Computers & Geosciences Vol. 17, No. 1, pp. 1–21, 1991
Printed in Great Britain. All rights reserved

0098-3004/91 $3.00 + 0.00
Copyright © 1991 Pergamon Press plc

VARIATM—A FORTRAN PROGRAM FOR OBJECTIVE ANALYSIS OF PSEUDOSTRESS WIND FIELDS USING LARGE-SCALE CONJUGATE-GRADIENT MINIMIZATION

DAVID M. LEGLER[1] and I. M. NAVON[2]

[1]Mesoscale Air–Sea Interaction Group, Mail Stop B-174, 012 Love, Florida State University, Tallahassee, FL 32306-3041 and [2]Department of Mathematics and Supercomputer Computations Research Institute, Florida State University, Tallahassee, FL 32306-4052, U.S.A.

(Received 28 February 1989; revised 7 March 1990)

Abstract—A FORTRAN computer program is presented and documented which implements a new approach to objective analysis of pseudostress data over the Indian Ocean. (A pseudostress vector is defined as the wind components multiplied by the wind magnitude.)

This method is a direct large-scale minimization approach of a cost functional expressed as a weighted sum of lack of fit to data as well as constraints on proximity to original observations and climatology, on a smoothing parameter and on kinematic equivalence to climatological patterns. Each of the constraints was weighted by selected coefficients controlling how closely the minimizing analysis fits each type of data or constraint.

The functional operates on 7330 variables (i.e. two wind components at each grid location) and was minimized using a highly efficient memoryless quasi-Newton-like conjugate-gradient method. Use of an independent subjective analysis of the same data provide for a direct quantitative comparison and confirm the adequacy of the objective analysis. This scheme now has been adopted operationally to generate monthly average pseudostress wind values on a 1°-grid over the Indian Ocean.

Key Words: Unconstrained minimization, Objective analysis, Wind Stress, Variational techniques, Indian Ocean.

INTRODUCTION

The variational analysis method allows us to combine information originating from a variety of sources by minimizing the lack of fit to the various sources.

Variational analysis methods for objective analysis of meteorological fields were proposed first by Sasaki (1955, 1958). Variational analysis methods also were used by Holl and Mendenhall (1971) and Holl, Cuming, and Mendenhall (1979) for blending meteorological fields.

Hoffman (1982, 1984) used a direct minimization technique to remove aliasing ambiguity of the SEASAT satellite scatterometer winds.

In this paper, we present the code used to produce monthly average pseudostress values on a 1° mesh over the entire Indian basin, 30°S–28°N, 30°E–120°E.

Pseudostress is defined as the magnitude of the wind times its components, that is

$$\tau_x = u\,|V|, \qquad \tau_y = v\,|V| \tag{1}$$

where u and v are the eastward and northward components of the wind respectively, and V is the wind magnitude.

In our approach here, based on work of Legler, Navon, and O'Brien (1989) and Navon and Legler (1987) we use a variational analysis method to minimize an objective cost functional F, which is a measure of various lacks of fit. The definition of the cost functional, F, is problem dependent and involves knowledge about the nature of the expected error of the data. A proper specification of the cost functional which is defined as a weighted sum of lack of fits to data and constraints is essential for obtaining a satisfactory solution of the objective analysis problem. The results have been used in forcing the ocean circulation model of Luther and O'Brien (1985) for the years 1977–1985. Results from this ocean model from the fall of 1985 have been validated by comparing them to collocated U.S. Navy bathythermograph and NOAA satellite data (Simmons and others, 1988).

The outline of this paper is the following. The data used and the variational cost function, F, are described in the first section of this paper. The CONMIN conjugate-gradient method used for carrying out the unconstrained minimization is detailed in the second section of the paper. The final section is devoted to the VARIATM program code and its implementation with real data sets for the Indian Ocean.

COST FUNCTIONAL

The purpose of the analysis is to obtain a high-quality monthly average representation of the winds over the Indian Ocean regime. In this study, the only information available will be (a) ship report averages on a 1° resolution mesh and (b) a 60-yr pseudostress climatology based on Hellerman and Rosenstein (1983) which was formed by averaging 60 yr of ship wind reports into calendar month means. The ship reports are averaged into boxes in the following way, all the ship wind observations for the analysis month as reported from merchant ships as well as from scientific cruises and meteorological buoys are first collected and screened for incorrect values (Fig. 1). Typically <2% of the observations are removed in these schemes. The remaining observations (typically about 10,000–20,000 observations in the region of interest) then are converted to pseudostress and filtered according to expected means. The resulting data are averaged within each 1° square and any data voids are filled using simple bilinear interpolation.

The first step in implementing direct minimization is designing the cost functional which will be minimized. It will be a measure of lack of fit of the data according to certain prescribed conditions which may be dynamically or statistically motivated. We know from climatology the wind pattern should be "smooth". Thus some measure of roughness and some measure of lack of fit to climatology should be included in the cost functional.

The key ingredients in our objective scheme are the inclusion of two kinematic constraints into the cost functional to be minimized. We choose to require the analysis to be similar to the curl and divergence of the climatology as well as to the climatology itself.

The cost functional, F, which is used to determine objectively derived monthly maps of pseudostress is defined as follows:

$$\begin{aligned} F = {} & \frac{\rho}{L^2}\sum_x\sum_y[(\tau_x + \tau_{x_0})^2 + (\tau_y - \tau_{y_0})^2] \\ & + \frac{\gamma}{L^2}\sum_x\sum_y[(\tau_x - \tau_{x_c})^2 + (\tau_y - \tau_{y_c})^2] \\ & + L^2\lambda\sum_x\sum_y[(\nabla^2(\tau_x - \tau_{x_c}))^2 \\ & + (\nabla^2(\tau_y - \tau_{y_c}))^2] + \beta\sum_x\sum_y[\nabla\cdot(\boldsymbol{\tau} - \boldsymbol{\tau}_c)]^2 \\ & + \alpha\sum_x\sum_y[\hat{k}\cdot\nabla\times(\boldsymbol{\tau} - \boldsymbol{\tau}_c)]^2 \end{aligned} \quad (2)$$

where τ_x, τ_y are the resultant eastward and northward pseudostress components; τ_{x_0}, τ_{y_0} are the components of the 1° mean values determined by the ship wind reports; τ_{x_0}, τ_{y_0} are the components of the pseudostress climatology; $\boldsymbol{\tau}$, $\boldsymbol{\tau}_c$ are the resultant and climatology pseudostress vectors respectively; and L is a length scale (chosen to be 1° lat) which makes all terms uniform dimensionally, and scales them to the same order of magnitude. The coefficients (actually weights) ρ, γ, λ, β, and α control (i.e. they weight the component of the penalty function) how closely the direct minimization fits each constraint (lack of fit).

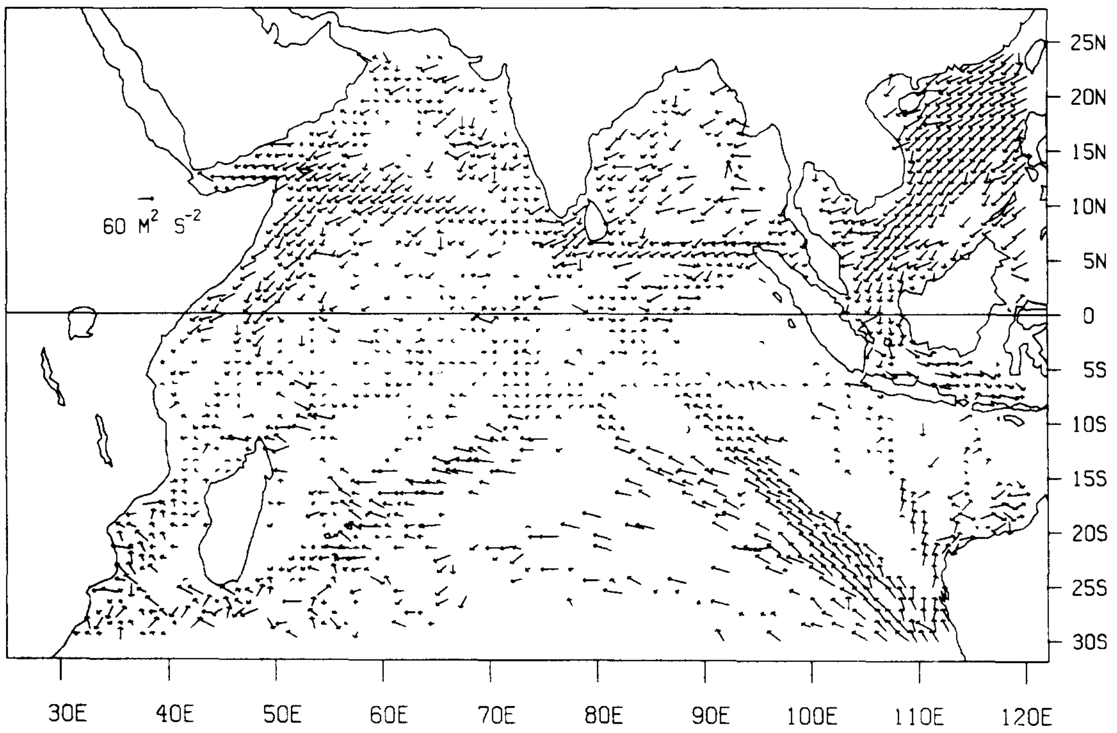

Figure 1. Marine wind observations (first converted to pseudostress—see text) during December 1988 are shown here after being filtered and binned on 1° grid. Data void regions indicate there were missing observations in those locations. Vector lengths indicate magnitude.

