UNITED STATES DEPARTMENT OF THE INTERIOR
GEOLOGICAL SURVEY

INVERSE, a FORTRAN Program to Solve for Two-Dimensional Slip On a Vertical Strike Slip Fault by Inversion of Line-Length Data

by

R.A. Harris and P. Segall[1]

Open-File Report 86-451

[1]Menlo Park, California

1986

TABLE OF CONTENTS

ILLUSTRATIONS

INTRODUCTION

INVERSE.FOR is a FORTRAN 77 program which runs on the VAX/VMS 785 computer. Its creation was motivated by the desire to use line-length measurements made at the earth's surface for information about two-dimensional slip at depth on the San Andreas fault near Parkfield, California. In the Parkfield area, a transition occurs as the surface expression of the San Andreas fault progresses from a creeping zone in the northwest to a locked zone in the southeast. We have used the inverse program to invert line-length measurements made in the Parkfield area for slip on the San Andreas fault. The results have included the imaging of a locked patch at depth, located beneath the transition zone. (Harris and Segall, 1985, Segall and Harris, 1986).

The program is designed to perform a two-dimensional inversion of trilateration line-length data or strain data to solve for strike-slip motion on a vertical strike-slip fault. It also solves for a component of strain perpendicular to the fault plane. A few parts of the program, such as the subroutine CHINSSL, have been borrowed from Will Prescott's program MAIN29.FOR. Interactive input routines and the subroutine which sends the slip model to a file where it can be plotted were provided by Bob Simpson.

There are five subroutines in the program. They are:

1) CHINSSL, which calculates the Green's functions using Chinnery's equations (Chinnery, 1961) for surface displacement caused by a vertical rectangular dislocation in an elastic half-space.

2) CRD, which translates and rotates coordinates to a fault-centered system.

3) FMODEL, which reads the fault geometry information and sets up a fault grid representing the fault plane.

4) MODELWT, which calculates a weighting matrix for the model.

5) SENSIT, which does a singular-value-decomposition of the data kernel matrix and computes a number of paramenters for each run. Included among these are the model and data resolution and covariance, the estimated model, data misfits, and the geodetically determined seismic moment. One may loop through SENSIT using a varying number of singular values from the decomposition. This permits the user to choose the optimal variance and resolution of the model.

Operation of the program requires the input of station coordinates, line-length or strain data with standard deviations, and geometry of the fault grid. One may also input constraints on the slip to be allowed in the model. Output consists of all input values, the calculated model, data misfits, and moment.

This report begins with a mathematical explanation of the program. Because the data are treated as a linear function of the model, the matrix equations are relatively simple. The math section shows how the model constraints are entered into the data kernel matrix and the data vector. It

also explains the model and data-weighting, in addition to computations for the fit of the model.

The portion of the report entitled INPUT consists of a line-by-line description of a sample input file, 22KSCC.DAT. Each element required by the program is defined. The section entitled OUTPUT, gives a list of the information shown in the sample output file, 22KSCC.OUT. Brief instructions are also provided to show the user how to run this FORTRAN program on the VAX/VMS 785. The last section contains a printout of the FORTRAN program INVERSE.FOR, the sample input file, 22KSCC.DAT, and the sample output file, 22KSCC.OUT.

MATHEMATICAL METHOD

The object of the FORTRAN program INVERSE.FOR is to solve the discretized linear matrix problem

$$Y = [A] \times X \tag{1}$$

for the unknown (mx1) model vector X, given the (nx1) data vector, Y, and the (nxm) Green's function matrix [A]. Treatment of this basic problem is found in many linear algebra texts (e.g. Strang, 1976, Lanczos, 1961). We assume that measurements at the earth's surface, Y, may be explained by some distribution of strike-slip motion on the fault plus a component of normal strain, which together form elements of the model vector, X. The "formal" inverse solution, in the discrete form, is

$$X_{est} = [A]^{-1} \times Y \tag{2}$$

where $[A]^{-1}$ is the inverse of [A], and X_{est} is an estimate of the solution.

In order to solve the inverse problem, (2), we need to specify the data kernel, [A], which relates behaviour on the fault to behaviour on the earth's surface. We begin by treating the fault as a 2-dimensional fault plane consisting of discrete dislocations in a homogeneous, isotropic, elastic, half-space. (Chinnery, 1961). The centers of the discrete dislocations are arranged in a rectangular grid pattern which we call the fault grid. Individual rectangular elements in this grid are referred to as blocks. Slip is constant in each block. The bottom of the fault grid is referred to as the transition depth. Below this depth the slip rate is assumed to be constant, both in space and time. To the left of the fault grid, down to the transition depth, the slip rate is also assumed constant. This simulates a freely creeping zone. To the right, the slip is assumed to be zero, simulating a completely locked portion of the fault. (See Figure 1.) Only strike-slip motion on the fault plane and a component of normal strain are modelled.

If there is auxiliary information available, such as near fault surface slip data, which we desire to strongly influence the model in certain blocks, we may accomplish this by adding elements to the data vector Y and to the data kernel, [A]. For each of the nff (number of constrained) model blocks, one row with an appropriate distribution of 1's and 0's is added to the data kernel matrix, after the first n rows of Chinnery functions. The corresponding slip value for each constrained block is added as an element to the data vector. Y then becomes an (n+nff x 1) vector of displacement measurements and [A] becomes an (n+nff x m) dimensioned data kernel.

In order to solve equation (2), we use the generalized or, Lanczos inverse, $[A]^L$ to operate on the data vector, y (Lanczos, 1961). For our case, which is a mixed-determined problem, the Lanczos inverse simultaneously minimizes the length of the model vector, X^2 and the length of the residual vector $|\varepsilon|^2$ where

$$|\varepsilon|^2 = (Y - [A] \times X)^2 \tag{3}$$

The mixed-determined problem assumes that for some model elements there is more than one data element resolving that area while for other model elements there are no data elements which are capable of resolving the slip distribution. In the overdetermined regions of the model, a least-squares fit to the data is used. Simultaneously, the underdetermined regions of the model are solved for with the help of some a-priori assumption, such as minimum model length, or smoothness, or closeness of fit to a prior model.

An efficient way of obtaining the Lanczos inverse of the data kernel, and at the same time neatly splitting the data kernel into underdetermined and overdetermined parts is to use the singular value decomposition approach (s.v.d.). Any (n x m) matrix, [A] may be decomposed into 3 matrices [U], an (nxn) square matrix of eigenvectors which span the data space of [A], an (nxm) matrix, [Λ], consisting of the singular values of [A] on the diagonal with zeroes elsewhere, and an (mxm) square matrix V-transpose, $[V]^t$ containing the eigenvectors spanning the model space.

$$[A] = [U] \times [\Lambda] \times [V]^t \tag{4}$$

This is referred to as the singular value decomposition (s.v.d.) of the matrix [A].

The singular value matrix, [Λ], may be split into two parts, a (pxp) part with non-zero singular values on the diagonal, $[\Lambda]_p$ and an (n-p x m-p) part with zero singular values, $[\Lambda]_0$. Correspondingly, the data eigenvector matrix splits into $[U]_p$ and $[U]_0$ and the model eigenvector matrix $[V]_p$ and $[V]_0$. This split is important because it tells us what we can and cannot determine about the model from the data. All of the information to be derived from the matrix [A] is contained in the three matrices $[U]_p$, $[\Lambda]_p$ and $[V]_p$. The other three matrices, $[U]_0$, $[\Lambda]_0$ and $[V]_0$ are orthogonal to the p-space, and lie in the null space of the data kernel.

An inverse, so-called the Lanczos or generalized inverse, $[A]^L$ may be formed

$$[A]^L = [V] \times [\Lambda]^{-1} \times [U]^t \tag{5}$$

where $[\Lambda]^{-1}$ is the inverse matrix of [Λ] and $[U]^t$ is the transpose matrix of [U]. One can solve the inverse problem for the estimated model parameters, X_{est},

$$X_{est} = [A]^{-L} \times Y \tag{6}$$

$$= [V] \times [\Lambda]^{-1} \times [U]^t \times [Y] \tag{7}$$

As discussed above, [A] can also be constructed as

$$[A] = [U]_p \times [\Lambda]_p \times [V]_p^t, \tag{8}$$

so we can also form

$$X_{est} = [V]_p \times [\Lambda]_p^{-1} \times [U]_p^T \times Y \tag{9}$$

As mentioned previously, the Lanczos inverse simultaneously minimizes the data residual and the length of the model vector. Such solutions are called "minimum-length" solutions. Another assumption may be made about the nature of the "underdetermined" regions of the model; one can require that the solution across such regions is as smooth as possible. This is a desired characteristic, as it ties some of the more poorly determined fault-grid blocks to the better known boundary constraints, such as the shallow slip-rates and the long term deep slip-rate.

In the smooth case what we minimize is the length of the second derivative of model parameters, rather than the model parameters themselves. This leads us to minimize

$$\left| [T_m] \times X \right|^2 \tag{10}$$

where T_m is an m x m matrix of Laplacian operators. (Menke, 1984). By using this model weighting matrix, T_m, the model estimate resulting from the Lanczos inverse becomes a smooth model. The "overdetermined" parts of the model are still best fits to the data while the "underdetermined" parts now smoothly blend into the better determined regions.

The new equations for X' which is the new weighted version of the model vector, X and for Y' which is the new weighted version of the data vector, Y are:

$$X' = [T_m] \times X \tag{11}$$

$$Y' = [T_d] \times Y. \tag{12}$$

T_m is the smoothing matrix for the model and T_d is a data weighting matrix in which each data element is weighted by the inverse of the standard deviation of that element.

Equation 1 can now be written as

$$Y' = [A'] \times [X'] \tag{13}$$

with

$$[A]' = [T_d] \times [A] \times [T_m]^{-1} \tag{14}$$

We may perform an s.v.d. on A' to get U', $[\Lambda]'$, and $[V]^{t'}$. A smoothed model estimate is then obtained by using a T_m consisting of Laplacian operators in 2-d:

$$\begin{aligned} X_{est} &= [T_m]^{-1} \times [A]'^{L} \times [T_d] \times Y \\ &= [T_m]^{-1} \times [V]'_p \times [\Lambda]_p^{-1,} \times [U]_p'^{t} \times [T_d] \times Y \end{aligned} \tag{15}$$

In addition to the model estimate, by using the matrices from the singular value decomposition of the data kernel, it is easy to write equations

for the model resolution, [R], the model covariance, [C], and the data resolution, [D]. The model resolution, which maps a true model into the model estimate,is given by

$$[R] = [T_m]^{-1} \times [V]'_p \times [V]'^{t}_p \times [T_m] \tag{16}$$

The data resolution, which maps the observed data into the calculated data is

$$[D] = [T_d]^{-1} \times [U]'_p \times [U]^{t}_p \times [T_d] \tag{17}$$

and the model covariance, which provides error bars on the model is

$$[C] = [T_m]^{-1} \times [V]'_p \times [\Lambda]^{-2}{}' \times [V]^{t}_p \times [T_m]^{-1t} \tag{18}$$

The model covariance may be used to obtain a root-mean-square standard deviation of the model elements.

In order to choose the correct number of singular values to use in the inversion, we also calculate a parameter called chekmod. This parameter, which is a rotated form of the model,

$$\text{chekmod} = [\Lambda]^{-1}{}' \times [U]'^{t} \times Y' \tag{19}$$

$$= [\Lambda]^{-1}{}' \times Y'' \tag{20}$$

where

$$Y'' = [U]'^{t} \times Y' \tag{21}$$

can be compared with its expected variance, $\sigma(Y'') \times [\Lambda]'^{-2}$, for each singular value, to determine which elements exceed their expected uncertainties. Because the matrix [cov Y"] is just the identity matrix, we end up comparing chekmod with the inverse of the smallest singular value. When chekmod is greater than this value, it has exceeded its expected uncertainty.

The geodetically determined seismic moment is the most robust quantity determined from the inversion of line-length data. It can be calculated after the model vector has been obtained, by summing the slip over all of the fault grid blocks, multiplied by the area of a block, and the shear modulus, μ

$$M_o = \mu \times \text{area} \times \sum_{i=2,m-2} X_{est}(i) \tag{22}$$

The elements which are not included in the sum are the deep slip, the slip in the left-hand block, and the normal strain component.

Two statistical parameters which are useful to calculate are the misfit of the data to the model, χ^2, and the root mean square standard deviation of the model elements. The misfit, or χ^2 is given by

$$\chi^2 = \sum_{\text{over all lines}} ((o(i)-c(i))/\sigma(i))^2 \qquad (23)$$

where o(i) is the observed data measurement, c(i) is the calculated data measurement, and $\sigma(i)$ is the standard deviation of that line. The root mean square standard deviation of the model parameters, rmsstdev is defined as

$$\text{rmsstdev} = \sqrt{(\sum_i C(i,i))/\text{nvf}} \qquad (24)$$

where nvf is the number of variable faults.

PROGRAM OVERVIEW

A step-by-step summary of the program flow is presented below.

1) Read (interactively) the number of singular values to use in the inversion, the number of additional singular values to use, the type of model weighting, and whether or not to include a normal strain component.

2) Read file containing line-length or strain data.

3) Read fault model using SUBROUTINE FMODEL and set up geometry.

4) Rotate stations to fault-centered coordinate system using SUBROUTINE CRD.

5) Compute station displacements due to each fault, in fault-centered coordinates.

6) Calculate Chinnery functions with SUBROUTINE CHINSSL.

