NPS-MA-95-006

NAVAL POSTGRADUATE SCHOOL
Monterey, California

FAST INTERPOLATION FOR GLOBAL POSITIONING SYSTEM (GPS) SATELLITE ORBITS

by

C.P. Sagovac
D.A. Danielson
J.R. Clynch
B. Neta

August 1995

Prepared for: Naval Postgraduate School
Monterey, CA 93943-5000

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943

Rear Admiral T.A. Mercer
Superintendent

Richard Elster
Provost

This report was prepared in conjunction with research conducted for the Naval Postgraduate School and funded by the NISE West Coast Division.

This report was prepared by:

C. P. Sagovac

D. A. Danielson
Professor of Mathematics

J.R. Clynch
Professor of Oceanography

B. Neta
Professor of Mathematics

Accesion For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

Reviewed by:

RICHARD FRANKE
Chairman

Released by:

PAUL J. MARTO
Dean of Research

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 11, 1995	July-August 1995

4. TITLE AND SUBTITLE
Fast Interpolation for Global Positioning System (GPS) Satellite Orbits

5. FUNDING NUMBERS

6. AUTHOR(S)
C. P. Sagovac D. A. Danielson
J. R. Clynch Beny Neta

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER
NPS-MA-95-006

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
NISE West Coast Division
4297 Pacific Highway
San Diego, CA 92110

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The Views expressed in this report are those of the authors and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

In this report, we discuss and compare several methods for polynomial interpolation of Global Positioning System ephemeris data.

14. SUBJECT TERMS
Comparison of several methods for the polynomial interpolation

15. NUMBER OF PAGES
31

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	

Fast Interpolation for Global Positioning System (GPS) Satellite Orbits

C. P. Sagovac
819 South Charles Street
Baltimore, MD 21230-3938

D. A. Danielson
Naval Postgraduate School
Department of Mathematics
Code MA/Dd
Monterey, CA 93943

J. R. Clynch
Naval Postgraduate School
Department of Oceanography
Code OC/Cl
Monterey, CA 93943

Beny Neta
Naval Postgraduate School
Department of Mathematics
Code MA/Nd
Monterey, CA 93943

August 29, 1995

Abstract

In this report, we discuss and compare several methods for polynomial interpolation of Global Positioning System ephemeris data.

Intoduction

The problem of interpolating Global Positioning System (GPS) ephemeris data is an important aspect of GPS. Given that a high accuracy ($<$ 1m), high precision (1 cm) orbit can be generated though the use of dense observations and special integrations, it is necessary to interpolate these ephemeris at high accuracy to utilize these orbits.

These high accuracy orbits are produced by several organizations (DMA, NGS, JPL, several Universities) and are widely available. An ephemeris typically consists of satellite positions at evenly spaced times over a week. Most ephemeris are given at 900 sec (15 min) time steps although the NGS ones are at 1200 sec (20 min). The GPS satellites are in 12 hr circular orbits making 900 sec ephemeris steps 7.5 deg of arc.

The typical user collects GPS data at intervals from 1 sec to 30 sec and needs to find the satellite position at the times of that data. The times needed are really not the evenly spaced received times, but the transmit times that are about 60 msec before reception. A precise value for this propagation delay is not known until the solution process in partially done. Therefore usually one needs to find a cluster of satellite positions a few msec from a nominal evenly spaced interval.

In the past the typical technique used by the DMA [Malys, 1989], NGS [Remondi, 1991], JPL [Watkins, 1995] and others is a Lagrange interpolation. The orders vary from 8^{th} to 11^{th}. This approach directly computes the value of the function (the three Cartesian Earth centered earth fixed coordinates) from the unique polynomial going through the data points. The coefficients are not found, and finding them may introduce errors [Press et al, 1992]. Several evaluations of the accuracy of this method [Remondi, 1991, Smith and Curtis, 1983] have been made. It is generally found that an 8^{th} order Lagrangian interpolation using 900 sec data with the unknown in the center of the points gives values that compare with numerical integration at the 1 cm level.

The problem addressed here is to find if a more efficient numerical method that achieves the same accuracy can be used. This is motivated by the movement of processing from mainframes to PC's (486's and above).

Several aspects unique to the GPS satellites make this problem of interpolating the data different from the general problem of interpolation. Though we are interpolating GPS satellite orbital position data, it may be that the methods here are applicable to a broader class of problems. Where possible, we intend to take full advantage of the special geometry of the GPS satellite orbits.

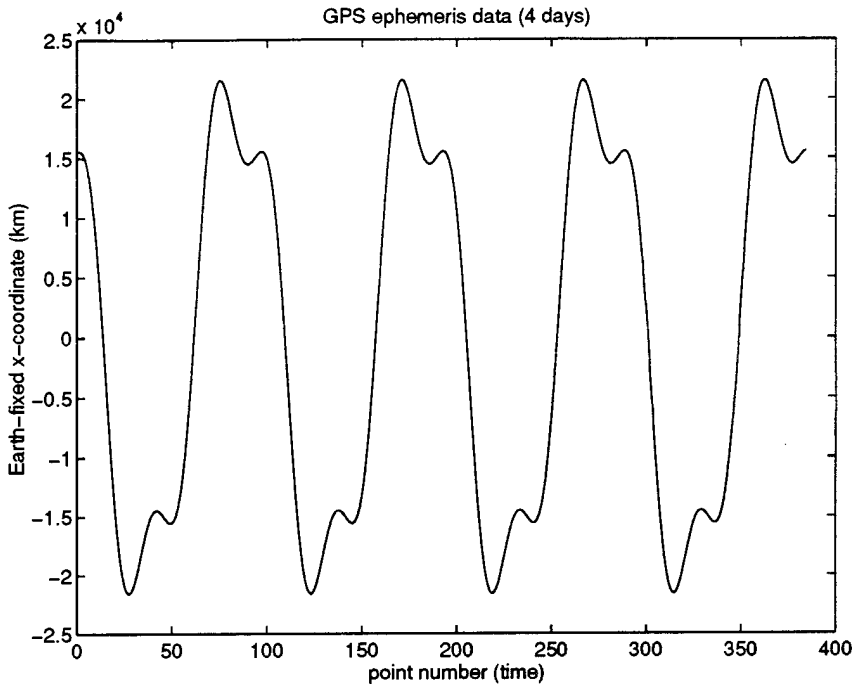

Figure (1) – 4 day x-coordinate GPS ephemeris data at 900 second intervals

A typical precise ephemeris orbit is supplied over an interval of eight days. Each ephemeris overlaps 1 day at each end with another ephemeris. It consists of position data which is Earth-centered, Earth-fixed Cartesian coordinates given every 900 seconds (t, x, y, z). We will not include velocity data which may in some cases be available. A plot of the data shows that it is "almost" periodic with a period of 24 hours or 96 intervals of 900-second each; Of course the data would be periodic in inertial coordinates. The data is given in a rotating frame (Earth centered - Earth fixed) however. This is done to place several subtle effects, such as polar motion, into the ephemeris generation problem. The user then does not have to have access to current data for geophysical effects to compute an Earth fixed position solution. Figure (1) is a plot of x-coordinate data (in kilometers) over a four day period. Note that the plot is with respect to the node (point) number which ranges from 1 to 384. It is convenient to use node numbers since we may choose to map the interval of interest into different subintervals. For instance, the Lagrange interpolation method maps the interval of interest into the interval $[-1, 1)$ while the trigonometric polynomial interpolation method maps the interval of interest into the interval $[0, 2\pi)$. Later we will use the node numbers to compare the residuals for different methods.

