Statistical Applications of the Complex-step Method of Numerical Differentiation

M. S. RIDOUT
University of Kent, Canterbury, Kent, U.K.
email: M.S.Ridout@kent.ac.uk

M.S.Ridout is Reader in Statistics, Institute of Mathematics, Statistics and Actuarial Science, University of Kent, Canterbury, Kent CT2 7NF. U.K.
ABSTRACT
The complex-step method is a clever way of obtaining a numerical approximation to the first derivative of a function, avoiding the round-off error that plagues standard finite difference approximations. An extension of the method allows second derivatives to be calculated with reduced round-off error. This article provides an overview of the method, discusses its practical implementation, with particular reference to R, and studies its effectiveness in several statistical examples.

KEY WORDS: Automatic differentiation; finite difference; gradient; Hessian; Richardson extrapolation
1. INTRODUCTION

In recent years there has been growing interest, particularly in the engineering literature, in a method of numerical differentiation known as the complex-step approximation, which provides an alternative to finite difference approximation. The purpose of this note is to outline the complex-step method and investigate its performance for several statistical examples. Section 2 describes the basic method and Section 3 discusses implementation issues. Section 4 presents a variety of statistical examples. Other approaches to differentiation are considered briefly in the Discussion.

2. THE COMPLEX-STEP METHOD

Suppose that we are interested in the first and second derivatives of the function $f(\theta)$, where $\theta$ is a $p$-dimensional vector of real values. In statistical applications, $f(\theta)$ will often be a log-likelihood function. Although our interest is in real values of $\theta$, we sometimes treat $\theta$ as a vector of complex values in the sequel and assume that $f$ is analytic at $\theta$.

2.1 First derivatives

Suppose initially that $\theta$ is a scalar parameter. Two commonly used finite difference approximations to $f'(\theta)$ are

$$g_1(\theta) = \frac{f(\theta + \delta) - f(\theta)}{\delta}$$

and

$$g_2(\theta) = \frac{f(\theta + \delta) - f(\theta - \delta)}{2\delta}.$$  

There are two sources of error in using finite difference approximations. The truncation error is the amount by which the approximation, when calculated
exactly, differs from the true value. Simple manipulation of Taylor series shows that this error is $O(\delta)$ for $g_1(\theta)$ and $O(\delta^2)$ for $g_2(\theta)$.

Truncation error is minimised by choosing $\delta$ as small as possible. However, numerical calculation of the approximations is also subject to round-off error which is potentially severe for small $\delta$, since the calculation then involves the subtraction of two near-equal quantities. The round-off error is $O(\delta^{-1})$ for both of the approximations above. Thus, some compromise value of $\delta$ is needed to balance the truncation and round-off errors. A common suggestion is to take $\delta = \varepsilon_f^{1/2} \min(\theta, \theta_c)$ for $g_1$ and $\delta = \varepsilon_f^{1/3} \min(\theta, \theta_c)$ for $g_2$, where $\varepsilon_f$ is the fractional accuracy to which the function $f$ can be computed and $\theta_c$ is a cut-off value introduced to avoid very small values of $\delta$ when $\theta$ is small (Dennis and Schnabel 1983, sec. 5.4). More generally, if round-off error is $O(\delta^{-m})$ and truncation error is $O(\delta^n)$, with $m, n > 0$, then one should choose the step size to be $\delta = \varepsilon_f^{1/(m+n)} \theta$. In the examples in this paper we choose $\theta_c < \theta$, so that $\delta$ is always proportional to $\theta$, and set $\varepsilon_f$ to be the machine accuracy, $\varepsilon$, defined as the smallest positive number for which $1 + \varepsilon > 1$ within the computer.

The complex-step approximation (Squire and Trapp 1998) results from the Taylor series expansion

$$f(\theta + i\delta) = f(\theta) + i\delta f'(\theta) - \frac{\delta^2 f''(\theta)}{2!} - \frac{i\delta^3 f'''(\theta)}{3!} + \ldots, \quad (3)$$

where $i = \sqrt{-1}$. Hence

$$\frac{\text{Im}[f(\theta + i\delta)]}{\delta} = f'(\theta) - \frac{\delta^2 f''(\theta)}{3!} + \ldots, \quad (4)$$

where $\text{Im}(z)$ denotes the imaginary part of the complex number $z$. So

$$g_4(\theta) = \frac{\text{Im}[f(\theta + i\delta)]}{\delta}. \quad (5)$$
is another approximation to $f'(\theta)$ with $O(\delta^2)$ truncation error. In fact, the leading error terms of $g_2(\theta)$ and $g_4(\theta)$ are of equal magnitude, but opposite sign. However, the key point about $g_4(\theta)$ is that in most circumstances the truncation error can be eliminated almost completely by choosing a very small value of $h$, without fear of round-off error due to subtractive cancellation; exceptions to this are discussed in Section 3.

