NOAA Technical Memorandum NOS NGS-39

HEART OF GOLD: COMPUTER ROUTINES FOR LARGE, SPARSE, LEAST SQUARES COMPUTATIONS

Dennis G. Milbert

Rockville, Md.
April 1984

Reprinted May 1997

UNITED STATES DEPARTMENT OF COMMERCE
Malcolm Baldrige, Secretary

National Oceanic and Atmospheric Administration
John V. Byrne, Administrator

National Ocean Service
Paul M. Wolff, Asst. Administrator

Charting and Geodetic Services
R. Adm. John D. Bossler, Director

NOAA Technical Memorandum NOS NGS-39

HEART OF GOLD: COMPUTER ROUTINES FOR LARGE, SPARSE, LEAST SQUARES COMPUTATIONS

Rockville, Md.
April 1984

Reprinted May 1997

U.S. DEPARTMENT OF COMMERCE / National Oceanic and Atmospheric Administration / National Ocean Service

Contents

HEART OF GOLD: COMPUTER ROUTINES FOR LARGE, SPARSE, LEAST SQUARES COMPUTATIONS

Dennis G. Milbert
National Geodetic Survey
Charting and Geodetic Services
National Ocean Service, NOAA
Rockville, Maryland 20852

ABSTRACT. A collection of routines for processing large, sparse, least squares systems is described. The routines are highly transportable and support structured development of programs.

INTRODUCTION

A standard problem in the analysis of data is the least squares solution. Large agencies may easily compute a hundred solutions each day, and even small solutions involve more than one hundred parameters. Clearly, massive computational resources are expended on least squares computations.

With the need to more efficiently compute least squares problems came research to handle larger and larger equation systems. This research, as it applies to the direct elimination method of solution, covered two areas:

1) Exploitation of the sparsity of large least squares problems by the use of data structures.

2) Use of backing (auxiliary) storage to hold sections (pages) of the equations until needed for computations in main memory.

The need for more computer resources, driven by a community much larger than one which only performs least squares solutions, stimulated research in three key fields of computer science:

1) Development of Very Large Scale Integration (VLSI).

2) Creation of sophisticated computer operating systems which supported multiprograming, multiprocessing, and virtual memory allocation.

3) Structured programing.

The work in VLSI has led to the development of both large and small computers. Each type has impressive capabilities when compared to versions only a few years old. The advances in operating systems have allowed more efficient utilization of resources in a given computer system. The software engineering techniques generally named "structured programing" address the problems in software development and maintenance.

Research in computer science has made a far-reaching change in the environment of data analysis. Each day fewer people keypunch cards, submit them to

a computer operator, and wait for the printout. And, more people have access to personal computers which can also function as terminals to a time-sharing system for larger problems.

The proliferation of computers at all levels of capabilities has seriously increased the problem of program conversion. With the price of hardware dropping rapidly, the time and human resources involved in software conversion, development, and maintenance have become key factors: Expressed simply, "Chips are cheaper than people."

In this paper I describe a collection of subroutines for the creation, solution, and inversion of sparse systems of equations for least squares problems. I have named these routines HEART OF GOLD, after the Infinite Improbability Drive powered starship from the novel, The Hitchhiker's Guide to the Galaxy (Adams 1980).

The routines in the package were developed to meet particular research needs of the author, and are presented here in the spirit that these routines may be of more general use. These routines are currently implemented in the FORTRAN 77 language subset. No machine dependent features, no operating system calls, and no assembly language routines are used in this package. The routines are highly transportable. This software is available by contacting the National Geodetic Information Center (N/CG174), National Oceanic and Atmospheric Administration, Rockville, Maryland 20852 (telephone 301-443-8623).

STATEMENT OF THE LEAST SQUARES PROBLEM

The least squares problem is

$$\text{minimize} \qquad \phi(X) = \tfrac{1}{2}\, V(X)' \sum_{L_b}^{-1} V(X) \tag{1}$$

where V(X) is a residual function (Dennis and Schnabel 1983). Consider the method of observation equations (Mikhail 1976, Schwarz 1974, or Uotila 1967)

$$L_a = F(X_a) \tag{2}$$

where L_a is a vector of computed observation values of length n, X is a vector of model parameters of length u, and F is a vector of functions that is a theoretical model which describes the observations in terms of the parameters. In this method

$$V(X) = F(X_a) - L_b \tag{3}$$

where L_b is the vector of actual observations.

Assume model $F(X_a)$ is not strongly nonlinear and the problem is a small residual problem. These simplifying assumptions allow selection of the Gauss-Newton method of solution.

The design matrix, A, is defined as

$$A = \left.\frac{\partial F}{\partial X_a}\right|_{X_a = X_0} \tag{4}$$

where A is a matrix of differential changes in the observation model with respect to the parameters, X_a, evaluated at a particular set of parameter values, X_0. A vector of observation misclosures is

$$L = L_b - L_a \tag{5}$$

where L_b and L_a are described above.

Associated with the observation vector L_b is a symmetric variance-covariance matrix $\sum_{L_b}$, which contains information on observation precision and correlation.

The observation equation may now be written as

$$AX = L + V \tag{6}$$

where V is a vector of residual errors and X is a vector of corrections to the parameter vector X_a. The least squares estimate of X is

$$X = \left(A^t \sum_{L_b}^{-1} A \right)^{-1} A^t \sum_{L_b}^{-1} L. \tag{7}$$

This is a solution of the normal equations

$$NX = U \tag{8}$$

where the matrix of normal equation coefficients is

$$N = A^t \sum_{L_b}^{-1} A \tag{9}$$

and the right hand vector is

$$U = A^t \sum_{L_b}^{-1} L \tag{10}$$

One technique for solution is to compute a Cholesky decomposition of the normal matrix, N,

$$NX = R^t RX = U. \tag{11}$$

The estimate of X provides a new set of values for our parameters by

$$X_a + X \longrightarrow X_a \tag{12}$$

If the observation model $F(X_a)$ is nonlinear (that is, A is not constant for any set of X_a), then the entire process, starting with eq. (2), must be iterated until the vector X_a reaches a stationary point.

