doi: 10.1111/j.1365-246X.2007.03400.x

Automatic differentiation in geophysical inverse problems

M. Sambridge, P. Rickwood, N. Rawlinson and S.Sommacal

Research School of Earth Sciences, Australian National University, Canberra, ACT 0200*, Australia. E-mail. malcolm.sambridge@anu.edu.au*

Accepted 2007 February 13. Received 2007 February 11; in original form 2006 October 29

SUMMARY

Automatic differentiation (AD) is the technique whereby output variables of a computer code evaluating any complicated function (e.g. the solution to a differential equation) can be differentiated with respect to the input variables. Often AD tools take the form of source to source translators and produce computer code without the need for deriving and hand coding of explicit mathematical formulae by the user. The power of AD lies in the fact that it combines the generality of finite difference techniques and the accuracy and efficiency of analytical derivatives, while at the same time eliminating 'human' coding errors. It also provides the possibility of accurate, efficient derivative calculation from complex 'forward' codes where no analytical derivatives are possible and finite difference techniques are too cumbersome. AD is already having a major impact in areas such as optimization, meteorology and oceanography. Similarly it has considerable potential for use in non-linear inverse problems in geophysics where linearization is desirable, or for sensitivity analysis of large numerical simulation codes, for example, wave propagation and geodynamic modelling. At present, however, AD tools appear to be little used in the geosciences. Here we report on experiments using a state of the art AD tool to perform source to source code translation in a range of geoscience problems. These include calculating derivatives for Gibbs free energy minimization, seismic receiver function inversion, and seismic ray tracing. Issues of accuracy and efficiency are discussed.

Key words: Derivatives, inverse problems, sensitivity kernel.

1 INTRODUCTION

Over the past 20 yr numerical simulation of geophysical processes has become widespread. The availability of high performance computing and advanced numerical techniques has led researchers to develop increasingly sophisticated numerical algorithms for modelling physical processes such as deformation within the Earth's lithosphere (Moresi *et al.* 2003), convection within the deep interior (Moresi *et al.* 2000) and propagation of seismic waves through fully 3-D models of the whole Earth (Komatitsch *et al.* 2005). The primary use of such tools is to make predictions that can be compared to observations. An issue of much concern is to determine the sensitivity of these predictions to the input parameters upon which they are based, for example, how a synthetic seismogram calculated for a particular earthquake at a particular point on the surface varies with parameters controlling the source, or the assumed earth model between source and receiver. Here sensitivity means the derivatives, or Jacobian, of the output variables with respect to input variables. Many areas of the sciences have need of derivatives and indeed one could argue that there is often little point in knowing what a mathematical model predicts if one doesn't know the sensitivity of that prediction to the parameters controlling the model.

Another area where derivatives are much used is inversion of geophysical data (Aster *et al.* 2005; Tarantola 2005). In this case one fits models of the Earth to observations, often in an automated manner making use of optimization techniques, which themselves depend on derivatives. Not all inversion techniques are based on derivatives (see Sambridge & Mosegaard 2002), but if the number of unknowns is large, say in the 100 s or greater, and the mathematical relationship between data and the unknowns is non-linear, then derivative calculations are unavoidable. In non-linear inverse problems such as seismic tomography of the Earth's mantle and lithosphere (see Rawlinson & Sambridge 2003, for a recent review), iterative algorithms making use of derivative information have become the norm over the past 10 yr.

The calculation of derivatives can be problematic in many cases. If analytical expressions are available then these can be hand coded. This has the (potential) advantage of efficiency in the final code, but at the cost of much development time (in debugging hand coded expressions) and a certain amount of inflexibility. For example, in an inverse problem, if the misfit function, or parametrization, is changed then derivatives must be recoded. If no analytical formulae are available then one often resorts to numerical estimates of derivatives, for example, using finite difference techniques. In this case one would typically perturb each input variable one at a time by a specified amount and recalculate the output variables with the 'source' code. This has the advantage of flexibility since there is no dependence on the details of the source code, however, the accuracy will depend on the size of the perturbation and may require tuning. Furthermore, the overall computational cost scales with the number of input variables. For computationally intensive source codes with large numbers of input variables (e.g. 10^2–10^6) this can become prohibitive.

With these issues in mind, one might define an 'ideal' derivative method as having the following properties.

(i) Calculates exact derivatives (i.e. to within machine precision).
(ii) Computational cost independent of the number of input variables.
(iii) Minimal human effort to generate and update a Jacobian.

Computer scientists have grappled with these issues for many years and the field of automatic differentiation (AD) has emerged as a result. To date these developments seem to have made little impact on the Geosciences. The purpose of this paper is to help bring AD tools to the attention of the wider geoscience community, provide some useful references and illustrate its potential through a series of numerical experiments on common geoscience problems. We begin with a brief outline of how current AD methods work. We then present results of our experiments applying current state-of-the art AD tools to particular geoscience problems. Performance is evaluated in terms of the three ideals above, with particular attention given to issues of accuracy and efficiency.

2 ELEMENTS OF AUTOMATIC DIFFERENTIATION

The origins of AD date back to the 1950s and 1960s, and in particular to the works of Beda *et al.* (1959),Wengert (1964). Modern developments can be traced to a resurgence in activity in the 1970s and 1980s. Reviews can be found in Rall (1981), Griewank (1991), Rall & Corliss (1996), Griewank (2000) and Cusdin & Müller (2003), while a recent work containing many applications of AD to engineering and optimization, as well as an extensive bibliography, is Corliss *et al.* (2002). A comprehensive online resource has also emerged and may be found at http://www.autodiff.org.

The underlying principle of AD is that computer programs can be decomposed into a finite set of instructions (usually executed serially). The source code then represents a (composite) function which takes a set of n input variables and produces a set of m output variables thorough application of a set of elementary functions. The Jacobian of each elementary function can be obtained using simple rules, for example, from tables of differentiation (see Talagrand 1991; Giering & Kaminski 1998). Repeated application of the chain rule of differentiation yields the desired Jacobian of the composite function. An important issue here is that at the elementary level differentiation can be carried out exactly, that is, without the need to introduce numerical approximations like finite difference operators, (an example is given below). From the user's perspective modern AD tools act the same way as a compiler, but rather than producing object or executable code, they create a second source code which evaluates the derivatives of the original function for any values of the input variables. In this sense 'source to source' translator AD tools directly replace the hand coding of derivatives (see Fig. 1).