To illustrate the complex-step approach, consider the log-likelihood function for a single observation $y$ from a Poisson distribution with mean $\theta$,

$$f(\theta) = y \log(\theta) - \theta.$$

Here and elsewhere in the paper we exclude constant terms from the log-likelihood function. For illustration, let $y = 4$ and $\theta = 5$, so that $f'(\theta) = 1/5$. Figure 1 shows the absolute error of the approximations $g_1$, $g_2$ and $g_4$. The calculations were done in R with a machine accuracy, given by the system variable \texttt{Machine$\cdot$double.eps}, of $2.22 \times 10^{-16}$. Initially, as $\delta$ decreases, the absolute errors decrease as truncation error declines. The rates of decline are essentially equal for $g_2$ and $g_4$, but $g_1$ declines more slowly, reflecting its inferior $O(h)$ truncation error. As $\delta$ decreases further, however, round-off error becomes important and causes the absolute errors of $g_1$ and $g_2$ to start increasing again, at essentially the same rate. Eventually, as $\delta$ approaches machine accuracy, these approximations return a value of zero. However, the absolute error of $g_4$ continues to decrease until it is around the machine accuracy and this accuracy is maintained for values of $\delta$ well below machine accuracy.

Note that the use of complex values of $\theta$ in this example is purely an artifice to get good numerical results; it makes no statistical sense to consider
a Poisson distribution whose mean is a complex number.

If $\theta$ is a vector of dimension $p$, the gradient vector may be approximated by applying any of the approximations above to each element of $\theta$ in turn. For example, the complex-step approximation to $\partial f / \partial \theta_j$ is

$$g_{4,j}(\theta) = \frac{\text{Im}[f(\theta + \delta_j e_j)]}{\delta_j},$$

where $e_j$ is the vector with $j$th element equal to one and other elements equal to zero, and where the step size $\delta_j$ may vary with $j$. For the approximations $g_1$, $g_2$ and $g_4$, calculation of the complete gradient vector requires respectively $p + 1$, $2p$ and $p$ function evaluations. However, the evaluations for $g_4$ will generally be slower, because they involve complex values of the argument.

### 2.2 Second derivatives

Three commonly used finite difference approximations to the partial derivative $\partial^2 f / \partial \theta_j \theta_k$ are

$$h_{1,j,k}(\theta) = \frac{1}{\delta_j \delta_k} \left\{ \frac{f(\theta + \delta_j e_j + \delta_k e_k) - f(\theta + \delta_j e_j)}{\delta_k} - \frac{f(\theta + \delta_k e_k)}{\delta_k} \right\},$$

$$h_{2,j,k}(\theta) = \frac{1}{2\delta_j \delta_k} \left\{ \frac{f(\theta + \delta_j e_j + \delta_k e_k) - f(\theta + \delta_j e_j)}{\delta_k} + \frac{f(\theta + \delta_k e_k)}{\delta_k} \right\} +$$

$$\frac{f(\theta - \delta_j e_j - \delta_k e_k) - f(\theta - \delta_j e_j)}{\delta_k} -$$

$$\frac{f(\theta - \delta_k e_k) - f(\theta)}{\delta_k} \right\},$$

and

$$h_{3,j,k}(\theta) = \frac{1}{4\delta_j \delta_k} \left\{ \frac{f(\theta + \delta_j e_j + \delta_k e_k) - f(\theta + \delta_j e_j - \delta_k e_k)}{\delta_k} -$$

$$\frac{f(\theta - \delta_j e_j + \delta_k e_k) - f(\theta - \delta_j e_j - \delta_k e_k)}{\delta_k} \right\},$$

where $e_j$ is the vector with $j$th element equal to one and other elements equal to zero, and where the step size $\delta_j$ may vary with $j$. For the approximations $g_1$, $g_2$ and $g_4$, calculation of the complete gradient vector requires respectively $p + 1$, $2p$ and $p$ function evaluations. However, the evaluations for $g_4$ will generally be slower, because they involve complex values of the argument.
The first two formulae are given, for example, by Monahan (2001, sec. 8.6), who discusses the bracketing of terms to reduce round-off error. The third formula is used, for example, by the `optim` function in R, with $\delta_j = \delta_k = 0.001$ by default.

There appears to be no way of avoiding subtraction for approximating second derivatives, even if complex steps are allowed. However, one simple possibility is to apply the finite difference approximation $f_2$ to the first derivative approximation based on $f_4$. This leads to the formula

$$h_{4,j,k}(\theta) = \frac{1}{2\delta_j\delta_k} \text{Im}[f(\theta + i\delta_j e_j + \delta_k e_k) - f(\theta + i\delta_j e_j - \delta_k e_k)]. \quad (10)$$

This approximation has been suggested independently by several authors. Abokhodair (2007), provides MATLAB code and recommends choosing $\delta_j \ll \delta_k$, on the grounds that $f_2$ is susceptible to round-off error but $f_4$ is not. Cai (2008) gives similar Matlab code, but with $\delta_j = \delta_k = \sqrt{\varepsilon}$, where $\varepsilon$ is the machine accuracy. Lai and Crassidis (2007) obtain an equivalent approximation by a somewhat different route, implicitly taking $\delta_j = \delta_k$.

Table 1 gives the leading term of the truncation error for pure and mixed second derivatives for each approximation. Inevitably, comparisons between approximations depend on the magnitude of the various derivatives involved. However, for pure second derivatives, the truncation error will typically be smallest for $h_4$, unless the magnitude of the 6th derivative is much greater than that of the 4th derivative. As well as having a better truncation error, $h_4$ also has an improved round-off error which is $O(h^{-1})$ compared to the $O(h^{-2})$ of other methods. Thus the ‘optimal’ step size is $\varepsilon^{1/3}\theta$ for $h_1$, $\varepsilon^{1/4}\theta$ for $h_2$ and $h_3$, and $\varepsilon^{1/5}\theta$ for $h_4$.

