The Sequential Givens Method for Adjustment Computations in Photogrammetry

by

Theodore David Johnson

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Civil Engineering

APPROVED:

Steven D. Johnson, Chairman

Christopher A. Beattie

Roger W. Ehrich

May, 1988

Blacksburg, Virginia

The Sequential Givens Method for Adjustment Computations in Photogrammetry

by

Theodore David Johnson

Steven D. Johnson, Chairman

Civil Engineering

(ABSTRACT)

The Givens orthogonalization algorithm is an efficient alternative to the normal equations method for solving many adjustment problems in photogrammetry. The Givens method is one of a class of methods for solving linear systems known generally as orthogonalization or *QR* methods. It allows for sequential processing and greatly simplifies the computation of statistics on the observations and residuals. The underlying reason for these advantages is the immediate availability of the orthogonal *Q* matrix, which is computed as the data are processed and is intimately related to the statistics needed for blunder detection. One of these statistics, the F statistic computed from externally studentized residuals, is both easily obtained and well-suited for blunder detection.

The Givens method requires nearly four times the number of computations as compared to the normal equations approach in order to reach a solution. However, depending on the size of the problem, blunder detection through the normal equations requires far more computer time than is required when starting with a Givens decomposition.

The method allows a user to review intermediate results, test residuals and modify the solution without having to compute a full solution. Adjustments of a level net and a single-photo resection are used to demonstrate the method.

Because of the advantage in computational time, the Givens method is superior to the normal equations approach when rigorous blunder detection is required.

Acknowledgements

I would like to thank Drs. Steven D. Johnson and Christopher A. Beattie for their guidance and patience, and Dr. Roger W. Ehrich for his interest in this project. I would also like to thank my wife, , for her loving encouragement when I needed it.

Table of Contents

List of Illustrations

List of Tables

The Sequential Givens Method for Adjustment Computations in Photogrammetry

1. Introduction and motivation for the project

1.1 Photogrammetric adjustments

The art of adjusting photogrammetric measurements consists of fitting measurements of object coordinates, of image coordinates of those object coordinates, and of camera lens locations and orientations into a physically realistic mathematical model. The term "photogrammetric adjustments" covers a wide range of applications, from recovering object coordinates from measurements on two oriented images, to the simultaneous orientation and control densification of hundreds of images. In all cases, the basic task is to determine the exact geometry of the camera, image and object for each image. The technology of phototriangulation is well-known, primarily for the special case of aerotriangulation. Analytic methods are widely used and typically center on the normal equations approach to least-squares optimization.

Beyond the assigning of coordinates to points, we may ask more of a photogrammetric adjustment: to identify spurious image measurements (blunders), to apply constraints to the coordinates, to use *a priori* statistical properties of the observations to determine the statistical properties of the coordinates, to accept partial data sets and give back intermediate answers. In many ways, the normal equations approach is ideal for the task. In particular, statistical properties are typically expressed as matrices of the form B^TB, which carries directly over to the formulation of the normal equations: given

$$\min(Bx - f)^T(Bx - f)$$

the least squares solution is

$$x = (B^TB)^{-1}B^Tf.$$

However, the normal equations approach does not lend itself easily to all of these additional tasks. Intermediate results are obtained only by interrupting the process of data accumulation, and blunder detection is often an entirely separate operation. In addition, the normal equations approach involves the formation of B^TB, which leads to exaggerated numerical instability.

1.2 Aspects of the Givens method

A promising method, which has only recently been applied to photogrammetry and surveying, uses the so-called *Q-R* decompositon (e.g., Golub, Luk and Pagano, 1979; Blais, 1983; Teskey, 1983). This involves orthogonalizing the coefficient matrix, typically using sequential Givens rotations, and then solving the resulting triangular system of equations. This promises a number of advantages over the normal equations approach, including the following:

- Numerical stability is that of the problem itself and is not exaggerated by the solution technique.

- The inherent numerical instabilities of a problem are easily identified.

- Intermediate solutions may be calculated with little trouble, even in cases where there is rank-deficiency (insufficient data).

- Most of the computation time can be hidden between the moments when the user actually collects data.

- Blunder detection using the R-student statistic may be done easily by computing and manipulating the residuals directly without computing the unknown parameters.

- The algorithm adapts easily to handle robust estimators based on least squares.

The objective of this thesis is to investigate the applicability of the Givens orthogonalization algorithm to a simple photogrammetric problem, the resection of a single photograph from known control points. We will concentrate on two aspect of the algorithm: sequential solution and blunder detection. Other advantages will be mentioned in passing but not developed. For example, it will become apparent that a simple variation leads directly to a certain class of robust estimators, those in which the weights are varied within a least squares solution. However, we will not cover this topic in any detail. And although the wish for greater numerical stability is a prime motivator for the development of orthogonalization methods for many least squares applications, and although numerical instability can be a problem in photogrammetric adjustments, it is not nearly as great a problem as undetected blunders. The stability of orthogonalization methods is well known, and so, as with robust estimators, this topic will not be discussed in detail.

Blunder detection and orthogonalization would appear at first glance to have nothing to do with one another. There is no reason to expect that blunder detection would be made any easier because we solved a least squares problem differently. Yet this is precisely what we will find: that the method of solution greatly simplifies the task of detecting blunders. The principal disadvantage of the Givens orthogonalization algorithm for standard least-squares solutions is the amount of additional computer time required. When we consider blunder detection as a part of the solution, we find that this is more than compensated for. With a normal-equations solution, one must compute a very large matrix of order equal to the number of observations; with any orthogonalization method, this matrix is readily available. In this case, having computed an intermediate solution, the test of any single observation involves little more than the formation of an inner product. (Tests for multiple blunders are likewise simplified.)

The other advantage of the Givens orthogonalization algorithm that interests us here, its sequential nature, is only heightened by the simplified blunder detection. The use of the Givens orthogonalization algorithm for sequential solutions is becoming more common in the photogrammetric community (see, for example, Gruen, 1985 and Runge, 1987). The ideal application for a sequential method is during actual measurement: while the computer waits for the operator to collect the next datum, the solution matrices can be updated. Ideally, blunders would be detected at this time, so that the operator could immediately remeasure. The fact that one of the best sequential adjustment methods, the Givens orthogonalization algorithm, leads naturally to blunder detection has not been widely recognized. As this recognition comes, Givens could effectively supplant other least squares methods in a wide class of problems.

1.3 Examples

Two examples are used: a simple level net and close-range photogrammetric strip. The level net is much simpler, being a linear problem with only a few observations and parameters. This is used to show the details of the method. The photogrammetric resection is used to prove the method in a larger, non-linear case. It is much easier to see the essentials of the Givens orthogonalization algorithm in the simpler problem, in which we can limit the number of unknowns and not have to deal with non-linearity. The desired results are obvious before any solution is performed - we can see the blunders in the data before they are detected automatically in the course of solution.

2. Development of the Givens method

2.1 Formulation of the photogrammetric adjustment problem

Photogrammetric adjustment problems usually involve finding the best estimators for a group of parameters subject to certain conditions which have an error term. In this context, "best" means the solution that minimizes the sum of the weighted squares of the estimated errors, or residuals. Rao (1973, p. 232) shows that the solution to the constrained least-squares problem has the same statistical properties as the solution to the unconstrained least-squares problem, i.e., it yields unbiased parameter estimates that have minimum overall variance. Note that the means by which the minimization is achieved is not specified. A (local) minimum is necessarily unique, so that we are free to choose our method and be assured of the same statistical properties. In particular, we wish to choose a method that is at least as good as the normal equations approach, since that represents the most common method used today.

The least-squares phototriangulation problem may be represented by the following linear statistical model:

$$(\varepsilon + l) + Bx = d \qquad 2.1.1$$

or

$$\varepsilon + Bx = f \qquad 2.1.2$$

with stochastic model

$$\text{variance}(\varepsilon) = \Sigma.$$

Letting v estimate ε, the problem may be written as

$$\min(v^T W v) \quad \text{subject to} \quad v + Bx = f, \qquad 2.1.3$$

where

$l \in \mathbf{R}^{c\times 1}$ is the vector of observations,

$\varepsilon \in \mathbf{R}^{c\times 1}$ is the vector of unknown errors in the c condition equations,

$v = \hat{\varepsilon} \in \mathbf{R}^{c\times 1}$ is the vector of estimates of ε,

$d \in \mathbf{R}^{c\times 1}$ is a constant vector,

$f = d - l \in \mathbf{R}^{c\times 1}$,

$x \in \mathbf{R}^{u\times 1}$ is the vector of unknown parameters:

rotation angles ω (about x), ϕ (about y) and κ (about z)

and camera lens positions X_L, $Y_L$and Z_L,

$B \in \mathbf{R}^{c\times u}$ is the coefficient matrix for x in the condition equations,

$W \in \mathbf{R}^{c\times c}$ is the weight matrix for the residuals, c is the number of condition equations,

u is the number of unknown parameters,

$c \geq u$.

$$W = V^{-1} = \frac{1}{\sigma_0^2} \Sigma \qquad 2.1.4$$

Σ = variance(ε)

σ_0^2 is the *a priori* reference variance for statistical testing

V is the cofactor matrix for ε.

Note that Σ represents the variance of ε, not of v. Since we assume a variance for all errors included in the model, Σ is positive definite and so W is also positive definite. The use of W as the weight matrix for the residuals and the interpretation of V as variance (ε) (Mikhail, 1976, p. 104) are well founded in statistical theory (Rao, 1973, sec. 4a.1), provided that the model is correct, i.e, the variance structure of the errors is correctly represented by Σ. Note also that σ_0^2 is included only for statistical convenience and has no effect on the minimization.

Implicit in our development of adjustment techniques for phototriangulation will be the assumption that all errors in the observations are normally distributed. This may seem unnecessarily arbitrary, but it is really just a simplified route to a fact that is otherwise true. Various formulations of the Central Limit Theorem in statistics (see, for example, Rao, 1973, pp. 127-128) state that as the number of observations increases, however they are distributed, the distribution of derived quantities linearly related to those observations will approach the multinormal distribution (again unbiased and with minimum variance). Because we will use a linear model and many observations, this implies that our parameters, as well as other derived quantities, will have a multinormal distribution. Thus, the assumption of normality is not necessary, it only simplifies matters. Since we will be interested in variance propagation and in developing statistical tests, it will be very useful to be able to assume normality in the parameters and in other derived quantities.

Because we assume that the only covariance is between x and y measurements of the same point on a photograph, which will be represented by adjacent rows in the condition equations, Σ will have at most 2×2 blocks on the diagonal. Typically Σ will be diagonal, with no covariances at all. This is the typical assumption and is used here. In other cases, Σ might be block diagonal.

2.2 *Solving the least squares problem using Givens rotations*

2.2.1 The QR decomposition

The typical treatment of the least squares problem in elementary textbooks (e.g., Mikhail, 1976 or Myers, 1985) utilizes differential calculus to obtain the minimum-residual solution. An alternative derivation uses only linear algebra and leads directly to application of the *QR* decomposition.

It will be convenient to minimize norms rather than inner products. If we define

$$\|x\|_2 = \sqrt{x^T x} \quad \text{(the euclidean length of } x\text{)}$$

then

$$\|x\|_2^2 = x^T x.$$

Also, note that since any orthogonal matrix Q represents a non-deforming rotation possibly combined with a reflection, the length of Qx is the same as the length of x, or

$$\|Qx\|_2 = \|x\|_2$$

$$\|Qx\|_2^2 = \|x\|_2^2.$$

In the simplest case, ignoring constraints, we will solve

$$\min(v^T W v) \quad \text{subject to} \quad v + Bx = f \tag{2.2.1}$$

using the nomenclature defined above. Letting

$$W = LL^T \tag{2.2.2}$$

and

$$u = L^T v \tag{2.2.3}$$

$$v = L^{-T} u \quad (\text{where } L^{-T} = (L^T)^{-1}\ ) \tag{2.2.4}$$

we have

$$\min\|u\| \quad \text{subject to} \quad u + L^T Bx = L^T f \tag{2.2.5}$$

or

$$\min\|L^T Bx - L^T f\|_2. \tag{2.2.6}$$

L could be obtained through cholesky decomposition or eigenvalue decomposition. The normal equations solution to this problem is

$$B^T LL^T Bx = B^T LL^T f, \quad \text{solving for } x\,,$$

or

$$B^T W Bx = B^T W f. \tag{2.2.7}$$

Another solution to this problem may be derived by considering the minimization directly without forming the normal equations. We will begin by working with the square of the 2-norm; whatever minimizes this will also minimize the 2-norm.

$$\min\|L^T Bx - L^T f\|_2^2 = \min\|Q^T L^T Bx - Q^T L^T f\|_2^2 \qquad 2.2.8$$

$$= \min\|Rx - Q^T L^T f\|_2^2$$

where

$$Q \in \mathbf{R}^{c \times c}, \quad \text{orthogonal,}$$

$$R = Q^T L^T B \in \mathbf{R}^{c \times u}, \quad \text{upper triangular.}$$

Setting

$$R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \qquad 2.2.9$$

$$R_1 \in \mathbf{R}^{u \times u}, \quad \text{upper triangular}$$

and

$$Q = [Q_1 \;\; Q_2] \qquad 2.2.10$$

$$Q_1 \in \mathbf{R}^{c \times u}$$

$$Q_2 \in \mathbf{R}^{c \times (c-u)}$$

we then have

$$\min\|L^T Bx - L^T f\|_2^2 = \min\left\|\begin{bmatrix} R_1 \\ 0 \end{bmatrix} x - \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix} L^T f\right\|_2^2 \qquad 2.2.10$$

$$\min \left\| \begin{bmatrix} R_1 x - Q_1^T L^T f \\ 0 - Q_2^T L^T f \end{bmatrix} \right\|_2^2$$

$$= \min\left[\|R_1 x - Q_1^T L^T f\|_2^2 + \|Q_2^T L^T f\|_2^2 \right]$$

$$= \min\left[\|R_1 x - Q_1^T L^T f\|_2^2 \right] + \|Q_2^T L^T f\|_2^2 \text{ since } f \text{ is known.}$$

Finally,

$$\min\left[\|R_1 x - Q_1^T L^T f\|_2^2 \right] = 0 \qquad 2.2.11$$

when we solve the system

$$R_1 x = Q_1^T L^T f, \qquad 2.2.12$$

giving us the least-squares solution. A method which decomposes a matrix into an orthogonal factor (Q) and an upper triangular factor (R) is known as a QR method. This decomposition is not unique; for this thesis it will be computed using Givens rotations. It is always computable, regardless of any column rank deficiency in B. Assuming exact arithmetic, the number of non-zero rows in R will equal Rank(B).

L appears in this formulation only to premultiply B and f, and it will appear in our solution only when we obtain $v = L^{-T}u$. The present development will be greatly simplified if we assume that the weights, in the form of L, are applied at the beginning and then ignored until the final residuals are needed. The method will be just as rigorous if we apply this transformation to the data. In particular, blunder detection will be much simpler if we treat the transformed residuals rather than the true residuals. Thus, for the remainder of this thesis, it will be assumed that this transformation, the application of weights on the observations, has already been performed on the incoming data, and that it will be reversed when the final residuals are required. This will allow us to focus more directly on the problem at hand, the implementation of a Givens orthogonalization algorithm, and leave questions of weighting as a side issue. For notation, we will use B, f and v to represent the transformed matrices.

There are two statistically important matrices which we may define now for use later,

$$(B^T B)^{-1} = (R^T Q^T Q R)^{-1} = (R^T R)^{-1} = (R_1^T R_1)^{-1} = R_1^{-1} R_1^{-T} \qquad 2.2.13$$

(covariance matrix for the parameters), and

$$B(B^T B)^{-1} B^T = Q_1 R_1 R_1^{-1} R_1^{-T} R_1^T Q_1^T = Q_1 Q_1^T \qquad (\neq I!) \qquad 2.2.14$$

(used in outlier [blunder] detection: see Beisley, Kuh and Welsch, 1980, pp. 72-73). The latter is also referred to as H.

2.2.2 The Givens QR decomposition

In this thesis, the QR decomposition will be computed using Givens rotations. A Givens rotation matrix represents a simple plane rotation. The general rotation matrix is

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}.$$

This matrix is always constructed in order to eliminate a specific element of B, b_{ji} by the computation

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} b_{ii} \\ b_{ji} \end{bmatrix}$$

In order to do this, we apply the rotation to the ith and jth rows of B. This is equivalent to applying this rotation to the i,j plane in $\mathbf{R}^m$. To do this we may construct the appropriate rotation matrix, Q^{ji} by starting with the identity matrix and making four substitutions, as follows:

$$\begin{bmatrix} q_{ii} & q_{ij} \\ q_{ji} & q_{jj} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

$$Q^{ij} = \begin{array}{c} \\ \\ \\ i \\ \\ j \\ \\ \\ \\ \end{array} \begin{bmatrix} 1 & 0 & \dots & 0 & \dots & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & \dots & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \cos\theta & \dots & \sin\theta & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & -\sin\theta & \dots & \cos\theta & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & \dots & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & \dots & 0 & \dots & 0 & 1 \end{bmatrix}$$

(In the sequel we often write J_k for Q^{ij} , to avoid having to keep track of rows and columns.)

