JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS
Vol. 30, No. 2, March–April 2007

Global Positioning System Integer Ambiguity Resolution Using Factorized Least-Squares Techniques

Mark L. Psiaki* and Shan Mohiuddin†
Cornell University, Ithaca, New York 14853-7501

DOI: 10.2514/1.21982

Factorized methods are developed for rapid solution of integer least-squares problems that arise when resolving Global Positioning System carrier cycle ambiguities. Such algorithms can enhance batch estimators and Kalman filters that use carrier-phase differential Global Positioning System data for relative spacecraft position estimation or for attitude determination. The solution of mixed real/integer linear least-squares problems is reviewed, and new algorithms are developed to speed the solution of the integer part of such problems. One new algorithm generates a candidate set of integer vectors that is bounded by an ellipsoid and that is guaranteed to contain the solution. Once generated, this set is searched by brute force to find the integer optimum. The set generator is based on the principle of backsubstitution for upper-triangular linear systems. Two new preconditioning algorithms are developed based on a principle of least-squares ambiguity decorrelation adjustment that seeks an increasing order in the magnitudes of the diagonal elements of the problem's upper-triangular square-root information matrix. These new algorithms decrease computation times in comparison to their nearest competitors by factors ranging from 2 to 4 for a random set of problems that have between 11 and 50 integer unknowns.

I. Introduction

THE Global Positioning System (GPS) uses radio navigation signals to provide absolute position accuracy on the order of 10 m anywhere near the Earth up to an altitude of about 3000 km. The basic measurement is the time delay between the transmission and the reception of a pseudorandom number code. This time delay translates into a range measurement that is accurate to several meters.

Very precise GPS measurements can be made by using the carrier phase. The nominal carrier wavelength of the GPS L1 frequency is 19.03 cm, and a typical GPS receiver can measure carrier phase to an accuracy of about 0.5 cm, that is, to an accuracy of about 1/38th of a wavelength. Carrier-phase differential GPS (CDGPS) measurements can be used to determine the position of one antenna relative to another to cm-level accuracy.

Carrier-phase techniques have been used successfully for attitude determination [1] and for surveying [2], and research is ongoing to develop CDGPS methods for doing relative navigation of formations of spacecraft [3–5]. Three-axis attitude determination accuracies on the order of 0.5 deg 1-σ can be achieved in low-Earth orbit using an array of four antennas spaced along 1-m baselines [1]. CDGPS-based relative navigation of spacecraft formations can provide cm-level relative position accuracy over 1–10 km baselines [3–5]. This application of CDGPS can provide an enabling technology for autonomous rendezvous and docking, for synthetic aperture radar, and for long-baseline optical interferometry. If used in real time to provide feedback signals for attitude control or formation keeping, then CDGPS estimation algorithms must execute rapidly.

CDGPS techniques are based on single- or double-differenced carrier-phase measurements that normally include an integer ambiguity in the number of carrier cycles. The basic CDGPS measurement and the resultant carrier-phase ambiguity are illustrated in Fig. 1. GPS satellite k transmits the carrier signal whose waves are illustrated by the dotted sinusoid in the figure. The carrier wavelength is λ. The carrier-phase difference between the signal received at antenna j and the signal received at antenna i gives a measure of the relative position vector between the two antennas, r_{ji}. In the absence of carrier cycle ambiguities, the phase difference between the two antennas would be 3.5 cycles, and it would indicate that the projection of r_{ji} onto the direction vector to the satellite, $\hat{s}^k$, was $r_{ji}^T \hat{s}^k = -3.5\lambda$. Carrier cycle ambiguities arise because receiver i and receiver j have no way to guarantee that they both measure carrier phase using the same phase datum [6]. The resulting carrier-phase difference normally contains an integer error. This error might indicate that $r_{ji}^T \hat{s}^k = -0.5\lambda$ or that $r_{ji}^T \hat{s}^k = +2.5\lambda$, as illustrated in Fig. 1 by the two possible alternate locations of antenna j relative to antenna i.

Many CDGPS techniques estimate the integer ambiguities and correct for them, for example, see [2,5]. There exist several approaches for ambiguity estimation while simultaneously determining the relative position of two independent receivers [7–10]. The resulting estimation problems are mixed real/integer linear problems. The real unknowns are the elements of the relative position vector r_{ji}. Typical solution procedures solve for the real unknowns as functions of the integers, eliminate the real unknowns, and solve the resulting integer linear least-squares problem using specialized techniques. The trick to solving the latter problem is to avoid brute-force search as much as possible, especially when the number of integer ambiguities is large.

The present paper has two principal goals. The first is to serve as a tutorial on the integer least-squares problems associated with GPS integer ambiguity estimation. The second is to develop improved solution algorithms for such problems, algorithms that execute more rapidly on average. Although some existing algorithms are relatively fast, even faster algorithms are useful for new applications that demand either very rapid real-time execution or a reasonable execution speed when a large number of ambiguities must be estimated. One such application is the GPS attitude determination procedure of [11]; it requires multiple calls of a linear ambiguity estimator in order to implement its global nonlinear attitude estimation algorithm.

This paper also touches on the issue of determining the reliability of a solution. Reliability is measured by the probability that the computed solution is correct. The method of computing solution reliability can impact the method used to solve for the ambiguity estimates, which makes reliability computation a concern of this paper.

Received 22 December 2005; revision received 2 October 2006; accepted for publication 2 October 2006.

*Professor, Sibley School of Mechanical and Aerospace Engineering. Associate Fellow AIAA.

†Graduate Student, Sibley School of Mechanical and Aerospace Engineering.

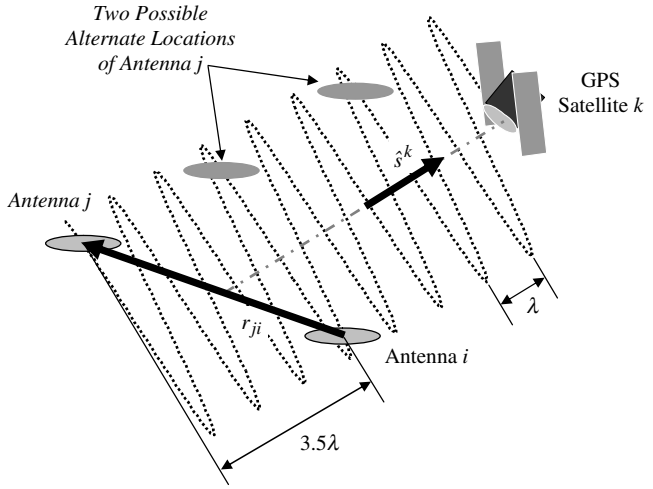

Fig. 1 An illustration of a differential carrier phase GPS position measurement and its integer cycle ambiguity.

The present paper makes four contributions. The first contribution is a set of tutorial discussions of various aspects of the solution of integer linear least-squares (ILLS) problems posed in a square-root information equation format. These discussions include simple examples to help the reader understand what a solution algorithm must do. Other tutorials on this subject can be found in [12–14], but they do not maintain the square-root information format throughout their presentations. The second contribution is an improvement of an existing algorithm for solving an ILLS problem. This algorithm executes rapidly if the problem has been properly preconditioned, and it computes its solution in a way that is compatible with a Bayesian analysis of the solution's probability of correctness. The third contribution is a pair of new algorithms for the computation of preconditioning matrix factorizations that speed up the solution of the original ILLS problem. These factorizations are commonly known as least-squares ambiguity decorrelation adjustment (LAMBDA) methods [7,9]. The fourth contribution is a set of computational results that measure the speed of various solution algorithms when used with various LAMBDA preconditioning factorizations. The goal of this part of the paper is to determine which algorithms are the fastest.

This paper defines the problem being solved, develops its new algorithms, and presents its computation speed results in four main sections. Section II defines a mixed real/integer linear least-squares problem and explains how to reduce it to an ILLS problem. This section also explains how to calculate the Bayesian probability that the solution is correct. Section III presents solution techniques for the ILLS problem, including both approximate and exact techniques, and it illustrates the exact technique using a low-dimensional example. Section IV explains the use of the LAMBDA method preconditioning factorization, develops two new LAMBDA factorization algorithms, justifies their efficacy, and illustrates their application using an example. Section V develops a set of Monte Carlo simulations of low-dimensional and high-dimensional ILLS problems, and it uses these simulation problems to evaluate the speed of this paper's new algorithms and of several existing algorithms. Section VI briefly summarizes results obtained when using the new algorithms on data from real GPS receivers and from high-fidelity simulators. Section VII presents the conclusions.

II. Definition of a Mixed Real/Integer Linear Least-Squares Problem and Reduction to an ILLS Problem

A. Mixed Real/Integer Problem Definition

Many CDGPS estimation problems can be put into the following square-root information form after appropriate linearizations and transformations [5,11]:

$$\boldsymbol{y} = A_x\boldsymbol{x} + A_n\boldsymbol{n} + \boldsymbol{v} \tag{1}$$

where $\boldsymbol{y}$ is a p-dimensional vector of transformed measurements and a priori information terms, $\boldsymbol{x}$ is a q-dimensional vector of real-valued unknowns such as relative position components or small-angle attitude parameter errors, $\boldsymbol{n}$ is an m-dimensional vector of integer ambiguities, and $\boldsymbol{v}$ is a p-dimensional vector of zero-mean, unit-variance, uncorrelated Gaussian noise terms. The matrices A_x and A_n model transformed measurements and, if present, a priori information [5]. Their respective dimensions are $p \times q$ and $p \times m$. The dimensions in Eq. (1) obey $p \geq q + m$ so that the system is well determined or overdetermined. Although Eq. (1) looks like a typical linear measurement equation, the zero-mean, unit-variance, uncorrelated nature of the errors in $\boldsymbol{v}$ causes this equation to conform to the square-root information format.

The estimation problem associated with Eq. (1) takes the form of a batch mixed real/integer linear least-squares problem. The optimal estimates are found by minimizing the cost function:

$$J_{\text{mixed}}(\boldsymbol{x}, \boldsymbol{n}) = \tfrac{1}{2}(\boldsymbol{y} - A_x\boldsymbol{x} - A_n\boldsymbol{n})^T(\boldsymbol{y} - A_x\boldsymbol{x} - A_n\boldsymbol{n}) \tag{2}$$

with respect to $\boldsymbol{x}$ and $\boldsymbol{n}$.