The first term of the functional expresses the proximity to the original (input) data. Because one of the weights is arbitrary, in this study we shall set ρ to unity. The second term is concerned with the closeness of fit to the climatological value for that month. A higher value of its corresponding weight leads to a closer approximation to the climatological value. The third term is a measure of the data roughness, and controls the "radius of influence" of an anomaly in the input winds or in the climatological values. It can be termed a "smoothing term" or a "penalty function".

The last two terms are the boundary layer kinematic terms. They force the results to be comparable with climatology, but not in the direct sense. They control the degree to which the divergence and curl of the resulting vector field approximate the kinematics of the climatology. The five terms of the cost functional

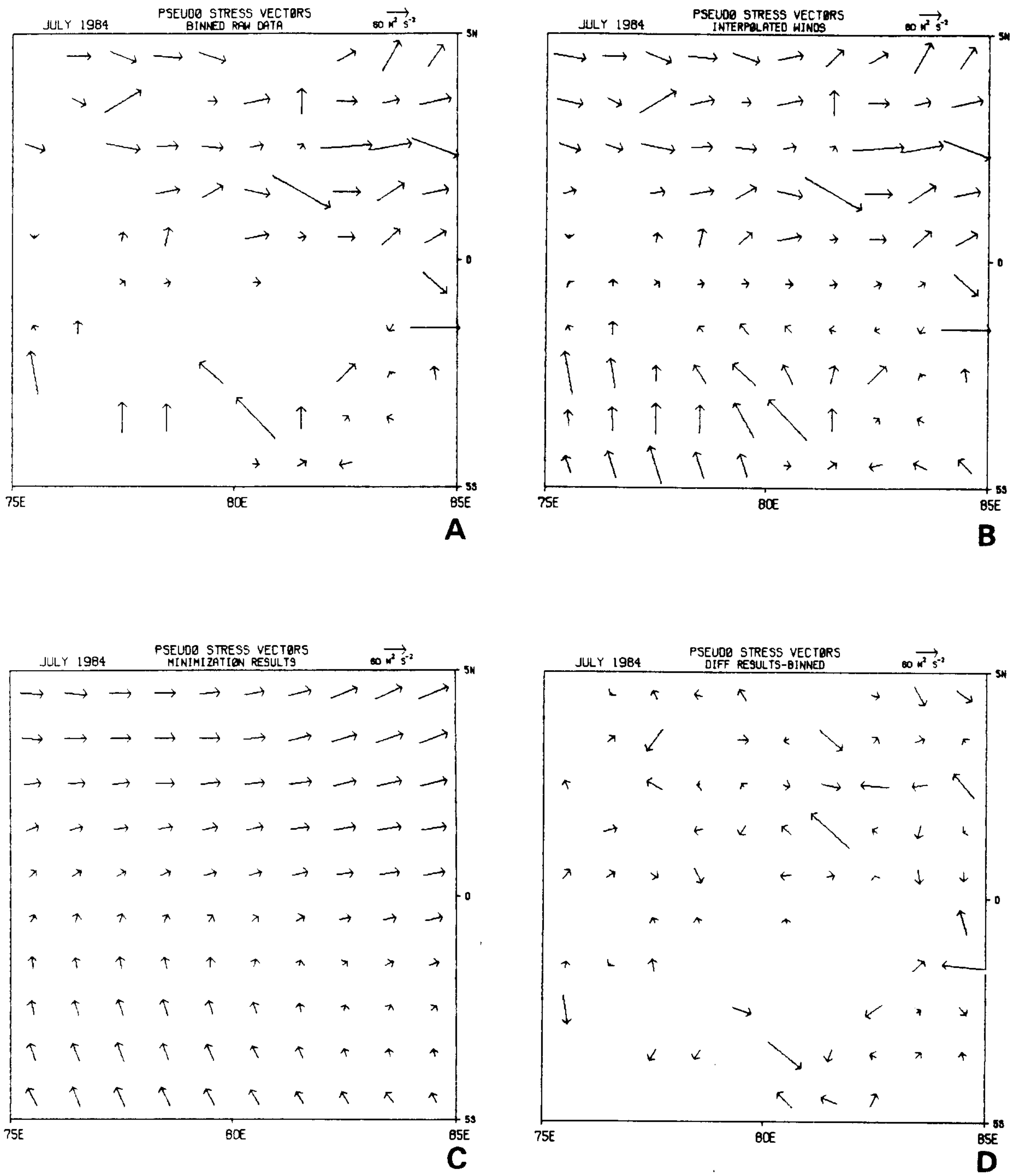

Figure 2. Four stages of data analysis in sample region for July 1984. A—Filtered and binned pseudostress values; B—data field after interpolation has been applied to fill data-void regions; C—results of variational analysis using VARIATM; D—vector difference: minimization results (Fig. 2C)—binned data field (Fig. 2A).

address some of the possible constraints. Other possibilities include time evolution constraints, kinetic energy constraints, pressure gradient terms.

The process of calculating the results is demonstrated in Figures 2 and 3. A selected region in the Indian Ocean is shown in detail in Figure 2A, the ship wind reports have been averaged into 1° boxes, thus some boxes have no data. These voids are filled using bilinear interpolation (Fig. 2B). The result of the minimization (Fig. 2C) and the difference between Figures 2C and 2A (Fig. 2D) indicate satisfactory and expected results.

The manner of selecting the optimal weights for each lack of fit is not addressed in this paper. From experimental results in Legler, Navon, and O'Brien (1989), small variations in the weights for the derivative terms (smoothness, divergence, and curl) had little effect on the results. The second weight, γ, was critical, for it balanced the overall magnitude of the results between the climatological norms and the ship reports (usually of larger magnitude). The weights can be thought of as empirically determined tuning parameters. These weights could be selected by objective means: in theory the method of generalized cross validation (Wahba and Wendelberger, 1980) could be used, but the computation would be impractical with this size data set. In addition, cross validation requires "valid" data, something which is difficult to assess. The adjoint model optimal control technique (Cacuci, 1981; Hall and Cacuci, 1983; LeDimet and Talagrand, 1986; Talagrand, 1985) can aid in a sensitivity study of the critical tuning parameters. None of these objective methods could tell us what the "correct" values of the weights should be because the "correct" solution is not known. Instead, in this study comparisons to independent analyses were used to determine appropriate values (Legler, Navon, and O'Brien, 1989).

Only the 3665 points located over the ocean were included in the direct minimization process, and because a τ_x and a τ_y must be varied at each point, the cost functional included a total of $N = 7330$ variables.

CONJUGATE GRADIENT LARGE-SCALE UNCONSTRAINED MINIMIZATION

The conjugate gradient method for solving large-scale unconstrained, nonlinear minimization problems has been shown to be efficient both from the computational complexity viewpoint as well from the storage requirements viewpoint (Navon and Legler, 1987). The subroutine CONJ is a modified version of the Beale restarted memoryless quasi-Newton algorithm developed by Shanno (1978a, 1978b) and documented by Shanno and Phua (1980).

