

# Automatic Approximation of the Marginal Likelihood in Nonlinear Hierarchical Models

Hans J. Skaug\* and David Fournier†

October 29, 2004

## Abstract

We show that the fitting of nonlinear hierarchical random effects models by maximum likelihood can be made automatic to the same extent that Bayesian model fitting can be automated by the program BUGS. The word ‘automatic’ here means that the technical details of computation are made transparent to the user. We achieve this by combining a technique from computer science known as ‘automatic differentiation’ with the Laplace approximation for calculating the marginal likelihood. Automatic differentiation, which should not be confused with symbolic differentiation, is mostly unknown to statisticians, and hence we review basic ideas and results. A software prototype that implements our approach has been developed, and the computational performance is reported for a selection of models from the mixed-model literature. In general, our approach performs well in comparison with existing software.

**Keywords:** Automatic differentiation, Empirical Bayes, Laplace approximation, Mixed models, Random effects.

---

\*Corresponding author. Address: Institute of Marine Research, Box 1870 Nordnes, 5024 Bergen, Norway. Email: skaug@imr.no

†Otter Research Ltd., Sidney, Canada

# 1 INTRODUCTION

Hierarchical models have gained widespread use in statistics during the last few decades (Hobert 2000). It is the invention of new computational algorithms, as well as the increased speed of computers, that has made the use of such models practical. Both Bayesians and frequentists will agree that the heart of the computational problem is the numerical evaluation of a high dimensional integral. Laplace’s method of integration has been used both for calculating the normalizing constant of Bayesian posterior distributions (Tierney & Kadane 1986) and to obtain the marginal likelihood in frequentist random effects models (Breslow & Lin 1995). In order to evaluate the Laplace approximation, one needs to evaluate the Hessian matrix  $\mathbf{H}$  of the loglikelihood with respect to the random effects. For complex hierarchical models, calculating all second order partial derivatives by hand is both tedious and error prone. Our main message is that this burden can be lifted from the shoulders of the statistician, and we show that  $\mathbf{H}$  can be obtained automatically by a technique known as Automatic Differentiation (AD) (Griewank 2000). Given as input a computer code which evaluates a mathematical function  $g(\mathbf{u})$ , an AD tool transforms the code into a new program which evaluates the derivatives of  $g$  to machine precision. There is also the reverse algorithm for differentiation, and the accompanying “cheap gradient principle”, that we discuss later. Automatic Differentiation is largely unknown to statisticians (Skaug 2002), and among those who have heard about it there seems to be a confusion with symbolic differentiation, as performed by for instance the program Mathematica.

In the present paper we discuss hierarchical models from a frequentist, or empirical Bayes, perspective. Our models contain latent random variables  $\mathbf{u}$  that are integrated out of the likelihood through the Laplace approximation, and parameters  $\boldsymbol{\theta}$  that are estimated by maximum likelihood. In the mixed model terminology,  $\mathbf{u}$  is the vector of

random effects, and  $\boldsymbol{\theta}$  contains the fixed effects and the variance components. Denote by  $l^*(\boldsymbol{\theta})$  the loglikelihood approximation based on the Laplace approximation. Since direct (analytical) maximization of  $l^*(\boldsymbol{\theta})$  usually is impossible, we must resort to numerical optimization. Accurate derivative information greatly helps numerical optimization algorithms in locating the optimum of the objective function. As noted by several authors (Raudenbush et al. 2000, Bell 2001, e.g.), the expression for the gradient of the Laplace approximation  $l^*(\boldsymbol{\theta})$  involves up to third order partial derivatives of the joint loglikelihood  $g(\mathbf{u}, \boldsymbol{\theta})$ . We shall devise a scheme for evaluating these derivatives by AD, and derive its computational complexity.

Bayesian hierarchical models have been made available to a large group of research workers through the appearance of the software system BUGS (Gilks et al. 1994). From a description of the model in a language that resembles S-Plus, BUGS automatically sets up the Markov chain needed for sampling from the posterior distribution. Software with the same level of flexibility and automatization as offered by BUGS is lacking in the frequentist domain, but provision for fitting restricted classes of hierarchical models, such as nonlinear mixed effects models (Pineiro & Bates 2000) and generalized linear mixed models (Ruppert et al. 2003, p. 203), has been added to the major statistical software packages. According to Gilks et al. (1994), one of the original goals for the BUGS project was to create a system that could “accommodate a very large class of models”. As a result, today we see BUGS being used in a wide range of scientific disciplines. A second goal for BUGS was that it should be “automatic”, in the sense of hiding from the user the technical details of (MCMC) computations. In the present paper we explore how AD, in conjunction with the Laplace approximation, can be used to fulfil the same goals in an empirical Bayes setting.

