Optimization Methods and Software
Vol. 00, No. 00, November 2010, 1–31

RESEARCH ARTICLE

Second order adjoints for solving PDE-constrained optimization problems

Alexandru Cioaca*, **Mihai Alexe*** and **Adrian Sandu***

(*Received 00 Month 200x; in final form 00 Month 200x*)

Inverse problems are of utmost importance in many fields of science and engineering. In the variational approach inverse problems are formulated as PDE-constrained optimization problems, where the optimal estimate of the uncertain parameters is the minimizer of a certain cost functional subject to the constraints posed by the model equations. The numerical solution of such optimization problems requires the computation of derivatives of the model output with respect to model parameters. The first order derivatives of a cost functional (defined on the model output) with respect to a large number of model parameters can be calculated efficiently through first order adjoint sensitivity analysis. Second order adjoint models give second derivative information in the form of matrix-vector products between the Hessian of the cost functional and user defined vectors. Traditionally, the construction of second order derivatives for large scale models has been considered too costly. Consequently, data assimilation applications employ optimization algorithms that use only first order derivative information, like nonlinear conjugate gradients and quasi-Newton methods.

In this paper we discuss the mathematical foundations of second order adjoint sensitivity analysis and show that it provides an efficient approach to obtain Hessian-vector products. We study the benefits of using of second order information in the numerical optimization process for data assimilation applications. The numerical studies are performed in a twin experiment setting with a two-dimensional shallow water model. Different scenarios are considered with different discretization approaches, observation sets, and noise levels. Optimization algorithms that employ second order derivatives are tested against widely used methods that require only first order derivatives. Conclusions are drawn regarding the potential benefits and the limitations of using high-order information in large scale data assimilation problems.

Keywords: Inverse Problems, Second Order Adjoints, Stiff Equations, Sensitivity Analysis, Shallow Water Equations, Data Assimilation, Numerical Optimization, PDE-constrained Optimization

1. Introduction

In the variational approach inverse problems are formulated as minimization problems, where the control variables are the model parameters to estimate, and the cost functions are defined on the model output. The minimization is constrained by the model equations which relate model outputs to model parameters. A family of inverse problems of particular interest in this paper is data assimilation. Data assimilation seeks to adjust the model initial, boundary, and parameter values, such that the mismatch between the model predictions and observations of reality is minimized in a least squares sense. The underlying models represent discretized systems of partial differential equations; the variational approach to data assimilation requires the solution of large "PDE-constrained" optimization problems.

An efficient numerical solution to the large scale optimization problem requires the derivatives of the cost function with respect to the model parameters. The traditional first order adjoint models (FOA) compute the gradient of the cost function, i.e., its first order derivatives with respect to model parameters. Second order adjoint models (SOA) provide second derivative information in the form of products between the Hessian of the cost function and a user defined vector.

ISSN: 1055-6788 print/ISSN 1029-4937 online
© 2010 Taylor & Francis
DOI: 10.1080/1055678xxxxxxxxxxxxx
http://www.informaworld.com

Second order adjoints have been used in data assimilation for numerical weather prediction within the numerical optimization algorithms by Wang et al., [1, 2], Le Dimet et al. [3], Ozyurt et al., [4], Daescu and Navon [5]. Other applications of second order adjoints are discussed in Raffard and Tomlin [6], Charpentier et al. [7], Le Dimet et al. [3], and Alekseev and Navon [8]. Ozyurt and Barton [4] have discussed the evaluation of second order adjoints for embedded functions of stiff systems.

Most applications to date have focused on continuous second order adjoints (obtained by linearizing the underlying ordinary or partial differential equation models) [1, 3, 4]. Discrete second order adjoints (obtained by linearizing the numerical approximations of the model) have been obtained by automatic differentiation [7, 9, 10].

Sandu *et. al.* [11] have developed a rigorous approach to deriving discrete second order adjoints for Runge Kutta and Rosenbrock methods, and have applied them to chemical transport models. Alexe *et. al.* [**?**] used second order adjoints for the shallow water equations in sensitivity analysis and uncertainty quantification. Alekseev *et. al.* compared the performance of different minimization algorithms in the solution of inverse problems [13]. Exploring this previous work constitutes the motivation behind the following research results.

In this paper we study the modalities of using second order adjoint information in a popular large-scale optimization problem from the field of computational fluid dynamics called "variational data assimilation". We consider the two dimensional shallow water system in Cartesian coordinates and build two numerical models to solve them in space-time. For each model we implement first order and second order adjoints and use them in several data assimilation scenarios to provide derivative information to various minimization algorithms. By comparing the performance and accuracy of the algorithms making use only of FOA against those who can also employ SOA, we are trying to assess the benefits and the drawbacks of the latter ones.

The paper is organized as follows. In Section 2 we review the derivation of continuous and discrete second order adjoint equations in the context of ordinary differential equations (ODEs). Section 3 introduces the reader to the minimization algorithms used in this study. Section 4 describes the two different implementations for solving the shallow water equations and their associated adjoint models and presents the optimization problem posed in the context of data assimilation. In Section 5 we comment on the numerical results obtained for each minimization algorithm when solving the optimization problem for both implementations. Section 6 summarizes the conclusions of our work and future directions of interest.

2. Second Order Adjoints

2.1. *Continuous SOA*

Consider a general nonlinear dynamical system described by the following ODE

$$\frac{dU}{dt} = f(t,U) \;, \qquad U\left(t^0\right) = U^0 \;, \qquad t^0 \le t \le t^{\mathrm{F}} \;. \tag{1}$$

Equation (1) defines the *forward model.* It can also represent a PDE after discretization in space in the method of lines framework. The solution is $U(t) \in \mathbb{R}^n$, and the model parameters are the initial conditions U^0. Without loss of generality any model parameters can be transformed into variables by appending additional formal evolution equations for parameters.

Consider a cost functional that depends on the solution of the forward model; without loss of generality it is defined as a function of the state at the final time

$$\mathcal{J}\Big(U^0\Big) = \Psi\Big(U\left(t^{\mathrm{F}}\right)\Big) \ . \tag{2}$$

Any cost function that depends on the solution along the entire trajectory can be brought to the form (2) by introducing additional "quadrature variables" [11]. In this paper we assume that the functions f and Ψ are at least twice continuously differentiable.

We are interested to efficiently evaluate the first and second order sensitivities of the cost function (2) with respect to changes in initial conditions

$$\frac{\partial \mathcal{J}}{\partial U_i^0} \quad \text{and} \quad \frac{\partial^2 \mathcal{J}}{\partial U_i^0 \partial U_j^0} \ , \quad 1 \le i,j \le n \ .$$

Throughout this paper vectors will be represented in column format and an upper script $(\cdot)^T$ will denote the transposition operator. The gradient of a scalar function (e.g., $\partial \mathcal{J}/\partial U^0$) is a row vector. We denote the Jacobian of the time derivative function in (1) by

$$f'_{i,j}(t,U) = \frac{\partial f_i\left(t,U\right)}{\partial U_j} \ , \qquad 1 \le i,j \le n \ . \tag{3}$$

The Hessian of the time derivative function in (1) is a 3-tensor of second order derivatives

$$f''_{i,j,k}\left(t,U\right) = \frac{\partial^2 f_i\left(t,U\right)}{\partial U_j \, \partial U_k} \ , \quad 1 \le i,j,k \le n \ . \tag{4}$$

The Hessian allows to express the derivatives of the Jacobian times a user vector. For any vectors v and w we have that

$$\frac{\partial}{\partial U}\left[f'(t,U)\cdot w\right]\cdot v = \Big(f''(t,U)\cdot v\Big)\cdot w \ ,$$
$$\frac{\partial}{\partial U}\left[f'(t,U)^T\cdot w\right]\cdot v = \Big(f''(t,U)\cdot v\Big)^T\cdot w \ ,$$

where the dot operator $(\cdot)$ denotes the regular tensor-vector product.

Small perturbations of the solution (due to infinitesimally small changes δU^0 in the initial conditions) propagate forward in time according to the *tangent linear model* $\delta U' = f'(t,U)\cdot\delta U$. Since the evolution of perturbations depends on the forward solution via the argument of f', the forward (1) and the tangent linear model (5) are evolved together forward in time:

$$\frac{d}{dt}\begin{bmatrix} U \\ \delta U \end{bmatrix} = \begin{bmatrix} f(t,U) \\ f'(t,U)\cdot\delta U \end{bmatrix} \ , \quad t^0 \le t \le t^{\mathrm{F}} \ ,$$
$$\begin{bmatrix} U \\ \delta U \end{bmatrix}\left(t^0\right) = \begin{bmatrix} U^0 \\ \delta U^0 \end{bmatrix} \ . \tag{5}$$

The change in the cost function (2) due to the small change δU^0 in the initial conditions is

$$\delta\mathcal{J} = \frac{\partial \Psi}{\partial U}\left(t^{\mathrm{F}}\right) \cdot \delta U\left(t^{\mathrm{F}}\right) = \frac{\partial \mathcal{J}}{\partial U^0} \cdot \delta U^0 \ .$$

In the forward sensitivity analysis each integration of the tangent linear model (5) allows to compute the dot product of the gradient $\partial\mathcal{J}/\partial U^0$ with the vector of initial perturbations δU^0. The gradient is recovered after n tangent linear model (5) integrations initialized with linearly independent perturbation vectors.

A more efficient way of calculating the gradient $\partial\mathcal{J}/\partial U^0$ is provided by the *first order adjoint model* [14–16]

$$\begin{aligned} \frac{d\lambda}{dt} &= -f'(t,U)^T \cdot \lambda \ , \quad t^{\mathrm{F}} \geq t \geq t^0 \ , \\ \lambda\left(t^{\mathrm{F}}\right) &= \left(\frac{\partial \Psi}{\partial U}\right)^T \left(U(t^{\mathrm{F}})\right) \ . \end{aligned} \tag{6}$$

The adjoint variables $\lambda(t) \in \mathbb{R}^n$ represent the sensitivities of the cost function with respect to (changes in) the model solution

$$\lambda(t) = \left(\frac{\partial \mathcal{J}}{\partial U(t)}\right)^T \ , \tag{7}$$

and in particular we have that the adjoint variable at the initial time is the transposed gradient of the cost function with respect to initial conditions.

We are now interested in obtaining the second order derivatives of the cost function with respect to initial conditions [4]. The Hessian of $\mathcal{J}$ reads

$$\left(\nabla^2 \mathcal{J}\right)_{i,j} = \frac{\partial^2 \mathcal{J}}{\partial U_i^0 \partial U_j^0} = \frac{\partial \lambda_i(t^0)}{\partial U_j^0} \ , \qquad 1 \leq i,j \leq n \ . \tag{8}$$

The Hessian has n^2 elements. In most engineering problems n is very large and computing the entire Hessian is not practical. We will therefore look to compute Hessian times vector products $\sigma = \nabla^2\mathcal{J} \cdot u$ for any user-defined vector u

$$\sigma_i = \left(\nabla^2 \mathcal{J} \cdot u\right)_i = \frac{\partial \lambda_i(t^0)}{\partial U^0} \cdot u \ , \tag{9}$$

or, in slightly greater generality, we consider initial perturbations $\delta U^0 = u$ and look to compute the *second order adjoint variables*

$$\sigma(t) = \frac{\partial \lambda(t)}{\partial U^0} \cdot \delta U^0 = \frac{\partial \lambda(t)}{\partial U(t)} \cdot \delta U(t) \ .$$

The second order adjoint variables are small perturbations of the first order adjoint variables resulting from changes δU^0 in the initial forward conditions U^0. Consequently, the time evolution of second order adjoints is described by the tangent linear model of the first order adjoint equation (6). Like with (5) the first and second order adjoint variables have to be solved for together, backward in time:

$$\frac{d}{dt}\begin{bmatrix}\lambda\\ \sigma\end{bmatrix} = \begin{bmatrix} -f'(t,U)^T\cdot\lambda \\ -f'(t,U)^T\cdot\sigma - \Big(f''(t,U)\cdot\delta U\Big)^T\cdot\lambda\end{bmatrix}$$

$$\begin{bmatrix}\lambda\\ \sigma\end{bmatrix}(t^{\mathrm{F}}) = \begin{bmatrix} (d\Psi/dy)^T\, U(t^{\mathrm{F}}) \\ \left(d^2\Psi/dU^2\right)\left(U(t^{\mathrm{F}})\right)\cdot\delta U\left(t^{\mathrm{F}}\right)\big)\end{bmatrix}\,,\quad t^{\mathrm{F}}\ge t\ge t^0\,.$$

The first equation in (10) is the *second order adjoint ordinary differential equation* and defines the time evolution of the *second order adjoint variable* σ.