As an example, consider again the Poisson log-likelihood function given
by equation (6), for which
\[ f^{(k)}(\theta) = (-1)^{k-1}(k-1)!y/\theta^k \quad k \geq 1. \]

For \( y = 4 \) and \( \mu = 5 \), the magnitudes of the leading truncation error terms are
0.0213\( \delta \) for \( h_1 \), 0.0224\( \delta^2 \) for \( h_2 \), 0.0128\( \delta^2 \) for \( h_3 \) and 0.00034\( \delta^4 \) for \( h_4 \). Thus the truncation error of \( h_4 \) is many orders of magnitude smaller than that of the other approximations. Figure 2 shows the absolute errors in approximating the \( f''(\theta) \). For optimally chosen step sizes, \( h_4 \) performs much better than \( h_2 \) and \( h_3 \), which in turn perform much better than \( h_1 \). Even if the step size for \( h_4 \) is not chosen optimally, this approximation performs better than the other methods over quite a wide range of values of \( \delta \).

For pure second derivatives, a generalization of \( h_4 \) that might be considered is
\[ h^*_4,(\theta) = \frac{1}{2\delta_j \delta_j} \text{Im}[f(\theta + i\delta_j e_j + \delta_j \delta_j e_j) - f(\theta + i\delta_j e_j - \delta_j \delta_j e_j)], \]
with \( \delta_j \) very small, since it contributes only to truncation error, and \( \delta_j \) not too small, as the round-off error is \( O(\delta_j^{-1}) \) (Abokhodair, 2007). However, if \( \delta_j \neq \delta_j \), the truncation error is increased by the presence of a term proportional to \( (\delta_j^2 - \delta_j^2) \); only when \( \delta_j = \delta_j = \delta_j \) is the truncation error \( O(\delta_j^4) \).

For mixed derivatives, the situation is more complicated. However, with the simplifying assumption that \( \delta_j = \delta_k = \delta_j k \), the error terms remain unchanged except that the truncation error for \( h_4 \) is now \( O(\delta_j^2) \). With a common value of \( \delta_j k \), a reasonable choice for this step size is
\[ \delta_j k = \varepsilon^{1/s} \sqrt{\theta_j \theta_k} \]
with \( s = 3 \) for \( h_1 \) and \( h_4 \) and \( s = 4 \) for \( h_2 \) and \( h_3 \).
To illustrate a mixed derivative calculation we consider the log-likelihood for observations $y_1, y_2$ from Poisson distributions with mean $\mu_1, \mu_2$, where $
olimits \mu_j = \theta_1 + \theta_2 x_j$,

$$f(\theta_1, \theta_2) = y_1 \log(\theta_1 + \theta_2 x_1) + y_2 \log(\theta_1 + \theta_2 x_2) - \theta_2 (x_1 + x_2). \tag{11}$$

Figure 3 compares the absolute errors of the different approximations when $y_1 = 4, y_2 = 5, x_1 = 2, x_2 = 4, \theta_1 = 3$ and $\theta_2 = 1$. For these parameter values, the true value of the derivative is $-892/1225$. Despite its increased truncation error, $h_4$ still outperforms the other estimators as a result of its better round-off error.

Table 1 also shows the number of function evaluations needed to calculate the full Hessian matrix of second derivatives. We have assumed here that only the upper (or lower) triangular half of the matrix is calculated, with the remainder determined by symmetry. The approximations $h_1, h_2$ and $h_3$ are symmetrical in $\theta_j$ and $\theta_k$, though it is still possible that the approximations to $\partial^2 f/\partial \theta_j \partial \theta_k$ and $\partial^2 f/\partial \theta_k \partial \theta_j$ will differ slightly due to round-off error. However, in general, $h_{4,j,k}(\theta) \neq h_{4,k,j}(\theta)$ and an alternative is to calculate all elements of the Hessian matrix $H$ and then convert this to the symmetric matrix $(H + H^t)/2$, where $H^t$ denotes the transpose matrix. However, this increases the number of function evaluations to $2p^2$ and often provides little improvement.

In some instances, approximations to both the first and second derivatives of the function are required. If equation (10) is used to approximate the second derivatives, then the first derivative with respect to $\theta_j$ may be approximated without any additional function evaluations as

$$\frac{1}{2\delta_j} \text{Im} \left[ f(\theta + \delta_j(1 + i)e_j) + f(\theta + \delta_j(-1 + i)e_j) \right].$$
Like equation (5), this approximation has $O(\delta_j^2)$ truncation error and does not suffer from round-off error, though as already noted, because the second derivative approximation is also needed, the value of $\delta_j$ cannot be taken to be too small.