The methods requires $7N$ single/double precision words of working storage and offers the option of two methods for determining the local minimum of a function of N variables: (a) a limited-recovery Beale (1972) restarted quasi-Newton-like conjugate-gradient algorithm and (b) a Broyden–Fletcher–Goldfarb–

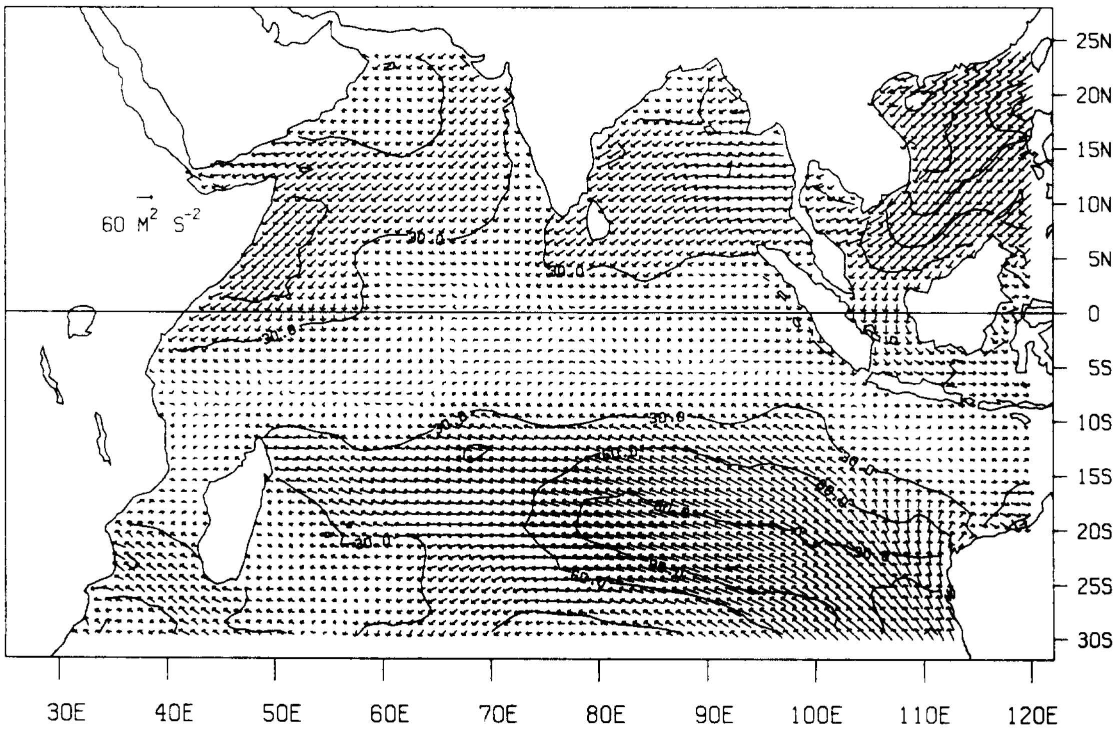

Figure 3. Results of variational analysis using VARIATM for December 1988. Contours of equal magnitude are drawn with contour interval of $30\,m^2\,sec^{-2}$. Vectors with magnitude $> 75\,m^2\,sec^{-2}$ are truncated to $75\,m^2\,sec^{-2}$.

Shanno (BFGS) (Luenberger, 1984, p. 268) quasi-Newton method which requires the storage of the Hessian ($N \times N$) matrix of the objective function—an option which is not feasible for large-scale problems because of memory limitations.

The different steps implementing CONJG are the following:

(i) Initialization

The first-guess field, $X_0 = (U_{11}, \ldots, U_{N_1N}, V_{11}, \ldots, V_{N_1N})^T$, (pseudostress) and an initial guess of the Hessian matrix; $H_0 = I$ (the unit matrix) are input to the routine. (It only stores vector updates to the matrix but never the matrix itself.)

Compute

$$f_k = f(X_k)$$
$$g_k = g(X_k) = \nabla f(X_k). \quad (3)$$

CONJG set the initial search direction, S_k, in the direction of steepest descent

$$S_k = -g_k. \quad (4)$$

(ii) Determination of the step-size

In this step an inexact linear search procedure (see Shanno and Phua, 1980) is implemented.

The basic linear search uses Davidon's (1959) cubic interpolation to determine an optimal step-size α_k which satisfies the following two conditions:

$$f(X_k + \alpha_k S_k) \leqslant f(X_k) + 0.0001\alpha_k S_k^T g_k \quad (5)$$

$$|S_k^T g(X_k + \alpha_k S_k)/S_k^T g_k| < 0.9. \quad (6)$$

(iii) Test for convergence

Update X_k by

$$X_{k+1} = X_k + \alpha_k S_k$$
$$f_{k+1} = f(X_{k+1})$$
$$g_{k+1} = g(X_{k+1})$$
$$p_k = X_{k+1} - X_k$$
$$y_k = g_{k+1} - g_k \quad (7)$$

where

X_k = current point estimate of the minimum
g_k = gradient vector evaluated at the current point
S_k = current search direction
X_{k+1} = new estimate point of the minimum
g_{k+1} = the gradient evaluated at $X = X_{k+1}$
S_t = the Beale restart search direction
y_t = The Beale restart gradient difference vector $y_t = g_{t+1} - g_t$.

(iv) Perform the Beale restart according to the Powell (1977) criteria

As we work with a nonlinear problem, there is a loss of conjugacy and the convergence of the conjugate-gradient algorithm slows down unless restarted every N, where N is the number of components in the vector X, steps in the direction of steepest descent $S_k = -g_k$. Powell (1977) proposed to use Beale's (1972) restart method whenever

(a) The conjugate gradient iteration k is a multiple of N and/or

(b) $|g_{k+1}^T g_k| \geqslant 0.2\|g_{k+1}\|^2 \quad (8)$

where $\| \;\|$ is the Euclidean norm.

This method has been proven to be computationally more efficient (Shanno, 1978a) if either of these two conditions holds. Compute a restart search direction by

$$S_{k+1} = \gamma g_{k+1} - \left[I + \frac{\gamma y_k^T y_k}{p_k^T y_k} \cdot \frac{p_k^T g_{k+1}}{p_k^T y_k} - \frac{\gamma y_k^T g_{k+1}}{p_k^T y_k}\right] p_k$$
$$+ \frac{\gamma p_k^T g_{k+1}}{p_k^T y_k} y_k \quad (9)$$

where

$$\gamma = \frac{p_k^T y_k}{y_k^T y_k}.$$

one then gets $p_t = S_k$, $y_t = y_k$ and goes to step (ii).

(v) Compute a new search direction using the 2-step memoryless BFGS method (Shanno, 1978a)

$$S_{k+1} = -\hat{H}_k g_{k+1} + \frac{p_k^T g_{k+1}}{p_k^T y_k} \hat{H} y_k$$
$$-\left(1 + \frac{y_k^T H_k y_k}{p_k^T y_k} \cdot \frac{p_k^T g_{k+1}}{p_k^T y_k} - \frac{y_k^T \hat{H}_k g_{k+1}}{p_k^T y_k}\right) p_k. \quad (10)$$

Here $\hat{H}_k$ is an approximation of the inverse Hessian of f, where only two rank-two matrix updates of the initial $H_0 = I$ (unit matrix) are required, that is no matrix storage, and the vectors $\hat{H}_k g_{k+1}$ and $\hat{H}_k g_k$ are defined by

$$\hat{H}_k g_{k+1} = \frac{p_t^T y_t}{y_t^T y_t} g_{k+1} - \frac{p_t^T g_{k+1}}{y_t^T y_t} y_t$$
$$+ \left(\frac{p_t^T g_{k+1}}{p_t^T y_t} - \frac{y_t^T g_{k+1}}{y_t^T y_t}\right) p_t \quad (11)$$

and

$$\hat{H}_k y_k = \frac{p_t^T y_t}{y_t^T y_t}\left(\frac{y_k p_t^T y_k}{y_t^T y_t} y_t + \frac{2 p_t^T y_k}{p_t^T y_t} - \frac{y_t^T y_k}{y_t^T y_t}\right) p_t \quad (12)$$

where y_t and p_t are values obtained at a restart step.

In our algorithm convergence is determined to have occurred if

$$\|g\| \leqslant \|g_0\| \epsilon \quad (13)$$

where $\| \;\|$ is the Euclidean norm and ϵ is user supplied. Here g_0 is the initial gradient of the functional f.

We need $\epsilon = 10^{-2}$ and for the present problem CONJ converged within 20 iterations.

PROGRAM VARIATM

This program illustrates the use of the quasi-Newton-like conjugate gradient method applied to the problem of determining the unconstrained minimum of the large scale cost functional F given a first guess field, an appropriate climatology, and selected weights. In the program VARIATM, the sequence of events is to set some variables, read in the first guess field, read in the climatology, then minimize the cost functional by calling the main subroutine, VARY. Upon return from VARY, the results are printed to an output file. Diagnostic output is printed to a file throughout the process.