7) Obtain station displacements in NS-EW coordinate system (rotate again, using SUBROUTINE CRD).

8) Calculate green's functions, including normal component of strain, if desired.

9) Add extra lines to green's functions for the constrained grid blocks.

10) Read the fixed block slip constraints and their standard deviations.

11) Form data-weighting matrix, and its calculate its inverse.

12) Form model-weighting matrix, with call to SUBROUTINE MODELWT.

13) Form inverse of model-weighting matrix.

14) Call SUBROUTINE SENSIT to determine the singular value decomposition of the data kernel [A], and to obtain the model estimate, the data-fit to the model, the root-mean-square standard deviation of the model elements, the moment, etc. Loop through SENSIT for each requested additional singular value, and recalculate the above parameters.

INPUT DESCRIPTION

Interactively, the program INVERSE.FOR will ask for information about the number of singular values to use in the inversion and the type of model weighting desired by the user. The input file for the FORTRAN program requires (x,y) station coordinates in meters, line-length or strain data for the corresponding lines, (described in detail in King et. al., unpub. data, 1984), and a fault model. A sample input file, given in Appendix A, is 22KSCC.DAT. Following is a line-by-line description of the input parameters. The words between asterisks are the names of variables used in the program. Below each term is an explanation of what is to be input.

Interactively, the program asks for:

NSTART
The number of singular values to start with

NUMRUN
The number of times to perform the singular value loop after NSTART. An iterative loop in the Subroutine SENSIT starts with NSTART singular values in the singular value decomposition and runs NUMRUN additional times through the process.

NWT
The type of model-weighting desired. NWT=1 implies that no model-weighting is to be used, and the solution will be the minimum-length solution. NWT=2 implies that a model-weighting matrix which results in a smooth model will be used.

NSTRAIN
The model parameter involving a component of strain normal to the fault plane. NSTRAIN=1 leads to no normal strain and a resulting model of 2-d slip on the fault. NSTRAIN=2 leads to a model of 2-d slip on the fault plane plus a component of strain perpendicular to the fault plane.

The input file requires the following lines of information:
Line 1 format(i1,i4,3i5,5x,10a4)

IOPT
A single digit which indicates whether strain data or line-length data is to be used
'0' for strain cards
'1' for pairs of line-length cards

NF
The number of grid blocks (constrained and variable), including large block to left of fault grid and block beneath transition depth

NLINES
The number of trilateration lines to be used in the inversion

NSTNS
The number of stations (endpoints of the trilateration lines)

NFF
The number of grid blocks with constrained slip

DESCRI
A short (word) description of the program run

Line 2 to Line 2+NSTNS format(a8,6x,f16.0,6x,f16.0)

Each line has the following station information

STA
Station name

XX
X-coordinate in meters

YY
Y-coordinate in meters

Next NLINES lines (for strain cards)
Each line has the following line-length information:
format(a5,8x,a8,1x,a8,t44,f10.3,t54,f7.4)

NET
Network name

SN1
Station 1 endpoint name

SN2
Station 2 endpoint name

DATE
Decimal years - 1900
(Note that this is not a required input parameter and is not used in the program.)

NMEAS
Number of measurements made for this line
(Note this is not a required input parameter)

DISTAN
Line-length (meters)

STR
Rate of line-length change (meters/year)

SD
Standard deviation of the rate (meters/year)

AZI
Line azimuth measured clockwise from North in degrees, SN1 to SN2.

or, for pairs of line-length cards,
Next 2*NLINES

First card in the pair

NET

SN1

SN2

DATE

AZ

DISMARK1
Mark-to-mark distance (meters)
Note this is not used in the program.

DISTAN
Arc-distance (meters)

SD
Standard deviation of the line-length change (meters)

Second card in the pair

NET

SN1

SN2

DATE

AZ

DISMARK2
Mark-to-mark distance of the second distance measurement (meters)

DIST2
Arc-distance of the second distance measurement (meters)

SD

The following cards in the input file are used to set up the fault grid model. Fault blocks can be specified in 3 ways by defining "kode"

1 = one endpoint, azimuth, length

2 = two endpoints

3 = midpoint, azimuth, half-length

4 = grid of blocks

Each line in the input is for a different fault block or fault-grid block geometry. The fault information to be input in each line is:
format(3i4,4x,8f8.0)

KODE
A number (1,2,3,4) as defined above

NHE
The number of horizontal elements in the grid
(NHE = 0 if kode is not 4)

NVE
The number of vertical elements in the grid
(NVE = 0 if kode is not 4)

DU
The depth to the top of the fault block (or grid) in kilometers

DL
The depth to the bottom of the fault partition in kilometers

AZ
Azimuth from north, in degrees

LENGTH
Total fault length in kilometers
If kode = 3 then fault half-length

X1
Endpoint coordinate (meters)
If kode = 3 then midpoint coordinate

Y1
Endpoint coordinate (midpoint if kode = 3)

X2
"

Y2
"

After reading in the fault model parameters, the next NFF lines are the model slip constraints.
Each line consists of:1(i4,2e12.4)

INDEX
The block number which is assigned fixed slip

*CONSRAT
The constrained slip value in meters/year

CONSWT
The standard deviation of the constraint (meters/year)

The final line signals that the input is complete:

0

A comment on the system for numbering the fault blocks:
The model weighting routine is presently arranged so that blocks are numbered in the manner depicted in Figure 1. The left hand block is number 1, the top left block in the grid is 2, and so on to the last block, which is below the transition depth.

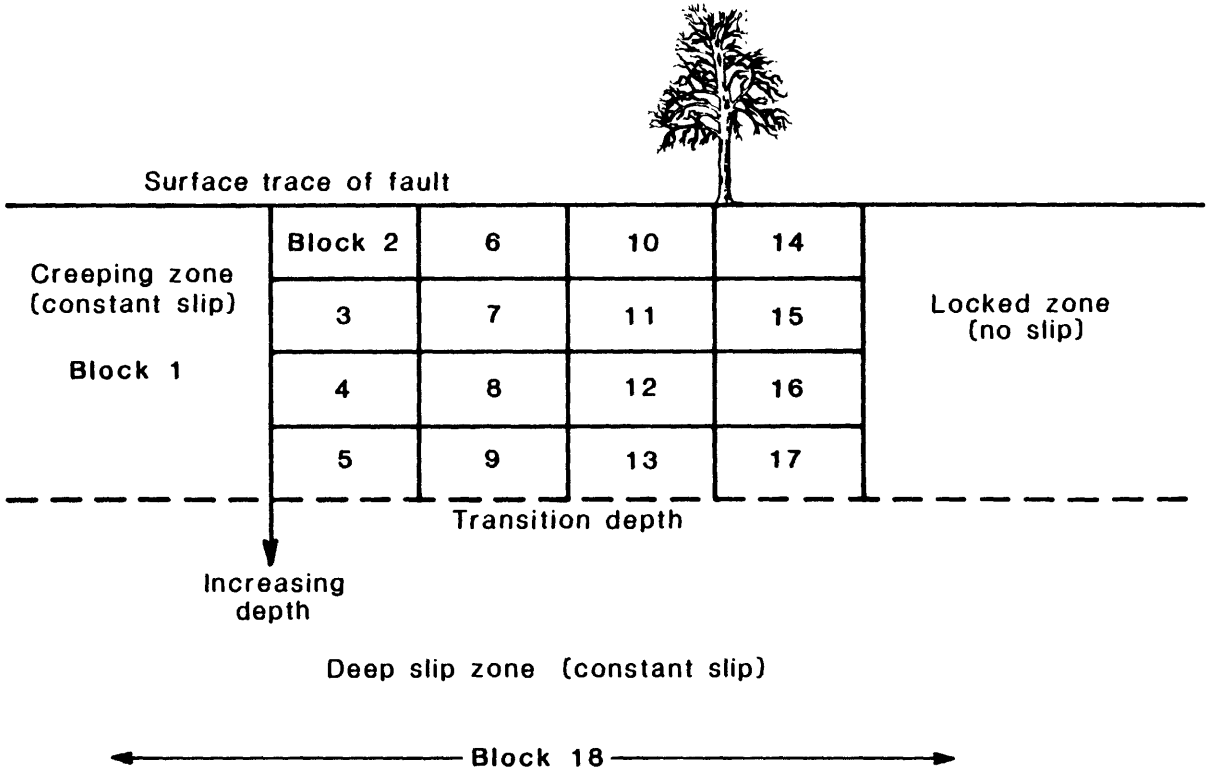

Figure 1. A cross section of the fault. To the left of the fault grid, in Block 1, the slip is constant, equal to the creep in that region. To the right of the grid, the fault is locked and there is no slip. The line running between the fault plane (consisting of Blocks 1 to 17 and the locked zone) and the deep slip zone (Block 18) is referred to as the transition depth. Below this line, slip is constant.

OUTPUT DESCRIPTION

The output file (see Appendix B for an example) reproduces the input file, then shows the calculated block coordinates and lists the assigned constraints. In detail, this includes:

- Information about the singular value decomposition of the rotated data kernel matrix, [A]'. The IMSL warning error may be ignored, since this program does not use all of the singular values.

- For each run of the singular loop,
  - The number of singular values used
  - The smallest singular value used
  - Chekmod and 1/(smallest singular value used)
  - The r.m.s. model standard deviation (m/yr)
  - The fit of the data to the model (dimensionless)
  - The moment rate of the solution (dyne-cm/yr)
  - The normal strain component of the solution (micro-strain/yr)

- For each line
  - The observed data
  - The calculated data
  - The misfit of the predicted line-length to the observed
  - The station displacements

- For each fault block
  - The estimated slip rate (m/yr)
  - The standard deviation (m/yr)

PROGRAM OPERATION; RUNNING THE PROGRAM ON THE VAX/VMS 785 COMPUTER

To copy the program to your areas type:

```
COPY [HARRIS.PARKLONG]INVERSE.FOR
```

Compile the program by typing:

```
FOR INVERSE.FOR
```

Link the program to the single precision IMSL library:

```
$ LINK INVERSE,IMSLS/LIBR
```

To run the program create the following run command file, RUNINV.COM:

```
$ ASSIGN SYS$INPUT FOR015
$ ASSIGN SYS$OUTPUT FOR016
$ ASSIGN 'P2'.DAT FOR005
$ ASSIGN 'P2'.OUT FOR006
$ ASSIGN 'P2'.STV FOR085
$ DEFINE SYS$INPUT SYS$COMMAND:
$ RUN 'P1'
$ DEASSIGN SYS$INPUT
$ DEASSIGN SYS$OUTPUT
$ DEASSIGN FOR005
$ DEASSIGN FOR006
$ DEASSIGN FOR085
```

and, run the program with the input line, where 22kscc is the name of the input file

```
@RUNINV INVERSE 22KSCC
```

Three output files will be created. The first is the output file described above. The second is a file which contains the slip elements in code and can be used for color and contour plotting. The third is a file consisting of the predicted station velocities for each model.

REFERENCES

Chinnery, M.A. (1961), "The deformation of the ground around surface faults." Bull. Seism. Soc. Am. 51, 355-372.

Harris, R., and P. Segall (1985), "Determination of the slip deficit along the Parkfield CA section of the San Andreas fault from the inversion of trilateration data." EOS Trans. AGU (abstract), 46, 985.

King, N.E., Prescott, W.H., and K.J. Wendt, Unpublished line-length record information. August 1983, revised January 1984.

Lanczos, C. (1961), Linear Differential Operators. Van Nostrand-Reinhold, Princeton, New Jersey.

Menke, W. (1984), Geophysical Data Analysis: Discrete Inverse Theory. Academic Press, Orlando, Florida.

Segall, P., and R. Harris (1986), "Slip deficit on the Parkfield, California section of the San Andreas fault as revealed by inversion of geodetic data." Science, in press.

Strang, G. (1976), Linear Algebra and its Applications. Academic Press, New York, New York.