2.2. *Discrete SOA*

Similar considerations hold for discrete systems, following [11]

$$\widehat{U}^{k+1} = \mathcal{N}_k\left(\widehat{U}^k\right)\,,\quad k=0,\cdots,N-1\,,\quad \widehat{U}^0\ \text{ given}\,, \tag{10}$$

and the discrete cost function of the form

$$\widehat{\mathcal{J}}\left(\widehat{U}^0\right) = \widehat{\Psi}\left(\widehat{U}^N\right)\,. \tag{11}$$

The discrete system (10) represents a numerical discretization of (1) with a one-step numerical method. The cost function (11) is defined on the numerical solution, and approximates the continuous cost function (2) defined on the exact solution.

We denote the Jacobian matrix of the discrete time-marching operator by $\mathcal{N}_k'(\widehat{U}) = \partial\mathcal{N}_k/\partial\widehat{U}$, and the Hessian three-tensor by $\mathcal{N}_k''(\widehat{U}) = \partial^2\mathcal{N}_k/\partial\widehat{U}^2$. The tangent linear model of (10) is

$$\delta\widehat{U}^{k+1} = \mathcal{N}_k'\left(\widehat{U}^k\right)\cdot\delta\widehat{U}^k\,,\quad 0\le k\le N-1\,,\quad \delta\widehat{U}^0 = u\,. \tag{12}$$

The first and second order discrete adjoint equations are [11]

$$\begin{bmatrix}\widehat{\lambda}^k\\ \\ \widehat{\sigma}^k\end{bmatrix} = \begin{bmatrix} -\left(\mathcal{N}_k'(\widehat{U}^k)\right)^T\cdot\widehat{\lambda}^{k+1} \\ -\left(\mathcal{N}_k'(\widehat{U}^k)\right)^T\cdot\widehat{\sigma}^{k+1} - \left(\mathcal{N}_k''(\widehat{U}^k)\cdot\delta\widehat{U}^k\right)^T\cdot\widehat{\lambda}^{k+1}\end{bmatrix} \tag{13}$$

$$\begin{bmatrix}\widehat{\lambda}^N\\ \widehat{\sigma}^N\end{bmatrix} = \begin{bmatrix}(\partial\widehat{\Psi}/\partial\widehat{U})^T(\widehat{U}^N) \\ (\partial^2\widehat{\Psi}/\partial\widehat{U}^2)\left(\widehat{U}^N\right)\cdot\delta\widehat{U}^N\end{bmatrix}\,,\qquad N-1\ge k\ge 0\,.$$

At the end of the backward in time integration (13) provides the gradient and the Hessian vector product

$$\widehat{\lambda}^0 = \left(\frac{\partial\widehat{\mathcal{J}}}{\partial\widehat{U}^0}\right)^T\,,\qquad \widehat{\sigma}^0 = \frac{\partial^2\widehat{\mathcal{J}}}{\left(\partial\widehat{U}^0\right)^2}\cdot u\,.$$

3. Optimization Algorithms

Two types of methods are popular for solving large-scale, unconstrained optimization problems: Newton methods and nonlinear conjugate gradients. Both classes use derivative information such as the gradient and/or Hessian of the function to be minimized. The particular methods used in this study are summarized in Table 1 along with the class to which they belong, their use of second order derivatives and the name of the particular software implementation used here. The abbreviation "CG" stands for "Nonlinear Conjugate Gradients".

Table 1. Optimization Algorithms

Method	Class	Second-Order Derivative	Software Available
L-BFGS	quasi-Newton	NO	YES (L-BFGS-B)
Truncated Newton	Newton + CG	YES	YES (TNPACK)
Moser-Hald	Newton	YES	NO
Hybrid	quasi-Newton + CG	NO	YES (HYBRID)
Nonlinear CG	CG	NO	YES (CG+)
CG Descent	CG	NO	YES (CGDESCENT)
Daniel CG	CG	YES	NO

The value of the cost function (described in detail in the next section, for now we will refer to it as $\mathcal{J}$) is computed on the solution obtained by running the forward model in time. The gradient of the cost function ($\nabla\mathcal{J}$) is computed from the FOA, presented in the previous section under the notation λ. The explicit Hessian ($\nabla^2 J$) is not available but we can compute its action on a vector by using the SOA (σ).

The software packages have been optimized over the course of years by the authors and community members to improve performance and robustness. The methods that are not available as maintained software packages were implemented by the authors of this paper following the same principles as the curated versions.

Global convergence is ensured by a linesearch procedure. The software libraries for nonlinear conjugate gradients and all the Newton methods presented below use the linesearch procedure from MINPACK [17] with the exception of CG Descent which uses the linesearch described in [18]. Daniel-CG and Moser-Hald use a linesearch procedure implemented by the authors around the same procedure from CG Descent.

A presentation of each of these methods follows.

3.1. *Newton methods*

Newton methods look for the minimizer U^* of the given cost function $\mathcal{J}(U)$ by updating $U_{[k]}$ in an iterative fashion (index between brackets indicates iteration number) based on the formula (14a) where $p_{[k]}$ is the search direction obtained from the Newton iteration (14b).

$$U_{[k+1]} = U_{[k]} + p_{[k]} \ , \tag{14a}$$

$$p_{[k]} = B_{[k]} \cdot \nabla\mathcal{J}(U_{[k]}) \ . \tag{14b}$$

In the classic version of this algorithm, the matrix $B_{[k]}$ is computed as the inverse of the Hessian of the function to be minimized evaluated at the current iterate (15a). Since the cost of computing the Hessian and inverting it is very high, many methods of approximating this operation are available. Also, in finite precision calculations it might not always be positive definite which makes the search direction $p_{[k]}$ no longer a descent direction. One workaround to the cost associated with $B_{[k]}$ is to compute the matrix only for the initial solution $U_{[0]}$ and reuse it throughout all subsequent iterations (15b). This approach is known as "simplified Newton".

$$B_{[k]}^{\text{classic}} = -\left(\nabla^2 \mathcal{J}(U_{[k]})\right)^{-1} \ , \tag{15a}$$

$$B_{[k]}^{\text{simplified}} = -\left(\nabla^2 \mathcal{J}(U_{[0]})\right)^{-1} \ . \tag{15b}$$

The following methods take a more sophisticated approach towards approximating B.

3.1.1. L-BFGS-B

The BFGS method is considered to be the "gold standard" algorithm for variational data assimilation. It is described in great detail in [19, 20] where it is compared against other quasi-Newton methods.

BFGS generates an approximation of the second order information through an update formula that uses data from the model output and its gradient at each iteration and also preserves symmetry and positive definiteness of this approximate Hessian. The idea was originally proposed by Davidon in [21] and improved by Fletcher in [22].

"Limited memory" BFGS (L-BFGS) is a version of BFGS that is suited for large-scale problems whose Hessian matrix cannot be computed or stored at a reasonable cost [23]. The idea is to use only information generated by the model during the most recent iterations. Information from earlier iterations is discarded as it is considered to be less relevant to the current behavior of the Hessian. Despite this approximation the method yields a linear rate of convergence. Many studies have shown that an optimal value for the number of iterations stored in the memory belongs in the range [3, 7] and increasing it beyond 7 usually does not improve the performance of the algorithm.

The L-BFGS-B implementation of Nocedal, Zhu et al (introduced in [24]) has been used throughout the following tests. This is a standard optimization package used in industry and academia for variational data assimilation. The letter "B" stands for "bounded" as this particular implementation is able to handle bound constraints on the iterates.

The number of recent iterations stored in memory (variable 'm' in L-BFGS-B) was set to 7. The parameter 'factr' controls the desired accuracy of the iterative algorithm and it was set to 10^2 (corresponding to high accuracy).

3.1.2. Truncated Newton - TNPACK

Truncated Newton (TN) or inexact Newton methods is another class of methods derived from Newton-Raphson focusing on generating the search direction by solving the linear system (14b) in $p_{[k]}$ with $B_{[k]}^{-1}$ as the system matrix. An iterative linear solver can be employed for this task that only needs Hessian-vector products for the evaluation of the left-hand side. The particular version of TN used for the following tests is also known as Newton-CG and it consists of series of inner iterations to solve the linear system for the search direction using conjugate gradients and outer iterations that use this search direction along with a linesearch algorithm to update the solution. Because CG is designed to solve positive definite systems and the Hessian may have negative eigenvalues when not close to a solution, the CG iteration terminates as soon as a direction of negative curvature is generated. This guarantees that the search direction is a descent direction and convergence rate is quadratic.

The code used for this algorithm is TNPACK of Schlick, Fogelson and Xie [25, 26].

It is a robust and flexible implementation. Most of the parameters are customizable. The user can select among various factorization methods and termination criteria. Another important feature is the choice between computing the Hessian using finite-differences or second order adjoints. Tests have been performed with both methods (referred to as TNPACK-FD for Hessian approximated through finite differences of the gradient and TNPACK-SOA for Hessian-vector product computed using second-order adjoints). Some parameters specific to this software package were changed from the default values to improve convergence. The maximum number of PCG iterations in each Newton iteration was set to 50 and the maximum number of function calls inside the line search routine was set to 20.

3.1.3. Hybrid

The Hybrid method [27, 28] (HYBR) alternates m iterations of BFGS with k iterations of TN). The inverse of the approximate Hessian is passed between these two algorithms: TN uses it in the CG inner iterations as a preconditioner for the linear system whose solution is the search direction and L-BFGS uses it as an initial approximation for the inverse Hessian. This way the information gathered by one method is used by the other. It is hoped that the advantages of the two methods complement each other's shortcomings. TN needs few iterations to reach the solution but it is computationally intensive and the second order derivative information gathered is lost from one outer iteration to the other. L-BFGS needs many iterations that are inexpensive but the algorithm to update the Hessian is less accurate than TN and it can perform poorly on ill-conditioned problems.

In the following tests, the implementation by Morales and Nocedal [27] was used. The parameters used for the BFGS module were assigned the same values as specified above for L-BFGS-B. Similary, the parameters of the truncated Newton part have been aligned to those used for TNPACK. The best results were obtained when alternating 15 iterations of BFGS with 5 iterations of TN.

3.1.4. Moser-Hald

This algorithm is a Newton-type inversion free method proposed by Jurgen Moser [29] and improved by Ole Hald [30]. The method is quadratically convergent and sharp error estimates were found by Potra and Ptak [31]. Starting from a given initial condition $U_{[0]}$ and an approximation $G_{[0]}$ of the inverse of the Hessian (usually the identity matrix) the iterations of the algorithm go as follows:

$$U_{[k+1]} = U_{[k]} - G_{[k]} \cdot \nabla \mathcal{J}(U_{[k]}) \tag{16a}$$

$$G_{[k+1]} = \left(2I - G_{[k]} \cdot \nabla^2 \mathcal{J}(U_{[k+1]})\right) \cdot G_{[k]} \tag{16b}$$

Implementing this algorithm using second order adjoints is not straightforward as it makes use of the full Hessian which is usually not available. A workaround around this issue is to multiply the recursion formula (16b) with $\nabla \mathcal{J}(U_{[k]})$ in order to obtain the expression of a Hessian-vector product (17a) which allows us to employ SOA models. Expanding similarly the expression for the second step to aim for obtaining Hessian-vector products wherever we have a Hessian increases their number to three (equivalent with three SOA runs). We can continue recursively with subsequent iterations and generalize that we need exactly $2k + 1$ SOA runs for iteration k.

We implemented a reduced version of Moser-Hald consisting of the following steps inside each iteration:

(i) First MH step - steepest descent

$$U_{[k+1]} = U_{[k]} + \alpha_{[k]} \cdot \nabla \mathcal{J}(U_{[k]}) \tag{17a}$$

(ii) Second MH step requiring one Hessian vector product

$$\begin{aligned} U_{[k+2]} = U_{[k+1]} - \alpha_{[k+1]} \cdot \Big(& 2G_{[k]} \cdot \nabla \mathcal{J}(U_{[k+1]}) + \\ & G_{[k]} \cdot \nabla^2 \mathcal{J}(U_{[k+1]}) \cdot G_{[k]} \cdot \nabla \mathcal{J}(U_{[k+1]}) \Big) \end{aligned} \tag{17b}$$

(iii) Third MH step requiring three Hessian vector products
(iv) Fourth MH step requiring seven Hessian vector products
(v) Restart

Future work for this algorithm must consist in finding a way to store the information about the Hessian from one cycle to another and increasing the robustness of the code. The linesearch routine used for each step was implemented according to Hager and Zhang [18]. Although the information accumulated during the three steps is lost at the end of each cycle, the results are very promising for regular unconstrained optimization problems.

3.2. *Nonlinear Conjugate Gradients*

The nonlinear conjugate gradients algorithm is an extension of the well-known linear conjugate gradient method of solving linear systems of equations [32]. Fletcher and Reeves first proposed the nonlinear version in [33] and it consists in two important changes to the linear case:

- The residual r is replaced by the gradient of the objective function
- The next iterate is obtained by performing a linesearch along the search direction generated by the conjugate gradient formulas.

If the cost function is a strongly convex quadratic and $\alpha_{[k]}$ the exact minimizer then the algorithm reduces to the linear version. Each iteration requires only one evaluation of the objective function and its gradient and no matrix operations are involved.

