NOTICE

THIS DOCUMENT HAS BEEN REPRODUCED FROM MICROFICHE. ALTHOUGH IT IS RECOGNIZED THAT CERTAIN PORTIONS ARE ILLEGIBLE, IT IS BEING RELEASED IN THE INTEREST OF MAKING AVAILABLE AS MUCH INFORMATION AS POSSIBLE
A STUDY OF AUTONOMOUS SATELLITE NAVIGATION METHODS USING THE GLOBAL POSITIONING SATELLITE SYSTEM

Submitted by
Byron D. Tapley
April 20, 1980

Prepared by
Department of Aerospace Engineering and Engineering Mechanics
The University of Texas at Austin
Austin, Texas 78712

Introduction

The development of the NAVSTAR Global Positioning System (GPS) will allow satellites to perform orbit determination calculations in real time with on-board computers. Special orbit determination algorithms are being developed to accommodate the size and speed limitations of on-board computer systems. One class of these algorithms consists of square root sequential filtering methods. The purpose of the square root filters is to reduce the likelihood of filter divergence which can occur as a consequence of a small computer word length.

In this initial study, a new method for the time update of the square root covariance matrix was developed. In addition, this time update method is compared with another square root covariance propagation method to determine relative performance characteristics. Comparisons are based on the results of computer simulations of the LANDSAT-D satellite processing pseudo range and pseudo range-rate measurements from the Phase I GPS. A summary of the comparison results is contained in the following paragraphs.

Summary of Results

In square root algorithms that employ triangular square root covariance matrices, time propagation by transition matrix methods destroys the triangularity of the square root covariance matrix. Retriangularization is required after each time update. Such a procedure is required of the square root filter (the UDU^T algorithm) currently proposed for the LANDSAT-D computer. In addition to the computational burden of the
retriangularization, the effects of process noise can only be approximated, if the algorithm is to be computationally efficient. As a part of this study, an algorithm has been developed which integrates the square root covariance directly in its triangular form. Retriangularizations are not necessary in this algorithm, and the effect of process noise is included exactly.

Appendix A contains a derivation of the proposed propagation algorithm for the $\text{UDU}^T$ filter, as well as the results of a performance comparison between the proposed method and a $\text{UDU}^T$ algorithm using a transition matrix time update. Two versions of a standard formulation of the Extended Kalman Filter (EKF) are included in the comparison, also. The results show that, for this test problem, the proposed propagation method has superior performance to the transition matrix update formulation in terms of efficiency and accuracy. Its efficiency is only marginally less than that of the EKF formulations. Detailed results are available in Appendix A.

The results of additional algorithm comparisons are contained in Appendix B. Two more square root methods, the Potter and Carlson algorithms have been included in the comparisons. A directly integrated square root covariance propagation algorithm, similar to that derived in Appendix B show, again, that the direct square root covariance updates can be competitive with the transition matrix formulations in terms of computation efficiency and estimation accuracy.
A SEQUENTIAL ESTIMATION ALGORITHM
USING A CONTINUOUS UDUᵀ COVARIANCE FACTORIZATION

B. D. Tapley* and J. G. Peters†

The University of Texas at Austin
Austin, Texas 78712

Nomenclature

Abstract

A method for propagating the square root of the state error covariance matrix in lower triangular UDUᵀ form is described. The propagation method can be combined with the UDUᵀ measurement incorporation algorithm to obtain a complete square root free triangular estimation algorithm. The method is compared with (1) the UDUᵀ state transition matrix propagation algorithm and (2) the conventional sequential estimation algorithms on the basis of estimation accuracy, computational efficiency and storage requirements, by a simulation of LANDSAT-D processing data from the Phase I Global Positioning System. The numerical

The research was supported by the National Aeronautics & Space Administration, Goddard Space Flight Center under Grant No. NSG 5154.

*Professor, Department of Aerospace Engineering & Engineering Mechanics; Associate Fellow, AIAA.
†Research Assistant, Department of Aerospace Engineering & Engineering Mechanics; Student Member, AIAA.
INDEX CATEGORIES: Spacecraft navigation, guidance and flight-path control; computer communications, information processing and software; spacecraft systems.
results indicate that, while slower than the conventional methods, the method proposed here is more efficient than the previous UD factorizations with regard to computer storage and computation time, and leads to the most accurate estimate of any of the methods considered.

Introduction

Square root filter formulations have been proposed as a means of eliminating the problem of filter divergence in the real-time application of sequential estimation algorithms. In these methods, the state error covariance matrix is replaced by its square root during the propagation and update of the estimate. The state error covariance matrix does not appear explicitly and, if it is required, it can be obtained by multiplying the square root covariance by its transpose. Consequently, it will always be semi-positive definite.

In the initial formulations, the enhanced numerical stability was obtained at the expense of increased computation complexity, and an associated increase in computation time. In Ref. 5, a square root measurement update method is proposed which offers potential improvement in the computational efficiency of the square root filtering methods. This efficiency is created by maintaining the square root covariance matrix in triangular form. Following a procedure based on Givens's transformation, an algorithm has been proposed which factors the state error covariance \( P \) into the form \( P = U D U^T \), where \( U \) is unit upper triangular and \( D \) is diagonal. Using the \( U D U^T \) factorization eliminates the square root functions present in the algorithms discussed in Refs. 1-5. The measurement incorporation formulation derived for this factorization technique is summarized in the Appendix. The proposed algorithm for propagating the estimate is summarized in the following paragraphs.
The discrete time propagation equation for the state error covariance matrix is

$$\mathbf{P}_{k+1} = \Phi(t_{k+1}, t_k) \mathbf{P}_k \Phi^T(t_{k+1}, t_k) + \Gamma_{k+1}$$

(1)

where $\mathbf{P}_{k+1}$ is the a priori state error covariance matrix at $t_{k+1}$, $\mathbf{P}_k$ is the a posteriori state error covariance matrix at $t_k$, $\Phi(t_{k+1}, t_k)$ is the state transition matrix used to map the state from $t_k$ to $t_{k+1}$, and $\Gamma_{k+1}$ is the matrix which accounts for the effects of process noise in the interval from $t_k$ to $t_{k+1}$. The matrices $\Phi(t_{k+1}, t_k)$ and $\Gamma_{k+1}$ satisfy the following equations:

$$\Phi(t, t_k) = \Lambda(t) \Phi(t, t_k); \Phi(t_k, t_k) = 1$$

(2)

$$\Gamma_{k+1} = \int_{t_k}^{t_{k+1}} \Phi(t, t_k) Q(\tau) \Phi^T(t, t_k) d\tau$$

(3)

where $\Lambda(t)$ is a known $nxn$ time-dependent matrix and $Q(t)$ is the process noise covariance matrix.

The discrete square root time propagation algorithm, based on the $UDU^T$ transformation, can be summarized as follows: Form the two matrices

$$W_{k+1} = [\Phi(t_{k+1}, t_k) \mathbf{U}_k : \mathbf{B}_{k+1}]$$

(4)

$$\mathbf{D}_k = \begin{bmatrix} \mathbf{D}_k & 0 \\ \vdots & \ddots \\ 0 & Q_k/\Delta t \end{bmatrix}$$

(5)
where

\[ B_{k+1} = \int_{t_k}^{t_{k+1}} B(t) \, dt \]  

(6)

The process noise covariance matrix, \( Q_k \), is assumed to be constant in the integration interval \( \Delta t = t_{k+1} - t_k \), and \( \Gamma_{k+1} \), as defined in Eq. (3), is approximated as

\[ \Gamma_{k+1} = B_{k+1} (Q_k / \Delta t) B_{k+1}^T \]  

(7)

Then, the updated factors \((\hat{U}_{k+1}, \hat{D}_{k+1})\) are obtained in upper triangular and diagonal forms, respectively, by performing a Modified Weighted Gram-Schmidt orthogonalization on the matrix \( W_{k+1} \), where its columns are weighted by the diagonal matrix \( \hat{D}_k \).

The calculation of \( \Phi(t_{k+1}, t_k) \) requires the integration of \( nxn \) equations in addition to the \( n \)-state equations. The determination of \( B_{k+1} \) necessitates an \( nxn \) quadrature. Therefore, the total number of equations to be integrated is \( 2(nxn) + n \). The UD\(_T^T\) formulation proposed in Ref. 7 approximates \( B_{k+1} \) by an analytical trapezoid-rule integration which eliminates the \( nxn \) quadrature. The error introduced by this approximation can be neglected if the propagation interval \( (t_{k+1} - t_k) \) is small. The effect of error accumulated over long prediction intervals, during loss of tracking or data drop-outs, must be considered to ascertain the accuracy of this approximation.

The matrix multiplication, \( \hat{U}_k \), combined with the creation of the augmented matrix \( W_{k+1} \), destroys the triangularity of the square root covariance matrix. The application of the modified Gram-Schmidt orthogonalization procedure is required to retriangularize the UD factors.
at the time each measurement is processed. The added computational burden of this orthogonalization at each observation point could be eliminated if the square root covariance matrix were propagated without the loss of its triangularity.

In this investigation, a method is proposed which allows the integration of the continuous state-error covariance differential equations in square root form. The derivation follows the approach used in Ref. 8, but the results are based on the $P = UDU^T$ decomposition. The new algorithm can be combined with a triangular measurement update algorithm to obtain a complete square root estimation algorithm for which square roots are avoided. In addition, the effects of state process noise are included without approximation.

The Square-Root Propagation Equations in Triangular Form

The differential equation for propagating the state error covariance matrix can be expressed as

$$
\dot{\bar{P}}(t) = A(t)\bar{P}(t) + \bar{P}(t)A^T(t) + Q(t)
$$

where $\bar{P}(t)$ is the a priori state error covariance matrix, $A(t)$ is the $n \times n$ linearized dynamics matrix, and $Q(t)$ is the process noise covariance matrix. Each of the matrices in Eq. (8) is time-dependent in the general case. However, for simplicity, the time dependence will not be noted specifically in the following discussion.

If the following definitions are used,

$$
\bar{P} = UDU^T ; \quad Q = Q/2
$$

and if the first part of Eq. (9) is differentiated with respect to time and substituted into Eq. (8), the results can be rearranged to form
\[
(\dot{U}D + \ddot{U}D - \dddot{Q}U^T - \Lambda \dddot{U}D)U^T + \dddot{U}(\dot{U}U^T + \dddot{U}U^T + \dddot{U}U^T - \dddot{U}U^T A) = 0.
\] (10)

Noting that the first term of (10) is the transpose of the second term, and making the following definition:

\[
C(t) = (\dot{U}D + \ddot{U}D - \dddot{Q}U^T - \Lambda \dddot{U}D)U^T,
\] (11)

one obtains

\[
C(t) + C^T(t) = 0
\] (12)

Relation (12) requires that \(C(t)\) be either the null matrix or, more generally, skew symmetric.

Equation (11) can be simplified by selectively carrying out the multiplication of the \(-\dddot{Q}U^T\) term by \(U^T\) to yield, after terms are rearranged,

\[
(\dddot{U}D + \dddot{U}D - \dddot{A}U^D)U^T = \dddot{Q} + C(t) \equiv \dddot{C}(t)
\] (13)

Equation (13) defines the differential equations for \(U\) and \(D\) to the degree of uncertainty in \(C(t)\). Since the unknown matrix \(C(t)\) is skew symmetric, there exist \(n(n-1)/2\) unknown scalar quantities in Eq. (13). The problem considered here is one of specifying the elements of \(C(t)\) so that \(U\) is maintained in triangular form during the integration of Eq. (13). (The derivation pursued here assumes that \(U\) is lower triangular and \(D\) is diagonal, although an algorithm for an upper triangular \(U\) can be obtained as easily.) The following definitions are made to facilitate the solution to the problem posed above.

\[
T \equiv A \dddot{U}D ; \quad M \equiv \dddot{U}D + \dddot{U}D - T
\] (14)

With these definitions, (13) is expressed as

\[
\dddot{M}U^T = \dddot{C} = \dddot{Q} + C(t)
\] (15)
Since $\tilde{U}$ and $\bar{U}$ in Eq.(13) are lower triangular, and since from (12), $C(t)$ is skew symmetric, several observations can be made regarding Eq.(15). There are $n(n-1)/2$ unknown elements in $\bar{C}$. The products $\tilde{U}\bar{U}$ and $\bar{U}\tilde{U}$ are lower triangular creating $n(n+1)/2$ unknowns. Therefore, the $n \times n$ system of equations (15) has $[n(n-1)/2 + n(n+1)/2] = n \times n$ unknowns which can be determined uniquely.

