hr,min,sec,hun,hr2,min2,sec2,hun2);
  WRITELN('hedman # of iterations: ',i);
  totalit:=totalit+i;
end;{procedure hedman}

PROCEDURE romao87(x,y,z:extended;VAR phi,he,time:extended);
{A direct solution is given through a 4th degree equation}
VAR
   p:extended;                {distance in two dimensions}
rr,aone,ss,tt,ff,gg,jj,uu,vv,capd,capa,caps,wone,wtwo,beta0:extended;
  capn,muone,capu,capw,capv,capt,x0,y0:extended;
  ee:extended;     {the first eccentricity squared}
BEGIN{procedure romao87}
  gettime(hr,min,sec,hun);           {time of beginning of procedure}
  writeln('x: ',x,' y: ',y,' z: ',z);
  if((x=0)and(y=0))then{at the pole}
    begin
      phi:=90*pi/180;
      he:=z-b
    end
  else
  if(z=0)then{at the equator}
    begin
      phi:=0;
      he:=sqrt(sqr(x)+sqr(y))-a
    end
  else
    begin
      ee:=(sqr(a)-sqr(b))/sqr(a);
      p:=SQRT(SQR(x)+SQR(y));
      rr:=(2*sqr(p)-2*sqr(a)*sqr(ee)-rpower(z,4,1)*(1-ee))/
      (2*sqr(p)*(1-ee));
        ss:=(-z*sqr(p)-sqr(a)*sqr(ee)*z)/(p*sqr(p)*(1-ee));
tt:=(rpower(z,4,1)*(1-ee)+4*sqr(p)*sqr(z)-4*sqr(a)*sqr(ee)*sqr(z))/
              (16*rpower(p,4,1)*(1-ee));
ff:=(2*sqr(p)-2*sqr(a)*sqr(ee)-sqr(z)*(1-ee))/(2*sqr(p)*(1-ee));
gg:=(sqr(ff)/4)-(rpower(z,4,1)*(1-ee)+4*sqr(p)*sqr(z)-4*sqr(a)*
        sqr(ee)*sqr(z))/(16*rpower(p,4,1)*(1-ee));
jj:=-sqr(-z*sqr(p)-sqr(a)*sqr(ee)*z)/(8*sqr(rpower(p,3,1)*(1-ee)));
      uu:=gg-sqr(ff)/3;
      vv:=(2*rpower(ff,3,1)/27)-(ff*gg/3)+jj;
      capd:=abs(sqr(vv)/4+rpower(uu,3,1)/27);{absolute value taken}
      capa:=-vv/2;
      caps:=abs(capa+sqrt(capd));{absolute value taken}
      wone:=rpower(ss,1,3);
      wtwo:=-uu/(3*wone);
      beta0:=wone+wtwo;
      muone:=abs(beta0-ff/3);{absolute value taken}
```

```
      capu:=sqrt(2*nuone);
      capw:=2*ss/uu;
      capv:=-2*(rr+nuone);
      capt:=abs(capv-capw);{absolute value taken}
      x0:=(capu+capt)/2;
      aone:=-2*z/p;
      y0:=x0-aone/4;
      writeln('x0: ',x0,' aone: ',aone,' y0: ',y0);
      phi:=arctan(y0);
      capn:=a/sqrt(1-ee*sqr(sin(phi)));
      he:=(p-capn*cos(phi))/cos(phi);
    end;
  gettime(hr2,min2,sec2,hun2);
  time:=timediff(hr,min,sec,hun,hr2,min2,sec2,hun2);
END;{procedure romao87}

PROCEDURE bowring76(x,y,z:extended;VAR phi,h,time:extended);
{An iterative algorithm to generate geodetic coordinates; developed by Bowring and explained
by Rapp, p123}
VAR
  oldphi,beta,ff,capn,capm,eprimesqr,ee:extended;
  i:INTEGER;                                {counter}
BEGIN{procedure bowring76}
  gettime(hr,min,sec,hun);
  i:=0;
  IF((x=0.0)and(y=0.0))THEN{at the pole}
    begin
      i:=i+1;
      phi:=90*pi/180;
      h:=z-b
    end
  else
    begin
      ee:=(sqr(a)-sqr(b))/sqr(a);
      beta:=arctan((a/b)*(z/sqrt(sqr(x)+sqr(y))));{eqn.6.106}
      ff:=(a-b)/a;{eqn.3.3}
      eprimesqr:=(sqr(a)-sqr(b))/sqr(b);{eqn.3.5}
      phi:=arctan(z/sqrt(sqr(x)+sqr(y)));{initial value}
      REPEAT{Newton-Raphson method}
        i:=i+1;
        oldphi:=phi;
          phi:=arctan((z+eprimesqr*b*sqr(sin(beta))*sin(beta))/
(sqrt(sqr(x)+sqr(y))-a*ee*sqr(cos(beta))*cos(beta)));{eqn.6.105}
        beta:=arctan((1-ff)*tan(phi));{eqn.6.107}
      UNTIL((abs(phi-oldphi)<=accuracy)OR(i=maxcount));
      capm:=a*(1-ee)/rpower((1-ee*sqr(sin(phi))),3,2);{eqn.3.87}
      capn:=a/sqrt(1-ee*sqr(sin(phi)));{eqn.3.99}
      if(phi<=0.78)then h:=sqrt(sqr(x)+sqr(y))/cos(phi)-capn;{eqn.6.98}
      if(phi>0.78)then h:=z/sin(phi)-capn+ee*capn;{eqn.6.99}
```

```
   (*   h:=sqrt(sqr(sqrt(sqr(x)+sqr(y))-a*cos(beta))+sqr(z-b*sin(beta)));
      {eqn.6.108}    *)
    end;
  gettime(hr2,min2,sec2,hun2);
  time:=timediff(hr,min,sec,hun,hr2,min2,sec2,hun2);
  WRITELN('Bowring76 # of iterations: ',i);
  totalit:=totalit+i;
END;{procedure bowring76}

end{unit algoritm}.^Z
(********************END PROCEDURE "ALGORITM"***************)
(********************BEGIN MAIN PROGRAM*********************)

{$D+}                    {Debug information on}
{$R+}                    {Range checking on}
{$B+}                    {Boolean complete evaluation on}
{$F+}                    {Force far call}
{$S+}                    {Stack checking on}
{$I+}                    {I/O checking on}
{$N+}                    {8087 numeric coprocessor enable/disable} {$E+}
{Emulate enable/disable}
{$M 65520,0,655360}      {stack size,low heap limit,high heap limit:bytes}
{$O+}                    {overlay directive}

PROGRAM evaluate;        {by ROBERT W. VOLL}
{This program is designed to evaluate geodetic equations for efficiency and accuracy. It
was created using a Turbo PASCAL (Borland version 5.0) compiler for use by an IBM/PC and is
capable of 20 digit accuracy when used in conjunction with an 8087 math coprocessor.
Constants of the following ellipsoids are contained within the program:

1.WGS84
2.NAD27 - based on the Clarke 1866 Ellipsoid
3.Airy 1830
4.Bessel 1841
5.Clarke 1880 (modified)
6.Clarke 1880
7.Everest 1830
8.International 1924
9.Krassovski 1940
10.Mercury 1960
11.Modified Mercury 1968
12.Australian National
13.South America 1969
14.GRS67
15.WGS72
16.International Association of Geodesy 1975
17.International Association of Geodesy 1983
18.GRS80}
```

