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

2.1 First derivatives

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

$$g_1(\theta) = \frac{f(\theta + \delta) - f(\theta)}{\delta} \tag{1}$$

and

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

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 δ as small as possible. However, numerical calculation of the approximations is also subject to *round-off error* which is potentially severe for small δ, 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 δ 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 ε_f is the fractional accuracy to which the function f can be computed and θ_c is a cut-off value introduced to avoid very small values of δ when θ 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 δ is always proportional to θ, and set ε_f to be the machine accuracy, ε, 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

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

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

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

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

$$g_4(\theta) = \frac{\mathrm{Im}[f(\theta + i\delta)]}{\delta} \tag{5}$$

is another approximation to $f'(\theta)$ with $O(\delta^2)$ truncation error. In fact, the leading error terms of $g_2(\boldsymbol{\theta})$ and $g_4(\boldsymbol{\theta})$ 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 θ,

$$f(\theta) = y \log(\theta) - \theta. \tag{6}$$

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 `.Machine$double.eps`, of 2.22×10^{-16}. Initially, as δ 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 δ 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 δ 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 δ well below machine accuracy.

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

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

where $\boldsymbol{e_j}$ is the vector with jth element equal to one and other elements equal to zero, and where the step size δ_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

$$\begin{aligned} h_{1,j,k}(\boldsymbol{\theta}) = \frac{1}{\delta_j\delta_k}\Big\{ & [f(\boldsymbol{\theta} + \delta_j\boldsymbol{e_j} + \delta_k\boldsymbol{e_k}) - f(\boldsymbol{\theta} + \delta_j\boldsymbol{e_j})] - \\ & [f(\boldsymbol{\theta} + \delta_k\boldsymbol{e_k}) - f(\boldsymbol{\theta})]\Big\}, \end{aligned} \tag{7}$$

$$\begin{aligned} h_{2,j,k}(\theta) = \frac{1}{2\delta_j\delta_k}\Big\{ & [f(\boldsymbol{\theta} + \delta_j\boldsymbol{e_j} + \delta_k\boldsymbol{e_k}) - f(\boldsymbol{\theta} + \delta_j\boldsymbol{e_j})] - \\ & [f(\boldsymbol{\theta} + \delta_k\boldsymbol{e_k}) - f(\boldsymbol{\theta})] + \\ & [f(\boldsymbol{\theta} - h_j\boldsymbol{e_j} - \delta_k\boldsymbol{e_k}) - f(\boldsymbol{\theta} - \delta_j\boldsymbol{e_j})] - \\ & [f(\boldsymbol{\theta} - \delta_k\boldsymbol{e_k}) - f(\boldsymbol{\theta})]\Big\} \end{aligned} \tag{8}$$

and

$$\begin{aligned} h_{3,j,k}(\boldsymbol{\theta}) = \frac{1}{4\delta_j\delta_k}\Big\{ & [f(\boldsymbol{\theta} + \delta_j\boldsymbol{e_j} + \delta_k\boldsymbol{e_k}) - f(\boldsymbol{\theta} + \delta_j\boldsymbol{e_j} - \delta_k\boldsymbol{e_k})] - \\ & [f(\boldsymbol{\theta} - \delta_j\boldsymbol{e_j} + \delta_k\boldsymbol{e_k}) - f(\boldsymbol{\theta} - \delta_j\boldsymbol{e_j} - \delta_k\boldsymbol{e_k})]\Big\}, \end{aligned} \tag{9}$$

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

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

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 ε 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

$$f^{(k)}(\boldsymbol{\theta}) = (-1)^{k-1}(k-1)!y/\theta^k \qquad\qquad k \geq 1.$$

For $y = 4$ and $\mu = 5$, the magnitudes of the leading truncation error terms are 0.0213δ 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 δ.

For pure second derivatives, a generalization of h_4 that might be considered is

$$h^*_{4,j,j}(\boldsymbol{\theta}) = \frac{1}{2\delta_{j_1}\delta_{j_2}}\mathrm{Im}[f(\boldsymbol{\theta} + i\delta_{j_1}\boldsymbol{e_j} + \delta_{j_2}\boldsymbol{e_j}) - f(\boldsymbol{\theta} + i\delta_{j_1}\boldsymbol{e_j} - \delta_{j_2}\boldsymbol{e_j})],$$

with δ_{j_1} very small, since it contributes only to truncation error, and δ_{j_2} not too small, as the round-off error is $O(\delta_{j_2}^{-1})$ (Abokhodair, 2007). However, if $\delta_{j_1} \neq \delta_{j_2}$, the truncation error is increased by the presence of a term proportional to $(\delta_{j_1}^2 - \delta_{j_2}^2)$; only when $\delta_{j_1} = \delta_{j_2} = \delta_j$ is the truncation error $O(\delta_j^4)$.