An expansion of Eq.(15) into matrix elements indicates the method of solution.

\[
\begin{pmatrix}
M_{11} & -T_{12} & \cdots & -T_{1n} \\
M_{21} & M_{22} & \cdots & -T_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
M_{n1} & M_{n2} & \cdots & M_{nn}
\end{pmatrix} \begin{pmatrix}
1 & \tilde{U}_{21} & \cdots & \tilde{U}_{n1} \\
& 1 & \cdots & \tilde{U}_{n2} \\
& & \ddots & \vdots \\
& & & 0 & 1
\end{pmatrix}
\]

\[
\begin{pmatrix}
\bar{q}_{11} & -C_{12} & \cdots & -C_{1n} \\
C_{21} & \bar{q}_{22} & \cdots & -C_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
C_{n1} & C_{n2} & \cdots & \bar{q}_{nn}
\end{pmatrix}
\]

In Eq.(16), $\bar{Q}$ is assumed to be a diagonal matrix with elements $\bar{q}_{ii} = q_{ii}/2$; $i=1, \ldots, n$. (This assumption can be generalized to allow other non-zero terms in the $\bar{Q}$ matrix with only a slight increase in algebraic complexity.) Each row of the upper triangular
portion of the \( C \) matrix in Eq.(16) is determined as the product of the corresponding row of the \( M \) matrix with the appropriate column of the \( \hat{U}^T \) matrix. After an upper triangular row of \( C \) is determined, the condition from Eq.(12) that \( C_{ij} = -C_{ji} \) \((i=1, \ldots, n; j=1, \ldots, i-1)\) is invoked to evaluate the corresponding lower triangular column of \( C \). Then a column of the lower triangular elements of \( M \) can be evaluated. Once the elements of the \( M \) matrix are determined, the next row of the upper triangular \( C \) elements can be computed along with a column of the \( \hat{U} \) and \( \hat{D} \) elements. This process is repeated until all \( \hat{U} \) and \( \hat{D} \) values are determined. The implementation of this approach proceeds as follows.

From Eqs. (13) and (14) one can write

\[
M + T = \hat{U}\hat{D} + \hat{D}
\]

(17)

The expansion of (17) in summation notation gives

\[
M_{ij} + T_{ij} = \sum_{k=1}^{n} \hat{U}_{ik} \hat{D}_{kj} + \sum_{k=1}^{n} \frac{\hat{U}_{ik} \delta_{kj}}{2}
\]

\(i=1, \ldots, n; j=1, \ldots, i\) \hspace{1cm} (18)

But, since \( \hat{D} \) is diagonal, (18) becomes

\[
M_{ij} + T_{ij} = \hat{U}_{ij} \delta_{jj} + \frac{\hat{U}_{ij} \delta_{jj}}{2}
\]

\(i=1, \ldots, n; j=1, \ldots, i\) \hspace{1cm} (19)

For \( i=j \), \( \hat{U}_{ij} = 1 \) and \( \hat{U}_{ij} = 0 \). Therefore, (19) becomes

\[
\delta_{ii} = 2(M_{ii} + T_{ii}) \quad i=1, \ldots, n
\]

(20)
For \( i > j \), (19) is rearranged to obtain the differential equation

\[
\dot{\bar{U}}_{ij} = (\bar{M}_{ij} + \bar{T}_{ij} - \frac{\bar{U}_{ij} \bar{d}_{jj}}{2}) / \bar{d}_{jj}
\]

\( i=2, \ldots, n ; \ j=1, \ldots, i-1 \) \hspace{1cm} (21)

Equations (20) and (21) are the forms of the differential equations to be employed in the derivative routine of a numerical integrator. The elements \( T_{ij} \) and \( M_{ij} \) are computed as defined in Eq.(14). The pertinent equations can be combined to obtain the following algorithm.

**Triangular Square-Root Propagation Algorithm**

Given the elements of the square root state error covariance in lower triangular \( \bar{U}\bar{D} \) form, \( \bar{Q} = Q/2 \), and \( \Lambda(t) \), the differential equations \( \dot{\bar{U}}_{ij} \) and \( \dot{\bar{d}}_{ii} \) can be computed as follows:

\[
T_{ij} = \sum_{k=j}^{n} A_{ik} \bar{U}_{kj} \bar{d}_{jj} \hspace{1cm} i=1, \ldots, n ; \ j=1, \ldots, n \hspace{1cm} (22)
\]

\[
\bar{C}_{ij} = \sum_{k=1}^{i-1} M_{ik} \bar{U}_{jk} - \sum_{k=i+1}^{j} T_{ik} \bar{U}_{jk} \hspace{1cm} i=1, \ldots, n ; \ j=i+1, \ldots, n \hspace{1cm} (23)
\]

\[
M_{ii} = \frac{q_{ii}}{2} - \sum_{k=1}^{i-1} M_{ik} \bar{U}_{ik} \hspace{1cm} i=1, \ldots, n \hspace{1cm} (24)
\]

\[
M_{ij} = -\bar{C}_{ij} - \sum_{k=1}^{j-1} M_{ik} \bar{U}_{jk} \hspace{1cm} i=2, \ldots, n ; \ j=1, \ldots, i-1 \hspace{1cm} (25)
\]

\[
\dot{\bar{d}}_{ii} = 2(M_{ii} + T_{ii}) \hspace{1cm} i=1, \ldots, n \hspace{1cm} (26)
\]

\[
\dot{\bar{U}}_{ij} = (\bar{M}_{ij} + \bar{T}_{ij} - \frac{\bar{U}_{ij} \bar{d}_{jj}}{2}) / \bar{d}_{jj} \hspace{1cm} i=2, \ldots, n ; \ j=1, \ldots, i-1 \hspace{1cm} (27)
\]
The propagation algorithm summarized in Eqs. (22) through (27) can be combined with the algorithm for incorporating an observation to obtain a complete sequential estimation algorithm in which the covariance matrices $\hat{P}$ and $\hat{P}$ are replaced by the factors $(\hat{U}, \hat{D})$ and $(\hat{\bar{U}}, \hat{\bar{D}})$, respectively. The algorithm given in the Appendix assumes that only a single scalar observation is processed at each observation epoch; however, the algorithm is applicable to the case of multiple observations at a given epoch, if the observation errors are assumed to be uncorrelated.

Numerical Comparison

In the following numerical example, the two methods for propagation of the $\text{UDU}^T$ factored covariance matrix are compared to determine the relative computation speed and estimation accuracy. As a basis for determining their absolute performance, numerical results are obtained with the conventional Extended Kalman-Bucy filter using both Eqs. (1) and (8) as the bases for propagating the state error covariance matrix. The numerical comparisons are made by using each of the algorithms to process a set of simulated Global Positioning System (GPS) range and range-rate observations obtained by the LANDSAT-D spacecraft. A detailed discussion of the GPS and the associated navigation measurements is given in Ref. 12. Since the range measurement will require a precise measurement of the time interval between signal transmission from one of the GPS satellites to reception at the LANDSAT-D spacecraft, the clock error must be modeled and included as part of the overall state vector. The primary clock errors are the bias and drift.
The dynamic model used for the motion of LANDSAT and the associated model of the satellite's clock are combined to obtain a filter model which contains nine state variables. The nine components of the state vector are: position \((\mathbf{r}; \, 3\times1)\), velocity \((\mathbf{v}; \, 3\times1)\), clock bias \((b)\), clock drift \((n)\), and clock-drift model correlation parameter \((\beta)\). The differential equations defining these parameters in the filter are:

\[
\begin{align*}
\dot{\mathbf{r}} &= \mathbf{v} \\
\dot{\mathbf{v}} &= \mathbf{g} + \mathbf{a}_d + \mathbf{\xi}_v \\
\dot{b} &= n \\
\dot{n} &= \beta n + \xi_n \\
\dot{\beta} &= \xi_\beta
\end{align*}
\]

The stochastic processes, \(\mathbf{\xi}_v\), \(\xi_n\) and \(\xi_\beta\), are white noise forcing functions which are assumed to have the following statistics:

\[
E[\xi(t)]_i = 0 ; \quad E[\xi(t)\xi^T(\tau)]_i = \delta(t)\delta(t-\tau)
\]

where the subscript \(i\) indicates the appropriate member of the set \(\{\mathbf{v}, n, \beta\}\) and \(\delta(t-\tau)\) is the Dirac delta function.

For computational efficiency, the filter assumes simple models for the gravitational acceleration, \(\mathbf{g}\), and for the atmospheric drag acceleration, \(\mathbf{a}_d\). The geopotential model adopted for the filter is obtained by truncating the Goddard Earth Model (GEM7)\(^\text{11}\) to the fourth degree and order. The drag acceleration is calculated as
\[
\ddot{d} = -d\dot{y}_{rel} + \dot{y}_{rel} \tag{34}
\]

where the atmospheric density, \( \gamma \), is approximated by the exponential model:

\[
\gamma = \gamma_0 e^{-k(h-h_0)} \tag{35}
\]

The values of the drag model parameters assumed for this investigation are: \( h_0 = 840,000 \text{ m} \), \( \gamma_0 = 5.75 \times 10^{-15} \text{ kg/m}^3 \), \( k = 7.48 \times 10^{-6} \text{ m} \), and \( d = 1.18 \times 10^{-7} \text{ m}^2/\text{kg} \).

The estimates of the clock bias, \( b \), and drift, \( n \), are used to predict the true time of the user's clock, \( t \), by the equation

\[
t = T - b_o - n_o (t-t_o) \tag{36}
\]

where the subscript \((o)\) indicates the epoch of the last estimate of the parameters. The quantity \( T \) is the time as indicated by the LANDSAT-D clock.

The observations used for this study are pseudo range and pseudo range-rate measurements as observed by the LANDSAT-D satellite using the six satellites of the Phase I GPS constellation. The computer software used to simulate the observations as well as further details on the simulation procedure are discussed in Ref. 10.

The models used to generate the simulated measurements have the form:

\[
Y_{\rho} = \rho + (b_{\rho} - b_o) c + \varepsilon_{\rho} \tag{37}
\]

\[
Y_{\dot{\rho}} = \dot{\rho} + (b_{\dot{\rho}} - n_o) c + \varepsilon_{\dot{\rho}} \tag{38}
\]

where \( \rho \) and \( \dot{\rho} \) are the true values of range and range-rate between LANDSAT-D and a given GPS satellite, \( Y_{\rho} \) and \( Y_{\dot{\rho}} \) are the measured
values of the range and range-rate; \( b, n, b_s, \) and \( n_s \) are the biases and drifts in the satellite clock and in the GPS clocks, respectively; and \( c \) is the speed of light.

The measurement \( Y \) and \( Y' \) is processed by the LANDSAT-D navigation filter using the model

\[
Y = \rho + \rho + \xi_p
\]
\[
Y' = \rho + \rho + \xi_p
\]

That is, in the filter model the GPS clocks are assumed to be perfect and the total time error is assumed to be contained in the LANDSAT-D satellite clock. The interval between observations is assumed to be six seconds and observations from only one GPS satellite can be obtained during any six-second interval. The observations from the visible satellites are processed sequentially.

The GPS satellites are assumed to be in circular orbits \((e=0)\) about a point mass earth with inclinations of 6° and periods of 12 hours (43,200 sec). Three satellites are equally spaced on each of two orbital planes. The orbital elements are referenced to a coordinate system whose \( xy \)-plane is the earth's equator and whose \( xz \)-plane lies along the Greenwich meridian.

