Efficient sensitivities for the spin-up phase

Thomas Kaminski, Ralf Giering, and Michael Voßbeck

FastOpt, Schanzenstr. 36, 20357 Hamburg, Germany, (http://www.FastOpt.com)

Summary. In geosciences, it is common to spin up models by integrating with anually repeated boundary conditions. AD-generated code for evaluating sensitivities of the final cyclo-stationary state with respect to model parameters or boundary conditions usually includes a similar iteration for the derivative statements, possibly with a reduced number of iterations. We evaluate an alternative strategy that first carries out the spin-up, then evaluates the full Jacobian for the final iteration and from there applies the implicit function theorem to solve for the sensitivites of the cyclo-stationary state. We demonstrate the benefit of the strategy for the spin-up of a simple box-model of the atmospheric transport. We derive a heuristic inequality for this benefit, which increases with the number of iterations and decreases with the size of the state space.

Keywords: automatic differentiation, spin-up, sensitivities, source-to-source transformation, TAF, implicit function

1 Introduction

In geosciences, it is common to spin up models to a cyclo-stationary state with periodic boundary conditions (forcing) as is illustrated by Fig. 1. For instance, to simulate the global carbon dioxide distribution in the atmosphere, one runs an atmospheric transport model with a repeated seasonal cycle of carbon dioxide fluxes at the Earth's surface [15]. The spin-up is completed once the simulated seasonal cycle of atmospheric carbon dioxide no longer changes from one year to the next. Other examples are simulations of the global thermo-haline ocean circulation [17] or of the terrestrial biosphere [24]. Especially for coupled model integrations, required spin-up times are often prohibitively long. Sophisticated techniques have been devised to reduce them (see, e.g, [11, 16, 22]).

Often, the sensitivity of the equilibrium state (spin-up sensitivity) with respect to the forcing or internal parameters of the model is required. This

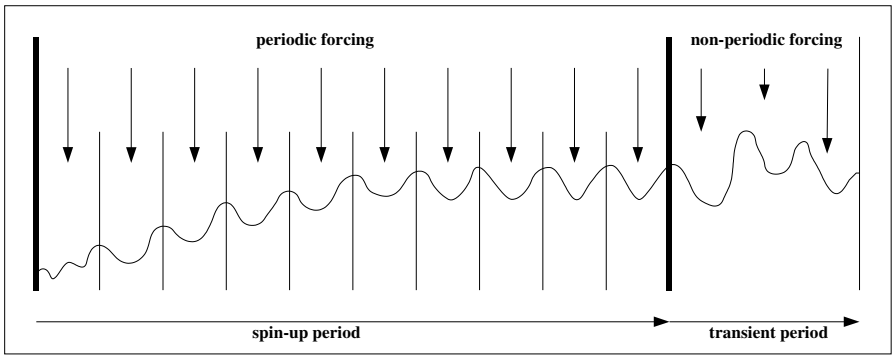

Fig. 1. Schematic representation of spin-up phase.

sensitivity might be interesting per se, or it might be a part of an extended sensitivity computation of other quantities that depend on the equilibrium state through a transient integration (see Fig. 1).

Spin-ups are also carried out in fields other than geosciences. For example, the simulation of a steady aerodynamic flow around an airfoil constitutes a special case of a spin-up integration with constant forcing (reflecting the airfoil's shape and the far field) and a period of one time step. Spin-up sensitivities provide important information for design optimisation.

Accurate spin-up sensitivities can be provided by automatic differentiation (AD,[9]). Applying AD in a straightforward way does, however, result in derivative propagation through the entire spin-up process, which is even more costly than the spin-up itself. Often, the computation is only rendered feasible at the cost of approximations in the model formulation. The terrestrial biosphere model BETHY [13, 14], which forms the core of the Carbon Cycle Data Assimilation System (CCDAS, see `http://CCDAS.org`), provides an example of such an approximation. Spinning up a pool for slowly decomposing organic carbon in soil is avoided by introducing a so-called β factor to parameterise the combined effect of both equilibrium pool size and turn-over time on the release of carbon dioxide [21, 23]. Using this approximation, only a fast decomposing soil carbon pool has to be spun up, which allows to reduce the spin-up time from thousands of years [24] to five years.

