MODELING, IDENTIFICATION AND CONTROL, 2005, VOL. 26, NO. 3, 121–134
doi:10.4173/mic.2005.3.1

Application of seeding and automatic differentiation in a large scale ocean circulation model

FRODE MARTINSEN† and DAG SLAGSTAD‡*

Keywords: *Jacobian, automatic differentiation, seeding*

Computation of the Jacobian in a 3-dimensional general ocean circulation model is considered in this paper. The Jacobian matrix considered in this paper is square, large and sparse. When a large and sparse Jacobian is being computed, proper seeding is essential to reduce computational times. This paper presents a manually designed seeding motivated by the Arakawa-C staggered grid, and gives results for the manually designed seeding as compated to identity seeding and optimal seeding. Finite differences is computed for reference.

1. Introduction

Computation of the full square and sparse Jacobian in a 3-dimensional general ocean circulation model is considered in this paper. When a large and sparse Jacobian is being computed, proper seeding is essential to reduce computational times. In addition, rounding errors should be kept at a minimum. Today the most widely used tool for computing Jacobians is automatic differentiation (AD). AD generates analytic derivates of functions provided as inputs in form of a source code, in e.g. C, `Fortran` or `Matlab`.

The AD tool parses the source code, and uses either source transformation or operator overloading to generate new source code for the desired gradient Griewank (2000). In either case, the mathematical founding is through usage of the chain rule. Consider the function $f(u, v) = u \cdot v$. By using the rules of differentiation we have $f'(u, v) = u' \cdot v + u \cdot v'$. To generate a source code for this function, the AD tool needs a list of the independent variables (u and v in this case). Then the AD tool can parse the source code looking for names contained in the list of independent variables. Whenever such a name is located, the rules of differentiation is applied to build $f'(u, v)$. Note that $f'(u, v)$ does not contain rounding errors if f is sufficiently smooth. If f is only continuous, special rules may be applied to trap non-smooth points.

If $f(u, v)$ appears at other places in the source code in building say $g(f)$, then the chain rule is used. This will generate a 'trace' of the independent variables though the code to the final dependent variables. This trace is maintained in a graph. The branches of the graph may be independent in computing the derivate of the dependent variables with respect to different independent variables. Therefore, the derivative with respect to a set of independent variables may be computed simultaneously. Finding such sets of independent branches is termed seeding. Finding the optimal seeding is NP-hard.

If the Jacobian matrix has low density, seeding is likely to give substantial reductions in computational times. This is often the case with functions specified as partial

*Corresponding author. Tel: +47-73592407. Fax: +47-73595660. E-mail: dag.slagstad@sintef.no
†Department of Engineering Cybernetics, NTNU, 7491 Trondheim, Norway.
‡SINTEF Fisheries and Aquaculture, SINTEF, Trondheim, Norway.

differential equations (considered in this paper) and in dynamical systems with instantiation over a time horizon as in nonlinear model predictive control (NMPC) and receding horizon estimation.

The following model is considered: The state variables are horizontal velocities, U, V, elevation, E, temperature, T, and salinity, S. In the following we define the notation for a scalar ψ and a vector $\phi = (U, V, W)$, where W is the vertical velocity:

$$\nabla\phi = \frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} + \frac{\partial W}{\partial z}$$

$$(\phi \cdot \nabla)\psi = U\frac{\partial \psi}{\partial x} + V\frac{\partial \psi}{\partial y} + W\frac{\partial \psi}{\partial z}$$

$$\Delta\psi_\theta = \frac{\partial}{\partial z}\left(A_z\frac{\partial \psi}{\partial z}\right) + A_\theta\left(\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2}\right) \tag{1}$$

Define the operator

$$FD_\theta\psi = (\phi \cdot \nabla)\psi - \Delta\psi_\theta \tag{2}$$

where A_z and A_θ are diffusivities. The subscript θ distinguishes between the horizontal diffusivity for the velocities (A_H) and properties temperature and salinity (A_Q). Using the notation (1), and assuming hydrostatic pressure, the state equations are given as:

Mass balance (continuity)

$$\nabla\phi = 0 \tag{3}$$

Impulse balance

$$\frac{\partial U}{\partial t} + FD_H U = fV - \frac{1}{\rho}\cdot\frac{\partial p}{\partial x} + b_U \tag{4}$$

$$\frac{\partial V}{\partial t} + FD_H V = -fU - \frac{1}{\rho}\cdot\frac{\partial p}{\partial y} + b_V \tag{5}$$

Temperature and salinity

$$\frac{\partial T}{\partial t} + FD_Q T = 0 \tag{6}$$

$$\frac{\partial S}{\partial t} + FD_Q S = 0 \tag{7}$$

f is the Coriolis term, ρ is density and b_U, b_V are friction terms, Note that the vertical velocity W is given algebraicly from (3), and that the elevation E is introduced as a state:

$$E = \int W_1 dt$$

Here, W_1 is the vertical velocity in the upper level. The hydrostatic assumption then gives the pressure p as:

$$p(z) = \int_z^E \rho g dz + P_a$$

Here, g is gravity and P_a is atmospheric pressure. The present model constitutes the hydrodynamic part of the biogeochemical model SINMOD, with details given in Slagstad & McClimans (2005). The SINMOD model is implemented with the Arakawa-C

staggered grid, z-levels, mode-splitting, FRS (Flow Relaxation Scheme) at the open boundaries and a super-bee limiter for the advection scheme.

