TECHNICAL MEMORANDUM

184

A method for implementing adjoint calculations in the discrete case

R.N. Hoffman, J F. Louis and T. Nehrkorn

Research Department

February 1992

European Centre for Medium-Range Weather Forecasts
Europäisches Zentrum für mittelfristige Wettervorhersage
Centre européen pour les prévisions météorologiques à moyen

ABSTRACT

We present a unified formalism of discrete adjoint calculations. The adjoint formalism is a technique to efficiently calculate the sensitivity of a single measure of the output of a model to a set of control variables. This formalism unifies the calculation of the sensitivity with respect to initial conditions, the sensitivity with respect to model parameters and the forcing of the adjoint model due to the data misfit. This view also results in a straightforward method for writing an adjoint model in practice. Two steps are required. First, the full model is linearized. Second, the linear model is transposed. The first step transforms the nonlinear model (NLM) into the linear tangent model (LTM). The last two steps transform the LTM into the adjoint model. Usually the transposition of the linear model is carried out by first dividing the model into parts (called stages here). In this case the order of the transposed stages must be reversed.

1. INTRODUCTION

The adjoint formalism is a technique to calculate the sensitivity of an objective function with respect to a set of control variables. The objective function is a single scalar measure of the output of a mathematical or numerical model, and the control parameters might be model initial conditions and/or parameters (*Hall et al.*, 1982; *Le Dimet and Talagrand*, 1986; *Thacker and Long*, 1988; *Thacker*, 1988). For numerical models, the chief advantage of the adjoint formalism is efficiency, since one integration of the adjoint is sufficient to compute the derivatives of the objective function with respect to all the parameters and initial conditions. While the adjoint formalism is entirely general, two of the principal applications are meteorological and oceanographic models. One interesting application of the adjoint formalism, which has recently garnered much attention, is to define the initial conditions for numerical weather prediction (*Talagrand and Courtier*, 1987; *Courtier and Talagrand*, 1987; *Lewis and Derber*, 1985; *Derber*, 1987, 1989). In this case, the objective function is a measure of the misfit of the model solution to observations of the atmosphere. This application is particularly challenging, because such models already tax the largest computers and because the analysis of the initial conditions together with the forecast must conform to the operational requirement of timeliness. This work is underway for the models of ECMWF and NMC (*Rabier and Courtier*, 1992; *Thépaut and Courtier*, 1991; *Le Dimet et al.*, 1991; *Zou et al.*, 1991).

In general, the adjoint is defined for a particular linear operator and particular inner products defined on the domain and range of the linear operator. In practical numerical cases, one can always choose to define (as we will in the remainder of this paper) the linear operators as matrices and the inner products as ordinary dot products for real (or complex) numbers. In these cases if A is a linear operator and $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ is the inner product of $\boldsymbol{x}$ and $\boldsymbol{y}$, then the adjoint of A, denoted A^*, is simply the transpose (or complex conjugate transpose) of A, and

$$\langle \boldsymbol{x} \, , \, A\boldsymbol{y} \rangle = \langle A^* \boldsymbol{x} \, , \, \boldsymbol{y} \rangle \, . \tag{1}$$

The purpose of this paper is to define the formalism for the general discrete calculation in such a way as to clarify the practical implementation of such a calculation. This formalism unifies the treatment of tunable parameters and the forcing due to the verification data, with the calculation of the adjoint solution. For the most part this material is implicit in earlier papers on the subject referenced

earlier. However our experience is that a considerable effort is required to bridge the gap between the published accounts to a practical methodology for developing an adjoint for an actual computer model. The audience for this paper are those contemplating coding an adjoint for the first time. A caveat: This paper presents only one possible approach for implementing an adjoint.

It will be seen that implementing the adjoint version of a model involves two almost entirely mechanical steps: First one writes the linearized version of the model, here denoted the linear tangent model (LTM). The LTM must include all perturbations due directly or indirectly to perturbations in those initial conditions and/or parameters under study. The LTM may be represented by a single, possibly very complicated, matrix. The (complex conjugate) transpose of this matrix is then the corresponding adjoint model. Usually, the LTM is considered to be the application of a series of matrices, rather than a single matrix. The adjoint model may then be derived as the transpose of the LTM matrices applied in reverse order. We note here that any type of time marching can be accommodated by the formalism outlined here.

In our notation we distinguish between initial conditions and parameters, because our work involves tuning the parameters of a meteorological forecast model. In other applications the distinction may be less clear. In general, parameters are variables whose values are imprecisely known "constants", while the initial conditions are all other variables needed for a complete specification of the problem. In other words, the initial conditions vary from case to case, while the parameters are constant, at least within the ensemble under study. The separation of initial conditions and parameters is not absolutely necessary in what follows — either set could be empty.