Each flavor of CG+ is different in the way it generates the search direction. This is accomplished through the formula to compute the scalar value $\beta_{[k]}$.

$$p_{[k+1]} = -\nabla \mathcal{J}(U_{[k]}) + \beta_{[k+1]} \cdot p_{[k]} \tag{18a}$$

The Fletcher-Reeves formula is:

$$\beta^{FR}_{[k+1]} = \frac{\nabla \mathcal{J}(U_{[k+1]})^T \cdot \nabla \mathcal{J}(U_{[k+1]})}{\nabla \mathcal{J}(U_{[k]})^T \cdot \nabla \mathcal{J}(U_{[k]})}\ , \tag{18b}$$

while the Polak-Ribiere formula is:

$$\beta^{PR}_{[k+1]} = \frac{\nabla \mathcal{J}(U_{[k+1]})^T \cdot \big(\nabla \mathcal{J}(U_{[k+1]}) - \nabla \mathcal{J}(U_{[k]})\big)}{\big\|\nabla \mathcal{J}(U_{[k]})\big\|^2} \tag{18c}$$

These are only two of the most popular examples but the literature contains many other versions developed over the years.

3.2.1. Classic Nonlinear Conjugate Gradients

The particular implementation of this method used for our studies is CG+, described in [34]. The bests results were obtained when choosing $\beta_{[k]}$ to be computed with the "Polak-Ribiere positive" method (18c).

3.2.2. CG Descent

CG Descent (conjugate gradient with guaranteed descent) is a nonlinear conjugate gradient algorithm recently developed by Hager and Zhang. The theory and the code (CGDESCENT, referred to as CGDES in this paper) are presented in [35]. The general discussion for nonlinear conjugate gradient holds for this algorithm too. Its main feature represents the way $\beta_{[k]}$ is computed in order to make sure the search directions are descent directions.

This new scheme addresses limitations in the previous conjugate gradient schemes as the search direction is always a descent direction so the algorithm improves the solution with each iteration. This is an improvement over other nonlinear conjugate gradient methods which sometimes get stuck at a particular solution and are not able to advance further, a phenomenon known as "jamming"

Most of the inner parameters of the algorithm are customizable making it one of the most flexible non-commercial optimization software. Since the parameters can influence the behavior of the method, a program searching for the parameter-space of a given problem was also made available (see [35]). For this particular model the default values of the parameters were used because no improvement was found when running the optimization with modified values.

3.2.3. Daniel CG

This method is another version of the nonlinear conjugate gradient. It was developed by J.W. Daniel, [36–38]. What makes Daniel CG distinct from other nonlinear CG methods is the use of second order information in the computation of each search direction. The $\beta_{[k]}$ parameter proposed by Daniel is:

$$\beta^D_{[k]} = \frac{\nabla \mathcal{J}(U_{[k+1]})^T \cdot \nabla^2 \mathcal{J}(U_{[k]}) \cdot p_{[k]}}{p^T_{[k]} \nabla^2 \mathcal{J}(U_{[k]}) \cdot p_{[k]}} \tag{19}$$

This method has been considered for a long time to be impractical as it required second order information. However since automatic differentiation made possible the computation of the (discrete) second order adjoints which can provide Hessian-vector products this became a viable method. Its only obvious drawback is the extra time needed to run the second order adjoint.

The code for this algorithm has been developed by the authors of the paper and it uses a linesearch routine heavily inspired by the one developed by Hager and Zhang for their CGDES package and presented in their paper [18]. During the tests 'jamming' was frequently noticed: Daniel CG sometimes enters an infinite cycle when using the strong Wolfe conditions for linesearch. This has been avoided by using approximate Wolfe. We will refer to our implementation as CGDAN in this paper.

4. Applications

4.1. ***The Problem Setting***

We consider the two dimensional Saint-Venant PDE system that approximates fluid flow inside a shallow basin (also known as "Shallow Water Equations" and introduced in [39]):

$$
\begin{aligned}
\frac{\partial}{\partial t}h + \frac{\partial}{\partial x}(uh) + \frac{\partial}{\partial y}(vh) &= 0 \\
\frac{\partial}{\partial t}(uh) + \frac{\partial}{\partial x}(u^2h + \frac{1}{2}gh^2) + \frac{\partial}{\partial y}(uvh) &= 0 \\
\frac{\partial}{\partial t}(vh) + \frac{\partial}{\partial x}(uvh) + \frac{\partial}{\partial y}(v^2h + \frac{1}{2}gh^2) &= 0\ .
\end{aligned}
$$

The spatial domain is square shaped ($\Omega = [-3,\, 3]^2$), and the integration window is $t^0 = 0 \le t \le t^F = 0.1$. Here $h(t,x,y)$ denotes the fluid layer thickness, and $u(t,x,y)$, $v(t,x,y)$ are the components of the velocity field. g is the standard value of the gravitational acceleration. The boundary conditions are periodic in both directions. For ease of presentation, we arrange the n discretized state variables in a column vector

$$
U = \begin{bmatrix} \widehat{h} \\ \widehat{uh} \\ \widehat{vh} \end{bmatrix} \in \mathbb{R}^n\ . \tag{20}
$$

We implemented two numerical models to solve these equations, presented below.

4.2. *Explicit timestepping model*

In the first version of the model we use a finite volume-type scheme for space discretization and a fourth-order Runge-Kutta scheme for timestepping. This method was introduced by Liska and Wendroff in [39].

The tangent linear model, first-order and second-order adjoints were generated through automatic differentiation of the forward model using TAMC [40, 41]. The operation is straight-forward as TAMC parses the source code that is implementing the numerical model and generates the derived code in the same programming language. For more details about automatic differentation see [42].

A question that arises naturally is how large is the overhead of the code generated through automatic differentation. Table 2 illustrates the CPU times of TLM, FOA and SOA models, normalized with respect to the CPU time of a single forward model run. It is seen that a full second order adjoint integration is about 3.5 times more expensive than a single first order adjoint run and the latter takes 3.7 times longer than the forward run. The time for SOA is a large value as it takes longer time to compute than approximating it though finite difference (2 FOA runs). We consider these to be large values but for the purpose of this test they still are computationally feasible. Please note that these values apply only for our particular implementation (method of lines) and the overhead can greatly vary with the complexity of the numerical method or the automatic differentiation tool used.

Because the overhead is already large and no code optimization was able to reduce it significantly, we decided to run the model and its adjoints in conjunction. For example each time the tangent linear model or first-order adjoint are run, the forward model is run beforehand. Similarly, in order to compute the second order adjoint one must first run the forward, first-order adjoint and tangent linear models. Hence the actual timing values are the ones displayed in the right hand side of the table. In automatic differentiation this is done naturally in order to create

Table 2. Wall-clock CPU times for explicit timestepping model, normalized with respect to a single forward model run

FWD	1		
TLM	2.5	FWD + TLM	3.5
FOA	3.7	FWD + FOA	4.7
SOA	12.8	FWD + TLM + FOA + SOA	20

checkpoints for the solution of the forward model. However, when the differences between the CPU time of the model and its adjoints are smaller, an implementation that avoids re-running the FWD model can be more effective.

4.3. *Implicit timestepping model*

The second version of the model puts an emphasis on performance and better CPU time ratios between the model and its adjoint. Discretization in space is accomplished by a third order upwind scheme on both axis (see [43]). For timestepping we use the Crank-Nicolson scheme in [44] and solve the nonlinear equation with Newton-Raphson's method. For this we derive offline the explicit Jacobian of the space discretization. Since its structure is sparse, the Jacobian is factorized with the sparse solvers library UMFPACK [45–48]. The computation and factorization of the Jacobian are the most time-consuming parts of the forward model; we decided to use a simplified version of Newton's method over five steps, where we would compute and factorize the Jacobian at the first step and the subsequent four would reuse this factorization.

From the derived expressions of the TLM (21b), FOA (21c) and SOA (21d) we can see that the factorization of $I - 0.5\Delta t \cdot f'$ obtained at each timestep can be reused in all these models. Also, the right hand side expression contains the actual Jacobian in a matrix-vector product so if we save it on disk during the forward model run then we can reload and reuse it at the corresponding timestep in the derived models. Tests have shown that using binary format for storing the Jacobian is faster than recomputing it. As for the SOA, it contains two extra terms and the symbol H denotes the Hessian of the space discretization. Since in each case we are confronted with a Hessian-vector-vector product we use automatic differentation on the space discretization twice to generate its second order adjoint and then just "plug in" the procedure generated by TAMC in our model. In equations for the forward model (21a), TLM (21b), FOA (21c) and SOA (21d), we use f for the discretization in space, f' for its Jacobian and f'' for the Hessian.

$$U^{n+1} - \frac{\Delta t}{2} f(U^{n+1}) = U^n + \frac{\Delta t}{2} f(U^n) \ , \tag{21a}$$

$$\left(I - \frac{\Delta t}{2} f'(U^{n+1})\right) \cdot \delta U^{n+1} = \left(I + \frac{\Delta t}{2} f'(U^n)\right) \cdot \delta U^n \ , \tag{21b}$$

$$\left(I - \frac{\Delta t}{2} f'(U^{n+1})^T\right) \cdot \lambda^n = \left(I + \frac{\Delta t}{2} f'(U^n)^T\right) \cdot \lambda^{n+1} \ , \tag{21c}$$

Table 3. Wall-clock CPU times for implicit timestepping model, normalized with respect to a single forward model run

FWD	1		
TLM	0.1		
FOA	0.1		
SOA	0.15	SOA+TLM	0.25

$$
\begin{aligned}
\left(I - \frac{\Delta t}{2} f'(U^{n+1})^T\right) \cdot \sigma^n = & \left(I + \frac{\Delta t}{2} f'(U^n)^T\right) \cdot \sigma^{n+1} \\
& + \frac{\Delta t}{2} \left(f''(U^n) \cdot \delta U^n\right)^T \cdot \lambda^{n+1} \\
& + \frac{\Delta t}{2} \left(f''(U^{n+1}) \cdot \delta U^{n+1}\right)^T \cdot \lambda^n \ .
\end{aligned} \tag{21d}
$$

The timing for each model in this implementation is presented in Table 3. The ratios are very different from what we obtained for explicit timestepping as all derived models need only about 10-15% of the forward model CPU time to execute. In contrast with the explicit timestepping case, this time we decided to improve the performance of the model even further by disentangling the execution of the forward, tangent-linear and adjoint models. This way, we can run the forward model only when necessary while the adjoints can be run independently by using the checkpoints of the appropriate previous forward model run. For example, one can run the forward model and store its timeseries and then run the TLM or FOA for the same solution as many times as desired without having to rerun the model. However, since the SOA depends on the TLM, these two are still ran in conjunction so the actual CPU time needed to obtain the Hessian-vector product is the sum of these two (Table 3) but they reuse the stored trajectories of the FWD and FOA. This approach needs more storage space and careful bookkeeping in both models and optimization software libraries but is definitely feasible and efficient.

Because of the nature of the numerical scheme chosen for timestepping, this implementation is more expensive per timestep iteration than the previous one (explicit timestepping) by a factor of 10^3. Almost 20% of the CPU time is spent computing the Jacobian of the space discretization and more than 60% factorizing it. However, implicit timestepping schemes have the advantages of being able to use large timesteps and to solve stiff differential equations. In our tests we used the same value for the time step in order to align the simulation scenarios as much as possible and we only present normalized CPU times in our results.

We can say that if the explicit timestepping model was a worse-case scenario because of the large overhead in the adjoint models, then the implicit timestepping could be a favorable-case scenario as the numerical method allowed several performance improvement tricks such as reusing factorizations or intermediate data. The lesson to learn from this implementation is that through the usage of a combination of favorable schemes and software tools we can come up with an implementation that does not necessarily have to obey the paradigm that derived models take longer to run and are unfeasible for computation.

We note that another case in which small overheads can be obtained is usually employed in large-scale models whose adjoints are very hard to derive on the whole through automatic or symbolic differentiation and finite differences would not be feasible as well. For these models, the adjoints are usually obtained from a simplified version of the forward model that can capture some relevant behavior of the processes. A good example is WRF (Weather Research & Forecast), a complex climate model implementing dynamics, physics and chemistry of the atmosphere, but whose tangent linear and adjoint models were generated through automatic differentiation applied only on the dynamical core [49]. Repeating the same process one can generate a reduced second order adjoint which can prove to be helpful in optimization problems, sensitivity analysis, uncertainty quantification, etc. However, this particular approach is not the scope of our study.