```
Uses
    algoritm,globals,Crt,Dos,printer;

PROCEDURE readuvwfile;
{Writes out contents of uvw.fil to screen or to printer}
VAR
  data:extended;i:INTEGER;
BEGIN
  CLRSCR;i:=0;RESET(uvwfile);
  writeln('          1) UVW.FIL to Screen');
  writeln('          2) UVW.FIL to Printer');
  readln(iii);
  if(iii=1)then
  begin
  WRITELN('uvw.fil(geocentric coordinates): ');
  WRITELN('            U                      V                      W');
  WHILE NOT EOF(uvwfile)DO
    BEGIN
      i:=i+1;READ(uvwfile,data);WRITE(data:23:15,' ');
      IF(i MOD 3 = 0)THEN WRITELN;
    END;
  end;
  if(iii=2)then
  begin
  WRITELN(lst,'uvw.fil(geocentric coordinates): ');
  WRITELN(lst,'            U                      V                      W');
  WHILE NOT EOF(uvwfile)DO
    BEGIN
      i:=i+1;READ(uvwfile,data);WRITE(lst,data:23:15,' ');
      IF(i MOD 3 = 0)THEN WRITELN(lst);
    END;
  end;
  CLOSE(uvwfile);
END{procedure readuvwfile};

PROCEDURE readllhfile;
{Writes out contents of llh.fil to screen or to printer}
VAR
  data:extended;i:INTEGER;
BEGIN
  CLRSCR;i:=0;RESET(llhfile);
  writeln('          1) LLH.FIL to Screen');
  writeln('          2) LLH.FIL to Printer');
  readln(iii);
  if(iii=1)then
  begin
  WRITELN('llh.fil(geodetic coordinates): ');
  WRITE('      Latitude(radians)       Longitude(radians)');
  writeln('      Height(meters)');WRITELN;
```

```
  WHILE NOT EOF(llhfile)DO
    BEGIN
      i:=i+1;READ(llhfile,data);WRITE(data:23:15,' ');
      IF(i MOD 3 = 0)THEN WRITELN;
    END;
  end;
  if(iii=2)then
  begin
  WRITELN(lst,'llh.fil(geodetic coordinates): ');
  WRITE(lst,'      Latitude(radians)       Longitude(radians)');
  writeln(lst,'      Height(meters)');WRITELN;
  WHILE NOT EOF(llhfile)DO
    BEGIN
      i:=i+1;READ(llhfile,data);WRITE(lst,data:23:15,' ');
      IF(i MOD 3 = 0)THEN WRITELN(lst);
    END;
  end;
  CLOSE(llhfile);
END{procedure readllhfile};

PROCEDURE filetype;
{used to determine if a file consists of geocentric or geodetic coordinates}
BEGIN
  geocentric_file:=FALSE;
  REPEAT{until input is correct}
    CLRSCR;WRITELN;WRITELN;
    WRITE('Is this a file of geocentric coordinates? y or n ');readln(choice);
  UNTIL(choice)IN ['Y','y','N','n'];
  write('Press any key to continue..');
  charactr:=readkey;
  IF(choice IN ['Y','y'])THEN geocentric_file:=TRUE;
END;{procedure filetype}

procedure caseout;
begin{procedure caseout}
  if(iii=1)then
  begin
  case j of
    1:WRITELN('gersten60 output:');
    2:WRITELN('baird64 output:');
    3:WRITELN('paul73 output:');
    4:WRITELN('heikkinen82 output:');
    5:WRITELN('lpash85A output:');
    6:WRITELN('bowring85 output:');
    7:WRITELN('weidirect86 output:');
    8:WRITELN('weiiterate86 output:');
    9:WRITELN('eissfeller86 output:');
    10:writeln('goad87 output:');
    11:writeln('bowring88 output:');
```

```
    12:writeln('morrison60 output:');
    13:writeln('torge output:');
    14:writeln('borkowski output:');
    15:writeln('Hirvonen & Moritz output:');
    16:writeln('nautiyal output:');
    17:writeln('hedman output:');
    18:writeln('romao87 output');
    19:writeln('bowring76 output');
  end;{case}
  end;{if}
  if(iii=2)then
  begin
  case j of
    1:WRITELN(lst,'gersten60 output:');
    2:WRITELN(lst,'baird64 output:');
    3:WRITELN(lst,'paul73 output:');
    4:WRITELN(lst,'heikkinen82 output:');
    5:WRITELN(lst,'lpash85A output:');
    6:WRITELN(lst,'bowring85 output:');
    7:WRITELN(lst,'weidirect86 output:');
    8:WRITELN(lst,'weiiterate86 output:');
    9:WRITELN(lst,'eissfeller86 output:');
    10:writeln(lst,'goad87 output:');
    11:writeln(lst,'bowring88 output:');
    12:writeln(lst,'morrison60 output:');
    13:writeln(lst,'torge output:');
    14:writeln(lst,'borkowski output:');
    15:writeln(lst,'Hirvonen & Moritz output:');
    16:writeln(lst,'nautiyal output:');
    17:writeln(lst,'hedman output:');
    18:writeln(lst,'romao87 output');
    19:writeln(lst,'bowring76 output');
  end;{case}
  end;{if}
end;{procedure caseout}

PROCEDURE readfile1(j:integer);
{called by procedure sequence1}
VAR
  data:string[23];i:integer;
BEGIN
  CLRSCR;i:=0;
  reset(file1a);
  writeln('         1) Output to Screen');
  writeln('         2) Output to Printer');
  readln(iii);
  caseout;
  if(iii=1)then
  begin
```

```
  WRITELN('   Latitude(radians)      Longitude(radians)      Height(meters)');
  WRITELN;
  WHILE NOT EOF(file1a) do
    BEGIN
      i:=i+1;
      READ(file1a,data);(*readln reads sequential files*)
      WRITE(data,' ');
      IF(i MOD 3 = 0)THEN WRITELN;
    END;
    end;{if}
  if(iii=2)then
  begin
  WRITELN(lst,'   Latitude(radians)      Longitude(radians)      Height(meters)');
  WRITELN(lst);
  WHILE NOT EOF(file1a) do
    BEGIN
      i:=i+1;
      READ(file1a,data);(*readln reads sequential files*)
      WRITE(lst,data,' ');
      IF(i MOD 3 = 0)THEN WRITELN(lst);
    END;
    end;{if}
    CLOSE(file1a);
END{procedure readfile1};

PROCEDURE readdiffile(j:integer);
VAR
  phidiff,htdiff:string[23];i:INTEGER;
BEGIN
  CLRSCR;i:=0;
  reset(file1b);{preparing to read from current difference file}
    if(iii=1)then
      begin
      case j of {selecting current difference file}
        1:WRITELN('gersten60 differences:');
        2:WRITELN('baird64 differences:');
        3:WRITELN('paul73 differences:');
        4:WRITELN('heikkinen82 differences:');
        5:WRITELN('lpash85A differences:');
        6:WRITELN('bowring85 differences:');
        7:WRITELN('weidirect86 differences:');
        8:WRITELN('weiiterate86 differences:');
        9:WRITELN('eissfeller86 differences:');
       10:writeln('goad87 differences:');
       11:writeln('bowring88 differences:');
       12:writeln('morrison60 differences:');
       13:writeln('torge differences:');
       14:writeln('borkowski differences:');
       15:writeln('Hirvonen & Moritz differences:');
```