The main program VARIATM sets memory space aside for the arrays of x and y components of the wind values. There are 94 locations in the east–west direction and 58 in the north–south direction. Because in this array there are locations over land (where there are no wind reports) these locations have as their values, 999. The array UV holds the results of the current minimization in the iteration. The array UVO contains the first guess field, and the array UVC contains the climatology field.

To make use of this program, it will be necessary to change the input files of course, and also the subroutine FUNCT which evaluates the functional as well as its gradient. The minimizer (a large-scale unconstrained local minimization procedure, usually a conjugate-gradient or limited-memory quasi-Newton) can be situated from any standard mathematical package (the one used here has a machine specific parameter, FACC, i.e. this accuracy parameter FACC indicates the smallest number for which $1.000 + \text{FACC} \neq 1.000$). CONJG was provided courtesy of Shanno and Phua (1980).

Acknowledgments—Support for David M. Legler was provided by NOAA under grant NA84AA-D-00049 and under JPL contract OT-2-3172-592. Support for I. M. Navon was provided by the Supercomputer Research Institute which is supported by contract number DE-FC05-85ER250000 from the Department of Energy. Many thanks to Rita Kuyper who cheerfully typed the manuscript.

REFERENCES

Beale, E. M., 1972, A derivation of conjugate-gradients, *in* Lootsma, F. A., ed., Numerical methods for non-linear optimization: Academic Press, London, p. 39–43.

Cacuci, D. G., 1981, Sensitivity theory for non-linear systems. I: Nonlinear functional analysis approach: Jour. Math. Phys., v. 22, no. 12, p. 2794–2802.

Davidon, W. C., 1959, Variable metric methods for minimization: A.E.C., Research and Development Report ANL-5990, Argonne National Laboratory, Argonne, Illinois, November 1959, 27 p.

Hall, M. C. G., and Cacuci, D. G., 1983, Physical interpretation of the adjoint functions for sensitivity analysis of atmospheric models: Jour. Atmos. Sci., v. 40, no. 10, p. 2537–2546.

Hellerman, S., and Rosenstein, M., 1983, Normal monthly wind stress over the world ocean with error estimates: Jour. Phys. Ocean., v. 13, no. 7, p. 1093–1104.

Hoffman, R. N., 1982, SASS wind ambiguity removal by direct minimization: Mon. Wea. Rev., v. 110, no. 3, p. 434–445.

Hoffman, R. N., 1984, SASS wind ambiguity removal by direct minimization. Part II: Use of smoothness and dynamical constraints: Mon. Wea. Rev., v. 112, no. 9, p. 1829–1852.

Holl, M. M., Cuming, M. J., and Mendenhall, B. R., 1979, The expanded ocean thermal-structure analysis system: A development based on the fields by information blending methodology: Final Report, MII Project M-241 (Contract N00014-79-C-0236). (NTIS AD A076 534; NASA 8N17661), unpaginated.

Holl, M. M., and Mendenhall, B. R., 1971, Fields by information blending. Sea-level pressure version: Final Report. Project M-167. Contract No. N66314-70-C-5225. Fleet Numerical Weather Central-Meteorology International Incorporated, Monterey, California, 6 p.

LeDimet, F. X., and Talagrand, O., 1986, Variational algorithms for analysis and assimilation of meteorological observations: Theoretical aspects: Tellus, v. 38a, no. 2, p. 97–110.

Legler, D. M., Navon, I. M., and O'Brien, J. J., 1989, Objective analysis of pseudo-stress over the Indian Ocean using a direct minimization approach: Mon. Wea. Rev., v. 117, no. 4, p. 709–720.

Luenberger, D. G., 1984, Linear and non-linear programming (2nd ed.): Addison-Wesley, Reading, Massachusetts, 491 p.

Luther, M. E., and O'Brien, J. J., 1985, A model of the seasonal circulation in the Arabian Sea forced by observed winds: Prog. Oceanog., v. 14, p. 353–385.

Navon, I. M., and Legler, D. M., 1987, Conjugate-gradient methods for large-scale minimization in meteorology: Mon. Wea. Rev., v. 115, no. 8, p. 1479–1502.

Powell, M. J. D., 1977, Restart procedures for the conjugate-gradient method: Mathematical Programming, v. 12, no. 3, p. 241–254.

Sasaki, Y. K., 1955, A fundamental study of the numerical prediction based on the variational principle: Jour. Meteor. Soc. Japan, v. 33, no. 6, p. 262–275.

Sasaki, Y. K., 1958, An objective analysis based on the variational method: Jour. Meteor. Soc. Japan, v. 36, no. 3, p. 77–88.

Shanno, D. F., 1978a, Conjugate-gradient methods with inexact searches: Math. Operations Res., v. 3, no. 3, p. 244–256.

Shanno, D. F., 1978b, On the convergence of a new conjugate gradient method: SIAM Jour. Num. Anal., v. 15, no. 6, p. 1247–1257.

Shanno, D. F., and Phua, K. H., 1980, Remark on algorithm 500—a variable method subroutine for unconstrained nonlinear minimization: ACM Trans. on Mathematical Software, v. 6, no. 4, p. 618–622.

Simmons, R. C., Luther, M. E., O'Brien, J. J., and Legler, D. M., 1988, Verification of a numerical ocean model of the Arabian Sea: Jour. Geophys. Res., v. 93, no. C12, p. 15437–15454.

Talagrand, O., 1985, Application of optimal control to meteorological problems, *in* Sasaki, Y. K., ed., Variational methods in geosciences: Elsevier Science Publ, Amsterdam, p. 13–28.

Wahba, G., and Wendelberger, J. 1980, Some new mathematical methods for variational objective analysis using splines and cross-validation: Mon. Wea. Rev., v. 108, no. 8, p. 1122–1143.