APPENDIX A

Sample input file
22KSCC.DAT

```
0 134   45   23   14     Parkfield
airway      x =  372863.34        y =  205517.50       meters       cal zone
almond      x =  387280.44        y =  230311.04       meters       cal zone
barren      x =  376124.53        y =  219925.67       meters       cal zone
bench       x =  397009.41        y =  251596.36       meters       cal zone
blhllres    x =  352361.27        y =  209785.62       meters       cal zone
bonnie      x =  405248.78        y =  266058.53       meters       cal zone
castle      x =  398499.18        y =  273018.53       meters       cal zone
chiches2    x =  396029.12        y =  212436.47       meters       cal zone
cotton      x =  408764.94        y =  256002.59       meters       cal zone
davis       x =  356267.64        y =  195729.09       meters       cal zone
gold        x =  397333.49        y =  261032.85       meters       cal zone
hatch       x =  412537.33        y =  244618.21       meters       cal zone
hopper      x =  394082.18        y =  243719.35       meters       cal zone
kenger      x =  397814.27        y =  270235.58       meters       cal zone
mason       x =  388953.45        y =  261439.02       meters       cal zone
mid f       x =  382038.42        y =  271317.48       meters       cal zone
mine mt     x =  390064.12        y =  276596.15       meters       cal zone
mine rm2    x =  390064.39        y =  276599.84       meters       cal zone
park        x =  401440.70        y =  268212.82       meters       cal zone
red hill    x =  404862.12        y =  235801.52       meters       cal zone
shade       x =  367312.91        y =  284025.23       meters       cal zone
tess        x =  364621.55        y =  212466.02       meters       cal zone
wild        x =  386399.82        y =  253809.22       meters       cal zone
sanlu   +    airway   barren     79.487   6 14772.850 0.0006 0.0018  11.3
sanlu   +    airway   blhllres   80.613   5 20942.138-0.0007 0.0013 280.3
sanlu   +    airway   chiches2   79.495   6 24165.589 0.0005 0.0013  72.1
sanlu   +    airway   davis      80.543   5 19268.005-0.0031 0.0026 237.9
sanlu   +    airway   tess       80.603   5 10780.267-0.0008 0.0009 308.7
sanlu   +    almond   barren     79.500   6 15242.453-0.0013 0.0009 225.7
sanlu   +    almond   bench      79.128   7 23402.636-0.0037 0.0011  23.2
sanlu   +    almond   chiches2   79.499   6 19901.080-0.0009 0.0011 152.5
sanlu   +    almond   red hill   81.929  23 18418.864 0.0012 0.0016  71.3
sanlu   +    barren   chiches2   79.489   6 21268.287 0.0012 0.0011 109.3
sanlu   +    barren   tess       79.820   5 13710.150-0.0014 0.0009 235.5
sanlu   +    bench    bonnie     77.421   8 16643.001-0.0052 0.0006  28.3
```

```
sanlu   +    bench    cotton     76.521  17 12553.207 0.0031 0.0004  68.2
sanlu   +    bench    gold       82.364   4  9441.189-0.0058 0.0027   0.6
sanlu   +    bench    hatch      78.857  10 17023.263 0.0059 0.0008 113.0
sanlu   +red bench    hopper     77.753   6  8402.882-0.0021 0.0009 199.0
sanlu   +    bench    kenger     76.522  17 18654.694-0.0076 0.0011   1.1
sanlu   +    bench    mason      76.744  14 12718.058 0.0017 0.0006 319.3
sanlu   +    bench    red hill   78.641  10 17638.621-0.0006 0.0008 152.3
sanlu   +    bench    wild       78.446   5 10837.091 0.0014 0.0006 280.4
sanlu   +    blhllres davis      80.371   4 14589.692-0.0011 0.0013 162.9
sanlu   +    blhllres tess       80.614   5 12550.087-0.0016 0.0016  76.1
sanlu   +    bonnie   cotton     78.134   7 10651.863-0.0031 0.0010 159.5
sanlu   +    bonnie   gold       78.462   5  9374.990-0.0007 0.0005 236.3
sanlu   +    bonnie   kenger     78.134   7  8526.553-0.0007 0.0005 298.0
sanlu   +    bonnie   mason      77.418   8 16935.680 0.0053 0.0006 252.9
sanlu   +    castle   shade      77.549  10 32455.025 0.0172 0.0013 288.1
sanlu   +    chiches2 red hill   79.489   6 24979.096-0.0036 0.0013  19.4
sanlu   +    cotton   gold       78.703   6 12488.113 0.0004 0.0011 292.4
sanlu   +    cotton   kenger     78.301   9 17956.211-0.0021 0.0013 321.1
sanlu   +    cotton   mason      76.040  17 20541.974 0.0108 0.0009 284.0
sanlu   +    cotton   red hill   80.384  27 20573.730-0.0073 0.0010 189.7
sanlu   +    davis    tess       83.315   3 18706.194 0.0031 0.0035  24.9
sanlu   +    gold     kenger     78.703   6  9214.194-0.0026 0.0014   1.7
sanlu   +    hatch    red hill   76.206  18 11688.836-0.0024 0.0004 219.8
sanlu   +red hopper   mason      78.475   5 18445.590 0.0024 0.0008 342.5
sanlu   +red hopper   red hill   80.940   4 13375.042-0.0004 0.0024 124.9
sanlu   +red hopper   wild       78.466   5 12680.805 0.0022 0.0008 321.3
sanlu   +    kenger   mason      76.573  17 12484.315-0.0022 0.0005 223.9
sanlu   +    kenger   mid f      80.474   8 15682.504 0.0076 0.0010 272.6
sanlu   +    mason    mine rm2   77.274  15 15196.422-0.0099 0.0007   2.8
sanlu   +    mason    wild       78.471   5  8045.069-0.0023 0.0006 197.1
sanlu   +    mid f    mine mt    79.127  10  9482.293 0.0023 0.0011  55.3
sanlu   +    mine mt  shade      77.811  21 23304.541 0.0186 0.0007 286.6
sanlu   +    park     red hill   78.153  27 32504.094-0.0086 0.0007 172.7
  1   0   0          0.     22.   319.3    100. 379999. 280068. 000000. 000000.
  4  12  11          0.     22.   139.3     36. 379999. 280068. 000000. 000000.
  3   0   0         22.   1000.   139.3   1000. 379999. 280068. 000000. 000000.
  1    2.51e-02     1.0e-05
  2     2.4e-02     1.0e-05
 13     2.2e-02     1.0e-05
 24     2.2e-02     1.0e-05
 35     1.8e-02     1.0e-05
 46     1.6e-02     1.0e-05
 57     1.4e-02     1.0e-05
 68     1.2e-02     1.0e-05
 79     1.1e-02     1.0e-05
 90     0.9e-02     1.0e-05
101     0.8e-02     1.0e-05
112     0.6e-02     1.0e-05
123     0.4e-02     1.0e-05
134     3.3e-02     1.0e-05
  0
```

APPENDIX B

Sample output file
22KSCC.OUT

```
nwt (1=no-modelwting, 2=smooth) =   2
iopt (0=strcards, 1=llcards)... =   0
number of faults.............. = 134
number of lines............... =  45
number of stations............ =  23
number of fixed faults........ =  14
Parkfield
```

station	xcoord(m)	ycoord(m)
airway	372863.344	205517.500
almond	387280.438	230311.047
barren	376124.531	219925.672
bench	397009.406	251596.359
blhllres	352361.281	209785.625
bonnie	405248.781	266058.531
castle	398499.188	273018.531
chiches2	396029.125	212436.469
cotton	408764.938	256002.594
davis	356267.625	195729.094
gold	397333.500	261032.844
hatch	412537.344	244618.203
hopper	394082.188	243719.344
kenger	397814.281	270235.594
mason	388953.438	261439.016
mid f	382038.406	271317.469
mine mt	390064.125	276596.156
mine rm2	390064.375	276599.844
park	401440.688	268212.813
red hill	404862.125	235801.516
shade	367312.906	284025.219
tess	364621.563	212466.016
wild	386399.813	253809.219

net	stn1	stn2	length(m)	dl/dt(m/yr)	stdev	az
sanlu	airway	barren	14772.850	0.0006	0.0018	11.3
sanlu	airway	blhllres	20942.138	-0.0007	0.0013	280.3
sanlu	airway	chiches2	24165.589	0.0005	0.0013	72.1
sanlu	airway	davis	19268.005	-0.0031	0.0026	237.9
sanlu	airway	tess	10780.267	-0.0008	0.0009	308.7
sanlu	almond	barren	15242.453	-0.0013	0.0009	225.7
sanlu	almond	bench	23402.636	-0.0037	0.0011	23.2
sanlu	almond	chiches2	19901.080	-0.0009	0.0011	152.5
sanlu	almond	red hill	18418.864	0.0012	0.0016	71.3
sanlu	barren	chiches2	21268.287	0.0012	0.0011	109.3
sanlu	barren	tess	13710.150	-0.0014	0.0009	235.5
sanlu	bench	bonnie	16643.001	-0.0052	0.0006	28.3
sanlu	bench	cotton	12553.207	0.0031	0.0004	68.2
sanlu	bench	gold	9441.189	-0.0058	0.0027	0.6
sanlu	bench	hatch	17023.263	0.0059	0.0008	113.0
sanlu	bench	hopper	8402.882	-0.0021	0.0009	199.0
sanlu	bench	kenger	18654.694	-0.0076	0.0011	1.1
sanlu	bench	mason	12718.058	0.0017	0.0006	319.3
sanlu	bench	red hill	17638.621	-0.0006	0.0008	152.3
sanlu	bench	wild	10837.091	0.0014	0.0006	280.4
sanlu	blhllres	davis	14589.692	-0.0011	0.0013	162.9
sanlu	blhllres	tess	12550.087	-0.0016	0.0016	76.1
sanlu	bonnie	cotton	10651.863	-0.0031	0.0010	159.5
sanlu	bonnie	gold	9374.990	-0.0007	0.0005	236.3
sanlu	bonnie	kenger	8526.553	-0.0007	0.0005	298.0
sanlu	bonnie	mason	16935.680	0.0053	0.0006	252.9
sanlu	castle	shade	32455.025	0.0172	0.0013	288.1
sanlu	chiches2	red hill	24979.096	-0.0036	0.0013	19.4
sanlu	cotton	gold	12488.113	0.0004	0.0011	292.4
sanlu	cotton	kenger	17956.211	-0.0021	0.0013	321.1
sanlu	cotton	mason	20541.974	0.0108	0.0009	284.0
sanlu	cotton	red hill	20573.730	-0.0073	0.0010	189.7
sanlu	davis	tess	18706.194	0.0031	0.0035	24.9
sanlu	gold	kenger	9214.194	-0.0026	0.0014	1.7
sanlu	hatch	red hill	11688.836	-0.0024	0.0004	219.8
sanlu	hopper	mason	18445.590	0.0024	0.0008	342.5
sanlu	hopper	red hill	13375.042	-0.0004	0.0024	124.9
sanlu	hopper	wild	12680.805	0.0022	0.0008	321.3
sanlu	kenger	mason	12484.315	-0.0022	0.0005	223.9
sanlu	kenger	mid f	15682.504	0.0076	0.0010	272.6
sanlu	mason	mine rm2	15196.422	-0.0099	0.0007	2.8
sanlu	mason	wild	8045.069	-0.0023	0.0006	197.1
sanlu	mid f	mine mt	9482.293	0.0023	0.0011	55.3
sanlu	mine mt	shade	23304.541	0.0186	0.0007	286.6
sanlu	park	red hill	32504.094	-0.0086	0.0007	172.7

INPUT MODEL GRIDDING

1	0	0	0.	22.	319.	100.	379999.	280068.	0.	0.

INPUT MODEL GRIDDING

4	12	11	0.	22.	139.	36.	379999.	280068.	0.	0.

# of horizontal elements in grid = 12
# of vertical elements in grid = 11

INPUT MODEL GRIDDING
3 0 0 22. 1000. 139. 1000. 379999. 280068. 0. 0.

MODEL PARAMETERS

du(km)	dl(km)	az	hfln(km)	xl(m)	yl(m)
0.0	22.0	319.3	50.0	347394.	317975.
0.0	2.0	139.3	1.5	380977.	278931.
2.0	4.0	139.3	1.5	380977.	278931.
4.0	6.0	139.3	1.5	380977.	278931.
6.0	8.0	139.3	1.5	380977.	278931.
8.0	10.0	139.3	1.5	380977.	278931.
10.0	12.0	139.3	1.5	380977.	278931.
12.0	14.0	139.3	1.5	380977.	278931.
14.0	16.0	139.3	1.5	380977.	278931.
16.0	18.0	139.3	1.5	380977.	278931.
18.0	20.0	139.3	1.5	380977.	278931.
20.0	22.0	139.3	1.5	380977.	278931.
0.0	2.0	139.3	1.5	382933.	276656.
2.0	4.0	139.3	1.5	382933.	276656.
4.0	6.0	139.3	1.5	382933.	276656.
6.0	8.0	139.3	1.5	382933.	276656.
8.0	10.0	139.3	1.5	382933.	276656.
10.0	12.0	139.3	1.5	382933.	276656.
12.0	14.0	139.3	1.5	382933.	276656.
14.0	16.0	139.3	1.5	382933.	276656.
16.0	18.0	139.3	1.5	382933.	276656.
18.0	20.0	139.3	1.5	382933.	276656.
20.0	22.0	139.3	1.5	382933.	276656.
0.0	2.0	139.3	1.5	384890.	274382.
2.0	4.0	139.3	1.5	384890.	274382.
4.0	6.0	139.3	1.5	384890.	274382.
6.0	8.0	139.3	1.5	384890.	274382.
8.0	10.0	139.3	1.5	384890.	274382.
10.0	12.0	139.3	1.5	384890.	274382.
12.0	14.0	139.3	1.5	384890.	274382.
14.0	16.0	139.3	1.5	384890.	274382.
16.0	18.0	139.3	1.5	384890.	274382.
18.0	20.0	139.3	1.5	384890.	274382.
20.0	22.0	139.3	1.5	384890.	274382.
0.0	2.0	139.3	1.5	386846.	272108.
2.0	4.0	139.3	1.5	386846.	272108.
4.0	6.0	139.3	1.5	386846.	272108.
6.0	8.0	139.3	1.5	386846.	272108.
8.0	10.0	139.3	1.5	386846.	272108.
10.0	12.0	139.3	1.5	386846.	272108.
12.0	14.0	139.3	1.5	386846.	272108.
14.0	16.0	139.3	1.5	386846.	272108.
16.0	18.0	139.3	1.5	386846.	272108.
18.0	20.0	139.3	1.5	386846.	272108.
20.0	22.0	139.3	1.5	386846.	272108.