In the present study, mode-splitting and FRS relaxation was excluded. Mode-splitting was excluded by setting the outer time step equal to the internal time step, and then only doing one iteration in the internal loop. This will maintain numerical stability and exposes the sparsity structure of the Jacobian. E.g. the Jacobian is defined at a time i, and including mode-splitting will in fact cause computation of a (dense) sensitivity matrix over the outer time step.

FRS relaxation is excluded due to the small model domain, see Section 2. E.g. including FRS relaxation on this domain, will dominate the elements of the Jacobian on nearly all grid points (since the FRS zone covers nearly all of the domain). This leads to a dense Jacobian on this small domain. FRS relaxation can be included on larger domains if identity seeding is used for the grid points in the FRS zone. In the case of identity seeding, nothing need to be done. In the case of manual seeding one needs to manually revert to identity seeding in the FRS zone, whereas optimal seeding reverts to identity seeding in the FRS zone automatically. Hence, the FRS relaxation pose no difficulties for proper seeding.

The Jacobian of model (3)–(7) is computed using automatic differentiation Griewank (2000) in this paper. This Jacobian matrix is square and sparse, and is needed in gradient based methods like nonlinear least squares (NLS) and implicit simulation schemes. Since the Jacobian is square, the forward mode of automatic differentiation (AD) is preferred.

The AD tool ADIFOR Bischof *et al.* (1996) has been applied in this paper. ADIFOR (v.2.0) is a forward mode, source transformation tool for F77 programs. ADIFOR has been applied to 3 dimensional models previously; Park *et al.* (1996) discusses the use of ADIFOR in a 3-D atmosphere model with Arakawa-C grid and Hovland *et al.* (1998) deals with a Navier-Stokes system. These references give no detail on the use of ADIFOR.

A discussion of the forward and reverse modes of automatic differentiation in the context of oceanographic models has been published by Giering & Kaminski (1998), Giering (2000), Elizondo *et al.* (2002). The reverse mode of AD is utilized in the 4DVAR and 3DVAR methods for sensitivity studies and data assimilation. These methods utilize the adjoint method Errico (1997), Navon (1997) from optimal control theory. In the adjoint method, assuming that there are few measurements, the resulting Jacobian will be non-square and dense (depending on the measurements). There are a number of applications with the adjoint method, see Dickey (2003) and the references therein. Robinson *et al.* (1998) gives an overview of data assimilation methods in oceanography. Non-smoothness and numerical issues are addressed by Thuburn & Haine (2001), Homescu & Navon (2003).

For large model domains, the order of the present Jacobian will be large and exploiting sparsity is essential. Sparsity can be exploited to reduce computational demands Curtis *et al.* (1974), Coleman & Moré (1983) and storage demands. Reducing the computational demands requires design of a seeding matrix that allows linearly independent columns to be computed simultaneously.

This paper presents a manually designed improved seeding which is motivated by the Arakawa-C staggered grid, and gives results for the improved seeding as compared to identity seeding and optimal seeding. Optimal seeding is provided by the DSM algorithm Coleman *et al.* (1984). Note that seeding issues applies to finite difference approximations of the Jacobian as well. Seeding for finite differences and reducing storage demands are not considered in this paper.

To the best of our knowledge, seeding has not been discussed in relation with oceanographic models previously. The reason for this is that the studies using automatic differentiation with oceanographic models mainly focus on sensitivity studies with few measurements. For such cases the reverse mode of automatic differentiation (giving a dense but relatively small sensitivity matrix) is the best choice. The usage of seeding is relevant in cases where measurements are distributed, e.g. elevation measurements from satelite data. Seeding is also relevant in cases where a general optimization algorithm is to be applied, e.g. in control applications. The form of the Jacobian in such a setting is discussed in Martinsen *et al.* (2004). Data assimilation for oceanographic models has been discussed and implemented in the Ensemble Kalman filter (EnKF) by Evensen, see e.g. Evensen (1994). The EnKF also depends on few measurements being used to allow effective computation of the filter update.

The paper is organized as follows. In section 2, the model implementation and its influence on the sparsity structure of the Jacobian is discussed. In section 3, the concept of seeding is introduced with examples of how this can be implemented for variables in 2 and 3 dimensions. Computational results are given in section 4, and conclusions and directions for future research are given in section 5.

2. Model implementation and seeding

In this section we describe the numerical model used to investigate the performance of automatic differentiation.

The model domain is covered by a 20×15 grid with resolution 1.5 km. There are 5 grid points in the vertical direction with level thickness [10, 10, 10, 35, 35] meters giving a total depth of 100 m. The depth matrix used in the model present setup is shown in Figure 1. This small domain is chosen because it allows storage and computation of the dense Jacobian on a personal computer. The dense Jacobian is included for reference only. Automatic differentiation and seeding on a sparse Jacobian is not limited by practical model domains. We also note that if the Jacobian is to be used with the extended Kalman filter, storage of the dense co-variance matrix does limit application on realistic model domains. The seeding described in this paper is extendible to larger domains.