When $B' = Q^{ij}B \in \mathbf{R}^{m\times m}$, the intermediate result, is formed, the following are important to bear in mind:

- θ is chosen such that the j,i entry of B becomes zero.

- The euclidean matrix norm is invariant under orthogonal transformations, so B' has the same numerical conditioning as B

- Only the ith and jth rows of B are affected.

- The computations for the kth column are

$$b_{ik} := \cos\theta b_{ik} + \sin\theta b_{jk}$$

$$b_{jk} := -\sin\theta b_{ik} + \cos\theta b_{jk}$$

and so the following rules hold:

- If b_{ik} or b_{jk} is nonzero, b_{ik} and b_{jk} will become nonzero,
- If b_{ik} and b_{jk} are zero, b_{ik} and b_{jk} will become zero.

Thus, Q is very selective and predictable in its influence on the zero-nonzero structure of B. We will eliminate entries of B in the following order:

- eliminate a_{21} against a_{11}
- eliminate a_{31} against a_{11}
- eliminate a_{32} against a_{22}
- ...
- eliminate a_{ij} against a_{jj} ($i > j$)
- ...
- eliminate $a_{n,n-1}$ against a_{11}

(Note that the entries of B change with each step.)

After the first n rows of B are rendered upper triangular, each succeeding row of B is completely eliminated. When the m^th row of B has been eliminated, we set the resulting upper triangular matrix equal to R and we have completed the computation

$$B = QR.$$

We have used θ as a parameter of our Givens rotation. It should be noted that θ need not be computed explicitly. Since θ is not a useful quantity by itself, we do not compute it. Instead, we compute $\cos\theta$ and $\sin\theta$ directly:

$$\cos\theta = \frac{b_{ii}}{\sqrt{b_{ii}^2 + \text{[illegible]}}} \qquad 2.2.14$$

$$\sin\theta = \frac{b_{ji}}{\sqrt{b_{ii}^2 - b_{ji}^2}} . \qquad 2.2.15$$

The QR decomposition has thus far been assumed to exist. This is relatively easy to demonstrate in the case of the Givens orthogonalization algorithm. The pathological condition occurs when b_{ii} and b_{ji} approach 0, causing the denominator to approach 0. The algorithm must check for this; when it occurs, we allow $Q^{ij} = I$. There is no case in which the algorithm cannot proceed.

The operational difficulty here is in determining whether an entry is so close to zero that its value is simply roundoff error. Such entries should be considered to be zero, but their identification is problematic. It is beyond the scope of this thesis to consider this problem further. Photogrammetric problems have well-known geometries, each parameter having a physical meaning. Aside from some well-known cases, in which object points and perspective centers lie on a "critical surface," it is known each parameter in a phototriangulation problem can be solved for, and we will make that assumption from here on. In the course of matrix decomposition, certain entries must become zero, and so they are set to zero explicitly. Aside from those that are easily predicted, we can assume that there are no computed zeroes in the solution matrices.

2.2.3 Some useful expressions

It will be useful at this time to collect, for later use, the expressions for some important vectors, Δ, $\hat{f}$, $\hat{l}$ and v .

$$\Delta = R_1^{-1} Q_1^T f \qquad 2.2.16$$

$$\hat{f} = B\Delta = QR\Delta = Q_1 R_1 \Delta = Q_1 R_1 R_1^{-1} Q_1^T f = Q_1 Q_1^T f = Hf (\text{Recall: } H = Q_1 Q_1^T) \qquad 2.2.17$$

$$\hat{l} = d - \hat{f} = d - Hf \qquad 2.2.18$$

$$v = f - \hat{f} = f - Hf = (I - H)f \qquad 2.2.19$$

With the exception of Δ, none of the vectors above depends on the solvability of the system or the availability of an inverse. The system may be rank deficient, yet we can compute $\hat{f}$, $\hat{l}$, and v with no difficulty, because Q and R are always defined. In practice, if we do this, the values we get are meaningless: $\hat{f}$ will be the same as f, $\hat{l}$ will be the same as l for the observations which cannot be recomputed from the existing Δ, and v will thus be zero for these observations.

2.2.4 Storing Q

We can preserve a representation of Q, rather than Q itself, in the subdiagonal area of B (noting that these are the elements reduced to zero by the orthogonalization algorithm), thus taking up no additional space. Many applications will not need Q as a whole matrix, but rather individual rows and columns. These may be generated on demand. The representation of Q used would be the ratio b_{ji}/b_{ii}; this is sufficient to generate P_{ji} and D_{ji}.

Blunder detection makes extensive and frequent use of Q. Thus, for the relatively small application at hand, we will assume that the host computer has enough storage space, making it more efficient to simply compute Q on the spot and store it whole. Otherwise, a great deal of redundant computation would be necessary to compute Q and H every time they are needed.

2.3 Computing externally studentized residuals for blunder detection

Only the omniscient among us can detect blunders in a data set; the rest of us have to guess. Because a phototriangulation adjustment can be ruined by blunders, there have been developed numerous schemes to detect them. In the photogrammetric and geodetic literature, Baarda (1960,1968), Pope (1976) and Kok (1984) illustrate the various approaches to the problem. All follow the same principle, asking: Does an observation fit the overall trend of the data? How sure can we be of our conclusion? The first of these questions is easy to answer. The difficulty comes later, when we must formulate statistical tests to answer the second question.

Externally studentized residuals were chosen for three reasons: they provides an answer to a rigorous question, they are convenient to calculate, and they follow the well-known F distribution which gives us greater power in our tests. Convenience is not a criterion for choice by itself, but it is not a trivial issue either. One does not, for example, want to solve the entire phototriangulation before searching for blunders. The approach taken here is to apply blunder detection at intervals, while the solution is being formed. At the same time, we may change the α level (the probability of missing a blunder). In this way, while the solution is incomplete we can be lax and purge only the most obvious blunders; when all the data are in we can be more stringent. The use of a well-known distribution is less important, and the advantages are mostly hypothetical. The F distribution is part of a family of related distributions which also includes the normal, χ^2 and Student's t distributions, allowing us more flexibility should we need it later.

We will want to test for single blunders, for groups of blunders, and for single blunders within a larger group of removed observations. Rather that pursue separate derivations, we will derive the statistic for testing an entire group of observations, referring to the set of removed observations as Z, containing m observations.

In testing the observations in Z, we might use the sum of their squared deviations from the predicted values as a measure of how well these observations fit with the rest of the data. This leads naturally into the F test, in which we compare this sum of squares against the residual sum of squares for the regression *without using the Z observations.* In order to use the F statistic, we need to ensure the independence of the two sums of squares; in this case, we do indeed have that. Letting SS_{-Z} be the sum of squares for the adjustment without the observations in Z, $SS_{Z,-Z}$ be the sum of squares for the observations in Z, as computed using the adjustment which excludes them, df be the degrees of freedom for the full adjustment, including the observations in Z, df_{-Z} be the degrees of freedom for the adjustment without the the observations in Z, and $df_{Z,-Z}$ be the degrees of freedom associated with $SS_{Z,-Z}$, we can write our test statistic as

$$F_{-Z} = \frac{SS_{Z,-Z}df_{-Z}}{SS_{-Z}df_{Z,-Z}} \sim F_{df_{-Z},\, df_{Z,-Z}} \qquad 2.3.1$$

(Cook and Weisberg, 1982). Under the null hypothesis, $F_{-Z} = 1$. We expect that if there are blunders in Z, $F_{-Z} > 1$, and we are not interested if $F_{-Z} < 1$. Therefore, we use the one-sided F test. We have $df = (c - u)$, $df_{-Z} = (c - u - m)$, and $df_{Z,-Z} = m$; we still need expressions for SS_{-Z} and $SS_{Z,-Z}$ to form F.

Obviously, one could recompute the solution without the observations in Z, but there exist formulae for the values we want. Let

$$H = B(B^T B)^{-1} B^T.$$

H is referred to as the HAT matrix (Myers, 1985), since $Hf = \hat{f}$ (f being the vector of observations). Define H_Z as the diagonal submatrix of H corresponding to the observations in Z. If the observations in Z are adjacent in B, then H_Z is just a square submatrix along the diagonal. If the observations in Z are not adjacent, then the H_Z consists of the diagonal entries of H and the corresponding off-diagonal entries. For example, if we have three observations and Z represents the first and third of these, we would have

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

and

$$H_Z = \begin{bmatrix} h_{11} & h_{13} \\ h_{31} & h_{33} \end{bmatrix}$$

Finally, define m as the number of observations in Z, that is, the number of observations being removed and v_Z as the vector of residuals on the observations in Z (computed from the full solution). We can now write

$$SS_{Z,-Z} = v_Z^T(I - H_Z)^{-1}v_Z \tag{2.3.2}$$

$$SS_{-Z} = SS - SS_{Z,-Z} \ . \tag{2.3.3}$$

Armed now with the F test, we are ready to test for blunders in our data set. However, we still need H and $(I - H_Z)^{-1}$. H_Z will typically be small (we will not test for more than a few blunders at a time) so the inversion is not of concern. The real effort is in computing H_Z .

Even if we remove just one observation and test it, all of the required quantities are still defined, in particular $SS_{Z,-Z}$. Although this could reduce to a Student't t variable (noting that $F_{1,df} = t^2_{df}$), we will test it as an F variable. If we wish to remove a set of observations and test just one of them, say the ith one, we could use $SS_{i,-Z} = v_i^2 h_{ii}$ (equivalent to using just v_i in computing $SS_{Z,-Z}$, replacing $v_{j,\, j\neq i}$ with 0).

2.3.1 Using the Givens solution to compute test statistics

The major effort in computing the test statistic we desire is in computing $H = B(B^T B)^{-1} B^T$. Recalling that

$$H = B(B^T B)^{-1} B^T = Q_1 Q_1^T$$

we see that computing any given element of H involves only computing the norm of a vector of u elements.

2.3.2 Properties of the H matrix

H has the following interesting properties:

1. H is idempotent, i.e. $HH = H$
2. H is symmetric and positive definite
3. tr$H$ (the trace of H) = p
4. $h_{ii} \leq 1$; to the extent that $h_{ii} \rightarrow 1$, observation i is excessively influential, or "by itself" in the data set, possibly being the only observation which has much effect in predicting a certain parameter. If observation i is alone in predicting this parameter, then $h_{ii} = 1$.

From this last property, we can see a connection to the formula for $SS_{Z,-Z} = v_Z^T (I - H_Z)^{-1} v_Z$. If $h_{ii} = 1$ then $h_{ij,\,j} \neq i = h_{ji,\,j} \neq i = 0$ and thus $(I - H_Z)$ is singular. This does not indicate an error. Simplistically, we can say that if an influential observation is removed, there may be no data remaining to predict a certain parameter, and the system is no longer solvable.

It is well worth recalling at this point that this entire development of a method for detecting blunders in a set of observations has used the transformed design (B), observation (f) and residual (v) matrices, yet there has been no place where we have had to make any assumptions in this regard. As long as there is a one-to-one correspondence between untransformed and transformed observations (i.e., as long as W and L are diagonal), the transformation simply represents a scaling of the observations. There is no advantage to using the untransformed matrices; all of our statistics can be based on the transformed matrices.

2.4 Updating the solution

Identifying blunders helps us only if we can remove or replace them. To do this, we need a method for removing a row of data from B and inserting a row of data into B. Obviously, a new solution could be computed from scratch, but this is not necessary. It is faster to update the existing Q and R factors directly by exploiting their special properties. (The remainder of this section expands on the development given in Golub and Van Loan, 1983, pp. 442-443.)

2.4.1 Removing a row of data

Suppose we wish to update the Q and R factors of B, such that the ith row of B is effectively removed from the solution. Let

$$B = \begin{bmatrix} B_1 \\ b_i^T \\ B_2 \end{bmatrix}, \qquad 2.4.1$$

where b_i^T is the ith row of B and let q^T be the ith row of Q. Compute a set of rotation matrices

$$J = J_{m-1} J_{m-2} \cdots J_1 \tag{2.4.2}$$

such that

$$J^T q^T = \alpha e_1 \tag{2.4.3}$$

where $\alpha = \pm 1$. The product QJ will have the form

$$QJ = \begin{bmatrix} 0 & Q_1 \\ \alpha & 0 \\ 0 & Q_2 \end{bmatrix}. \tag{2.4.4}$$

(Note: Q_1 and Q_2 do not have the same meaning as they had in the develpment of the QR method.) If we construct J properly, operating on successive adjacent pairs of rows starting at the top, J^T will be a Hessenberg matrix, that is, $j_{ij} \neq 0$ only where i > j . Thus,

$$J^T R = \begin{bmatrix} v^T \\ \\ R_1 \end{bmatrix} \tag{2.4.5}$$

will also be Hessenberg. We now have

$$QR = QJJ^T R = \begin{bmatrix} 0 & Q_1 \\ \alpha & 0 \\ 0 & Q_2 \end{bmatrix} \begin{bmatrix} v^T \\ \\ R_1 \end{bmatrix} \tag{2.4.6}$$

$$\begin{bmatrix} B_1 \\ b_1^T \\ B_2 \end{bmatrix} = \begin{bmatrix} 0 & Q_1 \\ \alpha & 0 \\ 0 & Q_2 \end{bmatrix} \begin{bmatrix} v^T \\ \\ R_1 \end{bmatrix} \tag{2.4.7}$$

It can now be seen that $v^T = \alpha b_1^T$. We now have the desired result, the Q-R decomposition of B without the ith row, B_{i-1} :

$$B_{i-1} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R_1 \tag{2.4.8}$$

2.4.2 Replacing a row of data

Instead of completely removing data from the solution we may wish to replace it, in which case we keep the factors temporarily in the form of eq. 2.7.7, above. We may then do the following: substitute the new data for b_i^T and v^T (recalling that $b_i = \alpha v$); compute another set of rotation matrices

$$J = J_{m-1}J_{m-2} \dots J_1 \qquad 2.4.9$$

but this time such that $J\begin{bmatrix} v^T \\ R_1 \end{bmatrix}$ is upper triangular (as was R originally); compute QJ ; and we will have the Q-R decomposition of the new B.

Note that if we are simply replacing an observation of a control point with another observation of the same control point, there is no reason to update Q or R . The error in an observation is contained in f , which plays no part in computing Q or R. In this case, the simplest thing is to insert the new entry into f and re-form the product Qf (all at once this time, not sequentially as the original had been formed), and we have accomplished our goal. This method for updating Q and R in replacing an observation really applies only in the case where we measure a different control point, so that B actually changes.

2.5 The effects of Rank deficiency

Problems of column rank deficiency in B have come up at various times during the development of the method. This subject is important enough to warrant a review of its effects on the method we have outlined for blunder detection and on the possibility of solving for the parameters.

Because Q is an orthogonal matrix, we have assumed that Q is not singular. Even if we have no data, Q will be the identity matrix. H is formed from part of this orthogonal matrix, Q_1, but we have no similar guarantee that it will not be singular; nor do we know that any submatrix H_Z will not be singular. The existence of H assures us that we may compute $\hat{f}$, $\hat{l}$, and v, but obtaining F_{-Z} may be impossible.

In order to solve the system completely, to obtain the parameter vector Δ as well as the residuals, we need B to have full column rank. Unless this is the case, R_1 will be rank deficient, making the system impossible to solve directly for the full set of parameters.

There are exceptions to this. One is where there is complete information for a subset of the parameters, and no information for any other parameters. Supposing for convenience that these are the first p parameters, we can look at the upper triangular matrix $R_p \in \mathbf{R}^{p \times p}$, where R_p consists of the first p rows and columns of R . Using R_p we can solve for Δ, ignoring all other parameters. This is the approach used in the example program, and it is appropriate for sequential adjustments in general.

Another approach involves solving not for the parameters themselves but for linear combinations of the parameters. We will not pursue this any farther here. Most of the parameters in photogrammetry have physical meanings which would be lost by linearly combining them. It is preferable in such cases simply to obtain more data or modify the geometry of the problem.