(part intentionally omitted)

0.0	2.0	139.3	1.5	398584.	258461.
2.0	4.0	139.3	1.5	398584.	258461.
4.0	6.0	139.3	1.5	398584.	258461.
6.0	8.0	139.3	1.5	398584.	258461.
8.0	10.0	139.3	1.5	398584.	258461.
10.0	12.0	139.3	1.5	398584.	258461.
12.0	14.0	139.3	1.5	398584.	258461.
14.0	16.0	139.3	1.5	398584.	258461.
16.0	18.0	139.3	1.5	398584.	258461.
18.0	20.0	139.3	1.5	398584.	258461.
20.0	22.0	139.3	1.5	398584.	258461.
0.0	2.0	139.3	1.5	400540.	256187.
2.0	4.0	139.3	1.5	400540.	256187.
4.0	6.0	139.3	1.5	400540.	256187.
6.0	8.0	139.3	1.5	400540.	256187.
8.0	10.0	139.3	1.5	400540.	256187.
10.0	12.0	139.3	1.5	400540.	256187.
12.0	14.0	139.3	1.5	400540.	256187.
14.0	16.0	139.3	1.5	400540.	256187.
16.0	18.0	139.3	1.5	400540.	256187.
18.0	20.0	139.3	1.5	400540.	256187.
20.0	22.0	139.3	1.5	400540.	256187.
0.0	2.0	139.3	1.5	402496.	253912.
2.0	4.0	139.3	1.5	402496.	253912.
4.0	6.0	139.3	1.5	402496.	253912.
6.0	8.0	139.3	1.5	402496.	253912.
8.0	10.0	139.3	1.5	402496.	253912.
10.0	12.0	139.3	1.5	402496.	253912.
12.0	14.0	139.3	1.5	402496.	253912.
14.0	16.0	139.3	1.5	402496.	253912.
16.0	18.0	139.3	1.5	402496.	253912.
18.0	20.0	139.3	1.5	402496.	253912.
20.0	22.0	139.3	1.5	402496.	253912.
22.0	1000.0	139.3	1000.0	379999.	280068.

CONSTRAINTS: RATE AND STANDARD DEVIATION (M/YR)

for block	1	rate = 0.2510E-01	+/- 0.1000E-04
for block	2	rate = 0.2400E-01	+/- 0.1000E-04
for block	13	rate = 0.2200E-01	+/- 0.1000E-04
for block	24	rate = 0.2200E-01	+/- 0.1000E-04
for block	35	rate = 0.1800E-01	+/- 0.1000E-04
for block	46	rate = 0.1600E-01	+/- 0.1000E-04
for block	57	rate = 0.1400E-01	+/- 0.1000E-04
for block	68	rate = 0.1200E-01	+/- 0.1000E-04
for block	79	rate = 0.1100E-01	+/- 0.1000E-04
for block	90	rate = 0.9000E-02	+/- 0.1000E-04
for block	101	rate = 0.8000E-02	+/- 0.1000E-04
for block	112	rate = 0.6000E-02	+/- 0.1000E-04
for block	123	rate = 0.4000E-02	+/- 0.1000E-04
for block	134	rate = 0.3300E-01	+/- 0.1000E-04

```
SINGULAR VALUES OF THE ROTATED GREENS FUNCS

singvals of aprime

       0.456486E+06   0.196466E+06   0.120834E+06   0.973487E+05
0.815308E+05   0.580905E+05   0.439687E+05   0.351016E+05
       0.292690E+05   0.253086E+05   0.225827E+05   0.207249E+05
0.195190E+05   0.188393E+05   0.237243E+03   0.105676E+03
       0.792925E+02   0.505534E+02   0.364254E+02   0.287318E+02
0.154786E+02   0.127340E+02   0.114169E+02   0.732972E+01
       0.418895E+01   0.309239E+01   0.291581E+01   0.201120E+01
0.121997E+01   0.995859E+00   0.640766E+00   0.506019E+00
       0.455233E+00   0.303286E+00   0.227816E+00   0.176209E+00
0.677878E-01   0.544228E-01   0.498610E-01   0.274274E-01
       0.257065E-01   0.135312E-01   0.959325E-02   0.386022E-02
0.183881E-02   0.140398E-02   0.799351E-03   0.794726E-03
       0.712292E-03   0.493906E-03   0.352511E-03   0.299378E-03
0.916494E-04   0.714557E-04   0.456862E-04   0.913640E-05
       0.835066E-05   0.632742E-05   0.329123E-05   0.000000E+00
0.000000E+00   0.000000E+00   0.000000E+00   0.000000E+00
       0.000000E+00   0.000000E+00   0.000000E+00   0.000000E+00
0.000000E+00   0.000000E+00   0.000000E+00   0.000000E+00
       0.000000E+00   0.000000E+00   0.000000E+00   0.000000E+00
0.000000E+00   0.000000E+00   0.000000E+00   0.000000E+00
       0.000000E+00   0.000000E+00   0.000000E+00   0.000000E+00
0.000000E+00   0.000000E+00   0.000000E+00   0.000000E+00
       0.000000E+00   0.000000E+00   0.000000E+00   0.000000E+00
0.000000E+00   0.000000E+00   0.000000E+00   0.000000E+00
       0.000000E+00   0.000000E+00   0.000000E+00   0.000000E+00
0.000000E+00   0.000000E+00   0.000000E+00   0.000000E+00
       0.000000E+00   0.000000E+00   0.000000E+00   0.000000E+00
0.000000E+00   0.000000E+00   0.000000E+00   0.000000E+00
       0.000000E+00   0.000000E+00   0.000000E+00   0.000000E+00
0.000000E+00   0.000000E+00   0.000000E+00   0.000000E+00
       0.000000E+00   0.000000E+00   0.000000E+00   0.000000E+00
0.000000E+00   0.000000E+00   0.000000E+00   0.000000E+00
       0.000000E+00   0.000000E+00   0.000000E+00   0.000000E+00
0.000000E+00   0.000000E+00   0.000000E+00


using   17singvals, smallest=  0.7929E+02
for 17 singvals
          rms model st.dev=  0.2281E-02
        the strain component is -0.6309E-07
          sum of, the misfits squared =   0.7561E+02
          moment rate =  0.2640E+25   +/-    0.2251E+24
chekmod =   0.5630E-01  1/smallest sinval =   0.1261E-01
```

		obs data	calc data	(obs-calc)/sigma
airway	barren	0.6000E-03	-0.6430E-03	0.6906E+00
airway	blhllres	-0.7000E-03	-0.1526E-03	-0.4211E+00
airway	chiches2	0.5000E-03	-0.5774E-03	0.8288E+00
airway	davis	-0.3100E-02	-0.9558E-03	-0.8247E+00
airway	tess	-0.8000E-03	0.1319E-03	-0.1035E+01
almond	barren	-0.1300E-02	-0.7585E-03	-0.6016E+00
almond	bench	-0.3700E-02	-0.2967E-02	-0.6663E+00
almond	chiches2	-0.9000E-03	-0.1235E-03	-0.7059E+00
almond	red hill	0.1200E-02	0.6226E-03	0.3609E+00
barren	chiches2	0.1200E-02	0.7165E-03	0.4395E+00
barren	tess	-0.1400E-02	-0.5906E-03	-0.8993E+00
bench	bonnie	-0.5200E-02	-0.5150E-02	-0.8288E-01
bench	cotton	0.3100E-02	0.2359E-02	0.1853E+01
bench	gold	-0.5800E-02	-0.7432E-02	0.6044E+00
bench	hatch	0.5900E-02	0.4369E-02	0.1914E+01
bench	hopper	-0.2100E-02	-0.1472E-02	-0.6974E+00
bench	kenger	-0.7600E-02	-0.9433E-02	0.1666E+01
bench	mason	0.1700E-02	0.1806E-02	-0.1760E+00
bench	red hill	-0.6000E-03	-0.4729E-03	-0.1588E+00
bench	wild	0.1400E-02	0.2612E-02	-0.2020E+01
blhllres	davis	-0.1100E-02	-0.9132E-04	-0.7759E+00
blhllres	tess	-0.1600E-02	-0.3971E-03	-0.7518E+00
bonnie	cotton	-0.3100E-02	-0.2369E-02	-0.7312E+00
bonnie	gold	-0.7000E-03	-0.6091E-04	-0.1278E+01
bonnie	kenger	-0.7000E-03	-0.3613E-03	-0.6775E+00
bonnie	mason	0.5300E-02	0.4806E-02	0.8235E+00
castle	shade	0.1720E-01	0.1704E-01	0.1238E+00
chiches2	red hill	-0.3600E-02	-0.3455E-02	-0.1118E+00
cotton	gold	0.4000E-03	0.5177E-05	0.3589E+00
cotton	kenger	-0.2100E-02	-0.2675E-02	0.4424E+00
cotton	mason	0.1080E-01	0.8904E-02	0.2107E+01
cotton	red hill	-0.7300E-02	-0.6172E-02	-0.1128E+01
davis	tess	0.3100E-02	-0.8577E-03	0.1131E+01
gold	kenger	-0.2600E-02	-0.2036E-02	-0.4027E+00
hatch	red hill	-0.2400E-02	-0.1795E-02	-0.1513E+01
hopper	mason	0.2400E-02	-0.9927E-03	0.4241E+01
hopper	red hill	-0.4000E-03	0.1599E-02	-0.8330E+00
hopper	wild	0.2200E-02	0.7240E-03	0.1845E+01
kenger	mason	-0.2200E-02	-0.1672E-02	-0.1057E+01
kenger	mid f	0.7600E-02	0.9875E-02	-0.2275E+01
mason	mine rm2	-0.9900E-02	-0.1029E-01	0.5587E+00
mason	wild	-0.2300E-02	-0.1194E-02	-0.1844E+01
mid f	mine mt	0.2300E-02	0.1657E-02	0.5844E+00
mine mt	shade	0.1860E-01	0.1779E-01	0.1161E+01
park	red hill	-0.8600E-02	-0.1047E-01	0.2677E+01

ESTIMATED MODEL (M/YR)

block	slip(m/yr)	st.dev.(m/yr)
1	0.2510E-01	0.0000E+00
2	0.2400E-01	0.0000E+00
3	0.2161E-01	0.3757E-03
4	0.1894E-01	0.8041E-03
5	0.1688E-01	0.1138E-02
6	0.1575E-01	0.1329E-02
7	0.1560E-01	0.1375E-02
8	0.1634E-01	0.1300E-02
9	0.1790E-01	0.1133E-02
10	0.2020E-01	0.8993E-03
11	0.2326E-01	0.6219E-03
12	0.2730E-01	0.3176E-03
13	0.2200E-01	0.0000E+00
14	0.1835E-01	0.6512E-03
15	0.1387E-01	0.1419E-02
16	0.1021E-01	0.2029E-02
17	0.8101E-02	0.2382E-02
18	0.7716E-02	0.2473E-02
19	0.8971E-02	0.2342E-02
20	0.1167E-01	0.2041E-02
21	0.1561E-01	0.1621E-02
22	0.2061E-01	0.1120E-02
23	0.2648E-01	0.5718E-03
24	0.2200E-01	0.0000E+00
25	0.1653E-01	0.8688E-03
26	0.1030E-01	0.1877E-02
27	0.5350E-02	0.2662E-02
28	0.2504E-02	0.3100E-02
29	0.1945E-02	0.3197E-02
30	0.3540E-02	0.3011E-02
31	0.7013E-02	0.2615E-02
32	0.1204E-01	0.2070E-02
33	0.1828E-01	0.1428E-02
34	0.2540E-01	0.7281E-03
35	0.1800E-01	0.0000E+00
36	0.1286E-01	0.1068E-02
37	0.6648E-02	0.2205E-02
38	0.1591E-02	0.3024E-02
39	-0.1297E-02	0.3437E-02
40	-0.1727E-02	0.3485E-02
41	0.1976E-03	0.3247E-02
42	0.4184E-02	0.2800E-02
43	0.9861E-02	0.2208E-02
44	0.1683E-01	0.1520E-02
45	0.2469E-01	0.7738E-03
46	0.1600E-01	0.0000E+00
47	0.1077E-01	0.1113E-02
48	0.4618E-02	0.2206E-02
49	-0.2652E-03	0.2929E-02
50	-0.2931E-02	0.3256E-02
51	-0.3120E-02	0.3257E-02
52	-0.9439E-03	0.3015E-02
53	0.3298E-02	0.2594E-02
54	0.9225E-02	0.2047E-02
55	0.1643E-01	0.1411E-02
56	0.2449E-01	0.7196E-03
57	0.1400E-01	0.0000E+00
58	0.9525E-02	0.8731E-03
59	0.4141E-02	0.1708E-02
60	-0.1262E-03	0.2267E-02
61	-0.2345E-02	0.2545E-02
62	-0.2255E-02	0.2590E-02
63	0.4144E-04	0.2444E-02
64	0.4265E-02	0.2145E-02
65	0.1006E-01	0.1722E-02
66	0.1704E-01	0.1203E-02
67	0.2482E-01	0.6185E-03
68	0.1200E-01	0.0000E+00
69	0.8734E-02	0.4658E-03
70	0.4673E-02	0.1033E-02
71	0.1482E-02	0.1532E-02
72	-0.2283E-04	0.1885E-02
73	0.4136E-03	0.2061E-02
74	0.2729E-02	0.2055E-02
75	0.6706E-02	0.1878E-02
76	0.1205E-01	0.1551E-02
77	0.1844E-01	0.1105E-02
78	0.2554E-01	0.5745E-03
79	0.1100E-01	0.0000E+00
80	0.8591E-02	0.5744E-03
81	0.5905E-02	0.1249E-02
82	0.3998E-02	0.1823E-02
83	0.3354E-02	0.2211E-02
84	0.4150E-02	0.2387E-02
85	0.6384E-02	0.2353E-02
86	0.9939E-02	0.2130E-02
87	0.1463E-01	0.1748E-02
88	0.2022E-01	0.1239E-02
89	0.2645E-01	0.6430E-03
90	0.9000E-02	0.0000E+00
91	0.8318E-02	0.8429E-03
92	0.7440E-02	0.1774E-02
93	0.6873E-02	0.2485E-02
94	0.6976E-02	0.2891E-02
95	0.7971E-02	0.3006E-02
96	0.9965E-02	0.2871E-02
97	0.1297E-01	0.2535E-02
98	0.1694E-01	0.2041E-02
99	0.2175E-01	0.1429E-02
100	0.2720E-01	0.7360E-03
101	0.8000E-02	0.0000E+00
102	0.8388E-02	0.9571E-03
103	0.8682E-02	0.1995E-02
104	0.8945E-02	0.2754E-02