After temporal discretization the model (3)–(7), can be written $X_{i,k+1} = f_i(X_{i,k})$ where k is the time index and X_i is a vector with the 5 states. With this notation, the following Jacobian is considered in this note:

$$J_k = \begin{bmatrix} \frac{\partial U}{\partial U}\Big|_k & \frac{\partial U}{\partial V}\Big|_k & \frac{\partial U}{\partial E}\Big|_k & \frac{\partial U}{\partial S}\Big|_k & \frac{\partial U}{\partial T}\Big|_k \\ \frac{\partial V}{\partial U}\Big|_k & \frac{\partial V}{\partial V}\Big|_k & \frac{\partial V}{\partial E}\Big|_k & \frac{\partial V}{\partial S}\Big|_k & \frac{\partial V}{\partial T}\Big|_k \\ \frac{\partial E}{\partial U}\Big|_k & \frac{\partial E}{\partial V}\Big|_k & \frac{\partial E}{\partial E}\Big|_k & \frac{\partial E}{\partial S}\Big|_k & \frac{\partial E}{\partial T}\Big|_k \\ \frac{\partial S}{\partial U}\Big|_k & \frac{\partial S}{\partial V}\Big|_k & \frac{\partial S}{\partial E}\Big|_k & \frac{\partial S}{\partial S}\Big|_k & \frac{\partial S}{\partial T}\Big|_k \\ \frac{\partial T}{\partial U}\Big|_k & \frac{\partial T}{\partial V}\Big|_k & \frac{\partial T}{\partial E}\Big|_k & \frac{\partial T}{\partial S}\Big|_k & \frac{\partial T}{\partial T}\Big|_k \end{bmatrix} \tag{8}$$

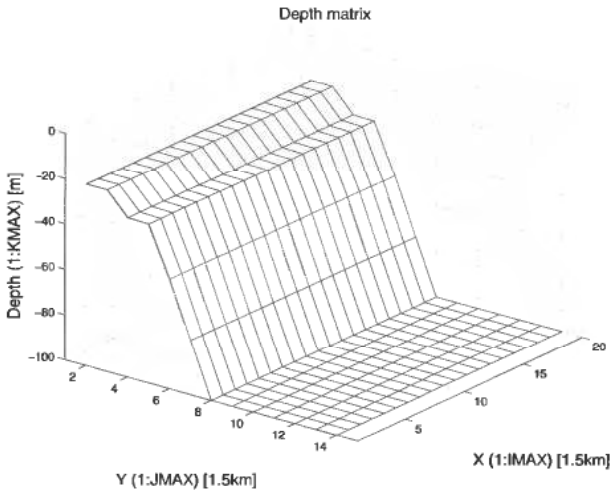

Figure 1. *Depth matrix.* The figure shows a mesh plot of the depth matrix used. The depth matrix has a shelf parallel to the x-axis. There is dry land along the two borders parallel to the x-axis. There are 20 grid points along the x-axis and 15 grid points along the y-axis.

Table 1. Variable dimensions

Variable	Dimensions
U	$\texttt{IMAX} + 1 \times \texttt{JMAX} \times \texttt{KMAX} = 21 \times 15 \times 5$
V	$\texttt{IMAX} \times \texttt{JMAX} + 1 \times \texttt{KMAX} = 20 \times 16 \times 5$
E	$\texttt{IMAX} + 2 \times \texttt{JMAX} + 2 = 22 \times 17$
S	$\texttt{IMAX} + 2 \times \texttt{JMAX} + 2 \times \texttt{KMAX} = 22 \times 17 \times 5$
T	$\texttt{IMAX} + 2 \times \texttt{JMAX} + 2 \times \texttt{KMAX} = 22 \times 17 \times 5$

The Jacobian (8) is evaluated at the initial state $(U, V, E) = (0, 0, 0)$ and with a sloping plane in S and T to avoid internal cancellation. The chosen initial state gives smooth gradients, except from evaluation of square roots at zero (which does not cause non-smoothness since the values are known to be non-negative). Since the sparsity pattern is given by the graph (an internal part of the automatic differentiation tool) of the numerical model, it is not affected by the choice of initial point. No wind stress, boundary flows or other forcing is present. FRS relaxation and mode-splitting is excluded, see discussion in Section 1.

The dimensions of the variables are listed in Table 1. In Table 1, the numerical values reflect the model domain shown in Figure 1. This gives the total Jacobian size of 7289×7289. This amounts to about 425Mbyte when stored as a dense matrix (in double real). The storage for the Jacobian as a sparse matrix will be discussed later.

2.1. *Spatial finite difference grid*

It is of interest to understand the spatial grid in detail, as it is related to the diagonal bands occurring in the Jacobian. The model uses the Arakawa-C grid. See Figure 2 for a detail.