3. IMPLEMENTATION ISSUES

The formulae of the previous section can be used directly in any programming environment that supports complex numbers, for example MATLAB, R or S-PLUS. This will suffice for many statistical applications, but sometimes there are additional implementation issues that must be addressed to ensure good results. Martins, Sturdza and Alonso (2003) discuss these issues, focusing particularly on implementation in Fortran or C and Shampine (2007) describes in detail a MATLAB package, PMAD, for complex-step first derivatives. This is implemented at two levels. The first introduces special versions of certain operators and functions so that the complex-step method works correctly. This is termed the informal approach because it relies on these modified functions and operators being used in the coding of the function $f(\theta)$. The second approach is a slower, but more reliable, object-oriented implementation.

Our aim here is to raise awareness of the issues that any potential user of the complex-step method should be aware of. Techniques for resolving these difficulties depend on the specific programming environment and for illustration we indicate how some of the issues may be resolved in R. What we propose constitutes an informal approach that will extend the range of statistical problems to which the complex-step method may be applied. Whilst
a more formal approach might be conceivable in \( R \), it seems unlikely that this would be worth the considerable programming effort required.

The first point is that the complex-step approximation may fail for some functions, if the step size is too small, because of the way that the software evaluates the function for complex arguments (Martins et al. 2003). In \( R \), this applies to the inverse trigonometric functions, for example. The method also fails for the \texttt{abs} function, since this always returns a real value. Other functions, such as \texttt{lgamma}, which returns the logarithm of the gamma function, only accept real arguments.

Often, these problems can be resolved by defining or redefining the way in which these functions behave for complex arguments (Martins et al. 2003). A trick that works for first derivatives is to define

\[
f(\theta + \delta i) = f(\theta) + i\delta f'(\theta),
\]

(12)

where \( \theta \) and \( \delta \) are real. Then it is easy to see that the complex-step method will return the \textit{exact} derivative \( f'(\theta) \), to within the accuracy with which this can be evaluated, irrespective of the value of \( \delta \). It requires us to know the derivative of the function, which might at first sight seem to defeat the object of using numerical differentiation. But the point is that once the function is coded, it can be used as a component of more complicated functions.

Unfortunately, this is just a computational trick. It does \textit{not} properly define the function for complex arguments in the mathematical sense, but simply uses the real and imaginary parts of complex numbers to store both the function and its derivative. As a result, it cannot be used when we require second derivatives, as will usually be the case in statistical applications. Instead, the function definition must be extended to complex arguments in
the proper mathematical sense.

To illustrate how a function may be extended to accept complex arguments in R, consider the function \texttt{lgamma}. Fortunately, there is a function \texttt{cgamma} in the package \texttt{fOptions}, which returns the gamma function for complex arguments. We may then extend the \texttt{lgamma} function to accept complex arguments using the following commands:

\begin{verbatim}
lgamma.complex <- function(z) log(cgamma(z))
lgamma <- function(z) UseMethod("lgamma")
lgamma.default <- base::lgamma
\end{verbatim}

With this redefinition, the complex-step method can be applied without modification to functions that use \texttt{lgamma}; Section 4 provides an example.

The Poisson log-likelihood function of equation (6) can be evaluated in R as \texttt{dpois(y, theta, log=TRUE)}, though this does include the constant term that was omitted from equation (6). A slightly more complicated approach is needed here, because it is the second argument of this function that we wish to allow to be complex. Figure 4 shows one possible approach. A disadvantage of redefining functions in this way is that inevitably it adds overhead. However, Table 2 indicates that in this instance the overhead is modest in comparison with the overhead of using the built-in function \texttt{dpois} instead of programming the function directly.

The usage of operators should also be considered. The standard arithmetic operators work for complex numbers. However, integer powers should be evaluated by direct multiplication to avoid problems with the complex-step approximations. For example, if \( \delta \) is too small, the approximation \( g_4(\theta) \) gives poor results for the function \( f(\theta) = (y - \theta)^2 \) when this is eval-
uated as $(y-theta)^2$ but accurate results when it is evaluated instead as $(y-theta)*(y-theta)$.

Relational operators such as $<$ and $>$ are not defined for complex numbers and will generate an error. The appropriate corrective action is to modify the code defining $f(\theta)$ so that it is the real parts that are compared, for example, modifying an expression such as

$$\text{theta}[1] < \text{theta}[2]$$

to

$$\text{Re}(\text{theta}[1]) < \text{Re}(\text{theta}[2])$$

Particular care should be taken with the operators $==$ and $!=$, representing logical equality and non-equality, since these are defined for complex numbers and so will not generate an error message. Nonetheless, comparisons should be modified to apply to the real parts of the arguments, as above, and failure to do this will lead to incorrect results in most instances.

4. EXAMPLES

To investigate the performance of the complex-step method in practice, we consider a particularly common application of numerical differentiation in statistics, namely numerical calculation of the observed information matrix after fitting a model by maximum likelihood, followed by inversion of the matrix to give the standard errors of the estimated parameters and their correlations. Thus the function $f$ will be a negative log-likelihood function.

We note two features of this problem at the outset. Firstly, we are unlikely to require very high accuracy in this context, because the methodology is based on asymptotic theory that will apply only approximately to the fi-
nite data available. Thus from a purely practical viewpoint, even relatively poor methods may be adequate. Secondly, the asymptotic theory indicates that in regular problems the log-likelihood function should be approximately quadratic in the neighbourhood of its maximum. This means that the numerical differentiation problem is relatively benign, since if the function were exactly quadratic the truncation error would be zero for all of the second derivative approximations that we have described. This may limit the potential for improvement by the complex-step method.