2. THE UNIFIED FORMALISM

We divide the model into a series of calculations, each of which might be simple or complicated. This division of the calculation is arbitrary and the divisions themselves we will call the stages of the calculation or simply stages. A stage might alternatively be defined in terms of time steps, procedures (i.e., subroutine calls) or individual equations (i.e., assignment statements). Ultimately we will apply our results to individual lines of computer code. In this case we may identify *each execution of an assignment statement as a stage*. This distinction - between an assignment statement and its execution - will be important in applying our results. When we derive the linear tangent model (LTM) we will view the full or nonlinear model (NLM) as the execution of a series of assignment statements. The model of course may contain conditional and looping structures, but these may be thought of as just a convenient short hand for specifying a series of assignment statements. Thus, *all control structures of the NLM are inherited with no change by the LTM*. For the LTM to be well defined the NLM should be continuously differentiable with respect to the control variables. For example, in the case of a conditional (`IF`) statement which depends on the value of a model variable, the solution and its derivatives should agree for the critical value of the model variable. (Iterative processes within the NLM are also troublesome in this regard if the convergence criteria depend on the model variables. We comment on this in the next section.)

Let x_n be the model state at the end of stage n of the calculation, and let $\mathcal{J}$ be the objective function. The model state vector contains all the variables needed to continue the model calculation, including the evaluation of the objective function. Therefore when $\mathcal{J}$ is a sum which is calculated during the model run, as is usually the case, the partial sum of $\mathcal{J}$ at the end of stage n, which we will call J_n is included in the model state x_n. At the end of the calculation, i.e. for the final value of n, J_n is equal to $\mathcal{J}$. As we will see, the advantage of including J_n in the model state vector, is that the forcing of the adjoint solution due to data misfit and the proper choice of the initial conditions for the adjoint calculation arise naturally.

We also include in the model state all parameters which might be adjusted to minimize the objective function. Such parameters are usually quantities which do not vary during one model run, such as coefficients appearing in parametrizations of surface drag, cloud fraction and diffusion. We might even consider the size of the time step to be such a parameter. Thus our model state vector x_n is an augmented model state, containing not only z_n, the variables of the model, but also α_n, the

tunable parameters of the model, and J_n, the objective function, as calculated through the end of stage n of the calculation,

$$x_n = \begin{pmatrix} z_n \\ \alpha_n \\ J_n \end{pmatrix}. \tag{2}$$

Note that z_n and α_n are themselves column vectors. In this section and in section 3, all partitioned vectors and matrices will be partitioned in this way. Since the α_n are constant in the NLM and LTM we may drop the subscript n. However the variables in the adjoint model which correspond to α_n do depend on n. These variables are used to calculate the sensitivity of the objective function to the model parameters.

The object of the adjoint calculation we are about to describe is to determine $\nabla_{x_0}\mathcal{J}$, the gradient of $\mathcal{J}$ with respect to the initial augmented model state. Since x_0 contains z_0 and α, this gradient contains the gradient of $\mathcal{J}$ with respect to the model initial conditions and model parameters. Given $\nabla_{x_0}\mathcal{J}$, we can calculate the perturbation of the cost function, $\delta\mathcal{J}$, for any infinitesimal initial perturbation, δx_0, of the model state vector x_0, according to

$$\delta\mathcal{J} = \langle \nabla_{x_0}\mathcal{J} \, , \, \delta x_0 \rangle \, . \tag{3}$$

To derive $\nabla_{x_0}\mathcal{J}$, we begin by constructing $\delta\mathcal{J}$ for an arbitrary δx_0, and then manipulate our solution so that it is in the form of (3).

The linear tangent model (LTM) may be written

$$\delta x_n = A_n \delta x_{n-1}. \tag{4}$$

where A_n is the linearized model dynamics for stage n. We note that the equations for $\delta\alpha$ are trivial, since the parameter perturbations are (arbitrary) constants. To integrate the LTM from arbitrary initial conditions δx_0 we repeatedly apply (4), obtaining δx_N in terms of the perturbation of the initial model state,

$$\delta x_N = A_N A_{N-1} \cdots A_2 A_1 \delta x_0. \tag{5}$$

Now let N identify the last stage of the calculation, so that $\mathcal{J} = J_N$. Our partitioning of vectors and matrices makes $\delta\mathcal{J}$ the last element of δx_N. Thus we may isolate $\delta\mathcal{J}$ by constructing the dot

product of δx_N with the unit vector u,

$$\delta \mathcal{J} = \langle u \, , \, \delta x_N \rangle \, , \tag{6}$$

where

$$u = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} . \tag{7}$$

Substituting the value of δx_N from (5), and making repeated use of (1), we obtain

$$\delta \mathcal{J} = \left\langle A_1^* A_2^* \cdots A_{N-1}^* A_N^* u \, , \, \delta x_0 \right\rangle . \tag{8}$$