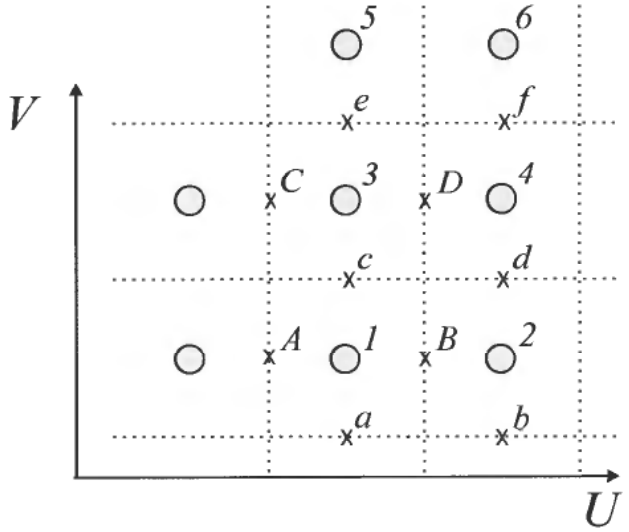

Figure 2. *Arakawa-C grid.* The figure shows a detail of some grid points in the horizontal plane. The state variable U is defined on points marked with capital letters, while V is defined on points marked with lower case letters. E, S, T are defined on the gray circles marked with numbers.

In the SINMOD implementation, the computational finite difference grid is defined by:

$$U_{ij}^{k+1} = U_{ij}^{k} + \Delta t f \bar{V}_{ij}^{k} + b_U \tag{9}$$

$$V_{ij}^{k+1} = V_{ij}^{k} - \Delta t f \bar{U}_{ij}^{k+1} + b_V \tag{10}$$

Here, k is the time index, i and j are the spatial index in the x and y direction respectively. Δt is the temporal discretization interval. Note that the U's are computed prior to the V's. In particular $\bar{U}$ and $\bar{V}$ are defined by:

$$\bar{U}_{ij} = \tfrac{1}{4}(U_{i,j} + U_{i+1,j} + U_{i,j-1} + U_{i+1,j-1}) \tag{11}$$

$$\bar{V}_{ij} = \tfrac{1}{4}(V_{i,j} + V_{i,j+1} + V_{i-1,j} + V_{i-1,j+1}) \tag{12}$$

Referring to Figure 2 and equation (9), a perturbation of U_D^k will only influence U_D^{k+1} (at the same point). However, due to equations and (10) and (11), the four points V_c^{k+1}, V_d^{k+1}, V_e^{k+1} and V_f^{k+1} will all be influenced. The six elevations (numbered 1–6) will all be influenced (since they all include cells that experience a change in either U or V).

If V_c^k is perturbed, due to equation (12) all of U_a^{k+1}, U_b^{k+1}, U_c^{k+1} and U_d^{k+1} will be influenced. Then, nine elements in V are influenced (all points including at least one of U_a^{k+1}, U_b^{k+1}, U_c^{k+1}, U_d^{k+1} in equation (11)). Similar explanations can be carried through for E, S, T as well.

An incidence plot[1] of the resulting Jacobian is shown in Figure 3. The density of the matrix is

$$d = 100\% \frac{nnz}{n \times m} = 100\% \frac{537969}{7289^2} \approx 1.0\%$$

[1]An incidence plot displays a dot at every non-zero location of the matrix.

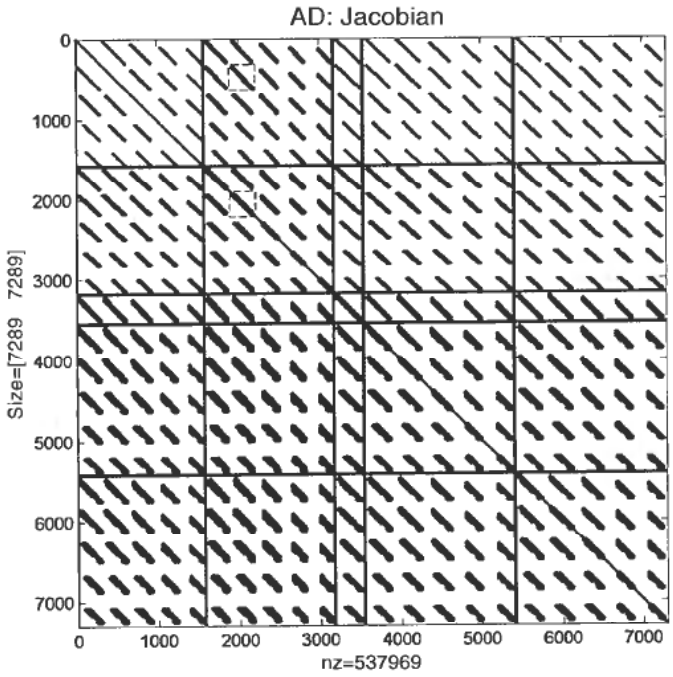

Figure 3. *Incidence plot of Jacobian.* The black lines separates the Jacobian into 25 blocks that corresponds to the blocks given in equation (8). Note the non-zeros reported in the figure. The dashed rectangles mark areas blown up in Figure 4.

where *nnz* is the number of non-zeros in the Jacobian, and n, m is the number of columns and rows in the Jacobian. A density of 1% implies that using linear algebra for sparse matrices will be efficient for this application. This sparse matrix requires about 6.5MByte storage in double real in co-ordinate sparse format e.g. about 1.5% of storage needed for the dense matrix.

Consider the upper left super-block $\partial U/\partial U$ from (8) in Figure 3. The 5 vertical levels in the model setup appears as 5 diagonal bands corresponding to the velocity component $U(x, y)$ along the x-axis for each vertical level k. The leftmost column in this super-block shows influence of $U(x, y)$ for $k = 1$ on the levels $1 \ldots 5$. The second column in this super-block shows influence of $U(x, y)$ for $k = 2$ on the levels $1 \ldots 5$. Likewise for columns 3, 4 and 5. The shelf shown in Figure 1 is causing the shortened diagonal bands at lower levels.