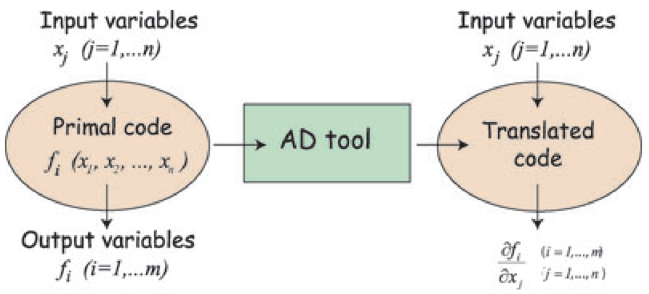


Figure 1. Schematic illustration of how a code-to-code translation is performed by an AD tool.

2.1 Forward mode

AD methods are divided into two types depending on the order in which they evaluate the elementary functions and their Jacobians. The forward or 'Tangent linear' mode evaluates them in the same order as the source (often called 'primal') code. This can be easier to implement and efficient on storage because once variables are no longer needed in the source code they can be discarded (overwritten) which reduces the storage needed to keep track of associated derivative variables. Fig. 2 shows an example of the elementary operations (code list) required to evaluate a simple trigonometrical function

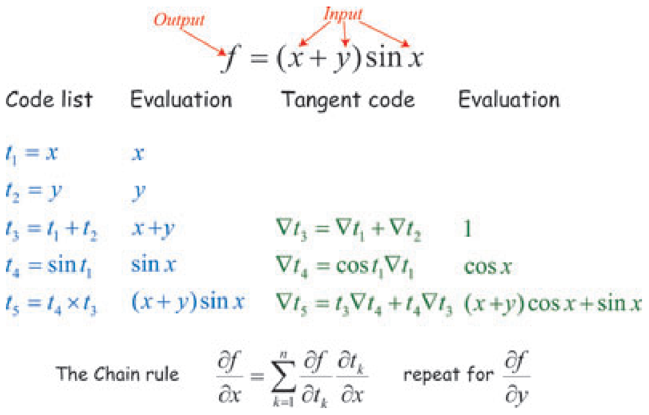
$$f = (x + y)\sin x. \tag{1}$$

As can be seen this is decomposed into five steps involving five separate variables (t_i, $i = 1, \dots, 5$). (Note that $t_1 = x$, $t_2 = y$ and $t_5 = f$). The corresponding tangent code for each step is shown in the central column and the equivalent mathematical expression for the derivative $\frac{\partial t_i}{\partial x}$ is shown in the fourth column. For the input variable x the tangent code list is initialized with

$$\nabla t_1 = 1, \nabla t_2 = 0. \tag{2}$$

As each step proceeds a new intermediate variable t_i is determined and the corresponding derivatives with respect to the input variable are found by evaluating the expression in the third column. By the time the fifth line is completed the required derivative of f has been assembled. The whole process is repeated for the second input variable y by changing the initial values to

$$\nabla t_1 = 0, \nabla t_2 = 1. \tag{3}$$


Output
Input
$f = (x + y)\sin x$

Code list	Evaluation	Tangent code	Evaluation
$t_1 = x$	x		
$t_2 = y$	y		
$t_3 = t_1 + t_2$	$x+y$	$\nabla t_3 = \nabla t_1 + \nabla t_2$	1
$t_4 = \sin t_1$	$\sin x$	$\nabla t_4 = \cos t_1 \nabla t_1$	$\cos x$
$t_5 = t_4 \times t_3$	$(x+y)\sin x$	$\nabla t_5 = t_3 \nabla t_4 + t_4 \nabla t_3$	$(x+y)\cos x + \sin x$

The Chain rule $\frac{\partial f}{\partial x} = \sum_{k=1}^{n} \frac{\partial f}{\partial t_k}\frac{\partial t_k}{\partial x}$ repeat for $\frac{\partial f}{\partial y}$

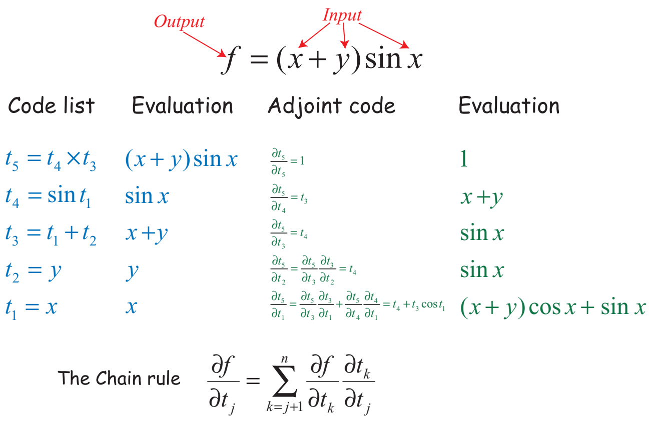
Figure 2. Example of AD of a simple function in forward mode. The first column shows the evaluation of the expression in eq. (1) broken down into five steps each defining a new variable $t_i (i = 1 \dots, 5)$. The second column shows corresponding expressions being evaluated. The third column is the differentiation of each intermediate variable with respect to input variable (x or y) using the chain rule expression written at the base of the figure. The fourth column shows the derivative expression corresponding to each line of the third column for the case where x is the independent variable. After a sweep through all five statements the derivative expression for x is constructed. The third column can then be repeated for y.

It is clear from this example that in forward mode the computational cost of evaluating the Jacobian is proportional to the number of input variables n, multiplied by the number of steps in the code list, the latter often being linear in n also. This is the same as a finite difference operator, however the accuracy of the resulting derivatives will be at machine precision due to the exact differentiation at the elemental level. Forward mode then satisfies property 1 and 3 of an idealized method but not property 2 (see Section 1). [Note, however, very recent developments of AD include a *vector mode* which involves simultaneous propagation of all partial derivatives (see Kaminski *et al.* 2003, for an example).]