There are two lengths of computation time in which we are interested for this problem. One time length is the generation time of the function (interpolant) which interpolates the evenly-spaced data. The other is the time needed to evaluate the interpolant at a point. For polynomial interpolation these might be the calculation of the coefficients of the polynomial interpolating the data and the time needed to evaluate that polynomial, once generated, at a particular point. As we shall see, however, there are clever ways in which to minimize the amount of work done in generating the desired data, and the two times might not be easily distinguished in these cases. In recommending a particular technique for interpolation it is

important to know whether the interpolant will be evaluated once (or just a few times) on an interval or if it will be evaluated many times throughout the interval. With a cluster of times to be interpolated on an interval, the cost of generating the interpolant should be amortized over the set of times. This means that the time needed to generate the interpolant (a one-time cost) would not be as much a concern as the time needed to evaluate the interpolant at each of the desired times (a recurring expense). On the other hand, if one or a very few points were to be calculated on a given interval, the time needed to generate the interpolant would probably be more significant a concern than the time needed to evaluate it at the desired times since in general the generation of an interpolant is much costlier (in time) than its evaluation. This is a similar argument as one finds in deciding whether to use Gaussian elimination or LU factorization in solving systems of linear equations.

One method of polynomial interpolation involves performing the interpolation at a point without actually calculating the coefficients of the interpolating polynomial. This involves many less operations than the evaluation of a polynomial of degree n at a particular point. This is a common technique currently employed. If we do not explicitly calculate the interpolant, however, we will in general still need to calculate some quantities prior to interpolation. For instance, we shall see that divided differences would need to be available prior to using the nested multiplication algorithm used in evaluating the Newton form of the interpolating polynomial.

Since ephemeris data is generated for an eight day time period, we have the opportunity to "front load" our work at the time of ephemeris receipt. By calculating needed data in advance we should be able to shorten the real time operation count. Thus we will be more concerned with the rapid evaluation of interpolants for specific times than their rapid generation. Of course in some cases the times of evaluation and generation will be closely related, as mentioned above. In others, they will be quite different and our hope will be to shift as much of the work as possible to the generation of efficient interpolants so as to allow the rapid evaluation of those interpolants.

In summary, we will describe the following methods:

- Lagrange polynomial interpolation
- Newton's divided difference interpolation polynomial
- Difference Tables
- Cubic Spline interpolation
- Trigonometric polynomial interpolation
- Tshebyshev polynomial interpolation

We will describe the advantages and disadvantages of each of these methods for the problem of interest, namely for the efficient interpolation at clusters of points.

Lagrange Polynomial Interpolation

Before we begin our investigation, it is necessary to describe the method which is currently being used. Simply put, given the $n+1$ ephemeris values $f(t_0), f(t_1), \ldots, f(t_n)$ at the distinct times $t_0, t_1, \ldots, t_n$, there exists a unique interpolating polynomial p_n satisfying

$$p_n(t_i) = f(t_i), \qquad i = 0, 1, \ldots, n$$

This polynomial can be written in the form (called the Lagrange interpolation polynomial)

$$p_n(t) = \sum_{i=0}^{n} f(t_i)\, l_i(t) \tag{1}$$

where

$$l_i(t) = \frac{(t-t_0)(t-t_1)\cdots(t-t_{i-1})(t-t_{i+1})\cdots(t-t_n)}{(t_i-t_0)(t_i-t_1)\cdots(t_i-t_{i-1})(t_i-t_{i+1})\cdots(t_i-t_n)}$$

The eleventh order Lagrange method uses twelve data points to generate an eleventh order polynomial according to equation (1). This polynomial can then be evaluated at desired times within the interval of interest. The error $R_n(t)$ in using the Lagrange interpolant $p_n(t)$ to estimate the function $f(t)$ (having at least $n+1$ derivatives throughout the open interval) at some point t can be written [Buchanan and Turner, 1992]:

$$R_n(t) = f(t) - p_n(t) = \frac{L_{n+1}(t)\, f^{(n+1)}(\xi)}{(n+1)!}$$

where ξ is some point in the interval $[t_0, t_n]$ and

$$L_{n+1}(t) = \prod_{i=0}^{n}(t - t_i)$$

One difficulty in implementing high degree polynomial interpolation routines of any kind is the fact that the error between the interpolating polynomial and the data or function being interpolated grows rapidly near the endpoints of the interval over which the interpolation is being performed. For this reason the eleventh order Lagrange method is overlapped as successive intervals are chosen within the ephemeris (we call this walk-along interpolation). Due to the high accuracy requirements, only the center subinterval is interpolated for each Lagrange polynomial which is generated. Whereas the initial interval spans points one through twelve, the second interval spans points two through thirteen in order to provide the highest degree of accuracy. The first eleventh order Lagrange polynomial would then be used for times between points six and seven, while the second polynomial would be valid for times between points seven and eight. The numerical accuracy of this method has been verified to the 1 cm level for the data we are interpolating [Remondi, 1991].

Difficulties arise in that the process of creating and evaluating the resulting eleventh order polynomials is computationally expensive. The cost of evaluating the Lagrange form at a point is provided by de Boor [1978]. It is

$$(2n-2)\,A + (n-2)\,M + (n-1)\,D$$

for each of the $n+1$ numbers $l_i(t)$, where A denotes an addition or subtraction and M and D denote multiplication and division, respectively. Forming equation (1) then takes another

$$(n-1)\,A + n\,M$$

operations, leaving the total count per component of the position vector at

$$(3n-3)\,A + (2n-2)\,M + (n-1)\,D$$

This is the number of operations per component in the implementation of Lagrange interpolation. A simple modification of the algorithm would reduce the amount of work to

$$(2n-1)\,A + n\,M + n\,D$$

operations (see de Boor [1978]). It consists of first forming the quantity

$$Y_i = \frac{f(t_i)}{\prod_{j \neq i}(t_i - t_j)}, \qquad i = 1, \ldots, n$$

Afterwards, $p_n(t)$ is calculated through

$$\phi_n(t) = \prod_{i=1}^{n} (t - t_i)$$

$$p_n(t) = \phi_n(t) \sum_{i=1}^{n} Y_i/(t - t_i)$$

This method is somewhat faster than the method currently being used and would be easy to implement (see program lagrange in Appendix A). In addition, no loss of accuracy would occur in its implementation. If the time t is very close to one of the interpolating points t_i, one must be careful in computing $p_n(t)$ because of the division of Y_i by a very small number.