The present paper evaluates an alternative approach to sensitivity calculation for the spin-up phase, the *full Jacobian* approach, which requires the Jacobian for only a single year integration in equilibrium. The layout for the remainder of the paper is as follows: Sect. 2 formalises the spin-up process,

discusses various ways of computing spin-up sensitivities, and presents the *full Jacobian* approach. Sect. 3 describes its implementation, and Sect. 4 demonstrates its application to the spin-up of a simple atmospheric transport model. Sect. 5 analyses the computational efficiency of the *full Jacobian* approach. A summary and an outlook are given in Sect. 6.

2 Spin-up Sensitivities

Formally, integration of a model over a forcing-period of, say, one year can be represented by a function $f : R^p \times R^n \to R^n$ mapping the state x at January 1, 0 am, to next year's state at the same time, y:

$$y = f(b, x), \tag{1}$$

where b denotes input quantities other than the initial state such as boundary conditions or internal parameters of the model.

The spin-up is the iteration of (1), with y taking the role of x for the subsequent iteration. A necessary condition for terminating the spin-up is convergence of y to a fixed point x_e (equilibrium state), i.e. the equation

$$x_e = f(b, x_e) \tag{2}$$

must hold within a specified accuracy.

The *spin-up sensitivity* is the derivative of the equilibrium state x_e with respect to b. AD of the source code for (1) can provide derivative code to compute this sensitivity. The most obvious approach (*standard*/black-box AD) would be to differentiate the code of the entire spin-up phase, say in forward mode [9] of AD. Using, for instance, our AD-tool Transformation of Algorithms in Fortran (TAF [6, 7]), one would specify b as independent/input variable and x_e as dependent/output variable. TAF then generates code that iterates

$$\frac{dy}{db} = \frac{\partial f}{\partial b} + \frac{\partial f}{\partial x} \cdot \frac{dx}{db} \tag{3}$$

along with (1). In this derivative code each relevant function code statement is preceded by the corresponding derivative statement. For a given combination of b and initial value of x, the spin-up sensitivity is evaluated by running the derivative code with $\frac{dx}{db}$ initialised to zero. In case there is only one stable equilibrium, the results of both the spin-up iteration and the derivative iteration are independent of the initial value of the state, x. Griewank [9] provides a formal analysis of the convergence criteria for (3).

There are, however, more efficient strategies of providing spin-up sensitivities, based on the implicit function theorem for (2): Presuming that f is sufficiently regular and $\frac{\partial f}{\partial x}(b, x_e(b)) - Id$ is non-zero for a given b, then there exists a regular function $\tilde{b} \mapsto x_e(\tilde{b})$ around b and its derivative fulfils

```
subroutine f( p, b, n, x, y)
integer p, n
real b(p), x(n), y(n)
```

File 1: Header of subroutine (file `f.f`) implementing (1).

$$0 = \frac{\partial f}{\partial b}(b, x_e(b)) + (\frac{\partial f}{\partial x}(b, x_e(b)) - Id)\frac{dx_e}{db}(b), \tag{4}$$

where Id denotes the identity in R^n. $\frac{dx_e}{db}$ is determined by local properties of f around b and the equilibrium state x_e. [5, 9] suggest a *delayed derivative propagation* strategy (two-phase-AD), which does not turn on derivative propagation until the iteration of (1) converges well. Bücker et al. [2] apply this strategy to a CFD code.

Christianson [3, 4] analyses reverse mode (adjoint) AD of the iteration of (1) and suggests an efficient alternative adjoint, which only uses the required values (trajectory) [6] from the last iteration of (1) and thus considerably reduces the necessary resources for storing/recomputing required values. TAF implements automatic generation of the Christianson scheme, triggered by a TAF-loop directive [7].

All above strategies (*standard*, *delayed derivative propagation*, *Christianson*) are matrix-free, i.e. they employ products of the Jacobian $\frac{\partial f}{\partial x}$ with a vector to solve (3) for $\frac{dx_e}{db}$. The present study explores the alternative *full Jacobian* strategy of first running the spin-up, then computing the full Jacobian, i.e. $\frac{\partial f}{\partial b}(b, x_e(b))$ and $\frac{\partial f}{\partial x}(b, x_e(b))$, and finally solving (4) for $\frac{dx_e}{db}$.

3 Implementation

We sketch a Fortran implementation of the *full Jacobian* approach based on our AD tool TAF [6, 7]. It is instructive to consider first the simpler case of a sensitivity calculation that is restricted to the spin-up phase.