4.4. *Data Assimilation Problem*

Data assimilation is the process by which measurements are used to constrain model predictions. The information from measurements can be used to obtain better initial and boundary conditions. Variational data assimilation allows the optimal combination of three sources of information: a priori (background) estimate of the state of the system, knowledge of the interrelationships among the variables of the model and observations of some of the state variables. The optimal initial state U^0 (also called the analysis) is obtained by minimizing the function

$$\begin{aligned}\mathcal{J}(U^0) &= \frac{1}{2}(U^0 - U^b)^T \cdot B^{-1} \cdot (U^0 - U^b) \\ &+ \frac{1}{2}\sum_{k=1}^{K} \left(\mathcal{H}^k(U^k) - z^k\right)^T \cdot R_k^{-1} \cdot \left(\mathcal{H}^k(U^k) - z^k\right) \qquad (22)\end{aligned}$$

The first term of the sum quantifies the departure from the background state at the initial time t_0, while the second term measures the distance to the observations, which are taken at times t^k inside the assimilation window $[t_0, t_F]$. The block-diagonal background error covariance matrix B is built to take into account of the spatial correlations of the variables, as well as the periodic boundary conditions. $\mathcal{H}^k$ is the observation operator defined at assimilation time t^k. It maps the discrete model state $U^k \approx U(t^k)$ to the observation space. R_k is the observations error covariance matrix.

The efficient numerical minimization of (22) requires the gradient of the cost function as well as second order derivative information when available. 4D-Var usually relies on adjoint sensitivity analysis to provide information about the first and second order derivatives of the objective function (see, e.g., [50, 51]). Variational data assimilation is a perfect example of nonlinear numerical optimization where the the minimization of the cost function is constrained by a numerical model associated with a set of PDEs, hence PDE-constrained optimization.

The main focus of this study is to determine whether the 4D-Var data assimilation process can be improved by using a minimization algorithm that employs second order derivatives to minimize the 4D-Var cost function. A comparison of the performance of each minimization algorithm on our particular models can provide valuable guidelines even for situations when a second-order adjoint is not available, but a first-order adjoint is. Our tests employ the optimization algorithms described in the section above for the minimization of the 4D-Var fit function in the context of data assimilation.

As seen in the description of the implementation of our explicit timestepping model, second order adjoints might take significantly longer to run than the forward model or its first-order adjoint but even in this case they could either improve the rate of convergence or help provide a more accurate solution. It is also a comprehensive test for the popular optimization algorithms (L-BFGS-B, CG+ or HYBR) against algorithms that were considered in the past to be less effective, mainly because of the expensive computations involved.

5. Numerical Results and Discussion

In the context of this paper, 4D-Var will be used to assess the potential benefits of second order adjoints in variational data assimilation and compare several minimization algorithms for this heavy nonlinear problem. The algorithms presented above will be used to minimize the cost function (22) and generate the analysis U^a. The scenario is presented below:

- The 2-D grid is divided into 40 grid points on each direction (1600 in total). The number of time steps for the model runs is selected to be $N = 50$ for both explicit and implicit timestepping models.
- The reference solution is generated for the height component h as a Gaussian pulse of amplitude $A = 30$ while the velocity fields u and v are assigned the value 2 at each grid point.
- The model is ran with the reference as initial condition in order to generate the synthetic observations. The observation frequency is set to once every 5 time steps.
- The observation error covariance R is a diagonal matrix (based on the assumption that the observational errors are uncorrelated). The standard deviation of these errors was set to 2% of the value of the observations.
- The background solution U^b is generated by applying a correlated perturbation on the reference profile for h. At each grid point, u and v are assigned the values 2.2 and respectively 1.8.
- Two versions of the background error covariance B were generated for a standard deviation of 8%. The first is a diagonal matrix corresponding to the case when the errors for each grid point are independent between them. This is obviously a simplification of the real case because neighboring grid points can affect each other and errors might propagate on localized areas. This real-case scenario was tested by using a second version of B with a nondiagonal structure and the correlation distance of 5 grid points. This will help the 4D-Var method to spread the information from observations in each cell based on information passed from its neighbors.

An important aspect in the usage of optimization algorithms is fine-tuning parameters such as convergence and tolerance constants, maximum number of model runs inside a specific context, etc. Several parameters were chosen to be common for all methods and assigned the following values:

- The stopping criterion is set to $\left\|\nabla \mathcal{J}(U^0_{[k]})\right\| < 10^{-6} \cdot \max(1, \left\|U^0_{[k]}\right\|)$.
- Wolfe conditions parameters are set to $c_1 = 10^{-4}$ and $c_2 = 0.9$ for Newton methods and $c_2 = 0.1$ for nonlinear conjugate gradients.
- A maximum number of 200 iterations is allowed for each optimization. The exception is TNPACK which was set to run for a maximum of 100 outer iterations.

When comparing the results obtained with each optimization method two important issues will be followed: how close does the method converge to the reference (accuracy) and how fast is this convergence over time (performance). The main indicators used to analyze the accuracy of the data assimilation process are the misfit between the optimized solution and the reference solution, and the cost function and gradient of the optimized solution. The misfit measure used is the root mean square error (RMSE) as in relation (23) where n stands for the dimension of the problem.

$$RMSE = \frac{\left\| U^{ref} - U^{a} \right\|}{\sqrt{n}} \tag{23}$$

It is important to remember that in 4D-Var data assimilation the effect of decreasing the RMSE between the analysis and the reference is a side-effect of minimizing the cost function. This means that there are two meanings for the relative performance of each algorithm: how much it minimizes the cost function, providing statistics of interest to the field of nonlinear optimization and how much the RMSE is reduced, an aspect of interest from a data assimilation point of view. Please remember that we generate a reference solution just so we have something to test the analysis against. In normal data assimilation operations one knows only the background and observations and the reduction in the misfit towards the actual state of the atmosphere might or might not have been reduced at the same scale that the 4D-Var was reduced.

Note that we can start from the same initial condition but obtain different analysis states for the 4D-Var cost function based on the number and type of observations used in assimilation and the error covariance matrices B and R. The reduction in the cost function achieved by the tested methods should be interpreted relative to each other only under the same scenario. The background field (initial condition) is the same for all experiments, though, so the median reduction in RMSE of a particular scenario might suggest a more favorable setting for data assimilation.

Three scenarios are presented for both models:

- Perfect dense observations, B diagonal (the "easy" scenario, useful to emphasize the general performance of each algorithm);
- Perturbed sparse observations, B diagonal (the least-available-information scenario);
- Perturbed sparse observations, B nondiagonal (the computationally demanding scenario but also the closest to an operational setting)

These scenarios have been selected from a larger set based on different combinations among these features and variations of parameters such as error covariances, total number of model timesteps or the frequency at which observations are used. The scenarios not presented here provided redundant conclusions.

The first scenario computes the cost function using observations for each point of the grid for all three variables (h, u and v). In real-case scenarios observations are usually not available for each point of the grid but it is useful to assess this ideal case. Also, this scenario can be equivalent to the methodology that interpolates sparse observations in space, which is becoming a popular practice in data assimilation tools such as GSI or WRFVAR. Synthetic observations were generated from a reference run of the forward model and are not perturbed for this scenario which should theoretically be a less challenging optimization problem.

For the second and third scenarios, synthetic observations are taken at every fourth point of the grid in both directions and perturbed for all three variables. Thus only 6.25% of the observations used in the dense case are available. This is consistent with the real-case setting where observations always come perturbed because the instruments used by meteorological stations or satellites are affected by inherent inaccuracies. The difference between these scenarios comes in the structure of the background error covariance matrix. The former uses a diagonal B and the latter a nondiagonal B. The advantage of using nondiagonal B consists in the fact that throughout the process of minimizing the cost function, the information can be spread out among correlated neighboring cells.

For each model and for each scenario we are presenting two plots on loglog scale, one of them for the relative reduction of the cost function and the other for the relative reduction in the RMSE (in time, normalized by one forward model run) and a table containing the value of the relative reduction in cost function, norm of its gradient and RMSE for the analysis obtained through each method and the optimization statistics: number of iterations, model runs and CPU time (again, normalized).

No minimization algorithm used any form of preconditioning. Also, BFGS and HYBR were initialized through a cold-start. The number of the inner iterations in truncated Newton was limited to 15 as larger values would not have been feasible for using second order adjoints for the first model and smaller values would not factorize the Hessian accurately enough.

5.1. ***Results for the explicit timestepping model***

5.1.1. Scenario 1

The cost function and RMSE plots (Figures 1, 2) show that each method behaves as predicted since the value of the 4D-Var cost function is decreased with each iteration.

CG+ and CGDES evolve head-to-head and from the iteration markers we can infer that CGDAN follows a similar trajectory but the use of second-order adjoints lags it behind the two other nonlinear conjugate gradient methods. The most accurate solution is provided by CG+. L-BFGS and HYBR generate similar iterates, especially in the short-run where the latter performs L-BFGS iterations. The fact that HYBR converges slower once it switches to iterations of truncated Newton is quite surprising and does not confirm the theoretical superiority of this method over pure L-BFGS. All these five methods converge to solutions of similar cost function values. Compared to them, the two versions of TNPACK are more accurate and minimize the cost function to a smaller value. Finally, Moser-Hald looks competitive for the initial iterations but restarting the algorithm every four iterations slows the convergence gradually until it reaches 200 iterations. The algorithm does not benefit from a robust implementation so this makes the authors hope its convergence properties could be exploited in future work. Unfortunately this is the only scenario where it worked so a good implementation should not only treat the issue of restarts through which the accumulated second order information is lost but also to be able to treat more difficult minimization scenarios.

The evolution of the RMSE (Figure 2) generally is correlated with that of the cost function as the rate at which the algorithms converge is the same. Two unusual aspects have to be underlined. Although we would expect the RMSE trajectory to monotonously decrease just like the cost function, this is not the case for all methods. L-BFGS, CGDAN and both truncated Newton decrease the RMSE along with the cost function down to a point and then the RMSE starts increasing although the cost function keeps decreasing. This may or may not be have implications on the data assimilation procedure (we will see later that it is application-specific) but it is difficult to detect in a real-case scenario because the reference will not be available to watch the evolution of the RMSE. In the course of the minimization process we would expect the analysis to be closest to the real state of the system as it is the minimizer of our fit function but we can definitely see this may not be the case. Moreover, except for the convergence criteria for the cost function, there is no other way of stopping the iterations when the overall lowest value of the RMSE has been attained. We can see from the RMSE plots that if it not were for this switch in monotony, TNPACK would have generated the solution with the smallest RMSE with a reduction of about 51% in both FD and SOA. The other unusual aspect is related but of lesser importance: the trajectories of L-BFGS and HYBR are wiggly in some regions so the RMSE keeps increasing and decreasing on small intervals but the overall tendency is to decrease.

Table 4 confirms numerically the conclusion drawn from the two figures regarding the reduction in the initial cost function and RMSE. The reduction in the gradient norm is also of interest because it shows the fact that truncated Newton, through its composite structure of outer and inner iterations, is closer to the actual minimum of the 4D-Var function. For the short time window (about 50 model runs) the most effective methods at reducing the RMSE are L-BFGS, HYBR and TNFD. This is a very unfavorable case for the algorithms using SOA since they lose from the start time in the equivalent of 20 model runs for executing only one SOA.

From the second part of Table 4 we can see that the algorithms which make use of the fewest model runs are CG+ and L-BFGS with a ratio of about 1.7 respectively 1.5 FWD and FWD+FOA runs for each iteration. However, TNFD also keeps up in time with these two algorithms as the decrease in the outer iterations is more significant than the decrease attained by CG+ and L-BFGS on each iteration due to the superior accuracy of the factorization computed inside the inner iterations. We also have to consider the fact that truncated Newton uses many inner iterations for the last outer iterations (usually the limit) but the decrease is not that significant anymore.

We can infer from this scenario that truncated Newton is more accurate than the other methods at minimizing the 4D-Var cost function but that does not necessarily translate into a superior accuracy of providing a minimizer of the misfit towards the reference.