105	0.9437E-02	0.3151E-02
106	0.1041E-01	0.3220E-02
107	0.1205E-01	0.3027E-02
108	0.1450E-01	0.2636E-02
109	0.1785E-01	0.2099E-02
110	0.2213E-01	0.1458E-02
111	0.2727E-01	0.7472E-03
112	0.6000E-02	0.0000E+00
113	0.6766E-02	0.8724E-03
114	0.7572E-02	0.1781E-02
115	0.8293E-02	0.2419E-02
116	0.9016E-02	0.2733E-02
117	0.9919E-02	0.2763E-02
118	0.1122E-01	0.2575E-02
119	0.1317E-01	0.2225E-02
120	0.1603E-01	0.1762E-02
121	0.2011E-01	0.1219E-02
122	0.2576E-01	0.6232E-03
123	0.4000E-02	0.0000E+00
124	0.3847E-02	0.5625E-03
125	0.4233E-02	0.1095E-02
126	0.4840E-02	0.1450E-02
127	0.5458E-02	0.1615E-02
128	0.6101E-02	0.1619E-02
129	0.6924E-02	0.1500E-02
130	0.8169E-02	0.1291E-02
131	0.1021E-01	0.1019E-02
132	0.1374E-01	0.7036E-03
133	0.2020E-01	0.3597E-03
134	0.3300E-01	0.0000E+00
135	-0.6309E-01	0.1236E-01

THE EXTENSION-RATE PERPENDICULAR TO THE F.P. IS -0.6309E-01 MICRO-STRAIN/YR

stn	displacement	x-direc	y-direc
airway		-0.4670E-02	0.1212E-01
almond		-0.4256E-02	0.9817E-02
barren		-0.4740E-02	0.1145E-01
bench		-0.1173E-02	0.5115E-02
blhllres		-0.4375E-02	0.1295E-01
bonnie		0.5248E-02	-0.4489E-02
castle		0.6248E-02	-0.5359E-02
chiches2		-0.4561E-02	0.9821E-02
cotton		0.4156E-02	-0.2351E-02
davis		-0.4162E-02	0.1312E-01
gold		0.4062E-02	-0.2515E-02
hatch		0.1262E-02	-0.1650E-03
hopper		-0.2620E-02	0.7235E-02
kenger		0.5565E-02	-0.4645E-02
mason		-0.2778E-02	0.6135E-02
mid f		-0.3447E-02	0.8556E-02
mine mt		0.7417E-02	-0.4954E-02
mine rm2		0.7418E-02	-0.4954E-02
park		0.5257E-02	-0.4618E-02
red hill		-0.2135E-02	0.5178E-02
shade		-0.6458E-02	0.9936E-02
tess		-0.4638E-02	0.1237E-01
wild		-0.3341E-02	0.7594E-02