2.6 The connection with robust estimation

Suppose that, during the course of Givens orthogonalization, we are at the point of eliminating b_{ji} from partially reduced coefficient matrix B_{ji} . In this case the two rows of interest are the ith and jth rows. Isolating these two rows, we have

$$\begin{bmatrix} 0 \dots 0 b_{ii} \dots b_{in} \\ 0 \dots 0 b_{ji} \dots b_{jn} \end{bmatrix}.$$

We may construct a rotation matrix which will serve to eliminate b_{ji} as

$$Q_{ji} = \begin{bmatrix} \dfrac{b_{ii}}{\sqrt{b_{ii}^2 + b_{ji}^2}} & \dfrac{-b_{ji}}{\sqrt{b_{ii}^2 + b_{ji}^2}} \\ \dfrac{-b_{ji}}{\sqrt{b_{ii}^2 + b_{ji}^2}} & \dfrac{b_{ii}}{\sqrt{b_{ii}^2 + b_{ji}^2}} \end{bmatrix} \qquad 2.6.1$$

(ignoring all but the ith and jth rows and columns). Defining

$$P_{ji} = \begin{bmatrix} 1 & \dfrac{b_{ji}}{b_{ii}} \\ \dfrac{-b_{ji}}{b_{ii}} & 1 \end{bmatrix}. \qquad 2.6.2$$

$$D_{ji}^{1/2} = \begin{bmatrix} \dfrac{b_{ii}}{\sqrt{b_{ii}^2 + b_{ji}^2}} & 0 \\ 0 & \dfrac{b_{ii}}{\sqrt{b_{ii}^2 + b_{ji}^2}} \end{bmatrix} \qquad 2.6.3$$

and

$$D_{ji} = \begin{bmatrix} \dfrac{b_{ii}}{b_{ii}^2 + b_{ji}^2} & 0 \\ 0 & \dfrac{b_{ii}}{b_{ii}^2 + b_{ji}^2} \end{bmatrix} \qquad 2.6.4$$

(noting that $D_{ji}^{1/2}D_{ji}^{1/2} = D_{ji}$) we have $Q_{ji} = P_{ji}D_{ji}^{1/2}$. Our goal is to form the product $Q_{ji}B = P_{ji}D_{ji}^{1/2}B_{ji}$. D simply serves to multiply the ith and jth rows of B by constants. Processing the ith and jth rows can now be done in two steps: multiplication by $D_{ji}^{1/2}$, then multiplication by P_{ji}

When B has been completely processed, the set of $D_{ji}^{1/2}$ matrices can be expressed as $D_{mn}^{1/2} \ldots D_{32}^{1/2} D_{31}^{1/2} D_{21}^{1/2}$; the order is unimportant, as all $D_{ji}^{1/2}$ are diagonal. The product is likewise a diagonal matrix. It may be advantageous to postpone the computation of the upper-triangular form R until it is needed: each row of B will then only be multiplied by a single factor; and the individual square roots need not be computed, just the roots of the elements of D. The result is that the number of arithmetic operations required is reduced somewhat. More importantly, however, we should note that $D^{1/2}$ is in the same position as L^T . This means that $D^{1/2}$ can serve the purpose of L^T; we can adjust D to change the weights on our observations. Many so-called robust estimators involve exactly this, a re-weighting of suspect observations. By these methods, the weights on observations depend not only on the *a priori* weight assigned to it, but also on the conformity of the observation to the rest of the data set (See Kubik, Merchant and Schenk, 1987). A solution by the method described here (using the sequential Givens algorithm) would thus lead naturally into robust estimation.

3. The projects at hand

3.1 Details of the present implementation

The Givens orthogonalization algorithm has been implemented in FORTRAN-77 and tested on a VAX computer operating under the VMS operating system. All the code has been written following strict language rules in order to be portable to other machines with FORTRAN-77. The core routines which comprise the actual adjustment portion call extra routines to obtain and modify data, control the iteration for nonlinear problems, and write the output to a file. These extra routines are specific to the application. Two sets of such routines have been written for this project, one set for the level net problem and the other set for the photogrammetric resection problem.

The core routines closely follow the development outlined herein. Wherever possible, redundant or meaningless computations are avoided. For example, some computations necessarily yield zeroes, so these computations are skipped and the zeroes assigned directly. Routines from the LINPACK linear algebra package (see Dongarra and others, 1978) are used extensively without modification. Because of this dependency on LINPACK, and in the interest of clarity, no advantage was taken of the natural sparsity available in a given system. If this were to be done, most of the routines would have to be rewritten.

Apart from the driver routine, the subroutines are all small, with typically fewer than fifty executable lines of code. All subroutines were designed to be as generic as possible. In this way the code is partitioned into modules which are easily developed, modified and reused.

Data input is from a file. In this aspect the program merely mimics a sequential solution, since all the data are available at once. However, the processing is done line by line, with the operator allowed to check and modify the solution after each line has been processed. Residuals for the processed observations are displayed, and the operator is allowed to ask for the F-statistic on any combination of them. If the statistic can be computed, the results are given; if there are insufficient data, the operator is told so. The program only provides the statistic and the degrees of freedom; the operator must perform the actual F-test himself. After performing the test, the operator is then given the choice of changing the data by modifying the observation or replacing the entire line of the B and f matrices (for the level net problem) or completely deleting the line (for both problems). Once the data set has been processed, the estimated overall variance of the observations, the estimated parameters and the residuals are written to an output file.

3.2 Comparison with the normal equations approach

Most photogrammetric and geodetic adjustment methods are based on the normal equations approach. The success of the present approach must be guaged partly by how well it performs as compared to a normal equations solution. The remainder of this section will look at the two methods based on accuracy, memory requirements and speed. (This discussion will ignore issues of sparsity in the design matrix B.)

3.2.1 Accuracy

A comparison of computational accuracy is not as important as it might seem. With a reasonably well-conditioned problem, the accuracies of the two methods are identical. In the ill-conditioned case, the Givens method should perform substantially better than the normal equations method, but this is really a moot point. In most cases, the photogrammetrist can detect ill-conditioning in the solution through extensive preanalysis. Furthermore, such ill-conditioning will be accompanied by high variances on the computed parameters, a problem serious enough to cause rejection of the solution and a revision of the geometry.

3.2.2 Memory requirements

If any statistical analysis of outliers (blunders) is being performed, memory requirements for a Givens solution and a normal equations solution will be similar. This is because blunder detection requires H, a $c \times c$ matrix. This is the largest matrix generated, and it is the same size regardless of method. In this case, savings in other areas will not be meaningful. If only an analysis of the variance structure of the parameters is required, then a $u \times u$ matrix, the inverse of the normal equations, is required regardless of the method. If no statistical analysis is called for, the Givens method holds a slight advantage with regard to memory requirements. This is because the normal equations are not formed explicitly, the Cholesky factor, R^t, being computed directly from the design matrix, B.

3.2.3 Speed

For speed, the Givens orthogonalization algorithm is a two-edged sword. Again, the advantage depends on whether blunder detection is required. In the simple case, where a simple solution is required, the normal equations method requires $\frac{cu^2}{2}$ floating point operations (FLOPs; one FLOP consists of one addition and one multiplication or division) to form the normal equations and $\frac{u^3}{6}$ FLOPs to generate the Cholesky factor. (Recall that c is the number of equations or observations, and u is the number of unkown parameters.) In this case, the Givens orthogonalization algorithm requires no FLOPs to generate the normal equations (not generated) and $2cu^2 - \frac{2}{3}u^3$ FLOPs to generate the Cholesky factor. This does not include updating Q. Once the Cholesky factor exists, in either case, solving requires $\frac{u^2}{2}$ FLOPs. We can see that the Givens orthogonalization algorithm requires $\frac{3}{2}cu^2 - \frac{5}{6}u^3$ extra FLOPs, or nearly four times the computer time that the normal equations method requires for the simple least-squares solution.

In the case where we need to detect blunders, the situation changes. The FLOP counts above and beyond the simple solution, above, are as follows. The normal equations method requires $\frac{2}{3}u^3$ FLOPs to invert the normal equations and $\frac{c^2u^2}{2}$ FLOPs to generate H, having already computed the normal equations. The Givens orthogonalization algorithm requires $2cu^2 + \frac{u^3}{2}$ FLOPs (again, ignoring sparsity) to generate Q_1 and $\frac{c^2u}{2}$ FLOPs to generate H, for a total of $2cu^2 + \frac{c^2u}{2} + \frac{u^3}{2}$ FLOPs. The normal equations method thus requires $\frac{c^2u^2}{2} - \frac{c^2u}{2} - 2cu^2 + \frac{1}{6}u^3$ more FLOPs than does the Givens orthogonalization algorithm to compute H. A rough computation shows that for $c > 4$, the normal equations method requires more FLOPs than the Givens orthogonalization algorithm, and for large problems, in which m greatly exceeds n, the normal equations method requires roughly $\frac{c^2u^2}{2}$ more FLOPs. Considering that H will be computed at intervals during the course of solution, in order to check for blunders periodically, this excess FLOP count negates the advantage in speed that the normal equations formerly enjoyed. Once H is in hand, the methods do not differ in operations required for blunder detection. From this point, computing the test statistic for a set Z of m observations requires roughly $\frac{c^2u}{2}$ floating point operations.

Finally, we will consider the case where we need to delete an observation. For the normal equations, H must be recomputed from scratch; for the Givens orthogonalization algorithm, Q is updated and H computed from Q. From the previous discussion it should be clear that the advantage of the Givens orthogonalization algorithm in computing H carries over into the deletion of observations.

Of course, any method for detecting blunders which makes use of H is a great improvement over the brute-force method of recomputing the solution from B and f.

Readability has been given priority, for the most part, over efficiency in the inner workings of the subroutines, so the operation counts given here are slightly lower than the actual operation counts while running the sample program.

3.3 Assumptions and deficiencies

The solution algorithm described here and implemented in the accompanying program makes the following critical assumptions:

- That reasonable initial approximations are available for all unknown parameters in the photogrammetric problem. "Reasonable" here means that quick convergence will not be prevented by poor initial approximations.

- That the error covariance matrix, Σ , is block diagonal; the software implementation assumes that $\Sigma = \sigma_0^2 I$.

- That no constraints are required.

- That the coefficients on l, the vector of observations, are all unity. In the notation of Mikhail (1976), there is no A matrix, and we are solving the problem by the adjustment of observations only.

As discussed above, the assumption of Σ diagonal is not critical; other forms of Σ will be simple and may be represented by a simple premultiplication of B and f by L^T where

$$\Sigma = LL^T.$$

Σ is not needed again until the end of processing, whereupon the process of removing it may be reversed and the necessary statistics computed. Thus, a minor modification of the existing program would add this feature.

The assumption of reasonable initial approximations is not altogether sound for a working phototriangulation system. Finding some of these parameters may be the very reason an adjustment is performed, and so requiring good *a priori* approximations may defeat the purpose. And the specific blunder detection routine in the present system is intended to serve as part of the first iteration in the solution of the problem as a whole and relies on these approximations for its own solution. Thus, bad approximations will not be corrected before they reach the solution of the whole system and so cause slow convergence, in addition to slowing up the blunder detection. But these problems, however serious, are essentially ignored here and left as a separate problem. The realistic assumption, that no good initial approximations are available, would require the decoupling of the blunder detection and solution routines and the installation of a separate routine for generating initial approximations, called from the routine that reads the data from the input file and generates the matrices. Thus, the present system might be thought of as treating a special case, an efficient means of reaching the solution when good approximations are available.

Bad approximations and blunder detection do not mix well at all. The blunder detection scheme is heavily dependent on the accuracy of the approximations, because these approximations set up the linear space in which we are trying to locate blunders. A significant departure in the approximations from the "true" model means that we are testing our data against a shifted model, and statistical tests may be rendered meaningless.

The introduction of constraints would pose some interesting problems. One method is to treat them as observations with zero variance. This approach requires manipulating Σ and W a great deal. Another method, that of Paige (1979) and Kourouklis and Paige (1981), handles a very general case, including both constraints and a coefficient matrix on the observations, and requires two orthogonalizations, of the design matrix and then of part of the orthogonal Q matrix. The latter is a proven method for solving least squares. Unlike the simple case with neither constraints nor coefficients on the observations, the deletion or replacement of individual observations is quite complex.

One issue which we have addressed only briefly has been that of sparsity. Photogrammetric problems often involve parameters that are spatially associated - for example, the coordinates of a camera exposure station and the coordinates of the object points shown on the corresponding image. Practically, this means that the resulting matrices may be very sparse. Because R and the Cholesky decomposition of B^TB are identical in structure, some of the techniques used for storing the normal equations efficiently (e.g., Snay, 1976 or Jennings, 1977) can be applied here. This structure is easily predicted (see, for example, Brown, 1974). It is much more difficult to predict the structure of Q. There is far more fill-in with Q, and the a storage scheme to take advantage of sparsity has to be quite complex. Since the present system is intended to demonstrate some of the advantages of the Givens orthogonalization algorithm, the problems of sparsity and core-storage limitations were ignored.

4. Examples

In this section we will review the results of applying the Givens method to two problems: a level net and a photogrammetric resection. The most important things to note are the correctness of the solutions and the sequential nature of the processing, in which the user interacts constantly by monitoring and making choices.

4.1 Level net example

The level net program is intended to be an example of the Givens orthogonalization algorithm solving a simple but realistic problem in surveying. The mathematical model is linear, relating the elevations of measured and known points on the Earth's surface. Each observation is an elevation difference. In this case, there is one known elevation (a benchmark, labeled M in Figure 1 on page 37, with an elevation of 1000.0 feet) and three unknown elevations (labeled A, B and C). Duplicate measurements between M and each of A, B and C account for the first six observations, and single measurements between the three unknown elevations are in the last three observations. The data are given in Table 1 on page 38. Intentional blunders were introduced into the constant vector of the fifth observation and into the constant vector and the coefficients for the ninth observation.

A portion of the output from running this example through the program are given in Figure 2 on page 39 showing the display of residuals and the computation of the F statistic. Figure 3 on page 40 shows the operator manipulating the data through modification (in which just the constant vector is corrected) and replacement (in which a new observation is introduced into the coefficient matrix and the constant vector). The success of the method is clear from the improvement in the residuals. Figure 4 on page 41 shows the resulting parameters (adjusted elevations of A, B and C).

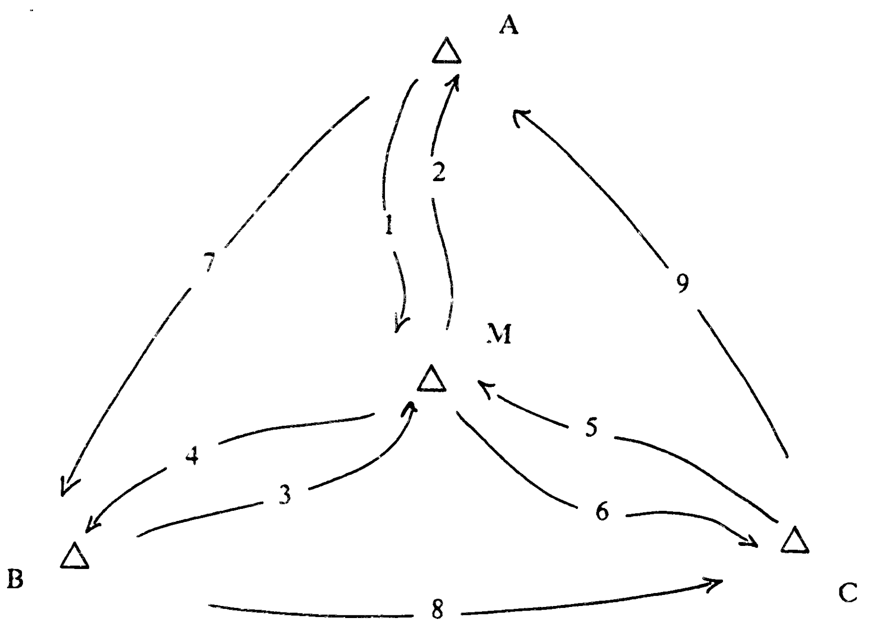

Figure 1. Geometry of level net example.

Table 1. Data for level net example.