We start from the code of a single year integration, more precisely an implementation of (1) a subroutine form of. The header of the subroutine is shown in file 1. A single code for evaluating the two Jacobians required by (4), i.e. $\frac{\partial f}{\partial b}(b, x_e(b))$ and $\frac{\partial f}{\partial x}(b, x_e(b))$, is generated by applying TAF with command line options **-forward -pure -toplevel f -input b,x -output y -jacobian m -ftlmark _dbx f.f**, where $m = p + n$ (the sum of the dimensions of b and x) and `f.f` contains the source code of the subroutine `f`. The option **-pure** invokes TAF's pure mode, i.e. the derivative code does not include a function evaluation, and function code statements are only included where necessary to provide required values. TAF generates a subroutine `f_dbx` that evaluates the Jacobian.

A simple driver program that runs the code for the spin-up and its derivative is shown in file 2. Subroutine `spinup` performs the spin-up, including the iterative call of the subroutine `f` and a termination condition. The field `x0`

```
      real b(p), x0(n), xe(n), x(n), y(n)
      real x_dbx(p+n,n), b_dbx(p+n,p)
      real y_dbx(p+n,n), y_dx(n,n), y_db(p,n), x_db(p,n)
      b = ...
! spin-up
      x0 = 1.
      call spinup( p, b, n, x0, xe)
! initialisation of derivative objects
      do j =1,n
         do i=1,p+n
             x_dbx(i,j)=0.
         enddo
         x_dbx(p+j,j)=1.
      enddo
      do j =1,p
         do i=1,p+n
             b_dbx(i,j)=0.
         enddo
         x_dbx(j,j)=1.
      enddo
! Jacobian evaluation
      x=xe
      call f_dbx( p, b, b_dbx, n, x, x_dbx, y, y_dbx)
! separating the Jacobians
      do i=1,p
         y_db(i,:) = y_dbx(i,:)
      enddo
      do i=1,n
         y_dx(i,:) = y_dbx(p+i,:)
      enddo
! solve for spin-up sensitivity
      call solve (n, p, y_dx, y_db, x_db)
```

File 2: A driver program for solving first (2) and then (4) using the *full Jacobian* approach.

```
!$taf subroutine spinup   input = 1,2,3,4
!$taf subroutine spinup  output = 5
!$taf subroutine spinup  active = 2,5
!$taf subroutine spinup  depend = 1,2,3,4
!$taf subroutine spinup  adname = spinup_ad
!$taf subroutine spinup ftlname = spinup_tl
```

File 3: TAF flow directives for the subroutine `spinup`.

contains the initial value of the state and the field `x_e` its equilibrium value. The matrices $\frac{dx}{db}$ and $\frac{db}{dx}$ are initialised to zero, and $\frac{dx}{dx}$ and $\frac{db}{db}$ to the identities in R^n and R^p, respectively. The Jacobian is evaluated for $x = x_e$, i.e. at the equilibrium, then the result is split up into the two Jacobians and passed to a solver routine which finally returns $\frac{dx_e}{db}$.

For cases in which the Jacobian evaluation in reverse mode is preferable, the TAF command line and the driver need to be modified. The command line arguments **-forward -ftlmark** have to be replaced by **-reverse -admark**. In the driver, it is now the field `y_dbx` that has to be initialised, and the Jacobian is returned in the fields `b_dbx` and `x_dbx`.

Let's now address the case in which the spin-up is part of a larger computation and some sensitivity involving the entire computation is needed. We apply

TAF to the source code of the entire computation. To handle the spin-up, we use the TAF flow directives for the subroutine `spinup` (see file 3) which trigger inclusion of a calling sequence for an externally provided derivative routine of `spinup`. The forward mode then calls a routine `spinup_tl` while the reverse mode calls a routine `spinup_ad`. For details on TAF flow directives see [8].

The two routines `spinup_tl` and `spinup_ad` are hand-written wrappers. They first compute $\frac{dx_e}{db}$ as shown in file 2 and then carry out a matrix multiplication to propagate the derivative through the spin-up. The form of this matrix multiplication depends on the mode in which the entire code is differentiated. In forward mode, `spinup_tl` multiplies $\frac{dx_e}{db}$ from the right with the derivative of b with respect to the independents, and in reverse mode from the left with the derivative of the dependents with respect to x_e.