To make the above ideas concrete, we have developed a prototype system, building

on the parameter estimation software AD Model Builder (Fournier 2001). The user of our system formulates a joint loglikelihood  $g(\mathbf{u}, \boldsymbol{\theta})$  in the programming language C++, and based on this input the software automatically calculates the maximum likelihood estimate of  $\boldsymbol{\theta}$  by maximizing  $l^*(\boldsymbol{\theta})$ . It is natural to make comparison to BUGS, with respect to the possibility and ease of formulating complex statistical models, and also with respect to computational efficiency. Apart from the obvious fact that the output from BUGS is the posterior distribution of  $\boldsymbol{\theta}$  (and  $\mathbf{u}$ ), while our system calculates the maximum likelihood estimate of  $\boldsymbol{\theta}$ , there are at least two differences that can be pointed out. Firstly, our system can not handle discrete latent variables (because it is based on the Laplace approximation), while in BUGS latent variables can be of both discrete and continuous type. The response variable can be discrete (as in Poisson regression for instance) in our system, however. Secondly, AD-algorithms can be parallelized (Griewank 2000, p. 330) so that they can run on a cluster of workstations, while this is not possible with the MCMC algorithms currently used in BUGS. In our view, the latter point is important, because the largest gain in computational power that will be available to statisticians in the near future is likely to come from the new 'grid technology' where thousands of workstations are connected by fast networks.

The rest of the paper is organized as follows: Section 2 sets up the framework for hierarchical models, Section 3 outlines the Laplace approximation and automatic differentiation, while examples are studied in Section 4. Section 5 provides some concluding remarks.

## 2 HIERARCHICAL MODELS

Let  $\mathbf{y} = (y_1, \dots, y_n)$  be a vector of observations, and let  $\mathbf{u} = (u_1, \dots, u_q)$  be a vector of latent random variables (random effects) influencing the value of  $\mathbf{y}$ . The conditional

density of  $\mathbf{y}$  given  $\mathbf{u}$  is denoted by  $f(\mathbf{y}|\mathbf{u})$ , and the marginal density of  $\mathbf{u}$  by  $h(\mathbf{u})$ . In most situations both  $f$  and  $h$  will depend on unknown parameters  $\boldsymbol{\theta} = (\theta_1, \dots, \theta_m)$ , and to indicate this we write  $f_{\boldsymbol{\theta}}$  and  $h_{\boldsymbol{\theta}}$ . This is the classical parametric empirical Bayes framework (Carlin & Louis 1996). There may of course be more than two levels in the hierarchy, but to keep the notation simple we will only consider two levels of random variables:  $\mathbf{u}$  and  $\mathbf{y}$ .

The likelihood function for  $\boldsymbol{\theta}$  must be based on the marginal distribution of  $\mathbf{y}$ , which is obtained by integrating out  $\mathbf{u}$  from the joint density  $f_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{u})h_{\boldsymbol{\theta}}(\mathbf{u})$ . This yields the marginal likelihood

$$L(\boldsymbol{\theta}) = \int f_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{u})h_{\boldsymbol{\theta}}(\mathbf{u}) d\mathbf{u} = \int \exp \{g(\mathbf{u}, \boldsymbol{\theta})\} d\mathbf{u}, \quad (1)$$

where

$$g(\mathbf{u}, \boldsymbol{\theta}) = \log \{f_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{u})\} + \log \{h_{\boldsymbol{\theta}}(\mathbf{u})\} \quad (2)$$

is the joint loglikelihood, also referred to as the penalized loglikelihood when studied as a function of  $\mathbf{u}$  only. The computational challenge lies in evaluating this integral with the view of maximizing  $L(\boldsymbol{\theta})$ .

This framework covers random effects models and state space models, as shown in the examples of Section 4 below. Moreover, as emphasized by Wand (2003), semiparametric regression may be treated within the mixed model framework. To illustrate this point we consider a model that is conceptually simple, but which to our knowledge cannot be fit in any standard software package.

## 2.1 Nonparametric estimation of mean and variance

For data  $\{(y_i, x_i), i = 1, \dots, n\}$  consider the following generalization of the standard linear regression:

$$y_i = \mu(x_i) + \sigma(x_i)\varepsilon_i,$$

where  $\mu(x)$  and  $\sigma(x)$  are smooth functions and  $\varepsilon_i \sim N(0, 1)$ . To enforce the constraint  $\sigma(x) > 0$  we define  $\sigma(x) = \exp\{v(x)\}$ , where  $v(x)$  is also a smooth function. A common approach to nonparametric estimation is to place roughness penalties on the functions to be estimated. The penalized likelihood may metaphorically be written

$$-\sum_{i=1}^n v(x_i) - \frac{1}{2} \sum_{i=1}^n \left\{ \frac{y_i - \mu(x_i)}{\exp v(x_i)} \right\}^2 - \lambda_1 \int \{\mu''(x)\}^2 dx - \lambda_2 \int \{v'(x)\}^2 dx,$$

which puts a penalty on variation in  $v(x)$  and a penalty on any variation in  $\mu(x)$  beyond a linear function. For practical calculations, the integrals will be replaced by quadratic forms  $\mathbf{u}_1^T \mathbf{S}_1 \mathbf{u}_1$  and  $\mathbf{u}_2^T \mathbf{S}_2 \mathbf{u}_2$ , where  $\mathbf{u}_1 = \{\mu(\mathbf{x}_1), \dots, \mu(\mathbf{x}_n)\}$ ,  $\mathbf{u}_2 = \{v(\mathbf{x}_1), \dots, v(\mathbf{x}_n)\}$ , and  $\mathbf{S}_1$  and  $\mathbf{S}_2$  are non-negative definite matrices determined by  $x_1, \dots, x_n$ . By viewing  $\mathbf{u}_1$  and  $\mathbf{u}_2$  as zero mean Gaussian vectors with covariance matrices  $\mathbf{S}_1^{-1}$  and  $\mathbf{S}_2^{-1}$ , we see that this model falls into the empirical Bayes framework with  $\boldsymbol{\theta} = (\lambda_1, \lambda_2)$ . (If either  $\mathbf{S}_1$  or  $\mathbf{S}_2$  are singular matrices a modification is needed to retain this interpretation.)