Obs. 1:	Δ Elevation (A to M) =	-1099.0 m	
Obs. 2:	Δ Elevation (M to A) =	1101.0 m	
Obs. 3:	Δ Elevation (B to M) =	-1200.0 m	
Obs. 4:	Δ Elevation (M to B) =	-1199.0 m	
Obs. 5:	Δ Elevation (C to M) =	-930.0 m	(blunder)
Obs. 6:	Δ Elevation (M to C) =	902.0 m	
Obs. 7:	Δ Elevation (A to B) =	102.0 m	
Obs. 8:	Δ Elevation (B to C) =	-299.0 m	
Obs. 9:	Δ Elevation (C to A) =	160.0 m	(blunder)

```
Residuals after processing row          8
Row   1:  0.137D+01    Row   2:  0.633D+00    Row   3:  0.310D+01
Row   4: -0.410D+01    Row   5: -0.180D+02    Row   6: -0.100D+02
Row   7: -0.733D+00    Row   8: -0.793D+01
Select the set to check, separated and ended by <CR>:
5
F statistic =    342.798  ([df1 =    1] / [df2 =    4])
Residuals after processing row          8
Row   1:  0.137D+01    Row   2:  0.633D+00    Row   3:  0.310D+01
Row   4: -0.410D+01    Row   5: -0.180D+02    Row   6: -0.100D+02
Row   7: -0.733D+00    Row   8: -0.793D+01
Select the set to check, separated and ended by <CR>:
6
F statistic =      1.783  ([df1 =    1] / [df2 =    4])
Residuals after processing row          8
Row   1:  0.137D+01    Row   2:  0.633D+00    Row   3:  0.310D+01
Row   4: -0.410D+01    Row   5: -0.180D+02    Row   6: -0.100D+02
Row   7: -0.733D+00    Row   8: -0.793D+01
```

Figure 2. Interactive screen display for level net problem, part 1.

```
Residuals after processing row           9
Row   1: -0.292D+02    Row   2:  0.312D+02    Row   3:  0.429D+00
Row   4: -0.143D+01    Row   5:  0.108D+02    Row   6: -0.388D+02
Row   7: -0.286D+02    Row   8: -0.394D+02    Row   9: -0.890D+02
Select the set to check, separated and ended by <CR>:
9
F statistic =    128.319  ([df1 =    1] / [df2 =    5])
Which observation to delete, replace or modify?
(0 to end):
9
D[elete], R[eplace], or M[odify]?
r
Current data for row   9:
B(   9) =       1.000,       0.100,      -1.000,
f(   9) =    160.000
Please give replacement entry for constant vector:
200
Please give complete replacement row
for design matrix:
1      0      -1
Residuals after processing row           9
Row   1:  0.370D+01    Row   2: -0.170D+01    Row   3:  0.310D+01
Row   4: -0.410D+01    Row   5: -0.203D+02    Row   6: -0.770D+01
Row   7:  0.160D+01    Row   8: -0.560D+01    Row   9:  0.700D+01
Select the set to check, separated and ended by <CR>:
5
F statistic =    342.267  ([df1 =    1] / [df2 =    5])
Which observation to delete, replace or modify?
(0 to end):
5
D[elete], R[eplace], or M[odify]?
m
Current data for row   5:
B(   5) =       0.000,       0.000,      -1.000,
f(   5) =   -930.000
Please give replacement entry for constant vector:
-900
Residuals after processing row           9
Row   1:  0.700D+00    Row   2:  0.130D+01    Row   3:  0.100D+00
Row   4: -0.110D+01    Row   5:  0.700D+00    Row   6:  0.130D+01
Row   7:  0.160D+01    Row   8:  0.400D+00    Row   9:  0.100D+01
Select the set to check, separated and ended by <CR>:
5
F statistic =      0.407  ([df1 =    1] / [df2 =    5])
```

Figure 3. Interactive screen display for level net problem, part 2.

```
Results of Level Net Adjustment:
Estimated sigma(0)-squared = 0.552519D-03


Elevation   1 =  1099.700000
Elevation   2 =  1200.100000
Elevation   3 =   900.700000

Residuals:
v(  1) = 0.700000D+00
v(  2) = 0.130000D+01
v(  3) = 0.100000D+00
v(  4) = -.110000D+01
v(  5) = 0.700000D+00
v(  6) = 0.130000D+01
v(  7) = 0.160000D+01
v(  8) = 0.400000D+00
v(  9) = 0.100000D+01
```

Figure 4. Results from level net problem.

4.2 Photogrammetric resection example

The photogrammetric resection program is a simple example of the Givens orthogonalization algorithm handling a non-linear problem in photogrammetry. Figure 5 on page 43 shows the basic geometry of the problem, and Table 2 on page 44 gives the raw data for this run. There is a blunder of .300 m in the y image coordinate of the first observation, roughly twenty times the variance of the observations but difficult to detect manually.

Figure 6 on page 45 shows an attempt to check two observations after only eight observations have been processed during the first iteration. Since this implies checking the removed observations against a unique solution, one which cannot yield any residuals, the F statistic cannot be computed. In Figure 7 on page 46, we see that after processing just one more image point, or two more observations, the statistic is computable and is significant at the 0.01 level. Figure 8 on page 47, also from the first iteration, shows the use of multiple tests. In both tests, the F statistic is significant at the 0.01 level. In response to these tests, the first and second observations, associated with the first point, have been removed during the course of all iterations. In Figure 9 on page 48, performed during the third iteration, we see the result of testing the first and second observations, the same test as was shown in Figure 8 on page 47. This time the magnitude of the F statistic has nearly tripled. A look at Figure 10 on page 49 shows the reason for this: the iteration has imroved the estimated variance of the observations, and this increases the contrast between the removed observations and the remainder.

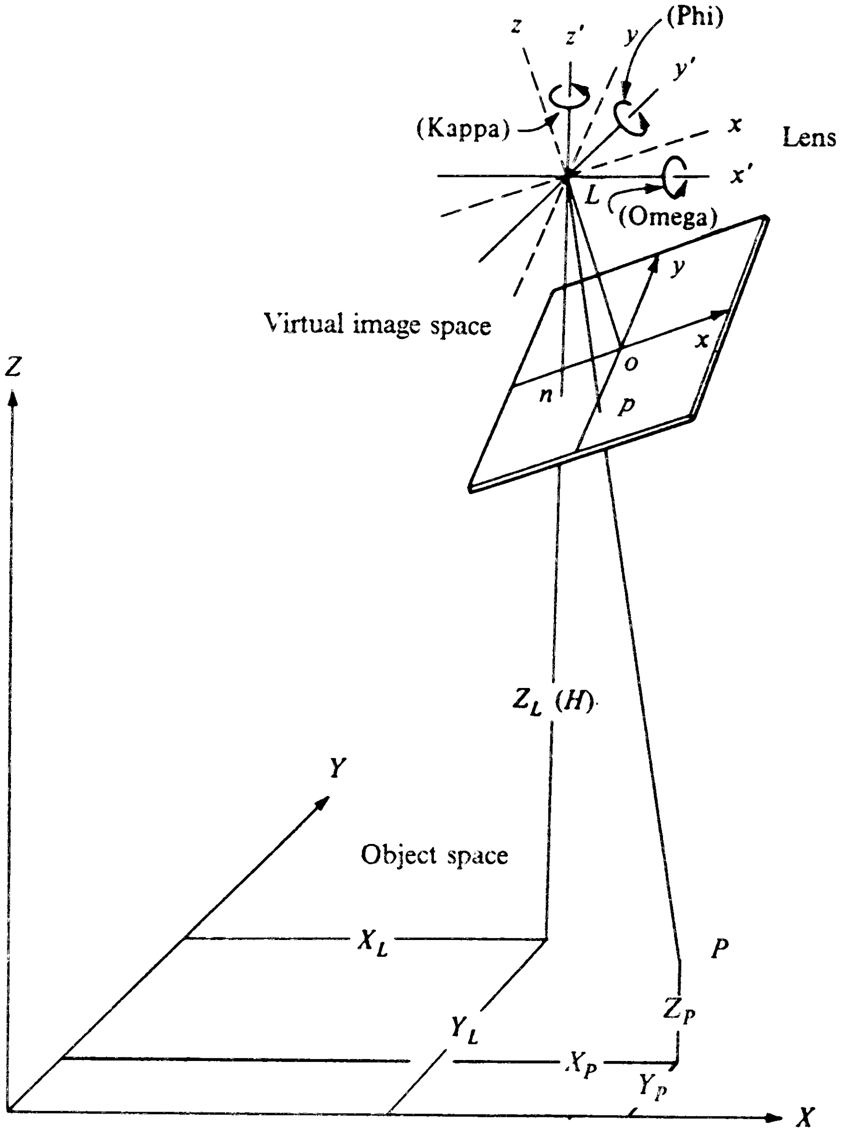

Figure 5. Orientation of a tilted image for photogrammetric resection example (adapted from Wolf, 1974, figure 13-7).

Table 2. Data for photogrammetric example.

Focal length = 100.0 mm

Image coordinates (mm)		Object coordinates (m)		
x	y	X	Y	Z
-110.881	-100.260	-10.000	-10.000	0.000
-12.123	-198.721	0.000	-10.000	5.000
93.314	-96.746	10.000	-10.000	0.000
-72.594	1.592	-10.000	0.000	5.000
-6.749	3.236	0.000	0.000	0.000
181.929	7.968	10.000	0.000	5.000
-106.775	103.292	-10.000	10.000	0.000
-5.065	67.478	0.000	10.000	5.000
90.067	99.806	10.000	10.000	0.000

```
Residuals after processing row          8
Row    1:  0.118D-01    Row    2: -0.335D-01    Row    3: -0.136D-01
Row    4: -0.287D-02    Row    5: -0.666D-02    Row    6:  0.139D-01
Row    7:  0.330D-01    Row    8:  0.381D-01
Select the set to check, separated and ended by <CR>:
1
2
F statistic is not computable for this set.
```

Figure 6. Interactive screen display for resection problem, part 1.

```
Residuals after processing row          10
Row   1:  0.486D-01    Row   2: -0.602D-01    Row   3: -0.275D-01
Row   4: -0.975D-02    Row   5: -0.470D-01    Row   6: -0.357D-02
Row   7:  0.238D-01    Row   8:  0.492D-02    Row   9:  0.375D-01
Row  10:  0.800D-01
Select the set to check, separated and ended by <CR>:
1
2
F statistic =     346.353  ([df1 =    2] / [df2 =    2])
Which observation to delete, replace or modify?
(0 to end):
0
```

Figure 7. Interactive screen display for resection problem, part 2.

```
Residuals after processing row          18
Row   1:  0.171D+00    Row   2: -0.284D-01    Row   3: -0.432D-01
Row   4: -0.654D-01    Row   5: -0.618D-01    Row   6:  0.524D-02
Row   7: -0.357D-01    Row   8:  0.309D-01    Row   9:  0.131D-02
Row  10:  0.421D-01    Row  11: -0.257D-01    Row  12:  0.504D-01
Row  13: -0.434D-02    Row  14:  0.367D-01    Row  15:  0.399D-01
Row  16:  0.382D-02    Row  17:  0.288D-01    Row  18: -0.490D-01
Select the set to check, separated and ended by <CR>:
1
2
F statistic =     34.752  ([df1 =    2] / [df2 =   10])
Select the set to check, separated and ended by <CR>:
1
2
9
10
F statistic =     14.268  ([df1 =    4] / [df2 =    8])
Which observation to delete, replace or modify?
(0 to end):
1
2
0
Residuals after processing row          16
Row   1: -0.218D-03    Row   2: -0.882D-02    Row   3: -0.418D-01
Row   4:  0.127D-01    Row   5: -0.826D-02    Row   6: -0.906D-03
Row   7:  0.364D-02    Row   8:  0.107D-01    Row   9:  0.253D-02
Row  10:  0.229D-01    Row  11: -0.176D-01    Row  12: -0.356D-01
Row  13:  0.301D-01    Row  14: -0.293D-02    Row  15:  0.365D-01
Row  16: -0.134D-01
```

Figure 8. Interactive screen display for resection problem, part 3.

```
Residuals after processing row          18
Row   1:  0.167D+00    Row   2: -0.137D-01    Row   3: -0.248D-01
Row   4: -0.703D-01    Row   5: -0.384D-01    Row   6: -0.277D-01
Row   7: -0.386D-01    Row   8:  0.482D-01    Row   9:  0.524D-02
Row  10:  0.494D-01    Row  11: -0.504D-01    Row  12:  0.271D-01
Row  13:  0.105D-01    Row  14:  0.539D-01    Row  15:  0.379D-01
Row  16: -0.953D-03    Row  17:  0.000D+00    Row  18:  0.000D+00
Select the set to check, separated and ended by <CR>:
1
2
F statistic =     90.329  ([df1 =     2] / [df2 =    10])
Which observation to delete, replace or modify?
(0 to end):
1
2
0
Residuals after processing row          16
Row   1:  0.885D-02    Row   2:  0.881D-03    Row   3: -0.168D-01
Row   4: -0.182D-01    Row   5: -0.860D-02    Row   6:  0.153D-01
Row   7:  0.213D-02    Row   8:  0.180D-01    Row   9:  0.818D-03
Row  10:  0.397D-02    Row  11: -0.158D-01    Row  12: -0.199D-01
Row  13:  0.248D-01    Row  14:  0.304D-03    Row  15:  0.000D+00
Row  16:  0.000D+00
```

Figure 9. Interactive screen display for resection problem, part 4.

```
Results of Iteration 1:
Estimated sigma(0)-squared = 0.671752D-03

Omega = 0.996121D+00
Phi   = -.100461D+01
Kappa = -.185216D-03
XL    = 0.499111D+00
YL    = -.498488D+00
ZL    = 0.999871D+01


Results of Iteration 2:
Estimated sigma(0)-squared = 0.260828D-03

Omega = 0.100073D+01
Phi   = -.100054D+01
Kappa = 0.232414D-04
XL    = 0.499817D+00
YL    = -.500012D+00
ZL    = 0.999994D+01


Results of Iteration 3:
Estimated sigma(0)-squared = 0.260758D-03

Omega = 0.100072D+01
Phi   = -.100054D+01
Kappa = 0.232167D-04
XL    = 0.499817D+00
YL    = -.500012D+00
ZL    = 0.999994D+01


Convergence achieved after 3 iterations.
```

```
Final residuals:
v(  1) = 0.885481D-02
v(  2) = 0.881199D-03
v(  3) = -.168067D-01
v(  4) = -.181629D-01
v(  5) = -.860450D-02
v(  6) = 0.152797D-01
v(  7) = 0.213354D-02
v(  8) = 0.180312D-01
v(  9) = 0.817513D-03
v( 10) = 0.397405D-02
v( 11) = -.158196D-01
v( 12) = -.198813D-01
v( 13) = 0.248350D-01
v( 14) = 0.304269D-03
v( 15) = 0.000000D+00
v( 16) = 0.000000D+00
```

Figure 10. Results from photogrammetric resection problem.

5. Conclusions and Future Directions

The present work shows that the Givens orthogonalization algorithm is a viable method for performing photogrammetric adjustments. We have seen that the method works well and is more efficient for blunder detection than the normal equations method. More fundamentally, we have seen the power of an orthogonalization method and its close association with important statistical methods. Having the orthogonal matrix available at all times enables much quicker access to important statistics related to our observations, which in turn enables us to detect errors in those observations. Furthermore, by choosing our orthogonalization method carefully, we have been able to process the entire problem sequentially. Using the Givens orthogonalization algorithm has permitted us to check the data almost as soon as it is available, and to replace suspicious data on the spot. A comparison of the present method with the normal equations method has shown that, while the normal equations method is more efficient in simply generating a solution, the Givens orthogonalization algorithm is even more efficient in detecting blunders.

Now that the Givens orthogonalization algorithm itself has been proven, the next step is to fully implement the method in more complex problems in photogrammetry and geodesy. Expanding the condition equations to handle the greater number and variety of unknowns is a straightforward proposition. The challenges are in handling sparsity, adding constraint equations to the condition equations, handling a populated weight matrix, handling a coefficient matrix on the unknowns and residuals, and modifying the matrices with deletions or new observations. These are all integral parts of any complete adjustment package, and must be treated thoroughly in order for the Givens method to gain acceptance.

None of these additional capabilities is intractable. The problem of sparsity is essentially a bookkeeping problem. Constraints and coefficients on the observations and residuals and a more fully populated weight matrix may all be handled in a similar manner, in the context of orthogonal decompositions, according to the method of Paige (1979) and Kourouklis and Paige (1981). Constraints and a more fully populated weight matrix may all be considered to be aspects of a single problem, coefficients on the observations and residuals. The possibility of updating an existing solution is highly dependant on the particular formulation; a QR solution is particularly easy to update. Withal, we still want to be able to compute statistics on the observations easily and from a sequential solution, just as we do with the blunder detection described here. The real challenge is to satisfy these demands and create a method which is sequential, efficiently stores sparse data, handles arbitrary coefficients on observations and residuals and on parameters, and lends itself to the rapid computation of observation statistics.

Bibliography

Baarda, W., 1960. *Precision, Accuracy and Reliability of Observations.* Delft: Geodetic Computing Centre, Delft University of Technology.

Baarda, W., 1968. *A Testing Procedure for Use in Geodetic Networks.* Delft: Netherlands Geodetic Commission.

Belsley, David F., Edwin Kuh and Roy E. Welsch, 1980. *Identifying Influential Data and Sources of Collinearity.* New York: John Wiley and Sons.

Blais, J. A. R., 1983. "Linear Least-squares computations using Givens transformations." *The Canadian Surveyor* 37(4):225-233.

Cook, R. Dennis and Sanford Weisberg, 1982. *Residuals and Influence in Regression.* New York: Chapman and Hall.

Dongarra, J., J. R. Bunch, C. B. Moler and G. W. Stewart, 1978. *LINPACK Users Guide.* Philadelphia: SIAM Publications.

Golub, Gene H., F. T. Luk and M. Pagano, 1979. "A sparse least-squares problem in photogrammetry." *Proceedings of Computer Science and Statistics: Twelfth Annual Conference on the Interface,* Waterloo, Canada.

Golub, Gene H. and Charles F. Van Loan, 1983. *Matrix Computations.* Baltimore: Johns Hopkins University Press.

Gruen, Armin. 1985. "Algorithmic aspects in on-line triangulation." *Photogrammetric Engineering and Remote Sensing* 51(4):419-436.

Jennings, A., 1977. *Matrix Computations for Engineers and Scientists.* New York: John Wiley and Sons.

Kok, Johan, 1984. "On data snooping and multiple outlier testing." *NOAA Technical Report* NOS NGS 30. Rockville: National Geodetic Information Center, NOS/NOAA.