Newton's Divided Difference Interpolation Polynomial

A more efficient means by which we may form the interpolating polynomial is through divided differences. We follow the developments given in Buchanan and Turner [1992] and de Boor [1978]. Suppose we have $n+1$ distinct interpolation points $t_0, t_1, \ldots, t_n$. Define the *zeroth divided difference* at t_i by $f[t_i] = f(t_i)$. The *first divided difference* at t_i, t_j is defined by

$$a_0 = f[t_i, t_j] = \frac{f[t_i] - f[t_j]}{t_i - t_j}$$

and the kth *divided difference* $f[t_0, t_1, \ldots, t_k]$ is defined by

$$a_k = f[t_0, t_1, \ldots, t_k] = \frac{f[t_0, t_1, \ldots, t_{k-1}] - f[t_1, t_2, \ldots, t_k]}{t_0 - t_k}, \quad k > 0$$

Newton's divided difference interpolation polynomial is the interpolation polynomial agreeing with the function f at the points $t_0, t_1, \ldots, t_n$ and is given by

$$p_n(t) = a_0 + (t - t_0)a_1 + (t - t_0)(t - t_1)a_2 + \cdots + (t - t_0)(t - t_1)\cdots(t - t_{n-1})a_n \quad (2)$$

or, rearranging,

$$p_n(t) = a_0 + (t - t_0)\{a_1 + (t - t_1)\{a_2 + \cdots + (t - t_{n-2})\{a_{n-1} + (t - t_{n-1})a_n \overbrace{\}\ldots\}\}}^{n-1 \text{ times}}$$

This form consists of two additions and a multiplication per level in the expression. Since there are n levels, the operation count is seen to be

$$n\,(2A + M)$$

which is more economical than the standard Lagrange form. This leads us to the so-called *nested multiplication or Horner's algorithm*:

Given the $n+1$ distinct points $t_0, t_1, \ldots, t_n$ with associated coefficients $a_0, a_1, \ldots, a_n$, the value of the interpolating polynomial $p_n(t)$ for some $t \in [t_0, t_n]$ is given by b_0 according to the following iteration:

Set $b_n = a_n$
For $k = n - 1$ to 0 by -1
$b_k = a_k + (t - t_k)\, b_{k+1}$
End For

By the uniqueness of the interpolating polynomial there can be no difference in comparison to Lagrange, but the gain in speed may be of importance for our purposes. Note that the divided differences a_k can be calculated and stored in advance of the actual interpolation so that the operation counts given here reflect the operations needed at interpolation time. A total operation count would have to include the operations needed to generate the divided differences. Also, calls from storage may need to be counted, depending on system architecture considerations.

The divided difference algorithm does not take advantage of the fact that our interval sizes are fixed – we required the nodes to be distinct but made no restriction on the spacing between nodes. In the next section we will investigate the special case when the interval sizes are constant.

Difference Tables

The case of equally-spaced data points is a special case of Newton's divided differences and leads to other interpolation formulas. The error and operation counts for the methods presented here are essentially identical to those presented above. The formulas are given in their simplest form and should not be used for computation. A nested multiplication approach similar to the one described in the previous section should be used for each of these in order to minimize the cost of computation. One important aspect of this method

is the determination of the differences and the method to be used to interpolate at a given time t. It will be necessary to include some code to determine which differences are to be used, though the differences themselves can be calculated when the given ephemeris becomes available. In addition, the chosen method will depend on the location of the interpolation time relative to the data times. Here we follow the description given in Buchanan and Turner [1992]. Our data occurs at times which can be expressed as

$$t_k = t_0 + kh$$

where t_0 is a reference time for the interval of interest and h is the constant steplength. We normally think of k as being a positive integer and t_0 as being the initial time in the interval of interest but in this case we will only require k to be an integer and t_0 to be any time corresponding to a data point in the interval. The sign of k will depend on the reference time t_0 in relation to the time of interest. There are now several *differences* which can be defined, one of which is the *forward difference.*

The general *forward difference* $\Delta f(t_i)$ is given by

$$\Delta f(t_i) = f(t_{i+1}) - f(t_i) = f(t_i + h) - f(t_i)$$

Its powers are calculated recursively according to the following

$$\Delta^k f(t_i) = \Delta(\Delta^{k-1} f(t_i)) = \Delta^{k-1} f(t_{i+1}) - \Delta^{k-1} f(t_i)$$

$$\Delta^k f_i = \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} f_{i+j}$$

where we have introduced the notation $f_i = f(t_i)$. Additionally, the differences are related to divided differences by

$$\Delta^k f(t_i) = k!\, h^k\, f[t_i, t_{i+1}, \ldots t_{i+k}]$$

An application of this last formula to equation (2) immediately yields a forward difference formula (called *Newton's forward difference formula* or *the Newton-Gregory forward difference formula*). Here we assume that the degree of the interpolating polynomial is n while the number of data points in our table is N:

$$p_n(t) = f_0 + \frac{(t - t_0)}{h}\Delta f_0 + \frac{(t - t_0)(t - t_1)}{2\,h^2}\Delta^2 f_0 + \cdots + \frac{(t - t_0)(t - t_1)\cdots(t - t_{n-1})}{n!\,h^n}\Delta^n f_0$$

A simple change of variable $\tau = (t - t_0)/h$ yields the compact form

$$p_n(\tau) = \sum_{j=0}^{n} \binom{\tau}{j} \Delta^j f_0$$

with the generalized binomial coefficients

$$\binom{\tau}{j} = \frac{\tau(\tau - 1)\cdots(\tau - j + 1)}{j!}$$

Examination of these formulas reveals that their operation counts could be made similar to the operation count for the Newton divided difference formula with a constant step size through the use of nested multiplication. Their accuracy is the same as the Newton divided difference formula so that none should be ruled out in the search for a more rapid method of interpolation.

Cubic Spline Interpolation

Cubic spline interpolation is computationally efficient and has an advantage with respect to walk-along Lagrange because it allows the user to calculate the interpolating polynomials over the entire interval at one time, at the beginning of the interpolation process. This calculation involves solving a tridiagonal system of equations. Additionally, the fact that each subinterval is represented by a cubic polynomial means that evaluations on those intervals are much quicker than their eleventh order polynomial counterparts, making the cubic spline method a good choice where accuracy is less important than speed.