The width of the diagonals are not fully resolved in Figure 3, but blow-ups of the two marked areas inside super-block (1, 2), $\partial U/\partial V$, and (2, 2), $\partial V/\partial V$, are shown in Figure 4. The detailed blow-up shown in the right column in Figure 4 confirms the above discussion in relation to Figure 2 and equation (9)–(12) on the Arakawa-C grid. The upper pane shows the effect of a perturbation in V on U (second vertical level). Observe that a perturbation of one $V^k_{i,j}$ alters 4 $U^{k+1}_{\cdot,\cdot}$ elements at the same vertical level (likewise at all levels due to the hydrostatic assumption, not shown in blow up). The two bands show U elements at two neighboring j-levels.

Similarly, the lower right pane shows that a perturbation of one $V^k_{i,j}$ alters 9 $V^{k+1}_{\cdot,\cdot}$ elements at the same vertical level (likewise at all levels due to the hydrostatic assumption, not shown in blow up). The three bands show V elements at three neighboring j-levels.

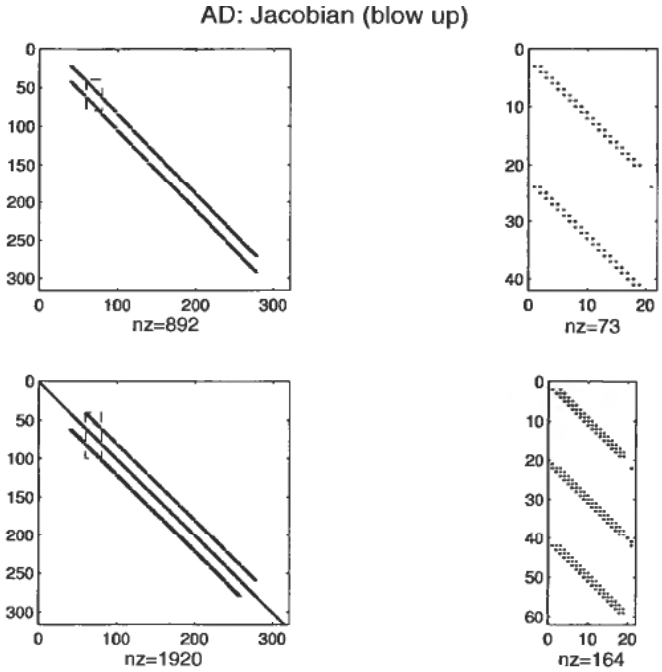

Figure 4. *Incidence plot of Jacobian (blow up).* The figure shows (left column) a blow up of the marked (dashed rectangles) from the incidence plot of the Jacobian in Figure 3. The marked areas in the left column are blown up again in the right column.

3. Automatic differentiation

Automatic differentiation (AD) is a technique for producing derivatives of a function automatically. The model must exist as source code in program that is input to the AD tool. An overview of AD tools is available at `www.autodiff.org`. The produced derivatives does not have round-off errors like finite differences do. That is AD produces analytic derivatives, in the sense that they are accurate to any order as long as the underlying function is sufficiently smooth. The underlying principle is the chain rule for differentiating expressions. See Griewank (2000) for further details, and Nocedal & Wright (1999) for an introduction. In the present paper, the AD tool ADIFOR Bischof *et al.* (1996) has been applied.

3.1. *Background*

The AD tool ADIFOR is a source transformation tool. That is, new source code is generated that must be compiled and linked. A call to this new code will produce a directional derivative along a user provided direction G_X. The notation G_X for the gradient of the variable X follows the ADIFOR users manual. Hence, ADIFOR produces the result:

$$G\_Y = J \cdot G\_X$$

Here J is the Jacobian. If $G\_X = I$, we have $G\_Y = J$ (the Jacobian). The matrix G_X is a termed a seed matrix. A general seed matrix is denoted G_S. E.g. if the variable X is to be differentiated, ADIFOR produces the derivative variable G_X with one added

dimension as compared to X. For instance, if $f=(f_1,f_2,f_3)'$ and $X=(x_1,x_2,x_3)'$, the Jacobian is:

$$J=\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix}$$

It is easily comprehended that differentiation gives an extra dimension in the Jacobian. Note that ADIFOR also adds one dimension to G_X to allow for a flexible interface. That is, instead of making repeated calls with the vector $X=e_i$, one can make one call with $X=I$ in order to evaluate the Jacobian.

The ADIFOR generated code executes faster if the seed matrix G_S has low dimension. This allows faster computation of the Jacobian if it has many linearly independent columns. Consider the example:

$$J_1=\begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ 0 & 0 & a_{33} \end{bmatrix}$$

Where column 3 is linearly independent from the first two. Instead of setting $G_{S_1}=I$, one can set

$$G_{S_1}=\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

to speed up calculation of J_1. In detail,

$$\begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ 0 & 0 & a_{33} \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ 0 & a_{33} \end{bmatrix} \tag{13}$$

which gives J_1 after unpacking. If, on the other hand, the seeding

$$G_{S_1}=\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

is applied, the resulting directional derivative is no longer the Jacobian. A Jacobian on the form given by equation (13) is called a *compressed* Jacobian. Before conventional linear algebra packages can be applied, this matrix must be unpacked into an uncompressed form. This unpacking adds overhead to the application. The computational results in the following shows that this overhead may be substantial. Formalism on seeding can be found in e.g. Hossain & Steihaug (2002).