### 3 COMPUTATIONAL TECHNIQUES

The Laplace approximation of the integral (1) is based on a second order Taylor expansion of the penalized loglikelihood  $g(\cdot, \boldsymbol{\theta})$  around the point

$$\hat{\mathbf{u}}(\boldsymbol{\theta}) = \underset{\mathbf{u}}{\operatorname{argmax}} g(\mathbf{u}, \boldsymbol{\theta}). \quad (3)$$

The likelihood approximation is given as

$$L^*(\boldsymbol{\theta}) = \det \{\mathbf{H}(\boldsymbol{\theta})\}^{-1/2} \exp [g \{\hat{\mathbf{u}}(\boldsymbol{\theta}), \boldsymbol{\theta}\}],$$

where

$$\mathbf{H}(\boldsymbol{\theta}) = -\frac{\partial^2}{\partial \mathbf{u}^2} g(\mathbf{u}, \boldsymbol{\theta}) \Big|_{\mathbf{u}=\hat{\mathbf{u}}(\boldsymbol{\theta})}. \quad (4)$$

In purely Gaussian models, i.e. when  $g(\cdot, \boldsymbol{\theta})$  is a quadratic function, the Laplace approximation is exact. In the nonlinear case, the approximation is defined only for those  $\boldsymbol{\theta}$

such that  $\mathbf{H}(\boldsymbol{\theta})$  is positive definite, i.e.  $\det \{\mathbf{H}(\boldsymbol{\theta})\} > 0$ , and hence we can evaluate the determinant by doing a Cholesky factorization of  $\mathbf{H}(\boldsymbol{\theta})$ . A more theoretical treatment of the Laplace approximation may be found in Barndorff-Nielsen & Cox (1989).

For numerical computations it is convenient to work with the logarithm of  $L^*(\boldsymbol{\theta})$ :

$$l^*(\boldsymbol{\theta}) = -0.5 \log \det \{\mathbf{H}(\boldsymbol{\theta})\} + g \{\hat{\mathbf{u}}(\boldsymbol{\theta}), \boldsymbol{\theta}\}. \quad (5)$$

There are two quantities appearing in (5) that are not readily available:  $\hat{\mathbf{u}}(\boldsymbol{\theta})$  and  $\mathbf{H}(\boldsymbol{\theta})$ . To calculate  $\hat{\mathbf{u}}(\boldsymbol{\theta})$  one can use a standard nonlinear function optimizer (Nocedal & Wright 1999). Note that when using an iterative method to maximize  $l^*(\boldsymbol{\theta})$ , we must re-evaluate  $\hat{\mathbf{u}}(\boldsymbol{\theta})$  at each iteration step. Evaluation of  $\mathbf{H}(\boldsymbol{\theta})$  by AD is methodologically more challenging. At this point we need to say a few more words about what AD really is.

### 3.1 Automatic differentiation

Automatic differentiation (AD) refers to a collection of techniques which exploit the chain rule of calculus to evaluate derivatives of functions defined in computer programs automatically. It is useful to initially think of AD as the process of physically inserting derivative code into the original program. We start by looking at first order derivatives. Higher order derivatives are obtained by repeated application of first order AD. Below we review basic algorithms and result. For a more comprehensive introduction to AD, including code examples, the reader is referred to Griewank (2000).

There are two “modes” referred to as forward and reverse. Consider a computer program which evaluates a scalar function  $g(\mathbf{u})$  of  $q$  independent variables  $\mathbf{u} = (u_1, \dots, u_q)$ . Conceptually, such a program can be viewed as a sequence of unary (exp, log, etc.) and binary operations (addition, subtraction, division, etc.) of the form

$$t = \varphi(r, s). \quad (6)$$

To illustrate the forward mode, we first aim at calculating the derivative of  $g$  w.r.t one of the independent variables,  $u_1$  say. Define a set of “derivative variables”  $\dot{r}, \dot{s}, \dot{t}$ , and assume that  $\dot{r}$  and  $\dot{s}$  already have been assigned the values  $\dot{r} = \partial r / \partial u_1$  and  $\dot{s} = \partial s / \partial u_1$ . It then follows by the chain rule that

$$\dot{t} = \frac{\partial}{\partial r} \varphi(r, s) \dot{r} + \frac{\partial}{\partial s} \varphi(r, s) \dot{s} \quad (7)$$

holds the value of  $\partial t / \partial u_1$ . In the special case of  $\varphi(s, t) = s \cdot t$  we have

$$\dot{t} = s \dot{r} + r \dot{s}. \quad (8)$$

Each time the code statement (6) is executed, the statement (7) is executed in addition. From the early days of scientific computing, people have been inserting derivative code into their programs by hand. It is not difficult to imagine that this line-by-line insertion of derivative code can be performed by a preprocessor, and hereof the name automatic differentiation.