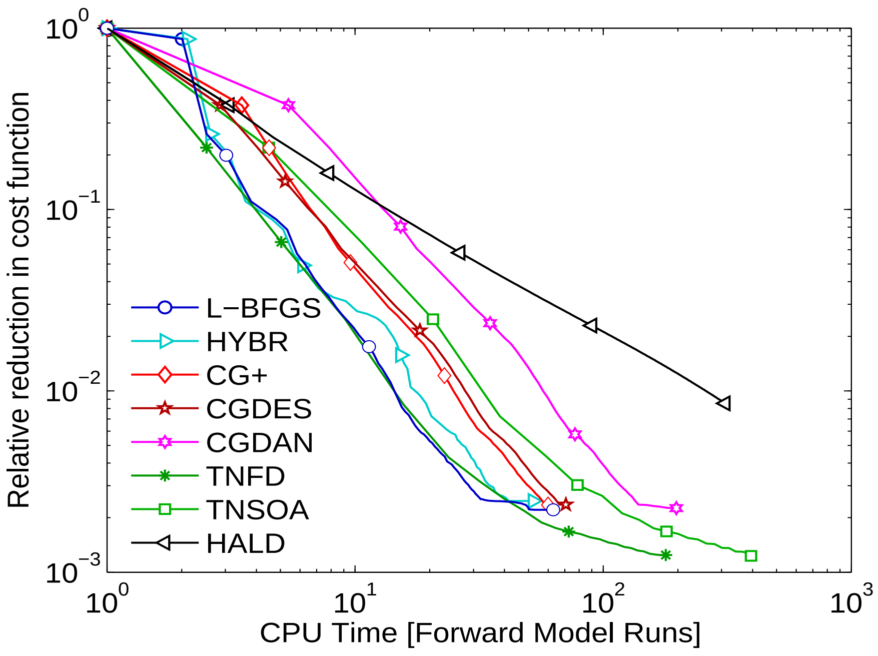

Figure 1. Cost function evolution with compute time for explicit timestepping model. Scenario 1: Perfect and dense observations, diagonal B

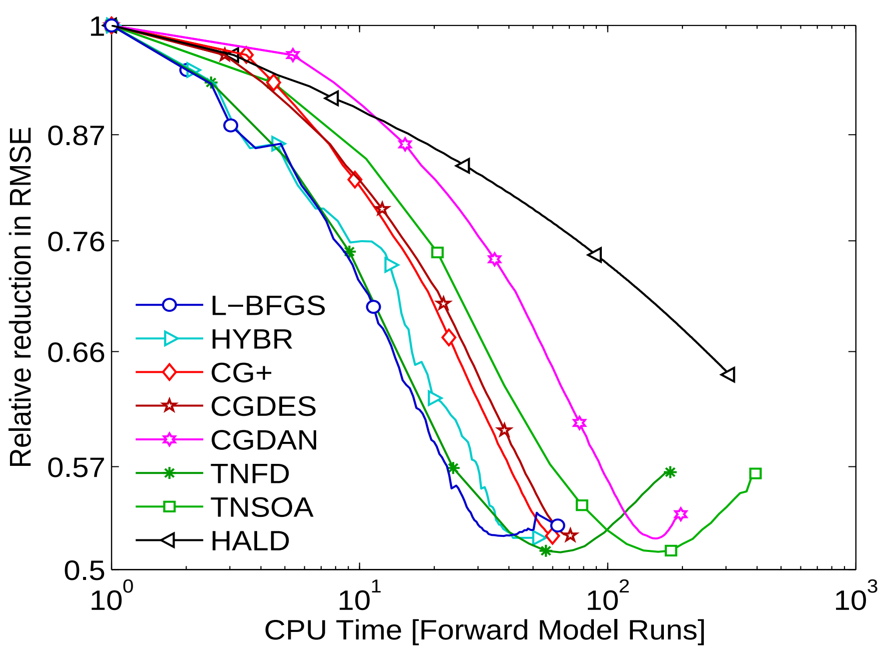

Figure 2. RMSE evolution with compute time for explicit timestepping model. Scenario 1: Perfect and dense observations, diagonal B

Table 4. Explicit timestepping model. Scenario 1: Perfect and dense observations, diagonal B

	L-BFGS	HYBR	CG+	CGDES	CGDAN	TNFD	TNSOA	HALD
RMSE	5.28e-1	5.21e-1	5.22e-1	5.21e-1	5.39e-1	5.66e-1	5.65e-1	6.4e-1
$\mathcal{J}(U^a)$	2.2e-3	2.4e-3	2.3e-3	2.3e-3	2.2e-3	1.24e-3	1.23e-3	8.4e-3
$\|\nabla\mathcal{J}(U^a)\|$	2.3e-3	6.6e-3	3.3e-3	3.7e-3	2.6e-3	5e-4	4e-4	1.54e-2
Iterations	84	63	57	57	76	25	23	200
FWD	123	102	131	113	141	427	137	400
FWD+FOA	123	102	131	170	217	427	137	600
FWD+FOA+ TLM+SOA	0	0	0	0	76	0	304	550
CPU Time	611	500	655	1044	2311	2166	4577	3100

5.1.2. Scenario 2

As described above, the second scenario uses sparse observations which reduce the space of the cost function. The background error covariance matrix is still diagonal so the algorithms face a more difficult problem, that of reconciling less observations that do not capture the entire state of the system with a background whose gridpoints contain errors that are strictly tied to each gridpoint and cannot be corrected through local correlations.

Figure 3 shows that the evolution of each cost function follows a global trend, just like in the previous scenario. However, the difference between the two truncated Newton and the rest is smaller. From Figure 4 we see that the evolution of the RMSE for each algorithm also behaves similarly to the previous scenario, including the unwanted increase in the misfit on the last iterations. The only algorithm who does not act in this manner is CG+. In the short-run TNFD, L-BFGS and HYBR are still the most well-performing algorithms.

The numbers in Table 5 confirm that this was a harder scenario for 4D-Var as the mean reduction in RMSE achieved by all algorithms is around 80% of the initial value, worse than the previous scenario's 40%.

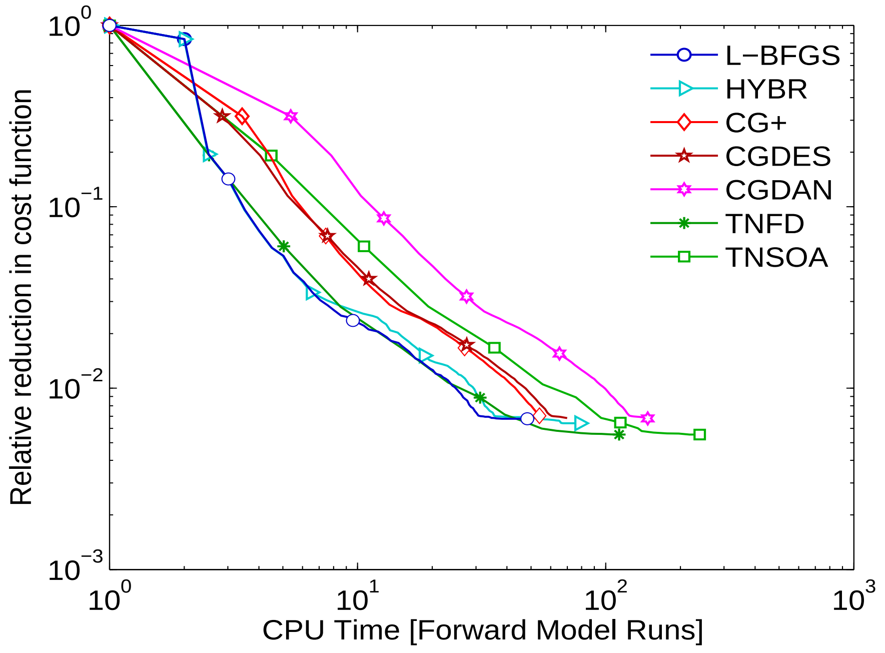

Figure 3. Cost function evolution with compute time for explicit timestepping model. Scenario 2: Perturbed and sparse observations, diagonal B

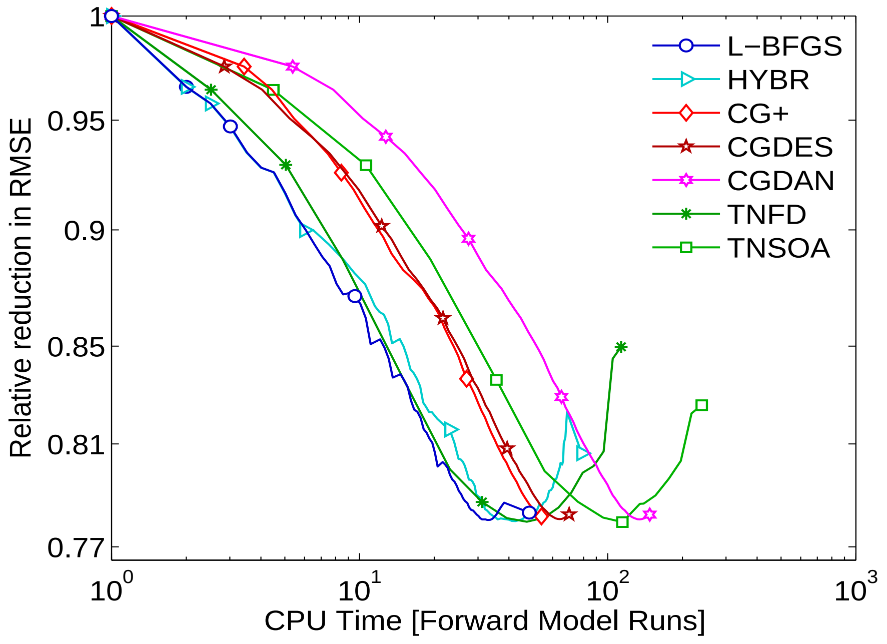

Figure 4. RMSE evolution with compute time for explicit timestepping model. Scenario 2: Perturbed and sparse observations, diagonal B

5.1.3. Scenario 3

The last scenario for this model makes use of a nondiagonal B with a correlation distance of 5 units. This means that we consider the error in each cell of the background field to arise not only from computational errors internal to it but also from propagating errors between neighbours, which is indeed an assumption closer to the way errors occur in real models.

Table 5. Explicit timestepping model. Scenario 2: Perturbed and sparse observations, diagonal B

	L-BFGS	HYBR	CG+	CGDES	CGDAN	TNFD	TNSOA
RMSE	7.8e-1	8.1e-1	7.8e-1	7.8e-1	7.8e-1	8.5e-1	8.2e-1
$\mathcal{J}(U^a)$	6.8e-3	6.4e-3	7e-3	6.8e-3	6.8e-3	5.6e-3	5.6e-3
$\|\nabla\mathcal{J}(U^a)\|$	3.1e-3	4.4e-3	4.3e-3	4.1e-3	2.7e-3	3.4e-3	2.3e-3
Iterations	69	91	51	59	59	17	16
FWD	95	155	116	114	113	324	107
FWD+FOA	95	155	116	173	172	324	107
FWD+FOA+ TLM+SOA	0	0	0	0	59	0	188
CPU Time	466	766	577	1055	1822	1611	2922

First thing to notice from the plots is that CGDES and CGDAN broke down at the second iteration. The problem persisted even after modifying the parameters of the optimization methods such as the Wolfe conditions but neither of them was able to pass the second iteration. The situation replicated even when using approximate Wolfe conditions.

Just like in the previous scenarios, the algorithms evolve in time tightly clustered together. The exception is TNFD which takes longer to converge. Looking at the evolution of the RMSE we can see that it is monotonous and it does not start increasing towards the final iterations. This behavior can be associated with the use of a nondiagonal B which gives more degrees of freedom for correcting the error.

Excluding the two algorithms that break, the mean reduction in RMSE is somewhere around 38% which is the best among all three scenarios. This means that there is more available information, also confirmed by the fact that all algorithms reached the maximum limit of iterations. Computationally, L-BFGS and HYBR are the least-intensive using about 1 FWD run and 1 FWD+FOA for each iteration.

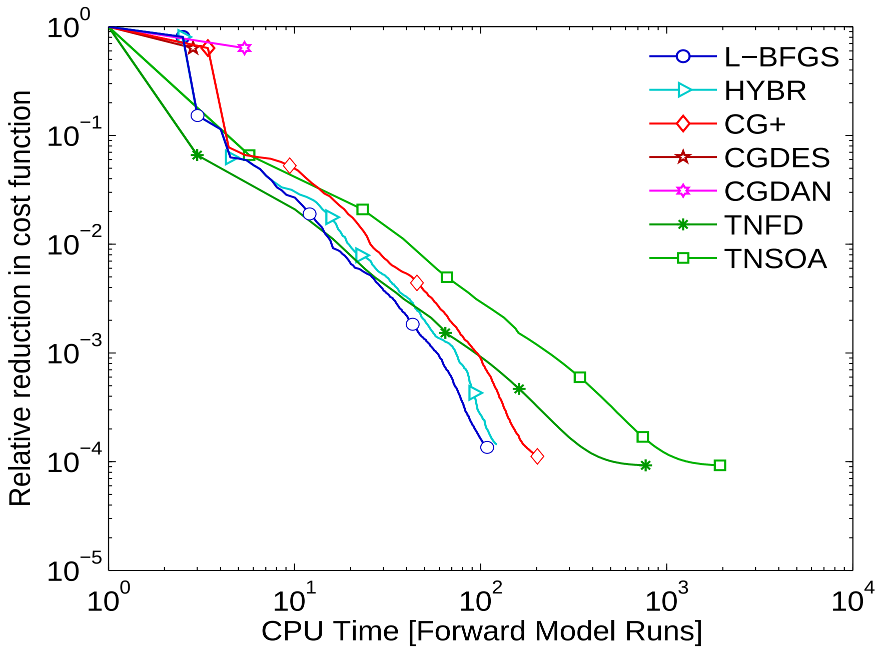

Figure 5. Cost function evolution with compute time for explicit timestepping model. Scenario 3: Perturbed and sparse observations, nondiagonal B