Estimates of parameter precision and correlations are given by the adjusted parameter variance-covariance matrix, $\sum_{x_a}$. This matrix is computed by

$$\sum_{x_a} = \left(A^t \sum_{L_b}^{-1} A \right)^{-1} \tag{13}$$

Of course, this matrix can always be scaled by an estimated variance of unit weight, if such an estimate is felt valid. The user may also compute the precision of any other quantity which can be derived from the parameters. Suppose one wishes to compute a vector of quantities, S,

$$S = S(X_a) \tag{14}$$

from the adjusted parameters, X_a. A geometry matrix, G, is defined as

$$G = \left. \frac{\partial S}{\partial X_a} \right|_{X_a = X_0} \tag{15}$$

where G is a matrix of differential changes in the functions, S, with respect to the parameters, X_a, evaluated at a particular set of parameter values, X.

By the principle of linear error propagation,

$$\sum_S = G \sum_{X_a} G^t \tag{16}$$

or

$$\sum_S = G \left(A^t \sum_{L_b}^{-1} A \right)^{-1} G^t \tag{17}$$

where $\sum_s$ is the variance-covariance matrix of the computed quantities.

This last equation is useful since its terms are quantities derivable from the parameters. It could be used, for example, to compute $\sum_v$ or $\sum_{L_a}$.

Use of this equation assumes that the model is not too nonlinear, that the parameter vector X has been adequately estimated by the method of least squares, that the design matrix, A, and the geometry matrix, G, are known, and that the variance-covariance matrix of the observations $\sum_{L_b}$ is known.

Another useful quantity is the correlation coefficient, ρ, where

$$\rho = \frac{\sigma_{ij}}{\sigma_{ii}\ \sigma_{jj}} \tag{18}$$

and σ_{ij} is the value in the i-th row and j-th column of the adjusted parameter variance-covariance matrix, $\sum_{x_a}$. These correlations can be thought of as a normalized covariance, since

$$-1 \leq \rho \leq 1. \tag{19}$$

Correlations between parameters that approach ±1 indicate ill-conditioning of the system.

APPLICATION OF HEART OF GOLD

The HEART OF GOLD routines are able to process large least squares systems by exploiting the sparse nonzero structure of a problem. Speed in execution is attained by using a static data structure. Specifically, the rows of the system are stored end-to-end in an array. The price paid for this rapid processing is a requirement that the structure of each least squares problem be known before the equations are accumulated and solved.

For this reason, HEART OF GOLD may be envisioned as a pair of modules. One module performs the analysis of the structure of a least squares problem. This structure is stored in an INTEGER array NX(). The other module performs the computations using the structure determined by the first module. The elements of the normal equations are stored in a REAL array A(). (Of course, if the user requires more precision than is provided by the REAL data type on a particular machine, then the routines can be easily modified to accommodate the DOUBLE PRECISION data type.)

Appendix A contains a highly detailed description of the HEART OF GOLD function calls. The reader may wish to refer to the appendix while reading the following material.

Structure Analysis of a Least Squares Problem

It is easily shown that the normal equations for a least squares problem are symmetric. The first step in exploiting structure is to save only the upper triangular (or lower triangular) portion of N (eq. 9). The model parameters, X, can be ordered so that the nonzero elements will fall in a diagonal band whose width is small with respect to the rank of the system (minimize bandwidth). Figure 1 shows an example of this band storage structure. Here the programer processes a system where all the nonzero elements fit into a bandwidth of three. This structure can be completely described by storing the array indices of the diagonal elements in an INTEGER index array, NX(). Clearly, if the bandwidth is one, then the system is strictly diagonal. And if the bandwidth is equal to the order, then the system is full, and sparsity is not exploited.

The index array NX() can be computed by executing the function BAND. The only knowledge required is the order of the system and an adequate bandwidth. If BAND returns a false value, then insufficient storage was allocated to NX(). As can be expected, the execution of BAND is very rapid.

Research has shown that even greater savings in storage and execution times can be achieved by using a variable bandwidth (profile) structure (Jennings 1977). Figure 2 illustrates the variable bandwidth scheme. As with the fixed bandwidth scheme, the structure is completely described by the index array NX().

Since it is not reasonable to expect the user to have prior knowledge of an optimum profile for a given least squares problem, HEART OF GOLD includes functions to determine a near-optimum profile. The user will first initialize