The epoch condition for LANDSAT-D was chosen so that the resulting simulated observations would accurately reflect the possible extremes of GPS satellite visibility. The epoch elements chosen are

\[
a = 7.086901 \times 10^6 \text{ m}, \quad e = 0.0001, \quad i = 98.218, \quad \Omega = 354.878, \quad \omega = 180°, \quad \text{and} \quad f(\text{true anomaly}) = -185°.
\]

The elements are specified at a GPS system time \( t_0 \). The epoch elements for GPS are specified at a system time of...
-7200 sec. The difference in initial epochs is included in the filter program's update of user and GPS states.

The values of the LANDSAT-D and GPS clock phase and frequency errors are simulated as the sum of three different error sources: a noise-free phase error with a polynomial form, an error due to exponentially correlated frequency noise, and a random walk bias error. The exact form of the error models and the coefficients used in the models are given in Ref. 10.

The numerical simulations of the filter performance were made with the following initial conditions:

<table>
<thead>
<tr>
<th>State</th>
<th>State Covariance</th>
<th>Noise Covariance</th>
</tr>
</thead>
<tbody>
<tr>
<td>X(1) = 7.046 x 10^6 m</td>
<td>P(1,1) = 6 x 10^6 m^2</td>
<td>Q(1,1) = 0</td>
</tr>
<tr>
<td>X(2) = -5.433 x 10^6 m</td>
<td>P(2,2) = 6 x 10^6 m^2</td>
<td>Q(2,2) = 0</td>
</tr>
<tr>
<td>X(3) = -6.120 x 10^5 m</td>
<td>P(3,3) = 6 x 10^6 m^2</td>
<td>Q(3,3) = 0</td>
</tr>
<tr>
<td>X(4) = 5.562 x 10^7 m/sec</td>
<td>P(4,4) = 1 x 10^7 (m/sec)^2</td>
<td>Q(4,4) = 1 x 10^{-6} m^2/sec^3</td>
</tr>
<tr>
<td>X(5) = -1.116 x 10^3 m/sec</td>
<td>P(5,5) = 1 x 10^4 (m/sec)^2</td>
<td>Q(5,5) = 1 x 10^{-6} m^2/sec^3</td>
</tr>
<tr>
<td>X(6) = 7.388 x 10^3 m/sec</td>
<td>P(6,6) = 1 x 10^4 (m/sec)^2</td>
<td>Q(6,6) = 1 x 10^{-6} m^2/sec^3</td>
</tr>
<tr>
<td>X(7) = 2.998 x 10^1 m</td>
<td>P(7,7) = 3.600 x 10^3 m^2</td>
<td>Q(7,7) = 0</td>
</tr>
<tr>
<td>X(8) = 5.996 x 10^{-1} m/sec</td>
<td>P(8,8) = 6. x 10^0 (m/sec)^2</td>
<td>Q(8,8) = 1 x 10^{-4} m^2/sec^3</td>
</tr>
<tr>
<td>X(9) = 5.550 x 10^{-4}</td>
<td>P(9,9) = 1 x 10^{-5}</td>
<td>Q(9,9) = 1 x 10^{-7}</td>
</tr>
</tbody>
</table>

The off-diagonal terms of the state-error covariance and noise covariance matrices are set to zero initially.
The data in Table 1 and Table 2 show the relative performance of the four algorithms when each processed a simulated observation data set with a duration of approximately 10000 sec. The five columns in each of the tables are, respectively: (1) the total CPU time required to perform measurement updates of the state and covariance, (2) the total CPU time required for propagating the state and covariance, (3) the total CPU propagation time normalized by the fastest CPU propagation time, (4) the total CPU time required for propagation and measurement updates (the sum of columns (1) and (2)), and (5) the RMS of the position magnitude errors and velocity magnitude errors over the duration of the simulation. All CPU times are listed in milliseconds. Position errors are given in meters and velocity errors are given in meters per second. The algorithms are ranked in the tables in order of increasing total computation time.

Time propagation is performed with a fixed-step modified Euler integrator, which requires two function evaluations per step. The integration step size for the data of Table 1 is six seconds, equal to the time interval between GPS observations. The data of Table 2 result from integration with a three-second step size.

The relations for the propagation of the time bias and drift as approximated by Eqs. (30), (31) and (32) have simple analytic solutions. This allows certain elements of the state transition matrix to be updated analytically. The implementation of the UD(U(0)) and EXP (S) algorithms has taken advantage of this simplification to reduce the number of numerically integrated differential equations below the theoretical value of n². The high degree of coupling in the covariance differential equations for the
(\dot{P}) and (\dot{U}, \dot{D}) algorithms does not permit a convenient reduction in the integration vector size. A total of \(n(n+1)/2\) covariance equations has been integrated in the simulations described here.

While the programming effort required to implement the (\dot{U}, \dot{D}) formulation will be greater because of the recursive nature of Eqs. (22) through (27), fewer computer storage locations will be required to execute this algorithm since the total number of equations involved in (22) through (27) is \(n(n+1)/2\). This value compares with the \((nxn)\) computer memory locations required to integrate and store the \(\dot{P}\) equations. Since there are some zeros in the \(\dot{P}\) equation, one can reduce the storage requirements of this method at the expense of added programming complexity. Further comparisons of the computer storage location requirements for these two algorithms are given in Ref. 10.

The numerical results shown in Table 1 indicate that the (\dot{U}, \dot{D}) algorithm is competitive with both the UDU\(^T\)(\dot{P}) and conventional formulations in terms of CPU times and estimation accuracy for this filtering problem. The (\dot{U}, \dot{D}) method is faster than the UDU\(^T\)(\dot{P}) algorithm for the six-second integration interval. Its position estimation error is lower than that for any of the other algorithms. For the three-second step size results in Table 2, the (\dot{U}, \dot{D}) algorithm remains competitive in terms of estimation accuracy, but is no longer as fast as the UDU\(^T\)(\dot{P}) algorithm. With the decrease in integration step size, the number of expensive (\dot{U}, \dot{D}) function evaluations has increased, but the number of time consuming orthogonalizations in the UDU\(^T\)(\dot{P}) algorithm remains the same. This factor causes the UDU\(^T\)(\dot{P}) algorithm to have a faster computation time, in this case. Again, for the three-second results, the (\dot{U}, \dot{D}) formulation yields the most accurate position estimate.
Both UDU$^T$ methods require higher total computation times than the EKF algorithms. This is not an unexpected result as square root filter formulations often incur computation time penalties as the price for increased numerical stability.

These numerical results were generated on a CDC6600 computer system. The relative performance of the algorithms will vary as a function of the computer system being used, the dynamic model assumed by the filter, the method and order of numerical integration, and the integration step size. The influence on performance of the latter two factors can be seen in the numerical results given in Ref. 10. The (U,D) method becomes more efficient as the total number of function evaluations within a given integration interval is decreased. Therefore, the choice of the time propagation method should depend on the formulation of the specific problem under consideration and on the supporting algorithms and computer system used to perform the calculations.

Conclusions

Based on the results presented in the previous discussion, it is concluded that, for the example problem considered here, the (U,D) algorithm is more efficient than the (U,D) algorithm based on the $\dot{\phi}$ propagation. Furthermore, the estimate obtained with the (U,D) formulation was more accurate than the estimate obtained by either the conventional EKF estimation algorithms or the (U,D)-$\dot{\phi}$ algorithm. The performance of the algorithms will be dependent on the computer architecture and software, the dynamic model assumed for the filter and the method used to perform the numerical integrations, and will vary as these factors change.
Table 1. Numerical Algorithm Comparison for the Modified Euler Integrator (Step Size = 6 seconds).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Meas. Update, ms</th>
<th>Time Update, ms</th>
<th>Norm. Time</th>
<th>Total Update, ms</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKF(\ddot{\gamma})</td>
<td>7.246</td>
<td>38.608</td>
<td>1.000</td>
<td>65.855</td>
<td>113.1</td>
</tr>
<tr>
<td>EKF(\dddot{\gamma})</td>
<td>8.057</td>
<td>42.792</td>
<td>1.108</td>
<td>60.849</td>
<td>120.3</td>
</tr>
<tr>
<td>UDU(\dddot{\gamma})</td>
<td>8.045</td>
<td>51.131</td>
<td>1.324</td>
<td>59.176</td>
<td>111.4</td>
</tr>
<tr>
<td>UDU(\dddot{\gamma})</td>
<td>8.216</td>
<td>61.296</td>
<td>1.588</td>
<td>60.512</td>
<td>117.6</td>
</tr>
</tbody>
</table>

Table 2. Numerical Algorithm Comparison for the Modified Euler Integrator (Step Size = 3 seconds).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Meas. Update, ms</th>
<th>Time Update, ms</th>
<th>Norm. Time</th>
<th>Total Update, ms</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKF(\ddot{\gamma})</td>
<td>7.981</td>
<td>64.688</td>
<td>1.000</td>
<td>72.669</td>
<td>118.5</td>
</tr>
<tr>
<td>EKF(\dddot{\gamma})</td>
<td>8.341</td>
<td>75.224</td>
<td>1.163</td>
<td>83.565</td>
<td>146.8</td>
</tr>
<tr>
<td>UDU(\dddot{\gamma})</td>
<td>7.867</td>
<td>82.872</td>
<td>1.281</td>
<td>90.739</td>
<td>146.8</td>
</tr>
<tr>
<td>UDU(\dddot{\gamma})</td>
<td>6.972</td>
<td>102.831</td>
<td>1.590</td>
<td>109.803</td>
<td>146.0</td>
</tr>
</tbody>
</table>
Appendix

The measurement update algorithm for the UDU* factorization has the following form. Using the observation \( Y_{k+1} = G(X_{k+1}, t_{k+1}) \), calculate:

\[
H_{k+1} = \{3G(\tilde{Y}_{k+1}, t_{k+1})/\tilde{Y}_{k+1}\} \quad (A.1)
\]

For \( i = 1 \to n \),

\[
\tilde{F}_i = H_i + \sum_{k=i+1}^{n} H_k U_{ki} \quad (A.2)
\]

\[
V_i = \tilde{d}_i \tilde{F}_i \quad (A.3)
\]

Set \( \beta_{n+1} = R_{k+1} \) (where \( R_{k+1} \) is the measurement noise) and calculate:

\[
\beta_i = \beta_{i+1} + V_i \tilde{F}_i \quad ; \ i = n \to 1 \quad (A.4)
\]

Calculate diagonal covariance elements:

\[
\hat{d}_i = \tilde{d}_1 + \beta_{i+1}/\beta_i \quad ; \ i = n \to 1 \quad (A.5)
\]

\[
\alpha = \beta_i \quad (A.6)
\]

For \( i = 2 \to n \), and \( j = 1 \to i-1 \), calculate:

\[
P_j = \tilde{P}_j / \beta_j \quad (A.7)
\]

\[
B_{ij} = V_i + \sum_{k=j+1}^{i-1} U_{ik} V_k \quad (A.8)
\]
Compute residual:

\[ y_{k+1} = y_{k+1} - C(\bar{x}_{k+1}, \tau_{k+1}) \]  

(A.10)

Calculate gain and update state:

\[ \tilde{K}_i = B_{i1} + \bar{U}_{i1}V_{1} \]  

(A.11)

\[ \hat{x}_i = \bar{x}_i + \tilde{K}_i y_{k+1} \]  

(A.12)
References


RELATIVE PERFORMANCE OF ALGORITHMS FOR AUTONOMOUS SATELLITE ORBIT DETERMINATION

By

Byron D. Tapley, G. Peters, and B. E. Schutz

University of Texas
Austin, Texas

AAS/AIAA Astrodynamics Specialist Conference

PROVINCETOWN, MASS/JUNE 25-27, 1979
Limited word size in contemporary microprocessors causes numerical problems in autonomous satellite navigation applications. Numerical error introduced in navigation computations performed on small wordlength machines can cause divergence of sequential estimation algorithms. To insure filter reliability, square root algorithms have been adopted in many applications. The optimal navigation algorithm requires a careful match of the estimation algorithm, dynamic model, and numerical integrator. In this investigation, different representations of these elements are evaluated to determine their relative performance for satellite navigation applications. Numerical simulations are conducted using the Phase I GPS constellation to determine the orbit of the LANDSAT-D satellite. Numerical comparisons are made of various square root filter formulations, and their dependence on the order of the integrator is examined.