Since this relationship is true for any δx_0, the desired gradient is given by

$$\nabla_{x_0} \mathcal{J} = A_1^* A_2^* \cdots A_{N-1}^* A_N^* u. \tag{9}$$

Comparing the expression for the gradient (9) with Eq. (5) we see that calculating the gradient is entirely analogous to integrating the LTM, but in reverse order and beginning from initial conditions u. For this reason, it is convenient to define the adjoint model by

$$x_{n-1}^* = A_n^* x_n^*, \tag{10}$$

and

$$x_N^* = u. \tag{11}$$

Repeated substitution of (10) into (9) results in

$$\nabla_{x_0} \mathcal{J} = x_0^*. \tag{12}$$

In the same manner as before, x_0^* is partitioned into three parts denoted z_0^*, α_0^* and J_0^*, which are respectively the gradients of $\mathcal{J}$ with respect to z_0, α and J_0. (The last of these derivatives is identically one, as we will see in the discussion below.) In conclusion, we may calculate the desired gradient by integrating the adjoint model (10) for $n = N, N-1, \cdots, 2, 1$, beginning with initial conditions (11).

3. DISCUSSION

The above formalism, while simple, treats in a unified way the general adjoint calculation for discrete models. With this approach, the gradient with respect to parameters of the model, and the forcing due to the data misfit do not require any special treatment in the adjoint model. All we must do is augment the model to calculate the objective function and include perturbations of the tunable parameters in the LTM. Considering the LTM to be the multiplication by a series of matrices, the adjoint model is simply the (complex conjugate) transposed LTM matrices applied in reverse order.

As we mentioned earlier, the division of the calculation into stages may be arbitrarily fine or coarse. The model state vector x_n must contain everything needed to continue the calculation. At any particular stage, say stage k of the calculation, the adjoint variables x_k^*, whether they correspond to model variables, model parameters or the objective function, respectively denoted z_k^*, $\alpha_k^*$and J_k^* may be interpreted as the sensitivity of $\mathcal{J}$ with respect to the corresponding components of the NLM variables, x_k at the end of stage k. (This can be seen by repeating the derivation of (12) for a calculation beginning with stage $k+1$.)

We now consider some special cases of (4). For this purpose, we first divide the computations finely enough so that we may consider a stage *either to not increment* the objective function *or to only increment* the objective function. In the first case, for example at stage k of the model integration, the parameters and the objective function are unchanged and this stage of the NLM may be written as

$$z_k = F_k\left(z_{k-1}, \alpha\right). \tag{13}$$

In the LTM, this is represented by

$$A_k = \begin{pmatrix} \nabla_z F_k & \nabla_\alpha F_k & 0 \\ 0 & I & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{14}$$

Here we see that small changes in α influence the calculation of δz_k at each stage involving α, through the $\nabla_\alpha F_k$ term. These effects are carried along as δz_k evolves, and accumulate during the solution. In the second case, for example at stage m, only the objective function is changed and this stage of the NLM may be written as

$$J_m = J_{m-1} + H_m\left(z_{m-1}, \alpha\right). \tag{15}$$

In the LTM, this is represented by

$$A_m = \begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ \nabla_z H_m & \nabla_\alpha H_m & 1 \end{pmatrix}. \tag{16}$$

In either case, the transpose of the LTM (14, 16) shows that $J^*_{n-1} = J^*_n$. The initial conditions (7, 11) therefore imply that J^*_n is always one, consistent with the fact that J^*_n is the derivative of $\mathcal{J}$ with respect to its partial sum at stage n.

While the parameter perturbations, $\delta\alpha_n$, are unchanged by the LTM, the corresponding adjoint model variables, α^*_n, accumulate the contributions to $\partial\mathcal{J}/\partial\alpha$ from the stages of the calculation. Usually, the objective function does not depend explicitly on the model parameters α, i.e., $\nabla_\alpha H_m = 0$. For this case (16) shows that a stage incrementing $\mathcal{J}$ reduces to

$$z^*_{m-1} = z^*_m + \nabla_z H_m, \tag{17}$$

since $J^*_m = 1$. The term $\nabla_z H_m$ represents the forcing of the adjoint solution due to data misfit. Furthermore, when $\nabla_\alpha H_m = 0$, specializing (16) shows that α evolves according to

$$\alpha^*_{k-1} = \alpha^*_k + \nabla_\alpha F_k \, z^*_k. \tag{18}$$

4. PRACTICAL IMPLEMENTATION

In the methodology described here, there are two main steps in writing an adjoint model. A required preliminary is to augment the full nonlinear model (NLM) to calculate the objective function $\mathcal{J}$. Then the first step is to write the LTM and the second step is to transpose the LTM. The second step is usually done for small subdivisions or stages of the calculation. Once the adjoint of each stage is created, it is necessary to reverse the order of the transposed stages. Before continuing we note that other approaches may be taken, but the rules described here are particularly straightforward. Many of the procedures described here can be performed nearly automatically. Further, the approach described here avoids errors associated with overwriting an adjoint variable which should be incremented or vice versa. Such errors are the most common errors made in writing an adjoint model.