B. Solution for Real Part and Reduction to ILLS Problem

A solution procedure for this problem first minimizes with respect to $\boldsymbol{x}$ and then uses the resulting $\boldsymbol{x}$ solution to eliminate this variable from the problem. This can be accomplished if one uses a left orthogonal/upper-triangular (QR) factorization [15] of the matrix $[A_x \quad A_n]$:

$$Q_a \begin{bmatrix} R_{xx} & R_{xn} \\ 0 & R \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} A_x & A_n \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} z_x \\ z \\ z_r \end{bmatrix} = Q_a^T\boldsymbol{y} \tag{3}$$

where Q_a is a $p \times p$ orthonormal matrix, that is, $Q_a^TQ_a = Q_aQ_a^T = I$, R_{xx} and R are square, nonsingular, upper-triangular matrices of dimensions $q \times q$ and $m \times m$, respectively, R_{xn} is a $q \times m$ matrix, and the vectors z_x, z, and z_r have the respective dimensions q, m, and $p - (q + m)$. The transformation in Eq. (3) can be used to rewrite the cost function in Eq. (2) in the following equivalent form:

$$\begin{aligned} J_{\text{mixed}}(\boldsymbol{x}, \boldsymbol{n}) &= \tfrac{1}{2}(z_x - R_{xx}\boldsymbol{x} - R_{xn}\boldsymbol{n})^T(z_x - R_{xx}\boldsymbol{x} - R_{xn}\boldsymbol{n}) \\ &\quad + \tfrac{1}{2}(z - R\boldsymbol{n})^T(z - R\boldsymbol{n}) + \tfrac{1}{2}z_r^Tz_r \end{aligned} \tag{4}$$

The last term in this cost formula is a residual error term that is not affected by $\boldsymbol{x}$ or $\boldsymbol{n}$; therefore, it has no effect on the optimal values of $\boldsymbol{x}$ and $\boldsymbol{n}$. The optimal $\boldsymbol{x}$ is derived by solving the linear first-order optimality necessary condition: $0 = \partial J_{\text{mixed}}/\partial\boldsymbol{x}$. The result is $\boldsymbol{x}_{\text{opt}} = R_{xx}^{-1}(z_x - R_{xn}\boldsymbol{n})$.

Substitution of $\boldsymbol{x}_{\text{opt}}$ into Eq. (4) and omission of the residual error term yields the following ILLS cost function:

$$J(\boldsymbol{n}) = \tfrac{1}{2}(z - R\boldsymbol{n})^T(z - R\boldsymbol{n}) \tag{5}$$

If $\boldsymbol{n}$ were allowed to be real valued, then the vector that minimized the cost in Eq. (5) would be $(\boldsymbol{n}_{\text{float}})_{\text{opt}} = R^{-1}z$. The fact that $\boldsymbol{n}$ must take on integer values makes the minimization of the Eq. (5) cost much more challenging, and the remainder of this paper focuses on this problem.

C. Bayesian Analysis of Solution Probability

The subject of integer ambiguity validation is related to integer ambiguity estimation. Given an estimate, one can validate it by calculating the probability that it is correct. If the probability is sufficiently near one, then the ambiguity estimate is deemed valid.

The probability that any given $\boldsymbol{n}$ is the correct integer ambiguity vector can be determined by using Bayesian analysis. Suppose that Eq. (1) contains measurement information and a priori information about $\boldsymbol{x}$ and $\boldsymbol{n}$ [a priori information can be modeled in Eq. (1) by adding rows to $\boldsymbol{y}$, A_x, and A_n that contain a priori square-root information equations for $\boldsymbol{x}$ and $\boldsymbol{n}$] or suppose that Eq. (1) contains only measurement information and that the diffuse prior assumption

is used for $\boldsymbol{x}$ and $\boldsymbol{n}$. Then the probability of the integer vector $\boldsymbol{n}_{\text{opt}}$ being the true $\boldsymbol{n}$ conditioned on the data and prior information in $\boldsymbol{y}$ is given by the formula [16]

$$P(\boldsymbol{n}_{\text{opt}} \mid \boldsymbol{y}) = \frac{\exp\{-J(\boldsymbol{n}_{\text{opt}})\}}{\sum_{\text{all integer-valued } n} \exp\{-J(\boldsymbol{n})\}} \tag{6}$$

where the normalizing summation in the denominator is an infinite sum over each element of the vector $\boldsymbol{n}$. This probability calculation is equivalent to the Wald sequential probability calculation of [17] under suitable assumptions.

It is apparent from Eq. (6) that the minimizing solution to the ILLS problem in Eq. (5) is the maximum a posteriori (MAP) estimate of $\boldsymbol{n}$ [16]. That is, $P(\boldsymbol{n}_{\text{opt}} \mid \boldsymbol{y}) > P(\boldsymbol{n} \mid \boldsymbol{y})$ for all $\boldsymbol{n} \neq \boldsymbol{n}_{\text{opt}}$ because $J(\boldsymbol{n}_{\text{opt}}) < J(\boldsymbol{n})$ for all $\boldsymbol{n} \neq \boldsymbol{n}_{\text{opt}}$. If $J(\boldsymbol{n}_{\text{opt}})$ is sufficiently smaller than all of the other $J(\boldsymbol{n})$ values, then $P(\boldsymbol{n}_{\text{opt}} \mid \boldsymbol{y})$ in Eq. (6) will be very near 1, and $\boldsymbol{n}_{\text{opt}}$ will be considered to be valid.

It is possible to approximate the multidimensional infinite sum in the denominator of Eq. (6). The value of $\exp\{-J(\boldsymbol{n})\}$ is nonnegligible only for a finite set of integer-valued $\boldsymbol{n}$ vectors. Therefore, a summation over a reasonable finite set should yield a good approximation of the full summation. This fact can impact the choice of the solution algorithm for the ILLS problem in Eq. (5).

This short section is intended only as a motivation for certain aspects of the ILLS solution algorithm. It is not intended to fully explore the subject of integer ambiguity validation, which can also involve non-Bayesian analysis before taking data to compute the likelihood of being able to validate a single integer solution. See [18,19] for more details about ambiguity validation.

III. An ILLS Solution Algorithm

A. Two Approximate Solution Algorithms

Before developing an exact solution algorithm, it is helpful to examine two approximate solution algorithms. The most obvious approximate method is to solve for the real-valued solution, as discussed after Eq. (5), and round the result: $\boldsymbol{n}_{\text{opt}} \cong \text{round}[(\boldsymbol{n}_{\text{float}})_{\text{opt}}]$, where the round[ ] function takes a real-valued vector input and rounds each of its elements to the nearest integer to produce a vector output. Unfortunately, this solution can be quite far from the true optimum, as illustrated by the 2-dimensional problem of Fig. 2. The figure depicts a lattice of integer-valued points in the n_1–n_2 plane along with the cost contour $J(\boldsymbol{n}) = 0.5$; n_1 and n_2 are the two

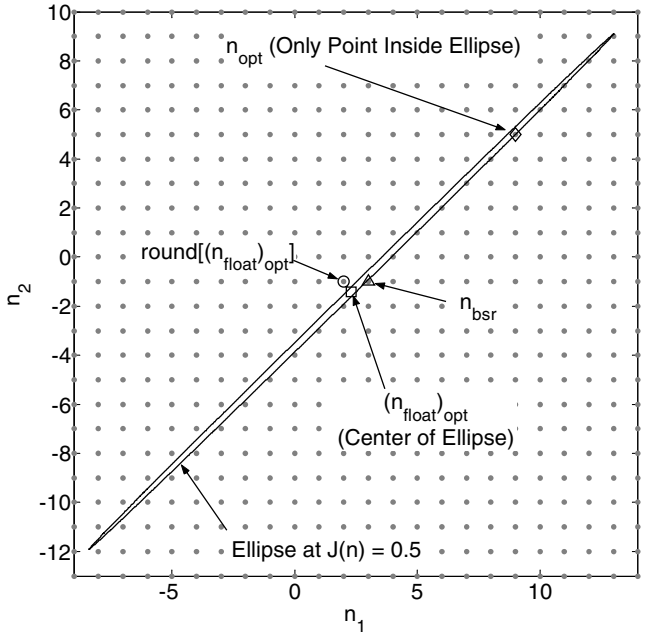

Fig. 2 Cost contour and float, rounded, alternating backsubstitution rounded, and optimal solutions for an ILLS problem.

elements of the vector $\boldsymbol{n}$. The real-valued solution is $(\boldsymbol{n}_{\text{float}})_{\text{opt}} = [2.310; -1.409]$, which rounds to the lattice point $[2; -1]$, denoted by a circle in Fig. 2. This point is far removed from the actual optimum $\boldsymbol{n}_{\text{opt}} = [9; 5]$, denoted by a diamond in Fig. 2. The cost at the rounded approximate solution, $J([2; -1]) = 5.757$, is more than 11 times larger than the optimal value $J(\boldsymbol{n}_{\text{opt}}) = 0.499$.

A better approximate solution can be derived by using rounding in a more judicious manner during the backsubstitution solution of the upper-triangular system of equations that defines $(\boldsymbol{n}_{\text{float}})_{\text{opt}}$. Recall that there are m elements of $\boldsymbol{n}$. The following procedure solves successive rows of the upper-triangular system of equations associated with the cost in Eq. (5) and rounds the result for each row before proceeding to the next higher row:

$$n_{\text{bsr}(m)} = \text{round}(z_m / R_{mm}) \tag{7a}$$

$$n_{\text{bsr}(i)} = \text{round}\left\{\left[z_i - \sum_{j=i+1}^{m} R_{ij} n_{\text{bsr}(j)}\right] \Big/ R_{ii}\right\} \quad \text{for } i = (m-1), \ldots, 1 \tag{7b}$$

where $n_{\text{bsr}(i)}$ is the ith element of the approximate solution vector $\boldsymbol{n}_{\text{bsr}}$, z_i is the ith element of the vector $\boldsymbol{z}$, and R_{ij} is the ijth element of the upper-triangular matrix R.

The improved solution $\boldsymbol{n}_{\text{bsr}} = [3; -1]$ has been calculated for the problem associated with Fig. 2, and it is denoted by a triangle in the figure. It is slightly closer to the true solution than is the brute-force rounded solution. More important, its cost, $J([3; -1]) = 0.820$, is about 7 times smaller than the cost associated with the brute-force rounded solution.

This latter approximate solution technique is known in the literature as the method of ambiguity bootstrapping [20]. This method has great practical importance because of its simplicity and because of its near optimal performance after problem transformations such as those that are discussed in Sec. IV.