APPENDIX C

The FORTRAN program
INVERSE.FOR

```
C  NEW VERSION OF THE INVERSE PROGRAM BY RAH JUNE 1986
C  THIS VERSION GIVES OPTION OF INCLUDING COMPONENT OF EXTENSION
C  PERPENDICULAR TO THE FAULT PLANE, IN ADDITION TO THE 2-D FAULT MODEL
C  IT ALSO GIVES A CHOICE OF SMOOTH MODEL-WEIGHTING OR
C  MODEL-WEIGHTING BY THE IDENTITY (WHICH GIVES A MINIMUM-LENGTH SOLN)
C
c     This Fortran program is used to invert geodetic line-length or strain
c     data for slip on a gridded 2-dimensional vertical strike-slip fault.
c     We use 2-d Chinnery functions for the dislocation model, and
c     incorporate data weighting and model weighting.
c     Subroutine Sensit does a singular value decomposition on the Chinnery
c     function matrix.  The resulting model estimates, covariance, resolution,
c     etc. are determined for a varying number of singular values.
c     Misfits and resolution are calculated for the data.
c     Fixed faults (constrained rates with standard deviations)
c     and variable faults are permitted.
c
c
c  Compile with single precision IMSL routines
c
c  input at terminal
c           nstart  -  number of singular values to start with
c           numrum  -  number of times to loop through the singular
c                      value loop
c           nwt     -  type of model-weighting desired
c                      nwt=1  no weighting, results in minimum-length model
c                      nwt=2  smoothing, results in smooth model
c           nstrain -  normal strain component in model
c                      nstrain = 1  no
c                      nstrain = 2  yes
c
c  input in data file
c           iopt    -  0 for strain cards, 1 for *pairs* of llcards
c           nf      -  total number of faults in single fit (fixed + variable)
c           nvf     -  number of variable faults
c           nlines  -  number of lines
c           nstns   -  number of stations
c           nff     -  number of fixed faults (nf includes nff and nvf)
c
c           station coordinate cards in meters
c           line length change cards (sta 1,sta 2,length,length change, std
c           dev)
c           all in meters
c
```

```
c           model
c               nhe     - number of horizontal elts in grid
c               nve     - number of vertical elts in grid
c               al      -  half length of fault (km)
c               du      -  depth to top of fault (km)
c               dl      -  depth to bottom of fault (km)
c               az      -  az of fault (clockwise from north,in degrees)
c               fx,fy   -  midpt coordinates of center of fault
c               consrat-  constrained slip(m) on specified block #
c               see subroutine FMODEL for more details
c     output
c        all input
c        program converts all coordinates of fault blocks and data
c        to a fault centered system
c        with xl parallel to fault, x2perpendicular to fault and x3 down
c        (xl = x,  x2 = y)
c        Right lateral slip is positive.
c
c     calls CHINSSL, CRD, FMODEL, GETINT, MODELWT, SENSIT, OUTSG
c           GETINT, WRHEADER, LENTRUE, LJUST, LEFTEND
c
c     this program uses IMSL routines:
c     EBALAF, EBBCKF, EGRH3F, EHBCKF, EHESSF, EIGRF, LSVDB,
c     LSVDF, LSVG1, LSVG2, UERTST, UGETIO, USWFM, USWFV, VHS12,
c     VMULFB, VMULFF, VMULFP
c
      real*8  sn(2,500), sta(200), distan(500), dist2, dist
      real*4  x(200), y(200), dx(2), dy(2), xx(200), yy(200)
      dimension tmstore(500,500),s(500)
      dimension green(500,500), deltal(500,500), descri(10)
      dimension azi(500), sdx(200), sdy(200), str(500)
      dimension sd(500),stamovx(500,500),stamovy(500,500)
      dimension stndispx(200), stndispy(200), stndisp(200)
      dimension xpos(500),ypos(500),proje(500),projn(500)
      character*5 net(500)
      character*1 icross(500,500), eff, space
      integer keep(2)
c
      dimension aprime(500,500),v(500,500),ut(500,500)
      dimension u(500,500),sinval(500),wk(1000)
      dimension workar(500,500),td(500,500),tdinv(500,500)
      dimension tminv(500,500),tm(500,500),resolv(500,500)
      dimension datares(500,500),prod1(500,500),prod2(500,500)
      dimension prod3(500,500),sinvalm2(500),sinvalm1(500)
      dimension covm(500,500),utnew(500,500),c(500,500)
      dimension estmod1(500),datacalc(500),dmisfit(500)
      dimension sdmod(500),tmt(500,500),vnew(500,500)
      dimension als(500),dus(500),dls(500),us(500)
      dimension azs(500),fxs(500),fys(500)
      dimension chekmod(500),slipco(500),OMEGA(500)
c
c
c
```

```
      data blank/4h    /, nldim/500/, nfdim/500/, eff/'F'/, space/' '/
      data dtr/0.01745329/
c
c
c
c input at terminal number of singular values to start with
c and the number of times to loop through the singular value decomposition
c in subroutine SENSIT
c Also input if smooth model-weighting is wanted (1=no)
c
c
      call getint('# of svals to start with',17,nstart)
      call getint('# of additional times to loop thru the
     +s.v.d.',0,numrun)
      call getint('modelwting 1=no,2=yes',2,nwt)
      call getint('normal strain 1=no,2=yes',2,nstrain)
      write(16,503) nstart,numrun,nwt,nstrain
c
c
  105 continue
      jf=0
c
c read header card (data file)
c stop when nf (first 5 columns) = 0
c
      read (5,603) iopt,nf,nlines,nstns,nff,descri
      if(nf.eq.0) go to 100
      write (6,505) nwt,iopt,nf,nlines,nstns,nff,descri
c
c ***************************************************************
c read station coordinates.
c ***************************************************************
c
      read(5,605)(sta(i),xx(i),yy(i),i=1,nstns)
      write(6,507)
      write(6,509)(sta(i),xx(i),yy(i),i=1,nstns)
c
c ***************************************************************
c read data, stop when nlines=0 and jump to model generating section
c read strain cards if iopt=0
c read *pairs* of llcards if iopt=1
c ***************************************************************
c
      if (nlines.eq.0) go to 107
      if(iopt.eq.1) go to 200
      read (5,607) (net(j),(sn(i,j),i=1,2),distan(j),
     +             str(j),sd(j),azi(j),j=1,nlines)
       write(6,511)
       write(6,513) (net(j),(sn(i,j),i=1,2),distan(j),
     +              str(j),sd(j),azi(j),j=1,nlines)
      go to 202
c
```

```
200   do 201 i=1,nlines
         read(5,609) net(i),(sn(k,i),k=1,2),azi(i),distan(i)
         read(5,611) dist2,sd(i)
         str(i)=dist2-distan(i)
         write(6,515) net(i),(sn(k,i),k=1,2),distan(i),
     +                str(i),sd(i),azi(i)
201      continue
202      continue
c
c ****************************************************************
c At this point the data has been read.
c Subroutine fmodel generates the fault model.
c ****************************************************************
c
      call fmodel(nf,nff,nfdim,als,dus,dls,azs,fxs,fys,
     +            nve,nhe,elarea,ratio)
      nvet = nve
      nhet = nhe
c
c ****************************************************************
c generate the data kernel G.
c ****************************************************************
c
c for right lateral motion positive, slip = -1.0
c loop on fault segments, jf= # of segment
c
107   continue
      jf=jf+1
c
      al=als(jf)
      du=dus(jf)
      dl=dls(jf)
      slip=-1.0
      az=azs(jf)
      fx=fxs(jf)
      fy=fys(jf)
c
c convert to fault-centered coordinate system.
c first save xpos and ypos
c
      do 108 i=1,nstns
      xpos(i)=xx(i)
      ypos(i)=yy(i)
      x(i)=xx(i)
      y(i)=yy(i)
108   continue
c
      call crd (jf,x,y,nstns,nf,az,fx,fy,proje,projn)
c
c compute station displacements due to current fault,
c where x(i), y(i) are now in fault centered coords
c
```

```
        s11=0.
        s22=0.
        s12=0.
        do 106 i=1,nstns
        x(i)=x(i)/1000.
        y(i)=y(i)/1000.
        x3=0
        kk=i
c
c
c         a1,a2,a3,a4 are displacement components in the u1-direction
c         b1,b2,b3,b4                                    u2-direction
c         c1,c2,c3,c4                                    u3-direction
c         d1,d2,d3,d4 are the strain components   in the  1-direction
c         f1,f2,f3,f4                                     2-direction
c         h1,h2,h3,h4                                    12-direction
c         sdx is the total displacement in the x-direction (parallel fault)
c         sdy                                y-direction (perpendicular)
c         sdz                                z-direction (vertical)
c
      call chinssl (slip,+x(kk),y(kk),x3,+a1,+d1,a1,b1,c1,d1,f1,h1)
      call chinssl (slip,+x(kk),y(kk),x3,+a1,+du,a2,b2,c2,d2,f2,h2)
      call chinssl (slip,+x(kk),y(kk),x3,-a1,+d1,a3,b3,c3,d3,f3,h3)
      call chinssl (slip,+x(kk),y(kk),x3,-a1,+du,a4,b4,c4,d4,f4,h4)
      s11=(d1-d2-d3+d4)*1000.
      s22=(f1-f2-f3+f4)*1000.
      s12=(h1-h2-h3+h4)*1000.
c
      sdx(i)=a1-a2-a3+a4
      sdy(i)=b1-b2-b3+b4
      sdz=c1-c2-c3+c4
c
106   continue
c
c compute line length changes due to current fault.
c
       if (nlines.eq.0) go to 112
       do 104 j=1,nlines
c
           do 103 i=1,2
             keep(i) = 0
             do 102 k=1,nstns
             if (sn(i,j).ne.sta(k)) go to 102
             keep(i)=k
             kk=keep(i)
             go to 101
102          continue
101          continue
             dx(i)=sdx(kk)
             dy(i)=sdy(kk)
103          continue
c
```

```
        if ( keep(1)*keep(2) .eq. 0 ) then
           write (6,517) (keep(i),sn(i,j),i=1,2)
           dist = 0.0
           deltal(j,jf) = 999.
           azim = 999.
        else
           ll = keep(2)
           mm = keep(1)
           sx=x(ll)-x(mm)
           sy=y(ll)-y(mm)
           dist = sqrt(sx*sx+sy*sy)
           rx=dx(2)-dx(1)
           ry=dy(2)-dy(1)
           deltal(j,jf) = (rx*sx + ry*sy) / dist
           if ( y(ll)*y(mm) .ge. 0. ) then
              icross(j,jf) = space
           else
              icross(j,jf) = eff
           endif
           azim = atan2(-sx,sy) * 180./3.14158 + 90.
           if (azim .le. 0.) azim = azim + 360.
           dist = dist * 1000.
        endif
c
c . . . . . .
c
        dist = dist + deltal(j,jf)
c
c
104     continue
112     continue
c
c*******************************************************
c
c  express station displacements in ns-ew coord system.
c  save station displacements due to current fault before
c  obtaining next one.
c
      call crd(jf,sdx,sdy,nstns,nf,az,0.,0.,proje,projn)
      do 114 it=1,nstns
      stamovx(it,jf)=sdx(it)
      stamovy(it,jf)=sdy(it)
114   continue
c
c*********************************************
c . . . . . .
      if(jf.lt.nf) go to 107
c
c end of loop
c ************************************************************
```

```
c     calculate coefficient for normal strain component
       if(nstrain.ne.2) go to 458
       fx=fxs(nf)
       fy=fys(nf)
       az=azs(nf)
       call crd(jf,xpos,ypos,nstns,nf,az,fx,fy,proje,projn)
c
c ************************************************************
458    nvf=nf-nff
       lindex=nlines+nff
       nfplus1 = nf+1
       if(nstrain.ne.2) nfplus1=nf
c
c set Green's function matrix to the nf columns of delta1
c
       do 302 i=1,nlines
       do 303 j=1,nf
       green(i,j)=delta1(i,j)
303    continue
302    continue
```

```
c
c  add to 1st nf rows of green 1 column representing the strain component
c  perpendicular to the fault
c  alpha = angle of the fault parallel axis counterclockwise from north
c
      if(nstrain.ne.2) go to 313
      alpha = 180.0 - azs(nf)
      do 312 i=1,nlines
      omega(i) = (90.0 - (azi(i) + alpha)) * dtr
      green(i,nfplus1)=distan(i)*1.0e-06*cos(omega(i))*cos(omega(i))
312   continue
c
c
c  add lines to green's funcs for the nff constrained segments
c
313   if(nff.eq.0) go to 310
c
c         set nlines+1 to nlines+nff rows of the data kernel to zeros
       do 304 i=nlines+1,nlines+nff
       do 305 j=1,nfplus1
           green(i,j)=0.0
305    continue
304    continue
c
c ********************************************************************
c      read constraints and standard deviations (rates in m/yr)
c ********************************************************************
       write(6,519)
       do 306 i=1,nff
       read(5,615)index,consrat,conswt
       write(6,521) index,consrat,conswt
       green(nlines+i,index) = 1.0
       str(nlines+i)         = consrat
       sd(nlines+i)          = conswt
306    continue
c
310    continue
c
c ********************************************************************
c Form data weight matrix - weight by 1/variance of each datum
c ********************************************************************
      do 300 i=1,lindex
      do 301 j=1,lindex
          if(i.eq.j)    td(i,j) = 1.0/sd(i)
          if(i.eq.j) tdinv(i,j) = sd(i)
          if(i.ne.j)    td(i,j) = 0.0
          if(i.ne.j) tdinv(i,j) = 0.0
301   continue
300   continue
c ********************************************************************
c Form model weigting subroutine
c********************************************************************
       CALL MODELWT(NF,NFPLUS1,NFDIM,TM,TMT,NVET,NHET,RATIO,NWT)
c
```

```
c ******************************************************************
C FORM TMINV USING IMSL ROUTINE LGINF TO CALCULATE GENERALIZED INVERSE OF TM
C
         DO 111 I=1,NFPLUS1
         DO 111 J=1,NFPLUS1
            TMSTORE(I,J)=TM(I,J)
111      CONTINUE
         TOL=0.0
       CALL LGINF(TMSTORE,500,NFPLUS1,NFPLUS1,TOL,TMINV,
     +            500,S,WK,IER0)
C
c ******************************************************************
c Calculate model, resolution, misfits, covariance,
c chi-squared for weighted-model and data problem
c ******************************************************************
      call sensit (v,green,aprime,ut,sinval,wk,workar,
     +             u,td,tdinv,tm,tminv,prod1,prod2,prod3,
     +             resolv,datares,sinvalm2,sinvalm1,covm,
     +             utnew,c,estmod1,str,n1dim,nfdim,NFPLUS1,
     +             nlines,nf,nvf,nff,datacalc,sd,dmisfit,sdmod,
     +             lstsig,sn,nstart,numrun,lindex,vnew,
     +             chekmod,nhe,nve,elarea,slipco,sta,
     +             stamovx,stamovy,nstns,stndispx,stndispy,
     +             stndisp,proje,projn)
c