Here we will sketch the derivation of the cubic spline interpolation; thorough treatments of cubic spline interpolation can by found in [Ahlberg, Nilson and Walsh, 1967], [Buchanan and Turner, 1992], [de Boor, 1978], and [Press, 1992]. Given the $n+1$ ephemeris values $f(t_0), f(t_1), \ldots, f(t_n)$ at the distinct times $a = t_0, t_1, \ldots, t_n = b$, we construct a piecewise cubic interpolant p to f as follows. On each subinterval $[t_i, t_{i+1}]$ we wish to construct a cubic polynomial p_i in such a way that the resulting interpolation formula over the entire interval is continuous in its first and second derivatives. The result is

$$p_i(t) = A_i(t)\, f(t_i) + B_i(t)\, f(t_{i+1}) + C_i(t)\, f''(t_i) + D_i(t)\, f''(t_{i+1}), \qquad i = 0, \ldots, n-1 \quad (3)$$

where

$$A_i = \frac{t_{i+1} - t}{t_{i+1} - t_i}, \qquad B_i = \frac{t - t_i}{t_{i+1} - t_i}$$

$$C_i = \frac{1}{6}\,(A_i^3 - A_i)\,(t_{i+1} - t_i)^2, \qquad D_i = \frac{1}{6}\,(B_i^3 - B_i)\,(t_{i+1} - t_i)^2$$

We do not yet have the $n+1$ values of $f''(t_i)$ needed for the determination of the solution, but application of the continuity of the first derivative on the entire interval leads us to the following equations:

$$\frac{t_i - t_{i-1}}{6} f''(t_{i-1}) + \frac{t_{i+1} - t_{i-1}}{3} f''(t_i) + \frac{t_{i+1} - t_i}{6} f''(t_{i+1}) = \frac{f(t_{i+1}) - f(t_i)}{t_{i+1} - t_i} - \frac{f(t_i) - f(t_{i-1})}{t_i - t_{i-1}} \quad (4)$$

for $i = 1, 2, \ldots, n-1$. Note that there are $n-1$ equations for $n+1$ unknowns, leaving two second derivatives undetermined. The choice of the two boundary conditions $f''(a)$ and $f''(b)$ provides the required unique solution. In our case it makes sense to use the periodicity, i.e. apply (3) and (4) for $i = n$ and enforce $p_0(t) \equiv p_n(t + t_n)$ for $t_0 \le t \le t_1$. As a consequence, $f(t_0) = f(t_n)$, $f(t_1) = f(t_{n+1})$, $f''(t_0) = f''(t_n)$, and $f''(t_1) = f''(t_{n+1})$. Therefore if we use equal step size h, the cubic spline version of equation (4) can be written

as

$$\begin{pmatrix} 4 & 1 & 0 & 0 & \cdots & 1 \\ 1 & 4 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 4 & 1 & 0 & \cdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \\ 0 & \cdots & 0 & 1 & 4 & 1 \\ 1 & \cdots & 0 & 0 & 1 & 4 \end{pmatrix} \begin{pmatrix} f''(t_1) \\ f''(t_2) \\ f''(t_3) \\ \vdots \\ f''(t_{n-1}) \\ f''(t_n) \end{pmatrix} = \begin{pmatrix} R_1 \\ R_2 \\ R_3 \\ \vdots \\ R_{n-1} \\ R_n \end{pmatrix} \tag{5}$$

where $R_i = 24\, f[t_{i-1}, t_i, t_{i+1}]$.

If accuracy can be sacrificed for speed then the cubic spline method may be preferred over any of the methods here. However, the $O(h^4)$ accuracy provided by the cubic spline may be insufficient for GPS satellite interpolation requirements.

Trigonometric Polynomial Interpolation

The roots of this method of interpolation can be traced to the beginning of the nineteenth century. Briggs and Henson [1995] present a brief history of this method, in particular the fact that Gauss used it around 1800 to interpolate the orbit of the asteroid Pallas. The preceding methods are standard interpolation techniques typically used for continuous, differentiable functions defined on compact intervals. No other special information about the functions being interpolated is exploited by these methods. It is at this point that we examine some special properties of our GPS ephemeris data. As previously mentioned, our ephemeris data is supplied over an interval of eight days and consists of Earth-centered, Earth-fixed Cartesian coordinate position data given every 900 seconds. A plot of the data in Figure (1) shows that it is very close to periodic, and it is for this reason that we examine the trigonometric polynomial interpolation method.

Due to the fact that the position data has a period of twenty-four hours, we restrict our attention to a single twenty-four hour period and generate a trigonometric polynomial ρ_n using all the data available over that period. Again, we must remember that our satellite orbit is not truly periodic, but very close to that in inertial coordinates. Since the error incurred by assuming the data to be periodic over a j day period would be almost j times as great as the error incurred from assuming the orbit to be periodic over a single day, we discourage use of this routine over intervals exceeding the fundamental period of the data. In order to minimize the effects of the assumption of periodicity one should interpolate over a single period.

The idea in generating our interpolant is to assume that the data is from a periodic function of time defined over the interval $[0, 2\pi)$ which can be represented by a trigonometric polynomial of the form

$$\rho_n(t) = a_0 + \sum_{k=1}^{n} (a_k \cos kt + b_k \sin kt)$$

In our problem we simply map the twenty-four hour interval of interest into the interval $[0, 2\pi)$ using a linear change of variable. In deriving the coefficients we follow the discussion of interpolating polynomials and the Fast Fourier Transform found in Buchanan and Turner

[1992]. Clearly, the above equation can be written as

$$\rho_n(t) = \sum_{k=-n}^{n} \gamma_k \, e^{ikt}$$

where the coeffcients are given by

$$\gamma_k = \alpha_k + i\beta_k \qquad k = -n, \ldots, n$$

If we denote the point on the unit circle corresponding to $t \in [0, 2\pi)$ by

$$\xi = e^{it}$$

we may then write

$$\rho_n(t) = \sum_{k=-n}^{n} \gamma_k \, \xi^k$$

If we denote the points corresponding to nodes by

$$\xi_j = e^{it_j} = \cos t_j + i \sin t_j$$

the interpolation problem is to find the coefficients satisfying

$$\rho_n(t_j) = \sum_{k=-n}^{n} \gamma_k \, \xi_j^k = f(t_j)$$

where $f(t_j)$ are the function values for $j = 0, 1, \ldots, 2n$. For an odd number of nodes, it can be shown that

$$\gamma_p = \frac{1}{2n+1} \sum_{k=0}^{2n} f_k \, \xi_k^{-p} \qquad p = -n, -n+1, \ldots, n$$

Unfortunately, as pointed out by Buchanan and Turner [1992], the operation count for this discrete Fourier Transform is $O(n^2)$ making it too expensive for practical application. Luckily, there is a faster way to calculate the discrete Fourier coefficients.

Suppose that we have the case where the number of nodes is of the form $n = 2^N$. The task is then to find the trigonometric polynomial ρ_n which interpolates f at the uniformly spaced data set $t_0, t_1, \ldots, t_{n-1}$. In this case the trigonometric polynomial can be calculated by using the Fast Fourier Transform, which has an operation count of $O(n \log n)$. This makes it an attractive method for our purposes and we will refer to it as the FFT method. Unfortunately, however, over a twenty-four hour period our data will consist of ninety-six points separated by intervals of 900 seconds. Though ninety-six is not a power of two, we nevertheless can use $n = 2^5 = 32$ points in the interval to test our trigonometric polynomials. We simply select every third point in the interval and calculate the trigonometric polynomial which interpolates at those thirty-two points. We may then compare the Lagrange method to the FFT method by choosing some subinterval having a length of 36 nodes, twelve of which are used to generate the eleventh order Lagrange interpolating polynomial. These

twelve nodes (equally spaced with 3 times the spacing) should coincide with twelve of the thirty-two nodes used in the FFT method.