Nomenclature

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ddot{a}_d$</td>
<td>atmospheric drag acceleration</td>
</tr>
<tr>
<td>b</td>
<td>satellite clock bias</td>
</tr>
<tr>
<td>c</td>
<td>speed of light</td>
</tr>
<tr>
<td>d</td>
<td>ballistic coefficient</td>
</tr>
<tr>
<td>$g$</td>
<td>gravitational acceleration</td>
</tr>
<tr>
<td>$G_p$</td>
<td>computed range measurement</td>
</tr>
<tr>
<td>$G_r$</td>
<td>computed range-rate meas.</td>
</tr>
<tr>
<td>$h$</td>
<td>satellite altitude</td>
</tr>
<tr>
<td>$h_o$</td>
<td>scale height for density model</td>
</tr>
<tr>
<td>k</td>
<td>density model scaling factor</td>
</tr>
<tr>
<td>n</td>
<td>satellite clock drift</td>
</tr>
<tr>
<td>$\mathbf{r}$</td>
<td>inertial position vector</td>
</tr>
<tr>
<td>t</td>
<td>true time</td>
</tr>
<tr>
<td>$\mathbf{T}$</td>
<td>satellite clock indicated time</td>
</tr>
<tr>
<td>$\mathbf{V}$</td>
<td>inertial velocity vector</td>
</tr>
<tr>
<td>$v_{rel}$</td>
<td>magnitude of $\mathbf{v}_{rel}$</td>
</tr>
<tr>
<td>$\mathbf{v}_{rel}$</td>
<td>velocity vector relative to the atmosphere</td>
</tr>
<tr>
<td>$\xi$</td>
<td>correlation parameter for clock drift model</td>
</tr>
</tbody>
</table>

1 The research was supported by NASA Goddard Space Flight Center under Grant No. NSG5154.
2 Professor, Aerospace Engineering & Engineering Mechanics, The University of Texas at Austin.
3 Research Engineer, Dept. of Aerospace Engineering & Engineering Mechanics, The University of Texas at Austin.
4 Associate Professor, Aerospace Engineering & Engineering Mechanics, The University of Texas at Austin.
1. Introduction

The use of artificial earth satellites for accurate dissemination of time and frequency holds high potential for the development of an accurate and reliable autonomous navigation system [1]. Current satellite systems have demonstrated the ability to obtain global positioning of points, fixed on the surface of the earth, to an accuracy of two meters in three dimensions [2]. While the position error achieved by a dynamic navigator, i.e., one moving with respect to the earth's surface, would be considerably greater than two meters, this investigation indicates the potential inherent in satellite navigation methods.

The Global Positioning System (GPS)[3], which is being deployed currently, is designed to allow a user to satisfy real-time navigation requirements by the calculation of position and velocity using simultaneous pseudo range and pseudo range-rate measurements from several GPS satellites [4, 5, 6]. The requirements for determining the orbits of low altitude satellites in near real-time, coupled with the need for increased accuracy, generates an interest in evaluating the GPS as a means for satisfying satellite orbit determination requirements. With the development of compact low-power computers and atomic clocks, the ability to perform the satellite navigation function on-board the spacecraft in an autonomous navigation mode is an attractive alternative to telemetering the GPS range and range-rate measurements to the ground for processing by a ground-based orbit determination program [6].

Allowable computer storage and execution times will place constraints on the model and the algorithms which can be selected to estimate the satellite's state. To minimize the storage requirements and achieve a real-time state estimate, the estimate of the satellite orbit will be performed on-board, sequentially, using a Kalman-Bucy filter [7]. One problem which must be considered if a sequential data processing method is used is the problem of filter divergence [8]. The divergence occurs due to either (1) dynamic or measurement model error, or (2) numerical errors introduced during the computation process. Since most computers for autonomous satellite navigation will have a short wordlength, this second cause of divergence will be of considerable importance.

The problem of filter divergence has led to a number of studies aimed at the development of stable estimation algorithms. The square-root measurement update algorithm proposed by Potter [9,10] has been used in a number of applications to prevent divergence caused by a computed non-positive definite covariance matrix. The algorithm proposed by Carlson [11]
allows the measurement update to be accomplished in an efficient manner by maintaining a triangular square root covariance matrix. In [12], a triangular decomposition which is free of square root operations is proposed. The "square root free" algorithm is reported to be faster than the Carlson-Cholesky algorithm. The square root covariance matrix must be maintained in triangular form during the time propagation if these two algorithms are to be applicable. Usually, the square root covariance matrix time update destroys the matrix triangularity and a retriangularization at each potential observation epoch must be employed. This procedure requires an additional computation effort and an associated computation time penalty. A recent development by Tapley, et al [13,14], in which the time update of the square root covariance matrix is maintained in lower triangular form throughout the entire estimation process, can be adopted to obtain a complete triangular square root estimation algorithm.

The objective of this investigation is to evaluate the performance of the various square root algorithms in performing on-board satellite orbit determination using the Global Positioning System. The evaluation is based on a comparison of the computation time and the state estimate accuracy. The effect of the numerical integration method on the estimate accuracy is considered also.

2. Filter Model

The models of the satellite dynamics and of the observation state relation have a critical impact on the computer storage requirement and on the execution time for the navigation algorithm. If the models are too complex, unacceptably large storage requirements and computation times will occur. However, if the models are too simple, the accuracy of the navigation estimate will be degraded below an acceptable value. The following model is selected as a compromise between the requirements for accuracy and efficiency.

Dynamic Equations. The dynamic model for the motion of the user satellite and the associated model of the satellite clock behavior are combined to obtain a filter model which contains nine state variables. The nine components of the state vector are: position (\( \mathbf{r} \)); velocity (\( \mathbf{v} \)); clock bias (\( b \)); clock drift (\( n \)); and clock-drift model correlation parameter (\( \beta \)). The differential equations defining these parameters in the filter are:

\[
\begin{align*}
\frac{d\mathbf{r}}{dt} &= \mathbf{v} \\
\frac{d\mathbf{v}}{dt} &= \mathbf{g} + \mathbf{a}_d + \mathbf{a}_v \\
\dot{b} &= n \\
\dot{n} &= \beta n + \xi_n \\
\dot{\beta} &= \xi_\beta
\end{align*}
\]

where (\( ' \)) = \( d(\cdot)/dT \), e.g., the independent variable is the satellite clock time.
The stochastic processes, $\xi_\nu$, $\xi_n$ and $\xi_\theta$, are white noise forcing functions which are assumed to have the following statistics:

$$
\begin{align*}
E[\xi(t)]_i &= 0 \\
E[\xi(t)\xi^T(t)]_i &= Q_i(t)\delta(t-t')
\end{align*}
$$

where the subscript $i$ indicates the appropriate member of the set ($\nu,n,\theta$).

The filter uses relatively simple models for gravitational force, $\ddot{g}$, and the atmospheric drag forces, $\dot{a}_d$. The geopotential model adopted for the filter is obtained by truncating the Goddard Earth Model (GEM) 7 [15] to the fourth degree and order. The drag acceleration is calculated as

$$
\ddot{a}_d = -d\nu \nabla^2 \nu_{rel}
$$

where the atmospheric density, $\gamma$, is approximated by the exponential model:

$$
\gamma = \gamma_0 e^{-k(h-h_0)}
$$

The values of the drag model parameters assumed for this investigation are: $h_0 = 840,000$ m, $\gamma_0 = 5.74 \times 10^{-14}$ kg/m$^3$, $k = 7.58 \times 10^{-6}$ m, and $d = 1.18 \times 10^{-2}$ m$^2$/kg.

The clock bias, $b$, and drift, $n$, estimated by the filter, are used to predict the true time of the user's clock by the equation

$$
t = T - b_0 - n_0(t-t_{i0})
$$

where the subscript $(o)$ indicates the epoch of the last evaluation of the parameters.

Variational Equations. Linear variational equations are required to implement the sequential estimation algorithm [8]. The equations are derived by the linearization of Eqs. (1) through (5). Further details can be found in [15].

Observation State Relation. The observation types processed during the study are pseudo range and pseudo range-rate. The actual measurements can be expressed, mathematically, as:

$$
\begin{align*}
\rho_m &= \rho + \Delta\rho_m + \xi_{\rho} \\
\dot{\rho}_m &= \dot{\rho} + \Delta\dot{\rho}_m + \xi_{\dot{\rho}}
\end{align*}
$$

where $\rho_m$ and $\dot{\rho}_m$ are the measured values of the range and range-rate, $\rho$ and $\dot{\rho}$ are the actual geometric range and range-rate, $\Delta\rho_m$ and $\Delta\dot{\rho}_m$ are range and range-rate biases due to errors in the user's clock and the GPS satellite's clock, and the $\xi$ are zero mean white noise sequences with known variances.
The modeled values of the measurements have the form

\[ G_p = \rho_c + (b - b_s)c \]  
(12)

\[ G'_p = \dot{\rho}_c + (n - n_s)c \]  
(13)

where \( \rho_c \) and \( \dot{\rho}_c \) are the computed values of range and range-rate, \( b, n, b_s \) and \( n_s \) are the predicted biases and drifts in the satellite clock and in the GPS clocks, based on previous measurements, and \( c \) is the speed of light.

The linearized observation-state matrix \( H \) is computed by taking partial derivatives of \( G_p \) and \( G'_p \) with respect to the state. The complete expressions for \( G_p \) and \( G'_p \), along with the partial derivatives, are given in [15].

3. Observation Simulation Model

Observations generated for this study are pseudo range and pseudo range-rate measurements as observed by the LANDSAT-D satellite using the six satellites of the Phase I GPS constellation. The computer software used to generate the observations is a modification of the program developed by Kruczynski [5]. Further details on the software modifications, which were made to simulate the orbit of the LANDSAT satellite, are given in [15].

The simulation philosophy was to produce a physically realizable set of data points against which the filters could be tested and evaluated. Simplified models of the GPS satellites were used to reduce computer time requirements.

The description of the observation generation program can be broken into four basic areas: (1) simulation of GPS satellite motion, (2) simulation of LANDSAT-D motion, (3) simulation of clocks, and (4) simulation of the measurement process.

**Simulation of GPS Satellite Motion.** For simplicity, the GPS satellites are assumed to move in circular orbits around a point mass earth. The GPS satellite motion can be determined, then, by using a closed form solution and a set of Keplerian elements defined at some specified epoch. This approximation will not have a significant impact on the results presented here because of the relatively short time interval involved in the simulation.

The epoch orbital elements for the Phase I GPS constellation are the following:

<table>
<thead>
<tr>
<th>Satellite</th>
<th>Long. of Asc. Node (Deg)</th>
<th>Mean Anomaly (Deg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-130.</td>
<td>0.</td>
</tr>
<tr>
<td>2</td>
<td>-130.</td>
<td>40.</td>
</tr>
<tr>
<td>3</td>
<td>-130.</td>
<td>80.</td>
</tr>
</tbody>
</table>
The GPS satellites are assumed to be in circular orbits \( (a = 0) \) with inclinations of 63\(^\circ\) and periods of 12 hours (43,200 sec). These elements are referenced to a coordinate system whose xy-plane is the earth's equator and whose xz-plane lies along the Greenwich Meridian.

Simulation of LANDSAT-D Motion. The force model used to simulate the LANDSAT-D motion contains the effects of the earth's non-spherical mass distribution and the effects of atmospheric drag. The gravitational accelerations for the observation simulations are obtained using the GEM7 geopotential model truncated at order and degree 8. The drag acceleration is computed by using Eqs. (7) and (8), with the same set of constants as specified previously.