We consider several examples from the book by Brazzale, Davison and Reid (2007), who give references to the original data sources. The focus of this book is on improving inference by using higher-order asymptotic theory, but here we are simply looking at the numerical aspects of standard first-order theory. We use the notation BDR 4.3, for example, to refer to Section 4.3 of Brazzale et al. (2007). Brief details of the examples are as follows.

**Gamma data** (BRD 2.3)
This artificial example involves five observations, 0.2, 0.45, 0.78, 1.28, 2.28, assumed to be from the gamma density

\[ f(y; \lambda, \psi) = \frac{\lambda^\psi y^{\psi-1}}{\Gamma(\psi)} \exp(-\lambda y), \quad y > 0, \lambda, \psi > 0. \]

We include this example to illustrate the use of the redefined log-gamma function.

**Smoking data** (BDR 4.6)
The data are the number of British male doctors dying of lung cancer \((Y)\) and the man-years at risk \((T)\) cross-classified by number of cigarettes smoked per day \((c, 7 \text{ levels})\) and years of smoking \((d, 9 \text{ levels})\). \(Y\) is modelled as a
Poisson variable with mean

\[ \mu(\theta) = 10^{-5} T \times e^{\theta_1} d^{\theta_2} \left( 1 + e^{\theta_3} e^{\theta_4} \right). \]

Radioimmunoassay data (BDR 5.4)

This is a nonlinear regression problem. There are two replicates of each of eight levels of drug concentration \((x)\). The response variable is the percentage of radioactive gamma counts, assumed to be normally distributed with mean

\[ \mu(x; \beta) = \beta_1 + \frac{\beta_2 - \beta_1}{1 + (x/\beta_4)^{\beta_3}}, \quad x \geq 0, \beta_1, \ldots, \beta_4 \geq 0 \]

and variance

\[ \sigma^2(x; \beta, \gamma) = e^{\gamma_1} \mu(x; \beta)^{\gamma_2}. \]

We take \( \theta \) to be the full vector of parameters \((\beta_1, \beta_2, \beta_3, \beta_4, \gamma_1, \gamma_2)\).

Nuclear power station data (BDR 5.2)

This is a multiple linear regression problem, with 32 observations. The response variable is the logarithm of the construction cost of a nuclear reactor and there are ten potential explanatory variables. We consider the particular model in Table 5.2 of Brazzale et al. (2007), which includes a constant term and six covariates and was selected using AIC. The model is fitted by ordinary least squares and also by maximum likelihood assuming that the errors follow a scaled \( t_4 \) distribution.

Cell phone data (BDR 4.3)

The data are from a study of the association between cellular telephone use and vehicle collisions. The single parameter of interest, \( \exp(\psi) \), is an approximate odds-ratio. The (conditional) likelihood for this parameter is that of a binomial random variable with index 181, observed value 157 and probability \( \gamma \), where \( \gamma = e^{\psi}/(1 + e^{\psi}) \).
We use this example to illustrate numerical implementation of the delta method. That is, we write the log-likelihood in terms of \( \gamma \) and hence obtain a numerical estimate of \( \text{var}(\hat{\gamma}) \). Then we obtain the variance of the maximum likelihood estimator of the transformed parameter \( f(\gamma) = \gamma/(1 - \gamma) \) using the delta method. This requires calculation of the first derivative \( f'(\gamma) \) for which we use the approximation \( g_4 \) when the Hessian matrix is approximated by \( h_4 \) and the approximation \( g_2 \) for all other Hessian approximations.

*Therapy cost data* (BDR 3.5)
The data comprise costs (pounds sterling) of therapy for patients with a history of deliberate self harm. Patients received either a standard therapy or cognitive behaviour therapy. The parameter of interest is either the difference or the ratio of the group means and the costs are assumed to follow either exponential or log-normal distributions.

We use this as a further illustration of the use of the delta method. The log-likelihood for the exponential distribution is written in terms of the group mean parameters \( \mu_1 \) and \( \mu_2 \). In the log-normal case, the log-likelihood is written in terms of parameters \( \lambda_i \) and \( \sigma_i^2 \) \((i = 1, 2)\), with the group means given by \( \exp(\lambda_i + \sigma_i^2/2) \). In both cases, the covariance matrix of the maximum likelihood estimators of the defining parameters is obtained by inverting a numerical approximation to the Hessian matrix. Then the delta method is used to find the \( 2 \times 2 \) covariance matrix of the transformed parameters \( (\mu_1/\mu_2, \mu_1 - \mu_2) \), approximating the required first derivatives as in the previous example.

### 4.1 Accuracy of approximations

To investigate the accuracy of the approximations, MAPLE was used to
obtain the exact derivatives symbolically and evaluate them to high precision for a particular set of parameter values. The results were then compared with the various approximations evaluated at the same parameter values. In addition to the approximations to the Hessian matrix defined in Section 2, we also approximated the observed information matrix using the function hessian in the numDeriv package. This starts with a finite difference approximation and improves it, at the expense of further function evaluations, by the technique of Richardson extrapolation Press, Teukolsky, Vetterling and Flannery 1992, sec. 5.7). The hessian function has several controlling parameters; these were left at their default settings, under which the total number of function evaluations required is $4p^2 + 4p + 2$. The R programs used to generate all of the results presented here are available at web address removed for blinding.