B. Calculation of a Set that Contains the Global Optimum

The exact solution algorithm is based on a set of integer points in $\boldsymbol{n}$ space that must contain the global minimum. The algorithm tailors its set of points and its calculations to minimize the number of elements in the set subject to certain constraints and to allow rapid calculation of the elements. The set that gets computed is

$$S_e = \{\boldsymbol{n}\colon J_e \geq \tfrac{1}{2}(\boldsymbol{z} - R\boldsymbol{n})^T(\boldsymbol{z} - R\boldsymbol{n}), \boldsymbol{n} \in Z^m\} \tag{8}$$

This is the set of all integer-valued lattice points within and on the ellipsoidal cost contour that has the cost value $J(\boldsymbol{n}) = J_e$. Given this set, the optimal solution to the ILLS problem is $\boldsymbol{n}_{\text{opt}} \in S_e$ such that $J(\boldsymbol{n}_{\text{opt}}) \leq J(\boldsymbol{n})$ for all $\boldsymbol{n} \in S_e$; $\boldsymbol{n}_{\text{opt}}$ can be computed by a brute-force search within S_e. The optimum must be contained in S_e if S_e is not empty because all integer-valued $\boldsymbol{n}$ vectors in S_e have lower cost values than all integer vectors that are not in S_e.

The cost value J_e must be chosen large enough to ensure that S_e is not empty but small enough to keep S_e from containing prohibitively many points. A good cost value to use for defining S_e is $J_e = J(\boldsymbol{n}_{\text{bsr}})$, with $\boldsymbol{n}_{\text{bsr}}$ computed using Eqs. (7a) and (7b). This value guarantees that $\boldsymbol{n}_{\text{bsr}} \in S_e$ so that S_e is not empty, and it often keeps the number of points from being very large because $J(\boldsymbol{n}_{\text{bsr}})$ is usually fairly near to $J(\boldsymbol{n}_{\text{opt}})$.

As an example of this set, consider Fig. 2 once more. The set S_e corresponding to $J_e = J(\boldsymbol{n}_{\text{bsr}})$ is the set of grid points within a slightly enlarged ellipse (not shown) that passes through $\boldsymbol{n}_{\text{bsr}} = [3; -1]$. In this case, $S_e = \{[3; -1], [4; 0], [5; 1], \ldots, [15; 11]\}$ contains 13 grid points, all of which fall on the diagonal line that connects $\boldsymbol{n}_{\text{bsr}}$ and $\boldsymbol{n}_{\text{opt}}$.

An alternative brute-force method is to consider all grid points in the rectangle that bounds the $J(\boldsymbol{n}) = J_e$ ellipsoid. In the Fig. 2 example, this consists of the set $\{[i; j]$ for all $-11 \leq i \leq 16, -14 \leq j \leq 12\}$. This rectangular set is easier to calculate than S_e, but it contains many more points, 756. The ratio of

the number of grid points in a bounding hyperrectangle to the number of grid points in a hyperellipsoid can become even larger in higher dimensions if the ellipsoid is narrow and skewed, as in Fig. 2. Therefore, it is preferable to work with ellipsoidal sets.

This ILLS solution strategy has been taken directly from [9]. The present work makes two new contributions to the implementation of this strategy. First, it poses and solves the ILLS problem in a square-root information form, which can be easier to understand than the statistical form presented in [7,9]; [9] mentions the square-root information form, but it uses the statistical form to complete its algorithm development. The second contribution is the development of a breadth-first solution algorithm, as opposed to [9]'s depth-first algorithm. Breadth first refers to the process of determining all possible S_e candidates for the truncated vector $[n_{(m-j)}; \ldots; n_m]$ after determining all possible candidates for the truncated vector $[n_{(m-j+1)}; \ldots; n_m]$. This process iterates backward starting at $m - j = m - 1$ and finally determines the elements of S_e when it reaches $m - j = 1$. A depth-first algorithm computes complete candidate vectors $[n_1; \ldots; n_m]$ one at a time.

The elements of an ellipsoidally bounded set S_e can constitute the full set of nonnegligible points needed in the denominator summation of Eq. (6) for Bayesian validation of the estimate. Suppose that one defines S_e using a J_e value such that $J_e \geq J_{\text{opt}} + \Delta J_{\text{neg}}$, where ΔJ_{neg} is chosen large enough to make $\exp\{-\Delta J_{\text{neg}}\}$ negligible compared to 1. This J_e allows one to neglect all grid points outside of S_e in the summation of Eq. (6). In practice, one chooses $J_e = J(\boldsymbol{n}_{\text{bsr}}) + \Delta J_{\text{neg}}$ because J_{opt} is not known at the outset of the S_e calculation. This general strategy for approximately computing the infinite sum in Eq. (6) was originally proposed in [21].

C. Determination of the Points in S_e by a Generalized Backsubstitution with Rounding

A process that is like backsubstitution can be used to determine the elements of the set S_e. Before presenting the details of this process, it is helpful to explain it geometrically using an illustrative example.

Consider Fig. 3, which shows the $J(\boldsymbol{n}) = J_e$ bounding ellipse for an example problem of dimension $m = 2$. The integer grid points inside this ellipse constitute S_e. The backsubstitution process that calculates these points starts by considering the range of possible n_2 values for points in S_e. This range is determined algebraically by the condition that the cost not exceed J_e for the error in the last equation of the upper-triangular system $R\boldsymbol{n} = \boldsymbol{z}$. As readily observable from the figure, this range is $-7 \leq n_2 \leq -2$.

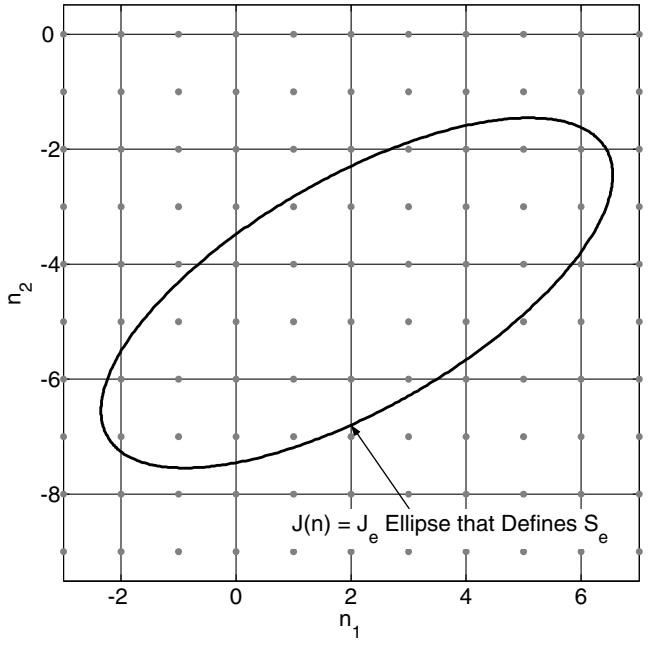

Fig. 3 Example cost contour that defines a set S_e and the n grid points.

The next step of the process is to determine the allowable range of n_1 values for each allowable n_2 value. This range varies as a function of n_2, as can be seen in the figure. It is calculated by requiring that the total cost not exceed J_e for the errors in both equations of the linear system $R\boldsymbol{n} = \boldsymbol{z}$. The allowable n_1 ranges, as read off of Fig. 3, are $-2 \leq n_1 \leq 1$ when $n_2 = -7$, $-2 \leq n_1 \leq 3$ when $n_2 = -6$, $-1 \leq n_1 \leq 4$ when $n_2 = -5$, $0 \leq n_1 \leq 5$ when $n_2 = -4$, $1 \leq n_1 \leq 6$ when $n_2 = -3$, and $3 \leq n_1 \leq 6$ when $n_2 = -2$. These six combinations of n_1 ranges and n_2 values fully specify the set S_e for this 2-dimensional case.

This same process works in higher dimensions. A valid range for $n_{(m-j)}$ is determined for each combination of the elements $n_{(m-j+1)}, \ldots, n_m$ that is a valid candidate for membership in S_e. This range is determined by enforcing a limit on the cost of the error in equation $(m - j)$ of the upper-triangular system $R\boldsymbol{n} = \boldsymbol{z}$. This cost limit equals J_e minus the error costs associated with equations $(m - j + 1)$ through m, which depend on the given values of $n_{(m-j+1)}, \ldots, n_m$.

Valid partial points can become invalid as this calculation proceeds backward. Suppose that a particular combination of the elements $n_{(m-j+1)}, \ldots, n_m$ produces no valid $n_{(m-j)}$ integers. Then this $n_{(m-j+1)}, \ldots, n_m$ combination is invalid. This combination produces an error cost associated with equations $(m - j + 1)$ through m that does not exceed J_e, but there is no integer value of $n_{(m-j)}$ such that the error cost associated with equations $(m - j)$ through m does not exceed J_e. This situation is illustrated in Fig. 2. Suppose that $J_e = 0.5$, so that the ellipse shown in the figure is used to define S_e. Then the value $n_2 = -11$ is deemed a valid possibility when considering only the error cost in the second equation, but no point with $n_2 = -11$ lies inside the ellipse because no such point yields a total error cost for the two equations that respects the upper limit $J_e = 0.5$.

The detailed steps of the S_e calculation algorithm are as follows (with explanatory material given after each step):

1) Calculate the minimum and maximum values for n_m,

$$(n_m)_{\min} = \text{ceil}\left(\frac{z_m}{R_{mm}} - \frac{\sqrt{2J_e}}{|R_{mm}|}\right) \quad \text{and}$$
$$(n_m)_{\max} = \text{floor}\left(\frac{z_m}{R_{mm}} + \frac{\sqrt{2J_e}}{|R_{mm}|}\right) \tag{9}$$

where the ceil( ) function rounds its input to the nearest integer toward $+\infty$ and the floor( ) function rounds to the nearest integer toward $-\infty$. Define $L_m = (n_m)_{\max} - (n_m)_{\min} + 1$, define $\boldsymbol{n}_l^m = (n_m)_{\min} + l - 1$ and $\alpha_l^m = 2J_e - (z_m - R_{mm}\boldsymbol{n}_l^m)^2$ for $l = 1, \ldots, L_m$, and if $m > 1$, then define

$$\Delta \boldsymbol{z}_l^{m-1} = \begin{bmatrix} z_1 - R_{1m}\boldsymbol{n}_l^m \\ \vdots \\ z_{(m-1)} - R_{(m-1)m}\boldsymbol{n}_l^m \end{bmatrix} \quad \text{for } l = 1, \ldots, L_m \tag{10}$$

Set $k = m$.

Explanation: The quantity L_k keeps track of the number of feasible partial vector elements of S_e. Each feasible partial vector extends from the kth element of $\boldsymbol{n}$ to the mth element. Each partial vector takes the $(m - k + 1)$-dimensional form $\boldsymbol{n}_l^k = [n_{kl}^k; \ldots; n_{ml}^k]$. The scalar n_{il}^k is the corresponding feasible value for the ith component of $\boldsymbol{n}$. When $k = m$ each partial vector is only one dimensional. The value α_l^k equals twice the error cost that remains to be used for the errors in the first through $(k - 1)$st equations of the system $R\boldsymbol{n} = \boldsymbol{z}$. $\alpha_l^k \geq 0$ is required for feasibility. The $(k - 1)$-dimensional vector $\Delta \boldsymbol{z}_l^{k-1} = [\Delta z_{1l}^{k-1}; \ldots; \Delta z_{(k-1)l}^{k-1}]$ contains the errors in the first through $(k - 1)$st rows of $\boldsymbol{z} - R[0; \boldsymbol{n}_l^k]$.