It is clear that the number of machine operations needed to perform the calculation (7) is only a small number that required to perform (6). For instance, when  $\varphi(r, s) = r \cdot s$  it takes one multiplication to evaluate  $t$ , while it takes two multiplication and one addition to evaluate  $\dot{t}$  given by (8). In general, let  $C(f)$  denote the cost of evaluating a function  $f$ , measured in terms of the required number of unary and binary operations. It is not difficult to establish the following inequality.

**Lemma 1** *In forward-mode AD it holds that  $C(\partial g / \partial u_j) < 4C(g)$ ,  $j = 1, \dots, q$ .*

It is possible to make this inequality sharper (Griewank 2000, Sec. 3.2). Since a separate statement (7) must be executed for each component of  $\mathbf{u}$  we have  $C(\nabla g) < q \cdot 4C(g)$  under forward-mode AD, where  $\nabla g = (\partial g / \partial u_1, \dots, \partial g / \partial u_q)$ .

In reverse-mode AD one calculates sensitivities  $\partial g / \partial t$  of the dependent variable  $g$  with respect to all intermediate variables  $t$  in the program, including the independent variables

$u_1, \dots, u_q$ . This can be done very efficiently if the derivative calculations are performed in the reverse order of the original program flow. Consider again the elementary operation (6). If the derivative variable  $\bar{t}$  holds the value  $dg/dt$  it follows by the chain rule that  $\bar{r}$  and  $\bar{s}$ , defined through

$$\bar{r} = \frac{\partial}{\partial r} \varphi(r, s) \bar{t} \quad \text{and} \quad \bar{s} = \frac{\partial}{\partial s} \varphi(r, s) \bar{t}, \quad (9)$$

hold the values of  $dg/dr$  and  $dg/ds$ , respectively. (There is the additional complication that  $s$  may be a function of  $r$ , or vice versa.) Note that  $r$  and  $s$  must have been assigned values at the time when the calculation (9) takes place. Thus, prior to application of the reverse algorithm one must perform an ordinary run of the program (without any derivative calculations) in which all intermediate variables of the program get assigned their values. The reverse algorithm then starts out with the initialization  $\bar{g} = 1$  (because  $dg/dg = 1$ ) and propagates the sensitivities backwards by repeated application of (9). The key result in automatic differentiation is the following (known as the “cheap gradient principle”):

**Lemma 2** *For reverse-mode AD it holds that  $C(\nabla g) < 4C(g)$ .*

This result comes as a surprise, because we are so used to think that the cost of getting the gradient is  $(q + 1)C(g)$ , as is the case for the method of finite differences. With an understanding of how the reverse algorithm works it is not difficult to realize why the result holds. A formal proof can be found in Griewank (2000, Sec. 3.4). There is however a price to be paid: all the intermediate variables  $t$  must be kept in memory during the execution of the reverse algorithm, and this may cause the computer to run out of memory for large computations. This is a serious practical concern that takes away some of the glory of Lemma 2. The reverse mode of differentiation is known under various names in different fields, for instance in neural networks, where it is known as the back-propagation

algorithm (Rumelhart et al. 1986).

We have presented AD as being the process of inserting derivative code into the original program. This is in fact the approach used by many AD tools. We have used another approach, in which operator overloading in C++ is used to make the operation (7) happen as a consequence of (6) at run time (Griewank 2000, Sec. 5.1).

How is AD different from ‘symbolic differentiation’ as performed by for instance the software Mathematica? The aim of AD is to produce numerical derivatives, not analytical formulae meant to be read by the human eye. It is true that formulae produced with symbolic differentiation also can be evaluated numerically. The difference is, however, that AD can be applied to ordinary computer programs containing constructions such as if-statements and loops, which are outside the scope of symbolic differentiation. Also, the reverse mode of differentiation is not implemented in packages for symbolic differentiation.

### 3.2 Laplace approximation evaluated by AD

To evaluate the matrix  $\mathbf{H}$  defined in (4) we make repeated use of AD. First, forward-mode AD is used to obtain a program that evaluates  $\partial g/\partial u_j$ . This program is then the target for a second application of AD, now in reverse mode, producing

$$\left( \frac{\partial^2 g}{\partial u_j \partial u_1}, \dots, \frac{\partial^2 g}{\partial u_j \partial u_q} \right),$$

which is the  $j$ 'th row of  $\mathbf{H}$ . Invoking Lemma 1 and Lemma 2, we have that the cost of getting a single row in  $\mathbf{H}$  is less than  $4 C(\partial g/\partial u_j) < 16 C(g)$ . The cheap gradient principle can only be appealed to once, so to get the whole Hessian we must pay  $C(\mathbf{H}) < q \cdot 16 C(g)$ .