For mixed derivatives, the situation is more complicated. However, with the simplifying assumption that $\delta_j = \delta_k = \delta_{jk}$, the error terms remain unchanged except that the truncation error for h_4 is now $O(\delta_{jk}^2)$. With a common value of δ_{jk}, a reasonable choice for this step size is

$$\delta_{jk} = \varepsilon^{1/s}\sqrt{\theta_j\theta_k}$$

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 μ_1, μ_2, where $\mu_j = \theta_1 + \theta_2 x_j$,

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

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 θ_j and θ_k, though it is still possible that the approximations to $\partial^2 f/\partial\theta_j\theta_k$ and $\partial^2 f/\partial\theta_k\theta_j$ will differ slightly due to round-off error. However, in general, $h_{4,j,k}(\boldsymbol{\theta}) \neq h_{4,k,j}(\boldsymbol{\theta})$ 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 θ_j may be approximated without any additional function evaluations as

$$\frac{1}{2\delta_j}\mathrm{Im}\left[f(\boldsymbol{\theta} + \delta_j(1+i)\boldsymbol{e_j}) + f(\boldsymbol{\theta} + \delta_j(-1+i)\boldsymbol{e_j})\right].$$

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 δ_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 $f(\boldsymbol{\theta})$. 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 `abs` function, since this always returns a real value. Other functions, such as `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

$$f(\theta + \delta i) = f(\theta) + i\delta f'(\theta), \tag{12}$$

where θ and δ are real. Then it is easy to see that the complex-step method will return the *exact* derivative $f'(\theta)$, to within the accuracy with which this can be evaluated, irrespective of the value of δ. 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 *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 `lgamma`. Fortunately, there is a function `cgamma` in the package `fOptions`, which returns the gamma function for complex arguments. We may then extend the `lgamma` function to accept complex arguments using the following commands:

```
lgamma.complex <- function(z) log(cgamma(z))
lgamma <- function(z) UseMethod("lgamma")
lgamma.default <- base::lgamma
```

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

The Poisson log-likelihood function of equation (6) can be evaluated in R as `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 `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 δ is too small, the approximation $g_4(\boldsymbol{\theta})$ 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 $f(\boldsymbol{\theta})$ so that it is the real parts that are compared, for example, modifying an expression such as

```
theta[1] < theta[2]
```