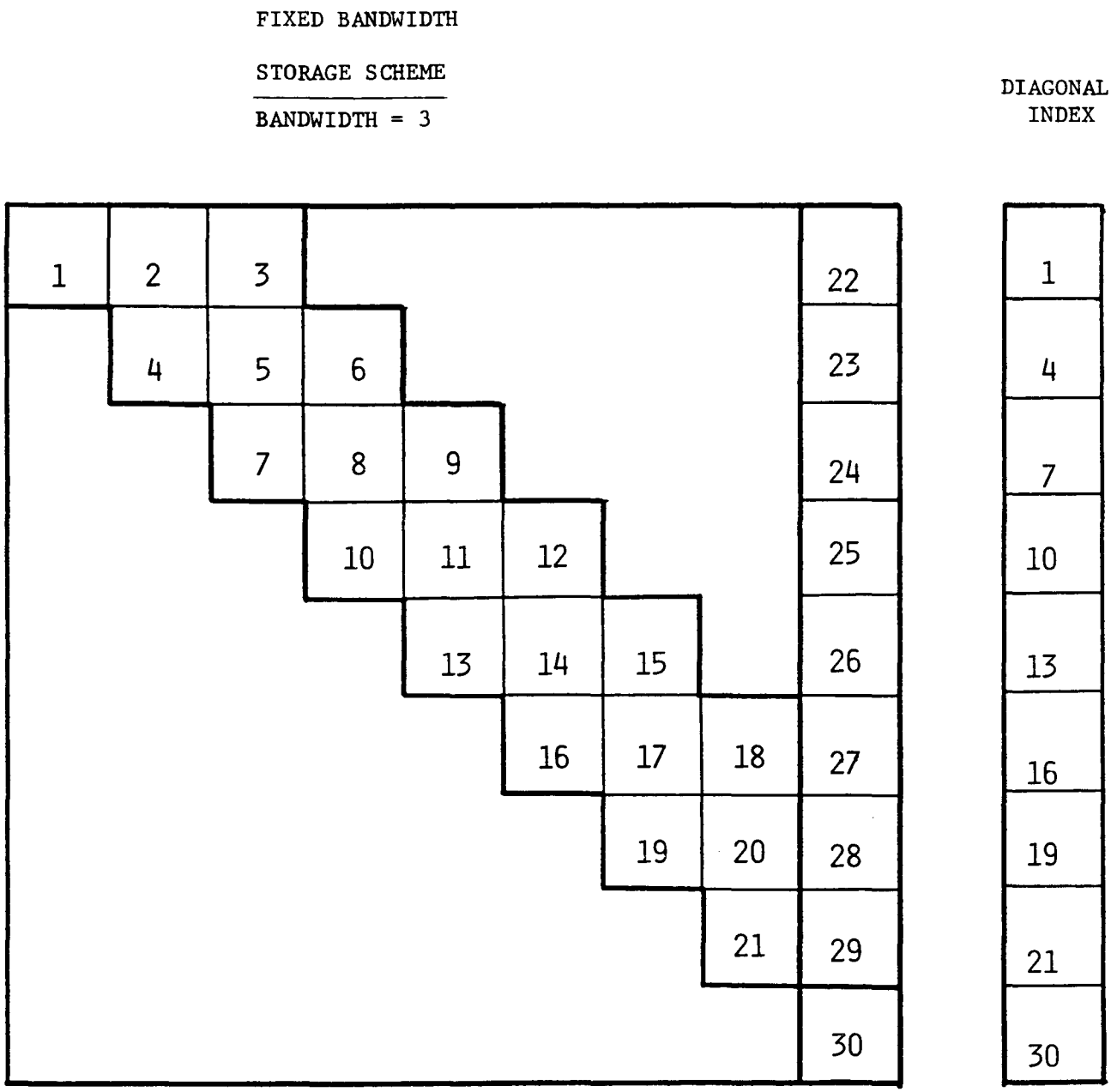

Figure 1.--Fixed bandwidth storage structure.

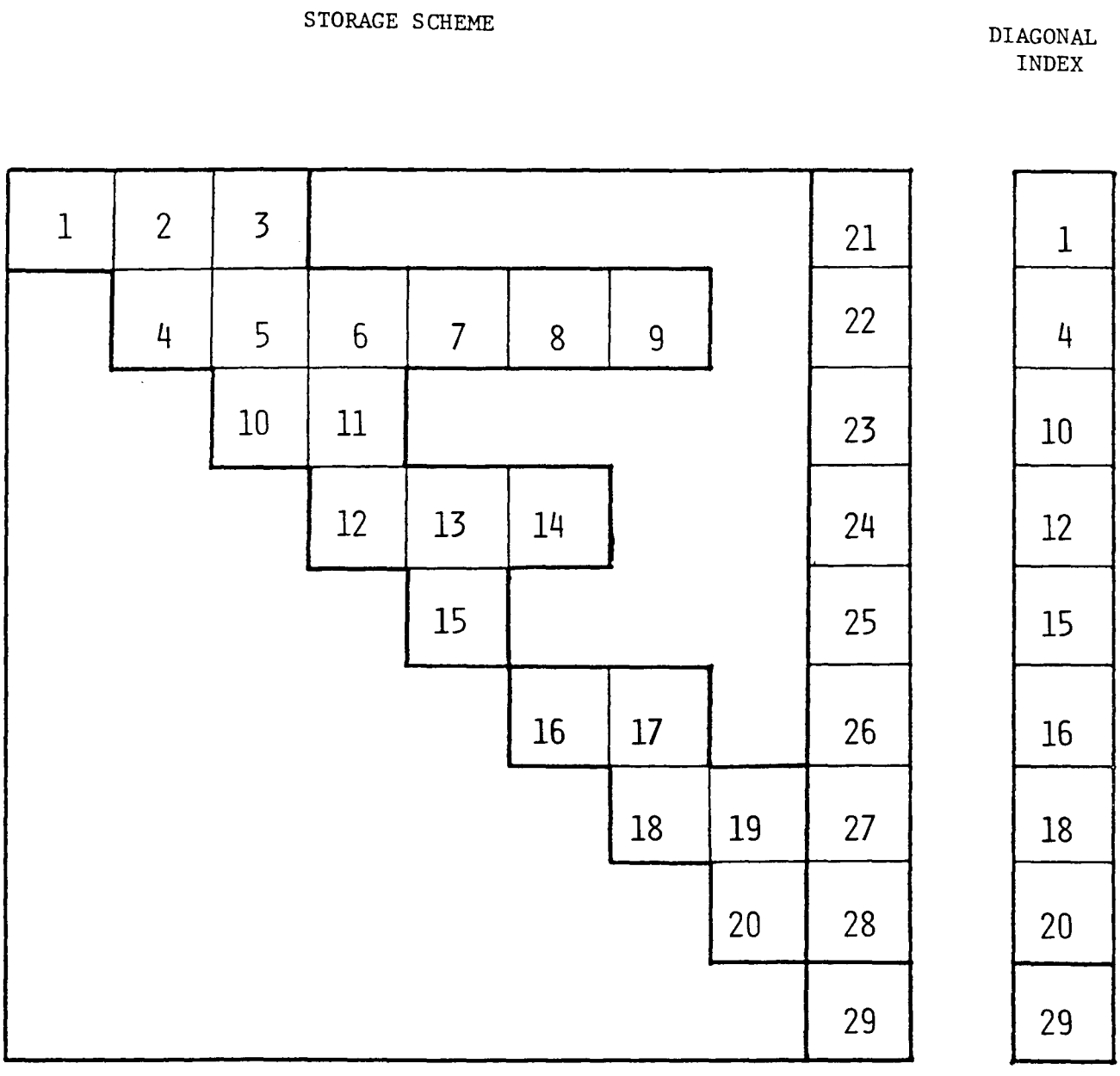

Figure 2.--Variable bandwidth storage structure.

the NX() array, then accumulate connectivity information, and finally analyze the structure.

The structure is initialized by calling OPENG. Knowledge of the order of the system is required. Connectivity information is provided by repeated calls of the function ADDCON. An INTEGER array IC() must be furnished which holds the indices of the nonzero coefficients of a given observation equation. If ADDCON returns a false value, then insufficient storage was allocated to NX(). Once all the connections have been accumulated, a near-optimal structure is determined by a call to subroutine REORDR.