The above calculations were performed with the aid of Maple (Appendix B) and a plot of residuals in centimeters is shown in Figures (2) and (3). A linear change of variable was used to map the trigonometric polynomial defined on the interval $[0, 93\pi/48]$ into the interval $[-1, 1]$.

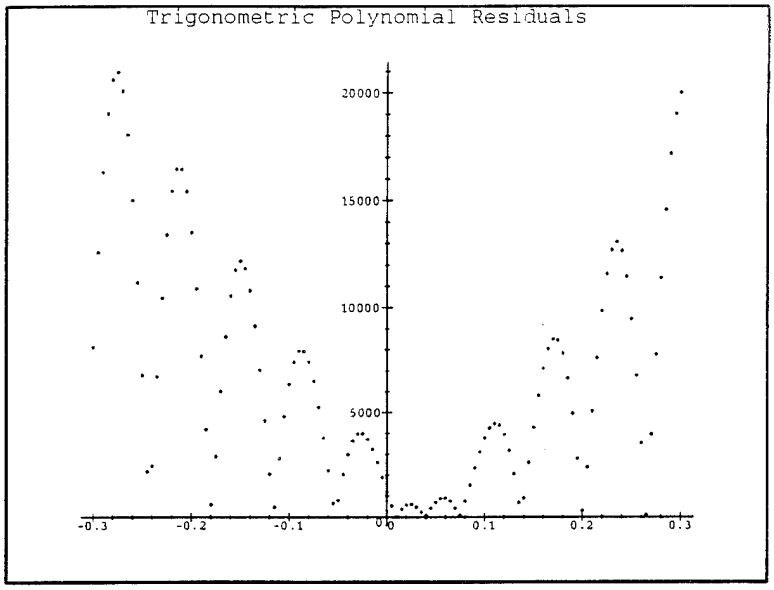

Figure (2) – Trigonometric polynomial residuals (cm)

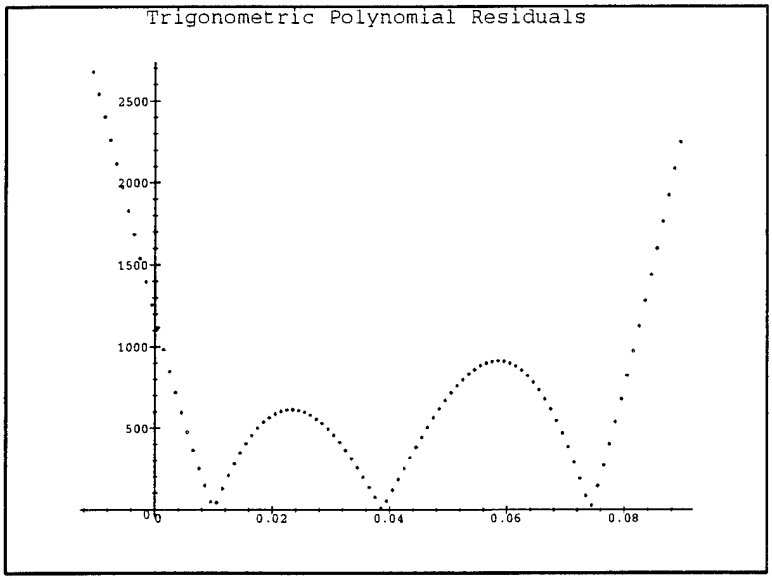

Figure (3) – Trigonometric polynomial residuals (cm), (zoomed)

The subinterval chosen for this analysis was located at the center of the set of nodes in order to show the location where the trigonometric interpolant is most accurate. Examina-

tion of Figure (5) reveals that on a single subinterval the trigonometric polynomial method agrees with the Lagrange interpolation method to within roughly ten meters. The Maple worksheet used to generate this figure is included as Appendix C at the end of this paper.

The effects of error growth near the ends of the intervals for the FFT method could be handled by shifting the twenty-four hour period over which the trigonometric polynomial is derived, placing the data point squarely in the center for the most accurate work. A bound on the mean square error incurred by approximating the periodic function f from which the data is sampled by the interpolant ρ is given in Briggs and Henson [1995] as

$$||f - \rho|| \leq \frac{C}{N^{p+1}}$$

where N is the number of data points, C is a constant and the periodic extension of f has $(p-1)$ continuous derivatives, $p \geq 1$. Since continuity of the periodic extension of f is required for this bound we cannot use it unless the function is truly periodic on the chosen interval. Briggs and Henson [1995] perform a trigonometric interpolation on an arbitrary polynomial defined on $[-1, 1]$ and comment that it is not unreasonable to suspect the mean square errors to decrease as N^{-1}.

Another concern will be the time required to evaluate the interpolant, so it is of interest to discuss the rapid calculation of the trigonometric polynomials, in particular the terms

$$\cos \pi t,\ \sin \pi t,\ \cos 2\pi t,\ \sin 2\pi t, \ldots, \cos n\pi t,\ \sin n\pi t.$$

Using the trigonometric summation formulas one can write the well-known recursion [Goertzel, 1960]

$$\sigma_k = \cos k\pi t$$

$$\tau_k = \sin k\pi t$$

$$\begin{pmatrix} \sigma_{k+1} \\ \tau_{k+1} \end{pmatrix} = \begin{pmatrix} \sigma_1 & -\tau_1 \\ \tau_1 & \sigma_1 \end{pmatrix} \begin{pmatrix} \sigma_k \\ \tau_k \end{pmatrix} \qquad k = 1, 2, \ldots$$

which requires only two trigonometric calculations in order to recover all the needed terms. Of course, the growth of round-off errors should always be checked when using a long sequence (i.e. n large).

It is important that we use all the available data in forming the trigonometric polynomial $\rho_n(t)$ because the mean square error involved in using the FFT method depends so critically on the number of data points. Since 96 is not a power of two and since the DFT is computationally expensive, we must resort to another algorithm.