2) If $k = 1$, then stop and set $S_e = \{n_1^1, \ldots, n_{L_1}^1\}$; otherwise, replace k by $k - 1$ and go to step 3.

Explanation: This is the start of the algorithm's main outer loop. It is also the loop's termination point. The main loop executes once for each value of k that is less than m. Each loop iteration uses L_{k+1} and $\boldsymbol{n}_l^{k+1}, \alpha_l^{k+1}$, and $\Delta \boldsymbol{z}_l^k$ for $l = 1, \ldots, L_{k+1}$ along with R and $\boldsymbol{z}$ as inputs,

and it computes the outputs L_k and $\boldsymbol{n}_j^k$, α_j^k, and Δz_j^{k-1} for $j = 1, \ldots, L_k$.

3) Initialize $L_k = 0$ and set $l = 1$.

Explanation: This step initializes the inner loop that works through all $\boldsymbol{n}_l^{k+1}$, α_l^{k+1}, and Δz_l^k for $l = 1, \ldots, L_{k+1}$ and calculates the corresponding $\boldsymbol{n}_j^k$, α_j^k, and Δz_j^{k-1} values by computing all feasible values of n_k.

4) Compute minimum and maximum values for n_k given $\boldsymbol{n}_l^{k+1}$, α_l^{k+1}, and Δz_l^k:

$$(n_k)_{\min} = \text{ceil}\left(\frac{\Delta z_{kl}^k}{R_{kk}} - \frac{\sqrt{\alpha_l^{k+1}}}{|R_{kk}|}\right) \quad \text{and}$$
$$(n_k)_{\max} = \text{floor}\left(\frac{\Delta z_{kl}^k}{R_{kk}} + \frac{\sqrt{\alpha_l^{k+1}}}{|R_{kk}|}\right) \tag{11}$$

Explanation: These equations are like Eq. (9), except that they account for the effects of the feasible selections for $n_{k+1}, \ldots, n_m$ that are contained in $\boldsymbol{n}_l^{k+1}$.

5) Compute $\Delta L_k = (n_k)_{\max} - (n_k)_{\min} + 1$. If ΔL_k is not positive, then go to step 6; otherwise, perform the following calculations for $j = (L_k + 1), \ldots, (L_k + \Delta L_k)$:

$$n_{kj}^k = (n_k)_{\min} + j - L_k - 1 \tag{12a}$$

$$\boldsymbol{n}_j^k = \begin{bmatrix} n_{kj}^k \\ \boldsymbol{n}_l^{k+1} \end{bmatrix} \tag{12b}$$

$$\alpha_j^k = \alpha_l^{k+1} - \{\Delta z_{kl}^k - R_{kk} n_{kj}^k\}^2 \tag{12c}$$

$$\Delta z_j^{k-1} = \begin{bmatrix} \Delta z_{1l}^k - R_{1k} n_{kj}^k \\ \vdots \\ \Delta z_{(k-1)l}^k - R_{(k-1)k} n_{kj}^k \end{bmatrix} \quad \text{if } k > 1 \tag{12d}$$

and replace L_k by $L_k + \Delta L_k$.

Explanation: If $\Delta L_k \leq 0$, then $\boldsymbol{n}_l^{k+1}$ is an infeasible partial vector because it does not correspond to any feasible n_k values. Otherwise, these computations use each feasible n_k value and the $\boldsymbol{n}_l^{k+1}$ partial vector to create a unique $\boldsymbol{n}_j^k$ partial vector. They subtract the error cost of the kth equation of the system $R\boldsymbol{n} = z$ from α_l^{k+1} to compute α_j^k. $\alpha_j^k \geq 0$ is guaranteed by Eq. (11). Equation (12d) computes Δz_j^{k-1} from the first $k-1$ components of Δz_l^k by including the effect of the new n_{kj}^k component.

6) If $l = L_{k+1}$, then go to step 2. Otherwise, replace l by $l + 1$ and go to step 4.

Explanation: This is the final step of both the inner loop and the outer loop. A new iteration of the inner loop is selected if there remain additional $\boldsymbol{n}_l^{k+1}$ partial vectors to consider for the given value of k; otherwise, the algorithm transitions to the next iteration of the outer loop, in which k gets decremented by 1.

A useful metric for the execution time of this algorithm is the sum $L_{\text{tot}} = L_1 + \cdots + L_m$. It measures the total number of possible integer values considered for each element of $\boldsymbol{n}$. It counts a particular candidate value for n_i multiple times if that candidate has been considered in conjunction with multiple combinations of $[n_{i+1}; \ldots, n_m]$. This is sensible because each such consideration requires an independent set of calculations. The L_{tot} metric is roughly proportional to the amount of computation required by the algorithm. It will be used in Sec. V as a measure of algorithm performance.

A slight variation of this algorithm tries to reduce J_e once per iteration of the outer loop. This modification represents an attempt to reduce the number of elements in S_e, which should also reduce the computation cost metric L_{tot} if successful. The heuristic strategy for trying to reduce J_e is to compare α_j^k for $j = 1, \ldots, L_k$ to the α_j^k value that results when $[n_k; \ldots; n_m]$ are the last $m - k + 1$ elements of the integer vector $\boldsymbol{n}_e$, where $\boldsymbol{n}_e$ is the vector that yields $J_e = J(\boldsymbol{n}_e)$. Recall that $\boldsymbol{n}_e$ is initialized to be $\boldsymbol{n}_{\text{bsr}}$ from Eqs. (7a) and (7b) and that α_j^k is the unused portion of $2J_e$ that is available for errors in the first $k-1$ equations of the system $R\boldsymbol{n} = z$. Suppose that α_i^k is the minimum value and that it is lower than the value corresponding to $\boldsymbol{n}_e$. In this case, a new $\boldsymbol{n}$ vector is produced by using $\boldsymbol{n}_i^k$ for the last $m - k + 1$ elements and by using an alternating backsubstitution/rounding operation as in Eq. (7b) to compute the first $k-1$ elements. Suppose that the resulting vector of integers is $\boldsymbol{n}_{e\text{new}}$. Then J_e gets replaced by $J(\boldsymbol{n}_{e\text{new}})$ and $\boldsymbol{n}_e$ gets replaced by $\boldsymbol{n}_{e\text{new}}$ if $J(\boldsymbol{n}_{e\text{new}}) < J_e$.

D. Comparison with Other Methods

This method bears similarities to the methods of [7,10]. These two methods are developed using different notation. The method of [7] works with variances and conditional expectations rather than square-root information matrices and information equations. The method of [10] works with square-root information matrices, but it uses orthonormal/lower-triangular factorization of the square-root information matrix and forward substitution rather than backsubstitution. The two methods' calculations both consider ellipses like that used to define S_e in Eq. (8), and they both use backsubstitution-like calculations to search in a range of n_k values that is restricted by the errors already produced by the given values of $n_{k+1}, \ldots, n_m$ and by the cost associated with the ellipse boundary J_e.

There is one main difference between the methods of [7,10] and the present method, the latter being like the method of [9]. The present method calculates all points in or on the ellipsoid. The methods of [7,10] seek just one point interior to the ellipse, $\boldsymbol{n}_{e\text{new}}$ such that $J(\boldsymbol{n}_{e\text{new}}) < J_e$. If no such point exists, then $\boldsymbol{n}_{\text{opt}} = \boldsymbol{n}_e$, the point that produces cost $J_e = J(\boldsymbol{n}_e)$. If an integer point exists inside the ellipse, then J_e is replaced by the new lower value $J(\boldsymbol{n}_{e\text{new}})$, $\boldsymbol{n}_e$ is replaced by $\boldsymbol{n}_{e\text{new}}$, and the process is repeated until it terminates with no new point $\boldsymbol{n}_{e\text{new}}$ inside the current ellipse. This difference means that the methods of [7,10] do not compute all of the points in a fixed ellipsoidally bounded set. Therefore, they are not compatible with an S_e-based approximation of the Bayesian probability calculation in Eq. (6) as outlined at the end of Sec. III.B. (Although the present algorithm has the advantage of being compatible with the S_e-based calculation of Bayesian probability approximations, this paper does not perform any Bayesian calculations during its evaluation of the algorithm.)

The new algorithm stores $L_k(m+1)$ real numbers in $\boldsymbol{n}_j^k$, α_j^k, and Δz_j^{k-1} for $j = 1, \ldots, L_k$, but the solution algorithms of [7,10] do not store any such large arrays. This storage requirement can be significant for large problems. It may be on the order of 10^6–10^7 real scalars if $m \cong 50$ because L_k can be on the order of 10^5–10^6.

The storage requirement also depends on the choice of J_e and on the peakedness of the underlying Gaussian distribution of the equivalent real-valued ambiguity estimates. The required storage increases if J_e is increased or if the distribution is less peaked, that is, if the peak is less sharp. It is beyond the scope of this paper to determine the exact relationship between the choice of J_e, the distribution's peakedness, and the algorithm's storage requirements.

IV. Least-Squares Ambiguity Decorrelation Adjustments

A. Decorrelation Adjustment Definition

A least-squares ambiguity decorrelation adjustment can be defined in terms of a matrix factorization of the upper-triangular, nonsingular square-root information matrix R that is used to define the ILLS cost function $J(\boldsymbol{n})$ of Eq. (5). The general form of this factorization is

$$Q\bar{R}Z = R \tag{13}$$

where Q is an orthonormal matrix, that is, $Q^TQ = QQ^T = I$, $\bar{R}$ is a transformed upper-triangular, nonsingular square-root information matrix, and Z is a unimodular matrix, that is, all elements are integers

and $|\det(Z)| = 1$. All of the matrices in Eq. (13) are $m \times m$ matrices. The properties of Z guarantee that all of the elements of Z^{-1} are also integers.

The matrices Q and Z can be used to transform the vectors $\boldsymbol{n}$ and z to define an equivalent transformed ILLS problem:

$$\bar{\boldsymbol{n}} = Z\boldsymbol{n} \quad \text{and} \quad \bar{z} = Q^T z \tag{14a}$$

$$\bar{J}(\bar{\boldsymbol{n}}) = \tfrac{1}{2}(\bar{z} - \bar{R}\,\bar{\boldsymbol{n}})^T(\bar{z} - \bar{R}\,\bar{\boldsymbol{n}}) \tag{14b}$$

The properties of Z and its inverse imply that all integer vectors $\boldsymbol{n}$ map into integer vectors $\bar{\boldsymbol{n}}$ and vice versa. The cost in Eq. (14b) is equivalent to the original cost in Eq. (5) evaluated at $\boldsymbol{n} = Z^{-1}\bar{\boldsymbol{n}}$. Therefore, the minimum of the ILLS problem associated with the cost function in Eq. (14b), $\bar{\boldsymbol{n}}_{\text{opt}}$, can be used to determine the minimum of the original problem $\boldsymbol{n}_{\text{opt}} = Z^{-1}\bar{\boldsymbol{n}}_{\text{opt}}$.