3.2. *Dense seeding in 2 and 3 dimensions*

The main user problem is setting up a correct seeding matrix G_S for a distributed system. In the SINMOD model, the independent variables have dimensions 2 and 3 as shown in Table 1. The ADIFOR user manual Bischof *et al.* (1998) only shows how to deal with variables in one dimension. The following sections extends this to 2 and 3 dimensional variables.

As an example, consider E which has dimension IMAX + 2 × JMAX + 2. ADIFOR will produce a variable G_E of dimension G_P × IMAX + 2 × JMAX + 2, see Figure 5. Here G_P is a user selected parameter that must be larger than or equal to the largest variable dimension of all variables.

To understand seeding in the 3-dimensional matrix G_E it is beneficial to consider what happens when the matrix is unpacked into 2 dimensions. Then the sub-level matrices along the j axis are stacked below each other to produce a matrix of dimension (G_P•(JMAX + 2)) × IMAX + 2. The identity seeding for this system must then be initialized for each j-level independently. That is, the ADIFOR generated code must be called JMAX + 2 times with seeding running through the j-levels. The following code illustrates this:

```
IF  (VNUM  .EQ.  1)  THEN
        DO  I=0, IMAX+1
                G_E (I, I, JNUM) =1.0
        END  DO
 ELSEIF  (VNUM  .EQ.  2)  THEN
        DO  I=1, IMAX+1
                G_U(I, I, JNUM, KNUM) =1.0
        END  DO
END  IF
```

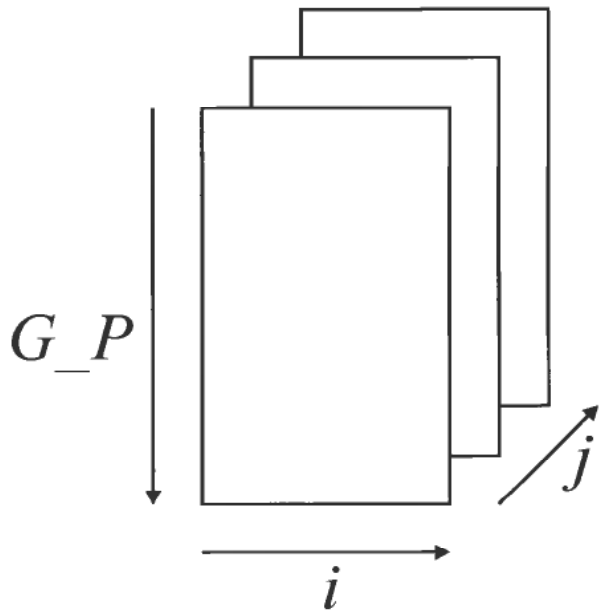

Figure 5. *The seed matrix G_E.* The figure shows the principal layout of the seed matrix of 3 dimensions. Note that G_P in ADIFOR is a user selected parameter that must be large enough to allow for storing all components of the derivative. According to Table 1, the minimum value of G_P is JMAX + 2 = 22 (for the elevation E). The indexes i, j addresses all elements up to (IMAX + 2, JMAX + 2).

The above code is snipped from the identity seeding implemented in the actual FORTRAN program. The user supplies the three parameters VNUM, JNUM and KNUM. The parameter VNUM controls which of the variables *G_E* and *G_U* is to be seeded. The parameter JNUM controls which *j*-level to be seeded, while KNUM controls which *k*-level is to be seeded (KNUM only applies for *G_U*, since *G_E* is of dimension 2). The user must keep track of (e.g. save) the intermediate results produced by the ADIFOR generated code. The optimal seeding discussed later, needs the sparsity pattern as input. The sparsity pattern is easily generated using identity seeding.

3.3. *Improved dense seeding*

The seeding described in section 3.2 above is termed the identity seeding for SINMOD. Now consider Figure 1 and the 2-dimensional variable *E*. When identity seeding is used, all points along the *x*-axis are perturbed for a given *j*-level. Looping through the JMAX + 2 *j*-levels, the full Jacobian is built.

However, Figure 2 and the discussion in section 2.1 shows that variables along every 3rd *j*-level are independent. Hence, a better seeding can be constructed by replacing the identity seeding with the following code:

```
DO  J = 1, JNUML
        JJ = JNUM(J)
        DO  I = 1, IMAX
                G_E(I, I, JJ) = 1.0
        END  DO
END  DO
```

Here JNUM = {1, 4, 7, 10, 13} and JNUML = 5 if JMAX = 15. (Separate calls need to be made for *j*-index 0 and 16.) The second call to the ADIFOR generated code would address *j*-levels = {2, 5, 8, 11, 14}, and the third addresses {3, 6, 9, 12, 15}. This will reduce the loops needed for computing the Jacobian with respect to *E* from 15 to 3 iterations. (Plus one extra for 0 and 16).

The vertical layers applicable to *U*, *V*, *S*, *T* may not be addressed in this manner since all layers in each coordinate (*i*, *j*) are linearly dependent.