c ******************************************************************
c ******************************************************************
cw       call uswfm('greens funcs',12,green,500,lindex,nf,3)
c
      go to 105
100   continue
c
c     main program read and write statement formats
c
c
  503 format (1x,'start with',i4,' singular values'/
     +        ' loop through the s.v.d.',i2,' more times'/
     +        ' model-weighting option',i2,' 1=none,2=smooth'/
     +        ' normal strain component',i2,' 1=no,2=yes')
  505 format (11x,'nwt (1=no-modelwting, 2=smooth) = ',i3/
     +        11x,'iopt (0=strcards, 1=llcards)... = ',i3/
     +        11x,'number of faults.............. = ',i3/
     +        11x,'number of lines............... = ',i3/
     +        11x,'number of stations............ = ',i3/
     +        11x,'number of fixed faults........ = ',i3/
     +        11x,10a4)
  507 format (//10x,'station',9x,'xcoord(m)',6x,'ycoord(m)')
  509 format (10x,a8,2x,2f15.3)
  511 format (//1x,'net',4x,'stn1',5x,'stn2',7x,'length(m)',
     +         ' dl/dt(m/yr)',1x,'stdev',2x,'az')
```

```
  513 format (1x,a5,2x,a8,1x,a8,2x,f10.3,2x,f7.4,2x,f6.4,2x,f5.1)
  515 format (1x,a5,1x,a8,1x,a8,2x,f10.3,2x,f7.4,2x,f6.4,2x,f5.1)
  517 format (1x,2(i3,'/',a8,2x),' 0 station not on X-Y list')
  519 format (//1x,' CONSTRAINTS: RATE AND STANDARD DEVIATION ',
     +        '(M/YR)')
  521 format (1x,'for block',i4,' rate =',e12.4,'+/-',e12.4)
  603 format (i1,i4,3i5,5x,10a4)
  605 format (a8,6x,f16.0,6x,f16.0)
  607 format (a5,8x,a8,1x,a8,t44,f10.3,t54,f7.4,
     +           1x,f6.4,t69,f5.1)
  609 format (a5,1x,a8,1x,a8,t37,f5.1,t56,f11.4)
  611 format (t56,f11.4,f10.4)
  615 format (i4,e12.4,e12.4)
c
c
      stop
      end
c
c
c
c
```

```
c ***********************************************
c SUBROUTINES CALLED BY INVERSE
c ***********************************************
c ***********************************************
c GETINT
c
c converts a string answer to an integer provided the string
c is not blank....
c        ques = question to be asked
c        intdefault = default answer
c        int = integer answer
c
      subroutine getint(ques,intdefault,int)
c
      integer int, intdefault
      character*(*) ques
      character*10 default, ans, fmt
      character bel*1
      parameter (bel=char(7))
c
      write(default,'(i10)') intdefault
      call ljust(default)
      lengdef = lentrue(default)
c
      lengq = lentrue(ques)
 10   print '(/,1h$,a)'
     &      ques(1:lengq)//':  ['//default(1:lengdef)//'] '
      read (*,'(a)') ans
      leng = lentrue(ans)
      if(leng.eq.0) then
         int = intdefault
      else
         write(fmt,'(a,i3,a)') '(i' leng, ')'
         read(ans(1:leng),fmt,err=90) int
      endif
c
      return
c
c.....error message
 90   print '(/,1x,a)',
     &'*** error, expecting an integer answer... try again...'
      print *, bel
      goto 10
      end
c
c
```

```
c ***********************************************
c MODELWT
c
c subroutine to calculate model-weight matrix
c for second differences (laplacian) in two dimensional inverse
c
      SUBROUTINE MODELWT(NF,NFPLUS1,NFDIM,D,DT,NVE,NHE,RATIO,NWT)
c
      DIMENSION d(nfdim,1),dt(nfdim,1)
      dimension xpartd(500,500), ypartd(500,500)
c
c
c  nf=number of faults
c  nhe=number of horizontal elements in grid
c  nve=number of vertical elements in grid
c  ngrid=number of elements in grid
c  nuprhs=upper r.h. block in grid
c  nlrhs=lower r.h. block in grid
c  ratio=length/height of blocks in grid=dx/dy
c  nwt=1 if no model-weighting is desired,
c  nwt=2 if smoothing is desired
c
        RATIOSQ = RATIO * RATIO
        NGRID = NHE * NVE
        NUPRHS = NGRID - NVE + 2
        NLRHS  = NGRID + 1
C
C       IF NO MODEL-WEIGHTING IS DESIRED, ASSIGN MODEL-WEIGHTING
C       MATRIX, [D], TO BE THE IDENTITY MATRIX
C
        IF(NWT.EQ.1)GO TO 500
C
C       INITIALIZE ARRAY ELEMENTS
        DO 5 I=1,NLRHS+2
        DO 5 J=1,NLRHS+2
        XPARTD(I,J) = 0.0
        YPARTD(I,J) = 0.0
        D(I,J) = 0.0
  5     CONTINUE
C
C       TO LEFT EDGE OF GRID ASSIGN XPART OF LAPLACIAN
        DO 10 I = 2,NVE+1
        XPARTD(I,1)     = + 1.0
        XPARTD(I,I)     = - 2.0
        XPARTD(I,I+NVE) = + 1.0
 10     CONTINUE
C
C       TO RIGHT EDGE OF GRID ASSIGN XPART OF LAPLACIAN
        DO 20 I = NUPRHS,NLRHS
        XPARTD(I,I-NVE) = + 1.0
        XPARTD(I,I)     = - 2.0
 20     CONTINUE
C
```

```
C       TO TOP ASSIGN YPART OF LAPLACIAN
        DO 30 I = 2,NUPRHS,NVE
        YPARTD(I,I)   =  - 1.0
        YPARTD(I,I+1) =  + 1.0
 30     CONTINUE
C
C       TO BOTTOM ASSIGN YPART OF LAPLACIAN
        DO 40 I = NVE+1,NLRHS,NVE
        YPARTD(I,I-1) =  + 1.0
        YPARTD(I,I)   =  - 2.0
        YPARTD(I,NGRID+2) = + 1.0
 40     CONTINUE
C
C       ASSIGN XPARTS TO ALL PREVIOUSLY UNASSIGNED
C       (EXCLUDE LEFT & RIGHT EDGES)
        DO 100 I  = NVE+2, NLRHS-NVE
        XPARTD(I,I-NVE) = + 1.0
        XPARTD(I,I)     = - 2.0
        XPARTD(I,I+NVE) = + 1.0
 100    CONTINUE
C
C       ASSIGN YPARTS TO ALL PREVIOUSLY UNASSIGNED
C       (EXCLUDE TOP & BOTTOM EDGES)
        K = 3
 150    L = K + NVE - 3
        DO 200 I = K,L
        IF(I.GT.NLRHS) GO TO 250
        YPARTD(I,I-1) =  + 1.0
        YPARTD(I,I)   =  - 2.0
        YPARTD(I,I+1) =  + 1.0
 200    CONTINUE
        K = K + NVE
        GO TO 150
C
C       ADD XPART & YPART OF LAPLACIAN TO FORM D(I,J)
C       WEIGHT SMOOTHING FOR NON-SQUARE BLOCKS
 250    DO 300 I = 1,NLRHS+1
        DO 300 J = 1,NLRHS+1
        D(I,J) = XPARTD(I,J) + RATIOSQ*YPARTD(I,J)
 300    CONTINUE
c
C       ASSIGN D(I,J) TO 2 SIDE CONSTRAINTS AND TO THE STRAIN COMPONENT
        D(1,1) = 1.0
        D(NLRHS+1,NLRHS+1) = 1.0
        D(NFPLUS1,NFPLUS1) = 1.0
        GO TO 700
C
500    DO 600 I=1,NFPLUS1
       DO 600 J=1,NFPLUS1
       IF(I.EQ.J) D(I,J)=1.0
       IF(I.NE.J) D(I,J)=0.0
600    CONTINUE
C
C
```

```
C     Form d-transpose (dt)
700   do 900 i=1,nfplus1
      do 900 j=1,nfplus1
      dt(i,j)=d(j,i)
900   continue
C
      return
      end
c
c
c
c ***************************************************************************
c CRD
c translates and rotates coordinates to fault-centered system, with x parallel
c to fault.
c
c called from inverse
c
      subroutine crd(jf,x,y,n,nf,azi,fx,fy,proje,projn)
      dimension x(n),y(n),proje(n),projn(n)
      alpha= ( 90.-azi)*2.*3.1415927/360.
      c=cos(alpha)
      s=sin(alpha)
      do 20 i=1,n
      xx=x(i)-fx
      yy=y(i)-fy
      x(i)=xx*c+yy*s
      y(i)=+xx*s-yy*c
   20 continue
c
      if(jf.ne.nf)go to 30
      do 25 i=1,n
      proje(i)=-y(i)*sin(-alpha)
      projn(i)=-y(i)*cos(-alpha)
   25 continue
c
   30 continue
      return
      end
c
```

```
c ***********************************************
c FMODEL
      subroutine fmodel(nf,nff,nfdim,als,dus,dls,azs,fxs,fys,
     +                  nve,nhe,elarea,ratio)
c
c  this subroutine reads fault cards and calculates
c     elements required by inverse.
c     faults can be specified in 3 ways:
c        one endpoint, azimuth, length
c        two endpoints
c        midpoint, azimuth, half-length
c     subroutine returns midpoint, azimuth, and half-length to inverse.
c . . . . . . . . . . . . . . . . . . . . . . . .
c
c  variables:
c
c     kode         =   1=one endpoint, azimuth, length
c                      2=both endpoints
c                      3=midpoint, azimuth, half-length
c                      4=grid; one endpoint azimuth length
c     nhe          =   number of horizontal elements, used for kode=4
c     nve          =   number of vertical elements, used for kode=4
c     du           =   depth to top (km)
c     dl           =   depth to bottom (km)
c     az           =   azimuth (from 1 to 2)
c     length       =   total fault length (km), or half-length if kode=3
c     x1,y1,x2,y2  =   end pt coordinates (m), or (x1,y1)=midpt if kode=3
c     al           =   half length (km)
c     fx,fy        =   mid pt coordinates (m)
c
      real length
      data dtr/0.01745329/
      dimension als(nfdim),dus(nfdim),dls(nfdim)
      dimension azs(nfdim),fxs(nfdim),fys(nfdim)
c
      k=0
c
c read model parameters
c
  700 read(5,617) kode,nhe,nve,du,dl,az,length,x1,y1,x2,y2
      write(6,523)
      write(6,525) kode,nhe,nve,du,dl,az,length,x1,y1,x2,y2
c
      go to (1,2,3,4) kode
c
c ************************************
c one endpoint, azimuth, length
c ************************************
1     angle = az * dtr
      al = length/2.
      fx = x1 + al * sin(angle) * 1000.
      fy = y1 + al * cos(angle) * 1000.
      go to 5
c ************************************
```

```
c two endpoints
c ************************************
2     continue
      fx = 0.5 * (x1 + x2)
      fy = 0.5 * (y1 + y2)
      dx = x2 - x1
      dy = y2 - y1
      a1 = sqrt(dx*dx + dy*dy) / 2000.
      az = atan2(dx,dy)/dtr
      if ( az .lt. 0.0 ) az = az + 360.
      go to 5
c ************************************
c midpoint, azimuth, half-length
c ************************************
3     fx=x1
      fy=y1
      a1=length
      go to 5
c ************************************
5     k=k+1
      als(k)=a1
      dus(k)=du
      dls(k)=d1
      azs(k)=az
      fxs(k)=fx
      fys(k)=fy
c
      if(k.eq.nf) go to 1000
      if(k.lt.nf) go to 700
c ************************************
c gridded fault elements
c ************************************
4     continue
c
      write(6,527) nhe
      write(6,529) nve
c
c
      nhesave=nhe
      nvesave=nve
      ngf=nhe*nve
      deltalen=length/float(nhe)
      deltaht=(d1-du)/float(nve)
      ratio=deltalen/deltaht
      elarea=deltalen*deltaht*1.0e+10
      azrads=(180-az)*3.14159/180.0
      deltax=1000*deltalen*sin(azrads)
      deltay=1000*deltalen*cos(azrads)
      xstart=x1-0.5*deltax
      ystart=y1+0.5*deltay
      gdu=du
c
```

```
      do 600 j=1,nhe
      fx=xstart+j*deltax
      fy=ystart-j*deltay
      do 500 i=1,nve
      k=k+1
      du=gdu + (i-1)*deltaht
      dl=gdu + i*deltaht
      als(k)=deltalen/2.0
      dus(k)=du
      dls(k)=dl
      azs(k)=az
      fxs(k)=fx
      fys(k)=fy
  500 continue
  600 continue
c
      if(k.lt.nf) go to 700
c
c ************************************
c print computed elements
 1000 continue
      write(6,531)
      write(6,533)
      do 1100 i=1,nf
      write(6,535) dus(i),dls(i),azs(i),als(i),fxs(i),fys(i)
 1100 continue
c
c
      nve = nvesave
      nhe = nhesave
c
c     read and write format statements
c
  523 format(//1x,' INPUT MODEL GRIDDING')
  525 format(1X,3i4,4x,8f8.0)
  527 format(/1x,'# of horizontal elements in grid =  ',i4)
  529 format(1x,'# of vertical   elements in grid =  ',i4)
  531 format(/1x,'MODEL PARAMETERS')
  533 format(4x,'du(km)',2x,'dl(km)',2x,'az',3x,
     +       'hfln(km)',1x,'xl(m)',1x,'yl(m)')
  535 format(4f8.1,2f8.0)
  617 format(3i4,4x,8f8.0)
c
      return
      end
c
c ***************************************************************
```

```
c SENSIT
      subroutine sensit (v,a,aprime,ut,sinval,wk,workar,
     +                   u,td,tdinv,tm,tminv,prod1,prod2,
     +                   prod3,resolv,datares,sinvalm2,sinvalm1,
     +                   covm,utnew,c,estmod1,str,nldim,nfdim,
     +                   NFPLUS1,nlines,nf,nvf,nff,
     +                   datacalc,sd,dmisfit,sdmod,lstsig,sn,
     +                   nstart,numrun,lindex,vnew,chekmod,
     +                   nhe,nve,elarea,slipco,sta,
     +                   stamovx,stamovy,nstns,stndispx,stndispy,
     +                   stndisp,proje,projn)
c
c  Computes the u,v and s matrices of a singular value decomposition.
c   Prints out singular values.
c   Also calculates model-resolution, covariance, and
c   estimated model, and data resolution
c   Does calculations which keep a varying number of singular values.
c   Can compare chekmod and 1/smallest singval to decide how many
c   singular values to keep- want chekmod | 1/smallest singular value.
c
c   Notes: nlines  =  number of baselines
c          nf      =  total number of faults (fixed + variable)
c          nff     =  number of fixed faults
c          nvf     =  number of variable faults
c          a       =  Greens func array in main program (unrotated)
c          aprime  =  Greens func array in Sensit (rotated space)
c
      dimension a(nldim,1),aprime(nldim,1),v(nldim,1)
      dimension ut(nldim,1),sinval(1),wk(1)
      dimension u(nldim,1),workar(nldim,1),resolv(nfdim,1)
      dimension td(nldim,1),tdinv(nldim,1),tm(nfdim,1)
      dimension tminv(nfdim,1),prod1(nfdim,1)
      dimension prod2(nfdim,1),prod3(nfdim,1)
      dimension sinvalm1(nfdim),sinvalm2(nfdim)
      dimension c(nfdim,1),covm(nfdim,1),utnew(nldim,1)
      dimension estmod1(nfdim),str(nldim),npar(4)
      dimension datares(nldim,1),datacalc(nldim),sd(nldim)
      dimension dmisfit(nldim),sdmod(nfdim),slipco(nfdim)
      dimension vnew(nldim,1),chekmod(nfdim),product(500)
      dimension stamovx(nfdim,1),stamovy(nfdim,1)
      dimension stndispx(nldim),stndispy(nldim)
      dimension stndisp(2*nldim),proje(nldim),projn(nldim)
      real*8    sn(2,nldim), sta(2*nldim)
c
c
c     Weight (or 'transform') the data kernel A; A'=Td*A*Tm**-1
c
      call vmulff(td,a,lindex,lindex,nfPLUS1,nldim,nldim,
     +            prod1,nldim,ier)
      call vmulff(prod1,tminv,lindex,nfPLUS1,nfPLUS1,
     +            nldim,nfdim,aprime,nldim,ier)
cw    call uswfm ('Rotated Greens func',19,aprime,nldim,
cw   +              lindex,nfPLUS1,3)
c
```

```
c     Form singular value decomposition of aprime where A'=U*S*Vt
c     note that U,S,and V are in transformed (primed) system
c     Create a space to store V
      do 1 i = 1,lindex
         do 1 j = 1,nfPLUS1
            v(i,j) = aprime(i,j)
1     continue
c
c      create a space to put U, where  Ut = U-transpose
       do 45 i=1,lindex
       do 42 j=1,lindex
       if(i.eq.j) ut(i,j) = 1.0
       if(i.ne.j) ut(i,j) = 0.0
42     continue
45     continue
c
c  **  Call IMSL SVD routine to find aprime = u*s*vt                **
c
       call lsvdf (v,nldim,lindex,nfPLUS1,ut,nldim,
     +             lindex,sinval,wk,ier)
c
c
       write(6,537)
537    format(//' SINGULAR VALUES OF THE ROTATED GREENS FUNCS'/)
c
       call ugetio(3,5,6)
       call uswfv('singvals of aprime',18,sinval,nfPLUS1,1,3)