```
    16:writeln('nautiyal differences:');
    17:writeln('hedman differences:');
    18:writeln('romao87 differences:');
    19:writeln('bowring76 differences:');
   end{case}
   end{if};
 if(iii=2)then
   begin
   case j of {selecting current difference file}
     1:WRITELN(lst,'gersten60 differences:');
     2:WRITELN(lst,'baird64 differences:');
     3:WRITELN(lst,'paul73 differences:');
     4:WRITELN(lst,'heikkinen82 differences:');
     5:WRITELN(lst,'lpash85A differences:');
     6:WRITELN(lst,'bowring85 differences:');
     7:WRITELN(lst,'weidirect86 differences:');
     8:WRITELN(lst,'weiiterate86 differences:');
     9:WRITELN(lst,'eissfeller86 differences:');
    10:writeln(lst,'goad87 differences:');
    11:writeln(lst,'bowring88 differences:');
    12:writeln(lst,'morrison60 differences:');
    13:writeln(lst,'torge differences:');
    14:writeln(lst,'borkowski differences:');
    15:writeln(lst,'Hirvonen & Moritz differences:');
    16:writeln(lst,'nautiyal differences:');
    17:writeln(lst,'hedman differences:');
    18:writeln(lst,'romao87 differences:');
    19:writeln(lst,'bowring76 differences:');
   end{case}
   end{if};
case j of {assigning current spreadsheet file}
  1:assign(file1a,'ger60.prn');
  2:assign(file1a,'baird64.prn');
  3:assign(file1a,'paul73.prn');
  4:assign(file1a,'heik82.prn');
  5:assign(file1a,'lpash85A.prn');
  6:assign(file1a,'bow85.prn');
  7:assign(file1a,'weidir86.prn');
  8:assign(file1a,'weiit86.prn');
  9:assign(file1a,'eisfel86.prn');
 10:assign(file1a,'goad87.prn');
 11:assign(file1a,'bowring88.prn');
 12:assign(file1a,'morrison.prn');
 13:assign(file1a,'torge.prn');
 14:assign(file1a,'geod.prn');
 15:assign(file1a,'hm63.prn');
 16:assign(file1a,'nautiyal.prn');
 17:assign(file1a,'hedman.prn');
 18:assign(file1a,'romao87.prn');
```

```
   19:assign(file1a,'bow76.prn');
  end{case};
  rewrite(file1a);
  {reading a sequential file, writing 2 columns to the screen, writing
  7 columns to spreadsheet export file}
  if(iii=1)then
  begin
  WRITELN('         Latitude Difference(meters          Height Difference(meters)');
  WRITELN;
  end;
  if(iii=2)then
  begin
  WRITELN(lst,'         Latitude Difference                Height Difference');
  WRITELN(lst);
  end;
  WHILE NOT EOF(file1b) DO
    BEGIN
      i:=i+1;
      READ(file1b,phidiff);read(file1b,htdiff);
      if(iii=1)then
      begin
      WRITE('       ',phidiff,'       ',htdiff);{writing to screen}
      WRITELN;
      end;
      if(iii=2)then
      begin
      WRITE(lst,'       ',phidiff,'       ',htdiff);{writing to printer}
      WRITELN(lst);
      end;
      write(file1a,phidiff);{writing phi differences to file1a}
      if(i mod 7 = 0)then writeln(file1a);
    END;
    writeln(file1a);
    CLOSE(file1b);close(file1a);
  reset(file1b);{preparing to read from file1b}
  append(file1a);{preparing to append height differences to file1a}
  i:=0;
  WHILE NOT EOF(file1b) DO
    BEGIN
      i:=i+1;
      READ(file1b,phidiff);read(file1b,htdiff);
      write(file1a,htdiff);{adding height differences to file1a}
      if(i mod 7 = 0)then writeln(file1a);
    END;
    CLOSE(file1b);close(file1a);
END{procedure readdiffile};

procedure choices;
begin
```

```
  WRITELN('                    1) Gersten (1960)(D)');
  WRITELN('                    2) Baird (1964)(I)');
  WRITELN('                    3) Paul (1973)(D)');
  WRITELN('                    4) Heikkinen (1982)(D)');
  WRITELN('                    5) Lupash (1985) algorithm "a"***(I)');
  WRITELN('                    6) Bowring (1985)(D)');
  WRITELN('                    7) Wei''s Direct Method (1986)(D)');
  WRITELN('                    8) Wei''s Iterative Method (1986)(I)');
  WRITELN('                    9) Eissfeller (1986)***(D)');
  writeln('                   10) Goad (1987)(I)');
  writeln('                   11) Modified Bowring (1988)(D)');
  writeln('                   12) Morrison (1960)***(D)');
  writeln('                   13) Torge (1975)(I)');
  writeln('                   14) Borkowski (1987)(D)');
  writeln('                   15) Hirvonen & Moritz (1963)(I)');
  writeln('                   16) Nautiyal (1986)(I)');
  writeln('                   17) Hedman (1969)***(I)');
  writeln('                   18) Romao (1987)***(D)');
  writeln('                   19) Bowring (1976)(I)');
  writeln('                   *** not yet complete.');
  writeln('                   (D): Direct Method, (I): Iterative Method')
end;{procedure choices}

PROCEDURE menu2;
{provides a menu for selecting a particular algorithm for calculations}
BEGIN{menu2}
    valid:=TRUE;CLRSCR;WRITELN;WRITELN;j:=0;jreal:=0.0;
    choices;writeln('                    20) Return To Main Menu');
    WRITELN;WRITELN;WRITE('Please enter a number between 1 and 20: ');
    checkinput(valid,jreal);
    IF(jreal>MAXINT)THEN jreal:=MAXINT;
    j:=TRUNC(jreal);
    if((j<1)or(j>20)or(not valid))then
      begin
        WRITELN;WRITELN;
        WRITELN('Only numbers between 1 and 20 are valid responses');
        write('Press any key to continue..');
        charactr:=readkey;
      end;
END;{menu2}

procedure caseuvw;
{selects proper algorithm}
begin{procedure caseuvw}
    case j of
      1:gersten60(u,v,w,latitude,height,time);
      2:baird64(u,v,w,latitude,height,time);
      3:paul73(u,v,w,latitude,height,time);
      4:heikkinen82(u,v,w,latitude,height,time);
```

```
      5:lpash85A(u,v,w,latitude,height,time);
      6:bowring85(u,v,w,latitude,height,time);
      7:weidirect86(u,v,w,latitude,height,time);
      8:weiiterate86(u,v,w,latitude,height,time);
      9:eissfeller86(u,v,w,latitude,height,time);
      10:goad87(u,v,w,latitude,height,time);
      11:bowring88(u,v,w,latitude,height,time);
      12:morrison(u,v,w,latitude,height,time);
      13:torge(u,v,w,latitude,height,time);
      14:geod(u,v,w,latitude,height,time);
      15:hm63(u,v,w,latitude,height,time);
      16:nautiyal(u,v,w,latitude,height,time);
      17:hedman(u,v,w,latitude,height,time);
      18:romao87(u,v,w,latitude,height,time);
      19:bowring76(u,v,w,latitude,height,time);
    end;{case}
end;{procedure caseuvw}