We then turn to the problem of calculating the first order derivative of the Laplace approximation (5) w.r.t.  $\boldsymbol{\theta}$ . In the following we write  $\hat{\mathbf{u}}$  instead of  $\hat{\mathbf{u}}(\boldsymbol{\theta})$ , and we denote partial derivatives by subscript, such that for instance  $\mathbf{H} = \mathbf{g}_{uu}$ . Several authors

(Raudenbush et al. 2000, Bell 2001) have noted that

$$\frac{\partial \hat{\mathbf{u}}}{\partial \boldsymbol{\theta}} = -\mathbf{H}(\hat{\mathbf{u}}, \boldsymbol{\theta})^{-1} \mathbf{g}_{u\theta}(\hat{\mathbf{u}}, \boldsymbol{\theta}),$$

which follows from the implicit function theorem. Define  $l(\mathbf{u}, \boldsymbol{\theta}) = -0.5 \log \det \{\mathbf{H}(\mathbf{u}, \boldsymbol{\theta})\} + g(\mathbf{u}, \boldsymbol{\theta})$ , such that  $l^*(\boldsymbol{\theta}) = l(\hat{\mathbf{u}}, \boldsymbol{\theta})$ . From the chain rule it follows that

$$\frac{\partial}{\partial \boldsymbol{\theta}} l(\hat{\mathbf{u}}, \boldsymbol{\theta}) = \left( \frac{\partial \hat{\mathbf{u}}}{\partial \boldsymbol{\theta}} \right)^T l_u(\hat{\mathbf{u}}, \boldsymbol{\theta}) + l_\theta(\hat{\mathbf{u}}, \boldsymbol{\theta}), \quad (10)$$

where the superscript  $T$  denotes the matrix transpose. Thus, we need to calculate the first order partial derivatives  $l_u$  and  $l_\theta$ . From an AD perspective there is no reason to distinguish between the parameters  $\mathbf{u}$  and  $\boldsymbol{\theta}$ , so in the following we subsume  $\boldsymbol{\theta}$  into  $\mathbf{u}$ . Having made this observation, it is clear that the matrix  $\mathbf{g}_{u\theta}$  can be obtained by the same machinery as being used to calculate  $\mathbf{H}$ .

The computationally most challenging part of the Laplace approximation is the term  $\psi(\mathbf{H}) = \log \det(\mathbf{H})$ . To calculate  $\psi(\mathbf{H})$  and  $\nabla_{\mathbf{u}} \psi(\mathbf{H}(\mathbf{u}))$  we have devised an AD scheme involving six ‘‘codes’’ (see Table 1). The chain rule is applied at two levels: on a micro scale when producing each of the codes by AD, and on a macro scale when ‘‘gluing’’ the codes together. In Code 5 the derivatives of  $\psi(\mathbf{H})$  w.r.t. the elements of  $\mathbf{H}$  are calculated by reverse mode AD. We could instead have used the existing analytical formula for  $\nabla_{\mathbf{H}} \psi(\mathbf{H})$  (Searle et al. 1992, p. 457), but it is an open question which of these approaches is computationally the most efficient. Viewing  $\mathbf{u} \rightarrow \psi\{\mathbf{H}(\mathbf{u})\}$  as a single (composite) transformation we get from Lemma 2 that  $C[\nabla_{\mathbf{u}} \psi\{\mathbf{H}(\mathbf{u})\}] < 4C[\psi\{\mathbf{H}(\mathbf{u})\}]$ .

### 3.3 Numerical optimization

Finding the value of  $\boldsymbol{\theta}$  that maximizes the Laplace approximation (5) constitutes a nested optimization problem. We solve the ‘inner’ problem (3) using either a quasi Newton algorithm (Nocedal & Wright 1999, Chp. 8) or a limited memory Newton method (Nocedal

& Wright 1999, Chp. 9) followed by one or two proper Newton steps. Both of these algorithms benefit from having access to exact gradient information, which we calculate using reverse mode AD. The 'outer' problem is solved using the quasi Newton algorithm that is built into AD Model Builder, in combination with the expression (10) for  $\nabla l^*(\boldsymbol{\theta})$ . The convergence criterion used in Section 4 below is that all components of  $\nabla l^*(\boldsymbol{\theta})$  should be less than  $10^{-4}$  in absolute value. To achieve this accuracy we found that it was necessary to use a stricter convergence criterion ( $10^{-9}$ ) in the inner optimization. The observed Fisher information matrix is obtained by taking finite differences of  $\nabla l^*(\boldsymbol{\theta})$ .

An alternative strategy would be to formulate a constrained optimization problem: maximize  $g(\mathbf{u}, \boldsymbol{\theta})$  jointly with respect to  $\mathbf{u}$  and  $\boldsymbol{\theta}$ , subject to the constraint  $\nabla_{\mathbf{u}}g(\mathbf{u}, \boldsymbol{\theta}) = 0$ . Several efficient algorithms for constrained optimization exist (Nocedal & Wright 1999, Chp. 15), but their performance has yet not been tested in the current context.

It is useful to do the optimization of  $l^*(\boldsymbol{\theta})$  in different 'phases'. In the first phase one fixes the random effects  $u_i$  at zero, while doing the optimization with respect to  $\boldsymbol{\theta}$  (except those components determining the variance-covariance structure of the random effects). Thus, there is no Laplace approximation involved in the first phase. At the beginning of the next phase we have good starting values for a subset of  $\boldsymbol{\theta}$ . This strategy can speed up the optimization process considerably.

### 3.4 Conditional independence

For many hierarchical models  $\mathbf{H}$  is a highly sparse matrix, i.e. most of its elements are zero throughout all calculations. The simplest types of sparsity patterns occur when  $\mathbf{H}$  is block diagonal (arising from nested models) or when  $\mathbf{H}$  is a banded matrix, arising from state-space models. In these situations, the sparsity structure of  $\mathbf{H}$  has a probabilistic interpretation, in terms of certain subsets of  $\mathbf{u}$  being conditionally independent under the

posterior distribution  $p(\mathbf{u}|\mathbf{y})$ .