APPENDIX

Program Listing

```
      PROGRAM variat
      INTEGER ounit
      CHARACTER resfil,fgfile,outfil,climfil
c
      PARAMETER (resfil='variat.res',fgfile='aug88fg',outfil='varyout',
     +          climfil='climat.fil',wgtno=1.0,wgtpo=1.07,nx=94,ny=58)
c
      COMMON /bounds/spval,wgt(nx,ny),ounit
      COMMON /data/uv(nx,ny,2),uvc(nx,ny,2),uvo(nx,ny,2),kount(nx,ny)
      COMMON /params/rho,gcof,alpha,beta,phi,dx,dy,dl,iter,ifun
      COMMON /spherc/clat(ny),radius
c
c****************************************************************
c
c     this program reads in the monthly first guess data(tape1 - first
c     guess) and the monthly climatology(tape2) and submits it for
c     the objective analysis scheme which will variate the winds
c     to best fit the prescribed  characteristics.  the routine will
c     use a conjugate gradient technique to find the minimun
c     of the functional  f.
c
c     version 0.1         legler    2-12-86
c     version 3.0    version for publication in computers in geosciences
c     version 4.0    version resubmitted to computers in geosciences
c                                 1-23-90
c
c****************************************************************
c     clat is cosine of latitude bands (used in spherical coordinates)
c     dl is the scaling length scale...
c     dl is chosen arbitrarily
c     dx,dy are spatial distance between two points in grid space
c     ifun is number of function calls
c     iter is the number of the iteration
c     array kount holds the places where the gradient can be found.
c     nx and ny are number of x and y direction grid points
c     radius of earth (for spherical coordinates)
c     spval is the special value indicating no data at this point
c     the array uv  holds the current results in the iterative process
c     the array uvo holds the first guess wind field
c     the array uvc holds the climatological wind field
c     wgt is array of weights, value depends on the number of wind
c          observations at each grid location
c     n is the number of data points submitted to be varied
c****************************************************************
c
c     set the multipliers here and also other necessary data
c
      DATA one,onelev,othous/1.0,111.1,1000./
      DATA tof/-31.5/
      DATA two,one80/2.0,180.0/
c
c=====================================
c
c     set parameters to be passed in common block
c
      radius = 6.37e06
      rho = 1.0
      phi = 1.5
      gcof = 2.0
      alpha = 30.0
      beta = 30.0
      spval = 999.0
      ounit = 6
c
c====================================
c
      dx = one*onelev*othous
      dy = dx
c
      dl = one*dx
c
      iter = -1
```

```
c
      raddeg = (asin(one)*two)/one80
c
      DO 10 j = 1,ny
          clat(j) = cos((tof+float(j-1))*raddeg)
   10 CONTINUE
c
c     open a file for collecting the diagnostic output
c
      OPEN (ounit,file=outfil,status='new',form='formatted')
c
      WRITE (ounit,*) 'the coefficient for the diff to clim is ',phi
      WRITE (ounit,*) 'the coefficient for the diff to orig obs is ',rho
      WRITE (ounit,*) 'the coefficient for the smoothness is ',gcof
      WRITE (ounit,*) 'the coefficient for the divg term is ',beta
      WRITE (ounit,*) 'the coefficient for the vort term is ',alpha
c
c     create output file for program results
c
      OPEN (unit=4,file=resfil,status='new',form='unformatted')
c
      WRITE (4) rho,phi,gcof,alpha,beta
c
c------------------------------------------------------------
c     make the units of each of the terms s**-2 so scale the
c     appropriate terms by this distance scale dl which is
c     initially set to 1 degrees
c
      rho = rho/dl**2
c
      phi = phi/dl**2
c
      gcof = gcof*dl**2
c
c------------------------------------------------------------
c     read in the first guess wind data
c
      OPEN (unit=1,file=fgfile,status='old',form='unformatted')
c
      READ (1,err=50) iyear,month
      READ (1,err=50) uvo
      READ (1,err=40) kount
c
      CLOSE (1)
c------------------------------------------------------------
c
c     if nobs is 1 or less then make weight of uv-uvo =wgtno otherwise wgtpo
c
      DO 30 i = 1,nx
          DO 20 j = 1,ny
              wgt(i,j) = wgtpo
              IF (kount(i,j).LE.1) wgt(i,j) = wgtno
   20     CONTINUE
   30 CONTINUE
c
      WRITE (ounit,*) 'the year and month of this run are ',iyear,month
c
c------------------------------------------------------------
c     read in the appropriate climatological month
c
      OPEN (unit=2,file=climfil,status='old',form='unformatted')
c
c     now read in the desired month's climatology
c
      READ (2) monc,uvc
c
      CLOSE (2)
c
c-------------------------------------------------------------------
c     call the variational method main subroutine
c
      CALL vary
c
c-------------------------------------------------------------------
c     write out the results of the objective analysis
```

```
c
      WRITE (4) iyear,month
      WRITE (4) uv,uvo,uvc
c
      CLOSE (4)
c-----------------------------------------------------------------
      STOP 'after successful run...'
c
c     read error messages
c
   40 CONTINUE
      STOP 'after read error first guess data kount array'

   50 CONTINUE
      STOP 'after read error first guess data'
c
      END
c
c
c*********************************************************************
c
c
      SUBROUTINE funcij(i,j,ifg)
      INTEGER           ounit
      PARAMETER         (nx=94,ny=58)
      COMMON            /bounds/spval,wgt(nx,ny),ounit
      COMMON            /data/uv(nx,ny,2),uvc(nx,ny,2),uvo(nx,ny,2),
     +                  kount(nx,ny)
c
c-------      -------------      -------------      -------------------
c
c     determine if at the location i,j the functional can be evaluated...
c
c     i,j are the indices (location) in the array that are to be checked
c                   to see if the functional can be evaluated there
c     ifg is a flag, it will have value 999 if the functional can be evaluated
c                                     0 if the functional cannot be eval
c                   at point i,j
c
c-------      -------------      -------------      -------------------
c
      ifg = 999
c
      IF (uv(i,j,1).EQ.spval) RETURN
c
      IF (uv(i+1,j,1).EQ.spval) RETURN
      IF (uv(i-1,j,1).EQ.spval) RETURN
      IF (uv(i,j+1,1).EQ.spval) RETURN
      IF (uv(i,j-1,1).EQ.spval) RETURN
c
      ifg = 0
c
      RETURN

      END
c
c
c**********************************************************************
c
c
      SUBROUTINE funct(n,x,f,g)
      INTEGER           ounit
      REAL              uc,uh,flap,divg,vort
      PARAMETER         (nx=94,ny=58)
      COMMON            /bounds/spval,wgt(nx,ny),ounit
      COMMON            /data/uv(nx,ny,2),uvc(nx,ny,2),uvo(nx,ny,2),
     +                  kount(nx,ny)
      COMMON            /params/rho,gcof,alpha,beta,phi,dx,dy,dl,iter,
     +                  ifun
      COMMON            /spherc/clat(ny),r
      DIMENSION         x(n),g(n)
      EXTERNAL          uc,funcij,uh,flap,divg,vort
c
c-------      -------------      -------------      -------------------
```

```
c     this is the user supplied function for calculating the function
c     that is to be minimized and also the gradient of that function
c     at each point for input into subroutine conjg (a conjugate
c     gradient method for finding the minimum of a function...)
c     legler       feb 18, 1986....
c
c     n is the size of the single array that is the combined east-west and
c       north-south values of uv.  in this case it is 7330.
c     x is array of current values of the resultant winds
c     g is array of gradient values
c     dell is angular distance between grid points
c
c-------       -------------       -------------       ------------------
c
      DATA             one,two,one80/1.0,2.0,180.0/
c
      raddeg = (asin(one)*two)/one80
      dell = one*raddeg
c
      n2 = n/2
      nxm2 = nx - 2
      nym2 = ny - 2
c
      nxml = nx - 1
      nyml = ny - 1