PROCEDURE sequence1;
{directs the sequence of operations for calculation of geodetic coordinates}
VAR
  lat,long,ht,latdiff,htdiff,ww,mm:extended;
BEGIN{procedure sequence1};
  textcolor(14);textbackground(0);
  choose;{selecting an ellipsoid}
  CLRSCR;uvwdat;{writing to uvw.fil and llh.fil}
  readllhfile;readuvwfile;{printing geodetic and geocentric data files}
  repeat {until j = 20} menu2;
  if(j<>20)then
  begin{if}
  RESET(llhfile);reset(uvwfile);time:=0.0;times:=0.0;totalit:=0;
  case j of
    1:assign(file1a,'ger60.fil');
    2:assign(file1a,'baird64.fil');
    3:assign(file1a,'paul73.fil');
    4:assign(file1a,'heik82.fil');
    5:assign(file1a,'lpash85A.fil');
    6:assign(file1a,'bow85.fil');
    7:assign(file1a,'weidir86.fil');
    8:assign(file1a,'weiit86.fil');
    9:assign(file1a,'eisfel86.fil');
   10:assign(file1a,'goad87.fil');
   11:assign(file1a,'bow88.fil');
   12:assign(file1a,'morrison.fil');
   13:assign(file1a,'torge.fil');
   14:assign(file1a,'geod.fil');
   15:assign(file1a,'hm63.fil');
   16:assign(file1a,'nautiyal.fil');
   17:assign(file1a,'hedman.fil');
```

```
   18:assign(file1a,'romao87.fil');
   19:assign(file1a,'bow76.fil');
  end{case};
  REWRITE(file1a);
  case j of
    1:assign(file1b,'ger60.dif');
    2:assign(file1b,'baird64.dif');
    3:assign(file1b,'paul73.dif');
    4:assign(file1b,'heik82.dif');
    5:assign(file1b,'lpash85A.dif');
    6:assign(file1b,'bow85.dif');
    7:assign(file1b,'weidir86.dif');
    8:assign(file1b,'weiit86.dif');
    9:assign(file1b,'eisfel86.dif');
   10:assign(file1b,'goad87.dif');
   11:assign(file1b,'bow88.dif');
   12:assign(file1b,'morrison.dif');
   13:assign(file1b,'torge.dif');
   14:assign(file1b,'geod.dif');
   15:assign(file1b,'hm63.dif');
   16:assign(file1b,'nautiyal.dif');
   17:assign(file1b,'hedman.dif');
   18:assign(file1b,'romao87.dif');
   19:assign(file1b,'bow76.dif');
  end{case};
  REWRITE(file1b);CLRSCR;
  if(iii=1)then
  begin
  WRITE('doing computations, filling *.fil, *.prn, and *.dif files...');
  end;{if}
  if(iii=2)then
  begin
  WRITE(lst,'doing computations, filling *.fil, *.prn, and *.dif files...');
  end;{if}
    WHILE((NOT EOF(uvwfile))AND(NOT EOF(llhfile))) DO
      BEGIN{while 2}
        READ(uvwfile,u);read(uvwfile,v);READ(uvwfile,w);
        READ(llhfile,lat);read(llhfile,long);READ(llhfile,ht);
        longitude:=lambda;{see function lambda}
        caseuvw;{routing variables to correct algorithm}
        subtract(latitude,height,lat,ht,latdiff,htdiff);{obtaining differences}
WRITE(file1a,latitude);write(file1a,longitude);
WRITE(file1a,height);
        {storing computations from chosen algorithm}
        ww:=sqrt(1-esqr*sqr(sin(latitude)));{intermediate term}
        mm:=a*(1-esqr)/rpower(ww,3,1);
        {computing principle radius of curvature in the plane of the meridian}
        latdiff:=latdiff*(mm+height);
        {converting latitude computation error from radians to meters}
```

```
      WRITE(file1b,latdiff);WRITE(file1b,htdiff);{storing differences}
      WRITE('.');times:=times+time;
    END;{while 2}
  CLRSCR;
  CLOSE(uvwfile);CLOSE(llhfile);
  close(file1a);close(file1b);
  readfile1(j);readdiffile(j);
  clrscr;
  caseout;{announcing chosen algorithm}
  if(iii=1)then
  begin
  writeln('Total computation time: ',times:10:2,' seconds.');
  if(j in[2,5,8,10,13,15,16,17,19])then
  begin
    writeln('Total # of iterations: ',totalit,'.');
    writeln('Average # of iterations: ',totalit/datacount);
    writeln('Average time per iteration: ',times/totalit,' seconds');
    totalit:=0;
  end;
  end;{if}
  if(iii=2)then
  begin
  writeln(lst,'Total computation time: ',times:10:2,' seconds.');
  if(j in[2,5,8,10,13,15,16,17,19])then
  begin
    writeln(lst,'Total # of iterations: ',totalit,'.');
    writeln(lst,'Average # of iterations: ',totalit/datacount);
    writeln(lst,'Average time per iteration: ',times/totalit,' seconds');
    totalit:=0;
  end;
  end;{if}
  write('Press any key to continue..');
  charactr:=readkey;
  end;{if}
  until(j=20)
END;{procedure sequence1}

PROCEDURE choose_a_file;{choose and print out data file}
VAR
  data:extended;i,ii:INTEGER;
  datastrg:string[23];
  dirinfo:searchrec;
BEGIN
  textcolor(0);textbackground(7);
  CLRSCR;i:=0;ii:=0;doserror:=0;
  REPEAT{until name of file is accepted by user}
    WRITELN('Data files currently on this disk: ');
    findfirst('*.fil',archive,dirinfo);
    while doserror = 0 do
```

```
      begin
        writeln(dirinfo.name);
        findnext(dirinfo);
      end;
    WRITE('Name of data file (ex: test.fil): ');
    READLN(fname);{accepts response typed by operator}WRITELN;
    WRITE('Is the correct name of your file ');
    WRITE(fname,' [y or n] ');READLN(choice);WRITELN;
    UNTIL(choice IN['y','Y']);clrscr;
    writeln('          1) Data file to Screen');
    writeln('          2) Data file to Printer');
    readln(iii);
      filetype;ASSIGN(file1a,fname);RESET(file1a);CLRSCR;
ASSIGN(file1b,'test.fil');REwrite(file1b);{*************debug*********}
    if(iii=1)then WRITELN(fname);if(iii=2)then WRITELN(lst,fname);
    IF(geocentric_file)THEN
      BEGIN
      if(iii=1)then
        begin
          WRITE('             U                          V');
          writeln('                          W');WRITELN;
        end;
      if(iii=2)then
        begin
          WRITE(lst,'             U                          V');
          writeln(lst,'                          W');WRITELN;
        end;
      END
      ELSE
      if(iii=1)then
        begin
          WRITE('      Latitude(radians)       Longitude(radians)');
          writeln('      Height(meters)');WRITELN;
        END;{if}
      if(iii=2)then
        begin
          WRITE(lst,'      Latitude(radians)       Longitude(radians)');
          writeln(lst,'      Height(meters)');WRITELN;
        end;
    WHILE NOT EOF(file1a)DO
      BEGIN
        i:=i+1;READ(file1a,datastrg);write(file1b,datastrg);
        if(iii=1)then WRITE(datastrg,' ');
        if(iii=2)then write(lst,datastrg,' ');
        IF(i MOD 3 = 0)THEN
          begin
            if(iii=1)then
              begin
                WRITELN;
```