The epoch conditions for LANDSAT-D and the GPS satellites were chosen so that the resulting simulated observations would accurately reflect the possible extremes of GPS satellite visibility. The epoch elements chosen for LANDSAT-D are:

\[
\begin{align*}
a &= 7.086901 \times 10^6 m \\
e &= 0.001 \\
i &= 98.181 \\
\Omega &= 354.878 \\
\omega &= 180^\circ \\
f (\text{true anomaly}) &= -185^\circ
\end{align*}
\]

These elements are specified at the GPS system time \( t = 0 \). The epoch elements previously listed for the GPS constellation are specified at the GPS system time \( t = -7200 \) sec. This difference in initial epoch is included in the simulation program's updates for the user and GPS states.

Simulation of Clocks. The values of the LANDSAT-D and GPS clock phase and frequency errors can be calculated as the sum of three different error sources: a noise-free phase error with a polynomial form \( (\xi_1) \); an error due to exponentially correlated frequency noise \( (\xi_2) \); and a random walk bias error \( (\xi_3) \).

The polynomial error term is expressed mathematically as:

\[
\xi_1(t) = a_2 + a_3(t-a_1) + (a_4/2)(t-a_1)^2 \quad (14)
\]

The exact form of the error models and the coefficients used in the models are given in [5].

<table>
<thead>
<tr>
<th>Satellite</th>
<th>Long. of Asc. Node (Deg)</th>
<th>Anomaly (Deg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>110.</td>
<td>40.</td>
</tr>
<tr>
<td>5</td>
<td>110.</td>
<td>80.</td>
</tr>
<tr>
<td>6</td>
<td>110.</td>
<td>120.</td>
</tr>
</tbody>
</table>
Simulation of Measurement Process. To generate the observations, the GPS satellites are "sampled" every six seconds in ascending numerical order. Upon calculation of the geometric range vector between the LANDSAT-D and a GPS satellite, a measurement is accepted if the GPS satellite is no more than 20° below the LANDSAT-D local horizontal. If a satellite is rejected because of this geometric constraint, the next higher numbered satellite is sampled in the same manner. The procedure is repeated until a "visible" GPS satellite is found. If none of the six satellites is considered visible at a particular time, no measurement is taken. Then, all satellite vehicles are propagated forward six seconds and the procedure is repeated. A random number of measurements is rejected in an effort to simulate measurement losses due to actual system problems such as periodic failures in signal acquisition or bad data in the GPS transmission. A Gaussian error term is added to each of the geometric observations to account for purely random anomalies in the measurement process. The standard deviations for the random errors are: $\sigma_d = 2.0$ m, and $\sigma_u = 0.2$ m/sec. If no satellites are visible at a particular time, the user's clock errors and position and velocity magnitudes are recorded on a file to be used to compute navigation errors during data gaps.

4. Filter Algorithms

The filter used to process the GPS measurements will consist of two major segments. These segments are: (1) the measurement update segment, and (2) the time propagation segment. The measurement update segment receives the observations at a given time epoch and processes these observations to obtain an updated estimate of the state. The propagation segment maps the estimate and the associated state error covariance matrix forward in time to the next observation epoch. For each measurement update algorithm, there are two propagation algorithms which can be considered. The primary difference in the propagation algorithms is determined by whether one integrates the state transition matrix for propagating the state error covariance matrix or whether the differential equation for the state error covariance matrix is integrated directly. The filter algorithms compared in this investigation are:

(1) The Extended Kalman-Bucy Filter [8,10]
(2) The Carlson-Cholesky Filter [11,13]
(3) The Potter Filter [9,10]
(4) The UDU Filter [12,14]

The state error covariance matrix can be based on the following set of differential equations:

$$\dot{P}(t) = A(t)P(t) + P(t)A^T(t) + Q(t) \quad (15)$$

where $A(t) = [\partial F(X,t)/\partial X]$, the $n \times 1$ state vector, $X(t)$ is defined to have the components $X^T(t) = [r^T, v^T, b, n, \varepsilon]$, $F(X,t)$ is an $n \times 1$ vector whose components are the right-hand sides of Eqs. (1) through (5), and $[ ]^*$
indicates that the elements of the matrix are evaluated on the reference solution $X^*(t)$. Alternately, the covariance matrix can be propagated by using the state-transition matrix, $\Phi(t,t_k)$, where $\Phi(t,t_k) = A(t)\Phi(t,t_k)$; $\Phi(t_k,t_k) = I$. The integral of Eq. (15) can be expressed, then, as:

$$
\dot{P}_{k+1} = \Phi(t_{k+1},t_k)P_k \Phi^T(t_{k+1},t_k) + \int_{t_k}^{t_{k+1}} \Phi(t,\tau)Q(\tau)\Phi^T(t,\tau)d\tau 
$$

(16)

Rather than evaluate the integral in Eq. (16), an average value, $\Gamma_k$, is used, where

$$
\Gamma_k = \text{avg} \int_{t_k}^{t_{k+1}} \Phi(t,\tau)Q(\tau)\Phi^T(t,\tau)d\tau 
$$

(17)

Using Eq. (17), Eq. (16) can be expressed as:

$$
\dot{P}_{k+1} = \Phi(t_{k+1},t_k)P_k \Phi^T(t_{k+1},t_k) + \Gamma_k 
$$

(18)

The square root estimation algorithms can be based on either Eq. (15) or Eq. (18). In the numerical simulations described in the next section, methods based on both approaches are compared. The use of Eq. (15) allows a triangular square root factorization for the covariance to be maintained during the propagation interval. Propagation with Eq. (18) will destroy triangularity and, after the propagation interval, special computation techniques are required to obtain a new square root covariance matrix. For the square root propagation algorithms based on Eq. (15), the relative advantage of maintaining the covariance matrix in triangular form is offset by a more complicated form for the governing differential equation.

Further discussion of the algorithms as well as the specific implementation used for this investigation is given in [15].

5. Algorithm Comparison

The numerical performance of the algorithms discussed in Section 4 was compared by conducting a series of computer simulations in which the algorithms were used to process GPS range and range-rate observations. Observations were simulated for a GPS system time interval of 10,000 sec. from the LANDSAT-D orbit. This is approximately 1.7 revolutions of LANDSAT-D. A history of the number of GPS satellites visible from LANDSAT-D, versus time, is shown in Fig. 1. It can be seen that the number of satellites visible varies from zero to six, the number of satellites in the Phase I GPS constellation.

The numerical results obtained with each filter are very similar to those shown in Figs. 2, 3, and 4. The plots in these three figures are, respectively, RSS position error versus time, RSS velocity error versus
time, and the error in the time estimate versus time. This solution was generated with the UDU(ϕ) algorithm and a modified Euler integrator with an integration step size of six seconds.

Two types of error growth are visible in each of the figures. First, there are two time periods in each of the plots during which large error growth occurs. Comparison of the three figures with Fig. 1 shows a coincidence between the occurrence of the large error and the periods when fewer than four GPS satellites are visible. The large errors result from the inability of the filter model to predict the state estimate through time periods of limited observation data.

The second type of error growth occurs gradually. After each period of low satellite visibility, the navigation error is reduced to a lower level. However, the average value of the lower-level navigation error increases during each successive period of good satellite visibility. Additional simulations run with algorithms other than the one used to generate Figs. 2, 3, and 4 produced nearly identical error plots. It is likely that the long-term error growth is caused by the influence of geopotential model error on the estimate of the LANDSAT-D clock parameters. The large errors result from the periodic reduction of GPS satellite visibility. Both error types require further study; however, since the primary purpose of this investigation is the comparison of the algorithm efficiencies, in-depth study of the causes of the large error growth and methods for their removal is deferred to a later investigation.

Operation Count Comparison. As an evaluation of the theoretical computational efficiency, operation counts of the number of numeric operations have been made for the seven different time update algorithms under consideration. The operations recorded are the additions (subtractions), multiplications, divisions, and square roots required to perform a single time update of the state error covariance matrix or the square root covariance matrix. The operation counts required to incorporate the measurements are discussed in [12].

Table 1 and 2 give the number of operations for the covariance time update of a system with an n-dimensional state vector, whose complete nxn-transition matrix is obtained by numerical integration. The dimension of the system's state noise covariance matrix is m. The counts are broken into three groups depending on whether the operations occur once per time update interval, once per integration step, or once per integration function evaluation.

The operation count per integration step depends on the type of numerical integrator being employed. Tables 1 and 2 give the values for a second-order Euler integrator with two function evaluations per step. The coefficients of the \( n^2 \) terms in the table will increase significantly for higher order integrators (roughly as a factor of \( k(k+1)/2 \), where \( k \) is the number of function evaluations).

If the assumption of an Euler integrator is maintained and the integration step is specified, then the three count groups can be combined to give the total number of operations for a covariance time update.
Assuming that a time update occurs every six seconds and the integration step size is six seconds, the total number of operation counts can be computed. The result is shown in Table 3.

The total counts show that direct covariance integration algorithms have fewer operations if all transition matrix elements are numerically integrated in the \((\Phi)\) algorithms. In this situation, the direct integration methods have fewer equations to numerically integrate, fewer operations in each function evaluation, and are not forced to retriangularize at the end of the update.

However, it is often the case that the solution to some of the elements of the transition matrix can be obtained analytically. In such a case, the number of terms to be numerically integrated can be reduced. A similar reduction in integration vector size has not proven feasible for the algorithm based on direct integration of the covariance matrix. Therefore, a full \(n(n+1)/2\) set of covariance elements must be integrated numerically.

The actual numerical operation counts for each time update algorithm are shown in Table 4. These values reflect the following set of assumptions:

1. The state vector size, \(n\), is 11; the measurement noise covariance vector, \(m\), has the dimension of 8. The eleven state filter is the maximum filter size used in these studies. For the 11-state filter, a two parameter model is used to estimate the drag coefficient [14].

2. Use of all possible analytical function matrix updates (elements updated analytically are those derived from the clock parameter differential equations).

3. The modified Euler integrator with two function evaluations per step is used.

4. An integration step size of six seconds is adopted.

The results can be combined into an equivalent addition count by weighting the multiplications, divisions and square roots by their relative execution times. The simulations have been performed on a CDC6600 computer system which has the following operation times weights:

- Add 1
- Mult 2.5
- Div 7.25
- Sq.Root 62.5

The use of these weights with the counts of Table 4 gives the operation counts in Table 5 expressed in equivalent numbers of additions. Total value of equivalent additions in Table 5 indicates the relative efficiency of the algorithms in performing covariance time updates under the assumptions described above. It can be observed from Table 5 that the
square root algorithm based on the $P=UDU^T$ transformation in combination with the $\dot{P}$ propagation compares quite favorably with the conventional extended Kalman-Bucy filter. Note that if the symmetric properties of Eq. (15) are used, the $\dot{P}$ operation count is substantially less than either of the other algorithms. Detailed investigations have indicated that there are numerical problems associated with integrating the $n \times (n+1)/2$ differential equations obtained by invoking the symmetry requirements on $\dot{P}$. This fact illustrates the point that the numerical stability is as important as numerical efficiency in any autonomous satellite application.

**Numerical Comparison.** In addition to the operation counts described in the previous section, specific numerical simulations were performed on the algorithms to determine their actual relative performance in performing navigation computations. Tables 6 through 9 contain the results obtained in these simulations. The numerical results were obtained using the following conditions for numerical integration of the appropriate differential equations.

1. Variable step (2)4 Runge-Kutta with the absolute single-step error tolerance of $10^{-2}$, and a relative single-step tolerance of $10^{-6}$,
2. Fixed step fourth order Runge-Kutta with a six-second step size,
3. Fixed step, second-order Euler integrator with a three-second step size, and
4. Fixed step, second-order Euler integrator with a step size of six seconds.

The four different integrators were used to show how the relative performance changes as a function of the order and method of integration, the integration step size, and the method by which the integration step is selected.