We evaluated the approximations using three measures - the relative error of the generalised variance (the determinant of the inverse of the observed information matrix), the maximum relative error of the standard errors of the parameter estimates and the maximum absolute error of the correlations between parameter estimates. Since all three measures gave a similar picture of the relative performance of the approximations, we present just the second of these here (Table 3).

Based on the orders of their truncation and round-off errors, one would expect to see progressive improvement in the approximations from $h_1$ through to $h_4$, though this general behavior will be distorted for some specific functions. Leaving aside the BDR 5.2 example for a moment, this pattern is borne out generally in Table 3. Differences between $h_2$ and $h_3$ are usually small, but these can reduce relative errors by two or more orders of magnitude.
compared to $h_1$. The complex-step approximation $h_4$ can offer a considerable further improvement; the improvement is particularly marked in the last three rows of Table 3, which involve the delta method, where the complex-step approach offers improvements to both the first and second derivatives that are required.

The performance of the hessian function is often similar to that of $h_4$, but it performs poorly for BDR 4.6 and BDR 4.3. In these examples, the errors can be reduced by increasing the number of iterations of the Richardson extrapolation procedure, but this increases the computational cost.

For the least squares fit to BDR 5.2, $h_4$ has slightly larger approximation error than $h_3$. This is due to the absence of truncation error. In the absence of truncation error, it is best to take the step sizes $\delta_j$ to be as large as possible, to minimize round-off error but, as discussed in Section 2, the step sizes used for $h_3$ and $h_4$ are different and favour $h_3$. If a step size of $\delta_j = \varepsilon^{1/4} \theta_j$ is used for $h_4$ as well as $h_3$ then, as in other examples, the error is smaller for $h_4$.

When the errors in example BDR 5.2 are assumed to follow a scaled $t_4$ distribution, none of the approximations work well (Table 3) and the approximation $h_2$ gives a matrix that is not even positive definite. There are several factors to consider here. The first is that the true Hessian matrix, $H$, is poorly conditioned; its largest element is $H_{2,2} = 8761818$ and its smallest elements are $H_{4,7} = 0$ and $H_{7,8} = 0.692$. Moreover, the 4th and 6th order derivatives of the negative log-likelihood function with respect to $\theta_2$ are very large and consequently the approximations have large truncation errors; for example, the absolute errors of $h_3$ and $h_4$ for approximating $H_{2,2}$ are $-408.99$ and $-7.91$ respectively. However, it remains true that all elements of the Hessian matrix are approximated more accurately by $h_4$ than
by \( h_3 \). But when the Hessian matrix is inverted, the approximation errors happen to combine in such a way that \( h_3 \) leads to a better approximation of the inverse matrix than \( h_4 \). Whilst one might expect that this would be a rather unusual phenomenon, this example shows that it can occur.

4.2 Computational times

Again we focus on the inverse of the Hessian matrix, since this is usually the quantity of most direct interest in statistical applications. Thus the computing times include the time needed for matrix inversion. We present timings relative to the time taken to compute \( h_3 \), which is the most accurate of the three finite difference approximations discussed in Section 2.

Generally, computational times increase through \( h_1, h_2, h_3 \) and \texttt{hessian}, reflecting the increasing numbers of function evaluations required, though the details depend on the specific function involved. The approximation \( h_4 \) requires only half as many function evaluations as \( h_3 \), but these involve complex arguments of the function. The impact of this on overall computational time depends strongly on the function being evaluated and as a result \( h_4 \) may be faster or slower than \( h_3 \). The heaviest computational cost is for BRD 2.3, which involves the \texttt{lgamma} function, since evaluating the gamma function with complex argument is 6–7 times slower than evaluating the function with real argument. Nonetheless, computational times for \( h_4 \) are always markedly less than those for the comparably accurate \texttt{hessian}.

5. DISCUSSION

Generally, either the complex-step approximation \( h_4 \) or the Richardson extrapolation scheme implemented in the \texttt{hessian} function gave the most ac-
curate results. The hessian function was considerably slower and gave poor results in some examples. Richardson extrapolation can also be implemented in the complex-step framework to reduce truncation errors (Lai and Crassidis 2007), but we have not investigated this.

The approximation $h_3$, though less accurate than $h_4$ was the best of the three finite difference approximations considered and would be adequate in practice for any of the examples considered. It was also the only method that performed at all well for the BDR 5.2 example with $t_4$ errors, though as discussed in Section 4, this is a fortuitous result of inverting the approximated Hessian matrix, and even in this example the Hessian matrix itself was more accurately approximated by $h_4$. However, the fundamental message of that example is that all of the numerical differentiation methods considered can fail if the function of interest has high-order derivatives of very large magnitude, because of large truncation errors.

The complex-step method is certainly worth considering if if high accuracy is required. As emphasised in Section 3, some caution is needed in implementing the method, though the only issues that arose in the examples of Section 4 were the need to redefine the \texttt{lgamma} function and calculate squared terms by multiplication rather than exponentiation.