```
              writeln(file1b)
            end;
          if(iii=2)then writeln(lst);
        end;
    END;
  CLOSE(file1a);close(file1b);
  write('Press any key to continue..');
  charactr:=readkey;
END;{procedure choose_a_file}

PROCEDURE sequence2;{called by procedure menu3}
BEGIN
  CLRSCR;{menu3;***************debug*****************}
  IF(k IN [1..2])THEN
    BEGIN
      caseout;
      IF(k=1)THEN
        BEGIN
          inuvw;{reading in coordinates};
          caseuvw;
          outllh;
          write('Press any key to continue..');
          charactr:=readkey;
        END;{if}
      IF(k=2)THEN
        BEGIN
          choose_a_file;{***************debug*******************}
           WRITELN('in sequence2, filename: ',fname);{*******debug******}
ASSIGN(file1a,fname);RESET(file1a);{*************debug*********}
          WHILE NOT EOF(file1a) DO
            BEGIN
                 READ(file1a,u);READ(file1a,v);READ(file1a,w);
WRITE('************************************************');
              WRITELN('*****************************');
              outuvw;
              caseuvw;
              longitude:=lambda;
              outllh;  WRITE('************************************************');
              WRITELN('*****************************');
              write('Press any key to continue..');
              charactr:=readkey;
            END;{while}
          CLOSE(file1a)
        END{if}
    END{if}
END;{sequence2}

PROCEDURE menu4;
{provides a menu for selecting a particular algorithm for calculations}
```

```
BEGIN{menu4}
  REPEAT{until input is valid}
    valid:=TRUE;CLRSCR;WRITELN;WRITELN;j:=0;jreal:=0.0;choices;
    WRITELN;WRITELN;WRITE('Please enter a number between 1 and 19: ');
    checkinput(valid,jreal);
    IF(jreal>MAXINT)THEN jreal:=MAXINT;
    j:=TRUNC(jreal);
    if((j<1) or (j>19) or (not valid))then
      begin
        WRITELN;WRITELN;
        WRITELN('Only numbers between 1 and 19 are valid responses');
        write('Press any key to continue..');
        charactr:=readkey;
      end
  UNTIL(j IN[1..19]);
END;{menu4}

procedure llh;{computes geodetic coordinates, called by menu}
begin
  textcolor(0);textbackground(2);
  choose;{selecting an ellipsoid}
  clrscr;inuvw;{read in geocentric coordinates}
  longitude:=lambda;{compute longitude}
  menu4;{selecting algorithm}
  caseuvw;{computing latitude and height}
  clrscr;
  WRITELN('    Input data: ');
  WRITELN('               u: ',u:20:15);
  WRITELN('               v: ',v:20:15);
  WRITELN('               w: ',w:20:15);
  writeln;writeln;outllh;{printing results}
  write('Press any key to continue..');
  charactr:=readkey;
end{procedure llh};

PROCEDURE menu3;{provides a menu for selecting method of input }
BEGIN{menu3}
  REPEAT{until k in[1..2]}
    valid:=TRUE;CLRSCR;WRITELN;WRITELN;k:=0;jreal:=0.0;
    WRITELN('                    1) Read in data from keyboard.');WRITELN;
    WRITELN('                    2) Read in data from a disk file.');WRITELN;
    WRITE('Please choose 1 or 2: ');checkinput(valid,jreal);
    IF(jreal>MAXINT)THEN jreal:=MAXINT;
    k:=TRUNC(jreal);
    if((k<1) or (k>2) or (not valid))then
      begin
        WRITELN;WRITELN;
        WRITELN('Only numbers between 1 and 2 are valid responses');
        write('Press any key to continue..');
```

```
        charactr:=readkey;
      end
      else
      begin
        CASE k OF
          1:k:=1;
          2:k:=2;
        END;{case}
      END{if}
  UNTIL(k IN[1..2]);
END;{menu3}
(*********************
PROCEDURE menu2;
{provides a menu for selecting a particular algorithm for calculations}
BEGIN{menu2}
  menu3;{***************debug*********************************}
  REPEAT{until "Return To Main Menu" selected}
    valid:=TRUE;CLRSCR;WRITELN;WRITELN;j:=0;jreal:=0.0;
    choices;WRITELN('                    20) Return To Main Menu');
    WRITELN;WRITELN;WRITE('Please enter a number between 1 and 20: ');
    checkinput(valid,jreal);
    IF
    (jreal>MAXINT)THEN jreal:=MAXINT;
    j:=TRUNC(jreal);
    if((j>20) or (j<1) or (not valid))then
      begin
        WRITELN;WRITELN;
        WRITELN('Only numbers between 1 and 20 are valid responses');
        write('Press any key to continue..');
        charactr:=readkey;
      end
      else
        begin
          caseuvw;
        end
  UNTIL(j=20)
END;{menu2}                          **************)

PROCEDURE filecreate;
{used by the operator to create a data file of geocentric or geodetic
coordinates}
VAR
  i:INTEGER;
  dirinfo:searchrec;
BEGIN{procedure filecreate}
  clrscr;
  textcolor(14);textbackground(3);
  i:=0;
  BEGIN
```

```
    REPEAT{until name of file is accepted by user}
      WRITELN('Data files currently on this disk: ');
      findfirst('*.fil',archive,dirinfo);
      while doserror = 0 do
        begin
          writeln(dirinfo.name);
          findnext(dirinfo);
        end;
      WRITE('Name of data file (ex: test.fil): ');
      READLN(fname);{accepts response typed by operator}WRITELN;
      WRITELN('The name of your file will be  ',fname);
      WRITELN('This procedure will erase any file with the same name!');
      WRITELN;WRITELN;WRITE('Continue? [y or n] ');readln(choice);WRITELN;
    until(choice IN['Y','y']);
    filetype;WRITELN;
    WRITE('************************************************');
    WRITELN('******************************');
{******receive coordinates as input from keyboard and fill disk data file*****}
    ASSIGN(infile,fname);REWRITE(infile);
    REPEAT{until input no longer desired}
      WRITELN;WRITELN;WRITELN('          Enter 999 at prompts to end input!');
      write('Press any key to continue..');
      charactr:=readkey;
      IF(geocentric_file)THEN
        BEGIN
          inuvw;{reading in geocentric coordinates};
          IF((998<u)AND(u<1000))THEN i:=999;
          IF(i<>999)THEN
          begin
            WRITE(infile,u);WRITE(infile,v);WRITE(infile,w)
          end
        END;
      IF(NOT geocentric_file)THEN
        BEGIN
          inllh;{reading in geodetic coordinates};
          IF((998<u)AND(u<1000))THEN i:=999;
          IF(i<>999)THEN
          begin  WRITE(infile,longitude);WRITE(infile,latitude);WRITE(infile,height)
          end
        END
    UNTIL(i=999);CLOSE(infile)
  END{if}
END;{procedure filecreate}