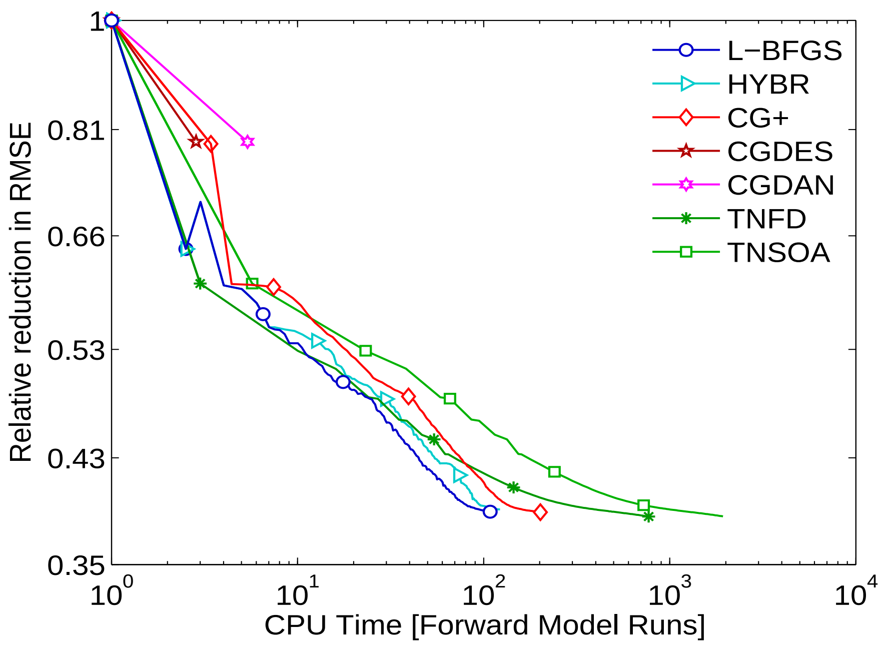

Figure 6. RMSE evolution with compute time for explicit timestepping model. Scenario 3: Perturbed and sparse observations, nondiagonal B

Table 6. Explicit timestepping model. Scenario 3: Perturbed and sparse observations, nondiagonal B

	L-BFGS	HYBR	CG+	CGDES	CGDAN	TNFD	TNSOA
RMSE	3.9e-1	3.9e-1	3.9e-1	7.9e-1	7.9e-1	3.8e-1	3.8e-1
$\mathcal{J}(U^a)$	1.3e-4	1.4e-4	1.1e-4	6.3e-1	6.3e-1	9e-5	9e-5
$\|\nabla\mathcal{J}(U^a)\|$	1.4e-2	5.2e-3	3.9e-3	-	-	8e-4	8e-4
Iterations	200	200	200	2	2	100	100
FWD	217	246	400	3	3	1536	200
FWD+FOA	217	246	400	3	3	1536	200
FWD+FOA+ TLM+SOA	0	0	0	0	2	0	1436
CPU Time	1078	1220	2012	22	33	7700	19310

5.2. ***Results for the implicit timestepping model***

5.2.1. Scenario 1

Because the TLM, FOA and SOA take about the same time to execute, the use of using second order derivative information in the optimization process does not bring a considerable overhead. Most of the computation is done in the forward model so this is a fair comparison between using first order and second order adjoints.

As predicted, the trajectories of the cost function (Figure 7) for each algorithm evolve closer together in time. L-BFGS, HYBR and TNSOA converge slightly faster than the conjugate gradients flavors. TNFD proves to be the slowest to converge since for the inner iterations the finite differences require that the forward model is ran each time. TNSOA is definitely faster, since its inner iterations need only the recomputation of the TLM and the SOA. Basically they switch roles from the explicit version where the CPU time for the SOA slow down significantly the latter. However, both converge again to the same value for the cost function and RMSE which means that they provided similar analyses disregarding the discretization scheme. One of them (TNFD) is suitable for use on models whose SOA is too costly or just unavailable but FWD and FOA are feasible and the other one for the case where the FWD and FOA are costly but the TLM and SOA can be ran many times. An interesting aspect for this implementation is the fact that the final value of the RMSE for all algorithms is the minimum on that particular trajectory. There is no increase in the RMSE like in the explicit timestepping model. Another difference from the results obtained with the explicit model for the same scenario setting is the fact that each algorithm undergoes more iterations and with the exception of TN the rest achieves the maximum limit of 200 iterations.

In the short-run TNSOA has a significant reduction in both cost function and RMSE, even for a window of CPU time equivalent to as few as 10 forward runs which means it is feasible even for operational use.

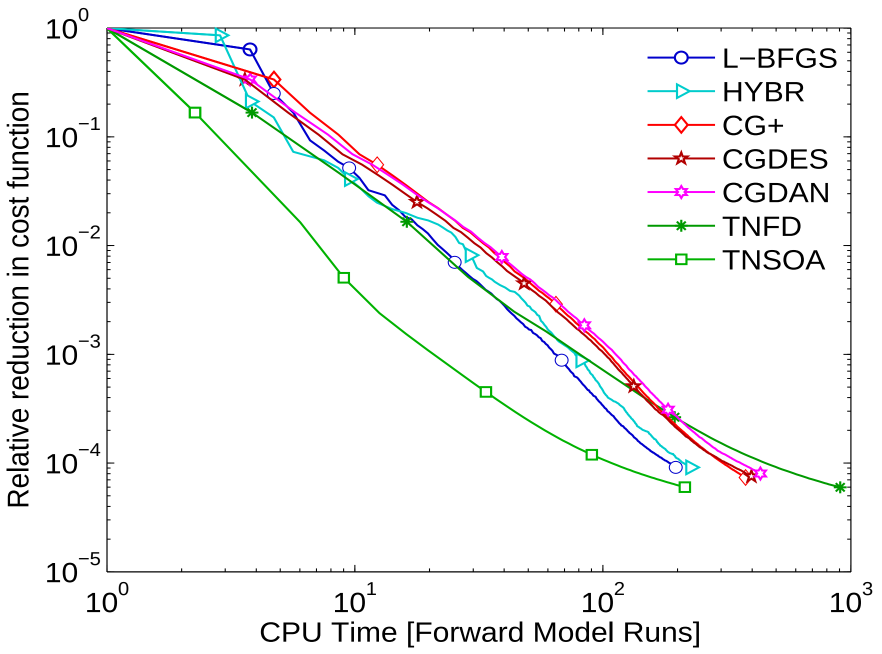

Figure 7. Cost function evolution with compute time for implicit timestepping model. Scenario 1: Perfect and dense observations, diagonal B

Table 7. Implicit timestepping model. Scenario 1: Perfect and dense observations, diagonal B

	L-BFGS	HYBR	CG+	CGDES	CGDAN	TNFD	TNSOA
RMSE	4e-1	4.1e-1	3.9e-1	3.9e-1	4e-1	3.7e-1	3.7e-1
$\mathcal{J}(U^a)$	9e-5	9.1e-5	7.3e-5	7.5e-5	7.9e-5	5.9e-5	6e-5
$\|\nabla\mathcal{J}(U^a)\|$	0.00072	0.00058	0.0005	0.00055	0.00119	0.00023	0.00024
Iterations	200	200	200	200	200	63	63
FWD	210	241	400	448	448	974	64
ADJ	210	241	400	200	200	974	64
SOA	0	0	0	0	200	0	910
CPU Time	200	250	380	398	435	914	214

5.2.2. Scenario 2

This scenario uncovers a weakness of TNSOA. From the two Figures (9, 10) we can see that this algorithm suffers most from the lack of information which characterizes this scenario as it converges to a solution much more inaccurate (large RMSE) than the solutions provided by the other algorithms, although the cost function is decreased at about the same rate. Even TNFD manages to converge to a better minimizer and it only uses 12 outer iterations as opposed to the 45 of TNSOA. It is debatable whether the apparent faster convergence of the latter on the first iterations (up to 10 forward model runs) might make it worth using.

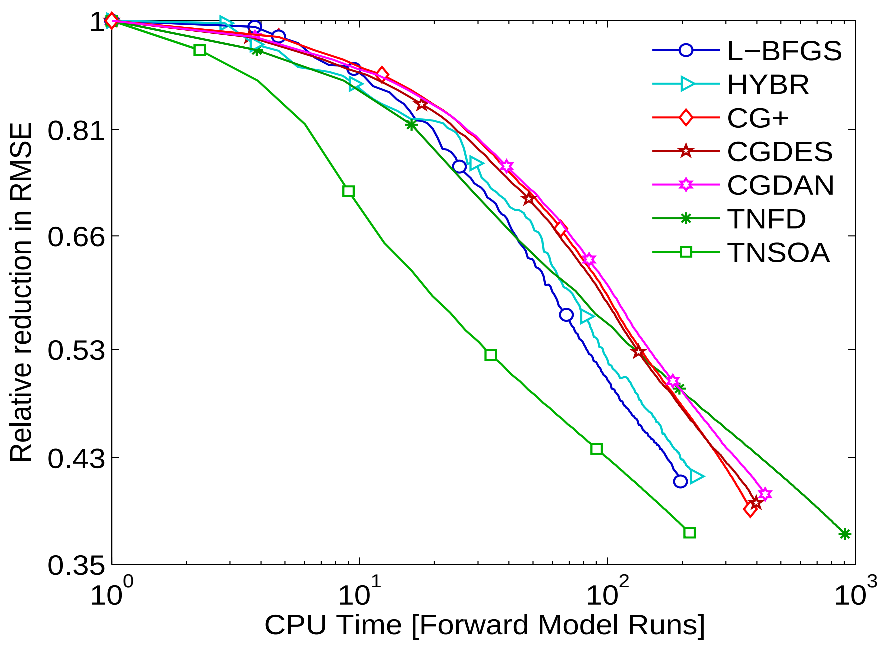

Figure 8. RMSE evolution with compute time for implicit timestepping model. Scenario 1: Perfect and dense observations, diagonal B

Another notable fact is the behavior of CGDES and CGDAN which are faster over the first many iterations than CG+. By looking at the iteration markers we can see that all three trajectories follow about the same steps so the superior performance of the former two should be attributed to the linesearch procedure of Hager [18] over the one implemented in MINPACK.

Table 8. Implicit timestepping model. Scenario 2: Perturbed and sparse observations, diagonal B

	L-BFGS	HYBR	CG+	CGDES	CGDAN	TNFD	TNSOA
RMSE	6.95e-1	6.96e-1	6.96e-1	6.96e-1	6.96e-1	7.2e-1	7.8e-1
$\mathcal{J}(U^a)$	4e-3	4e-3	4e-3	4e-3	4e-3	4.1e-3	6.1e-3
$\lVert\nabla\mathcal{J}(U^a)\rVert$	4e-4	8e-4	2e-4	4e-4	5e-4	1.6e-3	2.8e-2
Iterations	200	200	200	200	200	12	45
FWD	208	245	402	488	468	158	46
ADJ	208	245	402	244	227	158	46
SOA	0	0	0	0	200	0	134
CPU Time	197	232	380	430	448	134	65

5.2.3. Scenario 3

For the implicit model, the third scenario confirms the superior performance of TNSOA over other algorithms and also the high cost of doing finite differences for TN. A peculiarity occurs in the minimization undertaken by L-BFGS as although the cost function seems to evolve normally in time, the RMSE is not reduced significantly. Excluding L-BFGS, the mean relative reduction in RMSE over all algorithms amounts at about 37%.