Kourouklis, S. and C. C. Paige, 1981. "A constrained least squares approach to the general gauss-markov linear model." *Journal of the American Statistical Association* 76:620-625.

Kubik, Kurt, Dean Merchant and Toni Schenk, 1987. "Robust estimation in photogrammetry." *Photogrammetric Engineering and Remote Sensing* 53:167-169.

Lawson, C. L. and R. J. Hanson, 1974. *Solving Least Squares Problems.* Englewood Cliffs: Prentice-Hall.

Mikhail, Edward M., 1976. *Observations and Least Squares.* New York: University Press of America.

Myers, Raymond H., 1986. *Classical and Modern Regression with Applications.* Boston: Duxbury.

Paige, C. C., 1979. "Fast numerically stable computations for generalized linear least squares problems." *SIAM Journal of Numerical Analysis* 16:165-71.

Pope, Allen J., 1960. "The statistics of residuals and the detection of outliers." *NOAA Technical Report* NOS65 NGS1. Rockville: National Geodetic Information Center, NOS/NOAA.

Rao, C. Radhakrishna, 1973. *Linear Statistical Inference and its Applications, second edition.* New York: John Wiley and Sons.

Runge, Armin, 1987. "The use of Givens transformation [sic] in on-line phototriangulation." *Proceedings ISPRS Intercommission Conference on Fast Processing of Photogrammetric Data, Interlaken, Switzerland, June 2-4, 1987*, 179-192.

Snay, Richard, 1976. "Reducing the profile of sparse symmetric matrices." *NOAA Technical Memorandum* NOS NGS-4. Rockville: National Geodetic Information Center, NOS/NOAA.

Teskey, Theano, 1983. *Least-Squares Survey Computations Using Givens Transformations.* M.S. Thesis, Department of Civil Engineering, University of Calgary.

Wolf, Paul R, 1974. *Elements of Photogrammetry*. New York: McGraw-Hill.

Wolf, Paul R, 1983. *Elements of Photogrammetry, second edition.* New York: McGraw-Hill.

Appendix A. The condition equations for photogrammetric resection

In order to predict the ideal image coordinates of a point x_i, y_i (corrected for principal point offset) from the position of the camera, its orientation, its focal length and the object coordinates of the point, we need to know

$X_L =$ the x coordinate of the exposure station in the ground coordinate system,

$Y_L =$ the y coordinate of the exposure station in the ground coordinate system,

$Z_L =$ the z coordinate of the exposure station in the ground coordinate system,

$X_{P,i} =$ the x coordinate of object point i in the ground coordinate system,

$Y_{P,i} =$ the y coordinate of object point i in the ground coordinate system,

$Z_{P,i} =$ the z coordinate of object point i in theground coordinate system,

ω, ϕ, κ which give the orientation of the camera relative to object coordinate axes (Wolf, 1983, p. 237),

f = the focal length of the camera.

The collinearity equations model the coordinate transformation from object space coordinates to photograph coordinates. Given rotation matrix

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} = M(\omega, \phi, \kappa) \qquad A.1$$

$$M = \begin{bmatrix} -\cos\phi\cos\kappa & \sin\omega\sin\phi\cos\kappa + \cos\omega\sin\kappa & -\cos\omega\sin\phi\cos\kappa + \sin\omega\sin\kappa \\ \cos\phi\sin\kappa & -\sin\omega\sin\phi\sin\kappa + \cos\omega\cos\kappa & \cos\omega\sin\phi\sin\kappa + \sin\omega\cos\kappa \\ \sin\phi & -\sin\omega\cos\phi & \cos\omega\cos\phi \end{bmatrix} \qquad A.2$$

(Wolf, 1983, appendix B) we may write

$$x_i(X_L, Y_L, Z_L, X_{P,i}, Y_{P,i}, Z_{P,i}, \omega, \phi, \kappa, f)$$

$$y_i(X_L, Y_L, Z_L, X_{P,i}, Y_{P,i}, Z_{P,i}, \omega, \phi, \kappa, f)$$

as

$$x_i = -f\frac{(m_{11}(X_{P,i} - X_L) + m_{12}(Y_{P,i} - Y_L) + m_{13}(Z_{P,i} - Z_L))}{(m_{31}(X_{P,i} - X_L) + m_{32}(Y_{P,i} - Y_L) + m_{33}(Z_{P,i} - Z_L))} C_{X,i} \qquad A.3$$

$$y_i = -f\frac{(m_{21}(X_{P,i} - X_L) + m_{22}(Y_{P,i} - Y_L) + m_{23}(Z_{P,i} - Z_L))}{(m_{31}(X_{P,i} - X_L) + m_{32}(Y_{P,i} - Y_L) + m_{33}(Z_{P,i} - Z_L))} = C_{Y,i} \qquad A.4$$

(Wolf, 1983, appendix C).

Our full functional model gives x_i and y_i explicitly as

$$x_i = C_{X,i} \quad \rightarrow \quad f_x \qquad A.5$$

$$y_i = C_{Y,i} \quad \rightarrow \quad f_y \quad . \tag{A.6}$$

We now need to generate Taylor series approximations to the functions for x_i and y_i. These approximations will take the form

$$f_x = f_x^0 + \sum_{\text{all } \xi} \frac{\partial x_i}{\partial \xi} \Delta\xi \tag{A.7}$$

$$f_y = f_y^0 + \sum_{\text{all } \xi} \frac{\partial y_i}{\partial \xi} \Delta\xi \tag{A.8}$$

where

$$\xi \in \{\omega, \phi, \kappa, X_L, Y_L, Z_L\}$$

From Wolf (1983, appendix C) the partials of the collinearity equations without interior orientation parameters are as follows. Letting

$$\Delta X = (X_A - X_L)$$

$$\Delta Y = (Y_A - Y_L)$$

$$\Delta Z = (Z_A - Z_L)$$

$$q = m_{31}\Delta X + m_{32}\Delta Y + m_{33}\Delta Z$$

$$r = m_{11}\Delta X + m_{12}\Delta Y + m_{13}\Delta Z$$

$$s = m_{21}\Delta X + m_{22}\Delta Y + m_{23}\Delta Z$$

then

$$\frac{\partial x_i}{\partial \omega} = -\frac{x_i}{q}(-m_{33}\Delta Y + m_{32}\Delta Z) - \frac{f}{q}(-m_{13}\Delta Y + m_{12}\Delta Z) \qquad A.9$$

$$\frac{\partial x_i}{\partial \phi} = -\frac{x_i}{q}[\Delta X \cos\phi + \Delta Y(\sin\omega \sin\phi) + \Delta Z(-\sin\phi \cos\omega)]$$

$$-\frac{f}{q}[\Delta X(-\sin\phi \cos\kappa) - \Delta Y(\sin\omega \cos\phi \cos\kappa) - \Delta Z(-\cos\omega \cos\phi \cos\kappa)]$$

$$A.10$$

$$\frac{\partial x_i}{\partial \kappa} = -\frac{f \times s}{q} \qquad A.11$$

$$\frac{\partial x_i}{\partial X_L} = -\left[\frac{x_i}{q}(m_{31}) + \frac{f}{q}(m_{11})\right] \qquad A.12$$

$$\frac{\partial x_i}{\partial Y_L} = -\left[\frac{x_i}{q}(m_{32}) + \frac{f}{q}(m_{12})\right] \qquad A.13$$

$$\frac{\partial x_i}{\partial Z_L} = -\left[\frac{x_i}{q}(m_{33})\frac{f}{q}(m_{13})\right] \qquad A.14$$

$$\frac{\partial y_i}{\partial \omega} = -\frac{y_i}{q}(-m_{33}\Delta Y + m_{32}\Delta Z) - \frac{f}{q}(-m_{23}\Delta Y + m_{22}\Delta Z) \qquad A.15$$

$$\frac{\partial y_i}{\partial \phi} = -\frac{y_i}{q}[\Delta X \cos\phi + \Delta Y(\sin\omega \sin\phi) + \Delta Z(-\sin\phi \cos\omega)] \qquad A.16$$

$$-\frac{f}{q}[\Delta X(\sin\phi \sin\kappa) - \Delta Y(-\sin\omega \cos\phi \sin\kappa) - \Delta Z(\cos\omega \cos\phi \sin\kappa)] \qquad A.17$$

$$\frac{\partial y_i}{\partial \kappa} = \frac{f \times r}{q} \qquad A.18$$

$$\frac{\partial y_i}{\partial X_L} = -\left[\frac{y_i}{q}(m_{31}) + \frac{f}{q}(m_{21}) \right] \qquad A.19$$

$$\frac{\partial y_i}{\partial Y_L} = -\left[\frac{y_i}{q}(m_{32}) + \frac{f}{q}(m_{22}) \right] \qquad A.20$$

$$\frac{\partial y_i}{\partial X_L} = -\left[\frac{y_i}{q}(m_{33}) + \frac{f}{q}(m_{23}) \right]. \qquad A.21$$

We also need approximations to the functions at the estimated values of the parameters. Ignoring the dependence of L upon x_i and y_i, we may calculate the approximations to the funtions as

$$f_x^0 = C_{X,i} - x_p \qquad A.22$$

$$f_y^0 = C_{Y,i} - y_p \qquad A.23$$

Since these functions are now linear in { $\Delta\xi$ }, a matrix representation of the f_x and f_y may be written. Defining

$$B = \left[\frac{\partial x_i}{\partial \omega} \; \frac{\partial x_i}{\partial \varphi} \; \frac{\partial x_i}{\partial \kappa} \; \frac{\partial x_i}{\partial X_L} \; \frac{\partial x_i}{\partial Y_L} \; \frac{\partial x_i}{\partial Z_L} \; \frac{\partial y_i}{\partial \omega} \; \frac{\partial y_i}{\partial \phi} \; \frac{\partial y_i}{\partial \kappa} \; \frac{\partial y_i}{\partial X_L} \; \frac{\partial y_i}{\partial Y_L} \; \frac{\partial y_i}{\partial Z_L} \right] \qquad A.24$$

$$\Delta = \begin{bmatrix} \Delta x_p \\ \vdots \\ \Delta Z_{p,i} \end{bmatrix} \qquad A.25$$

$$v = \begin{bmatrix} v_{x,i} \\ \vdots \\ v_{y,i} \end{bmatrix} \qquad A.26$$

and

$$f = \begin{bmatrix} f_x - f_x^0 \\ f_y - f_y^0 \end{bmatrix} \qquad A.27$$

the two condition equations corresponding to the ith point, considering only nonzero entries, may be written as