PROCEDURE datumconvert;{***this procedure is not completed***}
var
  i:integer;
  data:extended;
BEGIN
```

```
  textcolor(14);textbackground(10);
  writeln('***this procedure is not completed***');
  WRITELN('Choose the ellipsoid to convert from: ');delay(3000);
  whichone;{choosing datum constants}
  asign(a,b,finv,f,esqr,eprimesq);          {assigning datum constants}
  WRITELN('Input geodetic coordinates for datum conversion: ');DELAY(3000);
  menu3;{choosing method of data input}
  if(k=1)then {calculate geocentric coordinates, with keyboard input}
    begin
      inllh;{read in longitude, latitude, and height}
      uvwcalc;{calculate geocentric coordinates}
      outuvw  {print out geocentric coordinates}
    end;
  if(k=2)then {calculate geocentric coordinates, with input from file}
    begin
      repeat
        WRITE('Name of input file (ex: llh1.fil): ');
        READLN(fname1);{accepts response typed by operator}WRITELN;
        WRITE('Is the correct name of your file ');
        WRITE(fname1,' [y or n] ');readln(choice);WRITELN;
      UNTIL(choice IN['y','Y']);
      clrscr;
    fname2:='llh2.fil';
    ASSIGN(llhfile,fname1);assign(uvwfile,'uvw.fil');
    rewrite(uvwfile);
    reset(llhfile);
    WHILE(NOT EOF(llhfile)) DO
      BEGIN
           {read longitude, latitude, and height from llhfile}
READ(llhfile,latitude);read(llhfile,longitude);READ(llhfile,height);
        uvwcalc;{compute geocentric coordinates}
        {write geocentric coordinates to uvwfile}
        write(uvwfile,u);write(uvwfile,v);write(uvwfile,w);
      END;{while 2}
    end;
  WRITELN('Choose the ellipsoid to convert to: ');DELAY(3000);
  whichone;{choosing new datum constants}
  asign(a,b,finv,f,esqr,eprimesq);          {assigning new datum constants}
(*************  insert table of delta values here  **********************)
  if(k=1)then {calculate new geodetic coordinates from keyboard input}
    begin
      weidirect86(u,v,w,latitude,height,time);longitude:=lambda;
      (***** write explanation to user ***************)
      outllh;{writing output to screen}
      write('Press any key to continue..');
      charactr:=readkey;
    end;
  if(k=2)then
    begin
```

```
      ASSIGN(llhfile,fname2);
      reset(uvwfile);rewrite(llhfile);
      WHILE(NOT EOF(uvwfile)) DO
      BEGIN
        {read geocentric coordinates from uvwfile}
        READ(uvwfile,u);read(uvwfile,v);READ(uvwfile,w);
        weidirect86(u,v,w,latitude,height,time);longitude:=lambda;
              {write geodetic coordinates to llhfile}
write(llhfile,latitude);write(llhfile,longitude);write(llhfile,height);
      END;
    reset(llhfile);i:=0;
    WRITELN('   Latitude(radians)         Longitude(radians)         Height');
    WRITELN;
    WHILE NOT EOF(llhfile) do
      BEGIN
        i:=i+1;
        READ(llhfile,data);(*readln reads sequential files*)
        WRITE(data,' ');
        IF(i MOD 3 = 0)THEN WRITELN;
      END;
    reset(llhfile);i:=0;
    assign(file1a,'llh2.prn');
    rewrite(file1a);
    WHILE NOT EOF(llhfile) DO
      BEGIN
        i:=i+1;   READ(llhfile,latitude);read(llhfile,longitude);read(llhfile,height);
write(file1a,latitude);write(file1a,longitude);write(file1a,height);
        if(i mod 3 = 0)then writeln(file1a);
      END;
      writeln(file1a);
      close(file1a);CLOSE(uvwfile);CLOSE(llhfile);
      write(fname1,' coordinate data has been converted and is located');
      writeln(' in llh2.prn.');
      writeln('It may be imported into LOTUS 123 for analysis.');
      writeln;writeln;
      write('Press any key to continue..');
      charactr:=readkey;
    end;
END;{datumconvert***this procedure is not completed***}

procedure menu;
BEGIN{menu}
  REPEAT{until "End program" selected}
    textcolor(14);textbackground(1);
    valid:=TRUE;CLRSCR;WRITELN;WRITELN;j:=0;jreal:=0.0;
    WRITELN;WRITELN;
    WRITELN('                    1) Perform algorithm tests');
    WRITELN;
    WRITELN('                    2) Compute curvilinear geodetic coordinates');
```

```
    WRITELN;
    WRITELN('                    3) Compute geocentric cartesian coordinates');
    WRITELN;
    WRITELN('                    4) Change ellipsoid');
    WRITELN;
    WRITELN('                    5) Create a data file.');
    WRITELN;
    WRITELN('                    6) Print a file.');
    WRITELN;
    WRITELN('                    7) Datum conversion***not completed***');
    WRITELN;
    WRITELN('                    8) End program');
    WRITELN;WRITE('Please enter a number between 1 and 8: ');
    checkinput(valid,jreal);
    IF(jreal>MAXINT)THEN jreal:=MAXINT;
    j:=TRUNC(jreal);
    if((j<1) or (j>8) or (not valid))then
      begin
        WRITELN;WRITELN;
        WRITELN('Only numbers between 1 and 8 are valid responses');
        write('Press any key to continue..');
        charactr:=readkey;writeln;
      end
    else
      begin
        WRITELN;textbackground(0);
        CASE j OF
          1:sequence1;{directs the sequence of operations for a calculation}
          2:llh;{computing geodetic coordinates}
          3:uvw;{computing geocentric coordinates}
          4:choose;{selecting an ellipsoid}
          5:filecreate;
          6:choose_a_file;
          7:datumconvert;
          8:WRITELN('Ending...');
        END;{case}
      end
  UNTIL(j=8);
END;{menu}