c
c             set a do loop to cycle through a # of singvals
c             if there are zero constrained blocks, then start
c             the loop with one singular value
c
         if(nff.eq.0) nstart=1
         do 884 ind=nstart,nstart+numrun
         if(ind.gt.nf) go to 999
         lstsig = ind
         write(6,539)lstsig,sinval(lstsig)
539      format(///' using',i5,'singvals, smallest=',e12.4)
c
c
c
c
c**    Calculate resolution matrix Tm**(-1)*V*Vt*Tm   **
c
       call vmulfp (v,v,nfPLUS1,lstsig,nfPLUS1,nldim,nldim,prod2,
     +             nfdim,ier)
       call vmulff (tminv,prod2,nfPLUS1,nfPLUS1,nfPLUS1,nfdim,nfdim,
     +             prod3,nfdim,ier)
       call vmulff (prod3,tm,nfPLUS1,nfPLUS1,nfPLUS1,nfdim,
     +             nfdim,resolv,nfdim,ier)
cw     call uswfm('resolution',10,resolv,nfdim,nfPLUS1,nfPLUS1,3)
c
c
```

```
c**    calculate data resolution (datares) where N=Td**(-1)*U*Ut*Td   **
c      form U from Ut
c
       do 150 i=1,lindex
       do 150 j=1,lindex
       u(i,j)=ut(j,i)
150    continue
c
c
       call vmulff (u,ut,lindex,lstsig,lindex,nldim,nldim,
     +              prod2,nfdim,ier)
       call vmulff (tdinv,prod2,lindex,lindex,lindex,nldim,nfdim,
     +              prod3,nfdim,ier)
       call vmulff (prod3,td,lindex,lindex,lindex,nfdim,nldim,
     +              datares,nldim,ier)
cw     call uswfm('data resolution',15,datares,nldim,lindex,lindex,3)
c
c
c
c**    Compute model covariance matrix   **
c      covm = Tm**(-1)*V*sinvalm2*Vt*(Tm**(-1))t
c
       do 5 i = 1,lstsig
         sinvalm2(i) = 1.0 / (sinval(i)*sinval(i))
5      continue
c
       npar(1) = nfPLUS1
       npar(2) = lstsig
       npar(3) = 0
       npar(4) = 0
       call vmulfb (v,nldim,sinvalm2,nfdim,npar,prod1,nfdim)
       call vmulfp (prod1,v,nfPLUS1,lstsig,nfPLUS1,nfdim,nldim,
     +              prod2,nfdim,ier)
       call vmulff (tminv,prod2,nfPLUS1,nfPLUS1,nfPLUS1,nfdim,
     +              nfdim,prod3,nfdim,ier)
       call vmulfp (prod3,tminv,nfPLUS1,nfPLUS1,nfPLUS1,nfdim,
     +              nfdim,covm,nfdim,ier)
cw     call uswfm('modelcov',8,covm,nfdim,nfPLUS1,nfPLUS1,3)
c
c
c  **  calculate sqrt(covm(i,i))= st.dev.model parameter i = sdmod(i) **
c  **  and the r.m.s. model standard deviation = rmsstdev             **
c  **  To avoid problems w/ roundoff set #'s with absolute values     **
c  **  less than 1.0e-08 equal to 0.0e-08                             **
c
c
       rmsstdev = 0.0
       do 617 i=1,nfplus1
       if(abs(covm(i,i)).lt.1.0e-08) covm(i,i)=0.0e-08
       sdmod(i)=sqrt(covm(i,i))
       rmsstdev = rmsstdev + covm(i,i)
 617   continue
       if(nvf.eq.0) go to 619
       rmsstdev = sqrt(rmsstdev/float(nvf))
c
```

```
c
      write(6,541) lstsig,rmsstdev
541   format(1x,'for',i3,' singvals'/
     +       10x,'  rms model st.dev=',e12.4)
c
c
c
c **  calculate mest = Tm**(-1)*V * 1/sinval * Ut * Td *data   **
c
619   do 16 i=1,lstsig
      sinvalm1(i) = sqrt(sinvalm2(i))
16    continue
cw    call uswfm ('v-matrix',8,v,nldim,nfPLUS1,nfPLUS1,3)
cw    call uswfm ('u-matrixt',9,ut,nldim,lindex,lindex,3)
c
c     only use first lstsig columns of v
      do 442 i=1,nfPLUS1
      do 446 j=1,lstsig
      vnew(i,j) = v(i,j)
446   continue
442   continue
c
c     only use first lstsig rows of ut
      do 335 i = 1,lstsig
      do 333 j = 1,lindex
      utnew(i,j) = ut(i,j)
333   continue
335   continue
c
c     form chekmod = sinvalm1 * utnew * td * data
c          prod1   = utnew * td
c          product = prod1 * data
c
      call vmulff(utnew,td,lstsig,lindex,lindex,
     +            nldim,nldim,prod1,nfdim,ier)
c
c
      do 222 i=1,lstsig
      product(i) = 0.0
      do 220 j=1,lindex
      product(i) = product(i) + prod1(i,j)*str(j)
220   continue
222   continue
c
      do 217 i=1,lstsig
      chekmod(i) = product(i) * sinvalm1(i)
217   continue
c
```

```
c       form estmod1 = tminv * vnew * chekmod
c            prod3   = tminv * vnew
c            estmod1 = prod3 * chekmod
c
        call vmulff(tminv,vnew,nfPLUS1,nfPLUS1,lstsig,nfdim,
     +             nldim,prod3,nfdim,ier)
c
        do 224 i=1,nfPLUS1
        estmod1(i)=0.0
        do 223 j=1,lstsig
        estmod1(i) = estmod1(i) + prod3(i,j)*chekmod(j)
223     continue
224     continue
c
c       form stndispx = stamovx * estmod1
c            stndispy = stamovy * estmod1
c
        do 209 1=1,nstns
        stndispx(1)=0.0
        stndispy(1)=0.0
        do 209 k=1,nf
        stndispx(1)=stndispx(1)+stamovx(1,k)*estmod1(k)
        stndispy(1)=stndispy(1)+stamovy(1,k)*estmod1(k)
 209    continue
c
        if(nfplus1.eq.nf) go to 211
c
c  add component for normal strain
c  first change units of normal micro-strain to normal strain
        estmod1s = 1.0e-06 * estmod1(nfplus1)
        write(6,319)estmod1s
 319    format(12x,'the strain component is',e12.4)
        do 210 i=1,nstns
        stndispx(i)=stndispx(i)+(proje(i)*estmod1s)
        stndispy(i)=stndispy(i)+(projn(i)*estmod1s)
  210   continue
c
c
c
```

```
c  ******************************************************
c     reformat model to plot on color graphics terminals
c
c     add 2 extra columns of blocks to grid on l.h.s to
c     replace large block
c     add 3 extra rows of blocks to grid on bottom
211   nec=2
      ner=3
c
      index=1
c
      do 300 i=1,nec
      do 310 j=1,nve
      slipco(index)=estmodl(1)
      index=index+1
  310 continue
      do 320 k=1,ner
      slipco(index)=estmodl(nf)
      index=index+1
  320 continue
  300 continue
c
c
      icount=2
      do 400 i=1,nhe
      do 410 j=icount,icount+nve-1
      slipco(index)=estmodl(j)
      index=index+1
  410 continue
      do 420 k=1,ner
      slipco(index)=estmodl(nf)
      index=index+1
  420 continue
      icount=j
  400 continue
c
c
c   CONVERT TO MILLIMETERS
c
      ncol=nhe+nec
      nrow=nve+ner
      ncolel=ncol*nrow
      do 900 i=1,ncolel
      slipco(i)=1000*slipco(i)
  900 continue
c
      xo=0.0
      yo=0.0
      dx=3.0
      dy=2.0
      call outsg(lstsig,ncol,nrow,xo,dx,yo,dy,slipco)
c
```

```
c*****************************
c
c
c     determine the calculated data using the model
c
c     do 334 i=1,lindex
      do 334 i=1,nlines
      datacalc(i)=0.0
      do 334 j=1,nfPLUS1
      datacalc(i)=datacalc(i)+a(i,j)*estmod1(j)
334   continue
c
c **  determine the data misfits  **
c **  = (obs. data - calc. data)i / st. dev. data point i  **
c
      do 339 i=1,nlines
      dmisfit(i)= (str(i)-datacalc(i))/sd(i)
339   continue
c
c
c
c **  determine sum (misfits)sq.  **
c **  = sum (dmisfit(i))sq.       **
      chisq=0.0
      do 356 i=1,nlines
      chisq = chisq + dmisfit(i)*dmisfit(i)
356   continue
      write(6,543)chisq
543   format(1x,11x,'sum of, the misfits squared = ',e12.4)
c
c
c
c     calculate moment rate and standard deviation
c     do not include slip rates in first or last model elements
c     shearmod = shear modulus of medium (c.g.s)
c     elarea =  area of grid elements (cm**2); returned from fmodel
c
c
      shearmod = 3.0e+11
      smoment = 0.0
      covmoment = 0.0
      do 448 i=2,nf-1
      smoment = smoment + estmod1(i)
      do 448 j=2,nf-1
      covmoment = covmoment + covm(i,j)
448   continue
      smoment = shearmod*elarea*smoment*100
      sdmoment = shearmod*elarea*sqrt(covmoment)*100
      write(6,545) smoment, sdmoment
545   format(12x,'moment rate = ',e12.4,2x,' +/- ',2x,e12.4)
c
c
```

```
c           more writing routines
c
      write(6,547) chekmod(1stsig),sinval1(1stsig)
547   format(1x,'chekmod = ',e12.4,'  1/smallest sinval = ',e12.4)
      write(6,549)
549   format(//24x,'obs data',7x,'calc data',5x,'(obs-calc)/sigma')
      do 444 i=1,nlines
      write(6,551)(sn(j,i),j=1,2),str(i),datacalc(i),dmisfit(i)
551   format(1x,a8,1x,a8,4x,e12.4,4x,e12.4,4x,e12.4)
444   continue
c
      write(6,553)
553   format(///' ESTIMATED MODEL (M/YR)')
      write(6,555)
555   format(1x,'block  slip(m/yr)     st.dev.(m/yr)')
      do 447 i=1,nfplus1
      write(6,557) i,estmod1(i),sdmod(i)
557   format(1x,i4,e12.4,3x,e12.4)
447   continue
c
      if(nfplus1.eq.nf) go to 560
      WRITE(6,559) ESTMODL(NFPLUS1)
559   FORMAT(1X,' THE EXTENSION-RATE PERPENDICULAR TO THE F.P. IS',
     +       E12.4,' MICRO-STRAIN/YR')
C
560   write(6,561)
561   format(//' stn displacement(m)'/
     +       17x,' x-direc      y-direc')
      do 458 i=1,nstns
      write(6,563) sta(i),stndispx(i),stndispy(i)
563   format(1x,a8,4x,e12.4,4x,e12.4)
458   continue
c
c     reformat stndisp to be 1 long vector w/ alternating x and y components
c     for input to main12
c
      do 882 i=1,nstns
      stndisp(2*i-1)  = stndispx(i)
      stndisp(2*i)    = stndispy(i)
882   continue
c
      do 881 i=1,2*nstns
      write(85,565) stndisp(i),sta((i+1)/2)
881   continue
565   format(e12.4,a8)
c
c
884   continue
999   return
      end
c
```

```
c ***********************************************************
c     CHINSSL
c     fault displacements for chinnery strike slip fault model
c     called from inverse
c
c     ul,u2,u3 are the components of the displacement vector u(k)
c
      subroutine chinssl (u,x1,x2,x3,p1,p3,ul,u2,u3,e1,e2,e12)
      r = sqrt((x1-p1)**2+x2**2+(x3-p3)**2)
      rp = r + p3
      f = 0.0
      if(r.lt.1.e9) f =(3.*r*rp-(3.*r+4.*p3)*(3.*r+p3))/(rp*r**3)
      e1=(u/25.1328)*(x2/rp**2)*(1.-(x1-p1)**2*f)
      e2=(u/25.1328)*(x2/rp**2)*(1. -2.*(3.*r+4.*p3)/r-x2**2*f)
c     e12= (u/50.2656)*(((x1-p1)/r)*(4.*p3/(x2**2+p3**2)-(7.*r+8.*p3)
c     1 /rp**2-x2**2*r*f/rp**2) +((x1-p1)/rp**2)*(1.-x2**2*f))
          ud = u/50.2656
          x1mp1 = x1 - p1
          x1p1dr = x1mp1/r
          x2sq = x2**2
          p3sq = p3**2
          x2p3 = x2sq + p3sq
          r78p3 = 7. *r + 8. *p3
          p3xp = p3/x2p3
          p3xpt4 = 4. * p3xp
          rpsq = rp**2
          r78pdr = r78p3/rpsq
          fdrpsq = f/rpsq
          x2sqr = x2sq * r
          xrfrp = x2sqr * fdrpsq
          x1p1rp = x1mp1 / rpsq
          x2sqf = x2sq * f
          x2sq1 = 1. - x2sqf
          y1 = p3xpt4 - r78pdr
          y2 = y1 - xrfrp
          y3 = x1p1rp * x2sq1
          y4 = x1p1dr * y2
          y5 = y4 + y3
          e12 = ud * y5
       u2 = (u/25.1328)*(alog(rp) +p3/rp -x2**2*(3.*r+4.*p3)/(r*rp**2))
       u3=(u/12.5664)*x2*(rp+p3)/(r*(r+p3))
       if (p3.eq.0.) p3=.00001
       ul = (u/25.1328)*(-x2*(x1-p1)*(3.*r+4.*p3)/(r*rp**2)
      1 +4.*atan(x2*r/(p3*(x1-p1))))
       return
       end
c
```

```
c ***************************************************************************
c OUTSG
c
c Converts a grid file to Standard Grid Format for color plotting.
c
      subroutine outsg(iter,ncol,nrow,xo,dx,yo,dy,array)
c
      character*10  line, ques
      parameter (line = '(/,1x,a)', ques = '(/,1h$,a)')
c
      character filename*50, fileroot*50, idroot*50, id*56, pgm*8
      character numb*4
      real array(*), grid(500,500)
c
      parameter (iout=22)
      common/outsgntimes/ ntimes
      ntimes = ntimes + 1
c
c.....Get info
      if(ntimes.eq.1) then
        print ques, ' Give root for SG filename:  '
        read '(a)', fileroot
        print ques, ' Give title (45 chars):  '
        read '(a)', idroot
        pgm = 'greform '
        dummy = 0.0
        nz = 1
        iprj = 0
        cm = 0.0
        bl = 0.0
      endif
c
c.....Open files.
      write(numb,'(i4)') iter
      call ljust(numb)
      filename = fileroot(1:lentrue(fileroot))//numb(1:lentrue(numb))
     &             //'.sg'
      open(iout,file=filename,form='unformatted',status='new')
c
c.....Convert array to Standard Grid form
      index=0
      do 30  i = 1,ncol
      do 30  j = 1,nrow
      index=index + 1
 30   grid(i,nrow-j+1) = array(index)
c
c.....Write out Standard Grid
      id = idroot(1:lentrue(idroot))//'; iter = '//numb(1:lentrue(numb))
      call wrheader(iout,id,pgm,ncol,nrow,nz,xo,dx,yo,dy,iprj,cm,bl)
      do 50 j=1,nrow
 50   write(iout) dummy,(grid(i,j), i=1,ncol)
      close(iout)
      return
      end
```

```
c ***************************************************************************
c LENTRUE
c called by GETINT, OUTSG
c
      integer function lentrue(string)
c     Gives position of last non-blank, non-tab, non-null
c      character in string.
c      Returns 0 if no such beast in string
c     Programmed by R.Simpson - U.S.G.S.
c
      character*(*) string
      character*1 blank, tab, null
      parameter (blank=' ', tab=char(9), null=char(0))
c
      lentrue=0
c
      do 100 i=len(string),1,-1
      if(string(i:i).ne.blank.and.
     &   string(i:i).ne.tab.and.
     &   string(i:i).ne.null) then
            lentrue=i
            return
      endif
 100  continue
c
      return
      end
c
c ***************************************************************
c LJUST
c called by GETINT, OUTSG
c
      subroutine ljust(string)
c     Left justifies a string by eliminating blanks, tabs, nulls
c      at left end of string.
c     Programmed by R.Simpson - U.S.G.S.
c
      character*(*) string
c
      ifirst=leftend(string)
c
      if(ifirst.le.1) then
        return
c
      else
        l2=len(string)-(ifirst-1)
        string(1:l2)=string(ifirst:ifirst+l2-1)
        string(l2+1:len(string))=' '
      endif
c
      return
      end
```

```
c ***************************************************************
c WRHEADER
c called by OUTSG
c
      subroutine wrheader(unit,id,pgm,ncol,nrow,nz,
     & xo,dx,yo,dy,iproj,cm,bl)
c
      character id*56, pgm*8
      integer unit
c
      write(unit) id,pgm,ncol,nrow,nz,xo,dx,yo,dy,iproj,cm,bl
c
      return
      end
c
c ***************************************************************
c LEFTEND
c called by LJUST
c
      integer function leftend(string)
c     Gives position of first non-blank, non-tab, non-null
c      character in string.
c      Returns 0 if no such beast in string.
c     Programmed by R.Simpson - U.S.G.S.
c
      character*(*) string
      character*1 blank, tab, null
      parameter (blank=' ', tab=char(9), null=char(0))
c
      leftend=0
c
      do 100 i=1,len(string)
      if(string(i:i).ne.blank.and.
     &   string(i:i).ne.tab.and.
     &   string(i:i).ne.null) then
            leftend=i
            return
      endif
 100  continue
c
      return
      end
c
```