The data are presented in each of the tables in the following manner. The columns are, from left to right:

1. The algorithm used for covariance matrix propagation,
2. The total measurement update time,
3. The total propagation time,
4. The propagation time normalized by the lowest propagation time,
5. The total computation time for time and measurement updates,
6. The RSS of the position magnitude error,
7. The RSS of the velocity magnitude error.

The times are given in seconds; the position errors are expressed in meters, and velocity errors are given in meters per second. The algorithms are listed in order of increasing total computation times.
The results indicate that the algorithm performance has a strong dependence on the characteristics of the numerical integration algorithm. In general, the ($\Phi$) algorithms have lower propagation times than their directly integrated counterparts. The significant deviation from this rule is shown in the results of Table 9, where the direct integration methods show computation time advantages.

The other variations in performance of the algorithms are quite unexpected. Although the computation times of all algorithms decrease as the order of integration is lowered and the step size is increased, the estimation accuracies of the algorithms increase with these same changes in integration characteristics. In one case, the (UD) algorithm fails to complete a simulation with the fourth order integrator and a six-second integration step but runs successfully and competitively with the second-order integrator at the same step size.

To rule out the possibility of computer roundoff error as a cause of this anomaly, tests were conducted on the Euler integrator. With step sizes as low as 0.5 sec., the Euler algorithm approached the results of a well validated high order multistep integration code [16]. These results ruled out the likelihood of roundoff error or coding error as the cause for the anomalous characteristics in the estimation accuracy. The effect is believed to be caused by a complex interaction between the stability characteristics of the estimator and the integrator.

The relative performance of the algorithms in Table 9 leads to a second unexpected trend in the numerical results. The predicted relative performance based on the operation counts in Table 5 does not agree with the actual results obtained in Table 9. Specifically, the numerical performance of the UD algorithm is better than the relative performance predicted by the operation counts while the $\hat{W}$ is not as good. The cause for this discrepancy is thought to be coding overhead not accounted for in the operation count or the possible parallel multiplication capability of the CDC6600. This question requires further investigation.

6. Conclusions

Based on the results given in the previous section, several general conclusions can be drawn. First, the estimation algorithm performance has a strong dependence on the order and method used for integrating the differential equations involved in propagating the state estimate and the state estimate covariance matrix between observation epochs. This dependence has not been fully understood and requires further consideration. Based on the results presented here, the square root method based on the UD transformation in combination with the state transition matrix propagation approach appears to be the best overall square root method. However, for the Euler integrator using a six-second integration step, the best overall results were obtained with the UD algorithm. Finally, the single-step Euler numerical integration algorithm yields a more accurate and stable estimate than the fourth order Runge-Kutta algorithms.

In relating the results presented in this paper to the microprocessor environment, several factors should be remembered. Of greatest importance is the fact that the timing comparisons have been obtained under conditions...
which are different than those existing for on-board computer implementations. Table 3 and 4 are of most significance for the on-board application in which a high-level language, such as FORTRAN, will probably not be used. The overhead associated with FORTRAN is the most probable cause of the discrepancies between the performance comparisons in terms of operation count versus execution times. The on-board implementation will be in assembly language and should approach the operation count performance given in Tables 3 and 4, although other factors must be considered also.

References


Table 1. Operation Counts for (ϕ) Algorithms
Broken Down by Frequency.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Adds</th>
<th>Mults</th>
<th>Divs</th>
<th>Sq.Rts.</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKF(ϕ) per update</td>
<td>2n^3 + 2m</td>
<td>2n^3 + m + 1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>per step*</td>
<td>3n^2</td>
<td>2n^2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>per fcn.eval.</td>
<td>n^3</td>
<td>n^3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>UDU(ϕ) per update</td>
<td>1.5n^3 + 5n^2 + n + 5m + n^2 m</td>
<td>1.5n^3 + 2n^2 + 0.5n + m + (n^2 + n)m + 1</td>
<td>n-1</td>
<td>-</td>
</tr>
<tr>
<td>per step</td>
<td>3n^2</td>
<td>2n^2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>per fcn.eval.</td>
<td>n^3</td>
<td>n^3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CARL(ϕ) per update</td>
<td>1.5n^3 + n^2 + n^2 m + m</td>
<td>1.5n^3 + n^2 - 0.5n + n^2 m + m</td>
<td>n-1 + m</td>
<td>-</td>
</tr>
<tr>
<td>per step</td>
<td>3n^2</td>
<td>2n^2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>per fcn.eval.</td>
<td>n^3</td>
<td>n^3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>POTT(ϕ) per update</td>
<td>2n^3 + n^2 m + m</td>
<td>2n^3 + 0.5n^2 - 0.5n + n^2 m + m</td>
<td>n-1 + m</td>
<td>-</td>
</tr>
<tr>
<td>per step</td>
<td>3n^2</td>
<td>2n^2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>per fcn.eval.</td>
<td>n^3</td>
<td>n^3</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

*per step calculations assume second-order Euler integrator with two function evaluations.

Table 2. Operation Counts for (P) Algorithms
Broken Down by Frequency.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Adds</th>
<th>Mults</th>
<th>Divs</th>
<th>Sq.Rts.</th>
</tr>
</thead>
<tbody>
<tr>
<td>P per step</td>
<td>3(n^2 + n)/2</td>
<td>(n^2 + n)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>per fcn. eval.</td>
<td>n^3 + n^2 + n^2</td>
<td>n^3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ÜD per step</td>
<td>3(n^2 + n)/2</td>
<td>n^2 + n</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>per fcn. eval.</td>
<td>n + 4n + 2n - 3</td>
<td>n + 3n + n - 3</td>
<td>n^2 + n</td>
<td></td>
</tr>
<tr>
<td>W per step</td>
<td>3(n^2 + n)/2</td>
<td>n^2 + n</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>per fcn. eval.</td>
<td>n^3 + n^2</td>
<td>n^3 - 0.5n^2 + 0.5n</td>
<td>0.5(n^2 + n)</td>
<td>-</td>
</tr>
</tbody>
</table>
Table 3. Total Operation Per Time Update
Weighted by Frequency

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Additions</th>
<th>Multiplications</th>
<th>Divs.</th>
<th>Sq.Rts.</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKF((\phi))</td>
<td>(4n^3+3n^2+2m)</td>
<td>(4n^3+2n^2+m+1)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>UDU((\phi))</td>
<td>(3.5n^3+3.5n^2+n)</td>
<td>(3.5n^3+4n^2+5n)</td>
<td>(n-1)</td>
<td>-</td>
</tr>
<tr>
<td>CARLSON((\phi))</td>
<td>(3.5n^3+4n^2+n^2+m)</td>
<td>(3.5n^3+3n^2-5n)</td>
<td>(n-1+m)</td>
<td>(n+m)</td>
</tr>
<tr>
<td>POTTER((\phi))</td>
<td>(4n^3+3n^2+n^2+m)</td>
<td>(4n^3+2.5n^2-5n)</td>
<td>(n-1+m)</td>
<td>(n+m)</td>
</tr>
<tr>
<td>(\dot{p})</td>
<td>(2n^3+5n^2+3n+m^2)</td>
<td>(2n^3+2n^2+2n)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(\ddot{u})</td>
<td>(2n^3+9.5n^2+5.5n-6)</td>
<td>(2n^3+7n^2+3n-6)</td>
<td>(2(n^2+n))</td>
<td></td>
</tr>
<tr>
<td>(\dot{w})</td>
<td>(2n^3+3.5n^2+1.5n)</td>
<td>(2n^3+2n)</td>
<td>-(n^2+n)</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. Numerical Operation Counts,
GPS Problem.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Additions</th>
<th>Multiplications</th>
<th>Divisions</th>
<th>Sq.Roots</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKF((\phi))</td>
<td>3434</td>
<td>3455</td>
<td>12</td>
<td>-</td>
</tr>
<tr>
<td>UDU((\phi))</td>
<td>3880</td>
<td>4035</td>
<td>22</td>
<td>-</td>
</tr>
<tr>
<td>CARLSON((\phi))</td>
<td>3837</td>
<td>3824</td>
<td>30</td>
<td>19</td>
</tr>
<tr>
<td>POTTER((\phi))</td>
<td>4442</td>
<td>4429</td>
<td>30</td>
<td>19</td>
</tr>
<tr>
<td>(\dot{p})</td>
<td>3230</td>
<td>2792</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(\ddot{u})</td>
<td>3866</td>
<td>3536</td>
<td>264</td>
<td>-</td>
</tr>
<tr>
<td>(\dot{w})</td>
<td>3102</td>
<td>2684</td>
<td>66</td>
<td>-</td>
</tr>
</tbody>
</table>
Table 5. Numerical Operation Counts (GPS Problem). Equivalent Additions (CDC6600) for Modified Euler, Step Size = 6.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>EKF(\phi)</td>
<td>3434.</td>
<td>8637.5</td>
<td>87.</td>
<td>-</td>
<td>12156.5</td>
</tr>
<tr>
<td>UDU(\phi)</td>
<td>3880.</td>
<td>10085.0</td>
<td>159.5</td>
<td>-</td>
<td>14124.5</td>
</tr>
<tr>
<td>CARLSON(\phi)</td>
<td>3837.</td>
<td>9560.0</td>
<td>217.5</td>
<td>1187.5</td>
<td>14802.0</td>
</tr>
<tr>
<td>POTTER(\phi)</td>
<td>4442.</td>
<td>11072.5</td>
<td>217.5</td>
<td>1187.5</td>
<td>16919.5</td>
</tr>
<tr>
<td>\dot{p}</td>
<td>3230.</td>
<td>6980.0</td>
<td>-</td>
<td>-</td>
<td>10210.0</td>
</tr>
<tr>
<td>\ddot{u}</td>
<td>3866.</td>
<td>8840.</td>
<td>264.</td>
<td>-</td>
<td>14620.0</td>
</tr>
<tr>
<td>\dot{w}</td>
<td>3102.</td>
<td>6710.</td>
<td>478.5</td>
<td>-</td>
<td>10290.5</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Meas. Update</th>
<th>Time Update</th>
<th>Norm Time</th>
<th>Total Update</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKF(\phi)</td>
<td>8.172</td>
<td>67.377</td>
<td>1.000</td>
<td>75.549</td>
<td>163.4</td>
</tr>
<tr>
<td>EKF(\dot{p})</td>
<td>8.255</td>
<td>83.325</td>
<td>1.237</td>
<td>91.580</td>
<td>163.3</td>
</tr>
<tr>
<td>UDU(\phi)</td>
<td>8.110</td>
<td>86.241</td>
<td>1.280</td>
<td>94.351</td>
<td>163.2</td>
</tr>
<tr>
<td>CARL(\phi)</td>
<td>18.459</td>
<td>87.820</td>
<td>1.303</td>
<td>106.279</td>
<td>163.2</td>
</tr>
<tr>
<td>POTT(\phi)</td>
<td>9.981</td>
<td>97.158</td>
<td>1.442</td>
<td>107.139</td>
<td>163.4</td>
</tr>
<tr>
<td>CARL(\dot{w})</td>
<td>20.198</td>
<td>98.321</td>
<td>1.459</td>
<td>118.519</td>
<td>163.4</td>
</tr>
<tr>
<td>UDU(\ddot{u})</td>
<td>7.729</td>
<td>119.456</td>
<td>1.773</td>
<td>127.185</td>
<td>163.3</td>
</tr>
</tbody>
</table>
Table 7. Numerical Algorithm Comparison.  
Fourth Order Runge-Kutta, Step Size = 6 sec.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Meas. Update</th>
<th>Time Update</th>
<th>Norm Time</th>
<th>Total Update</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKF(\hat{\phi})</td>
<td>7.909</td>
<td>65.340</td>
<td>1.000</td>
<td>73.249</td>
<td>163.4</td>
</tr>
<tr>
<td>EKF(\hat{P})</td>
<td>9.144</td>
<td>77.024</td>
<td>1.179</td>
<td>86.168</td>
<td>163.3</td>
</tr>
<tr>
<td>UDU(\hat{\phi})</td>
<td>8.015</td>
<td>84.362</td>
<td>1.291</td>
<td>92.377</td>
<td>163.2</td>
</tr>
<tr>
<td>CARL(\hat{\phi})</td>
<td>19.361</td>
<td>84.784</td>
<td>1.298</td>
<td>104.145</td>
<td>163.2</td>
</tr>
<tr>
<td>POTT(\hat{\phi})</td>
<td>9.379</td>
<td>95.122</td>
<td>1.456</td>
<td>104.501</td>
<td>163.2</td>
</tr>
<tr>
<td>CARL(\hat{W})</td>
<td>19.045</td>
<td>96.933</td>
<td>1.484</td>
<td>115.978</td>
<td>163.4</td>
</tr>
<tr>
<td>UDU(\hat{\theta})</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