2.2 Reverse mode

The second way of performing AD is to reverse the order in which the elementary functions and their Jacobians are determined. This is known as the reverse or 'Adjoint' mode. Again at the elemental level exact derivatives are calculated, but they are combined in the opposite order to before. Fig. 3 shows how reverse mode is carried out for the simple expression in (1). In this case the code list in the first two columns are reversed with respect to the forward case shown in Fig. 2. However, in the ith line of the third column the expression for the derivative $\frac{\partial f}{\partial t_i}$ is evaluated using the chain rule. By the time the fifth line is completed, expressions for the derivatives of $f(= t_5)$ with respect to both $x(= t_1)$ and $y(= t_2)$ have been determined. Hence with a single reverse sweep through the code list the derivatives of an output variable with respect to all input variables are available. The whole procedure must be repeated for every output variable, and hence a major advantage of the reverse mode is that the computational cost of evaluating the derivatives is proportional to the number of output variables m (multiplied by the length of the code list) rather than the number of input variables. We see then that unlike forward mode, the reverse mode of AD is the only one which can satisfy all three properties of an ideal method.

Implementation of this mode for arbitrary source codes presents a major challenge because the information flow must be reversed. In the simple example above this is trivial, however to do it in an automatic manner for a complicated code can become a challenging task, and this is a subject of ongoing research (Giering & Kaminski 2002). The simple example above highlights one of the main difficulties. At each stage of the reverse list, evaluation of the elemental Jacobians in the third column requires numerical values of the corresponding intermediate variables ($t_i, i = 1, \ldots, n$). (In general the entire state of the code including the values of all variables needs to be known at each time step.) In the forward mode this is not a problem, because the state of all variables is updated as the source code proceeds (using the primal code). However, in reverse mode variables are needed in the reverse order in which they are calculated, which can be a major obstacle if variables get overwritten. The situation is illustrated schematically in Fig. 4, where each dot represents a state of the source code. As time proceeds we move from the far right of the dotted line, the final state, to the initial state on the left.


Output
Input

$$f = (x + y)\sin x$$

Code list	Evaluation	Adjoint code	Evaluation
$t_5 = t_4 \times t_3$	$(x+y)\sin x$	$\frac{\partial t_5}{\partial t_5} = 1$	1
$t_4 = \sin t_1$	$\sin x$	$\frac{\partial t_5}{\partial t_4} = t_3$	$x+y$
$t_3 = t_1 + t_2$	$x+y$	$\frac{\partial t_5}{\partial t_3} = t_4$	$\sin x$
$t_2 = y$	y	$\frac{\partial t_5}{\partial t_2} = \frac{\partial t_5}{\partial t_3}\frac{\partial t_3}{\partial t_2} = t_4$	$\sin x$
$t_1 = x$	x	$\frac{\partial t_5}{\partial t_1} = \frac{\partial t_5}{\partial t_3}\frac{\partial t_3}{\partial t_1} + \frac{\partial t_5}{\partial t_4}\frac{\partial t_4}{\partial t_1} = t_4 + t_3 \cos t_1$	$(x+y)\cos x + \sin x$

The Chain rule $$\frac{\partial f}{\partial t_j} = \sum_{k=j+1}^{n} \frac{\partial f}{\partial t_k}\frac{\partial t_k}{\partial t_j}$$

Figure 3. Example of AD of a simple function in reverse mode. The first two columns are the reverse of those in Fig. 2. The third column shows how the chain rule expression at the base is applied to each line. The fourth column shows the corresponding evaluation of the third column. Note that after a single sweep through derivatives of f with respect to both y (fourth line) and x (fifth line) have been constructed.

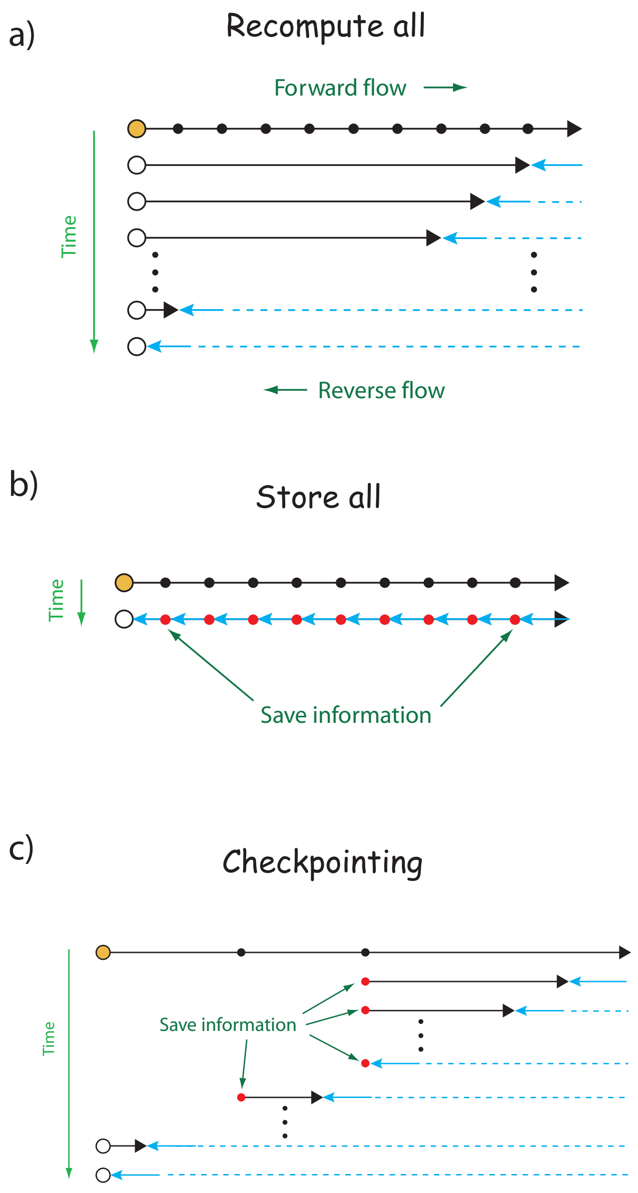


Figure 4. Schematic illustration of strategies for recovering the state of a computational system required by reverse mode of AD. (a) Recalculate all strategy, (b) store all strategy and (c) the checkpointing strategy which is a combination of the two.