c
      iter = iter + 1
c
c     set the new values of winds into an array for computations,etc
c     the u components into the first half of the array
c     the v components into the last  half of the array
c
c     the array x is the current values of the resultant winds
c     must put them back into the rectangular array for computing the
c     finite difference approximations
c
      DO 20 i = 1,nx
          DO 10 j = 1,ny
              IF (kount(i,j).EQ.0) GO TO 10
c
              uv(i,j,1) = x(kount(i,j))
              uv(i,j,2) = x(kount(i,j)+n2)
   10     CONTINUE
   20 CONTINUE
c
c     calculate and sum up the function at all points
c
c     the function is this: f=rho*sum(uv-uvo)**2+
c                                phi*sum(uv-uvc)**2+
c                     dl**4* gcof*sum(del**2(uv-uvc))**2+
c                     dl**2*alpha*sum(k dot del x(uv-uvc))**2+
c                     dl**2* beta*sum(del dot (uv-uvc))**2
c     rho,phi,gcof,alpha,beta,dl are set constants
c     uv is results, uvc is climatology,uvo is first guess
c     del is operator,k is vertical component,dot is operator
c
c     set the sum terms to 0: sc=sum climatology terms sm: sum laplacian
c       terms, sd=sum of divergence terms, sv=sum of vorticity terms
c
      sc = 0.0
      sm = 0.0
      sd = 0.0
      sv = 0.0


c
      DO 40 i = 2,nxml
          DO 30 j = 2,nyml



c
c         can f be evaluated here at i,j ???
c         if so, then skip to the next grid point in space
c
```

```
              CALL funcij(i,j,ifg)
              IF (ifg.EQ.999) GO TO 30
c
              sc = sc + rho* (uh(i,j,1)**2+uh(i,j,2)**2+
     +              two*uh(i,j,1)*uh(i,j,2)) +
     +              phi* (uc(i,j,1)**2+uc(i,j,2)**2+
     +              two*uc(i,j,1)*uc(i,j,2))
              sm = sm + gcof* (flap(uc,i,j,1,dell)**2+
     +              flap(uc,i,j,2,dell)**2)
              sd = sd + beta*divg(uc,i,j,dell)**2
              sv = sv + alpha*vort(uc,i,j,dell)**2
c
   30     CONTINUE
   40 CONTINUE
c
c     sum up the pieces of the functional
c
      f = sc + sm + sd + sv
c
c     print out the values of the terms for this iteration
c
      WRITE (ounit,*) 'this is for funct call ',iter
      WRITE (ounit,*) 'sum of obs diff term ',sc
      WRITE (ounit,*) 'sum of smoothness term ',sm
      WRITE (ounit,*) 'sum of divergence term ',sd
      WRITE (ounit,*) 'sum of vorticity term ',sv
c
c-----------------------------------------------------------------
c
c     now compute the gradient of the function...
c
c     uh is function to compute difference between current uv value and
c        original (first-guess) value
c     uc is function to compute difference between current uv value and
c        climatological value
c     flap computes the laplacian of the uc field
c     divg computes the divergence of the uc field
c     vort computes the vertical vorticity of the uc field
c
c-----------------------------------------------------------------
c
c     the gradient of f is made of two pieces, dg/du and dg/dv
c     after writing out all the finite difference approximations
c     for the function, then do the gradient computation.
c     for the gradient calculation as expressed in this code,
c     grid locations surrounding the point i,j have contributions to
c     the gradient at i,j.  the pieces below represent those
c     contributions.  for example, u11 and u11a are the pieces from
c     the point i,j for dg/du.  v01 is contribution of point
c     i-1,j for dg/dv, etc
c
c
c     compute dg/du at 3,3 then up to dg/du at 92,56 then for
c     the rest of the g values do dg/dv
c
c     note that each computation of dg/du and dg/dv requires a part
c     of five (5) evaluations of g
c
c     sum up the sum of squares of g (sg) to determine the norm of the grad
c
      sg = 0.0
c
      DO 60 i = 3,nxm2
          DO 50 j = 3,nym2
c
c         can the gradient be calculated here ??? (hint-check kount)
c         kount read in with first guess field...= number of obs at each point
c
              IF (kount(i,j).EQ.0) GO TO 50
c
c         do gradient dg/du at i,j
c
              u11 = 2.*rho* (uh(i,j,1)+uh(i,j,2)) +
     +              2.*phi* (uc(i,j,1)+uc(i,j,2))
```

```
            u11a = 2.*gcof*flap(uc,i,j,1,dell)*
     +              ((-2./dell**2)+ (clat(j)/dell**2)*
     +              (clat(j+1)* (-1.)-clat(j-1)))/ (r*clat(j))**2
            u21 = 2.*gcof*flap(uc,i+1,j,1,dell)/dell**2/
     +             (r*clat(j))**2
            u21a = 2.*beta*divg(uc,i+1,j,dell)* (-1.)/
     +              (2.*dell*r*clat(j))
            u12 = 2.*gcof*flap(uc,i,j+1,1,dell)*clat(j+1)*clat(j)/
     +             dell**2/ (r*clat(j+1))**2
            u12a = 2.*alpha*vort(uc,i,j+1,dell)*clat(j)/ (2.*dell)/
     +              (r*clat(j+1))
            u01 = 2.*gcof*flap(uc,i-1,j,1,dell)/ (dell*r*clat(j))**2
            u01a = 2.*beta*divg(uc,i-1,j,dell)/ (2.*dell*r*clat(j))
            u10 = 2.*gcof*flap(uc,i,j-1,1,dell)*clat(j-1)*clat(j)/
     +             dell**2/ (r*clat(j-1))**2
            u10a = 2.*alpha*vort(uc,i,j-1,dell)* (-1.)*clat(j)/
     +              (2.*dell)/ (r*clat(j-1))
c
c       now do the dg/dv at i,j ...
c
            v11 = 2.*rho* (uh(i,j,2)+uh(i,j,1)) +
     +             2.*phi* (uc(i,j,2)+uc(i,j,1)) +
     +             2.*gcof*flap(uc,i,j,2,dell)*
     +             ((-2./dell**2)+ (clat(j)/dell**2)*
     +             (clat(j+1)* (-1.)-clat(j-1)))/ (r*clat(j))**2
            v21 = 2.*gcof*flap(uc,i+1,j,2,dell)/ (dell*r*clat(j))**2 +
     +             2.*alpha*vort(uc,i+1,j,dell)* (-1.)/
     +             (2.*dell*r*clat(j))
            v12 = 2.*gcof*flap(uc,i,j+1,2,dell)*clat(j+1)*clat(j)/
     +             (dell*r*clat(j+1))**2 + 2.*beta*divg(uc,i,j+1,dell)*
     +              (-1.)*clat(j)/ (2.*dell*r*clat(j+1))
            v01 = 2.*gcof*flap(uc,i-1,j,2,dell)/ (dell*r*clat(j))**2 +
     +             2.*alpha*vort(uc,i-1,j,dell)/ (2.*dell*r*clat(j))
            v10 = 2.*gcof*flap(uc,i,j-1,2,dell)*clat(j-1)*clat(j)/
     +             (dell*r*clat(j-1))**2 + 2.*beta*divg(uc,i,j-1,dell)*
     +             clat(j)/ (2.*dell*r*clat(j-1))
c
c      add up all the pieces for the gradient of the u-component
c
            g(kount(i,j)) = u11 + u11a + u21 + u21a + u12 + u12a +
     +                      u01 + u01a + u10 + u10a
c
c      now add up all pieces for the gradient of v component
c
            g(kount(i,j)+n2) = v11 + v21 + v12 + v01 + v10
c
c      calculate the norm
c
            sg = sg + g(kount(i,j))**2 + g(kount(i,j)+n2)**2
c
   50    CONTINUE
   60 CONTINUE
c
c      print out the current value of the norm**2...sum(g**2)
c
      WRITE (ounit,9000) sg
c
 9000 FORMAT (/'the norm of grad (sum of the squares of grad) = ',e10.3)
c
      RETURN

      END
c
c
c***********************************************************************
c
c
      SUBROUTINE printo(ier,f)
      INTEGER             ounit
      PARAMETER           (nx=94,ny=58)
      COMMON              /bounds/spval,wgt(nx,ny),ounit
      CHARACTER*40        message(5)
c
      DATA                message/
     +                 'normal termination... the final value of f is '
     +                 ,
     +                 'gradient error check  the final value of f is '
     +                 ,
     +          'search direction on uphill..the final vvalue of f is '
     +                 ,'maxfn exceeded  the final value of f is ',
     +               'function not reducing..the final value of f is '
     +                 /
c
c-------       -------------       -------------       -------------------
c
c     print out the appropriate error condition and final value of f
c
c     ier is returned error code
c     f is final value of the functional
c
c-------       -------------       -------------       -------------------
c
      IF (ier.EQ.0) WRITE (ounit,9000) f,message(1)
      IF (ier.EQ.129) WRITE (ounit,9000) f,message(2)
      IF (ier.EQ.130) WRITE (ounit,9000) f,message(3)
      IF (ier.EQ.1) WRITE (ounit,9000) f,message(4)