4 Numerical Example

As a test code, we employ "boxmod", a simple model of the atmospheric transport, which uses an Euler scheme to integrate the continuity equation for a passive trace gas. In this context, passive means that the concentration does not influence the atmospheric transport. The model is described in [20, 25] and its forward and reverse mode derivatives in [20]. There is one box for each hemisphere, their tracer concentrations take the role of x in (1). The inter-hemispheric mixing rate [20] of 1/year is its single parameter and corresponds to b in (1). We use boxmod in its methyl chloroform setup described in [20], with a uniform sink term corresponding to an inverse lifetime of 1/4.7 years [10]. We repeat the 1978 surface source estimates of Prinn et al. [19], modulated by a cosine with a period of one year and an amplitude of 10% the source strength. To mimic a large-scale application, the model is integrated with 10^7 time steps per year. We use the same initial concentration of 100 ppt (parts per trillion) for both boxes, which is about 10% off the equilibrium.

Table 1 shows the convergence of the spin-up in double precision, the first column lists the iteration number, and the next two columns both components of the state vector; the last column will be discussed later. For our *base* case we choose to iterate until the relative difference between y and x (as defined in (1)) is less than 10^{-7} for both components, which is reached after 49 iterations. We also look at a *low accuracy* case with a relative difference of 10^{-3}, which is reached after six iterations.

To compare the *standard* approach and the *full Jacobian* approach, two derivative codes are generated in TAF's pure forward mode (see Sect. 3). As our state vector has only two components, solving (4) for $\frac{dx_e}{db}$ is trivial. For our *base* case with 49 iterations, the relative difference in the spin-up sensitivities from both approaches is below 10^{-14}.

We also test solving (4) by iterating (3) (with the full Jacobian). The convergence of $\frac{dy}{db}$ is shown in the last column of table 1. After only 8 iterations the relative difference to the $\frac{dy}{db}$ value from *standard* is below 10^{-7}. Note that this

Table 1. Convergence of the spin-up.

Iteration	x(1)	x(2)	dx(1)/db
1	108.51645929	91.94163466	-7.3450784858
2	109.60820945	91.22018250	-8.1485930691
3	109.85705135	91.27066920	-8.2364935166
4	109.98888128	91.38080007	-8.2461093827
5	110.08786208	91.47740710	-8.2471613100
6	110.16704315	91.55632850	-8.2472763855
8	110.28261339	91.67186723	-8.2472903514
10	110.35811940	91.74737286	-8.2472905185
12	110.40745656	91.79671001	-8.2472905205
20	110.48351777	91.87277122	-8.2472905205
30	110.49845465	91.88770810	-8.2472905205
40	110.50023387	91.88948732	-8.2472905205
49	110.50043900	91.88969246	-8.2472905205
50	110.50044580	91.88969925	-8.2472905205
60	110.50047104	91.88972449	-8.2472905205
70	110.50047405	91.88972750	-8.2472905205
80	110.50047441	91.88972786	-8.2472905205
90	110.50047445	91.88972790	-8.2472905205
100	110.50047446	91.88972791	-8.2472905205

Table 2. Performance for derivatives of boxmod equilibrium state with respect to mixing rate.

Case	Spin-up [s]	Std AD [s]	Jacobian [s]	Std/Jac
base	4.4	5.9	0.17	35.2
ifort	4.4	5.3	0.21	25.0
low accuracy	0.53	0.72	0.17	4.3

procedure is different from *delayed derivative propagation*, as we are using the precomputed Jacobian for the equilibrium state. Also, *delayed derivative propagation* faces the decision when to start the derivative propagation without knowing how many iterations are still needed by the function code iteration to converge.

For the *low accuracy* case, the derivatives of both approaches (*standard* and *full Jacobian*) each have a relative difference below 10^{-4} to the 'true' sensitivity (from 100 iterations in the *standard* approach, see Table 1).

We run performance tests on a 3GHz Pentium 4 processor. Each test is repeated 10 times and the average run time recorded. Table 2 lists run times for three test cases. The cases *base* and *low accuracy* use the Lahey-Fujitsu Fortran 95 compiler lf95 with high optimisation level and double precision (flags **-O3 --dbl**). Case *ifort* equals the *base* case, with lf95 replaced by the Intel Fortran compiler, again with high optimisation level and double precision (flags **-O3 --autodouble**). The second column shows the run time for the spin-up integration, columns three and four refer to the *standard* and *full Jacobian* approaches, respectively. The last column shows the quotient of columns three and four. The relative run times depend strongly on the compiler, but also on compiler options and platform (not shown here). The *full Jacobian* approach is considerably faster than *standard*, even in the *low accuracy* case.