Fig. 4(a) shows the 'recalculate all' strategy whereby the source code is restarted from the beginning each time the state changes.

This involves a large amount of computation, especially if there are a large number of states, but it requires relatively little extra storage. A second strategy is to store the complete values of all intermediate variables at every state, illustrated by Fig. 4(b). In this case there is little extra computation but the amount of storage required may be prohibitive. A compromise between the two extremes is a checkpointing strategy, illustrated by Fig. 4(c). Here the state is preserved at only selected points and recalculation proceeds from the last checkpointed position to the current state. Again it can be a complex task striking the best balance between these approaches. The main differences between various AD tools lies in whether they perform forward or reverse mode and if the latter, what type of strategy they use to recover the program state.

As has been noted previously (e.g. Griewank 1989) adjoint or reverse mode of AD is closely related to the determination of sensitivity kernels through adjoint differential equations. The latter has been used to determine partial derivatives in a range of areas where numerical simulation of complicated phenomena are involved. Examples include applications in nuclear reactor design for derivatives of temperature with respect to thousands of design parameters (Cacuci 1981a,b); in meteorology and oceanography for gradients of residual norms with respect to initial conditions (Talagrand & Courtier 1987; Thacker & Long 1988); in seismology for derivatives of seismic waveform residuals with respect to Earth structure parameters (Tarantola 1984, 1988; Tromp *et al.* 2005; Sieminski *et al.* 2007), and also in geodynamic calculations of mantle convection (Bunge *et al.* 2003). In each case the adjoint of the governing partial differential equation (PDE) is solved numerically and combined with the numerical solution of the original PDE to yield the required partial derivatives. For this reason the approach is often referred to as the continuous adjoint method, whereas the reverse mode of AD is often called the discrete adjoint method, since it is a line by line application of the chain rule. In principal, the discrete adjoint method has an advantage over the continuous approach in that it can avoid the build up of numerical error associated with solving of the differential equations. (With the discrete adjoint all calculations are essentially at machine precision). However, the continuous adjoint can be relatively straightforward to implement involving only a few repeat runs of the primal code. More importantly however, both have the property that the computational cost scales directly with the number of output parameters and not with the number of input parameters.

At present awareness of AD tools seems to be rather limited in the Earth Sciences. To our knowledge the first application in this field was in a PhD thesis by Sommacal (2004) for calculating derivatives in a Gibbs free energy minimization problem. (This is one of the test cases presented below.) More recently, applications have appeared in inverse problems in groundwater flow (Rath *et al.* 2006), and geodynamic modelling (Iaffaldano *et al.* 2007). Below we present results of some experiments with one of the most advanced AD tools available. This is known as Transformation of Algorithms in Fortran (TAF, available from http://www.fastopt.de) see Giering & Kaminski (2003). In all cases we compare performance to finite difference estimates, since this is a comparably general approach, requiring little human intervention.

3 NUMERICAL EXPERIMENTS

3.1 Gibbs free energy minimization

A common problem in igneous and metamorphic petrology is the calculation of equilibrium phase assemblages as a function of temperature, pressure and composition. Differing formulations of this problem have been proposed together with algorithms for their solution (e.g. Storey & van Zeggeren 1964; De Capitani & Brown 1987; Harvie *et al.* 1987). All of these use optimization techniques based on gradients for minimizing the Gibbs free energy. In this example we follow the formulation of Sommacal (2004) and write the Gibbs free energy of a single phase ϕ as a sum of three terms,

$$G^{\phi} = G^{\text{end-members}} + G^{\text{ideal}} + G^{\text{excess}}, \tag{4}$$

where $G^{\text{end-members}}$ is the free energy of its end-member constituents, G^{ideal} is the free energy due to an idealized configurational entropy from mixing, and G^{excess} is the excess free energy of mixing (see works cited above for details). Each of the these terms depend on various thermodynamic variables including the number and type of phases present in the system. The total Gibbs free energy must be minimized as a function of the variables $X_{i,j}$, which are the site occupancies of cation i on sublattice j. However, only the second and third terms depend on $X_{i,j}$. Specifically, we have

$$G^{\text{ideal}} = RT \sum_{j=1}^{p} q_j \sum_{i=1}^{n} X_{i,j} \ln X_{i,j} \tag{5}$$

$$\begin{aligned} G^{\text{excess}} = & \sum_{j=1}^{p} q_j \sum_{i=1}^{n} \sum_{l>i}^{n} X_{i,j} X_{l,j} \left\{ W_{i,l,j} \left[X_{l,j} + 1/2 \sum_{k\neq i,k\neq l}^{n} X_{k,j} \right] \right. \\ & \left. + W_{l,i,j} \left[X_{i,j} + 1/2 \sum_{k\neq i,k\neq l}^{n} X_{k,j} \right] \right\} \\ & + \sum_{i=1}^{n} \sum_{l>i}^{n} \sum_{k>l}^{n} X_{i,j} X_{l,j} X_{k,j} W_{i,l,k,j}, \end{aligned} \tag{6}$$

where R is the gas constant, T is temperature, q_j are stoichiometry factors which have values 1 or 2 depending on the type of phase, and $W_{l,i,j}$, $W_{i,l,k,j}$ are the three and four index Margules parameters describing the various interaction energies between sites. Summation occurs over sublattices and cations. All parameters other then the site occupancies $X_{i,j}$ may be treated as constants for our purposes, as minimization is performed with respect to these parameters only. The Gibbs free energy over all phases present in a system, G, is a weighted sum of G^{ϕ} terms. The Gibbs free energy has to be minimized (subject to certain constraints) and this can be done with gradient methods. Hence the derivatives $\frac{\partial G}{\partial X_{i,j}}$ are required. Since this is an analytical expression then it may in principal be differentiated analytically and coded by hand into a derivative routine. This would be both tedious and prone to human coding errors.