Tshebyshev Polynomial Interpolation

In order for us to understand Tshebyshev polynomial interpolation, it is necessary to re-examine the polynomial interpolation of $n+1$ data points on a given interval. Given $n+1$ points on an interval, the nth degree polynomial which interpolates those points over the given interval is unique. Of course this does not prevent us from writing the polynomial in a number of different ways. Before we describe what Tshebyshev polynomial interpolation

is, it may be a good idea to say what it is not. Tshebyshev polynomial interpolation is not the expression in the Tshebyshev polynomial basis of an nth order polynomial which interpolates $n+1$ *arbitrary data points* on a given interval. We are free to choose any suitable polynomial basis for expressing such a polynomial, but we must remember that the results of any calculations performed using that polynomial will be the same regardless of how the polynomial is expressed; simply writing a given polynomial in a different basis does not alter it. Therefore the residuals which are produced when using these different forms of the same polynomial will be the same. However, another set of $n+1$ points on the same interval would yield a different polynomial. As it turns out, polynomial interpolation is sensitive to the distribution of the points being interpolated. If we were allowed to choose the points on an interval to be interpolated, we would find that certain choices of $n+1$ points would yield polynomials which did a better job of interpolating than others. As stated, our data points are evenly spaced throughout the interval of interest, so that will not have the luxury of choosing a preferred set of $n+1$ points on any interval unless we are able to generate more data. The following discussion closely follows the treatment given by Press et al [1992]. The Tshebyshev polynomial of degree n on $[-1,1]$ is defined as

$$T_n(t) = \cos(n \arccos t)$$

It follows that the Tshebyshev polynomials satisfy the three term recurrence relation

$$T_{n+1}(t) = 2t\,T_n(t) - T_{n-1}(t) \qquad n = 1, 2, \ldots$$

In addition,

$$\int_{-1}^{1} \frac{T_n(t)\,T_k(t)}{\sqrt{1-t^2}} = 0, \qquad k \neq n$$

and

$$\int_{-1}^{1} \frac{T_n(t)\,T_k(t)}{\sqrt{1-t^2}} = \begin{cases} \pi & k = n = 0 \\ \pi/2 & k = n \neq 0 \end{cases}$$

so that the Tshebyshev polynomials are orthogonal on the interval $[-1,1]$ with respect to the weight function $w(t) = 1/\sqrt{1-t^2}$. The first few Tshebyshev polynomials are given by

$$T_0(t) = 1$$

$$T_1(t) = t$$

$$T_2(t) = 2t^2 - 1$$

$$T_3(t) = 4t^3 - 3t$$

$$T_4(t) = 8t^4 - 8t^2 + 1$$

$$T_5(t) = 16t^5 - 20t^3 + 5t$$

$$\vdots$$

It can be shown that the zeros of $T_n(t)$ on $[-1,1]$ are given by

$$t_j = \cos\left[\frac{\pi(j-1/2)}{n}\right] \qquad j = 1, 2, \ldots, n$$

and that the Tshebyshev polynomials satisfy a discrete orthogonality condition [Press et al, 1992]

$$\sum_{j=1}^{n} T_i(t_j)\, T_k(t_j) = \begin{cases} 0 & i \neq k \\ n/2 & i = k \neq 0 \\ n & i = k = 0 \end{cases}$$

The powers of t can also be expressed in the Tshebyshev polynomial basis.

Any function $f(t)$ can be approximated by a finite linear combination of Tshebyshev polynomials, i.e.

$$f(t) = \frac{1}{2}a_0 + \sum_{i=1}^{N} a_i T_i(t)$$

where

$$a_i = \frac{2}{\pi}\int_{-1}^{1} \frac{f(t)T_i(t)}{\sqrt{1-t^2}}dt.$$

These coefficients a_i can be computed numerically using the M equally spaced data points

$$t_j = -1 + j\Delta t, \qquad j = 0, 1, \cdots, M-1$$

$$\Delta t = \frac{2}{M-1}$$

e.g. via the composite midpoint rule (we have to avoid evaluating the integrand at the endpoints, t_0 and t_{M-1})

$$a_i = \frac{2}{\pi}\sum_{j=0}^{M-2} \frac{\frac{1}{2}f(t_{j+1/2})T_i(t_{j+1/2})}{\sqrt{1-t_{j+1/2}^2}}\Delta t,$$

where $f(t_{j+1/2})$ can be approximated by a quadratic or cubic polynomial. For example to use cubic splines to approximate the integrand, we get

$$\frac{2}{\pi}\left[\frac{h^4}{4}\sum_{j=0}^{M-2} a_j + \frac{h^3}{3}\sum_{j=0}^{M-2} b_j + \frac{h^2}{2}\sum_{j=0}^{M-2} c_j + h\sum_{j=0}^{M-2} d_j\right]$$

where the integrand $g(t)$ is approximated on the interval $[t_j, t_{j+1}]$ by the cubic polynomial

$$g(t) = a_j(t-t_j)^3 + b_j(t-t_j)^2 + c_j(t-t_j) + d_j$$

We can also use the midpoint rule on the two leftmost panels and the two rightmost panels only. On the rest of the panels we can use the fourth order Simpson's $\frac{1}{3}$ rule. This will yield the following approximation for the coefficients:

$$a_i = \frac{\Delta t}{\pi}\left\{4g_2 + \frac{2}{3}g_3 + \frac{8}{3}g_4 + \frac{4}{3}g_5 + \cdots + \frac{8}{3}g_{M-3} + \frac{2}{3}g_{M-2} + 4g_{M-1}\right\},$$

where

$$g_j = \frac{f(t_j)T_i(t_j)}{\sqrt{1-t_j^2}}.$$

If the number of panels left for Simpson's $\frac{1}{3}$ rule is not even, we can take the trapezoidal rule over one of the panels (in here we took the third from last panel). For example, we show below the formula for an odd number of panels and give the Fortran code for this calculation in Appendix C:

$$a_i = \frac{\Delta t}{\pi}\left\{4g_2 + \frac{2}{3}g_3 + \frac{8}{3}g_4 + \frac{4}{3}g_5 + \cdots + \frac{8}{3}g_{M-4} + \frac{5}{3}g_{M-3} + g_{M-2} + 4g_{M-1}\right\}.$$

To evaluate $f(\tau)$ we then use the following recursion:

$$d_{N+1} = d_{N+2} = 0$$

$$d_j = 2\tau d_{j+1} - d_{j+2} + a_j, \qquad j = N, N-1, \cdots, 2, 1$$

$$f(\tau) = d_0 = \tau d_1 - d_2 + \frac{1}{2}a_0.$$

The benefit of Tshebyshev expansion is that the maximum deviation on the interval is minimized. Thus it will not suffer from the disadvantage of higher order polynomials, that is the error here doesn't grow rapidly near the endpoints of the interval. Therefore we eliminate the need for "walking" interpolator.

Conclusions

This has been a preliminary study of various interpolation methods for GPS ephemeris data. The alternate long-arc methods of interpolation we have tried so far yield much greater than 1 *cm* error when compared to the short-arc eleventh order Lagrange polynomial. However, more work should be done before completely ruling out these alternate methods.

Our study does indicate that the use of difference tables should be more efficient than the direct method currently used to construct and evaluate the Lagrange polynomials. Precise run times of these faster methods should be measured.

Acknowledgement

This research was supported in part by the Naval Postgraduate School Research Program. The authors gratefully acknowledge the partial support of NISE West Coast Division.