Table 8. Numerical Algorithm Comparison.  
Modified Euler, Step Size = 3 sec.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Meas. Update</th>
<th>Time Update</th>
<th>Norm Time</th>
<th>Total Update</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKF(\hat{\phi})</td>
<td>7.981</td>
<td>64.668</td>
<td>1.000</td>
<td>72.039</td>
<td>148.5</td>
</tr>
<tr>
<td>EKF(\hat{P})</td>
<td>8.341</td>
<td>75.224</td>
<td>1.163</td>
<td>83.556</td>
<td>146.8</td>
</tr>
<tr>
<td>UDU(\hat{\phi})</td>
<td>7.867</td>
<td>82.872</td>
<td>1.281</td>
<td>90.739</td>
<td>146.8</td>
</tr>
<tr>
<td>CARL(\hat{\phi})</td>
<td>19.110</td>
<td>83.625</td>
<td>1.293</td>
<td>102.735</td>
<td>146.8</td>
</tr>
<tr>
<td>POTT(\hat{\phi})</td>
<td>9.893</td>
<td>93.319</td>
<td>1.443</td>
<td>103.212</td>
<td>146.8</td>
</tr>
<tr>
<td>UDU(\hat{\theta})</td>
<td>5.972</td>
<td>102.631</td>
<td>1.590</td>
<td>109.803</td>
<td>146.0</td>
</tr>
<tr>
<td>CARL(\hat{W})</td>
<td>18.818</td>
<td>95.112</td>
<td>1.470</td>
<td>113.930</td>
<td>146.5</td>
</tr>
</tbody>
</table>

Table 9. Numerical Algorithm Comparison.  
Modified Euler, Step Size = 6 sec.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Meas. Update</th>
<th>Time Update</th>
<th>Norm Time</th>
<th>Total Update</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKF(\hat{\phi})</td>
<td>7.246</td>
<td>38.609</td>
<td>1.000</td>
<td>45.555</td>
<td>113.1</td>
</tr>
<tr>
<td>EKF(\hat{P})</td>
<td>8.057</td>
<td>42.792</td>
<td>1.108</td>
<td>50.849</td>
<td>120.3</td>
</tr>
<tr>
<td>UDU(\hat{\theta})</td>
<td>8.045</td>
<td>51.131</td>
<td>1.324</td>
<td>59.176</td>
<td>111.4</td>
</tr>
<tr>
<td>CARL(\hat{W})</td>
<td>19.402</td>
<td>42.886</td>
<td>1.111</td>
<td>62.288</td>
<td>126.3</td>
</tr>
<tr>
<td>UDU(\hat{\phi})</td>
<td>8.216</td>
<td>61.296</td>
<td>1.588</td>
<td>69.508</td>
<td>117.6</td>
</tr>
<tr>
<td>CARL(\hat{\phi})</td>
<td>19.122</td>
<td>62.404</td>
<td>1.616</td>
<td>81.526</td>
<td>117.6</td>
</tr>
<tr>
<td>POTT(\hat{\phi})</td>
<td>10.186</td>
<td>71.897</td>
<td>1.862</td>
<td>82.083</td>
<td>117.6</td>
</tr>
</tbody>
</table>
Figure 1. GPS Satellite Visibility vs. Time

Figure 2. RSS Position Error vs. Time
Figure 3. RSS Velocity Error vs. Time

Figure 4. System Time Error vs. Time
A Triangular Covariance Factorization for Sequential Filtering Algorithms
B. D. Tapley and J. G. Peters, University of Texas, Austin, Texas
Derivation of the Square-Root Propagation Equations in Triangular Form

The differential equation for propagating the state error covariance matrix can be expressed as

\[ \dot{P} = AP + PA^T + Q, \] (1)

where

\[ P \] is state error covariance matrix,
\[ A \] is linearized dynamics matrix,
\[ Q \] is state noise covariance matrix.

If the following definitions are used:

\[ \Phi = UDU^T; \quad \Omega = Q/2 \] (2)

and, if the first part of (2) is differentiated with respect to time and substituted into (1), the results can be rearranged to form

\[ \dot{\Phi} + \Phi \dot{D} + \frac{1}{2} \dot{D}^T \Phi + \Phi \dot{D}^T + \frac{1}{2} D^T \Phi = 0 \] (3)

Noting that the first term of (3) is the transpose of the second term, the following definition is made:

\[ C(t) + C^T(t) = 0 \] (4)

Solution (4) in form (3) is the result which states that the state error covariance matrix \( \Phi \) is symmetric and zero diagonal. \( C(t) \) is then the propagated state error covariance matrix.

By orthogonality of \( U \) and \( U^T \), any expansion of matrix elements of the \( \Phi^T \) term in (3) resulting from these transformations:

\[ \Phi^T = \Phi \] (5)

The problem now is to find the elements of \( C(t) \) to maintain it in triangular form. (This derivation assumes \( U \) is lower triangular and \( D \) is diagonal, although an algorithm for an upper triangular \( U \) can be obtained as easily.) The following definitions are made to facilitate analysis of the solution to (6):

\[ \tilde{T} = UDU^T \] (7)

With these definitions, (6) is expressed as

\[ \tilde{T} + C(t) = \Phi \] (6)

Since \( U \) and \( U^T \) in (6) are lower triangular, and since from (5), \( C(t) \) is skew symmetric, the following observations can be made regarding (6):

There are \( n(n-1)/2 \) unknown elements in \( C \). The products \( UDU^T \) and \( U^T \) are lower triangular creating \( n(n+1)/2 \) unknowns. Therefore, the \( n \times n \) system of equations (6) has \( n(n-1)/2 + n(n+1)/2 \) \( n \times n \) unknowns which can be determined uniquely.

An expansion of (6) into matrix elements indicates the method of solution. (In this expansion \( \Phi \) is assumed to be a diagonal matrix with elements \( q_{ij} = q_{ij}/2, i=1, \ldots, n \)).

Each row of the upper triangular portion of the \( C \) matrix is determined as the product of the corresponding row of the \( M \) matrix with the columns of
Given the elements of the square-root state error covariance in lower triangular DD form, \( \mathbf{Q} = \mathbf{Q}/2 \), and \( \mathbf{A}(t) \), the differential equations \( \dot{\mathbf{U}}_{ij} \) and \( \dot{\mathbf{C}}_{ij} \) can be computed as follows:

\[
\dot{\mathbf{U}}_{ij} = \left( \mathbf{M}_{ij} + \frac{1}{2} \mathbf{T}_{ij} \right) \mathbf{U}_{ij} + \frac{1}{2} \frac{\partial \mathbf{U}_{ij}}{\partial \mathbf{U}_{ij}} \nabla \mathbf{U}_{ij} \nabla \mathbf{U}_{ij} \\
\dot{\mathbf{C}}_{ij} = \left( \mathbf{M}_{ij} + \frac{1}{2} \mathbf{T}_{ij} \right) \mathbf{C}_{ij} + \frac{1}{2} \frac{\partial \mathbf{C}_{ij}}{\partial \mathbf{U}_{ij}} \nabla \mathbf{C}_{ij} \nabla \mathbf{C}_{ij} 
\]

(1) \( \mathbf{M}_{ij} + \frac{1}{2} \mathbf{T}_{ij} = \mathbf{U}_{ij} \sum_{k=1}^{n} \mathbf{U}_{ik} \mathbf{U}_{kj} \)

(2) \( \mathbf{M}_{ij} + \frac{1}{2} \mathbf{T}_{ij} = \mathbf{C}_{ij} \sum_{k=1}^{n} \mathbf{C}_{ik} \mathbf{C}_{kj} \)

(3) \( \mathbf{C}_{ij} = \left( \mathbf{M}_{ij} + \frac{1}{2} \mathbf{T}_{ij} \right) \mathbf{C}_{ij} \)

(4) \( \mathbf{U}_{ij} = \left( \mathbf{M}_{ij} + \frac{1}{2} \mathbf{T}_{ij} \right) \mathbf{U}_{ij} \)

(5) \( \dot{\mathbf{U}}_{ij} = \frac{\partial \mathbf{U}_{ij}}{\partial \mathbf{U}_{ij}} \nabla \mathbf{U}_{ij} \nabla \mathbf{U}_{ij} \)

(6) \( \dot{\mathbf{C}}_{ij} = \frac{\partial \mathbf{C}_{ij}}{\partial \mathbf{U}_{ij}} \nabla \mathbf{C}_{ij} \nabla \mathbf{C}_{ij} \)

Preliminary Numerical Results

The two methods for time propagation of the square-root covariance matrix in the CUC algorithm were evaluated to determine the relative computational speed and integration accuracy. The test problem chosen was a planar Keplerian orbit at an altitude of 800 km above the earth. This problem was chosen for the following reasons:

1. The problem's simplicity allows for quick implementation and computation time.
2. The existence of an analytical solution gives a standard for measuring the accuracy of the numerical integration.
3. The orbit chosen approximates that of proposed operational satellites (such as LANDSAT-3) which will use square-root algorithms in onboard navigation computers.

The two methods were evaluated by integrating the state and square-root covariance matrix equations for one revolution, position-velocity components, state covariance, and integration times were tabulated at the end of the revolution as well as at 1/4, 1/2, and 3/4 points in the orbit. Integration was performed with a Runge-Kutta algorithm of fourth order (RK4/5)). The time comparisons were made with variable stepsize integration at relative error tolerances of 10^-7, 10^-8, and 10^-9. Comparisons were also made for three fixed step sizes.

The standard for measuring the accuracy of the numerical integration of the covariance matrix was generated by integrating the state and square-root covariance equations at a tight tolerance with a high-order multi-step integrator. Covariance accuracy was measured as the relative error in the magnitude of the diagonal position and velocity covariance elements. The error was measured relative to the standard solution described at the beginning of the paragraph.

Numerical comparisons were performed on a CDC 6600/ 6600 computer using single precision arithmetic. Integration times and accuracies will vary with different machines.
In the actual operation of the time propagation algorithm with measurement updates, the frequency of measurement has an effect on the computation time required for propagation. For the simulation tested here, this is true. The test has shown that a high measurement rate can cause the allowed stepsize that the integrator may take. Additionally, whenever the update algorithm requires a retriangulation at the time of each measurement, a high measurement rate requires that this procedure be performed more often, increasing the computation time. For reasons of simplicity, the measurement rate effect was simulated by the time interval values specified for each call to the numerical integrator. The simulations were run with different integration intervals to simulate the effects of different measurement rates.

The initial conditions for numerical integration were:

State:

\[ x_1 = 0, \quad x_2 = 1, \quad x_3 = 1, \quad x_4 = 15, \quad x_5 = 0 \]

Covariance (initially diagonal):

\[ P_{X_1} = 9 \times 10^{-6}, \quad P_{X_2} = 1.2 \times 10^{-6}, \quad P_{X_3} = 9 \times 10^{-6}, \quad P_{X_4} = 1.2 \times 10^{-6} \]

Then, at time 0, \( P(0) = P(0) \).

The normal state covariances were:

\[ Q = \begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \]

The results in tables 1 through 5 show the amount of time spent in the test and the amount of variance for the test time intervals. These quantities are tabulated as functions of integration tolerance and simulation rate. The test runs were divided into two parts: the (1) and the square-root variance integration methods denoted by (1/2) and (1/4). All simulation time is tabulated in minutes.