The original definition of the decorrelation adjustment in [7] only includes the Z unimodular transformation. The Q orthonormal transformation is added in Eq. (13) to make the transformed square-root information matrix $\bar{R}$ be upper triangular. This addition does not impact the efficacy of the adjustment nor does it complicate the adjustment calculations.

B. Goal of Decorrelation Adjustment

The original goal of the decorrelation adjustment was to make the matrix $(\bar{R}^T\bar{R})^{-1}$ be as nearly diagonal as possible [7,22]. This matrix is the covariance matrix of the real-valued $\bar{\boldsymbol{n}}$ vector. If it were diagonal, then $\bar{R}$ would also be diagonal, and the simple rounding method presented at the beginning of Sec. III.A would yield $\bar{\boldsymbol{n}}_{\text{opt}}$. The restrictions on Z normally make it impossible to transform to an exactly diagonal $\bar{R}$, but several algorithms have been developed with the goal of making $\bar{R}$ nearly diagonal [7,22,23].

This paper develops two new decorrelation adjustment methods based on a different goal for the properties of the $\bar{R}$ matrix. This different goal is better tailored to reduce the computational cost of the ILLS solution algorithm of Sec. III. A different goal is desirable because the closeness of $\bar{R}$ to being diagonal has no direct bearing on the number of calculations needed to carry out the computations of Sec. III. Any upper-triangular Z matrix, each of whose diagonal elements must have a magnitude of 1, produces an $\bar{R}$ matrix that yields no change in the number of ILLS solution algorithm calculations when compared to the original R. Yet, this $\bar{R}$ can be very far from diagonal. The equivalence of the solution algorithm speed for R and $\bar{R}$ in this case is a direct result of the backsubstitution calculations used in the algorithm of Sec. III.

The alternative goal for the decorrelation adjustment is to make the magnitudes of the diagonal elements of $\bar{R}$ increase as much as possible as one proceeds down the diagonal. If an increase is not always possible, then the goal is modified to make the magnitudes decrease as little as possible. Stated mathematically, the goal is to make each ratio $|\bar{R}_{ii}|/|\bar{R}_{(i-1)(i-1)}|$ for $i = 2, \ldots, m$ as large as possible.

An $\bar{R}$ that is derived based on this alternative goal has two beneficial effects, on average, when used with the solution algorithm of Sec. III. These benefits derive from the fact that the magnitude of the optimal error in the ith equation of the system $\bar{R}\,\bar{\boldsymbol{n}} = \bar{z}$ tends to be less than $|\bar{R}_{ii}|$. If $\bar{R}$ meets the alternative goal, then the largest errors will tend to occur in the latter equations.

The first benefit caused by the alternative decorrelation strategy is that the approximate solution $\bar{\boldsymbol{n}}_{\text{bsr}}$, which is derived using Eqs. (7a) and (7b), tends to have a lower cost than the $\bar{\boldsymbol{n}}_{\text{bsr}}$ for the original R matrix. This yields a lower initial J_e value thereby reducing the size of the set S_e. A reduction in the size of S_e tends to reduce the number of solution computations.

The second benefit is that L_k, the number of partial solutions that may fall within S_e, tends to be reduced when the alternative decorrelation adjustment goal is achieved. The reduction comes about for two reasons. First, multiple points in S_e tend to have common latter elements of $\bar{\boldsymbol{n}}$. Second, the algorithm produces fewer seemingly feasible sets of the latter elements of $\bar{\boldsymbol{n}}$ that do not actually correspond to points in S_e.

Both of these benefits are illustrated by a 3-dimensional example. Consider the following ILLS problem and its decorrelated equivalent:

$$R = \begin{bmatrix} 2.2652 & 0.9950 & 0.1999 \\ 0 & -0.4056 & -0.1830 \\ 0 & 0 & 0.3270 \end{bmatrix}, \qquad z = \begin{bmatrix} -2.6901 \\ -3.2403 \\ 6.4402 \end{bmatrix} \tag{15a}$$

$$\bar{R} = \begin{bmatrix} -0.4247 & -0.2047 & -0.2064 \\ 0 & -0.8278 & 0.1043 \\ 0 & 0 & -0.8544 \end{bmatrix}, \qquad \bar{z} = \begin{bmatrix} 5.0888 \\ -2.7166 \\ 5.0928 \end{bmatrix},$$

$$Z = \begin{bmatrix} -3 & -2 & -1 \\ -1 & 0 & 0 \\ 2 & 1 & 0 \end{bmatrix} \tag{15b}$$

Note how the diagonal elements of R decrease in magnitude as one moves down the main diagonal, but the diagonal elements of $\bar{R}$ increase in magnitude. Thus, $\bar{R}$ achieves the alternative decorrelation goal better than R. The solution algorithm of Sec. III.C has a computational cost of $L_{\text{tot}} = 59$ when applied to the original problem data in Eq. (15a), but the cost is only $L_{\text{tot}} = 8$ for the decorrelated problem in Eq. (15b), which means that the decorrelated problem can be solved about 7 times faster than the original problem.

Part of this speedup is the result of a reduced J_e. The calculations of Eqs. (7a) and (7b) applied to the R and z in Eq. (15a) yield $\boldsymbol{n}_{\text{bsr}} = [-3; -1; 20]$ with $J(\boldsymbol{n}_{\text{bsr}}) = 0.6130$. Similar calculations using $\bar{R}$ and $\bar{z}$ from Eq. (15b) yield $\bar{\boldsymbol{n}}_{\text{bsr}} = [-11; 3; -6]$ with $\bar{J}(\bar{\boldsymbol{n}}_{\text{bsr}}) = 0.0992$. This reduction of the initial J_e by a factor of more than 6 lowers the number of points in S_e.

The decorrelation would save computations even if the same J_e value were used for both versions of the example problem. This is illustrated by Fig. 4. The figure shows the $n_2 - n_3$ plane for the original problem (left-hand plot) and the $\bar{n}_2 - \bar{n}_3$ plane for the decorrelated problem (right-hand plot). The ellipse in each plot is the intersection of the S_e boundary ellipsoid with the plane when S_e is defined using the cost value $J_e = 0.2$. The smaller size of the decorrelated ellipse in the right-hand plot is a direct result of the fact that the magnitudes of $\bar{R}_{22}$ and $\bar{R}_{33}$ are larger than the magnitudes of R_{22} and R_{33}. Every point inside the ellipsoid corresponds to a feasible partial point $\boldsymbol{n}_j^2$ that exists after the $k = 2$ iteration of the outer loop of the algorithm of Sec. III.C. Thus, there are $L_2 = 9$ points for the original problem, but there are only $L_2 = 2$ points for the decorrelated problem. The set S_e only contains five points. The algorithm calculates four unneeded $\boldsymbol{n}_j^2$ points for the original problem, but it calculates only two points, both of which are needed, for the decorrelated problem. It can be seen from these graphs that $L_3 = 4$ for the original problem and 1 for the decorrelated problem. Although not obvious from the plots, $L_1 = 5$ for both problems because S_e contains five elements. Therefore, $L_{\text{tot}} = 5 + 9 + 4 = 18$ for the original problem, and $L_{\text{tot}} = 5 + 2 + 1 = 8$ for the decorrelated problem. Thus, the decorrelated problem could be solved more than twice as rapidly as the original problem even if both algorithms started with the initial ellipsoid boundary $J_e = 0.2$.

This alternative goal for the decorrelation adjustment has been considered by other researchers.[‡] One could optimize the alternative criteria for $\bar{R}$ by seeking the adjustment that yields the best possible magnitude ordering of its diagonal elements. Unfortunately, determination of the optimal adjustment relies on the solution of a sequence of auxiliary ILLS problems. It is unwise to try to speed up the solution of one ILLS problem by solving a whole sequence of auxiliary ILLS problems.

[‡]Private communication from P. J. G. Teunissen.

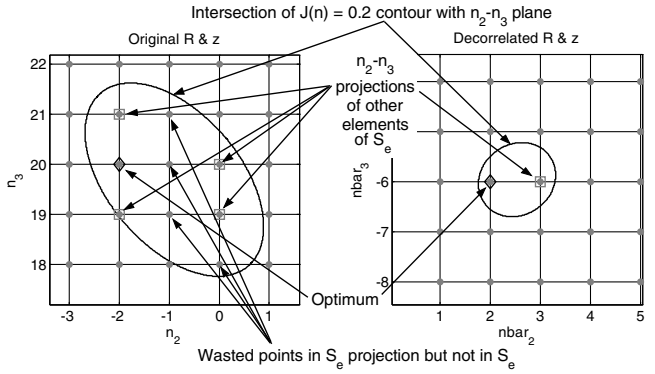

Fig. 4 Comparison of original and decorrelated projection of S_e onto the n_2–n_3 plane for a 3-dimensional example.

C. Two New Decorrelation Adjustment Algorithms

This section presents two new algorithms for computing the decorrelation adjustment matrices on the left-hand side of Eq. (13). The first algorithm is described in detail, and the second algorithm is developed in terms of the first algorithm and a series of transformations. The first algorithm operates directly on the R square-root information matrix, and the second algorithm operates on a square-root covariance matrix.

1. Utility Column Permutation Algorithm

A useful utility algorithm successively performs column permutations and QR factorizations on an input upper-triangular matrix R_{in} to produce the upper-triangular output matrix R_{out} in a way that successively makes the values $|R_{\text{out}(i)(i)}|$ for $i = 1, \ldots, (m-1)$ as small as possible. This procedure results in the following factorization of R_{in}:

$$Q_{\text{out}} R_{\text{out}} P_{\text{out}} = R_{\text{in}} \tag{16}$$

where Q_{out} is orthonormal, R_{out} is upper triangular, and P_{out} is a permutation matrix. All of these are $m \times m$ matrices. The steps of this factorization algorithm proceed as follows (with explanatory material following some steps):

1) Set $Q_{\text{out}} = I$, $R_{\text{out}} = R_{\text{in}}$, and $P_{\text{out}} = I$, and set $i = 1$.

2) If $i = m$, then stop.

Explanation: This is the normal point of termination and the starting point of each iteration of the algorithm's main loop.