      IF (ier.EQ.132) WRITE (ounit,9000) f,message(5)
c
 9000 FORMAT (e12.3,a40)
c
      RETURN

      END
c
c
c*************************************************************************
c
c
      SUBROUTINE vary
      INTEGER ounit
      PARAMETER (nx=94,ny=58)
      PARAMETER (n=7330)
c
      COMMON /data/uv(nx,ny,2),uvc(nx,ny,2),uvo(nx,ny,2),kount(nx,ny)
      COMMON /params/rho,gcof,alpha,beta,phi,dx,dy,dl,iter,ifun
      COMMON /bounds/spval,wgt(nx,ny),ounit
c
      DIMENSION g(n),x(n),w(6*n)
      EXTERNAL funct
c
c-------       -------------       -------------       -------------------
c
c     this subroutine will actually call the minimizing routine which
c     uses for now the conjugate gradient method subroutine conjg
c     note the external function funct
c     legler   2-21-86...
c
c========================================================
c========================================================
c
c     facc is smallest number such that facc+1.0<>facc
c     facc is machine dependent, change as necessary
c
      facc = 1.0e-15
c
c========================================================
c========================================================
c
      iunit = ounit
      iout = 1
      nw = 6*n
      maxfn = 9
c
c
c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c
c     n is the size of the array to be varied
c     x is the current values of the winds in a singularly dimensioned array
c     f is array of the functional values
```

```
c     g is array of gradient values
c     iff is number of function calls made
c     ic  number of iterations...
c     cacc is desired accuracy of the results
c     ier is returned error code
c     maxfn is maximum number of function calls allowed
c     w is array for working space (required to be 6*n) long
c     iout  output desired ? (0=no, otherwise indicates output every iout
c           iteractions)
c     nw   6*n for dimension of the working space
c     iunit  is output unit number
c     facc is estimate of machine accuracy
c     nmeth  indicator for method set to 0 for c-g
c     dfpred initial step size reduction in the functional
c
c     set the x array(current results) to the first guess
c     x is array of current values of the resultant winds
c     uv holds the current (varied) values of the analysis
c
      DO 40 i = 1,nx
          DO 30 j = 1,ny
c
              IF (kount(i,j).EQ.0) GO TO 10
c
              x(kount(i,j)) = uvo(i,j,1)
              x(kount(i,j)+n/2) = uvo(i,j,2)
   10         CONTINUE
c
c     now set the current results (in grid-space) to the first guess
c
              DO 20 k = 1,2
c
                  uv(i,j,k) = uvo(i,j,k)
c
   20         CONTINUE
   30     CONTINUE
   40 CONTINUE
c
c     in order to determine the desired 'accuracy' needed for the
c     conjugate-gradient to quit and return control to this subroutine,
c     initial values of the function and the norm of the gradient
c     are needed
c
      CALL funct(n,x,f,g)
c
c     find the norm of g
c
      sum = 0.0
      DO 50 i = 1,n
          sum = sum + g(i)**2
   50 CONTINUE
c
c-------       -------------       -------------       -------------------
c
c     set the accuracy desired (cacc), the initial decrease of f (dfpred)
c     and the maximum calls allowed of subroutine funct (maxfn)...
c
c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c
      acc = 1.0e-02*sum
      cacc = sqrt(acc)
      dfpred = f/2.5
c
c!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c
c
      CALL conjg(n,x,f,g,iff,ic,cacc,ier,maxfn,w,iout,nw,iunit,facc,
     +           nmeth,funct,dfpred)
c
      iter = ic
      ifun = iff
c
c     print out the meaning of the error code and the final f value
```

```
      CALL printo(ier,f)
c
      RETURN

      END
c
c*************************************************************************
c
c
      FUNCTION uc(i,j,ixy)
      INTEGER      ounit
      PARAMETER    (nx=94,ny=58)
      COMMON       /bounds/spval,wgt(nx,ny),ounit
      COMMON       /data/uv(nx,ny,2),uvc(nx,ny,2),uvo(nx,ny,2),
     +             kount(nx,ny)
c
c-------       -------------       -------------       -------------------
c
c     this is one of the difference operators which for location
c     i,j and component ixy finds the following value
c     uc=(current uv value - climatology uv value)*wgt
c
c     ixy is either 1 (east west component) or 2 (north-south component)
c
c     legler   2-19-86...
c
c-------       -------------       -------------       -------------------
c
      IF (uv(i,j,ixy).EQ.spval) STOP 'uv in tine uc is spval...'
c
      uc = 0.
      IF (uvc(i,j,ixy).EQ.999.) RETURN
c
      uc = (uv(i,j,ixy)-uvc(i,j,ixy))*wgt(i,j)
c
      RETURN

      END
c
c
c*************************************************************************
c
c
      FUNCTION uh(i,j,ixy)
      INTEGER      ounit
      PARAMETER    (nx=94,ny=58)
      COMMON       /bounds/spval,wgt(nx,ny),ounit
      COMMON       /data/uv(nx,ny,2),uvc(nx,ny,2),uvo(nx,ny,2),
     +             kount(nx,ny)
c
c-------       -------------       -------------       -------------------
c
c     this is one of the difference operators which for location
c     i,j and component ixy finds the following value
c     uh = current uv value - original observation value
c
c     legler   2-19-86...
c-------       -------------       -------------       -------------------
c
c
c
      IF (uv(i,j,ixy).EQ.spval) STOP 'uv in tine uh is spval...'
c
      uh = 0.
      IF (uvo(i,j,ixy).EQ.-999.) RETURN
      uh = uv(i,j,ixy) - uvo(i,j,ixy)
c
      RETURN

      END
c
c*************************************************************************
```

```
      FUNCTION divg(fun,i,j,dx)
      REAL          fun
      PARAMETER     (nx=94,ny=58)
      COMMON        /spherc/clat(ny),r
      EXTERNAL      fun
c
c----------         ----------          ----------          -----------
c
c     this is the function to calculate the finite difference form of
c     divergence in spherical coordinates in grid space.  legler  9-26-86
c
c----------         ----------          ----------          -----------
c
      divg = 1./ (r*clat(j)*2.*dx)* (fun(i+1,j,1)-fun(i-1,j,1)+
     +       fun(i,j+1,2)*clat(j+1)-fun(i,j-1,2)*clat(j-1))
c
      RETURN

      END
c
c
c**********************************************************************
c
c
      FUNCTION flap(fun,i,j,ixy,dx)
      REAL          fun
      PARAMETER     (nx=94,ny=58)
      COMMON        /spherc/clat(ny),r
      EXTERNAL      fun
c
c-------      -------------       -------------       ------------------
c
c     this is the function to calculate the laplacian (second order
c     finite difference operator) at a point i,j for the ixy
c     component with the operator fun.
c     this fun operator is one of the difference operators
c     uc (difference of uv - climat) or uh(diff of uv - original).
c     the involved values should not be special values since this is
c     a function called only for those locations found by subroutine
c     findn and stored in kount     legler  2-19-86
c
c     changed to reflect calculation in spherical coordinates
c     legler   9-26-86
c
c-------      -------------       --------------      -------------------
c
      flap = (1./ (r*clat(j)*dx)**2)* (fun(i+1,j,ixy)-2.*fun(i,j,ixy)+
     +       fun(i-1,j,ixy)+clat(j)* (clat(j+1)* (fun(i,j+1,ixy)-fun(i,
     +       j,ixy))-clat(j-1)* (fun(i,j,ixy)-fun(i,j-1,ixy))))
c
      RETURN

      END
c
c
c
      FUNCTION vort(fun,i,j,dx)
      REAL          fun
      PARAMETER     (nx=94,ny=58)
      COMMON        /spherc/clat(ny),r
      EXTERNAL      fun
c
c------       ----------          ----------          ----------
c
c     function to calculate vorticity in spherical coordinates
c     legler   9-26-86
c
c-----        ----------          ----------          ----------
c
      vort = 1./ (r*clat(j)*2.*dx)* (fun(i+1,j,2)-fun(i-1,j,2)-
     +       fun(i,j+1,1)*clat(j+1)+fun(i,j-1,1)*clat(j-1))
c
      RETURN

      END
      SUBROUTINE conjg(n,x,f,g,ifun,iter,eps,nflag,mxfun,w,iout,mdim,