$$v + B\Delta = f.$$

Appendix B. Fortran Program Listings

```
      program givens
C     ***  Some notational conventions:
C     ***    Z in the abstract represents the set of  ***
C     ***      observations which are being excluded  ***
C     ***      from the solution for the purpose of   ***
C     ***      blunder detection.                     ***
C     ***    Z in a variable name means "computed     ***
C     ***      from the solution which excludes the   ***
C     ***      observations in Z."                    ***
C     ***    ZZ in a variable name corresponds to     ***
C     ***      (Z,-Z) and means "computed for the     ***
C     ***      observations in Z, from the solution   ***
C     ***      which excludes the observations in Z." ***
C     ***    SSR in a variable name means "residual   ***
C     ***      sum of squares" as the term is used    ***
C     ***      by statisticians.                      ***
C     ***    df in a variable name means "degrees of  ***
C     ***      freedom", again in statistical usage.  ***

C     ******************************
C     ******************************
C     ***  DECLARATION STATMENTS  ***
C     ******************************
C     ******************************

C     ********************
C     ***  parameters  ***
C     ********************
      integer                    nobs
      parameter                  (nobs = 200)

      integer                    nunk
      parameter                  (nunk = 100)

C     *****************************
C     ***  regular declarations  ***
C     *****************************

C     usofar  logical(nunk)
C             Each entry tells whether the corresponding parameter in x
C             has any associated data yet in B.
C
C     slvabl  logical
C             (Translate: "solvable.")
C             Gives a necessary condition for whether the system up to
C             this point can be solved:  if the number of observations
C             is equal to or exceeds the number of unknowns (as given
C             by lstcol), and there are no gaps in usofar, then slvabl
C             is .true., otherwise .false..
C
C     lstcol  integer
C             Gives the column index of the last non-zero row in B.
C
C             Usofar, or usofar and lstcol together, tell what might be
C             solved for at any given moment during processing.  There
C             is no guarantee that, just because usofar says there is
C             information, a parameter can be solved for.  Usofar just
C             gives a necessary condition, not a sufficient one.
```

```
C
C     m       integer
C             The number of observations, modified during deletion.
C             Given by the user.
C
C     morig   integer
C             The number of observations, preserved for iteration
C
C     n       integer
C             The number of parameters, given by the user.
C
C     i, j, iq, jq, k        integer
C             Counters.
C
C     info    integer
C             Information being sent to LINPACK routines, indicating
C             choices.
C
C     step    integer
C             Tells what step of the processing we're on, increments
C             every time an entry of B is processed.
C
C     delndx  integer
C             Index of the observation being checked-temporary.
C     delnum  integer
C             Number of observations being checked.
C     delset  integer
C             Vector of indices or the observations being checked.
C
C     numrem  integer
C             Returned from MODIFY, the number of observations
C             remaining after the observation set has been modified.
C
C     B       double precision (nobs, nunk)
C             The design or coefficient matrix.
C
C     R       double precision (nobs, nunk)
C             The upper triangular matrix.
C
C     Q       double precision (nobs, nobs)
C             The orthogonal matrix.  Note: B = QR.
C
C     H       double precision (nobs, nobs)
C             The HAT matrix.  H = Q(1) * Q(1)-transpose, where
C             Q(1) consists of the first n rows of Q.
C             Note: f(est.) = H * f;  v = (I - H) * f.
C
C     HZ      double precison
C             The submatrix of H corresponding to the removed rows.
C
C     IHZ     double precision
C             (I - HZ)
C
C     IHZinv  double precision
C             (I - HZ)-inverse
C
C     bnew    double precision
C             A test matrix for recomputing B = QR.
C
C     v       double precision (nobs)
```

```
C               The residual vector.
C
C       vz      double precision (nobs)
C               Subset of v corresponding to the set of removed
C               residuals.
C
C       remain  integer
C               During the reduction of a given row, remain gives the
C               number of unreduced columns yet to be processed.
C
C       f       double precision (nobs)
C               The constant vector on input, processed during
C               orthogonalization.
C
C       forig   double precision (nobs)
C               The constant vector on input, preserved.
C
C       x       double precision (nunk)
C               The unknown parameters.  These are computed as
C               an afterthought, just to see if we got to
C               the right place with the adjustment.
C
C       c,s     double precision
C               Used in forming the Givens rotations.
C       zapped  double precision
C               The entry of the design matrix about to be zeroed-out.
C
C       pivot   double precision
C               The entry of the design matrix used to zero-out "zapped".
C               If zapped = A(i,j), then pivot = A(j,j).
C
C       SSR     double precision
C               Residual sum of squares.
C
C       df      double precision
C               degrees of freedom for the full adjustment up to
C               this observation.
C
C       SSRZ, SSRZZ, dfZ, dfZZ: see "notational conventions," above.
C
C       pivots  double precision (nobs)
C
C       work    double precision (nobs)
C               Dummies for called routines.
C
C       Fmult   double precision
C               The computed F statistic.

        logical             usofar(nunk), slvabl, looked

        integer             morig, m, n, remain, lstcol, status
        integer             i, j, iq, jq, k, info, step
        integer             delndx, delnum, delset(nobs), numrem
        integer             idumm1(nobs), idumm2(nobs), idumm3(nobs)

        double precision    B(nobs, nunk), Q(nobs, nobs), R(nobs, nunk)
        double precision    H(nobs, nobs), HZ(nobs, nobs)
        double precision    IHZ(nobs, nobs), IHZinv(nobs, nobs)
        double precision    bnew(nobs, nunk)
        double precision    f(nobs), forig(nobs), x(nunk)
```

```
      double precision  v(nobs), vz(nobs)
      double precision  dummy(2, nobs)
      double precision  c, s, pivot, zapped
      double precision  sum
      double precision  SSR, SSRZ, SSRZZ, df, dfZ, dfZZ
      double precision  pivots(nobs), work(nobs)
      double precision  Fmult

C     ******************************
C     ***  Subroutines called:   ***
C     ***  CHKROW                ***
C     ***  GETH                  ***
C     ***  GETV                  ***
C     ***  GETHVZ                ***
C     ***  MODIFY                ***
C     ***  GSSRZZ                ***
C     ***  GETOBS                ***
C     ***  ITERAT                ***
C     ***  DROTG                 ***
C     ***  DROT                  ***
C     ***  IDENT                 ***
C     ***  MAMULT                ***
C     ***  MZERO                 ***
C     ******************************

C     ******************************
C     ***  External functions    ***
C     ******************************

      double precision  ddot
```

```
C     ******************************
C     ******************************
C     ***  EXECUTABLE STATEMENTS  ***
C     ******************************
C     ******************************

      call getdat (nobs, nobs, morig, n, B, f)
C     ***  Preserve an original copy of m  ***
C     ***  in case we are iterating.       ***

   13 m = morig
      call ident (Q, nobs, m)

      call dcopy (m, f, 1, forig, 1)

C     ******************************************
C     ***  Initialize usofar, the array that  ***
C     ***  identifies those parameters for    ***
C     ***  which we have data.                ***
C     ******************************************

      do 12 i = 1, n
         usofar(i) = .false.
   12 continue

C     ******************************************
C     ***  Compute the Givens orthogonalization  ***
C     ******************************************

      i = 1
   20 continue
         looked = .false.

   22    if (.not. looked) then

C           ***  Copy row i of B into R.       ***
            do 30 j = 1, n
               R(i,j) = B(i,j)
   30       continue

C           ***  Check the current row to see  ***
C           ***  what parameters we have data  ***
C           ***  about yet.                    ***
            call chkrow (R(i, 1), nobs, i, n, usofar, lstcol, slvabl)
            do 40 j = 1, n
               if (i .gt. j) then
                  step = step + 1
                  pivot = R(j, j)
                  zapped = R(i, j)
                  call drotg( pivot, zapped, c, s )

C                 **************************************
C                 ***  Apply the Givens rotation to Q:  ***
C                 **************************************

                  call drot (i, Q(1, j), 1, Q(1, i), 1, c, s)

C                 **************************************
C                 ***  Apply the Givens rotation to B:   ***
```

```
C                   ***  Note: we know that R(i, j) = 0.0  ***
C                   ***  Also: to process a row, we don't  ***
C                   ***  have to start with column one.    ***
C                   ***  Remain = how many nozero columns  ***
C                   ***  remain in row i.                  ***
C                   ***  Again, we are using lstcol to     ***
C                   ***  indicate how far out we have to   ***
C                   ***  go in processing.                 ***
C                   ****************************************

                    remain = lstcol - j + 1
                    call drot(remain, R(j, j), nobs, R(i, j), nobs, c, s)
                    R(i, j) = 0.0
                    call drot(1, f(j), i, f(i), i, c, s)


C                   ****************************************
C                   ***  Write the matrices out to a file  ***
C                   ****************************************

  201               format ('1', 'Matrices after step ', i2,
     |                        ', row ', i2, ', column ', i2, ': Q, B, f, J')
                    do 203 iq = 1, m
  202                  format (' ', 20(f6.3, ' ') )
  203               continue
                    do 205 iq = 1, m
  204                  format (' ', 10f10.4, '          ', 10f10.4)
  205               continue
  206               format (' ', 2(f6.3, ' ') )
                 end if
C               *************************************
C               ***  Now we should have both      ***
C               ***  parts of this step completed.  ***
C               *************************************
   40           continue

            endif

C           ***  Get the HAT matrix.                  ***

            call geth (Q, nobs, i, lstcol, H)

C           ***  Obtain the raw residuals.            ***
            call getv(H, nobs, i, forig, v)

                do 212 iq = 1, i
  211              format (' ', 20(f6.3, ' ') )
  212           continue

                do 214 iq = 1, i
  213              format (' ', f10.3)
  214           continue

C           ***  Obtain the residual sum of           ***
C           ***  squares, SSR                         ***
            SSR = ddot(i, v, 1, v, 1)

C           ***  Degrees of freedom for full          ***
C           ***  solution:                            ***
            df = i - lstcol
```

```
C         ***************************************
C         ***  Now: show the results to the      ***
C         ***  operator and let him choose what  ***
C         ***  to check, delete or replace.      ***
C         ***  he chooses a set to check, we     ***
C         ***  come back here.                   ***
C         ***************************************

C         ***  Display the residuals             ***
          write (*,*) 'Residuals after processing row ', i
          do 45 k = 1, i, 3
             if (k .eq. i) then
                write (*,251) k, v(k)
  251           format (' Row ', i3, ': ', D10.3)
             else if (k .eq. (i - 1) ) then
                write (*,252) k, v(k), k+1, v(k+1)
  252           format (' Row ', i3, ': ', D10.3,
     |                '    Row ', i3, ': ', D10.3)
             else
                write (*,253) k, v(k), k+1, v(k+1), k+2, v(k+2)
  253           format (' Row ', i3, ': ', D10.3,
     |                '    Row ', i3, ': ', D10.3,
     |                '    Row ', i3, ': ', D10.3)
             endif
   45     continue

C         ***************************************
C         ***  Request the set of residuals to   ***
C         ***  check.  Variable meanings:        ***
C         ***    delnum is the number of         ***
C         ***       observations to check        ***
C         ***    delset is an integer array of   ***
C         ***       the indices of the obser-    ***
C         ***       vations to check             ***
C         ***    delndx is used to index         ***
C         ***       delset                       ***
C         ***************************************

          write(*,*)
     |         'Select the set to check, separated and ended by <CR>:'
          delnum = 0
          delndx = 0
          looked = .false.
   46     read (*,254,err=48) delset(delndx + 1)
  254     format (i2)
             if ( (delset(delndx + 1) .eq. 0) .or.
     |            ( (delndx + 1) .ge. i) ) goto 48
             delndx = delndx + 1
          goto 46
   48     continue
          if (delndx .eq. 0) goto 50
          delnum = delndx
          looked = .true.


C         ***************************************
C         ***  Process this set of residuals to  ***
C         ***  look for outliers (blunders).     ***
C         ***************************************
```

```
            if (delnum .gt. 0) then
C              *** Quick! get Hz and and vz.       ***
               call gethvz (h, nobs, v, i, hz, vz, delnum, delset)
            endif

C           *** Obtain the SSRZZ, the residual     ***
C           *** sum of squares for the set of      ***
C           *** observations in Z computed from    ***
C           *** the adjustment which excludes      ***
C           *** the observations in Z.             ***
            call gssrzz (HZ, nobs, delnum, IHZ, IHZinv,
     |                   vz, pivots, work, SSRZZ, status)
            if (status) then

C              *** Obtain SSRZ, the residual sum of  ***
C              *** squares for the solution which    ***
C              *** excludes the observations in Z.   ***
               SSRZ = SSR - SSRZZ

C              *** Compute dfZ = df(-Z)              ***
C              *** and dfZZ = df(Z,-Z)               ***
               dfZZ = delnum
               dfZ = df - dfZZ

            endif

C           *** And we can compute F-statistic:   ***
C           *** check status first!               ***
            if ( ( .not. status) .or.
     |         (SSRZ .lt. ( (SSRZZ * dfZ)/(1.0d+10 * dfZZ) ) ) .or.
     |         (SSRZ .lt. 1.0d-10) ) then
               write (*,*) 'F statistic is not computable for this set.'
               write (*,*)
            else
               Fmult = ( SSRZZ * dfZ ) / ( SSRZ * dfZZ )

               if (Fmult .gt. 1000.0) then
                  write (*,*) 'F statistic is very large (F > 1000)'
               else
                  write (*,262) Fmult, int(dfZZ), int(dfZ)
               end if

 262           format (' F statistic = ', F10.3,
     |                 '   ([dfl = ', I4, '] / [df2 = ', I4, '])' )

C              *** call MODIFY to modify the         ***
C              *** solution with modifications,      ***
C              *** deletions and replacements to     ***
C              *** the data.                         ***

               call modify (Q, nobs, m, B, R, nobs, n, f, forig, H,
     |                      lstcol, i, numrem, iduml, idumm2, idumm3)

            endif

C           *** If we checked some of the          ***
C           *** residuals, whether or not we       ***
C           *** modified the solution, have a      ***
C           *** look again.                        ***
            if (looked) goto 22
```

```
   50 i = i + 1
      if (i .le. m) goto 20

      call getv(H, nobs, i, forig, v)
      call mamult (Q, m, m, nobs,
     |             R, m, n, nobs,
     |             bnew, nobs, -1, 0)

      do 60 i = 1, n
         x(i) = f(i)
   60 continue
      call dtrsl(R, nobs, n, x, 01, info)
C     ***  ITERAT must:                      ***
C     ***    update parameters               ***
C     ***    assess convergence              ***
C     ***    stop or return                  ***
C     ***    write the results to unit 2     ***
C     ***    close unit 2, which getdat      ***
C     ***      opened                        ***
      call iterat (x, m, n, ssr, df, B, nobs, f, nobs, v)
C     ***  If ITERAT sends us back here,     ***
C     ***  we return to the point right      ***
C     ***  after we got our data.            ***
      goto 13

      end
      subroutine chkrow (newrow, inc, row, n, usofar, lstcol, slvabl)
C     ***  Chkrow checks the incoming row of data, newrow, to see    ***
C     ***  where the zeroes are, updates usofar so that the latter   ***
C     ***  tells what columns have had data in them yet, and         ***
C     ***  updates lstcol so that it gives the index of the last     ***
C     ***  row which has had data so far.                            ***
C
C     On entry:
C
C     newrow  double precision (n x inc)
C             The new row of data.
C
C     inc     integer
C             The increment between entries of newrow.
C
C     row     integer
C             Gives the index of the row being processed; used in
C             determining slvabl, below.
C
C     n       integer
C             The number of entries in newrow; also, the size of usofar
C
C     usofar  logical (n)
C             (Translate: "used so far.")
C             A vector giving the indices of those rows which have
C             had data already.  Updated.
C
C     lstcol  integer
C             (Translate: "last column.")
C             The index of the last column to have had data already.
C             Updated.
C
C     slvabl  logical
C             (Translate: "solvable.")
C             Gives a necessary condition for whether the system up to
C             this point can be solved:  if the number of observations
C             is equal to or exceeds the number of unknowns (as given
C             by lstcol), and there are no gaps in usofar, then slvabl
C             is .true., otherwise .false..
C

C     **********************
C     **********************
C     ***  Declarations  ***
C     **********************
C     **********************

C     **********************
C     ***  Arguments in  ***
C     **********************

      integer           inc, n, lstcol, row

      logical           usofar (n), slvabl

      double precision  newrow (n*inc)

C     **********************
```

```
C     *** Local vbls.    ***
C     *********************

      integer           i, j, maxcol, nrwndx
C     *** "nrwndx" translates as "new row index,"  ***
C     ***  the indexing variable for newrow.       ***

C     *****************************
C     *****************************
C     ***  Executable statements  ***
C     *****************************
C     *****************************

      maxcol = 0
C     ***  Update usofar   ***
      do 10 i = 1, n
         nrwndx = 1 + ( (i - 1) * inc )
         if (newrow (nrwndx) .ne. 0.0) then
            maxcol = i
            usofar (i) = .true.
         end if

   10 continue

C     ***  Update lstcol   ***
      lstcol = max (lstcol, maxcol)

C     ***  Update svlabl   ***
      slvabl = .false.
      if (lstcol .le. row) then
         do 20 i = 1, lstcol
            slvabl = slvabl .or. usofar (i)
   20    continue
      end if

      return
      end
      subroutine GETH (Q, ldq, m, n, H)
C     ***  This subroutine computes HAT matrix, H, given Q:     HAT
C     ***        H = Q(1) Q(1)-transpose                        HAT
C     ***  where Q(1) consists of the first n columns of Q,     HAT
C     ***  which is an m x m orthogonal matrix.                 HAT

C     ********************************
C     ********************************
C     ***  Declaration statements  ***
C     ********************************
C     ********************************

C     ******************
C     ***  Arguments  ***
C     ******************

      integer          ldq, m, n

      double precision  Q (ldq, n)
C     ***  Note: we only use the  ***
C     ***  first n columns of Q.  ***
      double precision  H (ldq, m)

C     ************************
C     ***  Local Variables  ***
C     ************************

      integer i, j

C     ************************
C     ***  Functions       ***
C     ************************

C     ***  The only external function  ***
C     ***  used is DDOT, from BLAS.    ***

      double precision ddot

C     ********************************
C     ********************************
C     ***  Executable statements   ***
C     ********************************
C     ********************************

      do 10 i = 1, m
         do 9 j = 1, m
            H(i, j) = ddot (n, Q(i, 1), ldq, Q(j, 1), ldq)
    9    continue
   10 continue

      return
      end
      subroutine gethvz (h, ldh, v, n, hz, vz, hzsize, hzset)
c     ***************************************************
c     ***  GETHVZ computes HZ and vZ, given matrix H,  ***