References

Ahlberg, J. H., Nilson, E. N., and Walsh, J. L., 1967, *The theory of Splines and Their Applications*, (New York: Springer Verlag)

Briggs, W.L. and Henson, V. E. 1995, *The DFT, An Owner's Manual for the Discrete Fourier Transform* (Philadelphia: SIAM)

Buchanan, J.L. and Turner, P.R. 1992, *Numerical Methods and Analysis* (New York: McGraw-Hill, Inc.)

de Boor, C. 1978, *A Practical Guide to Splines*, vol. 27 of *Applied Mathematical Sciences* (New York: Springer-Verlag)

Goertzel, G. 1960, Fourier Series, *Mathematical Methods for Digital Computers* (New York: Wiley)

Malys, S., and M. J. Ortiz, 1989, Geodetic absolute positioning with differenced GPS carrier beat phase data, Proceedings of the 5^{th} International Symposium on Satellite Positioning, 13-17 March 1989, Las Cruces, NM.

Press, W. H., B. P. Flannery, S. A. Teukolsky and W. T. Vetterling, 1992, *Numerical Recipes in FORTRAN, The Art of Scientific Computing, 2nd Edition* (New York: Cambridge University Press)

Remondi, B. W. 1989, *Extending the National Geodetic Survey Standard GPS Orbit Formats*, NOAA Technical Report NOS 133 NGS 46, (Rockville, MD: US Department of Commerce, NOAA)

Remondi, B. W. 1991, *NGS Second Generation ASCII and Binary Orbit Formats and Associated Interpolation Studies*, (Rockville, MD: US Department of Commerce, NOAA)

Smith, R. and V. Curtis, 1983, Interpolation Study, Naval Surface Weapons Center Dahlgren Laboratory Memorandum.

Watkins, M., 1995, Private communication.

Appendix A

This section provides a FORTRAN subroutine which can be used to perform the interpolation discussed in this paper.

```
      Program lagrange
c
c       implements lagrange interpolation  of  a function
c       given as equally spaced data points
c
c        let t_j=-1+j Delta t, Delta t=2/(M-1),  M=96
c
         implicit real*8 (a-h,o-z)
         real*8 t(96),f(96),y(96)
         real*4 taray(2),paray(2)
         atony=dtime(taray)
c         m is number of data points
         m=96
c     mp is the number of interpolating points
         mp=12
         print *,' m= ',m,' mp=',mp
c
c        get m(=96) equally spaced points on [-1,1]
c
         deltat=2./dfloat(m-1)

         print *, 'delta t ',deltat
         t(1)=-1.
         do 10 j=2,m
10       t(j)=t(j-1)+deltat
c
c        compute f values
c        synthetic data
c
         call finter(t,m,f)
c
c    print the points
c
          print 12,(j,t(j),f(j),j=1,m)
12        format(2x,i3,2e14.7)
c
```

```
c          compute y values
c
           call yi(t,f,mp,y)
         atony=etime(taray)
         write(6,*) ' User time = ', taray(1) ,' seconds'
         write(6,*) ' System time = ', taray(2),' seconds'
         write(6,*) ' Total time = ', taray(1) + taray(2),' seconds'
          btony=dtime(paray)
             ttt=-1.d0-.5d0*deltat
             print *,'     t     ','approximate  exact      error'
            do 5000 ij=1,mp
             ttt=ttt+deltat
c
c        compute phi_n (tau)
c
           call phi(t,ttt,mp,phin)
           sum=0.
           do 1000 j=1,mp
1000       sum=sum+y(j)/(ttt-t(j))
           pn=phin*sum
           call finter(ttt,1,ddd)
           print *,ttt,pn,ddd,pn-ddd
5000       continue

         btony=etime(paray)
         write(6,*) ' User time = ', paray(1) ,' seconds'
         write(6,*) ' System time = ', paray(2),' seconds'
         write(6,*) ' Total time = ', paray(1) + paray(2),' seconds'

           stop
           end

          subroutine finter(t,m,f)
          implicit real*8 (a-h,o-z)
          real*8 t(1),f(1)
          do 10 j=1,m
          f(j)=dsin(t(j))+dcos(t(j))
10        continue
          return
          end

          subroutine yi(t,f,m,y)
          implicit real*8 (a-h,o-z)
          real*8 t(1),f(1),y(1)
```

```
      do 20 i=1,m
      y(i)=f(i)
      do 10 j=1,m
      if(j.eq.i) go to 10
       tt=t(i)-t(j)
      y(i)=y(i)/tt
10    continue
20    continue
      return
      end

      subroutine phi(t,ttt,m,phin)
      implicit real*8 (a-h,o-z)
      real*8 t(1)
      phin=1.d0
      do 10 j=1,m
      phin=phin*(ttt-t(j))
10    continue
      return
      end
```