The dotted line in Figures 12 and 12 corresponds to a data assimilation scenario ran with TNSOA that is using a maximum limit of 30 inner iterations instead of 15. Factorizing the Hessian with a higher accuracy helps the convergence rate but it also proves to be less costly. In order to further investigate this issue we present in Figures 13, 14 a comparison for this third scenario setting between the two truncated Newton considered throughout these tests, TNFD and TNSOA with a limit of 15 inner iterations, against TNSOA ran with a cap of 5, 30 and 50 in-

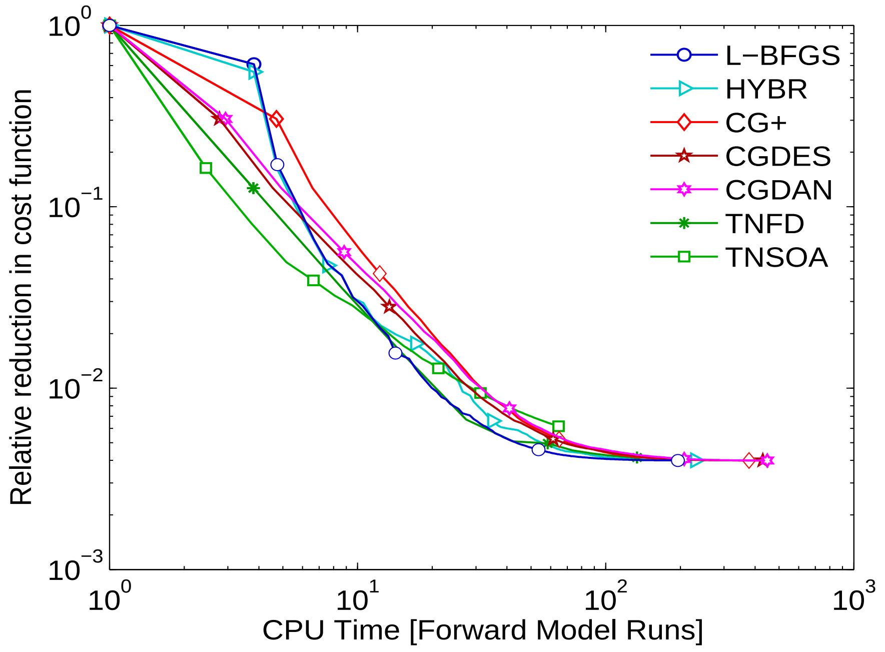

Figure 9. Cost function evolution with compute time for implicit timestepping model. Scenario 2: Perturbed and sparse observations, diagonal B

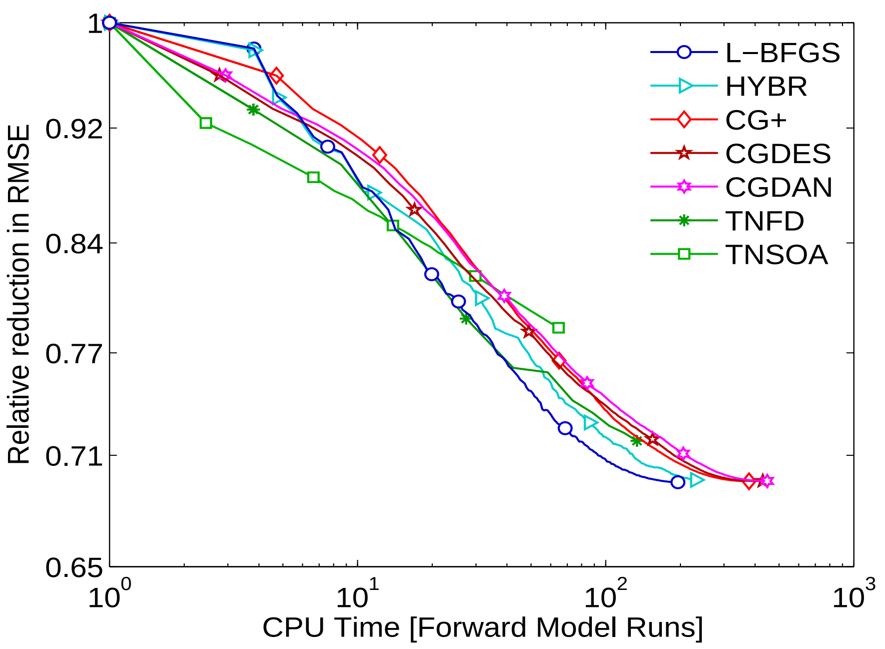

Figure 10. RMSE evolution with compute time for implicit timestepping model. Scenario 2: Perturbed and sparse observations, diagonal B

ner iterations. TNSOA-5 and TNFD-15 evolve the solution head-to-head although TNSOA-5 reaches the stopping criteria earlier on. TNSOA-30 is definitely an improvement over TNSOA-15 in speed of convergence but TNSOA-50 doesn't bring a significant improvement over the latter. Since the inner iterations are conjugate gradient iterations, the speed of convergence is related to the conditioning number of the system and for our particular system the optimum number of inner iterations seems to be somewhere around 30. In applications where truncated Newton is employed, this can be improved through preconditioning which would cluster the eigenvalues together and allow the factorization of the Hessian to be computed faster.

Table 9. Implicit timestepping model. Scenario 3: Perturbed and sparse observations, nondiag. B

	L-BFGS	HYBR	CG+	CGDES	CGDAN	TNFD	TNSOA
RMSE	8.3e-1	3.8e-1	3.7e-1	3.8e-1	3.8e-1	3.7e-1	3.7e-1
$\mathcal{J}(U^a)$	2.2e-3	1.1e-3	7.9e-4	8.4e-4	8.2e-4	8.4e-4	8.3e-4
$\|\nabla \mathcal{J}(U^a)\|$	3.1e-1	7e-1	1.4e-1	3.5e-1	3.2e-1	4.4e-1	2.8e-1
Iterations	200	200	200	200	200	57	57
FWD	210	242	402	204	212	901	58
ADJ	210	242	402	404	477	901	58
SOA	0	0	0	0	228	0	843
CPU Time	199	229	381	208	251	832	197

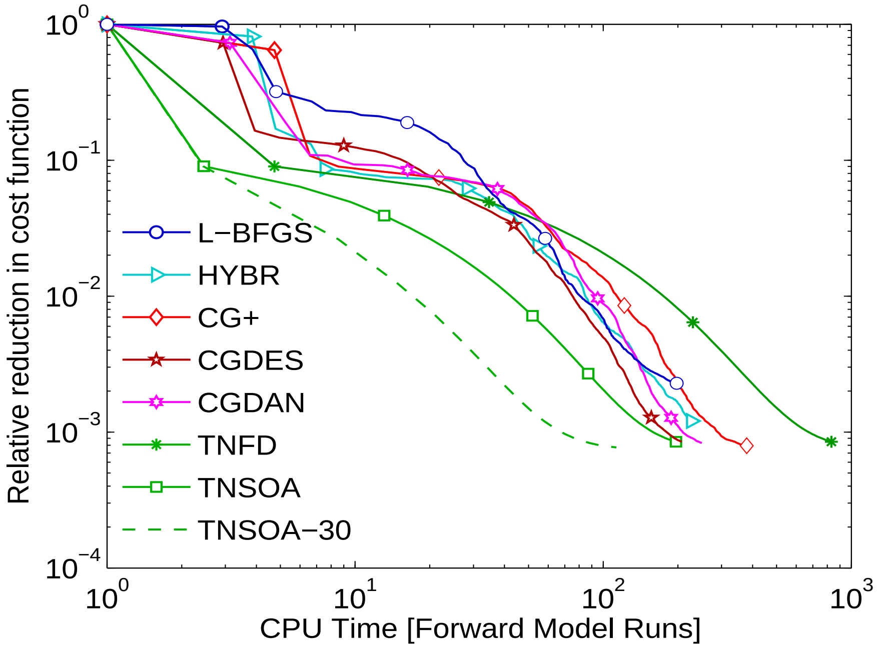

Figure 11. Cost function evolution with compute time for implicit timestepping model. Scenario 3: Perturbed and sparse observations, nondiagonal B

6. Conclusions and Future Work

This paper discusses the construction of second order adjoints for numerical models and their use in the solution of PDE-constrained optimization problems. While first order adjoints efficiently compute the gradient of a cost function (defined on the model output) with respect to a large number of model parameters, second order adjoints allow to efficiently calculate products between the Hessian of the cost function and user defined vectors. While it is well accepted that second order information is important for obtaining accurate minimizers, the large computational cost of obtaining second derivatives has restricted their use in large scale optimization. In this work we show that Hessian-vector products obtained via adjoint models can be usefully employed in PDE constrained optimization.

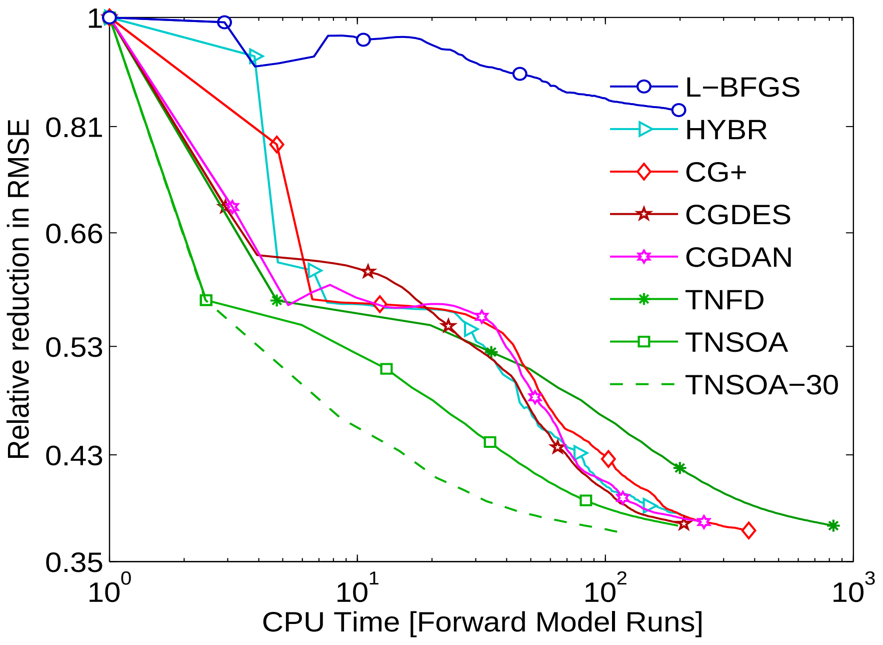

Figure 12. RMSE evolution with compute time for implicit timestepping model. Scenario 3: Perturbed and sparse observations, nondiagonal B

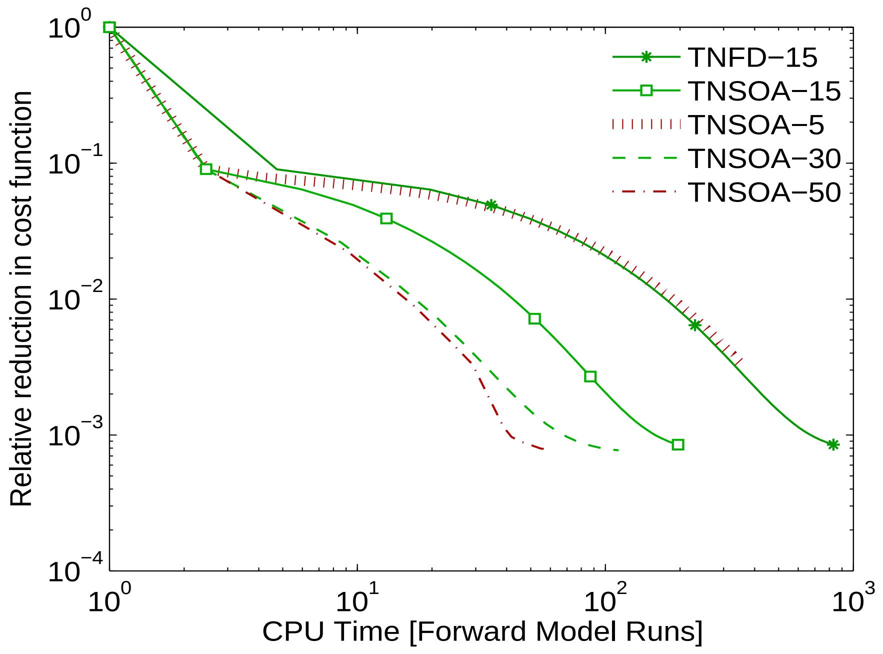

Figure 13. Cost function evolution with compute time for implicit timestepping model (TN comparison). Scenario 3: Perturbed and sparse observations, nondiagonal B

The construction of continuous and discrete SOA for time dependent models is presented. We implement discrete SOA for two different shallow water equation models. The first one uses explicit timestepping, and the associated adjoints are constructed using automatic differentiation. The resulting first and second order adjoint models have high computational costs, relative to the cost of the forward model. The CPU time for one FOA run is equivalent to that of 4 forward model

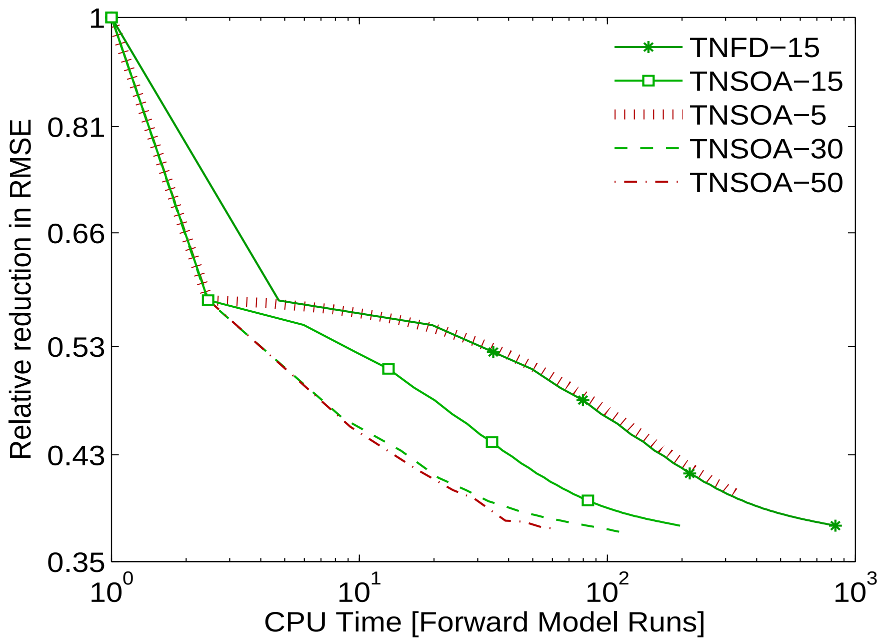

Figure 14. RMSE evolution with compute time for implicit timestepping model (TN comparison). Scenario 3: Perturbed and sparse observations, nondiagonal B

runs while SOA as 13 forward model runs. The second shallow water model uses implicit timestepping, and the discrete adjoints are derived and implemented by hand. The most expensive forward model calculations (LU decompositions) are checkpointed, and reused in the adjoint runs. As a consequence the cost of each adjoint is only a fraction of the cost of the forward model. Therefore, the two shallow water tests illustrate two very different scenarios: in the first the adjoint calculations are relatively expensive, while in the second their cost is relatively small.