3) Calculate the magnitudes of the vectors defined by the ith through mth elements of columns i through m of R_{out}:

$$\beta_j = \sqrt{\sum_{k=i}^{m} R^2_{\text{out}(k)(j)}} \quad \text{for } j = i, \ldots, m \tag{17}$$

Determine the index of the minimum magnitude partial column $j_{\min}$ such that $\beta_{j\,\min} \le \beta_j$ for $j = i, \ldots, m$, interchange columns i and $j_{\min}$ of R_{out}, and interchange rows i and $j_{\min}$ of P_{out}.

Explanation: Given a fixed choice for columns 1 through $(i-1)$ of R_{out}, this step chooses the ith column to ensure that the magnitude of the vector $[R_{\text{out}(i)(i)}; \ldots; R_{\text{out}(m)(i)}]$ is as small as possible. This magnitude will become the magnitude of $R_{\text{out}(i)(i)}$ during the next step.

4) Use QR factorization to compute the orthonormal matrix Q_{temp} and the upper-triangular matrix R_{temp} such that $Q_{\text{temp}} R_{\text{temp}} = R_{\text{out}}$. Afterward, set $R_{\text{out}} = R_{\text{temp}}$ and $Q_{\text{out}} = Q_{\text{out}} Q_{\text{temp}}$.

Explanation: This step re-upper triangularizes R_{out} because column i can have nonzero elements below the main diagonal as a result of step 3. If $j_{\min} = i$ in step 3, then this step is not needed. When needed, the QR factorization can be performed very rapidly using a QR update routine because the difference between R_{out} and an upper-triangular matrix is only rank 1 at the start of this step.

5) Replace i by $i + 1$ and go to step 2.

2. First Decorrelation Adjustment Algorithm

The first decorrelation adjustment algorithm is defined by the following steps with interspersed explanations:

1) Input the original ILLS problem's square, upper-triangular R matrix to the utility column permutation algorithm as R_{in}, and use the outputs of that algorithm in the assignments $Q = Q_{\text{out}}$, $\bar{R} = R_{\text{out}}$, and $Z = P_{\text{out}}$.

Explanation: This represents the first attempt to achieve ascending magnitudes in the diagonal elements of $\bar{R}$.

2) Set $i = m - 1$ and set modflag $= 0$.

Explanation: This is the return point of two loops. The inner loop works backward from $i = m-1$ to $i = 1$ and reduces the magnitudes of the elements above the main diagonal of $\bar{R}$, if possible, by appropriate modifications to Z. The outer loop tests whether any modifications have been made to $\bar{R}$ and Z during the inner loop by examining the flag "modflag." If modifications have been made, then the outer loop repermutes the columns of $\bar{R}$ to try to decrease the magnitudes of the initial diagonal elements and increase the magnitudes of the latter diagonal elements; that is, it tries to decrease $|\bar{R}_{ii}|$ for low values of i and increase $|\bar{R}_{ii}|$ for high values of i.

3) Calculate the integers

$$\gamma_k = \text{round}(\bar{R}_{ik}/\bar{R}_{ii}) \quad \text{for } k = (i+1), \ldots, m \tag{18}$$

Explanation: Nonzero γ_k integers occur when elements to the right of the main diagonal in row i of $\bar{R}$ have magnitudes that are more than half the magnitude of the ith diagonal element.

4) If any of the γ_k from step 3 are nonzero, then set modflag $= 1$ and perform the following modifications to $\bar{R}$ and Z:

$$\text{replace } \bar{R}_{jk} \text{ by } (\bar{R}_{jk} - \bar{R}_{ji}\gamma_k) \quad \text{for } j = 1, \ldots, i \quad \text{and} \quad k = (i+1), \ldots, m \tag{19a}$$

$$\text{replace } Z_{ik} \text{ by } \left(Z_{ik} + \sum_{j=i+1}^{m} \gamma_j Z_{jk}\right) \quad \text{for } k = 1, \ldots, m \tag{19b}$$

Otherwise, proceed to step 5.

Explanation: This step modifies $\bar{R}$ and Z if any of the γ_k integers are nonzero. It executes for $i = (m-1), \ldots, 1$ in the inner loop. At the end of the inner loop this step's cumulative effect leaves each diagonal element $\bar{R}_{jj}$ unchanged while modifying each $\bar{R}_{jk}$ for $k > j$, if needed, to achieve $|\bar{R}_{jk}| \le 0.5|\bar{R}_{jj}|$. These modifications to $\bar{R}$ do not affect the solution speed of the algorithm of Sec. III.C directly, but they have the potential to reduce the magnitudes of columns of $\bar{R}$ that corresponded to nonzero γ_k values. Such reductions are useful because they may provide the opportunity to improve the ordering of the magnitudes of the diagonal elements of $\bar{R}$ through a future column permutation.

5) If $i = 1$, then proceed to step 6; otherwise, replace i by $i - 1$ and return to step 3.

Explanation: This step tests for termination of the inner loop when $i = 1$ and branches appropriately.

6) If modflag $= 0$, then stop; the decorrelation factorization is complete. Otherwise, proceed to step 7.

Explanation: The outer loop terminates at this point if no modifications have been made by the inner loop. Otherwise, the outer loop proceeds to its column permutation step.

7) Input the $\bar{R}$ matrix to the utility column permutation algorithm as R_{in}. If $P_{\text{out}} = I$, that is, if no permutations have been performed by the algorithm, then stop. Otherwise, use the outputs of the permutation algorithm to replace Q by QQ_{out}, $\bar{R}$ by R_{out}, and Z by $P_{\text{out}}Z$ and return to step 2.

Explanation: The inner loop does not change the diagonal elements, and each column perturbation in this step reduces one diagonal element's magnitude while increasing another magnitude farther down the diagonal.

This algorithm exploits the fact that the product of the absolute values of the diagonal elements of $\bar{R}$ does not vary during its

execution. This product is constant because it equals $|\det(\bar{R})|$, which equals $|\det(R)|$ by virtue of Eq. (13). The constancy of this product implies that a decrease in the magnitude of any diagonal element must cause an increase in the magnitude of at least one other diagonal element. Any column permutation in steps 1 or 7 will lead to the decrease of at least one diagonal element, and the other diagonal elements that can increase can only be those further down the diagonal. Thus, the column permutation process brings $\bar{R}$ closer to the desired ordering of its diagonal element magnitudes.

This algorithm has no guarantee of finite termination of the outer loop. Computational experience on 2450 randomly selected problems suggests that the maximum number of outer-loop iterations is roughly $7.5 \times 10^{-5} m^4$ for problem dimensions in the range $30 \leq m \leq 50$ and that the mean number of outer-loop iterations is about 4 times smaller than this maximum value. This represents a rapid growth in execution time with problem dimension, but the leading coefficients are small enough to allow this algorithm to be used on problems with dimensions as large as $m = 50$.

This algorithm yields the following guaranteed minimum performance for the ordering of the magnitudes of the diagonal elements of $\bar{R}$:

$$|\bar{R}_{ii}| \geq \sqrt{\tfrac{3}{4}} |\bar{R}_{(i-1)(i-1)}| \quad \text{for } i = 2, \ldots, m \tag{20}$$

This bound is the direct result of the column ordering criterion of the utility column permutation algorithm and of the magnitude limitation of the off-diagonal elements that is enforced by Eqs. (18) and (19a). This conservative bound does not guarantee an increase in $|\bar{R}_{ii}|$ as i increases. Instead, it limits the amount of possible decrease. There is no guarantee that $|\bar{R}_{mm}| > |\bar{R}_{11}|$, and experience on larger problems (e.g., m between 40 and 50) indicates that $|\bar{R}_{mm}| < |\bar{R}_{11}|$ is common. Even so, the ratio $|\bar{R}_{mm}|/|\bar{R}_{11}|$ is typically much larger than the unadjusted ratio $|R_{mm}|/|R_{11}|$, which implies that this decorrelation adjustment expedites execution of the ILLS solution algorithm of Sec. III.C.

3. *Second Decorrelation Adjustment Algorithm*

The second decorrelation adjustment algorithm is like the first except that it works with an upper-triangular square root of the covariance of a reordered version of the unknown $\boldsymbol{n}$ vector. Direct operation on a covariance square-root matrix makes this second algorithm similar in spirit to the algorithm of [7]. The idea is to order conditional variances from smaller to larger values as much as possible and then to reverse that order when transforming back to a square-root information matrix form of the problem. In this way, the variance of $\bar{n}_m$ is kept as small as possible by allowing growth in the variance of $\bar{n}_1$.

The algorithm works as follows. Suppose that P_{inv} is the m-by-m permutation matrix that reverses the order of the rows or any matrix that it multiplies on the left and that reverses the order of the columns of any matrix that it multiplies on the right. It has all zeros except that $P_{\text{inv}(i)(m-i+1)} = 1$ for $i = 1, \ldots, m$. The square, upper-triangular matrix $R_{\text{cov}} = P_{\text{inv}} R^{-T} P_{\text{inv}}$ is a square root of the covariance matrix for the real-valued version of the reordered ambiguity vector $P_{\text{inv}}\boldsymbol{n}$. The notation $(\,)^{-T}$ refers to the inverse of the transpose of the matrix in question. The matrix R_{cov} is input to the first decorrelation adjustment algorithm in place of the matrix R to produce the factorization:

$$Q_{\text{cov}} \bar{R}_{\text{cov}} Z_{\text{cov}} = R_{\text{cov}} \tag{21}$$

so that the magnitudes of the diagonal elements of $\bar{R}_{\text{cov}}$ obey the condition in Eq. (20). These matrices are then used to compute the factorization of R as per Eq. (13) using the formulas

$$\begin{aligned} Q &= P_{\text{inv}} Q_{\text{cov}} P_{\text{inv}}, \\ \bar{R} &= P_{\text{inv}} \bar{R}_{\text{cov}}^{-T} P_{\text{inv}}, \quad \text{and} \quad Z = P_{\text{inv}} Z_{\text{cov}}^{-T} P_{\text{inv}} \end{aligned} \tag{22}$$

D. Competing Decorrelation Adjustment Algorithms

The computational investigation of Sec. V considers the performance of the two new decorrelation adjustment algorithms defined above and compares them with several existing algorithms. The classic algorithm of [7] is considered as is the algorithm of [22], which claims to have superior decorrelation performance. The algorithm from Sec. 1 of [23] is considered because Hassibi and Boyd [10] claim that it is superior to the algorithm of [7].