3.4. *Sparse seeding in 2 and 3 dimensions*

The sparse seeding for *G_E* is shown below.

```
DO  J = 1, JNUML
        JJ = JNUM(J)
        DO I = 0, IMAX + 1
                CALL SSPSD( G_E(I, JJ), I+1, 1.0, 1)
        END DO
END DO
```

Here, JNUM and JNUML is as given in section 3.3. Identity seeding is achieved by setting JNUM = {*i*} and JNUML = 1, and then calling the ADIFOR generated code with *i* = {0, ... , JMAX + 1}. E.g. a total of JMAX + 2 calls to the code is made.

Improved seeding is achieved by setting JNUM = {1, 4, 7, 10, 13, ...} and JNUML = length(JNUM). Here, only 3 calls to the ADIFOR generated code are needed. This is independent of JMAX. (One extra call for JNUM = {0, JMAX + 1} is needed.)

Sparse unpacking of the internal ADIFOR format into a 2-dimensional compressed Jacobian can be done as shown below for *G_U*.

```
DO  I=1, IMAX+1
    DO  J=1, JMAX
        DO  K=1, KMAX
            CALL  SSPXSQ(i_data, v_dataR, inlen, g_u(I, J, K),
&             outlen, info)
            IF  (outlen .GT. 0)  THEN
              NZU1-ZU1+outlen
              DO  P=1, outlen
                  g_u_ivec(NZU1-outlen+P)=I+  (K-1)*
&                     (JMAX+0)*(IMAX+1)+(J-1)*(IMAX+1)
                  g_u_jvec(NZU1-outlen+P)=i_data(P)
                  g_u_valR(NZU1-outlen+P)=v_dataR(P)
              END  DO
            END  IF
        END  DO
    END  DO
END  DO
```

This gives a coordinate representation of a sparse matrix. Observe that explicit knowledge of the sparsity pattern is not utilized since all *I, J, K* are looped through. Hence, there is a potential for making numerous calls to SSPXSQ that will return outlen = 0. Refer to the ADIFOR manual Bischof *et al.* (1998) for explanation of the SSPSD and SSPXSQ calls.

3.5. *Optimal sparse seeding*

A better seeding for the vertical layers can be addressed through application of graph coloring algorithms Curtis *et al.* (1974), Coleman *et al.* (1984), Hossain & Steihaug (2002). The software DSM Coleman *et al.* (1984) was downloaded from www.netlib.org, and used the sparsity pattern generated by identity seeding as input. Since DSM only accept 2-dimensional sparsity patterns as input, the sparsity pattern was unpacked into 2 dimensions for this purpose. The resulting seeding from DSM, was then translated into 3 dimensions. Since the number of groups returned by DSM equals the minimum number by a result in Curtis *et al.* (1974), optimal seeding is achieved. Note that DSM does not give optimal seeding in general Hossain & Steihaug (2002).

4. Results

The computational times for the sparse AD schemes from section 3 and a simple finite difference scheme are given in Table 2. The CPU times are measured with the Matlab cputime command, and include the uncompressing time.

Matlab accessed SINMOD and the ADIFOR code through a gateway. Matlab gathered the results from ADIFOR into preallocated matrices, and visualized the resulting matrix. Gathering the results in this context includes uncompressing into 2 dimensions the matrices returned from the ADIFOR generated code. Using Matlab for this is time consuming, and future uncompression should be done entirely in FORTRAN.

Table 2. Computational times

Method	Seeding	NGRP	CPU time		
			ADIFOR	uncompress	Total
fd	—	—	22 min	—	22 min
sparse ad	identity	7289	86 sec	11 sec	97 sec
sparse ad	improved	2230	29 sec	35 sec	64 sec
sparse ad	optimal	247	5 sec	—	—

The reported percentage of uncompressing is considerably higher for the improved seeding. Note that this fraction does not include the FORTRAN code pieces shown in section 3.5. The reported percentage of CPU time for uncompression was spent building Matlab sparse matrices (repeated calls to the sparse subroutine) and stacking these into preallocated sparse matrices.

As noted in section 5, the improved seeding makes 3 calls to the ADIFOR generated code independent of the size of JMAX. For a system setup on a larger domain, the difference between identity and improved seeding is expected to increase. This is under the assumption that the uncompression time does not increase even more.

The number of groups for the improved seeding is about a third of identity seeding (7289 groups). This is reasonable, since on average every third j-index is computed simultaneously. Still, this is almost 10 times higher than the optimal number groups. The only benefit of the improved manual seeding is that it can be made generic, whereas the optimal seeding need to be generated for each model setup. Note that it is not necessary to rerun ADIFOR when the model domain or other setup parameters are changed. Changes in the model itself will require a new run of ADIFOR.

Observe that the fd (finite differences) shows the CPU time for constructing the dense Jacobian by perturbing each column of the Jacobian. That is, a total of $n = 7289$ calls to the ADIFOR code is made. The reported 22 min includes the time to store each column in a preallocated matrix. The computer did some paging due to the large storage demand (425MByte). The computations were implemented in Matlab with some routines available as mex/dll-files on a Dell Latitude C800/Pentium III/1GHz/512Mb RAM running Windows 2000.