The dependence of the LTM on the solution of the NLM is formally through the A_n. Since the LTM is an infinitesimal perturbation about the NLM solution, all control structures - loops, if-then-else blocks, etc. - are inherited with no changes from the NLM. For this reason, a conditional involving a perturbation quantity is an error. In a practical implementation, matrix storage for the A_n must be avoided, since they are very sparse. Of course, the variables from each stage of the full model which are needed to evaluate the coefficients in the perturbation model must be saved or recreated as the perturbation model is evolved. In the adjoint model this becomes somewhat awkward since the adjoint model evolves backwards relative to the NLM. It is generally good practice to save the results of as many intermediate complex nonlinear calculations as possible. Simple nonlinear calculations may be repeated rather than stored and retrieved.

For each assignment statement in the NLM, there may be a corresponding statement in the LTM. If there are perturbations to any of the variables on the *rhs* of a NLM equation, then a perturbation form of the equation must be written. In this case a perturbation is defined corresponding to the variable on the *lhs* of the NLM. We note that in the case of the LTM, the initial conditions are arbitrary. We now briefly mention some situations one should be aware of in writing the LTM.

1. A procedure which is linear in terms of the model state vector may be used as originally written, simply by replacing the model state vector with the perturbation state vector.
2. When writing the perturbation equation of a complicated product, one can take advantage of the properties of logarithms. For example, a NLM assignment statement given by,

```
f = ((a+b)**2 * sqrt(c)) / (d*e)
```

becomes

```
Df = f*(2*(Da + Db)/(a+b) + Dc/(2*c) - Dd/d - De/e)
```

where the upper case D prefix denotes a perturbation variable. We note that the convention in the ECMWF Integrated Forecast System (IFS) is to add a suffix 5 to the full model state or trajectory variables. That is, the same variable name is used for the full model state variable in the NLM code as is used for the perturbation variable in the LTM and adjoint codes. But in the LTM and adjoint codes, the full variables, called trajectory variables in this context, are suffixed by a 5. This convention eliminates rewriting linear equations, but requires rewriting equations for nonlinear temporaries. Either approach, consistently used, is workable.

3. Use caution when code executed conditionally changes the value of a NLM variable which was used to evaluate the condition. For example,

```
a = min(a, amax)
```

should be considered equivalent to

```
if (a .ge. amax) a = amax
```

which may be written in the LTM as

```
if (a .ge. amax) Da = 0
```

Note the condition remains true after the change in the value of a, so that Da will be properly set to 0 if a is initially larger than amax. This would *not* be true if the conditional was written as (a .gt. amax).

4. In the case of a convergent iteration, it may not be necessary to repeat the control structure of the NLM in the LTM. This will be true when the process converges to machine accuracy; otherwise all the concerns one has about making sure the gradient one calculates, is the gradient of the actual finite process, come into play. In the former case, once a set of equations is iterated to machine accuracy convergence, the NLM variables may be taken to be known exactly. Then the corresponding perturbation equation can be easily solved since the perturbation form of the equations is linear in the perturbation quantities. (We do not have to worry if these equations are singular, since we already solved the nonlinear version.) As an example, we consider the calculation of z_0, the surface roughness length in a stable oceanic planetary boundary layer model we have studied (*Hoffman and Louis*, 1990),

$$\begin{aligned} z_0 &= v/y^2 \\ y &= \ln(z/z_0) \end{aligned} \tag{19}$$

where z is the constant elevation of the observations of the atmospheric variables and v depends on the wind speed and atmospheric stability. (In the unstable case v depends on z_0 as well.) These equations converge fairly quickly but we also accelerate the process by using Newton's

method after a few iterates. Perturbing these equations yields a system of two equations for δz_0 and δy, whose solution is

$$\begin{aligned} \delta z_0 &= \left(\delta v / y^2\right) / (1 - 2/y) \\ \delta y &= -\delta z_0 / z_0. \end{aligned} \tag{20}$$

One can therefore use (20) in the LTM instead of the iterative structure used to solve (19).