Two versions of the Ref. [23] algorithm are considered. One is the original algorithm followed by QR factorization to put $\bar{R}$ into upper-triangular form. The second version executes the Ref. [23] algorithm and then applies a right multiplication of the square-root information matrix and a left multiplication of Z both by P_{inv}. Finally, QR factorization is applied to put the square-root information matrix $\bar{R}$ into upper-triangular form. This second version of the Ref. [23] algorithm is needed because the ILLS solution algorithm of Sec. III. C is based on orthonormal/upper-triangular factorization of the square-root information matrix followed by backsubstitution. The algorithm of [10], however, is based on orthonormal/lower-triangular factorization followed by forward substitution. The reversal of the column ordering in the second version of the Ref. [23] algorithm makes it equivalent to the algorithm used in [10]. Note, however, that this reversal is likely to degrade the performance of the ILLS solution algorithm because the original decorrelation adjustment of [23] tends to have better magnitude ordering of the diagonal elements of the $\bar{R}$ matrix.

V. Computational Comparisons of Solution and Decorrelation Algorithms

A. Two Classes of Randomly Generated Example Problems

Randomly generated R matrices and z vectors have been used to define test problems for the ILLS solution algorithm of Sec. III and the least-squares ambiguity decorrelation adjustment algorithms of Sec. IV. This type of study is similar in spirit to the study reported in [22], but its details differ.

Two different recipes have been used for generating random problems with random numbers of ambiguities. The "kinematic" recipe generates small problems with each problem's dimension m sampled from the set $M_{\text{kine}} = \{3, 4, 5, \ldots, 9, 10\}$ with sampling probabilities of $\{1/14, 1/7, 1/7, \ldots, 1/7, 1/14\}$. The "network" recipe generates large problems with dimensions sampled from the set $M_{\text{netw}} = \{11, 12, 13, \ldots, 49, 50\}$ with sampling probabilities of $\{1/78, 1/39, 1/39, \ldots, 1/39, 1/78\}$. Given m, a random m-by-m matrix A_{raw} is generated by sampling each of its elements independently from a Gaussian distribution with zero mean and unit variance. The singular value decomposition of this matrix is then computed: $U\Sigma V^T = A_{\text{raw}}$, where U and V are orthonormal matrices and Σ is a diagonal matrix with positive diagonal elements. The condition number of R is then generated by sampling the base-10 logarithm of the condition number λ_c from a uniform distribution on the interval $[\lambda_{\text{clo}}, \lambda_{\text{chi}}]$ and then using it to redefine the diagonal elements of Σ: $\Sigma_{ii} = 10^{\lambda_c (i-1)/(m-1)}$ for $i = 1, \ldots, m$. The modified matrix $A_{\text{mod}} = U\Sigma V^T$ is formed using this modified Σ. A maximum measurement standard deviation in carrier cycles, $\sigma_{\max}$, is sampled randomly from a uniform distribution on the interval $[\sigma_{\max\text{lo}}, \sigma_{\max\text{hi}}]$, and this scalar standard deviation is used to define the square-root information matrix $A_{\text{inf}} = A_{\text{mod}}/\sigma_{\max}$. This matrix is then QR factorized to determine R for the ILLS problem: $Q_{\text{inf}} R = A_{\text{inf}}$. The elements of the m-dimensional z vector are generated as independent random samples from a Gaussian distribution with zero mean and unit variance.

The recipes use the following numerical limits for the distributions of λ_c and $\sigma_{\max}$. The kinematic recipe uses the limits $\lambda_{\text{clo}} = 0$ and $\lambda_{\text{chi}} = 4.5$ and the limits $\sigma_{\max\text{lo}} = 0.2$ cycles and $\sigma_{\max\text{hi}} = 3$ cycles. The network recipe uses the limits $\lambda_{\text{clo}} = 0$ if $m \leq 20$, $\lambda_{\text{clo}} = 3$ if $m > 20$, and $\lambda_{\text{chi}} = 4.5$ for all cases along with the limits $\sigma_{\max\text{lo}} = 0.2$ cycles and $\sigma_{\max\text{hi}} = 1$ cycles.

These problems are not actual GPS problems. The properties of actual GPS ambiguity resolution problems may differ enough to change the relative rankings of the algorithms as determined later.

These generic problems have been used because of the ease of generating them. The results generated in this section give an indication of the relative merits of the different algorithms, but further research will have to be done to determine which algorithms are better on actual GPS problems. The best sets of GPS ambiguity resolution problems for use in algorithm evaluation would have a strong geometry and, therefore, a high likelihood that a single ambiguity vector would be validated as the only likely one. Such problems are the only ones where ILLS techniques are considered to be useful.

B. Speed Comparison of ILLS Solution Algorithms

The execution speeds of two ILLS solution algorithms have been compared under various conditions. The new algorithm of Sec. III.C has been compared with the algorithm of [10], which is very similar to the algorithm of [7]. These algorithms have been encoded in MATLAB by the present authors. The quantity L_{tot} of Sec. III is used as a metric of the computational cost for the new method and for the method of [10]. The central processor unit (CPU) time t_{cpu} has been used as a second cost metric for both algorithms.

Table 1 compares the two solution algorithms' computation time metrics on 1000 randomly generated kinematic cases. Each solution algorithm's performance metrics are presented on a separate row of the table. The solution has been recalculated three different times using each algorithm and three different LAMBDA methods to precondition the problem. The three LAMBDA methods chosen for consideration were the best methods found. These are the two new methods of Sec. IV and the method of [7]. All of the solutions have been checked against each other to verify that each algorithm was executed correctly.

A comparison of the two lines of Table 1 shows that the new ILLS solution algorithm and the algorithm of [10] have similar performance. The new algorithm has a slightly lower average L_{tot} metric for all three LAMBDA methods, but its average execution time t_{cpu} is slightly higher for the last two LAMBDA methods. The maximum values of L_{tot} and t_{cpu}, on the other hand, indicate that the new algorithm may have a slight advantage.

A similar comparison has been performed for 1185 randomly generated network problems, that is, for larger problems than kinematic problems. These results are reported in Table 2, in which the performance metrics for each solution algorithm are reported in a separate row. The 1185 cases reported in the table are a subset of 1200 cases that have been run. Fifteen cases have been excluded because one of the two solution algorithms coupled with one of the three LAMBDA methods failed to find the solution with $L_{\text{tot}} \leq 1.5 \times 10^6$, at which point the algorithm was terminated prematurely to conserve computer time.

Table 2 indicates that the new solution algorithm of Sec. III.C is significantly faster than the solution algorithm of [10] when applied to the larger network problems. Each metric in the first row of the table is smaller than the corresponding value in the second row. The ratios of the second-row metrics to their corresponding first-row values range from 1.8 to 166, and the mean ratio of the mean L_{tot} and t_{cpu} metrics is 4.6. This comparison is further supported by the number of failures to find a solution with $L_{\text{tot}} \leq 1.5 \times 10^6$. The new algorithm experienced 0, 3, and 6 failures, respectively, when using the three LAMBDA methods associated with Table 2 to precondition the ILLS problem. The algorithm of [10], on the other hand, experienced 5, 7, and 7 failures, respectively, when using these same three preconditioning algorithms.

It seems surprising that the breadth-first approach executes more rapidly on large problems than does the depth-first approach of [7,9,10]. One might think that breadth first would tend to produce a larger number of candidate partial solutions that are not significant in the final analysis. This intuition appears to be false. As yet, no explanation has been found for this counterintuitive result.

C. Comparison of Decorrelation Adjustment Preconditioning Algorithms

Seven LAMBDA algorithms have been compared for their ability to reduce execution time for the ILLS solution algorithm of Sec. III. C. A set of randomly generated kinematic problems and a set of randomly generated network problems have been solved seven different times using the seven different LAMBDA methods to precondition the problem. The L_{tot} execution cost metric of this paper's new ILLS algorithm has been used as a measure of each LAMBDA algorithm's effectiveness.

Results for 1200 kinematic problems and 1250 network problems are reported in Table 3. The performance measures for each LAMBDA method occupy a row of the table. The first three data columns in a row correspond to the kinematic problems, and the last three columns correspond to the network problems. Good performance of a LAMBDA method is indicated by low values of the average and maximum L_{tot} values in the first, second, fourth, and fifth data columns and by low values of the number of failed cases in the third and sixth data columns. Note that the average and maximum L_{tot} values in columns 1 and 2 are calculated for the 1199 kinematic problems for which all of the LAMBDA methods allowed the ILLS solution algorithm to terminate with $L_{\text{tot}} \leq 5 \times 10^5$. The L_{tot} statistics in columns 4 and 5 are calculated for the 1242 cases in which the LAMBDA methods corresponding to the first three rows all produced ILLS solutions with $L_{\text{tot}} \leq 1.5 \times 10^6$. The mean and maximum L_{tot} values in the last four rows of columns 4 and 5 are

Table 1 Comparison of computation time metrics of two different ILLS solvers when using the three best LAMBDA preconditioning algorithms on 1000 kinematic cases

	First new LAMBDA method				Second new LAMBDA method				Ref. [7] LAMBDA method			
	L_{tot}		t_{cpu}, ms		L_{tot}		t_{cpu}, ms		L_{tot}		t_{cpu}, ms	
	avg	max	avg	max	avg	max	avg	max	avg	max	avg	max
New solution method	16.2	73	2.0	16	16.2	77	2.1	16	16.2	65	2.3	16
Ref. [10] solution method	16.4	127	2.2	16	16.4	115	1.9	47	16.5	125	1.8	16

Table 2 Comparison of computation time metrics of two different ILLS solvers when using the three best LAMBDA preconditioning algorithms on 1185 network cases

	First new LAMBDA method				Second new LAMBDA method				Ref. [7] LAMBDA method			
	L_{tot}		t_{cpu}, ms		L_{tot}		t_{cpu}, ms		L_{tot}		t_{cpu}, ms	
	avg	max	avg	max	avg	max	avg	max	avg	max	avg	max
New solution method	102	6908	12	313	242	1.1×10^5	21	6843	247	1.2×10^5	19	5750
Ref. [10] solution method	535	4.3×10^5	63	5.2×10^4	1258	4.8×10^5	156	6.1×10^4	447	2.5×10^5	53	3.1×10^4

Table 3 Comparison of ILLS solution cost metrics of seven different LAMBDA algorithms when used to precondition the new ILLS solver

	1200 kinematic problems			1250 network problems		
	L_{tot}		No. with $L_{tot} > 5 \times 10^5$	L_{tot}		No. with $L_{tot} > 1.5 \times 10^6$
	avg	max		avg	max	
First new method	16	92	0	396	2.3×10^5	3
Second new method	16	101	0	678	4.6×10^5	3
Ref. [7] method	16	107	0	859	5.2×10^5	6
Ref. [22] method	16	104	0	11617	1.5×10^6	12
Ref. [23] method	17	110	0	25038	1.5×10^6	28
Ref. [23]/[10] method	22	201	0	3.2×10^5	1.5×10^6	272
Descending diagonal mag. method	1369	1.4×10^5	1	3.0×10^5	1.5×10^6	254

lower than they would have been had their respective ILLS solution algorithms been allowed to run to completion for all 1242 cases.