     +                 idev,acc,nmeth,calcfg,f0)
      DIMENSION        x(n),g(n),w(mdim)
      EXTERNAL         calcfg
      LOGICAL          rsw
c
c
c     SUBROUTINE CONJG is provided by Shanno (see Shanno, D.F. and K.H. Phua
c     article: Remark on algorithm 500 a variable method subroutine for
c     unconstrained nonlinear minimization, ACM Transactions on
c     Mathematical Software, 1980, pp.618-622.)
c
c
      alpha = 1.
      iter = 0
      ifun = 0
      ioutk = 0
      nflag = 0
      nx = n
      ng = nx + n
      IF (nmeth.EQ.1) GO TO 10
      nry = ng + n
      nrd = nry + n
      ncons = 5*n
      ncons1 = ncons + 1
      ncons2 = ncons + 2
      GO TO 20

   10 CONTINUE
      ncons = 3*n
   20 CONTINUE
      CALL calcfg(n,x,f,g)
      ifun = ifun + 1
      nrst = n
      rsw = .true.
      dg1 = 0.
      xsq = 0.
      DO 30 i = 1,n
          w(i) = -g(i)
          xsq = xsq + x(i)*x(i)
          dg1 = dg1 - g(i)*g(i)
   30 CONTINUE
      dg = dg1
      gsq = -dg1
c     if(gsq.le.eps*eps*amax1(1.,xsq))return
c
c     new return criteria...
c
      IF (gsq.LE.eps*eps) RETURN
   40 CONTINUE
      fmin = f
      ncalls = ifun
      IF (iout.EQ.0) GO TO 60
      IF (ioutk.NE.0) GO TO 50
      WRITE (idev,9000) iter,ifun,fmin,gsq
   50 CONTINUE
      ioutk = ioutk + 1
      IF (ioutk.EQ.iout) ioutk = 0
   60 CONTINUE
      alpha = alpha*dg/dg1
      IF ((nrst.EQ.1) .OR. (nmeth.EQ.1)) alpha = 1.
      IF (rsw) alpha = abs(f0)/gsq
      ap = 0.
      fp = fmin
      dp = dg1
      dg = dg1
      iter = iter + 1
      step = 0.
      DO 70 i = 1,n
          step = step + w(i)*w(i)
          nxpi = nx + i
          ngpi = ng + i
          w(nxpi) = x(i)
          w(ngpi) = g(i)
```

```
 70 CONTINUE
    step = sqrt(step)
 80 CONTINUE
    IF (alpha*step.GT.acc) GO TO 90
    IF (.NOT.rsw) GO TO 20
    nflag = 2
    RETURN

 90 CONTINUE
    DO 100 i = 1,n
        nxpi = nx + i
        x(i) = w(nxpi) + alpha*w(i)
100 CONTINUE
    CALL calcfg(n,x,f,g)
    ifun = ifun + 1
    IF (ifun.LE.mxfun) GO TO 110
    nflag = 1
    RETURN

110 CONTINUE
    dal = 0.0
    DO 120 i = 1,n
        dal = dal + g(i)*w(i)
120 CONTINUE
    IF (f.GT.fmin .AND. dal.LT.0.) GO TO 160
    IF (f.GT. (fmin+.0001*alpha*dg) .OR.
   +    abs(dal/dg).GT.0.9) GO TO 130
    IF ((ifun-ncalls).LE.1 .AND. abs(dal/dg).GT.eps .AND.
   +    nmeth.EQ.0) GO TO 130
    GO TO 170

130 CONTINUE
    u1 = dp + dal - 3.0* (fp-f)/ (ap-alpha)
    u2 = u1*u1 - dp*dal
    IF (u2.LT.0.) u2 = 0.
    u2 = sqrt(u2)
    at = alpha - (alpha-ap)* (dal+u2-u1)/ (dal-dp+2.*u2)
    IF ((dal/dp).GT.0.) GO TO 140
    IF (at.LT. (1.01*amin1(alpha,ap)) .OR.
   +    at.GT. (.99*amax1(alpha,ap))) at = (alpha+ap)/2.0
    GO TO 150

140 CONTINUE
    IF (dal.GT.0.0 .AND. 0.0.LT.at .AND.
   +    at.LT. (.99*amin1(ap,alpha))) GO TO 150
    IF (dal.LE.0.0 .AND. at.GT. (1.01*amax1(ap,alpha))) GO TO 150
    IF (dal.LE.0.) at = 2.0*amax1(ap,alpha)
    IF (dal.GT.0.) at = amin1(ap,alpha)/2.0
150 CONTINUE
    ap = alpha
    fp = f
    dp = dal
    alpha = at
    GO TO 80

160 CONTINUE
    alpha = alpha/3.
    ap = 0.
    fp = fmin
    dp = dg
    GO TO 80

170 CONTINUE
    gsq = 0.0
    xsq = 0.0
    DO 180 i = 1,n
        gsq = gsq + g(i)*g(i)
        xsq = xsq + x(i)*x(i)
180 CONTINUE
c     if(gsq.le.eps*eps*amax1(1.0,xsq))return
    IF (gsq.LE.eps*eps) RETURN
    DO 190 i = 1,n
        w(i) = alpha*w(i)
190 CONTINUE
```

```
      IF (nmeth.EQ.1) GO TO 330
      rtst = 0.
      DO 200 i = 1,n
          ngpi = ng + i
          rtst = rtst + g(i)*w(ngpi)
200   CONTINUE
      IF (abs(rtst/gsq).GT.0.2) nrst = n
      IF (nrst.NE.n) GO TO 220
      WRITE (idev,*) ' beale restart '
      w(ncons+1) = 0.
      w(ncons+2) = 0.
      DO 210 i = 1,n
          nrdpi = nrd + i
          nrypi = nry + i
          ngpi = ng + i
          w(nrypi) = g(i) - w(ngpi)
          w(nrdpi) = w(i)
          w(ncons1) = w(ncons1) + w(nrypi)*w(nrypi)
          w(ncons2) = w(ncons2) + w(i)*w(nrypi)
210   CONTINUE
220   CONTINUE
      u1 = 0.0
      u2 = 0.0
      DO 230 i = 1,n
          nrdpi = nrd + i
          nrypi = nry + i
          u1 = u1 - w(nrdpi)*g(i)/w(ncons1)
          u2 = u2 + w(nrdpi)*g(i)*2./w(ncons2) - w(nrypi)*g(i)/w(ncons1)
230   CONTINUE
      u3 = w(ncons2)/w(ncons1)
      DO 240 i = 1,n
          nxpi = nx + i
          nrdpi = nrd + i
          nrypi = nry + i
          w(nxpi) = -u3*g(i) - u1*w(nrypi) - u2*w(nrdpi)
240   CONTINUE
      IF (nrst.EQ.n) GO TO 300
250   CONTINUE
      u1 = 0.
      u2 = 0.
      u3 = 0.
      u4 = 0.
      DO 260 i = 1,n
          ngpi = ng + i
          nrdpi = nrd + i
          nrypi = nry + i
          u1 = u1 - (g(i)-w(ngpi))*w(nrdpi)/w(ncons1)
          u2 = u2 - (g(i)-w(ngpi))*w(nrypi)/w(ncons1) +
     +         2.0*w(nrdpi)* (g(i)-w(ngpi))/w(ncons2)
          u3 = u3 + w(i)* (g(i)-w(ngpi))
260   CONTINUE
      step = 0.
      DO 270 i = 1,n
          ngpi = ng + i
          nrdpi = nrd + i
          nrypi = nry + i
          step = (w(ncons2)/w(ncons1))* (g(i)-w(ngpi)) + u1*w(nrypi) +
     +           u2*w(nrdpi)
          u4 = u4 + step* (g(i)-w(ngpi))
          w(ngpi) = step
270   CONTINUE
      u1 = 0.0
      u2 = 0.0
      DO 280 i = 1,n
          u1 = u1 - w(i)*g(i)/u3
          ngpi = ng + i
          u2 = u2 + (1.0+u4/u3)*w(i)*g(i)/u3 - w(ngpi)*g(i)/u3
280   CONTINUE
      DO 290 i = 1,n
          ngpi = ng + i
          nxpi = nx + i
          w(nxpi) = w(nxpi) - u1*w(ngpi) - u2*w(i)
290   CONTINUE
300   CONTINUE
      dg1 = 0.
```

```
    DO 310 i = 1,n
        nxpi = nx + i
        w(i) = w(nxpi)
        dg1 = dg1 + w(i)*g(i)
310 CONTINUE
    IF (dg1.GT.0.) GO TO 320
    IF (nrst.EQ.n) nrst = 0
    nrst = nrst + 1
    rsw = .false.
    GO TO 40

320 CONTINUE
    nflag = 3
    RETURN

330 CONTINUE
    u1 = 0.0
    DO 340 i = 1,n
        ngpi = ng + i
        w(ngpi) = g(i) - w(ngpi)
        u1 = u1 + w(i)*w(ngpi)
340 CONTINUE
    IF (.NOT.rsw) GO TO 380
    u2 = 0.
    DO 350 i = 1,n
        ngpi = ng + i
        u2 = u2 + w(ngpi)*w(ngpi)
350 CONTINUE
    ij = 1
    u3 = u1/u2
    DO 370 i = 1,n
        DO 360 j = 1,n
            ncons1 = ncons + ij
            w(ncons1) = 0.0
            IF (i.EQ.j) w(ncons1) = u3
            ij = ij + 1
360     CONTINUE
        nxpi = nx + i
        ngpi = ng + i
        w(nxpi) = u3*w(ngpi)
370 CONTINUE
    u2 = u3*u2
    GO TO 430

380 CONTINUE
    u2 = 0.0
    DO 420 i = 1,n
        u3 = 0.
        ij = i
        IF (i.EQ.1) GO TO 400
        ii = i - 1
        DO 390 j = 1,ii
            ngpj = ng + j
            ncons1 = ncons + ij
            u3 = u3 + w(ncons1)*w(ngpj)
            ij = ij + n - j
390     CONTINUE
400     CONTINUE
        DO 410 j = 1,n
            ncons1 = ncons + ij
            ngpj = ng + j
            u3 = u3 + w(ncons1)*w(ngpj)
            ij = ij + 1
410     CONTINUE
        ngpi = ng + i
        u2 = u2 + u3*w(ngpi)
        nxpi = nx + i
        w(nxpi) = u3
420 CONTINUE
430 CONTINUE
    u4 = 1.0 + u2/u1
    DO 440 i = 1,n
        nxpi = nx + i
        ngpi = ng + i
        w(ngpi) = u4*w(i) - w(nxpi)
440 CONTINUE
    ij = 1
    DO 460 i = 1,n
        nxpi = nx + i
        u3 = w(i)/u1
        u4 = w(nxpi)/u1
        DO 450 j = 1,n
            ncons1 = ncons + ij
            ngpj = ng + j
            w(ncons1) = w(ncons1) + u3*w(ngpj) - u4*w(j)
            ij = ij + 1
450     CONTINUE
460 CONTINUE
    dg1 = 0.0
    DO 500 i = 1,n
        u3 = 0.0
        ij = i
        IF (i.EQ.1) GO TO 480
        ii = i - 1
        DO 470 j = 1,ii
            ncons1 = ncons + ij
            u3 = u3 - w(ncons1)*g(j)
            ij = ij + n - j
470     CONTINUE
480     CONTINUE
        DO 490 j = 1,n
            ncons1 = ncons + ij
            u3 = u3 - w(ncons1)*g(j)
            ij = ij + 1
490     CONTINUE
        dg1 = dg1 + u3*g(i)
        w(i) = u3
500 CONTINUE
    IF (dg1.GT.0.) GO TO 320
    rsw = .false.
    GO TO 40

9000 FORMAT (10H iteration,i5,20H      function calls,i6/5H f = ,e15.8,
   +        13H g-squared = ,e15.8/)

    END
```