To create the adjoint model, it is useful to consider each assignment statement in the LTM to be a stage. Then converting the LTM into the adjoint model is divided into two steps, as mentioned before. In the first step, each single LTM assignment statement is converted into a series of simple adjoint assignment statements. In the second step, the order of execution of all the adjoint assignment statements is reversed.

To initialize the adjoint model, as per (11), all adjoint variables are set to zero, except that $J^* = 1$. In particular, local adjoint variables must be set to zero. Accomplishing this depends on the programming environment. In standard Fortran, one approach is to set to zero any actual adjoint variable or array at the start of the program unit in which they first become defined. Dummy arguments should not be initialized in this way. Variables and arrays appearing in COMMON and SAVE statements should be initialized before the adjoint model begins execution. (A precise technical definition of how entities become defined or undefined is given in Section 17.2 and 17.3 of ANSI X3.9-1978. Likewise see Section 15.9 for dummy arguments.)

In order to transpose a single LTM assignment statement and to properly initialize the adjoint variables, it suffices to consider two very simple cases. Since the LTM is linear in the perturbation variables, any LTM statement may be considered (and in fact may be easily rewritten) as a series of simple LTM statements, the first of the form

$$\delta r = a \delta r \tag{21}$$

and subsequent ones of the form

$$\delta r = \delta r + b \delta s. \tag{22}$$

These statements may be represented in the form of (4), in which the matrix A_n is equal to the identity matrix except for one diagonal element equal to a in the first case and for one off diagonal element equal to b in the second case. It is then easy to see that the adjoints of these two cases are

$$r^* = a r^* \tag{23}$$

and

$$s^* = s^* + br^* \tag{24}$$

Often $a = 0$, that is, r^* is reinitialized. *This case must not be overlooked.* These equations must be put in reverse order from that shown here. In other words, for each simple LTM assignment statement arising from the single original LTM assignment statement, there results a single simple adjoint statement and of these, the first, which redefines r^*, will be executed last, once all the adjoint statements have been put in reverse order.

One result of this approach is that there will sometimes be code which sets a variable equal to itself, or which sets a variable equal to zero when that variable is not referenced again, or in which several statements incrementing a variable might easily be combined. We prefer not to clean up such code, so that the derivation of the adjoint code is clear. An optimizing compiler will eliminate any inefficiency in such cases.

In order to reverse the order of the adjoint assignment statements, it is useful to have a properly indented code, in which every time the start (or end) of a control structure is encountered, the indentation level is incremented (or decremented, respectively). All control structures and assignment statements at the same indentation level within a single control structure (or procedure) must be reverse ordered. In general, all looping structures must be rewritten to execute in reverse order. In cases when the order of execution of the loop is immaterial, this reversal is not necessary. Recalculation of NLM variables must occur before they are used. Often, it is sufficient to float the NLM calculation to the top of the control structure in which they are found.

It should be remembered that it is not always necessary to develop an adjoint code. In some cases an operator is self-adjoint, or nearly self-adjoint. Transforms between different sets of orthogonal basis functions often fall into this category. (See for example, *Courtier and Talagrand* (1987).)

5. AN EXAMPLE

As an example we consider a very simple predictive model, namely the model of the growth of forecast error variance of *Dalcher and Kalnay* (1987). In this section, we will optimize the initial conditions of this model for an ensemble of 5 cases while simultaneously optimizing the 3 parameters of the model. For this demonstration we treat one term of the tendency equation semi-implicitly. Further, the data used in the objective function are not model variables, but are calculated from the model variables, through what may be termed a diagnostic relationship.

The Dalcher and Kalnay model is

$$\frac{dV}{dt} = (aV + S)(1 - V/V_\infty) \equiv \dot{V} \tag{25}$$

where V is the mean square forecast error, averaged over some region, and a, S and V_∞ are empirical parameters. Here a is the inherent growth rate, S is the error source and V_∞ is the asymptotic level of V. These parameters should be constant for a given data assimilation and forecast system, but depend on the variable in question, the region, the data sources, etc. A closed form solution of (25) is given by Dalcher and Kalnay. For our purposes we discretize the time domain, using a superscript m to denote time level m. During the initial forecast period, V grows rapidly. Accordingly, we treat the linear term aV semi-implicitly. That is, we replace aV^m by $a\left(\mu V^{m+1} + (1-\mu)V^m\right)$. We obtain

$$\left(V^{m+1} - V^m\right)/\Delta t = \dot{V}^m + a\mu\left(V^{m+1} - V^m\right), \tag{26}$$

or

$$V^{m+1} = V^m + G^{-1}\Delta t\,\dot{V}^m, \tag{27}$$

where

$$G = (1 - a\mu\Delta t). \tag{28}$$

We will choose $\Delta t = 0.1$ days and $\mu = 1/2$ in what follows.