We applied the TAF AD tool to generate derivative code in both forward and reverse mode to this problem (note the source code was written in Fortran77). In this case we have one output variable, G, and by varying the number of phases present we can adjust the number of input variables, $X_{i,j}$. Fig. 5 shows the relative compute time for execution of the resulting derivative code as a function of the number of input variables. In each case the same derivatives were also computed with finite differences (after some tuning of the perturbation to achieve accuracy). The total number of derivatives calculated across all trials was 638. For forward mode we observe a quadratic dependence of CPU time on number of independent variables, N, for both the AD generated code and the finite difference code. This is consistent with theoretical predictions. From (5) and (6) we see that the compute cost of evaluating the Gibbs free energy depends linearly on the number of input variables, N. Hence if the derivative calculation requires of the order of N forward solutions the overall dependence will be quadratic in N. We note that in all cases the AD generated code is more efficient than the finite difference

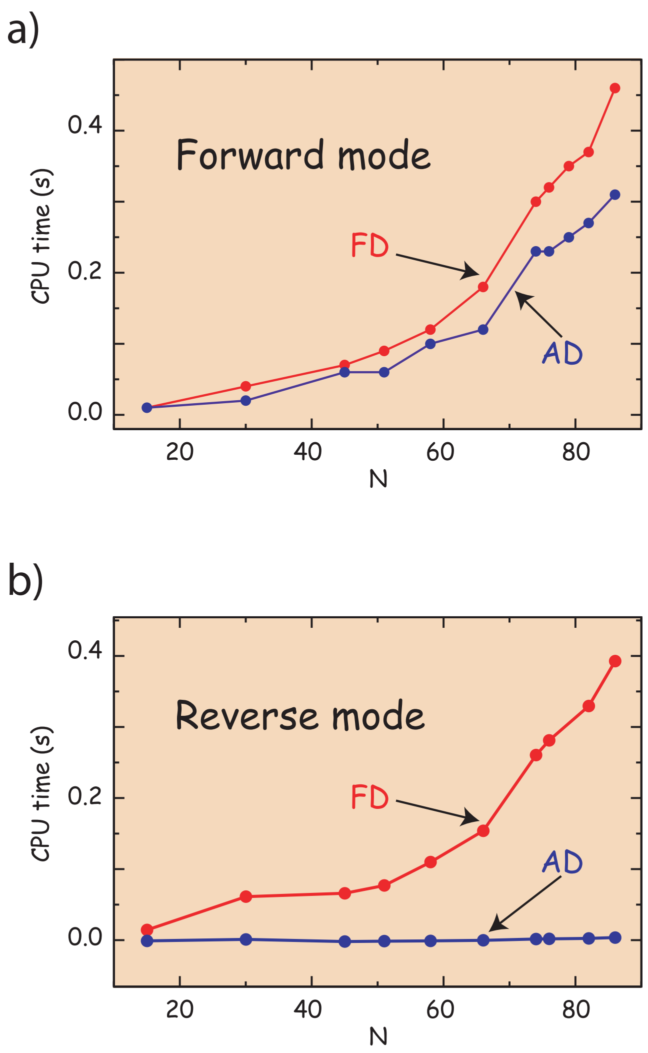


Figure 5. Scaling of CPU time with number of input variables to evaluate the Jacobian for the Gibbs free energy objective function, (a) in forward mode and (b) in reverse mode. Note that the observed CPU time for the forward mode scales approximately linearly with the N while in reverse mode it is independent of N, as predicted by theory.

derivatives (by up to a factor of 2). Given that the average difference in derivative values is less than 0.006 per cent, we see that in forward mode the AD tool has automatically generated code which gives correct values and is an efficient alternative to finite differences.

In reverse mode Fig. 5 clearly shows the independence between the CPU time for derivative evaluation, and the number of input variables. Again this is as predicted by theory, but nevertheless quite remarkable. As N grows the ratio of compute times varies between 4.1 and 41.2. Differences between derivative values for forward and reverse mode are within machine precision. In this case the AD tool has managed to automatically reverse the flow of the code and generate derivative routines whose computational cost is independent of the number of input parameters.

3.2 Seismic receiver functions

Our second test problem arises in the inversion of seismic body waveforms for crustal structure. Receiver function inversion (Langston 1989; Ammon *et al.* 1990) is a much used technique for estimating shear wave speeds in the Earth's crust as a function of depth beneath a seismic recorder. Receiver functions are derived from observed seismograms of distant events and may be compared to those predicted from a crustal velocity profile. A misfit function measuring the discrepancy between the two waveforms is specified, and minimized as a function of the parameters controlling the velocity model (see Fig. 6). Here the profile consists of a series of six linear segments with depth. Also there is the potential for discontinuities in wave speed between layers, which are themselves of variable thickness (see Shibutani *et al.* 1996, for full details of the parametrization).

Receiver function inversion is similar to other waveform fitting problems in seismology, where the relationship between parameters describing the Earth model are non-linearly related to the observed seismograms. Fig. 6 shows a waveform misfit surface (based on the L2 norm of the waveform discrepancy) as a function of two of the velocity model parameters. In this case the surfaces are clearly not quadratic. Their multimodal character means that gradient methods of optimization require many iterations and hence many derivative evaluations.

In contrast to the Gibbs free energy example the evaluation of the synthetic receiver function from a given velocity profile is purely numerical, that is, no analytical expression is available. Here we use a standard approach known as the Thomson-Haskell method (Thomson 1950; Haskell 1953). For this problem we applied the AD tool in forward mode to generate derivative code; in this example there is one output variable (the waveform misfit function) and 24 input variables (describing the velocity profile). Again the accuracy was compared to finite differences. With the latter we experienced some difficulty in tuning the step size to achieve a stable result. As a consequence the average difference between the two sets of derivatives was approximately 4 per cent which is greater than in the first example and probably due to inaccuracy of the finite difference estimates. In this case the AD generated derivative code took 2.28 s to evaluate all 24 derivatives compared to 3.03 s with finite derivatives (all calculations performed on a P4 Linux desktop workstation with 1Gb RAM). We also calculated gradients 'element by element', where the derivative code was executed once for each input parameter. This gave the same numerical values to within machine precision and took 3.67 s. The difference in time reflects the repeated execution of overhead in the element by element case.