There can be a huge computational gain in exploiting the sparsity of  $\mathbf{H}$  in the calculations. Numerical linear algebra libraries underlying statistical software often come in two versions: one for dense (non-sparse) matrices and one for sparse matrices. Also, the sparseness of  $\mathbf{H}$  affects the way the AD calculations described in Section 3.2 should be organized.

## 4 EXAMPLES

The main purpose of this section is to measure the performance (computational speed) of our approach, and to provide comparison to other mixed-model software. Results for ten models/datasets are presented in Table 2. Eight of these datasets are taken from the literature and the remaining two are generated by simulation as explained below. All datasets and AD Model Builder source code are available as text files from <http://bemata.imr.no/software.html>.

### 4.1 Mixed logistic regression

This example provides a comparison to WinBUGS (<http://www.mrc-bsu.cam.ac.uk/bugs/>).

For  $i = 1, \dots, n$ , let the  $y_i$  be conditionally independent, with

$$\begin{aligned} y_i|\mathbf{u} &\sim \text{Bernoulli}(\pi_i), \\ \eta_i &= \log\left(\frac{\pi_i}{1-\pi_i}\right) = \mathbf{X}_i^T\boldsymbol{\beta} + \mathbf{Z}_i^T\mathbf{u}, \\ u_i &\stackrel{\text{i.i.d.}}{\sim} N(0, \sigma^2), \end{aligned}$$

where  $\mathbf{X}_i$  and  $\mathbf{Z}_i$  are covariate vectors of length  $p$  and  $q$ , respectively, and  $\boldsymbol{\beta}$  is a vector of regression parameters. Hence,  $\boldsymbol{\theta} = (\boldsymbol{\beta}, \sigma)$  are the parameters to be estimated by maximum likelihood. Often,  $\mathbf{Z}_i$  will be a dummy vector coding for one or more categorical covariates, and hence mainly have zero elements. However, for the purpose of making the model

computationally challenging, we drew all the elements of  $\mathbf{X}_i$  and  $\mathbf{Z}_i$  randomly (from a uniform  $[-2, 2]$  distribution). The parameters used were:  $n = 200$ ,  $p = 5$ ,  $q = 30$ ,  $\sigma = 0.1$  and  $\beta_j = 0$  for all  $j$ . As starting values for both AD Model Builder and WinBUGS we used  $\beta_j = -1$  and  $\sigma = 4.5$ . In WinBUGS we used a uniform  $[-10, 10]$  prior for  $\beta_j$  and a standard (in the WinBUGS literature) noninformative gamma prior on  $\tau = \sigma^{-2}$ . In AD Model Builder the parameter constraints  $\beta_j \in [-10, 10]$  and  $\log(\sigma) \in [-5, 3]$  were used in the optimization process.

On the simulated dataset AD Model Builder used 15 seconds to converge to the optimum of likelihood surface. On the same dataset we first ran WinBUGS (Version 1.4) for 5,000 iterations. The recommended convergence diagnostic in WinBUGS is the Gelman-Rubin plot (see the help files available from the menus in WinBUGS), which requires that at least two Markov chains are run in parallel. From the Gelman-Rubin plots for  $\sigma$  and  $\beta$  it was apparent that convergence occurred after approximately 2,000 iterations. A few of the components of  $\mathbf{u}$  did not yet seem to have reached equilibrium. The time taken by WinBUGS to perform 2,000 iterations of two chains was approximately 700 seconds. The maximum likelihood estimate of  $\sigma$  was 0.1692 as calculated by AD Model Builder, while the posterior mean estimate calculated by WinBUGS  $\sigma$  was 0.1862.

The robustness of the two approaches was investigated by decreasing the amount of data while holding  $p$  and  $q$  fixed. For  $n = 50$ , WinBUGS came up with an error message before convergence was reached. AD Model Builder converged without problems to the optimum of the likelihood function, although the observed Fisher information matrix showed that the parameters were weakly determined.

## 4.2 Poisson regression with spatially correlated random effects

Let  $\{u(\mathbf{z}), \mathbf{z} \in R^2\}$  be a Gaussian random field with covariance structure

$$\text{cov}\{u(\mathbf{z}), u(\mathbf{z}')\} = \sigma^2 \exp\{-\alpha^{-1}d(\mathbf{z}, \mathbf{z}')\},$$

where  $d(\mathbf{z}, \mathbf{z}')$  is the Euclidean distance between the points  $\mathbf{z}$  and  $\mathbf{z}'$  in the plane. Let  $y_1, \dots, y_n$  be observations made at locations  $\mathbf{z}_1, \dots, \mathbf{z}_n$ , respectively. Conditionally on  $\mathbf{u} = \{u(\mathbf{z}_1), \dots, u(\mathbf{z}_n)\}$  the  $y$ 's are independent with  $y_i \sim \text{Poisson}(\lambda_i)$ , where