For this exercise we simulated data every 0.5 days from 0.5 to 4.0 days, for 5 different initial conditions, V_i^0 , $i = 1, \cdots, 5$ and take as our observables the root mean square forecast error,

$$R_{ik} = \sqrt{V_i^{5k}} \tag{29}$$

for the 5 forecasts ($i = 1, \cdots, 5$) and the 8 observing times ($k = 1, \cdots, 8$). The factor of 5 in the superscript of V accounts for the fact that observations are made only every 5 time steps. The

Case	Objective function (m^2)	Scaled rms gradient	Number of function calls	Number of gradient calls
Initial estimate	1340	11.4	0	0
UMING	.2E-8	.1E-4	30	28
UMIDH	3.3E-8	.3E-4	13	47
UMCGG	1.0E-8	.2E-4	84	84

Table 1 Results of minimization experiments.

different initial conditions might correspond to different scenarios. For example, in a set of observing system experiments one might have forecast scores for several forecasts which differed only in the data sources used in the initial analyses.

In order to simulate the true data, denoted $\hat{R}_{ik}$ we use the values $a = 1$ day^{-1}, $S = 1000$ m^2/day, $V_\infty = 10000$ m^2 and $V_i^0 = 100i$ m^2, for $i = 1, \cdots, 5$. These data and parameter values, which we will refer to as the truth, are the right order of magnitude for a 500 mb geopotential height forecast. We will try to recover these parameter values in what follows by minimizing an objective function, starting from an incorrect initial estimate. For this initial estimate a, S, and V_∞ take on the true values given above, but $V_i^0 = 0$ m^2.

We now seek to minimize the objective function

$$\mathcal{J} = \sum_{i,k} \left(R_{ik} - \hat{R}_{ik} \right)^2 \tag{30}$$

with respect to the parameters a, S and V_∞, and initial conditions V_i^0. Note that R_{ik} depends on the parameters and initial conditions, but the $\hat{R}_{ik}$ are the constant true observations we calculated previously. The initial estimate used corresponds to a value of $\mathcal{J}$ of 1340 m^2 and an rms scaled gradient of 11.4 (see Table 1). By way of comparison, a value of 49.3 m^2 for $\mathcal{J}$ is calculated by comparing the analytic solution of (25) for the true values of the parameter and initial conditions, to the true observations obtained from the finite difference scheme (27). To accomplish the minimization we first write a computer program to evaluate $\mathcal{J}$. This is shown in Fig. 1. In writing this program we have made several choices which are significant for the form of the particular LTM and adjoint model we develop, but not for applying our approach in general. Note that `Ginv` is calculated every time `Model` is called, that the array `V` has values for all time levels m, that `V` contains data only for the ith forecast and that a temporary variable `Vdot` is defined.

```
      subroutine ObjFun (V0,a,S,Vinf,Rhat,J)
      parameter (Nm=40,Nk=8,Ni=5,Ns=Nm/Nk)
      real V0(Ni),V(0:Nm),Rhat(Ni,Nk)
      real J
      J=0
      do 100 i=1,Ni
         call Model (V0(i),a,S,Vinf,V)
         do 50 k=1,Nk
            J=J+(sqrt(V(Ns*k))-Rhat(i,k))**2
50       continue
100   continue
      return
      end

      subroutine Model (V0,a,S,Vinf,V)
      real mu
      parameter (Nm=40,mu=0.5,dt=0.1)
      real V(0:Nm)
      Ginv=1/(1-a*mu*dt)
      V(0)=V0
      do 200 m=0,Nm-1
         Vdot=(a*V(m)+S)*(1-V(m)/Vinf)
         V(m+1)=V(m)+Ginv*dt*Vdot
200   continue
      return
      end
```

Fig. 1 The NLM code for the calculation of the objective function described in the text.

The code for the LTM is shown in Fig. 2. This routine calculates $\delta\mathcal{J}$ corresponding to particular δa, δS, δV_∞ and δV_i^0. The transformation from Fig. 1 to Fig. 2 is for the most part mechanical, requiring one to write the first order variation of each equation, bearing in mind which quantities may be perturbed. Note that substantial amounts of code from the NLM must be repeated because we have not saved V_i^m, G^{-1} and $\dot{V}$. Clearly, deciding which variables to save and which to recalculate is an important concern in a large model. Passing from the LTM to the adjoint model (Fig. 3) is also fairly mechanical. Note that in Fig. 3 the prefix "D" denotes an adjoint variable, while it denotes an LTM variable in Fig. 2. In Fig. 3, note the initialization of the local adjoint variables DV, DGinv and DVdot. We note that the order of execution of the DO 50 loop is immaterial and one might easily augment this loop with a calculation of J, so as to calculate the objective function and its gradient in tandem.