One disadvantage of analyzing the connectivity is that it takes longer than a simple invocation of BAND. However, this disadvantage is offset by requiring less knowledge about a given least squares problem. The structure of a least squares problem (either fixed or variable bandwidth) depends upon how the parameter indices are associated with the model parameters. Structure will vary as the user permutes (reorders) the parameters. The structure analysis performed in REORDR permutes the parameter indices to give a near-optimal profile and stores this profile in NX(). In addition, tables are computed which relate the user parameter indices to the internal parameter indices that produced the near-optimal profile. The user need never consider the internal indices.

Solution of a Least Squares Problem

Once a structure has been determined by invocation of BAND or REORDR, the least squares problem is ready to be solved. The normal equations are initialized by calling function OPENN. It returns a false value if insufficient storage was allocated to A().

The observation equations are accumulated one at a time into the normal equations by repeated calls of the ADDOBS function. The user supplies the INTEGER array IC() that holds the indices of the nonzero coefficients for a given observation equation. A REAL array C() that holds the nonzero coefficients themselves must also be provided, as well as the weight, WT, of the observation equation, and the misclosure, OBS, computed by eq. (5). Observation equations may be accumulated in any order. ADDOBS returns a false value if the structure defined earlier cannot hold the observation equation. In this circumstance, the programer must use a larger bandwidth or a different scheme of numbering the parameters or actually analyze the structure using the HEART OF GOLD routines.

In some instances, sets of correlated observations must be processed rather than uncorrelated observations. The coefficient array C() is now two dimensional, where the second dimension spans all the correlated observations in the set. The misclosure, OBS(), is now an array with a length of the number of correlated observations, and the weight, WT, is replaced by the two dimensional variance-covariance matrix, SIGOBS(). This array holds the covariance information. The call to function ADDCOR will decorrelate the set of observation equations and accumulate them into the normal equations. ADDCOR will return a false value if an incorrect variance-covariance matrix is supplied or if an incorrect structure is being used.

Once all the observations have been processed, the system is solved using a Cholesky factorization by calling function SOLVE. It will return a false value if the system was never initialized by OPENN, or if singularities were

encountered. A singularity is detected by using a small, positive singularity tolerance, TOL, supplied by the user. The number of singularities is returned in LSING. The parameter index numbers are returned in an array, ISING(), and the associated normalized diagonal values are returned in an array, GSING(). The singularities are repaired by replacing the singular diagonal with a large positive number. This is equivalent to constraining that singular unknown to its preliminary value. The Cholesky factor and the values of the corrections to the model parameters replace the normal equations in the structure in all cases. Of course, if the system is singular, then the user must be cautious in the use of the model parameter corrections or any other quantities derived from the singular Cholesky factor.

The values of the parameter corrections may be retreived by calling function GETA. A row (or column) of IORDR+1 should be included. The column (or row) will be the parameter index. In fact, GETA can be used to inspect any element of the normal equations or Cholesky factor by supplying the appropriate row and column indices, I & J. GETA will return a false value if the requested element is not in the profile.

Elements of the normal equations may be overwritten by a complementary function, PUTA. It will return a false value if the element to be replaced is not in the profile. PUTA should be used with caution, since an incorrect value can inadvertently change the normal equations. Use of ADDOBS or ADDCOR will accumulate the sum of the squared, weighted misclosures in location (IORDR+1,IORDR+1). Calling SOLVE will also compute the sum of the squared, weighted, linear residuals and store this value in the same location. This quantity could be monitored in an iterative solution to indicate convergence.

On rare occasions, the need arises to solve the same system with a new right hand vector (eq. 10), U, computed from a new misclosure vector, L. In these situations, the work involved in computing the Cholesky factor can be retained. Repeated calls of PUTA are required to place the new elements of the right hand vector, U, into the system. Calling function RESOLV computes a new set of parameter corrections, X, for the new system. RESOLV returns a false value if the equations have not yet been factored by SOLVE.

Analysis of a Least Squares Problem

In addition to the solution vector, X, of a least squares problem, the programer often needs statistics about the solution. One useful element is the variance of a parameter, which is on the diagonal of $\sum_{x_a}$. Hanson (1978) has shown the inverse elements of the normal equations can be computed within the profile in place, at no greater cost than in computing only the diagonal elements of the inverse. Calling the function INVERT computes the inverse within profile from the Cholesky factor. GETA can then be used to inspect the elements of the variance-covariance matrix of the adjusted parameters (eq. 13). INVERT returns a false value if the equations have not yet been factored by SOLVE.

Other advantages accrue from computing all the elements of the inverse within the profile. If desired, variance of quantities can be computed and can be derived from the parameters. Among the quantities which can be so determined, without involving any elements of the inverse outside the profile, are variances of the adjusted observations and variances of the residuals. This computation is known as error propagation, and is governed by eq. (16).

The variance of a quantity, VAR, can be computed by calling function PROP. The user must supply the coefficient array, C(), and the associated index array, IC(), for the linearized equation. Similarly, the covariance between two equations, COV, is computed by calling function PROPCV. The user supplies two coefficient arrays, C1() and C2(), and associated index arrays, IC1() and IC2(), for the pair of linearized equations. PROP and PROPCV return a false value if the system has not yet been inverted by INVERT or if the computation requires covariance elements not stored within the profile.

Finally, the user may desire to compute the correlation coefficients for all the elements within the profile. This is done by calling function CORR. It returns a false value if the system has not yet been inverted.

IMPLEMENTATION FEATURES

In this section I will cover a few points on HEART OF GOLD, and present information useful to the advanced user of these routines.

Transportability

Great care was taken to ensure HEART OF GOLD was implemented in the American National Standards Institute (ANSI) standard ANSI X3.9-1978 (FORTRAN 77). As of this writing, HEART OF GOLD has been compiled in APPLE FORTRAN (APPLE), VS FORTRAN (IBM), FORTRAN level 10R1 (UNIVAC), and FORTRAN 77 (HP) with no editing required and no error messages generated. The proliferation of computers and language variants makes strict adherence to the ANSI standard the central virtue of HEART OF GOLD.

Data Abstraction and Modularity

One of the major advances in computer science in the past two decades has been development of "structured programing" techniques. Two ideas imbedded in HEART OF GOLD are those of data abstraction and modularity. Data abstraction separates the logical view of the data from the internal storage structures (Isner 1982). Modules perform specific tasks and make a minimum number of "assumptions" about processes in other parts of a program (Turner 1980).