c     ***  vector v, and index vector HZset.  GETHVZ   ***
c     ***  simply extracts the appropriate entries     ***
c     ***  from H and v to create HZ and vZ.           ***
c     ***  This subroutine is designed for the Givens  ***
c     ***  least-squares package.  HZ and vZ are       ***
c     ***  subsets of H and v corresponding to the     ***
c     ***  removed observations specified in set Z.    ***
c     ***  Except as described, declarations carry     ***
c     ***  the same meanings as in program GIVENS.     ***
c     ***************************************************

c      ********************************
c      ********************************
c      ***  Declaration statements  ***
c      ********************************
c      ********************************

c      ********************************
c      ***  Arguments in            ***
c      ********************************

c      ***  hzsize gives the number ***
c      ***  of indices contained in ***
c      ***  hzset.                  ***
      integer           ldh, n, hzsize
      integer           hzset(hzsize)
      double precision  h(ldh, n)
      double precision  v(n)

c      ********************************
c      ***  Arguments out           ***
c      ********************************

      double precision  hz(ldh, hzsize)
      double precision  vz(hzsize)

c      ********************************
c      ***  Local variables         ***
c      ********************************

      integer           i, j

c      ********************************
c      ********************************
c      ***  Executable statements   ***
c      ********************************
c      ********************************

      do 9 i = 1, hzsize
         vz(i) = v(hzset(i))
         do 8 j = 1, hzsize
            hz(i, j) = h(hzset(i), hzset(j))
    8    continue
    9 continue

      return

      end
      subroutine GETV (H, ldq, m, f, v)

C     ***********************************************************QR
C     ***  GETV obtains the vector of residuals, v, from          QR
C     ***  the QR solution to a least-squares problem.            QR
C     ***  The H matrix is assumed to exist already.              QR
C     ***  The formula used is                                    QR
C     ***        v = ( I - H ) f                                  QR
C     ***  where                                                  QR
C     ***        H = Q(1) * Q(1)-transpose                        QR
C     ***        H = B * (B-transpose * B)-inverse * B-transpose  QR
C     ***  That portion of v past index m will not be affected.  QR
C     ***********************************************************QR

C     *********************
C     *********************
C     ***  Declarations ***
C     *********************
C     *********************

C     *********************
C     ***  Arguments in ***
C     *********************

      integer            ldq, m

      double precision  H (ldq, m)
      double precision  f (m)

C     *********************
C     ***  Arguments out ***
C     *********************

      double precision  v (m)

C     *********************
C     ***  Local Vbls.   ***
C     *********************

      integer i, j

      double precision  sum

C     ******************************
C     ******************************
C     ***  Executable statements ***
C     ******************************
C     ******************************


      do 10 i = 1, m
         sum = 0.0
         do 9 j = 1, m
            if (j .eq. i) then
               sum = sum + ( 1.0 - H (i, j) ) * f (j)
            else
               sum = sum + ( - H (i, j) * f (j) )
            end if
    9    continue
```

```
        v (i) = sum
10   continue

     return
     end
```

```
       subroutine gssrzz (HZ, ldhz, n, IHZ, IHZinv,
      |                   vz, pivots, work, SSRZZ, status)
c      *****************************************************************
c      ***  GSSRZZ:                                                  ***
c      ***  Get Sum of Squares-Residual, set Z, computed without Z  ***
c      *****************************************************************

c      ****************************************************
c      ***  Computes the residual sum of squares for the  ***
c      ***  removed observations.  Matrix HZ is the sub-  ***
c      ***  matrix of H corresponding to the removed      ***
c      ***  observations; vz is the subvector of v cor-   ***
c      ***  responding to the removed observations.  The  ***
c      ***  formula is:                                   ***
c      ***                                                ***
c      ***  SSRZZ := vz-transpose * (I - HZ)-inv * vz     ***
c      ***                                                ***
c      ***  This routine calls LINPACK routines           ***
c      ***    DSICO to factor HZ and determine its rank;  ***
c      ***    DSIDI to compute HZ-inverse                 ***
c      ***  and                                           ***
c      ***    QDRTIC to compute the quadratic form above. ***
c      ****************************************************

c      *******************************
c      *******************************
c      ***  Declaration statements  ***
c      *******************************
c      *******************************

       integer            nobs
       parameter          (nobs = 20)

c      *******************************
c      ***  Arguments in           ***
c      *******************************
c       subroutine gssrzz (HZ, ldhz, n, IHZ, IHZinv, vz, n, SSRZZ)

c      ***  status tells the calling routine whether  ***
c      ***  GSSRZZ has succeeded.  1=success,         ***
c      ***  0 = failure.                              ***
       integer            ldhz, n

c      ***  pivots and work are dummy vectors required by LINPACK  ***
       double precision  HZ(ldhz, n), vz(n), pivots(n), work(n)

c      *******************************
c      ***  Arguments out          ***
c      *******************************

       integer            status

c      ***  IHZ contains matrix (I - HZ)        ***
c      ***  IHZinv contains the inverse of IHZ  ***
       double precision  IHZ(ldhz, n), IHZinv(ldhz, n)
       double precision  SSRZZ

c      *******************************
c      ***  Local variables        ***
```

```
C     ******************************

      integer           i, j, choice

      double precision  rcond

C     ***  det and inert are        ***
C     ***  determinant and inertia  ***
C     ***  required by DSIDI.       ***
      double precision  det, inert

C     ******************************
C     ******************************
C     ***  Executable statements   ***
C     ******************************
C     ******************************

C     ******************************
C     ***  build IHZ                ***
C     ******************************

      do 10 i = 1, n
         do 9 j = 1, n
            if (i .eq. j) then
               IHZ(i, j) = 1.0 - HZ(i, j)
            else
               IHZ(i, j) = - HZ(i, j)
            endif
    9    continue
   10 continue

      call mtxcpy (IHZ, ldhz, IHZinv, ldhz, n, n)

C     ******************************
C     ***  invert IHZ -> IHZinv    ***
C     ******************************

      status = 1
      call DSICO (IHZinv, ldhz, n, pivots, rcond, work)
      if (real(1.0 + rcond) .eq. 1.0) then
C        ***  In this case, the condition number rcond  ***
C        ***  is getting close to zero, and HZ is       ***
C        ***  nearly singular.  The real() is not nec-  ***
C        ***  essary, but it makes things conservative. ***
         status = 0
         return
      endif

      choice = 1
      call DSIDI (IHZinv, ldhz, n, pivots, det, inert, work, choice)

C     ******************************
C     ***  copy the upper triangle ***
C     ***  into the lower -        ***
C     ***  payment for using the   ***
C     ***  LINPACK routines for    ***
C     ***  symmetric matrices.     ***
C     ******************************

      do 20 i = 2, n
```

```
          do 19 j = 1, i
             IHZinv(i, j) = IHZinv(j, i)
   19     continue
   20 continue

C     ********************************
C     ***  compute quadratic form  ***
C     ********************************

      call qdrtc (IHZinv, ldhz, vz, n, SSRZZ)

      return
      end
      subroutine modify (Q, ldq, m, B, R, ldr, n, f, forig, H,
     |                    lstcol, numobs, numrem,
     |                    delvec, repvec, modvec )

C     ***  MODIFY handles all on-line modification of an existing  ***
C     ***  Q-R solution by deleting observations, replacing         ***
C     ***  observations with completely new observations and        ***
C     ***  modifying observations by replacing just the constant    ***
C     ***  term.  The words "delete," "replace" and "modify" will   ***
C     ***  be used to mean these three processes.                   ***
C     ***  Note:  this routine is written to work correctly under   ***
C     ***  FORTRAN-77, not FORTRAN-66.  This is because there are   ***
C     ***  DO-loops that may never be executed; if this were        ***
C     ***  compiled under FORTRAN-66, this could not be assumed.    ***


C     ********************************
C     ********************************
C     ***  Declaration Statements  ***
C     ********************************
C     ********************************

C     ********************************
C     ***  Arguments               ***
C     ********************************

      integer            ldq, m, ldr, n, lstcol, numrem, numobs

C     ***  delvec gives the indices of these observations      ***
C     ***  repvec gives the indices of these observations      ***
C     ***  modvec gives the indices of these observations      ***
      integer            delvec(1), repvec(1), modvec(1)

      double precision   Q(ldq, m), B(ldr, n), R(ldr, n),
     |                   f(ldr), forig(ldr), H(ldq, m)

C     ********************************
C     ***  Local variables         ***
C     ********************************

C     ***  delnum gives the number of observations to delete   ***
C     ***  repnum gives the number of observations to replace  ***
C     ***  modnum gives the number of observations to modify   ***

      logical            delete, replce, modif

      integer            delnum, repnum, modnum, i, j, choice, obs

      character*1        dummyc, ok

C     ********************************
C     ********************************
C     ***  Executable  Statements  ***
C     ********************************
C     ********************************

C     ***  Initialization  ***
      delnum = 0
      modnum = 0
      repnum = 0
```

```
C     ********************************************************************
C     ***  First: find out what's to be done with which observations  ***
C     ********************************************************************

   10 delete = .false.
      replce = .false.
      modif  = .false.
      write (*,*) 'Which observation to delete, replace or modify?'
      write (*,*) '(0 to end):'
      read (*,*) obs
      if (obs .gt. numobs ) then
         write (*.*) 'There are not that many observations!'
         goto 10
      endif
      if (obs .le. 0) goto 20
   12 write (*,*) 'D[elete], R[eplace], or M[odify]?'
      read (*,101) dummyc
  101 format (A)
      if      ( (dummyc .eq. 'D') .or. (dummyc .eq. 'd') ) then
         delete = .true.
      else if ( (dummyc .eq. 'R') .or. (dummyc .eq. 'r') ) then
         replce = .true.
      else if ( (dummyc .eq. 'M') .or. (dummyc .eq. 'm') ) then
         modif = .true.
      else
         write (*,*) 'Invalid choice.'
         goto 12
      end if
      write (*,*) 'OK? (Y or N)'
      read (*,101) ok
      if ( .not. ( (ok .eq. 'Y') .or. (ok .eq. 'y') ) ) then
         write (*,*) 'Try again:'
         goto 10
      end if

      if (delete) then
         delnum = delnum + 1
         delvec(delnum) = obs
      else if (replce) then
         repnum = repnum + 1
         repvec(repnum) = obs
      else if (modif) then
         modnum = modnum + 1
         modvec(modnum) = obs
      endif

      goto 10

C     ********************************************************************
C     ***  Now take care of deleting, then modifying, then replacing  ***
C     ********************************************************************

C     *****************
C     ***  Deleting  ***
C     *****************

   20 continue

      do 35 i = 1, delnum
         call remove (Q, ldq, numobs, R, ldr, n, f, delvec(i) )
```

```
            call squish (Q, ldq, numobs, m, B, R, ldr, n,
     |                   f, forig, delvec(i), H, lstcol)
            do 30 j = i + 1, delnum
C              ***  realign the entries of delvec  ***
               if ( delvec(j) .gt. delvec(i) ) then
                  delvec(j) = delvec(j) - 1
               endif
   30       continue

            do 32 j = 1, modnum
               if ( modvec(j) .gt. delvec(i) ) then
                  modvec(j) = modvec(j) - 1
               endif
   32       continue

            do 34 j = 1, repnum
               if ( repvec(j) .gt. delvec(i) ) then
                  repvec(j) = repvec(j) - 1
               endif
   34       continue

            numobs = numobs - 1
            m = m - 1

   35 continue

C     ******************
C     ***  Modifying  ***
C     ******************

C     ***  We need to re-create Q-transpose * f from scratch  ***

      do 50 i = 1, modnum
C        ***  Get the constant for this observation.  Since    ***
C        ***  we are just replacing the observed quantity,     ***
C        ***  not the row of the design matrix (hence          ***
C        ***  modifying and not replacing), a quick multi-     ***
C        ***  plication takes care of things.  Nothing         ***
C        ***  elegant here, just re-create Q-transpose * f     ***
C        ***  from scratch.  (Choice = 1 for just constant.)  ***

         choice = 1
         call getobs (choice, modvec(i), forig(modvec(i)),
     |                B(modvec(i), 1), ldr, n)

         call mamult(Q, numobs, numobs, ldq,
     |               forig, numobs, 1, ldr,
     |               f, ldr, 0, 0 )

   50 continue

      do 60 i = 1, repnum

C        ***  Choice = 2 for constant and design matrix.  ***
         choice = 2
         call remove (Q, ldq, numobs, R, ldr, n, f, repvec(i) )
         call getobs (choice, repvec(i), forig(repvec(i)),
     |                B(repvec(i), 1), ldr, n)
         call newobs (Q, ldq, numobs, B, R, ldr, n,
     |                forig, f, ldr, repvec(i) )
   60 continue

      return
      end
      subroutine qdrtc (A, lda, x, n, SS)

C     ************************************************
C     ***  Computes the quadratic form for vector x and  ***
C     ***  matrix A:                                     ***
C     ***         SS := v-transpose * A * v              ***
C     ***  Since the quadratic form make sense only for  ***
C     ***  A symmetric (in a statistical context) and    ***
C     ***  in order to conform to the LINPACK,           ***
C     ***  only the upper triangle of A is used here.    ***
C     ************************************************

C     ********************************
C     ********************************
C     ***  Declaration statements  ***
C     ********************************
C     ********************************

C     ********************************
C     ***  Arguments in            ***
C     ********************************

      integer          lda, n

      double precision  A(lda, n), x(n)

C     ********************************
C     ***  Arguments out           ***
C     ********************************

      double precision  SS

C     ********************************
C     ***  Local variables         ***
C     ********************************

      integer          k, l

      double precision  Aentry

C     ********************************
C     ********************************
C     ***  Executable statements   ***
C     ********************************
C     ********************************

      SS = 0.0

      do 10 k = 1, n
         do 9 l = 1, n
            if (l .gt. k) then
               Aentry = A(k, l)
            else
               Aentry = A(l, k)
            endif
            SS = SS + x(k) * Aentry * x(l)
    9    continue
   10 continue

      return

      end
      subroutine remove (Q, ldq, m, R, ldr, n, const, row)

C     ***  This routine serves to remove the i-th row from a Q-R  ***
C     ***  solution, represented by Q and R here.  The following  ***
C     ***  steps are used:
C     ***    1.

C     ********************************
C     ********************************
C     ***  Declaration Statements  ***
C     ********************************
C     ********************************

C     ********************************
C     ***  Arguments             ***
C     ********************************

      integer           ldq, m, ldr, n, row

      double precision  Q(ldq, m), R(ldr, n), const(ldr)

C     ********************************
C     ***  Local variables       ***
C     ********************************

C     *************************************************************
C     ***  These may look like stupid variable names, but they  ***
C     ***  make sense.  zappee is the entry that's about to be  ***
C     ***  zeroed out by the Givens transformation; zapper is   ***
C     ***  the entry on the diagonal straight above zappee--    ***
C     ***  i.e. zappee = R(i,j) ==> zapper = R(j,j).  "ndxzpe"  ***
C     ***  simply means "iNDeX of ZaPpeE," and likewise "ndxzpr"***
C     ***  means "iNDeX of ZaPpeR."  remain gives the number    ***
C     ***  of entries remaining in the pertinent portion of R.  ***
C     *************************************************************

      integer           i, ndxzpe, ndxzpr, remain

      double precision  zappee, zapper, c, s

C     ********************************
C     ********************************
C     ***  Executable Statements  ***
C     ********************************
C     ********************************

      do 40 i = m, 2, -1

C        *************************************************************
C        ***  Determine the Givens transformation required to zap  ***
C        ***  q(row, i) against q(row, (i-1))                      ***
C        *************************************************************

         ndxzpr = (i-1)
         ndxzpe = i
         zapper = Q(row, ndxzpr)
         zappee = Q(row, ndxzpe)
         call drotg(zapper, zappee, c, s)
```

```
C         ***********************************************************
C         ***  Apply the current Givens transformation to Q       ***
C         ***********************************************************

          call drot (m, Q(1, ndxzpr), 1, Q(1, ndxzpe), 1, c, s)
C         ***********************************************************
C         ***  Apply the current Givens transformation to R       ***
C         ***  Variable "remain" tells how far it is from the     ***
C         ***  starting point to the end of the rows of R.        ***
C         ***********************************************************

          remain = n - ndxzpr + 1
          if (remain .gt. 0) then
             call drot(remain, R(ndxzpr, ndxzpr), ldr,
     |                 R(ndxzpe, ndxzpr), ldr, c, s)
          end if

C         ***********************************************************
C         ***  Apply the current Givens transformation to const    ***
C         ***********************************************************

          call drot(1, const(ndxzpr), ldr, const(ndxzpe), ldr, c, s)
   40 continue

C     ***********************************************************
C     ***  Clean up: zero the entries we know have to be zero,  ***
C     ***  make Q(row, 1) = 1.0, and if Q(row, 1) was computed  ***
C     ***  to be < 0, change the sign of the first row of R.    ***
C     ***********************************************************

C     ***  Take care of the first row of R:  ***
      if ( Q(row, 1) .lt. 0.0) then
         do 50 i = 1, n
            R(1, i) = - R(1, i)
   50    continue
      end if

C     ***  Zero out the appropriate entries of Q  ***
      do 60 i = 2, m
         Q(row, i) = 0.0
   60 continue
      do 70 i = 1, m
         if ( i .ne. row ) then
            Q(i, 1) = 0.0
         end if
   70 continue

C     ***  Set Q(row, 1) = 1.0  ***
      Q(row, 1) = 1.0

      return
      end
      subroutine squish (Q, ldq, numobs, m, B, R, ldr, n,
     |                   f, forig, row, H, lstcol)

c     ***************************************************
c     ***  SQUISH compresses the various matrices used in a  ***
c     ***  Givens orthogonalization for least squares.  For  ***
c     ***  an explanation of what's being squished out and   ***
c     ***  why, consult T. D. Johnson's M.S. thesis,         ***
c     ***  "The sequential Givens Method for Adjustment      ***
c     ***  Computations in Photogrammetry", Virginia Poly-   ***
c     ***  technic Institute and State University, 1988.     ***
c     ***************************************************

c     ********************************
c     ********************************
c     ***  Declaration Statements  ***
c     ********************************
c     ********************************

c     ***********************************************
c     ***  Variable names are similar to those used in  ***
c     ***  program GIVENS.                              ***
c     ***********************************************

c     ********************************
c     ***  Arguments               ***
c     ********************************

      integer           ldq, numobs, m, ldr, n, row, lstcol

      double precision  Q(ldq, numobs), B(ldr, n), R(ldr, n)
      double precision  f(1), forig(1), H(ldq, numobs)

c     ********************************
c     ***  Local variables         ***
c     ********************************

      integer           i, j