Appendix B

This section gives a Maple program for Lagrange/FFT Comparison using GPS Ephemeris.

```
\# Christopher P. Sagovac, 29 May 1995

\# The following x-coordinate data from satellite 1 in the file jpl07365.sp3
is used to compare the Lagrange Polynomial and FFT methods of
interpolation. 96 points separated at intervals of 900 seconds cover 23hrs
45mins.  11th order Lagrange interpolation is performed on 12 evenly
distributed points on a subinterval within the ephemeris; remaining points
are used to generate residuals.  The user specifies the subinterval over
which the interpolation takes place by setting the starting node number in
the range a=1..61.  For the Lagrange interpolation, the  ephemeris data is
mapped into the interval [-1,1].  Selected points are separated  by 2700
seconds.  The FFT is performed over the entire ephemeris using 32 equally
spaced points with subinterval spacing of 2700 seconds; remaining points
are used  to generate residuals. For the FFT interpolation, the ephemeris
data is mapped  into the interval [0,95*$\pi$/48]. This allows a comparison
of the Lagrange interpolating polynomial and the FFT method over an ephemeris
interval.
Residual comparison takes place at corresponding node numbers on each of the
different subintervals.
First we generate a list of the ninety-six abscissas for both intervals:

> list1:=[seq(m*2*$\pi$/96, m=0..95)]:

> list2:=[seq(-1+m*2/95, m=0..95)]:

\# A list of the corresponding ninety-six ordinate values taken from file
jpl07365.sp3

\# (these are the same for both sets of abscissas):

> satlist2:=
[15500.414560,15268.853410,14843.274690,14200.160540,13322.919650,

>12202.689520,10838.841980,9239.173760,7419.776750,5404.594490,3224.683250,

>917.206990,-1475.795270,-3908.822870,-6334.455100,-8704.963870,-10973.938070,

>-13097.860380,-15037.578740,-16759.618700,-18237.288490,-19451.535890,
```

```
>-20391.524780,-21054.909440,-21447.795240,-21584.386110,-21486.330880,
>-21181.791750,-20704.269190,-20091.227140,-19382.570580,-18619.033950,
>-17840.542760,-17084.613010,-16384.851470,-15769.617150,-15260.897820,
>-14873.447530,-14614.220590,-14482.125710,-14468.110980,-14555.576720,
>-14721.100000,-14935.441300,-15164.792660,-15372.216220,-15519.214910,
>-15567.371330,-15479.988750,-15223.668540,-14769.761120,-14095.633690,
>-13185.705450,-12032.211050,-1063.663960,-9005.003600,-7157.422240,
>-5117.879990,-2918.327780,-596.668940,1804.500890,4239.327820,6660.254360,
>9019.637090,11271.368100,13372.440740,15284.402460,16974.641660,
>18417.461210,19594.898880,20497.263810,21123.368570,21480.447000,
>21583.759770,21455.901300,21125.832970,20627.678050,19999.323680,
>19280.882770,18513.075050,17735.590120,16985.496810,16295.762020,
>15693.938430,15201.073980,14830.887900,14589.247140,14473.965140,
>14474.931940,14574.570650,14748.602380,14967.088360,15195.707160,
>15397.214850,15533.028660,15564.870070]:

\# Normalize the data by dividing by $10^5$:

> list3:=[seq(evalf(satlist2[m]/100000,11),m=1..96)]:

\# We next form the 11th order Lagrange interpolating polynomial using twelve
evenly spaced points on the interval.  The other unused points will be used to
examine residuals.  Setting the parameter "a" (below) chooses the point at which
the interval begins; valid choices are a=1..61.  Subintervals chosen to lie
near the ends of the ephemeris interval represent a worst case for the FFT
method, which attains its greatest accuracy in the center of the interval.
For instance, the leftmost interval of length 36 points corresponds to a=1.
The rightmost interval of length 36 corresponds to a=61.  In this way one can
peruse the entire ephemeris interval to see how the 32 point FFT method
compares to the local step along Lagrange method.  Choose a value in the
```

```
center of the interval, a=32:

> a:=32:

\# Here are the values of the abscissas for the chosen subinterval on the
interval [-1,1):

> plist1:=[seq(evalf(list2[a+3*m],16), m=0..11)]:

\# Here are the values of the ordinates (nodes) for the chosen subinterval
(from list3):

> plist2:=[seq(list3[a+3*m],m=0..11)]:

\# Now calculate the 11th degree Lagrange interpolating polynomial over the
chosen subinterval:

> p:=interp(plist1, plist2, z):

> readlib(FFT):

\# Form a list of 32 evenly distributed data points from list3:

> fftlist2:=[seq(list3[1+3*m],m=0..31)]:

> x := array(fftlist2):

\# There is no imaginary component for this real data set; y is a zero array:

> y :=
array([0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]):

\# Use hardware floating point number system for fast calculation:

> evalhf(FFT(5,var(x),var(y))):

> r:=map(x->x/32,convert(x, list)):

> s:=map(y->y/32,convert(y,list)):

> evenf:=r[1]+r[17]*cos(16*z):

> for n from 1 to 15 do evenf:= evenf+2*r[n+1]*cos(n*z) od:
```

```
> oddf:=0:

> for n from 1 to 15 do oddf:=oddf+ (-2)*s[n+1]*sin(n*z) od:

\# Here then is the interpolating trigonometric polynomial, good for the
entire 24 hour period:

> g:=oddf+evenf:

\# Plot the difference between the FFT method and the Lagrange interpolating
polynomial on [-1,1]:

> FFTResids1:=[seq([-.3+m/150,$10^{10}$*abs(evalf(subs(z=evalf(-.3+m/150,20),
p),16)-

> evalf(subs(z=95/96*$\pi$*(.7+m/150), g),20))],m=0..90)]:

\# interface(plotdevice=postscript, plotoutput=interp4);

> plot(FFTResids1, style=point, title='Trigonometric Polynomial Residuals');

>
FFTResids2:=[seq([evalf(m/1000),$10^{10}$*abs(evalf(subs(z=evalf(evalf(m/1000)
,20), p),16)-

> evalf(subs(z =95/96*$\pi$*(1+m/1000), g),20))],m=0..90)]:

\# interface(plotdevice=postscript, plotoutput=interp5);

> plot(FFTResids2, style=point, title='Trigonometric Polynomial Residuals');
```

Appendix C

This section gives a Fortran program for Tshebyshev approximation using Simpson'n $\frac{1}{3}$ rule for the numerical approximations of the coefficients.

```
          subroutine cheb(rr,a,ti,n,alf,t,f,m)
          implicit real*8 (a-h,o-z)
           real*8 a(1),ti(1),alf(1),t(1),f(1)
c
c  rr is delta t / pi
c  (t,f) are the equally spaced data points
c   n is the highest degree of Tshebyshev polynomial in the expansion
c   m  is the number of data points
c
c   a(i) are the coefficients of the expansion in  Tshebyshev polynomials
c
         do 20 i=1,n
20       a(i)=0.
c
c        weights (alf) for midpoint/trapezoidal/simpson combination
c
          alf(2)=4.d0
          alf(3)=2.d0/3.d0
          do 22 i=4,m-4,2
          alf(i)=8.d0/3.d0
          alf(i+1)=4.d0/3.d0
22        continue
          alf(m-3)=5.d0/3.d0
          alf(m-2)=1.d0
          alf(m-1)=4.d0
c
c         integration
c
          do 30 j=2,m-2
          tt=t(j)
           den=dsqrt(1.-tt*tt)
c          print *,'j,tt,den=',j,tt,den
         do 40 i=1,n

c
c    compute all Tshebyshev polynomial up to degree n at point tt
c
         call cheby(ti,n,tt)
```

```
            term=ti(i)*f(j)*alf(j)/den*rr
            a(i)=a(i)+term
c            print *,'j i ti(j),a(i)',j,i,ti(i),a(i)
40          continue
30          continue
c             print *,' a(i)= ', a

          return
          end
```

Distribution List

	No. of copies
Director Defense Technology Information Center Cameron Station Alexandria, VA 22314	2
Director of Research Administration Code 81 Naval Postgraduate School Monterey, CA 93943	1
Library Code 52 Naval Postgraduate School Monterey, CA 93943	2
Department of Mathematics Code MA Naval Postgraduate School Monterey, CA 93943	1
Professor D. A. Danielson Code MA/Dd Naval Postgraduate School Monterey, CA 93943	10
Professor Beny Neta Code MA/Nd Naval Postgraduate School Monterey, CA 93943	2
Professor J. R. Clynch Code OC/Cl Naval Postgraduate School Monterey, CA 93943	1

Professor Rudolf Panholzer 1
Code SP/Pz
Naval Postgraduate School
Monterey, CA 93943

Professor Terry Alfriend 1
Code SP/Al
Naval Postgraduate School
Monterey, CA 93943

Professor Mike Ross 1
Code AA/Ro
Naval Postgraduate School
Monterey, CA 93943

Lt. Chris Sagovac, USN 5
819 South Charles St.
Baltimore, MD 21230

Dr. Stephen H. Knowles 1
U.S. Naval Space Command
Code N4/GT
Dahlgren, VA 22448-5170

Dr. Paul Schumacher 1
U.S. Naval Space Command
Code 63T
Dahlgren, VA 22448-5170

Capt Daniel J. Fonte, Jr., USAF 1
PL/VTA
3550 Aberdeen Ave SE
Kirtland AFB, NM 87117-5776

Major Dave Vallado 1
PL/VTA
3550 Aberdeen Ave SE
Kirtland AFB, NM 87117-6008