It can be seen from the earlier section on application, that two data abstractions are used in HEART OF GOLD. The structure and connectivity of the least squares problem are managed by OPENG, ADDCON, REORDR, and BAND. The accumulation, solution, and analysis of the least squares problem is managed by the remaining routines. A knowledge of internal storage structure is not needed to use HEART OF GOLD. However, some knowledge is helpful in selecting between fixed and variable bandwidth for a particular least squares problem.

It may also be inferred that HEART OF GOLD is a pair of modules. One module operates on structure. The other module operates on the numbers. The modules are constructed from the way in which the least squares problem is abstracted.

To allow the user ease in discarding routines not needed for a particular problem, the modules were designed as subprograms sharing common variables. The equation module uses a COMMON block named DENNIS, and the structure module uses a COMMON block named KATHY (named after the author and his wife). These COMMON blocks carry "housekeeping" details not needed by the user. Storage of

data in this fashion is known as "information hiding," a concept derived from data abstraction and modularity.

The idea of information hiding has been deliberately violated in HEART OF GOLD. This was done by giving the user access to A() and NX(), primarily to achieve transportability by strictly conforming to the FORTRAN 77 standard. An advanced user may access A() and NX(). For most applications, however, the programer will not directly access the arrays except by use of functions GETA and PUTA.

Whenever an operation on the structure or the equations had the potential of failure, this fact was always signaled back to the calling routine. This was done by writing most of the HEART OF GOLD subprograms using a LOGICAL FUNCTION, allowing the user to detect and handle errors with nicely structured code. For example,

```
IF(.NOT.INVERT(A,NX)) THEN
  WRITE(6,'('' PROGRAMMER ERROR -- EQUATIONS NOT INITIALIZED'')')
  STOP 666
ENDIF
```

The IF() THEN ... ELSE ... ENDIF structure is also useful for error handling. A LOGICAL FUNCTION supports "defensive programing," and leads to more robust software.

Memory Management

Advances in VLSI and computer operating systems have greatly changed the computational environment. In the past, throughput on a mainframe was linked to the user, minimizing the amount of random-access memory (RAM) required. This led to development of software which would page the normal equations in and out of backing storage, while it accumulated and solved the equations (Dillinger 1981). With personal computers, a user now has exclusive access to 128K bytes or more of RAM. One personal computer is shipped with one full megabyte of RAM. On these systems, the user would simply allocate A() and NX() to be as large as desired.

Mainframe computing has kept pace by development of virtual operating systems. These systems automatically transfer programs and data between two or more memory levels (Schmitt 1983). These have greatly increased address space in a mainframe, and lowered programing costs by eliminating the need for manual overlays and data transfer. A user should again allocate A() and NX() as large as desired on virtual systems.

A piece of folk wisdom from the early days of virtual systems was: "A user can page memory to backing storage better than the operating system, since the user knows more about his or her problem." This statement can be debated on a number of levels. A virtual system will page memory on a global basis, increasing total throughput, while a user can only invoke some strategy local to the particular least squares problem. Now, virtual systems have special hardware, computer architectures, and input-output channels for address translation and memory paging. Paging strategies for virtual systems are based on some variation of locality of reference, where the seldom accessed memory segment is paged out first. The HEART OF GOLD storage structures (either fixed or variable bandwidth) support a locality of reference paging strategy. If the reader envisions a Cholesky factorization proceeding

row-by-row on the row storage structure, one finds excellent locality of reference. Any user defined strategy would not significantly deviate from paging by the virtual system. User paging is hampered by address translation in the software and by the slow speed of normal input-output. The latter drawback can be alleviated somewhat by doing input-output without buffers or by allocating larger buffers. Input-output without buffers is invariably system dependent. It requires assembly language routines or special operating system calls, which degrade the transportability of the software. Allocation of large buffers on a virtual system will lead to two levels of paging. The virtual system will page the buffers and the normal equations which are, in turn, paged by the user.

Storage Structure

The information in this section is intended for advanced users of HEART OF GOLD. Reference to George (1981) is helpful.

The user can easily envision how to compute the diagonal indices for a fixed bandwidth, when given the order of the system and the bandwidth, by examining figure 1. So, I will concentrate on the variable bandwidth (profile) structure of figure 2. Here the diagonal indices are trivially computed from the row widths. The row widths can be detemined by processing an adjacency structure (George 1981: 41-42). However, an adjacency structure is best determined from a connection table, since a connection table is easily updated while an adjacency structure is not. As seen in figure 3, NX() is partitioned by HEART OF GOLD into several segments.

Figure 3.--Memory partition for connectivity accumulation.

IHEAD is of length IORDR, NBR is the length of two times the maximum number of edges (IEMX), and LINK is of length IEMX. These three segments hold the connection table built by function ADDCON. Upon calling REORDR, the connection table is translated into an adjacency structure. This is stored in NBRLST of length IEMX and in NBRPNT of length IORDR+1. The user does not need to provide IEMX. It is computed as a maximum to utilize all the storage available to NX().

Recall that the row width and storage structure vary with the order of the parameters, even though the connectivity and adjacency do not. HEART OF GOLD uses a Cuthill-McKee ordering of the unknowns computed by the algorithms and routines presented in George (1981). This ordering scheme was selected because of the low processing times found by George (1981: 277-294). A Cuthill-McKee order for a row storage structure is identical to a Reverse Cuthill-McKee order for a column structure. By envisioning the normal equations in the lower triangular form, these orders are identical to a Reverse Cuthill-McKee order for a row structure, and a Cuthill-McKee order for a column structure.

In the Cuthill-McKee computation, NX() is repartitioned as shown in figure 4.

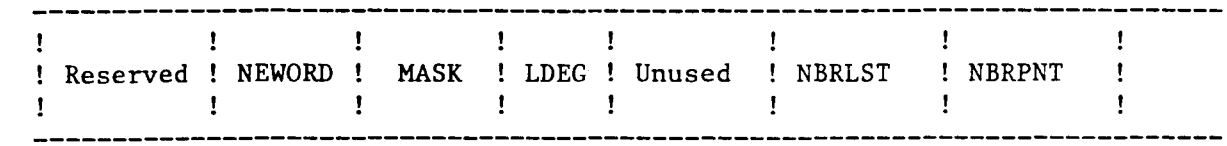

Figure 4.--Memory partition for Cuthill-McKee computation.