$$\log(\lambda_i) = \mathbf{X}_i^T \boldsymbol{\beta} + u(\mathbf{z}_i).$$

Here,  $\mathbf{X}_i$  and  $\boldsymbol{\beta}$  are as in the mixed logistic regression above, and the vector of parameters is  $\boldsymbol{\theta} = (\boldsymbol{\beta}, \sigma, \alpha)$ . Denote by  $\Sigma_{\boldsymbol{\theta}}$  the covariance matrix of  $\mathbf{u}$ . The contribution to the joint loglikelihood (2) coming from the marginal distribution of  $\mathbf{u}$  is

$$\log(h_{\boldsymbol{\theta}}(\mathbf{u})) = -\frac{n}{2} \log(2\pi) - \frac{1}{2} \log \det \Sigma_{\boldsymbol{\theta}} - \frac{1}{2} \mathbf{u}' \Sigma_{\boldsymbol{\theta}}^{-1} \mathbf{u}.$$

Our goal is to determine the size of the largest grid that can be fit with the software prototype. For this purpose we simulated the process  $\{u(\mathbf{z})\}$  on the regular grid  $\{(i, j); i, j = 1, \dots, k\}$ , for  $k = 10$ . The parameter values used were  $\boldsymbol{\beta} = (10, 0)$ ,  $\alpha = 0.7$  and  $\sigma = 0.5$ , and the two columns of  $\mathbf{X}$  were taken to be orthonormal. In the simulated data set, the average value of  $y$  was 2.8. The time taken for the model to converge was 2 minutes. The largest model that could be fit appeared to be  $k = 25$ , i.e.  $n = k^2 = 625$ , with the large memory requirements of reverse-mode AD being the limiting factor.

## 4.3 Computational efficiency

Table 2 shows computation times for a variety of random effects models. Our implementation in AD Model Builder performs relatively well in comparison to the special purpose packages aML and NLME. As these packages are able to exploit special properties of the

model classes they are designed for, they can be expected to be faster than a general purpose tool such as ours.

The table clearly shows that our approach is computationally efficient compared to WinBUGS. It may be argued that comparison with WinBUGS is inappropriate because WinBUGS is generating samples from a posterior distribution, while our system is calculating an (approximate) maximum likelihood estimate. However, both approaches are meant as tools for making statistical inference, and inference can first start after convergence has been reached, although ‘convergence’ is defined somewhat differently for the two approaches. We thus believe that time to convergence is one of the important measures to compare.

## 5 DISCUSSION

We have demonstrated that empirical Bayes analysis of hierarchical models can be automated in the sense of Gilks et al. (1994). A strength of our approach is that we calculate the gradient  $\nabla l^*(\boldsymbol{\theta})$  to machine precision. Accurate gradient information is important in nonlinear models, both for following the path to the top of  $l^*(\boldsymbol{\theta})$  and for judging whether a satisfactory optimum has been reached. Of course, in situations where  $l^*(\boldsymbol{\theta})$  is multimodal our method can be trapped in a local maximum. To guard against this one should use multiple starting points for the parameter search.

In situations where the Laplace approximation is inaccurate, what can be done to improve it? For a given value of  $\boldsymbol{\theta}$ , the Laplace approximation (5) provides a Gaussian approximation, with mean vector  $\hat{\mathbf{u}}(\boldsymbol{\theta})$  and covariance matrix  $-\mathbf{H}(\boldsymbol{\theta})^{-1}$ , to the conditional distribution of  $\mathbf{u}$  given  $\mathbf{y}$ . This approximation can be used either as the sampling distribution in importance sampling or as part of a proposal distribution in the Metropolis-Hastings algorithm. Skaug (2002) used importance sampling with a  $N\{\hat{\mathbf{u}}(\boldsymbol{\theta}'), -\mathbf{H}(\boldsymbol{\theta}')^{-1}\}$

sampling distribution to improve upon the Laplace approximation (5). Alternatively, higher order Taylor expansions of the penalized loglikelihood (2) may be used as in Raudenbush et al. (2000). However, as the variance  $\sigma^2$  of the random effects  $u_i$  turns to zero, the Laplace approximation becomes increasingly accurate, and hence one gets the correct answer to the question whether the maximum of the likelihood function lies at the boundary ( $\sigma = 0$ ) or not.

As a final remark, we believe that AD has a large potential in statistics, also outside the context of the present paper. Many computations undertaken by statisticians would benefit from having access to accurate derivative information.

## References

- Barndorff-Nielsen, O. E. & Cox, D. R. (1989), *Asymptotic techniques for use in statistics*, Chapman & Hall.
- Bell, B. (2001), ‘Approximating the marginal likelihood estimate for models with random parameters’, *Applied mathematics and computation* **119**, 57–75.
- Breslow, N. E. & Lin, X. (1995), ‘Bias correction in generalised linear mixed models with a single component of dispersion’, *Biometrika* **82**, 81–91.
- Carlin, B. P. & Louis, T. A. (1996), *Bayes and Empirical Bayes Methods for Dana Analysis*, Chapman & Hall.
- Fournier, D. (2001), An introduction to AD MODEL BUILDER Version 6.0.2 for use in nonlinear modeling and statistics, Available from <http://otter-sch.com/admodel.htm>.
- Gilks, W. R., Thomas, A. & Spiegelhalter, D. J. (1994), ‘A language and program for complex Bayesian modelling’, *The Statistician* **43**, 169–178.