Discussion of Numerical Results

Based on the tabulated data shown in tables 1 through 5, the results obtained using the (1) time update algorithm is superior to the (1/2) method in terms of computational efficiency for this particular test problem. Specifically, the following trends in relative performance have been deduced from the results:

1. For all measurement rates tested, the (1) algorithm appears superior in terms of simulation time. The difference in computation speed increases nonlinearly as integration tolerance becomes tighter.
2. For the fixed step sizes and for the measurement rates tested, the (1) algorithm appears faster integration time and higher accuracy.

Derivative routine call is cheaper for the (1) code compared to (1/2) for no less.

The (1/2) algorithm gives a higher relative accuracy at tight tolerances, but with a serious penalty in integration time. The most accurate solution results with the (1/2) equation being integrated at a single step relative error tolerance of 1%, but as measurement rates increase, the difference in relative performance of the two update methods becomes evident. The major remaining difference in integration time occurs at tight tolerances, and differences in accuracy occur at loose tolerances, and in fixed-step cases.

A margin of difference in the performance of the two algorithms was unexpected. The cause of the difference is, evidently, the significantly smaller step sizes required in the (1/2) integration in stepsize as small that the method's advantage of integrating fewer equations is lost.

This indicates that the (1/2) update would be more competitive in a problem where an external factor, such as high data rate, constrains the integration step size. Such a constraint would not allow the (1) method to use its ability to take larger integration steps at a given tolerance, and could force it to take steps of the order of magnitude of those used by the (1/2) code. Also of note is that the higher the measurement rate, the more often the (1) algorithm is forced to perform matrix normalizations, which increase computation time. The numerical results in the tables indicate that the large disparity in computational performance decreases as the measurement rate increases. It seems clear that the direction of search for an algorithm that the (1) step size method would be more efficient in fixed systems with measurement rates that allow larger integration time step size.

The integration time can be freely chosen by the user, but the (1/2) method is more efficient.

While the small integration steps appear to be the main reason for an efficient operation of the (1/2) method in this problem, three other factors influence integration speed and should be considered in further computational efficiency studies:

1. A perceived advantage of the (1/2) update is its requirement for numerically integrating only the equations of motion, as opposed to the (1) equations integrated by the (1) algorithm. For the test problem here, the number of equations to be integrated are 10 and 16, respectively. Derivative routine calls still require less computation time for the (1) method. Perhaps a larger state vector would be necessary for the (1/2) to show a time savings due to its smaller integration vector size. This result will be highly problem-dependent, but tests with larger state vectors would be enlightening.

2. The (1) algorithm presented here approximates the process noise with an analytic trapezoidal rule integration. This approximation removes an explicit integrator from the (1) algorithm. The (1/2)
method accounts for process noise exactly. In the simulations, the process noise approximation had no discernible effect on covariance accuracy. Prediction intervals were small. Intuitively, the process noise approximation will become less accurate as the measurement rate decreases, or if there is a long prediction interval with no measurements. The degree of accuracy degradation that would occur in an actual filtering problem with long prediction intervals has not been addressed here, but merits study. If the approximation were not to give the desired accuracy, an n x n quadrature would have to be added to the (Φ) algorithm. The associated increase in computational burden might improve the competitiveness of the (U, ̄U) update.

Table 1
Integration Times & Covariance Error
25-Second Data Rate

<table>
<thead>
<tr>
<th>TOLERANCE</th>
<th>Rev</th>
<th>10^{-6}</th>
<th>10^{-4}</th>
<th>10^{-2}</th>
<th>Fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1/4 U, ̄U</td>
<td>58558</td>
<td>6240</td>
<td>955</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9548</td>
<td>1246</td>
<td>751</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1/2 U, ̄U</td>
<td>73211</td>
<td>7698</td>
<td>1726</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>19063</td>
<td>2468</td>
<td>1496</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>3/4 U, ̄U</td>
<td>106970</td>
<td>11535</td>
<td>2495</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>28590</td>
<td>3671</td>
<td>2229</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>1 U, ̄U</td>
<td>147757</td>
<td>15503</td>
<td>3260</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>38078</td>
<td>4895</td>
<td>2942</td>
<td>-</td>
</tr>
</tbody>
</table>

Covariance Error (1 Revolution)

| Pos U, ̄U | 0 | 9.74x10^{-8} | 5.14x10^{-7} | - |
| Vel U, ̄U | 2.32x10^{-11} | 9.27x10^{-9} | 5.34x10^{-8} | - |
| Pos d | 1.62x10^{-4} | 1.61x10^{-4} | 1.51x10^{-4} | - |
| Vel d | 1.46x10^{-6} | 1.46x10^{-6} | 1.36x10^{-6} | - |

Table 2
Integration Times & Covariance Error
10-Second Data Rate

<table>
<thead>
<tr>
<th>TOLERANCE</th>
<th>Rev</th>
<th>10^{-6}</th>
<th>10^{-4}</th>
<th>10^{-2}</th>
<th>Fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1/4 U, ̄U</td>
<td>59640</td>
<td>6831</td>
<td>2075</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10232</td>
<td>1817</td>
<td>1820</td>
<td>1709</td>
</tr>
<tr>
<td></td>
<td>1/2 U, ̄U</td>
<td>74635</td>
<td>8723</td>
<td>3980</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20515</td>
<td>3594</td>
<td>3462</td>
<td>3441</td>
</tr>
<tr>
<td></td>
<td>3/4 U, ̄U</td>
<td>109016</td>
<td>12917</td>
<td>5904</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30854</td>
<td>5370</td>
<td>5391</td>
<td>5310</td>
</tr>
<tr>
<td></td>
<td>1 U, ̄U</td>
<td>148035</td>
<td>17468</td>
<td>7816</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>41090</td>
<td>7166</td>
<td>7170</td>
<td>6778</td>
</tr>
</tbody>
</table>

Covariance Error (1 Revolution)

| Pos U, ̄U | 0 | 8.49x10^{-8} | 2.99x10^{-7} | - |
| Vel U, ̄U | 8.34x10^{-8} | 2.99x10^{-7} | - |
| Pos d | 2.57x10^{-7} | 2.53x10^{-7} | 2.55x10^{-7} | 2.55x10^{-7} |
| Vel d | 2.34x10^{-7} | 2.32x10^{-7} | 2.32x10^{-7} | 2.32x10^{-7} |

Table 3
Integration Time & Covariance Error
5-Second Data Rate

<table>
<thead>
<tr>
<th>TOLERANCE</th>
<th>Rev</th>
<th>10^{-6}</th>
<th>10^{-4}</th>
<th>10^{-2}</th>
<th>Fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1/4 U, ̄U</td>
<td>61281</td>
<td>7923</td>
<td>3812</td>
<td>3688</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11237</td>
<td>3619</td>
<td>3599</td>
<td>3410</td>
</tr>
<tr>
<td></td>
<td>1/2 U, ̄U</td>
<td>76846</td>
<td>11728</td>
<td>7576</td>
<td>7499</td>
</tr>
<tr>
<td></td>
<td></td>
<td>22512</td>
<td>7251</td>
<td>7203</td>
<td>6980</td>
</tr>
<tr>
<td></td>
<td>3/4 U, ̄U</td>
<td>112956</td>
<td>16886</td>
<td>11404</td>
<td>11103</td>
</tr>
<tr>
<td></td>
<td></td>
<td>34125</td>
<td>10871</td>
<td>10836</td>
<td>10327</td>
</tr>
<tr>
<td></td>
<td>1 U, ̄U</td>
<td>153250</td>
<td>22182</td>
<td>15224</td>
<td>14810</td>
</tr>
<tr>
<td></td>
<td></td>
<td>45457</td>
<td>14487</td>
<td>14428</td>
<td>13668</td>
</tr>
</tbody>
</table>

Covariance Error (1 Revolution)

| Pos U, ̄U | 0 | 7.25x10^{-8} | 2.73x10^{-7} | - |
| Vel U, ̄U | 2.32x10^{-11} | 7.64x10^{-9} | 2.73x10^{-7} | - |
| Pos d | 6.25x10^{-4} | 6.25x10^{-4} | 6.25x10^{-4} | - |
| Vel d | 5.79x10^{-6} | 5.79x10^{-6} | 5.79x10^{-6} | - |

Table 4
Integration Times & Covariance Error
2-Second Data Rate

<table>
<thead>
<tr>
<th>TOLERANCE</th>
<th>Rev</th>
<th>10^{-6}</th>
<th>10^{-4}</th>
<th>10^{-2}</th>
<th>Fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1/4 U, ̄U</td>
<td>63370</td>
<td>12638</td>
<td>9534</td>
<td>9238</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15439</td>
<td>8857</td>
<td>8916</td>
<td>8403</td>
</tr>
<tr>
<td></td>
<td>1/2 U, ̄U</td>
<td>82088</td>
<td>22013</td>
<td>19025</td>
<td>18553</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30918</td>
<td>17701</td>
<td>17872</td>
<td>17083</td>
</tr>
<tr>
<td></td>
<td>3/4 U, ̄U</td>
<td>119973</td>
<td>31768</td>
<td>28456</td>
<td>27711</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46552</td>
<td>26648</td>
<td>26883</td>
<td>25633</td>
</tr>
<tr>
<td></td>
<td>1 U, ̄U</td>
<td>163553</td>
<td>41182</td>
<td>37966</td>
<td>36741</td>
</tr>
<tr>
<td></td>
<td></td>
<td>62087</td>
<td>35562</td>
<td>35876</td>
<td>34187</td>
</tr>
</tbody>
</table>

Covariance Error (1 Revolution)

| Pos U, ̄U | 0 | 5.25x10^{-4} | 3.72x10^{-3} | 1.87x10^{-3} |
| Vel U, ̄U | 2.32x10^{-11} | 5.79x10^{-8} | 3.74x10^{-3} | 1.87x10^{-3} |
| Pos d | 1.00x10^{-7} | 1.00x10^{-7} | 1.00x10^{-7} | 1.00x10^{-7} |
| Vel d | 9.27x10^{-3} | 9.27x10^{-3} | 9.27x10^{-3} | 9.27x10^{-3} |
Conclusions

Based on the numerical results obtained for the example problem considered in the investigation, it is concluded that:

- the (U, D) algorithm is the more efficient in terms of the integration time required to achieve a specified computation accuracy
- the (U, D) method can achieve the highest computation accuracy but with a heavy penalty in integration time
- the (U, D) algorithm has an advantage in terms of core storage, as it requires an integration vector of n(n+1)/2 elements, as opposed to the n x n elements required on the (Σ) method
- the disparity in efficiency between the two methods becomes less as the simulated measurement rate is increased.

The performance of the (U, D) propagation algorithm for this test problem was not what had been desired. Yet, results indicate there are at least two situations where the (U, D) method may offer improved efficiencies. These are in systems with measurement rates high enough to constrain integration stepsize, and during large prediction intervals without measurements, where analytic approximation of the process noise may not be adequate. These are the areas where future research will continue.

Acknowledgments

The authors wish to thank John Landberg and Paul Wing of the Department of Aerospace Engineering and Engineering Mechanics for their respective efforts in preparing the simulation software and in typing manuscripts of this paper.

APPENDIX

The measurement update algorithm for the CRF factorization [7] has the following form. Using the observation \( Y_{k+1} = G(X_{k+1|k}) \) calculate:

\[
H_{k+1} = \left[ G(U_{k+1|k+1})/\delta Y_{k+1} \right]
\]

\[
F_{j} = H_{j} + \sum_{k=j+1}^{n} \delta H_{k|k-1}
\]

\[
V_{j} = \delta F_{j}
\]

Set \( F_{j+1} = R_{j+1} \) (where \( R_{j+1} \) is the measurement noise) and calculate:

\[
F_{j} = F_{j} \cdot V_{j}^{n-1}
\]

\[
V_{j} = V_{j} \cdot F_{j}^{n-1}
\]

\[
X_{j} = X_{j} \cdot U_{j}^{n-1}
\]

References