BEGIN{main}
{  explain;explains operation of the program}
  fillarray;{filling arrays with constants and ellipsoid names}
  menu;
END{main}.
```

8.4 Appendix D, Data Set 1, llh.fil

llh.fil(geodetic coordinates):

Latitude(radians)	Longitude(radians)	Height(meters)
0.000000000000000	0.000000000000000	0.000000000000000
0.000000000000000	0.000000000000000	5000000.000000000000000
0.000000000000000	0.000000000000000	10000000.000000000000000
0.000000000000000	0.000000000000000	15000000.000000000000000
0.000000000000000	0.000000000000000	20000000.000000000000000
0.000000000000000	0.000000000000000	25000000.000000000000000
0.000000000000000	0.000000000000000	30000000.000000000000000
0.087266462599716	0.000000000000000	0.000000000000000
0.087266462599716	0.000000000000000	5000000.000000000000000
0.087266462599716	0.000000000000000	10000000.000000000000000
0.087266462599716	0.000000000000000	15000000.000000000000000
0.087266462599716	0.000000000000000	20000000.000000000000000
0.087266462599716	0.000000000000000	25000000.000000000000000
0.087266462599716	0.000000000000000	30000000.000000000000000
0.174532925199433	0.000000000000000	0.000000000000000
0.174532925199433	0.000000000000000	5000000.000000000000000
0.174532925199433	0.000000000000000	10000000.000000000000000
0.174532925199433	0.000000000000000	15000000.000000000000000
0.174532925199433	0.000000000000000	20000000.000000000000000
0.174532925199433	0.000000000000000	25000000.000000000000000
0.174532925199433	0.000000000000000	30000000.000000000000000
0.261799387799149	0.000000000000000	0.000000000000000
0.261799387799149	0.000000000000000	5000000.000000000000000
0.261799387799149	0.000000000000000	10000000.000000000000000
0.261799387799149	0.000000000000000	15000000.000000000000000
0.261799387799149	0.000000000000000	20000000.000000000000000
0.261799387799149	0.000000000000000	25000000.000000000000000
0.261799387799149	0.000000000000000	30000000.000000000000000
0.349065850398866	0.000000000000000	0.000000000000000

0.349065850398866	0.000000000000000	5000000.000000000000000
0.349065850398866	0.000000000000000	10000000.000000000000000
0.349065850398866	0.000000000000000	15000000.000000000000000
0.349065850398866	0.000000000000000	20000000.000000000000000
0.349065850398866	0.000000000000000	25000000.000000000000000
0.349065850398866	0.000000000000000	30000000.000000000000000
0.436332312998582	0.000000000000000	0.000000000000000
0.436332312998582	0.000000000000000	5000000.000000000000000
0.436332312998582	0.000000000000000	10000000.000000000000000
0.436332312998582	0.000000000000000	15000000.000000000000000
0.436332312998582	0.000000000000000	20000000.000000000000000
0.436332312998582	0.000000000000000	25000000.000000000000000
0.436332312998582	0.000000000000000	30000000.000000000000000
0.523598775598299	0.000000000000000	0.000000000000000
0.523598775598299	0.000000000000000	5000000.000000000000000
0.523598775598299	0.000000000000000	10000000.000000000000000
0.523598775598299	0.000000000000000	15000000.000000000000000
0.523598775598299	0.000000000000000	20000000.000000000000000
0.523598775598299	0.000000000000000	25000000.000000000000000
0.523598775598299	0.000000000000000	30000000.000000000000000
0.610865238198015	0.000000000000000	0.000000000000000
0.610865238198015	0.000000000000000	5000000.000000000000000
0.610865238198015	0.000000000000000	10000000.000000000000000
0.610865238198015	0.000000000000000	15000000.000000000000000
0.610865238198015	0.000000000000000	20000000.000000000000000
0.610865238198015	0.000000000000000	25000000.000000000000000
0.610865238198015	0.000000000000000	30000000.000000000000000
0.698131700797732	0.000000000000000	0.000000000000000
0.698131700797732	0.000000000000000	5000000.000000000000000

0.698131700797732	0.000000000000000	10000000.000000000000000
0.698131700797732	0.000000000000000	15000000.000000000000000
0.698131700797732	0.000000000000000	20000000.000000000000000
0.698131700797732	0.000000000000000	25000000.000000000000000
0.698131700797732	0.000000000000000	30000000.000000000000000
0.785398163397448	0.000000000000000	0.000000000000000
0.785398163397448	0.000000000000000	5000000.000000000000000
0.785398163397448	0.000000000000000	10000000.000000000000000
0.785398163397448	0.000000000000000	15000000.000000000000000
0.785398163397448	0.000000000000000	20000000.000000000000000
0.785398163397448	0.000000000000000	25000000.000000000000000
0.785398163397448	0.000000000000000	30000000.000000000000000
0.872664625997165	0.000000000000000	0.000000000000000
0.872664625997165	0.000000000000000	5000000.000000000000000
0.872664625997165	0.000000000000000	10000000.000000000000000
0.872664625997165	0.000000000000000	15000000.000000000000000
0.872664625997165	0.000000000000000	20000000.000000000000000
0.872664625997165	0.000000000000000	25000000.000000000000000
0.872664625997165	0.000000000000000	30000000.000000000000000
0.959931088596881	0.000000000000000	0.000000000000000
0.959931088596881	0.000000000000000	5000000.000000000000000
0.959931088596881	0.000000000000000	10000000.000000000000000
0.959931088596881	0.000000000000000	15000000.000000000000000
0.959931088596881	0.000000000000000	20000000.000000000000000
0.959931088596881	0.000000000000000	25000000.000000000000000
0.959931088596881	0.000000000000000	30000000.000000000000000
1.047197551196598	0.000000000000000	0.000000000000000
1.047197551196598	0.000000000000000	5000000.000000000000000
1.047197551196598	0.000000000000000	10000000.000000000000000
1.047197551196598	0.000000000000000	15000000.000000000000000
1.047197551196598	0.000000000000000	20000000.000000000000000

1.047197551196598	0.000000000000000	25000000.000000000000000
1.047197551196598	0.000000000000000	30000000.000000000000000
1.134464013796314	0.000000000000000	0.000000000000000
1.134464013796314	0.000000000000000	5000000.000000000000000
1.134464013796314	0.000000000000000	10000000.000000000000000
1.134464013796314	0.000000000000000	15000000.000000000000000
1.134464013796314	0.000000000000000	20000000.000000000000000
1.134464013796314	0.000000000000000	25000000.000000000000000
1.134464013796314	0.000000000000000	30000000.000000000000000
1.221730476396031	0.000000000000000	0.000000000000000
1.221730476396031	0.000000000000000	5000000.000000000000000
1.221730476396031	0.000000000000000	10000000.000000000000000
1.221730476396031	0.000000000000000	15000000.000000000000000
1.221730476396031	0.000000000000000	20000000.000000000000000
1.221730476396031	0.000000000000000	25000000.000000000000000
1.221730476396031	0.000000000000000	30000000.000000000000000
1.308996938995747	0.000000000000000	0.000000000000000
1.308996938995747	0.000000000000000	5000000.000000000000000
1.308996938995747	0.000000000000000	10000000.000000000000000
1.308996938995747	0.000000000000000	15000000.000000000000000
1.308996938995747	0.000000000000000	20000000.000000000000000
1.308996938995747	0.000000000000000	25000000.000000000000000
1.308996938995747	0.000000000000000	30000000.000000000000000
1.396263401595464	0.000000000000000	0.000000000000000
1.396263401595464	0.000000000000000	5000000.000000000000000
1.396263401595464	0.000000000000000	10000000.000000000000000
1.396263401595464	0.000000000000000	15000000.000000000000000
1.396263401595464	0.000000000000000	20000000.000000000000000
1.396263401595464	0.000000000000000	25000000.000000000000000

1.396263401595464	0.000000000000000	30000000.000000000000000
1.483529864195180	0.000000000000000	0.000000000000000
1.483529864195180	0.000000000000000	5000000.000000000000000