For this problem we were unsuccessful in getting a result in reverse mode. Although the AD tool successfully generated derivative code, upon execution it failed to complete. It seems likely that this was due to some difficulty in efficiently reversing the flow of the synthetic receiver function algorithm. As noted above this can be a major problem. We did not investigate the cause in any great detail, nor employ any of the advanced features of this particular AD tool which allows users to insert directives in the code to assist in differentiation. Our results, therefore, reflect the experience of a 'novice' or 'hands off' user of AD, who, as noted above, is the intended target of the idealized AD tool. Nevertheless, for the receiver function problem forward mode has stably and efficiently generated derivatives in an automated fashion.

3.3 Seismic ray tracing

Our final test problem is the most numerically intensive of all. The experimental setup is shown in Fig. 7. Here we apply the AD tool to calculate the derivatives of seismic traveltimes with respect to seismic velocities of a laterally varying medium. This is a common

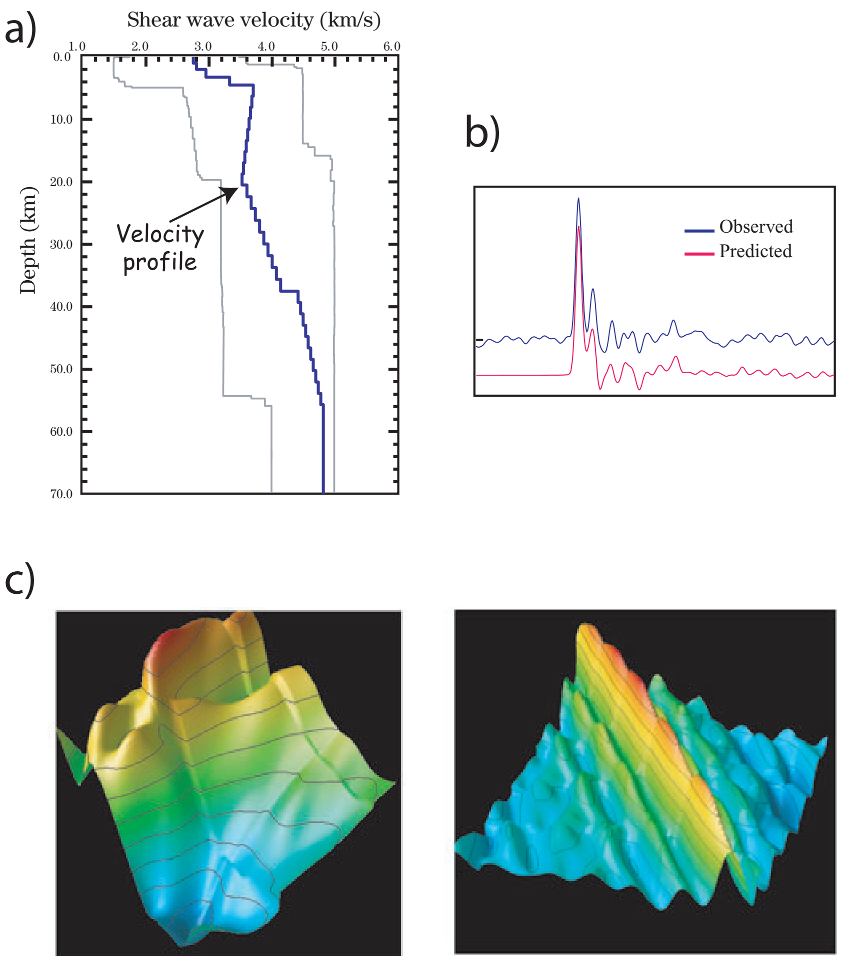


Figure 6. (a) Parametrization of shear wave velocity model used in the receiver function problem. Here, 24 parameters control the profile as a function of depth using linear segments between variable thickness layers; (b) an 'observed' and predicted receiver function. The squared difference between the two traces corresponds to the data misfit function used in the inversion and (c) two examples of the data misfit surface defined by the difference in receiver functions, plotted as a function of two pairs of model parameters. The irregular (non-quadratic) shape is due to the high non-linearity in the misfit surface. In the second example AD generated code is used to calculate derivatives of this misfit surface at any point.

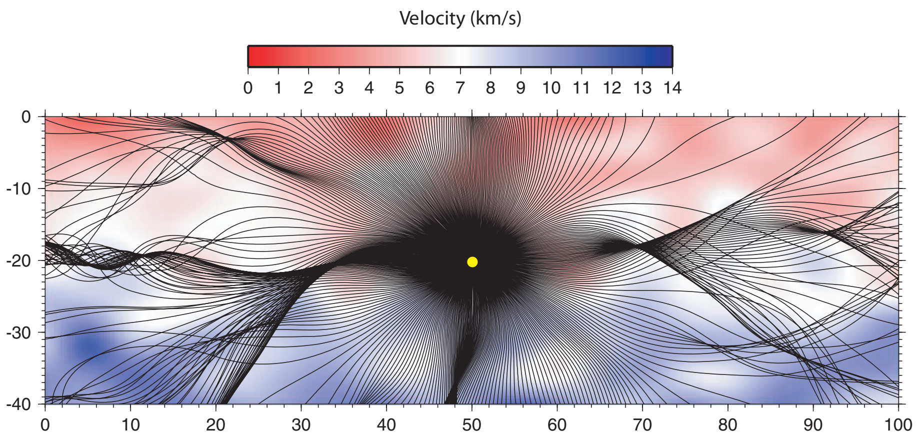


Figure 7. Seismic rays produced by the ray tracer in highly heterogeneous 2-D medium. Here rays are initially shot out at equal angles and their trajectory depends on the local wave speed variations. The source point is denoted by a yellow dot. In this example AD generated code is used to calculate derivatives of traveltime with respect to the parameters controlling the seismic velocity field.

problem in seismic tomography with body wave arrival times (see Rawlinson & Sambridge 2003).