5 Performance analysis

In the box-model example, the *full Jacobian* strategy outscores the *standard* strategy considerably. Why is that? Let $r(m)$ denote the computational cost of a Jacobian product with m vectors for a single year run of boxmod. Our standard measure for computational cost of derivative code is CPU time in multiples of the CPU time spent for the evaluation of the underlying function. But let's use the number of operations for a moment. Then $r(\cdot)$ would be an affine function of m, i.e.

$$r(m) = r(1) + s \cdot (m - 1), \tag{5}$$

with $r(1)$ being the number of flops for the product of the Jacobian times the first vector and s being the number of flops per additional vector. The first vector is more expensive, because the computation of required values has to be included.

When returning to CPU time as performance measure, the form of $r(\cdot)$ depends on additional factors, most of which are platform and compiler dependent. Examples of such factors are data locality, vector length, level of compiler optimisation, and other compiler options. Also, from a certain m the computation will exceed the available memory and hence needs to be split up. A previous study [12] has tested the performance for TAF-generated code for forward and reverse mode Jacobian evaluations within CCDAS: In reverse mode, testing $r(m)$ for fourteen values of m between 1 and 96 (see Figure 4 of [12]) indicates that (5) is indeed a good approximation, with $s \approx 0.25$ and $r(1)$ between 3 and 4. In forward mode, $r(1) = 1.5$ and $r(58) = 12$ yield $s \approx 0.2$.

Our present example is a bit more complex as, in addition to increasing the number of Jacobian-vector products, we are also increasing the set of quantities with respect to which we are differentiating. On the other hand,

the state, x, is active [1, 6] even when differentiating only with respect to b (*standard*), i.e. derivatives of the state are propagated for both approaches *standard* and *full Jacobian*. We can estimate the extra cost for extending the Jacobian from derivatives with respect to b to derivatives with respect to b and x from the numbers in Table 2. With 10^7 time steps per year, we can safely neglect the CPU time spent outside boxmod and its derivatives. For the *base* case with its $k = 49$ iterations we have then $r(1) = (5.909/49)/(4.3651/49) \approx 1.35$, and $r(3) = 0.168/(4.3651/49) \approx 1.86$, which yields a slope s of about 0.25.

If we can neglect the cost of solving (4) and if (5) is a good approximation for capturing the cost of adding derivatives with respect to x, the *full Jacobian* strategy (left hand side) is preferable to *standard* (right hand side), if

$$k + r(1) + s \cdot (p + n - 1) \quad < \quad k \cdot (r(1) + s \cdot (p - 1)), \tag{6}$$

where p denotes the dimension of b. The left hand side k quantifies the cost of the spin-up itself. Since we have always $r(1) > 1$, the *full Jacobian* gets more favourable with increasing k, as seen in the example. Increasing n favours the *standard* approach, whereas increasing p favours the *full Jacobian* approach.

When increasing p, from a certain point the reverse mode is more efficient than the forward mode. This point depends on the number of dependent variables, q. If the dependent variables are the equilibrium state then $q = n$. The dependent variables may also be some function of the equilibrium state, including, e.g., a transient integration as illustrated in Fig. 1. In reverse mode, the *Christianson* approach is more efficient than the *standard* approach, as the former needs to provide fewer required values. If we denote the cost of evaluating the Jacobian times q vectors in reverse mode by $\tilde{r}(q)$ and its slope by $\tilde{s}$, the corresponding cost estimates for the *full Jacobian* and *Christianson* approaches are

$$k + \tilde{r}(1) + \tilde{s} \cdot (n - 1) \tag{7}$$

and

$$k \cdot (\tilde{r}(1) + \tilde{s} \cdot (q - 1)), \tag{8}$$

respectively.

We can illustrate (6) with numbers from [7], where the authors apply TAF to a Navier-Stokes solver, in a simple configuration with $k = 500$, $n = 5 \times 801$, $p = 2$ (Mach number and angle of attack), and $q = 1$ (scalar objective function of lift and drag). With their $r(1) = 2.4$ and an assumed $s = 0.25$, (6) yields a cost of about 1500 for the *full Jacobian* approach versus about 1300 for the *standard* approach. As the *Christianson* approach yields $\tilde{r}(1) = 3.4$, the *standard* approach (in forward mode) may be preferable up to $p = 5$ (using the assumed $s = 0.25$). For any larger p, (8) yields a cost of 1700. Hence, at first glance, the *full Jacobian* appears slightly favourable for $5 < p < 800$.