An extensive numerical study is performed to assess the impact of using second order adjoints in four dimensional variational data assimilation. This inverse problem is of much interest in operational oceanography, weather, and air quality forecasting. Several optimization algorithms that employ second order information (TNSOA, Daniel CG, Moser-Hald) are compared against methods that use only first order derivatives (L-BFGS, Hybrid, nonlinear CG, CG Descent). Some methods (e.g., Moser-Hald) need to be reorganized such as to employ Hessian-vector product operations. For some algorithms high quality implementations are available in existing software libraries. Other algorithms (Daniel CG, Moser-Hald) have been implemented by the authors; they show promising results, and their performance is likely to benefit from improved implementations.

When second order adjoints are costly relative to the forward and the first order adjoint models, they are not recommended for use. This situation is illustrated by the explicit shallow water model. The high cost per iteration cannot be offset by the smaller number of iterations, and the minimization process is overall more expensive. The analyses are not clearly superior to those obtained with first order methods. In other words, there is no trade-off between performance and accuracy; similar analyses take longer to compute. For the explicit model, L-BFGS, Hybrid and nonlinear conjugate gradients perform best. Their widespread popularity with the data assimilation community is fully justified.

However, when FOA and SOA have similar computational times, the minimization algorithms employing second order information are very competitive. This situation is illustrated by the implicit shallow water model, where the results of costly LU decompositions performed during the forward run are reused during the adjoint runs. The CGDAN algorithm (with our research implementation) performs similarly with other nonlinear conjugate gradients (with high quality implementations); all provide similar analyses. For this scenario TNSOA has the best performance in terms of the convergence speed and solution accuracy. When configured to use no more than 50 inner iterations, TNSOA provides the best minimizer among all methods, in the shortest CPU time (less than 100 forward model runs).

In an operational data assimilation setting, the analysis has to be provided in a fixed compute time interval. Therefore, we are also interested in those algorithms that decrease the 4D-Var cost function the most within the first few iterations. For explicit timestepping, L-BFGS and Hybrid perform best in regard to this metric. For implicit timestepping, TNSOA is superior to all other algorithms tested. In summary, the use of second order information for large-scale PDE-constrained optimization problems is beneficial whenever high order derivatives can be obtained at a low computational cost. This can be achieved by re-using the results of expensive forward computations during the adjoint run.

A direction of future research is to develop algorithms that combine iterations based on first and second order information; the second order information is used only selectively throughout the optimization process. Another direction we plan to pursue is building approximate but inexpensive adjoints that capture only the significant features of the model. Tuning, preconditioning, and restarting the minimization algorithms are strategies that can significantly improve performance or accuracy. We believe that as the methodologies will continue to mature, second order adjoints will become a useful tool for the solution of large-scale inverse problems.

Acknowledgments

This work was supported by the National Science Foundation through the awards NSF DMS-0915047, NSF CCF-0635194, NSF CCF-0916493 and NSF OCI-0904397, by NASA through the award AIST-2005, and by the Houston Advanced Research Center through the award H-98/2008. The authors thank prof. Florian Potra for suggesting the Moser-Hald method.

References

[1] Z. Wang, I.M. Navon, F.X. LeDimet and X. Zou. The second order adjoint analysis: theory and applications. *Meteorology and Atmospheric Physics*, 50(1-3):3–20, 1992.

[2] Z. Wang, K. Droegemeier and L. White. The adjoint newton algorithm for large-scale unconstrained optimization in meteorology applications. *Computational Optimization and Applications*, 10(3):283–320, 1998.

[3] F.X. LeDimet, I. Navon and D. Daescu. Second order information in data assimilation. *Monthly Weather Review*, 130(3):629–648, 2002.

[4] B.D. Ozyurt, and P.I. Barton. Cheap second order directional derivatives of stiff ode embedded functionals. *SIAM Journal on Scientific Computing*, 26(5):1725–1743, 2005.

[5] D. Daescu and I.M. Navon. Efficiency of a POD-based reduced second-order adjoint model in 4d-var data assimilation. *International Journal of Numerical Methods in Fluids*, 53:985–1004, 2007.

[6] R.L. Raffard and C.J. Tomlin. Second order adjoint-based optimization of ordinary and partial differential equations with application to air traffic flow. In *2005 American Control Conference*. Portland, OR, USA., June 8-10, 2005.

[7] I. Charpentier, N. Jakse and F. Veerse. Second order exact derivatives to perform optimization on self-consistent integral equations problems. *Automatic differentiation of algorithms: from simulation to optimization*, pages 189–195, 2002.

[8] A. Alekseev and I.M. Navon. The analysis of an ill-posed problem using multiscale resolution and

second order adjoint techniques. *Computer Methods in Applied Mechanics and Engineering*, 190(15–17):1937–1953, 2001.

[9] R. Griesse and A. Walther. Towards matrix-Free AD-based preconditioning of KKT Systems in PDE-constrained optimization. In *GAMM Annual Meeting 2005 - Luxembourg*, pages 47–50. PAMM, 2005.

[10] B.D. Ozyurt and P.I. Barton. Application of targeted automatic differentiation to large scale dynamic optimization. *Lecture Notes in Computational Science and Engineering*, pages 235–247. Springer, 2005.

[11] A. Sandu and L. Zhang. Discrete second order adjoints in atmospheric chemical transport modeling. *Journal of Computational Physics*, 227(12):5949–5983, 2008.

[12] M. Alexe, A. Cioaca and A. Sandu Obtaining & using second order derivative information in the solution of large scale inverse problems. *High Performance Computing Symposium, Spring Simulation Multiconference*, Orlando, FL, USA., April 12-15, 2010

[13] A.K. Alekseev, I.M. Navon and J.L. Steward. Comparison of advanced large-scale minimization algorithms for the solution of inverse ill-posed problems. *Journal of Optimization Methods & Software*, Vol. 24, No. 1, February 2009, 63-87

[14] D. Daescu, A. Sandu and G.R. Carmichael. Direct and adjoint sensitivity analysis of chemical kinetic systems with KPP: II – Numerical validation and applications. *Atmospheric Environment*, 37:5097–5114, 2003.

[15] A. Sandu, D. Daescu and G.R. Carmichael. Direct and adjoint sensitivity analysis of chemical kinetic systems with KPP: I – Theory and Software Tools. *Atmospheric Environment*, 37:5083–5096, 2003.

[16] A. Sandu, D. Daescu, G.R. Carmichael, and T. Chai. Adjoint sensitivity analysis of regional air quality models. *Journal of Computational Physics*, 204:222–252, 2005.

[17] J.J., More', B. Garbow and K. Hillstrom. User guide for the MINPACK-2 test problem collection. *Argonne National Laboratory, Mathematics and Computer Science Division Report*, 1991.

[18] H. Zhang and W.W. Hager. A nonmonotone line search technique and its application to unconstrained optimization. *SIAM Journal on Optimization*, 14 (2004), pp. 1043-1056.

[19] J.E. Dennis and J.J, More'. Quasi-Newton methods, motivation and theory. *SIAM Review*, 19(1977), pp. 46-89

[20] J.E. Dennis and R.B. Schnabel. Numerical methods for unconstrained optimization and nonlinear equations. *Prentice-Hall, Englewood Cliffs*, NJ, 1983

[21] W.C. Davidon. Variable metric method for minimization. *SIAM Journal on Optimization*, 1(1991), pp. 1-17

[22] Unknown author. Practical methods of optimization. *John Wiley Sons, New York*, second ed., 1987

[23] R. H. Byrd, P. Lu, J. Nocedal and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16 (1995), no. 5, pp. 1190–1208.

[24] C. Zhu, R.H. Byrd, P. Lu and J. Nocedal. L-BFGS-B: a limited memory FORTRAN code for solving bound constrained optimization problems *Technical Report, NAM-11, EECS Department, Northwestern University*, 1994.

[25] D. Xie and T. Schlick. Efficient implementation of the truncated-Newton algorithm for large-scale chemistry applications. *SIAM Journal on Optimization*, 1998.

[26] D. Xie and T. Schlick. Remark on the updated truncated Newton minimization package, Algorithm 702 *ACM Transanctions on Mathematical Software*, 1998.

[27] J.L. Morales and J. Nocedal. Enriched methods for large-scale unconstrained optimization. *Computational Optimization Applied*, 21 (2002), pp. 143-154.

[28] D.N. Daescu and I.M. Navon. An analysis of a hybrid optimization method for variational data assimilation. *International Journal of Computational Fluid Dynamics*, 17(4) (2003), pp. 299-306.

[29] J. Moser. Stable and random motions in dynamical systems with special emphasis on celestial mechanics. *Herman Weyl Lectures, Annals of Mathematics Studies, Princeton University Press*, no. 77, 1973.

[30] O. Hald. On a Newton–Moser type method. *Numerical Mathematics*, v23, pp. 411-425.

[31] F.A. Potra and V. Ptak. Sharp error bounds for Newton process. *Numerical Mathematics* v3, pp. 63-72.

[32] M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 6 (1992).

[33] R. Fletcher and C.M. Reeves. Function minimization by conjugate gradients. *Computer Journal*, 7 (1964), pp. 149-154

[34] J.C. Gilbert and J. Nocedal. Global convergence properties of conjugate gradient methods. *SIAM Journal on Optimization*, Vol. 2 (1992), pp. 21-42.

[35] W.W. Hager and H. Zhang. CG-DESCENT, A conjugate gradient method with guaranteed descent. , Jan. 15 2004

[36] J.W. Daniel. The conjugate gradient method for linear and nonlinear operator equations. *SIAM Journal on Numerical Analysis*, 4(1):10-26, March 1967

[37] J.W. Daniel. A correction concerning the convergence rate for the conjugate gradient method. *SIAM Journal on Numerical Analysis*, 7:277-280, 1970.

[38] J.W. Daniel. Convergence of the conjugate gradient method with computationally convenient modifications. *Numerische Mathematik*, 10:125-131, 1967.

[39] R. Liska and B. Wendroff. Composite schemes for conservation laws. *SIAM Journal of Numerical Analysis*, Vol. 35, No. 6, pp. 2250-2271, December 1998.

[40] R. Giering and Th. Kaminski. Recipes for adjoint code construction. *ACM Transactions On Matematical Software*, Vol. 24, No. 4, pp. 437-474, 1998.

[41] R. Giering. Tangent linear and Adjoint Model Compiler, Users manual 1.4. *http://www.autodiff.com/tamc*, 1999.

[42] A. Griewank. On automatic differentiation. 1988.

[43] R. Courant, E. Isaacson and M. Rees. On the solution of nonlinear hyperbolic differential equations by finite differences. *Communications in Pure Applied Mathematics*, 5, 243-255, 1952.

[44] J. Crank and P. Nicolson. A practical method for numerical evaluation of solutions of partial differen-

tial equations of the heat conduction type. *Proceedings of Cambridge Philosophical Society*, 43, 50-67, 1947.

[45] T. A. Davis. A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, vol 30, no. 2, pp. 165-195, June 2004.

[46] T. A. Davis. Algorithm 832: UMFPACK, an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, vol 30, no. 2, pp. 196-199, June 2004.

[47] T. A. Davis and I. S. Duff. A combined unifrontal/multifrontal method for unsymmetric sparse matrices. *ACM Transactions on Mathematical Software*, vol. 25, no. 1, pp. 1-19, March 1999.

[48] T. A. Davis and I. S. Duff. An unsymmetric-pattern multifrontal method for sparse LU factorization. *SIAM Journal on Matrix Analysis and Applications*, vol 18, no. 1, pp. 140-158, Jan. 1997.

[49] Q. Xiao, Y. Kuo, Z. Ma, W. Huang, X. Huang, X. Zhang, D.M. Barker and J. Michalakes. Development of the WRF adjoint modeling system and its application to the investigation of the May 2004 McMurdo Antarctica severe wind event. *Monthly Weather Review*, 136, 3696-3713, 2008.

[50] R. Daley. Atmospheric data analysis. *Cambridge University Press*, 1991.

[51] E. Kalnay. Atmospheric modeling, data assimilation and predictability. *Cambridge University Press*, 2002.