The reserved area is of length IORDR+1. NEWORD holds the Cuthill-McKee order, and is of length IORDR. NEWORD determines an internal index number from a user-provided index number. Both MASK and LDEG are used by the reorder algorithms, and are of length IORDR. NBRLST and NBRPNT continue to hold the adjacency structure.

After the Cuthill-McKee computation, NX() is again repartitioned as shown in figure 5.

Figure 5.--Memory partition after reordering.

The order in NEWORD is inverted and stored in INVORD, which is of length IORDR. INVORD determines a user-defined index number from an internal index number. The profile is stored in IPROF, and is of length IORDR+1. As mentioned before, IPROF is determined from the row widths, which are determined from the adjacency structure still stored in the high end of NX().

It is seen that the NX() returned by REORDR holds not only the profile (IPROF), but also translation tables (NEWORD and INVORD) which relate the user parameter indices to the internal parameter indices. The remaining storage is no longer accessed by the HEART OF GOLD routines. The advanced user may wish to store the A() array in the unused portion of NX() by using a memory overlay. This would be done by

```
EQUIVALENCE (A(1),NX(1))
```

and by always referencing A() with a constant offset, IOFFST. For example, to initialize the normal equations,

```
IF(.NOT.OPENN(A(IOFFST),NX,IORDR,LENG-IOFFST)) THEN ....
```

A() may be single precision, double precision, or even extended double precision. NX() could be long integer or short integer. So the value of IOFFST will depend upon a particular application and the number of INTEGER words that may be stored in a REAL word.

In some least squares problems, the parameters are "clustered." For example, parameters might be coordinates (latitude, longitude, height). These parameters would always appear as a trio in any observation equation. Substantial savings in the structure analysis can be realized by accumulating connections and computing Cuthill-McKee order for the coordinate triplets themselves, rather than for individual parameters. In such a case, IORDR

passed in OPENG would not be the order of the least squares problem, but rather would be the number of coordinate triplets. When calling ADDCON, IC() would contain triplet indices rather than parameter indices.

Naturally, after calling REORDR, NX() will hold a profile and translation tables for the triplets. The profile and tables will have to be converted to their appropriate representations for parameters before invoking OPENN. However, the conversion should take less processing than the full structure analysis of a large least squares problem. The user may also be faced with a "variable clustering" problem when dealing with quadruplets, pairs, and single parameters as well as triplets. For example, the problem may require solving a number of instrument scale errors and constant biases, as well as coordinates. The instrument parameters cluster in pairs, and the coordinate parameters cluster in triplets.

Such issues complicate the conversion to a parameter structure, and will vary with each application. The user must determine if exploiting parameter clustering is worthwhile. HEART OF GOLD was written as two modules to allow this flexibility. Of course, this feature is optional. Normal invocation of HEART OF GOLD, where each parameter is single, will work correctly.

SUMMARY

HEART OF GOLD is a pair of modules which perform the structure analysis and the solution of large, sparse, least squares problems. The routines are designed to encourage good programing practices, yet allow the advanced user to directly access the structure and normal equation elements. Two structures, a fixed and a variable bandwidth, and one reordering algorithm are provided. Of course, the user could develop other reorder algorithms, and replace the appropriate elements of the structure array, NX(). Great care has been taken to adhere to the ANSI standard, and not to use system dependent features in the implementation of HEART OF GOLD.

REFERENCES

Adams, Douglas, 1980: The Hitchhiker's Guide to the Galaxy. Pocket Books division of Simon & Schuster, New York, N.Y., 215 pp.

Dennis, J. E. and Robert B. Schnabel, 1983: Numerical Methods for Unconstrained Optimization and Nonlinear Equations. Prentice-Hall, Inc., Englewood Cliffs, N.J., 378 pp.

Dillinger, William H., 1981: Subroutine package for large, sparse, least squares problems. NOAA Technical Memorandum NOS NGS 29, National Geodetic Information Center, NOAA, Rockville, Md. 20852, 18 pp.

George, Alan, 1981: Computer Solution of Large Sparse Positive Definite Systems. Prentice-Hall, Inc., Englewood Cliffs, N.J., 324 pp.

Hanson, Robert H., 1978: A posteriori error propagation. Proceedings of the Second International Symposium on Problems Related to the Redefinition of North American Geodetic Networks, Washington, D.C., 427-445. National Geodetic Information Center, NOAA, Rockville, Md. 20852.

Isner, John F., 1982: A programming methodology based on data abstraction. Bulletin Geodesique, 56(2), 149-164.

Jennings, Alan, 1977: Matrix Computation for Engineers and Scientists. John-Wiley & Sons, New York, N.Y., 330 pp.

Mikhail, Edward M., 1976: Observation and Least Squares, IEP Dun-Donnelley publisher, New York, pp. 285-288.

Schmitt, Stephen, 1983: Virtual memory for microcomputers. Byte, 8(4), 210-238.

Schwarz, Charles R., 1974-75: Adjustment computations (lecture notes), Department of Geography and Regional Science, George Washington University, Washington D.C.

Turner, Joshua, 1980: The structure of modular programs. Communications of the ACM, 23(5), 272-277.

Uotila, Urho A., 1967: Introduction to adjustment computations with matrices (lecture notes), Department of Geodetic Science, Ohio State University, Columbus, Ohio.

APPENDIX A. --HEART OF GOLD FUNCTION CALLS

```
LOGICAL = BAND(IORDR,NX,NXLENG,LBAND)

CALL OPENG(IORDR,NX,NXLENG)

LOGICAL = ADDCON(IC,LEN,NX)

CALL REORDR(NX)

LOGICAL = OPENN(A,NX,IORDR,LENG)

LOGICAL = ADDOBS(C,IC,LEN,OBS,WT,A,NX)

LOGICAL = ADDCOR(C,IC,OBS,SIGOBS,LUNK,LOBS,A,NX,IFLAG)

LOGICAL = SOLVE(A,NX,TOL,ISING,GSING,LSING)

LOGICAL = GETA(I,J,VAL,A,NX)

LOGICAL = PUTA(I,J,VAL,A,NX)

LOGICAL = RESOLV(A,NX)

LOGICAL = INVERT(A,NX)

LOGICAL = PROP(C,IC,LEN,VAR,A,NX,IFLAG)

LOGICAL = PROPCV(C1,IC1,LEN1,C2,IC2,LEN2,COV,A,NX,IFLAG)

LOGICAL = CORR(A,NX)

LOGICAL = IN(I,J,NX,INDEX)
```

LOGICAL FUNCTION BAND(IORDR,NX,NXLENG,LBAND)

FUNCTION: Determine the profile using a fixed bandwidth.