Here we use a simple spline mesh to represent a 2-D laterally varying Gaussian random field of seismic velocities, and 400 rays are shot out at equal angles from a source placed in the centre of the model (see Fig. 7). As in the previous example no analytical solution exists for finding the path of the ray in a medium of this complexity. For each ray a set of non-linear initial value ODE's are solved using a Runge-Kutta algorithm. The result is the endpoint of the ray as a function of traveltime (see Rawlinson & Sambridge 2003, for details). All rays are stopped when they meet the edge of the 2-D medium.

As the spatial scale length of the heterogeneity decreases and the amplitude increases the rays become highly distorted by the velocity gradients in the model. In this experiment the amplitude of heterogeneity varies by more than a factor of 10 (which is extreme for the real Earth) and rays soon become chaotic. The spline mesh has 299 coefficients controlling the velocity model giving a Jacobian with 18 338 non-zero entries in a matrix with 119 600 elements. We used the AD tool to generate derivative code in forward mode. (Note here the number of output variables is 400 which outnumber the input variables, 299, and hence forward mode is to be preferred.) Again to make results comparable to previous examples we also calculated derivatives with a simple finite difference code (after some tuning of step size). Note that, in this case, derivatives could also be determined by integrating the basis function of each spline in the velocity mesh along each ray which would require some significant hand coding.

The entire Jacobian was determined by both approaches. The average difference in estimated derivatives was less than 10^{-5} per cent, hence the accuracy of the AD generated code is effectively at machine precision (as theory predicts). The compute time for the entire Jacobian with finite differences was 2502.6 s. The AD generated code in an element by element mode took 829.2 s and in an 'all at once' mode took 108.9 s. Hence in this case the forward mode is highly accurate and more than an order of magnitude faster than the corresponding finite difference algorithm. As with the previous example we were unable to get reverse mode to work.

4 CONCLUSION

We have presented results of several experiments with a particular AD tool, TAF of Giering & Kaminski (2003). In all cases efficient and accurate derivative code has been generated using forward mode, and in the simplest example reverse mode was equally successful. The accuracy and computation cost of forward mode was consistent with theoretical predictions. We observed an independence of the reverse mode CPU time on the number of input parameters. This is an unusual result, given that the cost of the source code in this case scales linearly with the number of input parameters. (We can only speculate that there is an underlying linear dependence which has not reached above the noise level in our experiments.)

Our experiments were conducted in a deliberately 'hands off' manner, as our intention was to test whether such tools were truly 'Automatic' in this sense. It is clear that AD tools in forward mode have reached a level of sophistication which makes them reliably automatic, accurate and stable. In reverse mode this is not yet the case, although we were impressed that it worked as well as it did. This is quite unsurprising given the difficulty involved in automatically reversing the flow of complex source codes. AD is an active field of research and one can expect AD tools to continue to develop. The one used here (TAF) is probably the most sophisticated in the world to date. (During the course of this work we actually tested several other AD tools, including ADIFOR2.0, TAMC and TAPENADE. See www.autodiff.org for full details of these packages.) At present AD is not yet fully automatic in reverse mode, as is acknowledged by experts in the field (see Griewank 1989; Corliss *et al.* 2002). It seems likely that for the foreseeable future human intervention will be required to get the most out of AD in reverse mode, in much the same way that directives are used to optimize the performance of most modern compilers. We expect a careful re-examination of the cases where reverse mode failed, together with appropriate use of AD directives would most likely improve performance.

Current directions in AD research include ways of incorporating parallel libraries such as MPI, extending the range of programming languages and also allowing a combination of languages such as Fortran90 and C (see Corliss *et al.* 2002, for discussions). At the same time the number of applications are steadily growing. It seems inevitable that over the next 10 yr AD will become an indispensable part of numerical modelling and optimization software, used across the physical sciences. We hope the present paper encourages further applications in the geosciences.

ACKNOWLEDGMENTS

We thank Drs Thomas, Kaminski and Ralf Giering of FastOpt the creators of TAMC and TAF, for their advice and assistance, and also in providing us access to their software.

REFERENCES

Ammon, C.J., Randall, G.E. & Zandt, G., 1990. On the nonuniqueness of receiver function inversions, *J. Geophys. Res.,* **95,** 15 303–15 318.

Aster, R., Borchers, B. & Thurber, C.H., 2005. Parameter estimation and inverse problems, in *International Geophysics Series,* **Vol. 90,** Elsevier, Amsterdam.

Beda, L.M., Korolev, L.N., Sukkikh, N.V. & Frolova, T.S., 1959. Programs for automatic differentiation for the machine BESM, Technical Report, Institute for Precise Mechanics and Computation Techniques, Academy of Science, Moscow, USSR (In Russian).

Bunge, H.-P., Hagelberg, C.R. & Travis, B.J., 2003. Mantle circulation models with variational data assimilation: inferring past mantle flow and structure from plate motion histories and seismic tomography, *Geophys. J. Int.,* **152,** 280–301.

Cacuci, D.G., 1981a. Sensitivity theory for nonlinear systems. I. Nonlinear functional analysis approach, *J. Math. Phys.,* **22,** 2794–2802.

Cacuci, D.G., 1981b. Sensitivity theory for nonlinear systems. II. Extension to additional classes of responses, *J. Math. Phys.,* **22,** 2803–2812.

Corliss, G., Faure, C., Griewank, A., Hascoët, L. & Naumann, U., 2002. *Automatic Differentiation of Algorithms: From Simulation to Optimization*, Springer, New York.

Cusdin, P. & Müller, J.-D., 2003. Automatic Differetiation: Learning to Speak AD, Qub-sae-03-05, Queen's Unversity Belfast.

De Capitani, C. & Brown, T.H., 1987. The computation of chemical equilibrium in complex systems containing non-ideal solutions, *Geochim. Cosmochim. Acta,* **51,** 2639–2652.

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

Giering, R. & Kaminski, T., 2002. Recomputations in reverse mode AD, in *Automatic Differentiation of Algorithms: From Simulation to Optimization*, Chap. 33, pp. 283–291, eds Corliss, G., Faure, C., Griewank, A., Hascoët, L. & Naumann, U., Springer, New York, NY.