This paper has focused on numerical differentiation. Two other approaches to differentiation are symbolic differentiation and algorithmic differentiation. Symbolic algebra systems such as \texttt{MAPLE} allow one to differentiate functions symbolically and then generate code automatically in various languages to evaluate the resulting derivatives. Algorithmic differentiation (Griewank, 2000) is based on the idea that computer code for evaluating any function can be broken down into a series of elementary codes that may
be differentiated individually, by simple look-up, and then combined using the chain rule. This enables derivatives to be calculated as accurately as by symbolic differentiation; see Skaug and Fournier (2006) for a statistical application. The complex-step approximation used in conjunction with equation (12) is closely linked to what is known as forward-mode algorithmic differentiation; see Martins et al. (2003) for details. One advantage of the complex-step method is that it is easy to implement, at least following the informal approach of Section 3, as the basic algorithm is similar to finite difference methods. However, a proper comparison between automatic differentiation and the complex-step method is beyond the scope of this article.

REFERENCES


Griewank, A. (2000), Evaluating Derivatives: Principles and Techniques of
Algorithmic Differentiation, Philadelphia: SIAM.


Table 1. Leading terms of the truncation errors of different approximations to the second derivative, $\partial^2 f / \partial \theta_j \partial \theta_k$, and the number of function evaluations required to approximate the entire Hessian matrix assuming that symmetry is exploited. $D_r$ and $D_{r,s}$ denote respectively the partial derivatives $\partial^r f / \partial \theta_r^j$ and $\partial^{r+s} f / \partial \theta_r^j \theta_s^k$ evaluated at $\theta$.

<table>
<thead>
<tr>
<th>Approx.</th>
<th>Leading term in truncation error</th>
<th>Evaluations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_{1,j,k}(\theta)$</td>
<td>$\delta D_3$</td>
<td>$\frac{1}{2} (\delta_j D_{2,1} + \delta_k D_{1,2})$</td>
</tr>
<tr>
<td>$h_{2,j,k}(\theta)$</td>
<td>$\frac{7}{12} \delta^2 D_4$</td>
<td>$\frac{1}{12} (2\delta_j^2 D_{3,1} + 3\delta_j \delta_k D_{2,2} + 2\delta_k^2 D_{1,3})$</td>
</tr>
<tr>
<td>$h_{3,j,k}(\theta)$</td>
<td>$\frac{1}{3} \delta^2 D_4$</td>
<td>$\frac{1}{6} (\delta_j^2 D_{3,1} + \delta_k^2 D_{1,3})$</td>
</tr>
<tr>
<td>$h_{4,j,k}(\theta)$</td>
<td>$\frac{1}{90} \delta^4 D_6$</td>
<td>$-\frac{1}{6} (\delta_j^2 D_{3,1} - \delta_k^2 D_{1,3})$</td>
</tr>
</tbody>
</table>

Table 2. CPU times (seconds) for 1,000,000 evaluations of the log-likelihood function for a single Poisson observation by programming the function directly or by using the function `dpois`.

<table>
<thead>
<tr>
<th>Calculation method</th>
<th>Approximation</th>
<th>$g_1(\theta)$</th>
<th>$g_2(\theta)$</th>
<th>$g_4(\theta)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equation (6)</td>
<td></td>
<td>0.30</td>
<td>0.55</td>
<td>1.06</td>
</tr>
<tr>
<td>Standard <code>dpois</code></td>
<td></td>
<td>0.90</td>
<td>1.79</td>
<td>–</td>
</tr>
<tr>
<td>Extended <code>dpois</code></td>
<td></td>
<td>1.13</td>
<td>2.02</td>
<td>1.58</td>
</tr>
</tbody>
</table>
Table 3. Numerical errors in approximating the standard errors of the parameter estimates. The measure of error is the maximum over the different parameters of $\log_{10}(\{approxSE - trueSE\}/trueSE)$. The notation 2(2) in the final line indicates that there are two parameters of interest and two nuisance parameters whose standard errors are not considered.

<table>
<thead>
<tr>
<th>Example</th>
<th>Parameters</th>
<th>$h_1(\theta)$</th>
<th>$h_2(\theta)$</th>
<th>$h_3(\theta)$</th>
<th>$h_4(\theta)$</th>
<th>hessian</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDR 2.3</td>
<td>2</td>
<td>-4.49</td>
<td>-6.90</td>
<td>-7.11</td>
<td>-10.47</td>
<td>-11.53</td>
</tr>
<tr>
<td>BDR 4.6</td>
<td>4</td>
<td>-3.40</td>
<td>-3.31</td>
<td>-5.74</td>
<td>-7.34</td>
<td>-3.12</td>
</tr>
<tr>
<td>BDR 5.4</td>
<td>6</td>
<td>-3.66</td>
<td>-5.94</td>
<td>-6.06</td>
<td>-8.44</td>
<td>-8.11</td>
</tr>
<tr>
<td>BDR 5.2 LS</td>
<td>7</td>
<td>-4.36</td>
<td>-7.00</td>
<td>-7.93</td>
<td>-7.07</td>
<td>-10.77</td>
</tr>
<tr>
<td>BDR 5.2 $t_4$</td>
<td>8</td>
<td>-1.59</td>
<td>–</td>
<td>-3.67</td>
<td>-1.79</td>
<td>-0.01</td>
</tr>
<tr>
<td>BDR 4.3</td>
<td>1</td>
<td>-4.47</td>
<td>-6.01</td>
<td>-6.26</td>
<td>-9.50</td>
<td>-5.90</td>
</tr>
<tr>
<td>BDR 3.5 EXP</td>
<td>2</td>
<td>-4.88</td>
<td>-7.08</td>
<td>-7.31</td>
<td>-12.02</td>
<td>-10.68</td>
</tr>
<tr>
<td>BDR 3.5 LN</td>
<td>2(2)</td>
<td>-5.74</td>
<td>-7.53</td>
<td>-7.54</td>
<td>-11.17</td>
<td>-11.49</td>
</tr>
</tbody>
</table>
Table 4. CPU times of different approximations relative to the CPU time of $h_3$.