c     ********************************
c     ********************************
c     ***  Executable  Statements  ***
c     ********************************
c     ********************************

c     ********************************
c     ***  Take care of Q          ***
c     ********************************

      do 12 i = 1, row - 1
         do 10 j = 1, numobs - 1
            Q(i, j) = Q(i, j + 1)
   10    continue
   12 continue

      do 16 i = row, numobs - 1
         do 14 j = 1, numobs - 1
            Q (i, j) = Q(i + 1, j + 1)
   14    continue
   16 continue
```

```
         do 18 i = 1, numobs - 1
            Q(numobs, i) = 0.0
            Q(i, numobs) = 0.0
      18 continue
         q(numobs, numobs) = 1.0

C        ********************************
C        ***  Take care of B         ***
C        ********************************

         do 22 i = row, m - 1
            do 20 j = 1, lstcol
               B(i, j) = B(i + 1, j)
      20    continue
      22 continue
         do 24 j = 1, lstcol
            B(m, j) = 0.0
      24 continue

C        ********************************
C        ***  Take care of R         ***
C        ********************************

         do 32 i = 1, lstcol
            do 30 j = i, lstcol
               R(i, j) = R(i + 1, j)
      30    continue
      32 continue
         do 34 i = 2, lstcol + 1
            j = i - 1
            R(i, j) = 0.0
      34 continue

C        ********************************
C        ***  Take care of f         ***
C        ********************************

         do 40 i = 1, m - 1
            f(i) = f(i + 1)
      40 continue
         f(m) = 0.0
         do 42 i = row, m - 1
            forig(i) = forig(i + 1)
      42 continue
         forig(m) = 0.0

C        ********************************
C        ***  Take care of H         ***
C        ********************************

         call geth (Q, ldq, numobs, lstcol, H)

         return
         end
      subroutine getdat (ldb, ldf, m, n, B, f)

C     ***  Subroutine GETDAT read the data for the level   ***
C     ***  net adjustment problem.  It also opens unit 1   ***
C     ***  for input and unit 2 for output.                ***

C     *****************************
C     *****************************
C     ***  DECLARATION STATMENTS  ***
C     *****************************
C     *****************************

C     ****************************
C     ***  Arguments In        ***
C     ****************************

      integer           ldb, ldf

C     ****************************
C     ***  Arguments Out       ***
C     ****************************

      integer           m, n

      double precision  B(ldb, 1)
      double precision  f(ldf)

C     ****************************
C     ***  Local variables     ***
C     ****************************

      integer           i, j

C     *****************************
C     *****************************
C     ***  EXECUTABLE STATEMENTS  ***
C     *****************************
C     *****************************

C     ***********************************
C     ***  Open files; read parameters  ***
C     ***********************************

      open (unit=1,
     &      file = 'level.dat',
     &      status = 'old')

      read (1, 100) m, n
  100 format (2i4)

      open (unit=2,
     &      file = 'level.res',
     &      status = 'new')

C     ***************************
C     ***  Read the data file:  ***
C     ***************************
      do 11 i = 1, m
         read (1, 101) (B(i, j), j = 1, n), f(i)
  101    format (11f10.2)
```

```
11 continue
   end
```

```
      subroutine getobs (choice, row, fentry, Brow, ldb, n)
C     ***  GETOBS gets a new observation, to replace an old one.  ***
C     ******************************
C     ******************************
C     ***  Declaration Statements  ***
C     ******************************
C     ******************************

C     ******************************
C     ***  Arguments             ***
C     ******************************

      integer          choice, ldb, n, row

      double precision fentry, Brow(ldb, n)

C     ******************************
C     ***  Local variables       ***
C     ******************************

      integer          i

C     ******************************
C     ******************************
C     ***  Executable  Statements  ***
C     ******************************
C     ******************************

      write (*,100) row
      write (*,101) row, (Brow(1, i), i = 1, n)
      write (*,102) row, fentry

  100 format (' Current data for row', I4, ':')
  101 format (' B(', I4, ') = ', 10000(F10.3, ', ') )
  102 format (' f(', I4, ') = ', F10.3)

      write (*,*) 'Please give replacement entry for constant vector:'
      read (*,*) fentry

      if (choice .eq. 2) then
         write (*,*) 'Please give complete replacement row'
         write (*,*) 'for design matrix:'
         read (*,*) (Brow(1, i), i = 1, n)
      endif

      return
      end
      subroutine iterat (elev, m, n, ssr, df,
     |                   B, ldb, f, ldf, v)

C     ***  This routine controls the iteration for the  ***
C     ***  level net application of the Givens          ***
C     ***  adjustment package.  It writes the results   ***
C     ***  to the file level.dat, closes units 1 & 2,   ***
C     ***  ans stops.  ***

C     ********************************
C     ********************************
C     ***  Declaration statements  ***
C     ********************************
C     ********************************

C     ********************************
C     ***  Arguments In            ***
C     ********************************

      integer           m, n, df, ldb, ldf
      double precision  elev(1), ssr, v(1)

C     ********************************
C     ***  Arguments Out           ***
C     ********************************

      double precision  B(ldb, 1), f(1)

C     ********************************
C     ***  Local Variables         ***
C     ********************************

      integer           i

      double precision  var0

C     ********************************
C     ********************************
C     ***  Executable  statements  ***
C     ********************************
C     ********************************

      var0 = ssr / df
      write (2, 101)
  101 format (// , ' Results of Level Net Adjustment:')
      write (2,102) var0
  102 format (' Estimated sigma(0)-squared = ', D12.6,//)

      write (2,103) ( (i, elev(i)), i = 1, n)
  103 format (' Elevation ', i3, ' = ', F12.6)

      write (2,108)
  108 format (/, ' Residuals:')
      write (2,107) ((i,v(i)), i = 1, m)
  107 format (' v(', I3, ') = ', D12.6)
      close (unit = 1)
      close (unit = 2)
      stop

      return

      end
      subroutine newobs (Q, ldq, m, B, R, ldr, n,
     |                        forig, f, ldf, row )
C     ****************************************************************
C     ***  This subroutine integrates a new observation into an  ***
C     ***  existing Q-R decomposition.  This assumes that Q and  ***
C     ***  R, as well as F, need to be updated, and that REMOVE  ***
C     ***  has just been called.                                 ***
C     ****************************************************************

C     ********************************
C     ********************************
C     ***  Declaration Statements  ***
C     ********************************
C     ********************************

C     ********************************
C     ***  Arguments               ***
C     ********************************

      integer             ldq, m, n, ldr, ldf, row

      double precision  Q(ldq, m), B(ldr, n), R(ldr, n)
      double precision  forig(ldf), f(ldf)

C     ********************************
C     ***  Local variables         ***
C     ********************************

      integer             j, ndxzpr, ndxzpe, remain

      double precision  zapper, zappee, c, s

C     ********************************
C     ********************************
C     ***  Executable Statements   ***
C     ********************************
C     ********************************

      do 10 j = 1, n
         R(1,j) = B(row, j)
   10 continue

      f(1) = forig(row)

c      do 20 j = n, 1, -1
      do 20 j = 1, n
         ndxzpr = j
         ndxzpe = j + 1

C        ****************************************
C        ***  Compute c, s for Givens rotation ***
C        ****************************************

         zapper = R(ndxzpr, j)
         zappee = R(ndxzpe, j)
         call drotg( zapper, zappee, c, s )

C        ****************************************
C        ***  Apply the Givens rotation to Q:   ***
C        ***  Note that in updating Q and R,    ***
```

```
C        ***  some of the niceties, like       ***
C        ***  keeping track of the size of     ***
C        ***  non-trivial portions of Q and R,  ***
C        ***  are being ignored.  The calling   ***
C        ***  routine can take care of some of  ***
C        ***  this in specifying m & n.         ***
C        ***************************************

         call drot (m, Q(1, ndxzpr), 1, Q(1, ndxzpe), 1, c, s)

C        ***************************************
C        ***  Apply the Givens rotation to B:   ***
C        ***  Note: we know that R(j+1, j) = 0  ***
C        ***  Also: to process a row, we don't  ***
C        ***  have to start with column one.    ***
C        ***  Remain = how many nozero columns  ***
C        ***  remain in the row.                ***
C        ***************************************
c         zapper = R(ndxzpr, j)
c         zappee = R(ndxzpe, j)

         remain = n - j + 1
         call drot(remain, R(ndxzpr, j), ldr, R(ndxzpe, j), ldr, c, s)
         R(ndxzpe, j) = 0.0
         call drot(1, f(ndxzpr), n, f(ndxzpe), n, c, s)

   20 continue

      return
      end
      subroutine Brows ( row, B, ldb, f, ldf )
C     ******************************
C     ******************************
C     ***  Declaration statements  ***
C     ******************************
C     ******************************

C     ******************************
C     ***  Arguments In          ***
C     ******************************

      integer            row, ldb, ldf

C     ******************************
C     ***  Arguments Out         ***
C     ******************************

      double precision   B(ldb, 1), f(ldf)

C     ******************************
C     ***  Parameters            ***
C     ******************************

      integer            maxpho
      parameter          (maxpho = 10)
      integer            maxpts
      parameter          (maxpts = 25)
      double precision   rho
      parameter          (rho = 1.7453292519943296E-02)

C     *************************************************
C     ***  These are in common block PHOTOS, shared   ***
C     ***  among all the photogrammetric application  ***
C     ***  subroutines called by GIVENS               ***
C     *************************************************

      integer            numpho, numpts
      double precision   xpp, ypp
      double precision   XL(maxpho), YL(maxpho), ZL(maxpho)
      double precision   XP(maxpts), YP(maxpts), ZP(maxpts)
      double precision   omega(maxpho), phi(maxpho), kappa(maxpho)
      double precision   xi(maxpts, maxpho), yi(maxpts, maxpho)
      double precision   rotmtx(3,3,maxpho)
      double precision   foclen

      common / PHOTOS /
     |      numpho, numpts,
     |      xpp, ypp,
     |      XL, YL, ZL,
     |      XP, YP, ZP,
     |      omega, phi, kappa,
     |      xi, yi,
     |      rotmtx

      COMMON /BLK6/ foclen
C     ******************************
C     ***  Local Variables       ***
C     ******************************
```

```
      integer             Bndx, photo, point, oddrow
      integer             Bcoll, Bcol6, Ccoll, Ccol3
      integer             j, Bdmndx, Cdmndx
      double precision    Bdummy (2,6)
      double precision    Cdummy (2,3)
      double precision    fdummy (2)

C     ******************************
C     ******************************
C     ***  Executable  statements  ***
C     ******************************
C     ******************************

C     ***  Use oddrow to line up with the  ***
C     ***  natural order of the entries.   ***
      oddrow = ( ( (row - 1) / 2) * 2) + 1
      point = mod( ( ( (oddrow + 1) / 2) - 1 ), numpts) + 1
      photo = ( ( ( ( oddrow + 1) / 2) - point ) / numpts ) + 1
      Bcoll = ( (photo - 1) * 6) + 1
      Bcol6 = Bcoll + 5
      Ccoll = (numpho * 6) + ( (point - 1) * 3) + 1
      Ccol3 = Ccoll + 2

      call collin (xi(point, photo), yi(point, photo),
     |             XP(point), YP(point), ZP(point),
     |             omega, phi, kappa,
     |             XL(photo), YL(photo), ZL(photo),
     |             Bdummy, Cdummy, fdummy)

      do 10 j = Bcoll, Bcol6
         Bdmndx = j - Bcoll + 1
         B(oddrow, j) = Bdummy(1, Bdmndx)
         B( (oddrow + 1), j) = Bdummy(2, Bdmndx)
   10 continue

      do 15 j = Ccoll, Ccol3
         Cdmndx = j - Ccoll + 1
         B(oddrow, j) = Cdummy(1, Cdmndx)
         B( (oddrow + 1), j) = Cdummy(2, Cdmndx)
   15 continue

      f(oddrow) = fdummy(1)
      f(oddrow + 1) = fdummy(2)

      return
      end
      subroutine iterat (delta, m, n, ssr, df,
     |                   B, ldb, f, ldf, v)

C     ***  This routine controls the iteration for the  ***
C     ***  photogrammetric resection application of     ***
C     ***  Givens adjustment package.  It assesses the  ***
C     ***  convergence, updates parameters, writes the  ***
C     ***  results to file photo.dat, stops or returns  ***
C     ***  as appropriate, and closes units 1 & 2.      ***

C     *******************************
C     *******************************
C     ***  Declaration statements  ***
C     *******************************
C     *******************************

C     *******************************
C     ***  Parameters             ***
C     *******************************

      integer           numpar
      parameter         (numpar = 6)

      integer           maxpho
      parameter         (maxpho = 10)
      integer           maxpts
      parameter         (maxpts = 25)
      double precision  rho
      parameter         (rho = 1.7453292519943296E-02)

C     *******************************
C     ***  Arguments In           ***
C     *******************************

      integer           m, n, ldb, ldf
      double precision  delta(numpar), ssr, v(1), df

C     *******************************
C     ***  Arguments Out          ***
C     *******************************

      double precision  B(ldb, 1), f(1)

C     ***********************************************
C     ***  These are in common block PHOTOS, shared   ***
C     ***  among all the photogrammetric application  ***
C     ***  subroutines  called by GIVENS              ***
C     ***********************************************

      integer           numpho, numpts
      double precision  xpp, ypp
      double precision  XL(maxpho), YL(maxpho), ZL(maxpho)
      double precision  XP(maxpts), YP(maxpts), ZP(maxpts)
      double precision  omega(maxpho), phi(maxpho), kappa(maxpho)
      double precision  xi(maxpts, maxpho), yi(maxpts, maxpho)
      double precision  rotmtx(3,3,maxpho)
      double precision  foclen

      common / PHOTOS /
     |      numpho, numpts,
     |      xpp, ypp,
     |      XL, YL, ZL,
     |      XP, YP, ZP,
     |      omega, phi, kappa,
     |      xi, yi,
     |      rotmtx

      common / BLK6 / foclen

C     ******************************
C     ***  Local Variables       ***
C     ******************************

      integer           row, i

      logical           frstim
      data              frstim / .true. /
      integer           itcnt
      data              itcnt / 0 /

      double precision  maxdel, var0
      double precision  om, ph, ka

C     ******************************
C     ******************************
C     ***  Executable  statements  ***
C     ******************************
C     ******************************

C     ***  First:  assess the convergence  ***
      itcnt = itcnt + 1
      var0 = ssr / df
      maxdel = 0.0d0
      omega(1) = omega(1) + delta(1)
      om = omega(1) / rho
      phi(1) = phi(1) + delta(2)
      ph = phi(1) / rho
      kappa(1) = kappa(1) + delta(3)
      ka = kappa(1)
      XL(1) = XL(1) + delta(4)
      YL(1) = YL(1) + delta(5)
      ZL(1) = ZL(1) + delta(6)
      maxdel = max( abs(delta(1)),
     |              abs(delta(2)),
     |              abs(delta(3)),
     |              abs(delta(4)),
     |              abs(delta(5)),
     |              abs(delta(6)) )

      write (2,101) itcnt
  101 format (// , ' Results of Iteration ', I1, ':')
      write (2,102) var0
  102 format (' Estimated sigma(0)-squared = ', D12.6)
      write (2,103) om, ph, ka, XL(1), YL(1), ZL(1)
  103 format (/, ' Omega = ', D12.6,
     |        /, ' Phi   = ', D12.6,
     |        /, ' Kappa = ', D12.6,
     |        /, ' XL    = ', D12.6,
     |        /, ' YL    = ', D12.6,
     |        /, ' ZL    = ', D12.6 )
```

```
      if (maxdel .le. 1.0d-6) then
         write (2,104) itcnt
 104     format (//, ' Convergence achieved after ', I1, ' iterations.')
         write (2,108)
 108     format (/, ' Final residuals:')
         write (2,107) ((i,v(i)), i = 1, m)
 107     format (' v(', I3, ') = ', D12.6)
         close (unit = 1)
         close (unit = 2)
         stop
      else if ( itcnt .ge. 5 ) then
         write (2,105)
 105     format
     |     (//, ' Iteration limit (five) reached -- no convergence:')
         write (2,106) delta(1), delta(2), delta(3),
     |                 delta(4), delta(6), delta(7)
 106     format (/' , Delta-Omega = ', D12.6,
     |           /' , Delta-Phi   = ', D12.6,
     |           /' , Delta-Kappa = ', D12.6,
     |           /' , Delta-XL    = ', D12.6,
     |           /' , Delta-YL    = ', D12.6,
     |           /' , Delta-ZL    = ', D12.6 )
         write (2,108)
         write (2,107) ((i,v(i)), i = 1, m)
         close (unit = 1)
         close (unit = 2)
         stop
      else
C        ***  Carry on with the processing -- get  ***
C        ***  new design and constant matrices.    ***
         do 20 row = 1, (m-1), 2
            call Brows(row, B, ldb, f, ldf)
C           ***  Other variables are contained in  ***
C           ***  common block PHOTOS.              ***
 20      continue
      endif

      return
      end
      subroutine getdat (ldb, ldf, m, n, B, f)
C     ******************************
C     ******************************
C     ***  DECLARATION STATMENTS  ***
C     ******************************
C     ******************************

C     ******************************
C     ***  Arguments In          ***
C     ******************************

      integer            ldb, ldf

C     ******************************
C     ***  Arguments Out         ***
C     ******************************

      integer            m, n

      double precision   B(ldb, 1)
      double precision   f(ldf)

C     ********************************
C     ***  Parameters              ***
C     ********************************

      integer            maxpho
      parameter          (maxpho = 10)
      integer            maxpts
      parameter          (maxpts = 25)
      double precision   rho
      parameter          (rho = 1.7453292519943296E-02)

C     ***********************************************
C     ***  These are in common block PHOTOS, shared    ***
C     ***  among all the photogrammetric application   ***
C     ***  subroutines  called by GIVENS               ***
C     ***********************************************

      integer            numpho, numpts
      double precision   xpp, ypp
      double precision   XL(maxpho), YL(maxpho), ZL(maxpho)
      double precision   XP(maxpts), YP(maxpts), ZP(maxpts)
      double precision   omega(maxpho), phi(maxpho), kappa(maxpho)
      double precision   xi(maxpts, maxpho), yi(maxpts, maxpho)
      double precision   rotmtx(3,3,maxpho)
      double precision   foclen

      common / PHOTOS /
     |       numpho, numpts,
     |       xpp, ypp,
     |       XL, YL, ZL,
     |       XP, YP, ZP,
     |       omega, phi, kappa,
     |       xi, yi,
     |       rotmtx

      common / BLK6 / foclen
```

```
C     ***********************************************
C     ***  Local Variables                        ***
C     ***********************************************

      integer           i, j, k, photo, point, row
      double precision  om, ph, ka
      double precision  dX, dY, dZ, xnum, ynum, denom

C     ******************************
C     ******************************
C     ***  EXECUTABLE STATEMENTS ***
C     ******************************
C     ******************************

C     *************************************
C     ***  Open files; read parameters  ***
C     *************************************

      open (unit=1,
     &      file = 'photo.dat',
     &      status = 'old')

      read (1, 100) numpho, numpts
  100 format (2i4)
      m = numpho * numpts * 2
c      n = (numpho * 6) + (numpts * 3)
      n = (numpho * 6)

      read (1,101) foclen, xpp, ypp
  101 format (3f10.3)

C     ****************************
C     ***  Read the data file: ***
C     ****************************

      do 10 photo = 1, numpho
         read(1,101) XL(photo), YL(photo), ZL(photo)
         read(1,101) om, ph, ka
         omega(photo) = om * rho
         phi(photo) = ph * rho
         kappa(photo) = ka * rho
         do 5 point = 1, numpts
            read(1,101) xi(point, photo), yi(point, photo)
    5    continue
   10 continue
      do 15 point = 1, numpts
         read(1,101) XP(point), YP(point), ZP(point)
   15 continue


      do 20 row = 1, (m-1), 2
         call Brows(row, B, ldb, f, ldf)
C        ***  Other variables are contained in  ***
C        ***  common block PHOTOS.              ***
   20 continue


      open (unit=2,
     &      file = 'photo.res',
     &      status = 'new')
```

```
      return
      end
```