- Griewank, A. (2000), *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, SIAM, Philadelphia.
- Hobert, J. (2000), ‘Hierarchical models: A current computational perspective’, *Journal of the American Statistical Association* **95**, 1312–1316.
- Kuk, A. Y. C. & Cheng, Y. W. (1999), ‘Pointwise and functional approximations in Monte Carlo maximum likelihood estimation’, *Statistics and Computing* **9**, 91–99.
- Meyer, R. & Yu, J. (2000), ‘BUGS for a bayesian analysis of stochastic volatility models’, *Econometrics Journal* **3**, 198–215.
- Nocedal, J. & Wright, S. J. (1999), *Numerical Optimization*, Springer.
- Pinheiro, J. C. & Bates, D. M. (2000), *Mixed-Effects Models in S and S-PLUS*, Statistics and Computing, Springer.
- Raudenbush, S. W., Yang, M. L. & Yosef, M. (2000), ‘Maximum likelihood for generalized linear models with nested random effects via high-order, multivariate Laplace approximation’, *Journal of Computational and Graphical Statistics* **9**, 141–157.
- Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986), ‘Learning representations by back-propagating errors’, *Nature* **323**, 533–536.
- Ruppert, D., Wand, M. & Carroll, R. (2003), *Semiparametric Regression*, Cambridge University Press.
- Searle, S. R., Casella, G. & McCulloch, C. E. (1992), *Variance Components*, John Wiley & Sons.

- Skaug, H. J. (2002), ‘Automatic differentiation to facilitate maximum likelihood estimation in nonlinear random effects models’, *Journal of Computational and Graphical Statistics* **11**, 458–470.
- Tierney, L. & Kadane, J. B. (1986), ‘Accurate approximations for posterior moments and marginal distributions’, *Journal of the American Statistical Association* **81**, 82–86.
- Wand, M. (2003), ‘Smoothing and mixed models’, *Computational Statistics* **18**, 223–249.

Code	Description	AD-mode	Starting point	Result
1	Code for $g(\mathbf{u}, \boldsymbol{\theta})$	–	$\mathbf{u}$	$g$
2	AD of Code 1	Forward	$\mathbf{u}$	$\nabla_{\mathbf{u}} g$
3	AD of Code 2	Reverse	$\nabla_{\mathbf{u}} g$	$\mathbf{H}$
4	Cholesky algorithm	–	$\mathbf{H}$	$\psi(\mathbf{H})$
5	AD of Code 4	Reverse	$\psi(\mathbf{H})$	$\nabla_{\mathbf{H}} \{\psi(\mathbf{H})\}$
6	AD of Code 2+3	Reverse	$\nabla_{\mathbf{H}} \{\psi(\mathbf{H})\}$	$\nabla_{\mathbf{u}} [\psi \{\mathbf{H}(\mathbf{u})\}]$

Table 1: Summary of the different “codes” used to evaluate the term  $\psi(\mathbf{H}(\mathbf{u}))$  and its derivatives. In the table  $\nabla_{\mathbf{u}}$  and  $\nabla_{\mathbf{H}}$  denote the gradient with respect to the elements of  $\mathbf{u}$  and  $\mathbf{H}$ , respectively. The column “Starting point” shows which variables need to be initialized before execution of the code can start, and “Result” shows which variables the code evaluates.

Model description	Structure <sup>1</sup>	Comput. time		Software
		ADMB	Alt.	
Generalized additive model <sup>2</sup>		165		
Nonparametric est. of variance <sup>3</sup>		32		
Mixed logistic regression		27	700	WinBUGS <sup>4</sup>
Discrete valued time series <sup>5</sup>	Banded	66	3120	Kuk & Cheng (1999)
Ordered categorical responses <sup>6</sup>	B-D	30	10-15	aML <sup>6</sup>
Nonlinear mixed model <sup>7</sup>	B-D	1	2	NLME <sup>7</sup>
Pharmacokinetics model <sup>8</sup>	B-D	17	7	NLME <sup>8</sup>
Weibull regression <sup>9</sup>	B-D	9	7	WinBUGS <sup>9</sup>
GLM with spatial structure <sup>10</sup>		120		
Stochastic volatility model <sup>11</sup>	Banded	22	6171	WinBUGS <sup>11</sup>

NOTES: (1) Directives given to AD Model Builder about the structure of the Hessian matrix  $H$  (see Section 3.4), where B-D means that the Hessian is block diagonal; (2) Data taken from Ruppert et al. (2003, sec. 11.3); (3) Model described in Section 2.1, data taken from Ruppert et al. (2003, sec. 14.3); (4) See Section 4.1; (5) The classical “Polio” dataset (e.g. Kuk & Cheng 1999); (6) The “SOCATT” dataset used in <http://multilevel.ioe.ac.uk/softrev/tables.html>; (7) Timing based on S-Plus code from Pinheiro & Bates (2000, pp. 356-362); (8) Timing based on S-Plus code from Pinheiro & Bates (2000, sec. 6.4); (9) Based on the code and recommendations in the WinBUGS manual; (10) Section 4.2; (11) Timing taken from Meyer & Yu (2000).

Table 2: Computation times (in seconds) for AD Model Builder (ADMB) in comparison with other software packages.