For $n = 5 \times 801$ we cannot, however, ignore the cost of solving (4). The cost of standard methods for solving systems of linear equations (e.g. LU decomposition) grows with n^3 [18]. Solving (4) iteratively (as done in Sect. 4), requires one matrix-vector product in R^n per iteration. The only relevant component of the spin-up sensitivity $\frac{dx_e}{db}$ is given by the derivative of the objective function with respect to the equilibrium state. One can, thus, focus on convergence of that component.

The comparison looks much different for the spin-up of terrestrial biosphere models. As there is no exchange of information across borders of grid cells, the Jacobian $\frac{\partial f}{\partial x}$ has a block diagonal structure, with the block size equal to the dimension of the state space per grid cell. This dimension, n_{eff}, is the effective size of state space to be used in (6), as the sparse Jacobian can be retrieved from n_{eff} Jacobian-vector products. Also, (4) can be solved block by block. As n_{eff} is usually below 10, the computation of $\frac{\partial f}{\partial x}$ as well as solving (4) are inexpensive. Regarding s, one can, for instance, assume the above mentioned value for CCDAS of $s \approx 0.2$. With $n = 5$ and $k = 3000$ the cost of the *full Jacobian* approach is dominated by the cost of the spin-up itself, which is 3000. The sensitivities come at an extra cost of only about 2.5.

6 Summary and Outlook

We have explored the *full Jacobian* approach to the computation of sensitivities for the spin-up phase. For a simple box-model of the atmospheric transport, the *full Jacobian* approach is 4 to 35 times more efficient than the *standard* approach of propagating derivatives through the entire spin-up phase. This benefit increases with the number of iterations and decreases with the size of the state space.

We have shown that for terrestrial biosphere models, which are characterised by a low dimensional effective state-space, the *full Jacobian* approach looks promising. As a first large-scale application, we plan to compute spin-up sensitivities for a biosphere model within CCDAS.

Acknowledgements

The idea for this study originates from a discussion with Srikanth Akkaram on sensitivities of a structural mechanics code. Wolfgang Knorr and Marko Scholze have provided valuable advice and comments. Part of this work has been carried out in the project CAMELS, supported by the EU under contract no. EVK2-CT-2002-00151 within the 5th Framework Programme for Research and Technological Development.

References

1. Christian H. Bischof, Alan Carle, Peyvand Khademi, and Andrew Mauer. ADIFOR 2.0: Automatic differentiation of Fortran 77 programs. *IEEE Computational Science & Engineering*, 3(3):18–32, 1996.
2. H. M. Bücker, A. Rasch, E. Slusanschi, and C. H. Bischof. Delayed Propagation of Derivatives in a Two-dimensional Aircraft Design Optimization Problem. In D. Snchal, editor, *Proceedings of the 17th Annual International Symposium on High Performance Computing Systems and Applications and OSCAR Symposium, Sherbrooke, Qubec, Canada, May 11-14*, pages 123–126. NRC Research Press, 2003.
3. Bruce Christianson. Reverse accumulation and attractive fixed points. *Optimization Methods and Software*, 3:311–326, 1994.
4. Bruce Christianson. Reverse accumulation and implicit functions. *Optimization Methods and Software*, 9(4):307–322, 1998.
5. Shaun A. Forth and Trevor P. Evans. Aerofoil optimisation via ad of a multigrid cell-vertex Euler flow solver. In George Corliss, Christèle Faure, Andreas Griewank, Laurent Hascoët, and Uwe Naumann, editors, *Automatic Differentiation: From Simulation to Optimization*, Computer and Information Science, chapter 17, pages 153–160. Springer, New York, 2001.
6. R. Giering and T. Kaminski. Recipes for Adjoint Code Construction. *ACM Trans. Math. Software*, 24(4):437–474, 1998.
7. R. Giering, T. Kaminski, and T. Slawig. Applying TAF to a Navier-Stokes solver that simulates an Euler flow around an airfoil. *To appear in Future Generation Computer Systems*, 2005.
8. R. Giering, T. Kaminski, R. Todling, R. Errico, R. Gelaro, and N. Winslow. Generating tangent linear and adjoint versions of NASA/GMAO's Fortran-90 global weather forecast model. In H. M. Bücker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, editors, *Automatic Differentiation: Applications, Theory, and Tools*, Lecture Notes in Computational Science and Engineering. Springer, 2005.
9. A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, Philadelphia, 2000.
10. S. Houweling, Frank Dentener, and Jos Lelieveld. The impact of nonmethane hydrocarbon compounds on tropospheric photochemistry. *J. Geophys. Res.*, 103(D9):10673–10696, 1998.
11. Tim C. Johns, Ruth E. Carnell, Jenny F. Crossley, Jonathan M. Gregory, John F. B. Mitchell, Catherine A. Senior, Simon F. B. Tett, and Richard A. Wood. The Second Hadley Centre coupled ocean-atmosphere GCM: Model description, spinup and validation. *Clim. Dyn.*, 13:103–134, 1997.
12. T. Kaminski, R. Giering, M. Scholze, P. Rayner, and W. Knorr. An example of an automatic differentiation-based modelling system. In V. Kumar, L. Gavrilova, C. J. K. Tan, and P. L'Ecuyer, editors, *Computational Science – ICCSA 2003, International Conference Montreal, Canada, May 2003, Proceedings, Part II*, volume 2668 of *Lecture Notes in Computer Science*, pages 95–104, Berlin, 2003. Springer.
13. W. Knorr. *Satellitengestützte Fernerkundung und Modellierung des Globalen CO_2 -Austauschs der Landvegetation: Eine Synthese*. PhD thesis, Max-Planck-Institut für Meteorologie, Hamburg, Germany, 1997.