1.483529864195180	0.000000000000000	10000000.000000000000000
1.483529864195180	0.000000000000000	15000000.000000000000000
1.483529864195180	0.000000000000000	20000000.000000000000000
1.483529864195180	0.000000000000000	25000000.000000000000000
1.483529864195180	0.000000000000000	30000000.000000000000000
1.570796326794897	0.000000000000000	0.000000000000000
1.570796326794897	0.000000000000000	5000000.000000000000000
1.570796326794897	0.000000000000000	10000000.000000000000000
1.570796326794897	0.000000000000000	15000000.000000000000000
1.570796326794897	0.000000000000000	20000000.000000000000000
1.570796326794897	0.000000000000000	25000000.000000000000000
1.570796326794897	0.000000000000000	30000000.000000000000000

8.5 Appendix E, Data Set 2, uvw.fil

uvw.fil(geocentric coordinates):

U	V	W
6378137.000000000000000	0.000000000000000	0.000000000000000
11378137.000000000000000	0.000000000000000	0.000000000000000
16378137.000000000000000	0.000000000000000	0.000000000000000
21378137.000000000000000	0.000000000000000	0.000000000000000
26378137.000000000000000	0.000000000000000	0.000000000000000
31378137.000000000000000	0.000000000000000	0.000000000000000
36378137.000000000000000	0.000000000000000	0.000000000000000
6354027.820562848720000	0.000000000000000	552183.960009627218000
11335001.311021576400000	0.000000000000000	987962.673747918085000
16315974.801480304000000	0.000000000000000	1423741.387486208950000
21296948.291939031700000	0.000000000000000	1859520.101224499820000
26277921.782397759400000	0.000000000000000	2295298.814962790690000
31258895.272856487000000	0.000000000000000	2731077.528701081560000
36239868.763315214700000	0.000000000000000	3166856.242439372420000
6281872.829606556700000	0.000000000000000	1100248.547699618610000
11205911.594667597000000	0.000000000000000	1968489.436034270360000
16129950.359728637300000	0.000000000000000	2836730.324368922100000
21053989.124789677600000	0.000000000000000	3704971.212703573850000
25978027.889850717900000	0.000000000000000	4573212.101038225590000
30902066.654911758200000	0.000000000000000	5441452.989372877330000
35826105.419972798500000	0.000000000000000	6309693.877707529080000
6162189.088045102610000	0.000000000000000	1640100.140143601350000
10991818.219490444000000	0.000000000000000	2934195.365656205160000
15821447.350935785500000	0.000000000000000	4228290.591168808980000
20651076.482381126900000	0.000000000000000	5522385.816681412790000
25480705.613826468300000	0.000000000000000	6816481.042194016600000
30310334.745271809800000	0.000000000000000	8110576.267706620410000
35139963.876717151200000	0.000000000000000	9404671.493219224220000
5995836.383907842950000	0.000000000000000	2167696.787761423030000

10694299.487837384900000	0.000000000000000	3877797.504389766690000
15392762.591766926800000	0.000000000000000	5587898.221018110360000
20091225.695696468700000	0.000000000000000	7297998.937646454020000
24789688.799626010600000	0.000000000000000	9008099.654274797690000
29488151.903555552500000	0.000000000000000	10718200.370903141400000
34186615.007485094500000	0.000000000000000	12428301.087531485000000
5784014.114742660200000	0.000000000000000	2679074.462877277760000
10315553.049925910000000	0.000000000000000	4792165.771580774950000
14847091.985109159800000	0.000000000000000	6905257.080284272130000
19378630.920292409700000	0.000000000000000	9018348.388987769310000
23910169.855475659500000	0.000000000000000	11131439.697691266500000
28441708.790658909300000	0.000000000000000	13244531.006[illegible]94763700000
32973247.725842159100000	0.000000000000000	15357622.315[illegible]98260800000
5528256.639315511520000	0.000000000000000	3170373.735292082270000
9858383.658237704750000	0.000000000000000	5670373.735292082270000
14188510.677159898000000	0.000000000000000	8170373.735292082270000
18518637.696082091200000	0.000000000000000	10670373.735292082300000
22848764.715004284500000	0.000000000000000	13170373.735292082300000
27178891.733926477700000	0.000000000000000	15670373.735292082300000
31509018.752848670900000	0.000000000000000	18170373.735292082300000
5230426.840228606140000	0.000000000000000	3637866.909277764230000
9326187.061673565090000	0.000000000000000	6505749.091032994710000
13421947.283118524000000	0.000000000000000	9373631.272788225190000
17517707.504563483000000	0.000000000000000	12241513.454543455700000
21613467.726008441900000	0.000000000000000	15109395.636298686100000
25709227.947453400900000	0.000000000000000	17977277.818053916600000
29804988.168898359800000	0.000000000000000	20845159.999809147100000
4892707.600105896640000	0.000000000000000	4077985.572093558440000
8722929.815700786810000	0.000000000000000	7291923.620526255070000

12553152.031295677000000	0.000000000000000	10505861.668958951700000
16383374.246890567200000	0.000000000000000	13719799.717391648300000
20213596.462485457300000	0.000000000000000	16933737.765824345000000
24043818.678080347500000	0.000000000000000	20147675.814257041600000
27874040.893675237700000	0.000000000000000	23361613.862689738200000
4517590.878886053760000	0.000000000000000	4487348.408754800150000
8053124.784818791380000	0.000000000000000	8022882.314687537770000
11588658.690751529000000	0.000000000000000	11558416.220620275400000
15124192.596684266600000	0.000000000000000	15093950.126553013000000
18659726.502617004200000	0.000000000000000	18629484.032485750600000
22195260.408549741900000	0.000000000000000	22165017.938418488300000
25730794.314482479500000	0.000000000000000	25700551.844351225900000
4107864.091246423210000	0.000000000000000	4862789.037592982040000
7321802.139679119840000	0.000000000000000	8693011.253187872220000
10535740.188111816500000	0.000000000000000	12523233.468782762400000
13749678.236544513100000	0.000000000000000	16353455.684377652600000
16963616.284977209700000	0.000000000000000	20183677.899972542700000
20177554.333409906400000	0.000000000000000	24013900.115567432900000
23391492.381842603000000	0.000000000000000	27844122.331162323100000
3666593.522414654920000	0.000000000000000	5201383.523088155940000
6534475.704169885400000	0.000000000000000	9297143.744533114890000
9402357.885925115880000	0.000000000000000	13392903.965978073800000
12270240.067680346400000	0.000000000000000	17488664.187423032800000
15138122.249435576800000	0.000000000000000	21584424.408867991700000
18006004.431190807300000	0.000000000000000	25680184.630312950700000
20873886.612946037800000	0.000000000000000	29775944.851757909600000
3197104.586963421220000	0.000000000000000	5500477.133825145300000
5697104.586963421220000	0.000000000000000	9830604.152747338530000
8197104.586963421220000	0.000000000000000	14160731.171669531800000
10697104.586963421200000	0.000000000000000	18490858.190591725000000
13197104.586963421200000	0.000000000000000	22820985.209513918200000

VITA

Robert W. Voll, born on October 30, 1943, retired from the U.S. Air Force in 1981 as a Master Sergeant. During his 20 year Air Force career, all of which was served in the field of Meteorology, he was responsible for many studies of weather forecasting problems mainly in the Midwest and the Orient. In 1978 he obtained an Associate in Applied Science Degree in Weather Forecasting from The Community College of The Air Force.

He earned a Bachelor of Science Degree in Geology from Indiana University in 1984.

In 1985 Mr. Voll was employed as a Cartographer by the Defense Mapping Agency, Hydrographic/Topographic Center, Louisville, Ky Field Office. He is presently employed as a Physical Scientist by the Defense Mapping Agency, Systems Center, Reston, Va.