Several minimization algorithms from the IMSL package were tested. IMSL routine UMIDH uses a

```
      subroutine lObjFun (V0,a,S,Vinf,Rhat,DV0,Da,DS,DVinf,DJ)
      parameter (Nm=40,Nk=8,Ni=5,Ns=Nm/Nk)
      real V0(Ni),V(0:Nm),Rhat(Ni,Nk)
      real DV0(Ni),DV(0:Nm),DJ
      DJ=0
      do 100 i=1,Ni
         call Model (V0(i),a,S,Vinf,V)
         call lModel (a,S,Vinf,V,DV0(i),Da,DS,DVinf,DV)
         do 50 k=1,Nk
            DJ=DJ+(sqrt(V(Ns*k))-Rhat(i,k))*DV(Ns*k)/sqrt(V(Ns*k))
50       continue
100   continue
      return
      end

      subroutine lModel (a,S,Vinf,V,DV0,Da,DS,DVinf,DV)
      real mu
      parameter (Nm=40,mu=0.5,dt=0.1)
      real V(0:Nm)
      real DV(0:Nm)
      Ginv=1/(1-a*mu*dt)
      DGinv=Da*mu*dt*Ginv**2
      DV(0)=DV0
      do 200 m=0,Nm-1
         Vdot=(a*V(m)+S)*(1-V(m)/Vinf)
         DVdot=(Da*V(m)+a*DV(m)+DS)*(1-V(m)/Vinf)-
     +         (a*V(m)+S)*(DV(m)/Vinf-V(m)*DVinf/Vinf**2)
         DV(m+1)=DV(m)+dt*(DGinv*Vdot+Ginv*DVdot)
200   continue
      return
      end
```

Fig. 2 The LTM corresponding to Fig. 1.

Newton method modified by a trust region restriction (*Gay*, 1983). IMSL routine UMING is quasi-Newton method which uses BFGS updates (*Dennis and More*, 1977). IMSL routine UMCGG is based on the conjugate gradient method described by *Powell* (1977).

All three algorithms were successful at finding the true solution. Their relative performance is summarized in Table 1, which shows results using 32 bit IEEE arithmetic. Of the three algorithms, UMCGG is the least sophisticated and requires the most function and gradient calls. UMIDH requires fewer function calls, but more gradient calls than UMING for this case. In all cases shown in Table 1, the control variable, cost function and its gradient were scaled before being passed to the minimization

```
      subroutine aObjFun (V0,a,S,Vinf,Rhat,DV0,Da,DS,DVinf,DJ)
      parameter (Nm=40,Nk=8,Ni=5,Ns=Nm/Nk)
      real V0(Ni),V(0:Nm),Rhat(Ni,Nk)
      real DV0(Ni),DV(0:Nm),DJ
      DJ=1
      do 25 m=0,Nm
         DV(m)=0
25    continue
      do 100 i=Ni,1,-1
         call Model (V0(i),a,S,Vinf,V)
         do 50 k=Nk,1,-1
            DV(Ns*k)=DV(Ns*k)+(sqrt(V(Ns*k))-Rhat(i,k))*DJ/
     +          sqrt(V(Ns*k))
            DJ=DJ
50       continue
         call aModel (a,S,Vinf,V,DV0(i),Da,DS,DVinf,DV)
100   continue
      return
      end

      subroutine aModel (a,S,Vinf,V,DV0,Da,DS,DVinf,DV)
      real mu
      parameter (Nm=40,mu=0.5,dt=0.1)
      real V(0:Nm)
      real DV(0:Nm)
      Ginv=1/(1-a*mu*dt)
      DGinv=0
      DVdot=0
      do 200 m=Nm-1,0,-1
         Vdot=(a*V(m)+S)*(1-V(m)/Vinf)
         DVdot=DVdot+dt*Ginv*DV(m+1)
         DGinv=DGinv+dt*DV(m+1)*Vdot
         DV(m)=DV(m)+DV(m+1)
         DV(m+1)=0
         DVinf=DVinf+(a*V(m)+S)*V(m)*DVdot/Vinf**2
         DS=DS+DVdot*(1-V(m)/Vinf)
         DV(m)=DV(m)+a*DVdot*(1-V(m)/Vinf)-(a*V(m)+S)*DVdot/Vinf
         Da=Da+DVdot*V(m)*(1-V(m)/Vinf)
         DVdot=0
200   continue
      DV0=DV0+DV(0)
      DV(0)=0
      Da=Da+DGinv*mu*dt*Ginv**2
      DGinv=0
      return
      end
```

Fig. 3 The adjoint corresponding to Fig. 1.

algorithm. The scales used were 0.1 day^{-1} for a, 100 m^2/day for S, 1000 m^2 for V_∞, 100 m^2 for V_i^0 and 100 m^2 for $\mathcal{J}$.