14. W. Knorr. Annual and interannual CO_2 exchanges of the terrestrial biosphere: process based simulations and uncertainties. *Glob. Ecol. and Biogeogr.*, 9:225–252, 2000.
15. R. M. Law, P. J. Rayner, A. S. Denning, D. Erickson, M. Heimann, S. C. Piper, M. Ramonet, S. Taguchi, J. A. Taylor, C. M. Trudinger, and I. G. Watterson. Variations in modelled atmospheric transport of carbon dioxide and the consequences for CO_2 inversions. *Global Biogeochem. Cycles*, 10:783–796, 1996.
16. S. Manabe and R. Stouffer. Two climate equilibria of a coupled ocean-atmosphere model. *Climate Dyn.*, 1:841–866, 1988.
17. M. Nakamura, P. H. Stone, and J. Marotzke. Destabilization of the thermohaline circulation by atmospheric eddy transports. *J. Climate*, 7:1870–1882, 1994.
18. William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipies in Fortran. 2nd edn.* Cambridge University Press, Cambridge, 1992.
19. Prinn et al. Global average concentration and trend for hydroxil radical deduction from ALE/GAGE trichloroethane (methyl chloroform) data for 1987–1990. *J. Geophys. Res.*, 97:2445–2461, 1992.
20. P. Rayner, R. Giering, T. Kaminski, R. Ménard, R. Todling, and C. Trudinger. Exercises. In P. Kasibhatla, M. Heimann, D. Hartley, P. J. Rayner, N. Mahowald, and R. Prinn, editors, *Inverse Methods in Global Biogeochemical Cycles, Geophys. Monogr. Ser.*, volume 114, pages 324–347. Washington, D. C., 1999.
21. P. Rayner, M. Scholze, W. Knorr, T. Kaminski, R. Giering, and H. Widmann. Two decades of terrestrial Carbon fluxes from a Carbon Cycle Data Assimilation System (CCDAS). *Accepted for publication in to Global Biogeochemical Cycles*, 2004.
22. Andreas Schmittner and Thomas F. Stocker. A seasonally forced ocean-atmosphere model for paleoclimate studies. *J. Climate*, 14:1055–1068, 2001.
23. M. Scholze. *Model studies on the response of the terrestrial carbon cycle on climate change and variability.* Examensarbeit, Max-Planck-Institut für Meteorologie, Hamburg, Germany, 2003.
24. S. Sitch, I. C. Prentice, B. Smith, A. Arneth, A. Bondeau, W. Cramer, J. O. Kaplan, S. Levis, W. Lucht, M. T. Sykes, K. Thonicke, and S. Venevsky. Evaluation of ecosystem dynamics, plant geography and terrestrial carbon cycling in the LPJ dynamic global vegetation model. *Global Change Biology*, 9:161–185, 2003.
25. P. P. Tans. A note on isotopic ratios and the global atmospheric methane budget. *Global Biogeochem. Cycles*, 11:77–81, 1997.