<table>
<thead>
<tr>
<th>Example</th>
<th>Parameters</th>
<th>$h_1(\theta)$</th>
<th>$h_2(\theta)$</th>
<th>$h_3(\theta)$</th>
<th>$h_4(\theta)$</th>
<th>hessian</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDR 2.3</td>
<td>2</td>
<td>0.69</td>
<td>1.00</td>
<td>1.00</td>
<td>2.14</td>
<td>3.57</td>
</tr>
<tr>
<td>BDR 4.6</td>
<td>4</td>
<td>0.42</td>
<td>0.75</td>
<td>1.00</td>
<td>1.38</td>
<td>2.80</td>
</tr>
<tr>
<td>BDR 5.4</td>
<td>6</td>
<td>0.41</td>
<td>0.75</td>
<td>1.00</td>
<td>1.74</td>
<td>3.23</td>
</tr>
<tr>
<td>BDR 5.2 LS</td>
<td>7</td>
<td>0.34</td>
<td>0.72</td>
<td>1.00</td>
<td>0.36</td>
<td>2.49</td>
</tr>
<tr>
<td>BDR 5.2 $t_4$</td>
<td>8</td>
<td>0.33</td>
<td>0.63</td>
<td>1.00</td>
<td>0.40</td>
<td>2.35</td>
</tr>
<tr>
<td>BDR 4.3</td>
<td>1</td>
<td>0.99</td>
<td>1.04</td>
<td>1.00</td>
<td>0.95</td>
<td>1.63</td>
</tr>
<tr>
<td>BDR 3.5 EXP</td>
<td>2</td>
<td>0.96</td>
<td>1.09</td>
<td>1.00</td>
<td>0.91</td>
<td>1.96</td>
</tr>
<tr>
<td>BDR 3.5 LN</td>
<td>2(2)</td>
<td>0.60</td>
<td>0.84</td>
<td>1.00</td>
<td>0.76</td>
<td>2.38</td>
</tr>
</tbody>
</table>
Figure Captions

Figure 1. Absolute errors of the approximations $g_1(\theta)$ (upper black line), $g_2(\theta)$ (grey line) and $g_4(\theta)$ (lower black line) to the first derivative of the function $f(\theta) = 4 \log(\theta) - \theta$ at $\theta = 5$. Towards the right hand of the $x$-axis, the absolute error for $g_2(\theta)$ coincides with that of $g_4(\theta)$ and is not visible on the plot. The horizontal and vertical grey dashed lines indicate the machine accuracy. The triangular symbols indicate the ‘optimal’ choices of $\delta$ for $g_1(\theta)$ and $g_2(\theta)$ (see text).

Figure 2. Absolute errors of the approximations $h_1(\theta)$ (upper black line), $h_2(\theta)$ (grey line) and $h_3(\theta)$ (middle black line) and $h_4(\theta)$ (lower black line) to the second derivative of the function $f(\theta) = 4 \log(\theta) - \theta$ at $\theta = 5$. The horizontal and vertical grey dashed lines indicate the machine accuracy. The triangular symbols indicate the ‘optimal’ choices of $\delta$ for the different estimators (see text).

Figure 3. Absolute errors of the approximations $h_1(\theta)$ (upper black line), $h_2(\theta)$ (grey line) and $h_3(\theta)$ (middle black line) and $h_4(\theta)$ (lower black line) to the partial derivative $\partial^2 f / \partial \theta_1 \partial \theta_2$, where $f$ is given by equation (11). The horizontal and vertical grey dashed lines indicate the machine accuracy. The triangular symbols indicate the ‘optimal’ choices of $\delta$ for the different estimators (see text).

Figure 4. R code redefining the function `dpois` so that it can be used with the complex-step method.
Figure 1:
Figure 2:
Figure 3:
Figure 4:

dpois.new <- function(x, lambda, log=FALSE) {
  rlambda <- Re(lambda)
  dpois.val <- stats::dpois(x, rlambda, log)
  if (is.complex(lambda)) dpois.val <-
      dpois.val + 1i * Im(lambda) * 
      ifelse(log, x/rlambda-rlambda, (x/rlambda-1) * 
        exp(-rlambda + x*log(rlambda) - lgamma(x+1)))
  dpois.val
}

dpois <- function(x, lambda, log=FALSE) UseMethod("dpois")
dpois.default <- dpois.new