Without scaling the results for the three algorithms were very different. Without scaling, UMIDH was again successful at locating the true solution exactly, but required 58 function and 65 gradient evaluations. UMING did not converge at all. It appears to have taken a poor first step and was unable to recover. UMCGG stopped after using 28 function and gradient evaluations with an error condition, having reduced $\mathcal{J}$ to a value of 10 m^2. This is a reasonable fit to the data, but the parameter values found in this case are far from correct.

ACKNOWLEDGMENTS

The preparation of this report was supported in part by National Science Foundation grant ISI-8960592, Department of Energy grant DE-FG02-90ER61065 and JPL contract 957644, a subcontract to NASA contract NAS7–918. We thank our colleagues and reviewers who provided valuable comments on an earlier manuscript on this topic. In particular, suggestions of Jean-Noël Thépaut lead to considerable improvement of Section 4.

REFERENCES

[1] Philippe Courtier and Olivier Talagrand. Variational assimilation of meteorological observations with the adjoint vorticity equation. Part 2: Numerical results. *Quarterly Journal of the Royal Meteorological Society*, 113:1329–1368, 1987.

[2] A. Dalcher and E. Kalnay. Error growth and predictability in operational ECMWF forecasting. *Tellus*, 39A:474–491, 1987.

[3] J. E. Dennis and J. J. More. Quasi-Newton methods, motivation and theory. *SIAM Review*, 19:46–89, 1977.

[4] John C. Derber. Variational four-dimensional analysis using quasi-geostrophic constraints. *Monthly Weather Review*, 115:998–1008, 1987.

[5] John C. Derber. A variational continuous assimilation technique. *Monthly Weather Review*, 117:2437–2446, 1989.

[6] David M. Gay. Algorithm 611: Subroutine for unconstrained minimization using a model/trust-region approach. *ACM Transactions on Mathematical Software*, 9:503–524, 1983.

[7] M. C. G. Hall, D. G. Cacuci, and M. E. Schlesinger. Sensitivity analysis of a radiative-convective model by the adjoint method. *Journal of the Atmospheric Sciences*, 39:2038–2050, 1982.

[8] Ross N. Hoffman and Jean-François Louis. The influence of atmospheric stratification on scatterometer winds. *Journal of Geophysical Research*, 95(C6):9723–9730, 1990.

[9] F.-X. Le Dimet and O. Talagrand. Variational algorithms for analysis and assimilation of meteorological observations: Theoretical aspects. *Tellus*, 38A:97–110, 1986.

[10] F. X. LeDimet, I. M. Navon, and X. Zou. Incomplete observations and control of gravity waves in variational data assimilation. Part I: Theoretical aspects. Technical Report 91-91, Supercomputer Computations Research Institute, The Florida State University, Tallahassee, 1991.

[11] J. M. Lewis and J. C. Derber. The use of adjoint equations to solve a variational adjustment problem with advective constraints. *Tellus*, 37A:309–322, 1985.

[12] M. J. D. Powell. Restart procedures for the conjugate gradient method. *Math. Programming*, 12:241–254, 1977.

[13] F. Rabier and P. Courtier. Four-dimensional variational assimilation in the presence of baroclinic instability. Technical Memorandum 183, ECMWF, Reading, U.K., 1992.

[14] Olivier Talagrand and Philippe Courtier. Variational assimilation of meteorological observations with the adjoint vorticity equation. Part 1: Theory. *Quarterly Journal of the Royal Meteorological Society*, 113:1311–1328, 1987.

[15] W. C. Thacker. Fitting models to inadequate data by enforcing spatial and temporal smoothness. *Journal of Geophysical Research*, 93(C9):10655–10664, 1988.

[16] W. C. Thacker and R. B. Long. Fitting dynamics to data. *Journal of Geophysical Research*, 93(C2):1227–1240, 1988.

[17] J.-N. Thépaut and P. Courtier. Four-dimensional variational data assimilation using the adjoint of a multilevel primitive equation model. Technical Memorandum 178, ECMWF, Reading, U.K., February 1991. Submitted to QJRMS.

[18] X. Zou, I. M. Navon, and F. X. LeDimet. Incomplete observations and control of gravity waves in variational data assimilation. Part II: Applications and numerical results. Technical Report 91-92, Supercomputer Computations Research Institute, The Florida State University, Tallahassee, 1991.