It is clear from Table 3 that the first five LAMBDA methods have similar performance for kinematic problems while the first two methods are superior to the other methods for network problems. These two new methods yield lower average L_{tot} values for network problems, and they produce fewer cases in which the solution algorithm fails to terminate within the prescribed upper limit on L_{tot}.

In comparison to the first four algorithms, the LAMBDA method of [23] is significantly slower when solving network problems. It is also significantly less likely to enable solution subject to an upper bound on L_{tot}. Line 6 of Table 3 indicates that this algorithm is particularly poor when applied as in [10]. Its slowness in this situation comes about because of a misordering of the backsubstitution-like solution operations in relation to the LAMBDA method's ordering of $\bar{R}$ matrix diagonal element magnitudes.

The last line of Table 3 confirms the intuition that a proper ordering of the magnitudes of the diagonal elements of $\bar{R}$ is a key to good preconditioning of an ILLS problem. This last LAMBDA method intentionally orders the magnitudes opposite to the order that is thought to be good, that is, opposite to ascending order. This wrong type of decorrelation adjustment can be accomplished simply by column pivots during the QR factorization of $\bar{R}$. The matrix Z is a permutation matrix in this case. The results are disastrous: the computational cost of solving the ILLS problem skyrockets as does the likelihood of solution failure when the ILLS algorithm is subjected to a maximum bound on L_{tot}.

The practical significance of Table 3 is illustrated by considering the execution times for the algorithms. Although not recorded directly for the Table 3 cases, a particular case's CPU time can be estimated as a function of its L_{tot}. These estimates use a linear fit of the L_{tot} and t_{cpu} data that has been used to produce the first line of Table 2. The resulting fit yields $t_{cpu} \cong 5.7 \times 10^{-2} L_{tot}$ ms/problem for the MATLAB ILLS software to execute on a 3 GHz processor. Using this fit, the maximum execution time for a kinematic problem is only $t_{cpu} = 11.5$ ms/problem if one excludes the poorest LAMBDA algorithm, that is, the one corresponding to the last line of Table 3. Thus, the choice of LAMBDA method does not have a large impact for kinematic problems. For network problems, however, different LAMBDA methods can lead to dramatically different execution times. The average execution time can be as low as $t_{cpu} = 23$ ms/problem for the best new LAMBDA algorithm, but it can range up to $t_{cpu} = 662$ ms/problem for the LAMBDA method of [22] and up to $t_{cpu} = 18,000$ ms/problem for the LAMBDA method of [23]/[10]. The average time savings that results from using the best new algorithm on network problems can range from 26 ms/problem up to 18,000 ms/problem, depending on which existing LAMBDA method is being replaced.

VI. Experience with Real Data and with High-Fidelity Simulated Data

The new algorithms of this paper have been tested using actual field data and using data from high-fidelity simulators. Field data have been collected using survey-grade GPS receivers, and the algorithms have been applied to the data in order to estimate the lengths of known baselines. Another set of tests has used data from a GPS RF signal simulator and a space-qualified hardware receiver to evaluate the new algorithms' operation when applied to the problem of spacecraft formation flight [5]. A third set of tests has used spacecraft formation flight data that has been generated by a high-fidelity offline GPS signal simulation [5,6]. The new algorithms have performed well in all of these tests, and their performance has been similar to that found in the offline study of Sec. V. These tests demonstrate the new algorithms' ability to work in nonideal situations in which some noise sources, such as multipath, exhibit non-Gaussian distributions and time correlations.

VII. Conclusions

This paper has reviewed the subject of mixed real/integer linear least-squares problems and has developed new algorithms for solution of the difficult integer portion of such problems. One of the new algorithms is a modified form of an existing solution algorithm that uses ideas from backsubstitution of upper-triangular systems to compute the set of all integer-valued vectors whose cost is no higher than a given target value. The modification replaces a depth-first search with a breadth-first search. The algorithm efficiently determines the elements of the target set, and the global minimum solution is computed by a brute-force search within this set.

Two new preconditioning algorithms have been developed that help to expedite the solution of the integer linear least-squares problem. These fall in the class known as least-squares ambiguity decorrelation adjustment methods. They calculate unimodular transformations that bound the rate of decrease of the magnitudes of the diagonal elements of an upper-triangular square-root information matrix or an upper-triangular square-root covariance matrix. This bounding tends to decrease the number of calculations required by the solution algorithm.

These new algorithms have been shown to be comparable to existing algorithms for small problems with 3–10 integer unknowns, but they reduce computational costs significantly when applied to larger problems with 11–50 integer unknowns. In this latter case, the new solution algorithm is more than 4 times faster than its leading competitor, on average, and the best of the two new preconditioning algorithms halves the average time to solution.

Acknowledgements

This work has been supported in part by NASA Goddard Space Flight Center through cooperative agreement No. NCC5-722, with Michael Moreau as the agreement monitor, and through a NASA Graduate Student Researcher Program Fellowship, with Russell Carpenter as the fellowship monitor. The authors wish to thank Peter Teunissen of the Department of Mathematical Geodesy and Positioning at the Delft University of Technology for providing many useful comments about this work.

References

[1] Cohen, C. E., "Attitude Determination," *Global Positioning System:*

Theory and Applications, edited by B. W. Parkinson, and J. J. Spilker, Jr., Vol. II, AIAA, Washington, D.C., 1996, pp. 519–538.

[2] Goad, C., "Surveying with the Global Positioning System," *Global Positioning System: Theory and Applications*, edited by B. W. Parkinson, and J. J. Spilker, Jr., Vol. II, AIAA, Washington, D.C., 1996, pp. 501–517.

[3] Busse, F. D., How, J. P., and Simpson, J., "Demonstration of Adaptive Extended Kalman Filter for Low-Earth-Orbit Formation Estimation Using CDGPS," *Navigation*, Vol. 50, No. 2, 2003, pp. 79–93.

[4] Leung, S., and Montenbruck, O., "Real-Time Navigation of Formation-Flying Spacecraft Using Global-Positioning-System Measurements," *Journal of Guidance, Control, and Dynamics*, Vol. 28, No. 2, March–April 2005, pp. 226–235.

[5] Mohiuddin, S., and Psiaki, M. L., "Satellite Relative Navigation Using Carrier-Phase Differential GPS with Integer Ambiguities," *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, AIAA, Reston, VA, 2005.

[6] Psiaki, M. L., and Mohiuddin, S., "Modeling, Analysis, and Simulation of GPS Carrier Phase for Spacecraft Relative Navigation," *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, AIAA, Reston, VA, 2005.

[7] Teunissen, P. J. G., "The Least-Squares Ambiguity Decorrelation Adjustment: A Method for Fast GPS Integer Ambiguity Estimation," *Journal of Geodesy*, Vol. 70, Nos. 1–2, Nov. 1995, pp. 65–82.

[8] Chen, D., and Lachapelle, G., "A Comparison of the FASF and Least-Squares Search Algorithms for On-The-Fly Ambiguity Resolution," *Navigation*, Vol. 42, No. 2, 1995, pp. 371–390.

[9] de Jonge, P., and Tiberius, C., "The LAMBDA Method for Integer Ambiguity Estimation: Implementation Aspects," Delft Geodetic Computing Centre, LGR series, No. 12, available online at http://enterprise.lr.tudelft.nl/mgp/modules.php?op=modload&name=Lambda&file= index&func=displaypapers, Aug. 1996, pp. 1–41 [retrieved 21 Dec. 2005].

[10] Hassibi, A., and Boyd, S., "Integer Parameter Estimation in Linear Models with Applications to GPS," *IEEE Transactions on Signal Processing*, Vol. 46, No. 11, Nov. 1998, pp. 2938–2952.

[11] Psiaki, M. L., "Batch Algorithm for Global-Positioning-System Attitude Determination and Integer Ambiguity Resolution," *Journal of Guidance, Control, and Dynamics*, Vol. 29, No. 5, Sept.–Oct. 2006, pp. 1070–1079.

[12] Hofmann-Wellenhof, B., Lichtenegger, H., and Collins, J., *GPS Theory and Practice*, 4th ed., Springer–Verlag, New York, 1997, pp. 214–249.

[13] Teunissen, P. J. G., "GPS Carrier Phase Ambiguity Fixing Concepts," *GPS for Geodesy*, edited by P. J. G. Teunissen and A. Kleusberg, 2nd ed., Springer, New York, 1998, pp. 319–388.

[14] Misra, P., and Enge, P., *Global Positioning System Signals, Measurements, and Performance*, 2nd ed., Ganga–Jamuna Press, Lincoln, MA, 2006, pp. 258–272.

[15] Gill, P. E., Murray, W., and Wright, M. H., *Practical Optimization*, Academic Press, New York, 1981, pp. 30–31, 37–40.

[16] Bar-Shalom, Y., Li, X.-R., and Kirubarajan, T., *Estimation with Applications to Tracking and Navigation*, Wiley, New York, 2001, pp. 47–49, 92–98.

[17] Wolfe, J. D., Williamson, W. R., and Speyer, J. L., "Hypothesis Testing for Resolving Integer Ambiguity in GPS," *Navigation*, Vol. 50, No. 1, 2003, pp. 45–56.

[18] Joosten, P., and Tiberius, C., "Fixing the Ambiguities: Are You Sure They're Right?," *GPS World*, Vol. 11, No. 5, May 2000, pp. 46–51.

[19] Verhagen, S., and Teunissen, P. J. G., "New Global Navigation Satellite System Ambiguity Resolution Method Compared to Existing Approaches," *Journal of Guidance, Control, and Dynamics*, Vol. 29, No. 4, July–Aug. 2006, pp. 981–991.

[20] Teunissen, P. J. G., "GNSS Ambiguity Bootstrapping: Theory and Application," *Proceedings of the KIS2001, International Symposium on Kinematic Systems in Geodesy, Geomatics, and Navigation*, University of Calgary, Calgary, Canada, 5–8 June 2001, pp. 246–254.

[21] Verhagen, S., and Teunissen, P. J. G., "On the Probability Density Function of the GNSS Ambiguity Residuals," *GPS Solutions*, Vol. 10, No. 1, Feb. 2006, pp. 21–28.

[22] Xu, P., "Random Simulation and GPS Decorrelation," *Journal of Geodesy*, Vol. 75, Nos. 7–8, Sept. 2001, pp. 408–423.

[23] Lenstra, A. K., Lenstra, H. W., and Lovász, L., "Factoring Polynomials with Rational Coefficients," *Mathematische Annalen*, Vol. 261, No. 4, Dec. 1982, pp. 515–534.