BAND — Success of the profile computation. Output, LOGICAL.
TRUE -- Profile computed.
FALSE -- Insufficient storage allocated to NX.
(Increase NXLENG to 3 * IORDR + 1)

IORDR — Order of the normal equations. Input.
INTEGER scalar variable

NX — Index vector to describe the profile. Input.
INTEGER array, length of NXLENG

NXLENG — Length of the index vector. Input.
INTEGER scalar variable

LBAND — Fixed bandwidth for the profile. Input.
LBAND must be between 1 and IORDR inclusive.
INTEGER scalar variable

SUBROUTINE OPENG(IORDR,NX,NXLENG)

FUNCTION: Initialize the connection matrix for profile determination.

IORDR — Order of the normal equations. Input.
INTEGER scalar variable

NX — Index vector to describe the profile.
INTEGER array, length of NXLENG

NXLENG — Length of the index vector. Input.
Allocate at least 6 * number of unique connections + 2 * IORDR
INTEGER scalar variable

LOGICAL FUNCTION ADDCON(IC,LEN,NX)

FUNCTION: Accumulate observation equation connections into the profile array.

ADDCON — Success of the accumulation of the connections. Output,LOGICAL.
TRUE -- All connections accumulated.
FALSE -- Insufficient storage allocated to NX.
(Increase NXLENG in subsequent OPENG)

IC — Parameter index of the observation equation. Input.
INTEGER array, length of LEN.

LEN — Length of the parameter index array. Input.
INTEGER scalar variable.

NX — Index vector describing profile.
INTEGER array, length of NXLENG

SUBROUTINE REORDR(NX)

FUNCTION: Determine a minimum profile by internal reorder of parameter indices.

NX — Index vector describing profile.
INTEGER array, length of NXLENG

LOGICAL FUNCTION OPENN(A,NX,IORDR,LENG)

FUNCTION: Initialize the normal equations

OPENN — Success of the initialization. Output, LOGICAL
TRUE -- Equations initialized
FALSE -- LENG, length of storage, incompatable with profile requirements

A — Elements of the normal equations
REAL array, length of NX(IORDR+1)+IRANK

NX — Index vector describing profile
INTEGER array, length of NXLENG

IORDR — Order of the normal equations. Input
INTEGER scalar variable

LENG — Length of the normal equation array, A. Input
Must be greater than or equal to the value of NX(IORDR+1)+IORDR.
INTEGER scalar variable.

LOGICAL FUNCTION ADDOBS(C,IC,LEN,OBS,WT,A,NX)

FUNCTION: Accumulate an observation equation into the normal equations.

ADDOBS — Success of the accumulation of the observation equation. Output.
TRUE -- All elements accumulated within profile.
FALSE -- All elements not within profile.
(CAUTION: Normal equations damaged)

C — Coefficients of the observation equation. Input.
REAL array, length of LEN.

IC — Parameter index of the observation equation. Input.
INTEGER array, length of LEN.

LEN — Length of the observation equation arrays. Input.
INTEGER scalar variable.

OBS — Misclosure of the observation equation. Input.
REAL scalar variable.

WT — Weight of the observation equation. Input.
REAL scalar variable.

A — Elements of the normal equations.
REAL array, length of NX(IORDR+1)+IORDR

NX — Index vector describing profile. Input.
INTEGER array, length of NXLENG

LOGICAL FUNCTION ADDCOR(C,IC,OBS,SIGOBS,LUNK,LOBS,A,NX,IFLAG)

FUNCTION: Accumulate correlated observation equations into the normal equations.

ADDOBS Success of the accumulation of the observation equations. Output.
TRUE -- All elements accumulated within profile.
FALSE (IFLAG = 1) -- SIGOBS is not positive definite.
FALSE (IFLAG = 2) -- All elements not within profile.
(CAUTION: Normal equations damaged)

C Coefficients of the observation equations. Input.
One row for each observation equation, length of LOBS.
One column for each parameter, width of LUNK.
Two-dimensional REAL array, C(LOBS,LUNK)

IC Parameter index of the observation equations. Input.
INTEGER array, length of LUNK.

OBS Misclosures of the observation equations. Input.
REAL array, length of LOBS.

SIGOBS Positive definite covariance matrix of the obs. equations. Input.
One row and one column for each observation equation, length and width of LOBS.
Two-dimensional REAL array, SIGOBS(LOBS,LOBS).

LUNK Number of parameters in the observation equations. Input.
INTEGER scalar variable

LOBS Number of observation equations. Input.
INTEGER scalar variable

A Elements of the normal equations.
REAL array, length of NX(IORDR+1)+IORDR

NX Index vector describing profile. Input.
INTEGER array, length of NXLENG

IFLAG Failure return code. Output.
IFLAG = 0 -- Successful accumulation.
IFLAG = 1 -- SIGOBS is not positive definite.
IFLAG = 2 -- All elements not within profile.
(CAUTION: Normal equations damaged)
INTEGER scalar variable

LOGICAL FUNCTION SOLVE(A,NX,TOL,ISING,GSING,LSING)

FUNCTION: Cholesky factor and solve the normal equations.

SOLVE — Success of the Cholesky factorization and solution. Output.
TRUE -- Successful factor and solution.
FALSE (LSING=0) -- Attempt to solve non-initialized system.
FALSE (LSING#0) -- Singularities encountered.

A — Elements of the normal equations.
REAL array, length of NX(IRANK+1)+IRANK

NX — Index vector describing profile. Input.
INTEGER array, length of NXLENG

TOL — Singularity tolerance. Input.
REAL scalar variable.

ISING — Unknown index of the singularities. Output.
INTEGER array, length of LSING.

GSING — Googe number of the singularities. Output.
REAL array, length of LSING.

LSING — Number of singularities encountered. Output.
INTEGER scalar variable.

LOGICAL FUNCTION GETA(I,J,VAL,A,NX)

FUNCTION: Retreive an element of the normal equations.

GETA — Success of the retreival of element I,J. Output, LOGICAL
TRUE -- Element within profile placed in VAL
FALSE -- Element not within profile

I — Row of the element to be retreived. Input.
INTEGER scalar variable

J — Column of the element to be retreived. Input.
INTEGER scalar variable.