Giering, R. & Kaminski, T., 2003. Applying TAF to generate efficient derivative code of Fortran 77–95 programs, *Proc. Appl. Math. Mech*.

Griewank, A., 1989. On automatic differentiation, in *Mathematical Programming: Recent Developments and Applications*, pp. 83–108, eds Iri, M. & Tanabe, K., Kluwer Academic Publishers, Dordecht.

Griewank, A., 1991. The chain rule revisited in scientific computing, *SIAM News,* **24**(No. 3), p. 20 &(No. 4) p. 8.

Griewank, A., 2000. Evaluating derivatives: principles and techniques of algorithmic differentiation, in *Frontiers in Appl. Math.*, No. 19, SIAM, Philadelphia, PA.

Harvie, C.E., Greenberg, J.P. & Weare, J.H., 1987. A chemical equilibrium algorithm for highly non-ideal multiphase systems: Free energy minimization, *Geochim. Cosmochim. Acta,* **51,** 1045–1057.

Haskell, N.A., 1953. The dispersion of surface waves in multilayered media, *Bull. Seism. Soc. Am.,* **43,** 17–34.

Iaffaldano, G., Bunge, H.-P. & Bücker, M., 2007. Mountain belt growth inferred from past plate convergence: a new tectonic inverse problem, *Earth planet. Sci. Lett.,* submitted.

Kaminski, T., Giering, R., Scholze, M., Rayner, P.J. & Knorr, W., 2003. An example of an automatic differentiation-based modelling system, in *Computational Science and its Applications—ICCSA 2003, Proceedings of the International Conference on Computer Science and its Applications*, Montreal, Canada, May 18–21, 2003. Part - II, Lecture Notes in Computer Science, pp. 95–104, Springer, Berlin.

Komatitsch, D., Tsuboi, S. & Tromp, J., 2005. The spectral-element method in seismology, in *Seismic Earth: Array Analysis of Broadband Seismograms*, pp. 49–65, eds Lavender, A. & Nolet, G., AGU.

Langston, C.A., 1989. Scattering of teleseismic body waves under Pasadena, California, *J. Geophys. Res.,* **94,** 1935–1951.

Moresi, L., Gurnis, M. & Zhong, S.J., 2000. Plate tectonics and convection in the earth's mantle: toward a numerical simulation, *Comput. Sci. Eng.,* **2,** 22–33.

Moresi, L., Dufor, F. & Mulhlaus, H.B., 2003. Lagrangian integration point finite element method for large deformation modeling of viscoelastic geomaterials, *J. Comp. Phys.,* **184,** 476–497.

Rall, L.B., 1981. *Automatic Differentiation: Techniques and Applications*, Vol. 120 of Lecture Notes in Computer Science, Springer-Verlag, Berlin.

Rall, L.B. & Corliss, G.F., 1996. An introduction to automatic differentiation, in *Computational Differentiation: Techniques, Applications, and Tools*, pp. 1–17, eds Berz, M., Bischof, C.H., Corliss, G.F. & Griewank, A., SIAM, Philadelphia, PA.

Rath, V., Wolf, A. & Bücker, H., 2006. Joint three-dimensional inversion of coupled groundwater flow and heat transfer based on automatic differentiation: Sensitivity calculation, verification, and synthetic examples, *Geophys. J. Int.,* **167,** 453–466, doi:10.1111/j.1365–246X.2006.03074.x.

Rawlinson, N. & Sambridge, M., 2003. Seismic traveltime tomography of the crust and lithosphere, *Geophys. J. Int.,* **46,** 81–197.

Sambridge, M. & Mosegaard, K., 2002. Monte Carlo methods in geophysical inverse problems, *Rev. of Geophys.,* **40,** 3.1–3.29, doi:10.1029/2000RG000089.

Shibutani, T., Sambridge, M. & Kennett, B.L.N., 1996. Genetic algorithm inversion for receiver functions with application to crust and uppermost mantle structure beneath eastern australia, *Geophys. Res. Lett.,* **23,** 1829–1832.

Sieminski, A., Liu, Q., Trampert, J. & Tromp, J., 2007. Finite-frequency sensitivity of surface waves to anisotropy based on adjoint methods, *Geophys. J. Int.,* **16**(3), 1153–1174.

Sommacal, S., 2004. Computational petrology: Subsolidus equilibria in the upper mantle, *PhD thesis*, The Australian National University.

Storey, S.H. & van Zeggeren, F., 1964. Computation of chemical equilibrium compositions, *Can. J. Chem. Eng.,* **42,** 54–55.

Talagrand, O., 1991. The use of adjoint equations in numerical modelling of the atmospheric circulation, in *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pp. 169–180, eds Griewank, A. & Corlis, G.F., SIAM, Philadelphia, PA.

Talagrand, O. & Courtier, P., 1987. Variational assimilation of meteorological observarions with the adjoint vorticity equation. I: theory, *Q. J. R. Meteorol. Soc.,* **113,** 1311–1328.

Tarantola, A., 1984. Inversion of seismic reflection data in the acoustic approximation, *Geophysics,* **49,** 1259–1266.

Tarantola, A., 1988. Theoretical background for the inversion of seismic waveforms, including elasticity and attenuation, *Pure Appl. Geophys.,* **128,** 365–399.

Tarantola, A., 2005. *Inverse Problem Theory and Methods for Model Parameter Estimation*, SIAM, Philadelphia.

Thacker, W. & Long, R.B., 1988. Fitting dynamics to data, *J. Geophys. Res.,* **93,** 1227–1240.

Thomson, W.T., 1950. Transmission of elastic waves through a stratified solid, *J. Appl. Phys.,* **21,** 89–93.

Tromp, J., Tape, C. & Liu, Q., 2005. Seismic tomography, adjoint methods, time reversal, and banana–donut kernels, *Geophys. J. Int.,* **160,** 195–216, doi:10.1111/j.1365–246X.2004.02456.x.

Wengert, R., 1964. A simple automatic derivative evaluation program, *Commun. ACM,* **7**(8), 463–464.