The finite difference computational and storage demands can be improved by application of e.g. the FDSJ routine accompanying the DSM routine Coleman *et al.* (1984). Hence, the reported result for finite differences results from a naive implementation and should not be interpreted as a limitation of the finite difference approach. They are included to indicate that a careful implementation is essential for performance of both automatic differentiation and finite differences.

5. Conclusions

Seeding of 2 and 3 dimensional matrices has been presented. The scheme is trivially extendible to n dimensions. An improved seeding motivated by the finite difference grid was suggested. Computational results show some promise for these seeding approaches, still application of graph coloring methods yielded superior results.

A future work task is to reduce uncompression times, by writing this in FORTRAN. The resulting Jacobian can be used directly to discuss sensitivities of the model, or in gradient based estimation algorithms.

Acknowledgment

This paper was financially supported by the Research Council of Norway (project no. 128726/420 — MODTEQ).

References

Bischof, C., Carle, A., Kadhemi, P. & Mauer, A. (1996). ADIFOR2.0: Automatic differentiation of Fortran 77 programs, *IEEE Computational Science and Engineering* 3(3), pp. 18–32.

Bischof, C., Carle, A., Hovland, P., Khademi, P. & Mauer, A. (1998). ADIFOR 2.0 Users' guide, Tech. rep., Argonne National Laboratory, Argonne, IL., aNL/MCS-TM-192.

Coleman, T. F. & Moré, J. J. (1983). Estimation of sparse Jacobian matrices and graph coloring problems, *SIAM J. Numer. Anal.* 20(1), pp. 187–209.

Coleman, T. F., Garbow, B. S. & Moré, J. J. (1984). Software for estimating sparse Jacobian matrices, *ACM Trans. Math. Software* 10(3) pp. 329–345.

Curtis, A. R., Powell, M. J. D. & Reid, J. K. (1974). On the estimation of sparse Jacobian matrices, *J. Inst. Math. Appl.* 13, pp. 117–119.

Dickey, T. D. (2003). Emerging ocean observations for interdisciplinary data assimilation systems, *J. Mar. Syst. 40–41* pp. 5–48.

Elizondo, D., Cappelaere, B. & Faure, C. (2002). Automatic versus manual model differentiation to compute sensitivities and solve non-linear inverse problems, *Computers & Geoscience* 28, pp. 309–326.

Errico, R. M. (1997). What is an adjoint model?, *Bull. Amer. Met. Soc.* 78, pp. 2577–2591.

Evensen, G. (1994). Sequential data assimilation with a nonlinear quasi-geostrophic model using monte carlo methods to forecast error statistics, *J. Geophys. Res.* 99, pp. 10143–10162.

Giering, R. & Kaminski, T. (1998). Recipes for adjoint code construction, *ACM Trans. Math. Software* 24, pp. 437–474.

Giering, R. (2000). Tangent linear and adjoint biogeochemical models, in: P. Kasibhatla, M. Heimann, P. Rayner, N. Mahowad, R. Prinn & D. Hartley (Eds.), Inverse methods in global biogeochemical cycles, *American Geophysical Union*, pp. 33–48.

Griewank, A. (2000). Evaluating derivatives, Vol. 19 of Frontiers in Applied Mathematics, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, principles and techniques of algorithmic differentiation.

Homescu, C. & Navon, I. M. (2003). Numerical and theoretical considerations for sensitivity calculation of discontinuous flow, *Systems & Control Letters* 48, pp. 253–260.

Hossain, S. & Steihaug, T. (2002). Sparsity issues in the computation of Jacobian matrices, Tech. rep., Department of Informatics, University of Bergen, report 223.

Hovland, P., Mohammadi, B. & Bischof, C. (1998). Automatic differentiation and Navier-Stokes computations, in: Computational methods for optimal design and control (Arlington, VA, 1997), Vol. 24 of Progr. Systems Control Theory, Birkhäuser Boston, Boston, MA, pp. 265–284.

Martinsen, F., Biegler, L. T. & Foss, B. A. (2004). A new optimization algorithm with application to nonlinear MPC, *J. Proc. Cont.* 14(8), pp. 853–865.

Navon, I. M. (1997). Practical and theoretical aspects of adjoint parameter estimation and identifiability in meteorology and oceanography, *Dyn. Atmos. Oceans* 27, 55–79.

Nocedal, J. & Wright, S. J. (1999). Numerical optimization, Springer-Verlag, New York.

Park, S. K., Droegemeier, K. K. & Bischof, C. H. (1996). Automatic differentiation as a tool for sensitivity analysis of a convective storm in a 3-D cloud model, in: M. Berz, C. Bischof, G. Corliss & A. Griewank (Eds.), Computational differentiation: *Proceedings of the 2nd International Workshop* held in Santa Fe, NM, February 12–14, 1996. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, pp. 205–214, techniques, applications, and tools.

Robinson, A. R., Lermusiaux, P. F. J. & Sloan III, N. Q. (1998). Data assimilation, in: K. H. Brink & A. R. Robinson (Eds.), The sea, Wiley, NY., pp. 541–594.

Slagstad, D. & McClimans, T. (2005). Modelling the ecosystem dynamics of the Barents Sea including the marginal ice zone: I. Physical and chemical oceanography., *J. Mar. Syst.* Submitted.

Thuburn, J. & Haine, T. W. N. (2001). Adjoints of nonoscillatory advection schemes, *J. Comp. Phys.* 171, pp. 616–631.