VAL — Retreived value. Output from function.
(0.0 if unsuccessful retreival)
REAL scalar variable.

A — Elements of the normal equations
REAL array, length of NX(IORDR+1)+IORDR

NX — Index vector describing profile. Input.
INTEGER array, length of NXLENG

LOGICAL FUNCTION PUTA(I,J,VAL,A,NX)

FUNCTION: Replace an element of the normal equations.

PUTA — Success of the replacement of element I,J. Output, LOGICAL
TRUE -- Replaces I,J element within profile
FALSE -- Element not within profile

I — Row of the element to be replaced. Input.
INTEGER scalar variable

J — Column of the element to be replaced. Input.
INTEGER scalar variable.

VAL — Replacement value. Input.
REAL scalar variable.

A — Elements of the normal equations.
REAL array, length of NX(IORDR+1)+IORDR

NX — Index vector describing profile. Input.
INTEGER array, length of NXLENG

LOGICAL FUNCTION RESOLV(A,NX)

FUNCTION: Solve the factored normal equations for a new right-hand side.

RESOLV Success of the solution. Output, LOGICAL.
TRUE -- Successful solution.
FALSE -- Attempt to solve system not yet factored or already inverted.

A Elements of the normal equations.
REAL array, length of NX(IORDR+1)+IORDR

NX Index vector describing profile. Input.
INTEGER array, length of NXLENG

Note: Since the normal equations have been factored, one only needs to read the right-hand side into A before invoking RESOLV, and to read the solution out of the right-hand side after using RESOLV. Since the profile always includes the right hand side, one could skip using PUTA and GETA by use of the following code.

```
*** COMPUTE INDICES

      INDEX  = NX(IORDR)
      INDEX2 = 2 * IORDR + 1

*** LOAD RIGHT HAND SIDE

      DO 100 I=1,IORDR
      ISQ=NX(INDEX2 + I)
  100 A(INDEX + ISQ)=RHS(I)

*** SOLVE AGAIN

      IF(.NOT.RESOLV(A,NX)) THEN
        error condition
      ENDIF

*** UNLOAD RIGHT HAND SIDE

      DO 200 I=1,IORDR
      ISQ=NX(INDEX2 + I)
  200 X(I)=A(INDEX + ISQ)
```

LOGICAL FUNCTION INVERT(A,NX)

FUNCTION: Compute the inverse of the normal equations.

INVERT Success of the inversion. Output, LOGICAL.
TRUE -- Successful inversion.
FALSE -- Attempt to invert system not yet factored or already inverted.

A Elements of the normal equations.
REAL array, length of NX(IORDR+1)+IORDR

NX Index vector describing profile. Input.
INTEGER array, length of NXLENG

LOGICAL FUNCTION PROP(C,IC,LEN,VAR,A,NX,IFLAG)

FUNCTION: Compute a variance by linear error propagation.

PROP Success of the propagation. Output, LOGICAL.
TRUE -- All required elements within profile.
FALSE (IFLAG = 1) -- System not yet inverted or already correlated.
FALSE (IFLAG = 2) -- All covariance elements not within profile.

C Coefficients of the geometry equation. Input.
REAL array, length of LEN.

IC Parameter index of the geometry equation. Input.
INTEGER array, length of LEN.

LEN Length of the geometry equation arrays. Input.
INTEGER scalar variable.

VAR Propagated variance. Output.
REAL scalar variable.

A Elements of the normal equations.
REAL array, length of NX(IORDR+1)+IORDR

NX Index vector describing profile. Input.
INTEGER array, length of NXLENG

IFLAG Failure return code. Output.
IFLAG = 0 -- Successful propagation.
IFLAG = 1 -- System not inverted or already correlated.
IFLAG = 2 -- All covariance elements not within profile.
INTEGER scalar variable

LOGICAL FUNCTION PROPCV(C1,IC1,LEN1,C2,IC2,LEN2,VAR,A,NX,IFLAG)

FUNCTION: Compute a covariance by linear error propagation.

PROP — Success of the propagation. Output, LOGICAL.
- TRUE -- All required elements within profile.
- FALSE (IFLAG = 1) -- System not yet inverted or already correlated.
- FALSE (IFLAG = 2) -- All covariance elements not within profile.

C1,C2 — Coefficients of the geometry equations. Input.
REAL array, length of LEN1 and LEN2 respectively.

IC1,IC2 — Parameter index of the geometry equations. Input.
INTEGER array, length of LEN1 and LEN2 respectively.

LEN1,LEN2 — Length of the geometry equation arrays. Input.
INTEGER scalar variable.

COV — Propagated covariance. Output.
REAL scalar variable.

A — Elements of the normal equations.
REAL array, length of NX(IORDR+1)+IORDR

NX — Index vector describing profile. Input.
INTEGER array, length of NXLENG

IFLAG — Failure return code. Output.
- IFLAG = 0 -- Successful propagation.
- IFLAG = 1 -- System not inverted or already correlated.
- IFLAG = 2 -- All covariance elements not within profile.

INTEGER scalar variable

LOGICAL FUNCTION CORR(A,NX)

FUNCTION: Compute the correlation matrix from the inverse of the normal equations.

CORR Success of the computation. Output, LOGICAL.
TRUE -- Successful correlation computation.
FALSE -- Attempt to compute correlations for system not yet inverted or already correlated.

A Elements of the normal equations.
REAL array, length of NX(IORDR+1)+IORDR

NX Index vector describing profile. Input.
INTEGER array, length of NXLENG

LOGICAL FUNCTION IN(I,J,NX,INDEX)

FUNCTION: Is the I,J element within the profile? Output, LOGICAL.
TRUE -- Element within profile, index placed in INDEX.
FALSE -- Element not within profile.

I Internal sequence # of row of the element to be inspected. Input.
INTEGER scalar variable

J Internal sequence # of col. of the element to be inspected. Input.
INTEGER scalar variable.

NX Index vector describing profile. Input.
INTEGER array, length of NXLENG

INDEX Index into the normal equation array. Output.
INTEGER scalar variable.

Note: If the internal sequence for unknowns IROW and JCOL are not available they may be gotten by using the array elements

```
I = NX(2 * IORDR + 1 + IROW)
J = NX(2 * IORDR + 1 + JCOL)
```