to

```
Re(theta[1]) < Re(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 differentation 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

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

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 levels) and years of smoking (d, 9 levels). Y is modelled as a

Poisson variable with mean

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

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

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

and variance

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

We take θ 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 follows 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 γ, 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 γ and hence obtain a numerical estimate of $\text{var}(\widehat{\gamma})$. 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 μ_1 and μ_2. In the log-normal case, the log-likelihood is written in terms of parameters λ_i and σ_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 2×2 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 δ_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 θ_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 `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 `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 `hessian`.

5. DISCUSSION

Generally, either the complex-step approximation h_4 or the Richardson extrapolation scheme implemented in the `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 `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 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

Abokhodair, A.A. (2007), "Numerical Tools for Geoscience Computations: Semiautomatic Differentiation – SD," *Computational Geosciences*, 11, 283–296.

Brazzale A.R., Davison, A.C., and Reid, N. (2007), *Applied Asymptotics: Case Studies in Small-Sample Statistics*, Cambridge: Cambridge University Press.

Cai, Y. (2008), "HESSIANCSD: Complex Step Hessian," available at `http://www.mathworks.com/matlabcentral/files/18177/hessiancsd.m`.

Dennis, J.E., and Schnabel, R.J. (1983), *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Englewood Cliffs: Prentice-Hall.

Griewank, A. (2000), *Evaluating Derivatives: Principles and Techniques of*

Algorithmic Differentiation, Philadelphia: SIAM.

Lai, K.-L., and Crassidis, J.L. (2007), "Extensions of the First and Second Complex-Step Derivative Approximations," *Journal of Computational and Applied Mathetatics*, in press, doi:10.1016/j.cam.2007.07.026.

Martins, J.R., Sturdza, P., and Alonso, J.J. (2003), "The Complex-Step Derivative Approximation," *ACM Transactions on Mathematical Software*, 29, 245–262.

Monahan, J.F. (2001), *Numerical Methods of Statistics*, Cambridge: Cambridge University Press:

Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P. (1992), *Numerical Recipes in Fortran* (2nd ed.), Cambridge: Cambridge University Press.

Skaug, H.J., and Fournier, D. A., (2006), "Automatic Approximation of the Marginal Likelihood in Non-Gaussian Hierarchical Models," *Computational Statistics and Data Analysis*, 51, 699–709.

Squire, W. and Trapp, G. (1998), "Using Complex Variables to Estimate Derivatives of Real Functions," *SIAM Review*, 40, 110–112.

Table 1. Leading terms of the truncation errors of different approximations to the second derivative, $\partial^2 f/\partial\theta_j\theta_k$, 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 $\partial^r f/\partial\theta_j^r$ and $\partial^{r+s} f/\partial\theta_j^r\theta_k^s$ evaluated at $\boldsymbol{\theta}$.

	Leading term in truncation error		
Approx.	$j = k$	$j \neq k$	Evaluations
$h_{1,j,k}(\boldsymbol{\theta})$	δD_3	$\frac{1}{2}(\delta_j D_{2,1} + \delta_k D_{1,2})$	$\frac{1}{2}(p^2 + 3p + 2)$
$h_{2,j,k}(\boldsymbol{\theta})$	$\frac{7}{12}\delta^2 D_4$	$\frac{1}{12}\left(2\delta_j^2 D_{3,1} + 3\delta_j\delta_k D_{2,2} + 2\delta_k^2 D_{1,3}\right)$	$p^2 + p + 1$
$h_{3,j,k}(\boldsymbol{\theta})$	$\frac{1}{3}\delta^2 D_4$	$\frac{1}{6}\left(\delta_j^2 D_{3,1} + \delta_k^2 D_{1,3}\right)$	$2(p^2 + p)$
$h_{4,j,k}(\boldsymbol{\theta})$	$\frac{1}{90}\delta^4 D_6$	$-\frac{1}{6}\left(\delta_j^2 D_{3,1} - \delta_k^2 D_{1,3}\right)$	$p^2 + p$

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`.

Calculation	Approximation		
method	$g_1(\theta)$	$g_2(\theta)$	$g_4(\theta)$
Equation (6)	0.30	0.55	1.06
Standard `dpois`	0.90	1.79	–
Extended `dpois`	1.13	2.02	1.58

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.

			Approximation				
Example		Parameters	$h_1(\boldsymbol{\theta})$	$h_2(\boldsymbol{\theta})$	$h_3(\boldsymbol{\theta})$	$h_4(\boldsymbol{\theta})$	`hessian`
BDR 2.3		2	-4.49	-6.90	-7.11	-10.47	-11.53
BDR 4.6		4	-3.40	-3.31	-5.74	-7.34	-3.12
BDR 5.4		6	-3.66	-5.94	-6.06	-8.44	-8.11
BDR 5.2	LS	7	-4.36	-7.00	-7.93	-7.07	-10.77
BDR 5.2	t_4	8	-1.59	–	-3.67	-1.79	-0.01
BDR 4.3		1	-4.47	-6.01	-6.26	-9.50	-5.90
BDR 3.5	EXP	2	-4.88	-7.08	-7.31	-12.02	-10.68
BDR 3.5	LN	2(2)	-5.74	-7.53	-7.54	-11.17	-11.49

Table 4. CPU times of different approximations relative to the CPU time of h_3.

			Approximation				
Example		Parameters	$h_1(\boldsymbol{\theta})$	$h_2(\boldsymbol{\theta})$	$h_3(\boldsymbol{\theta})$	$h_4(\boldsymbol{\theta})$	`hessian`
BDR 2.3		2	0.69	1.00	1.00	2.14	3.57
BDR 4.6		4	0.42	0.75	1.00	1.38	2.80
BDR 5.4		6	0.41	0.75	1.00	1.74	3.23
BDR 5.2	LS	7	0.34	0.72	1.00	0.36	2.49
BDR 5.2	t_4	8	0.33	0.63	1.00	0.40	2.35
BDR 4.3		1	0.99	1.04	1.00	0.95	1.63
BDR 3.5	EXP	2	0.96	1.09	1.00	0.91	1.96
BDR 3.5	LN	2(2)	0.60	0.84	1.00	0.76	2.38

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 δ 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 δ 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 δ 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:

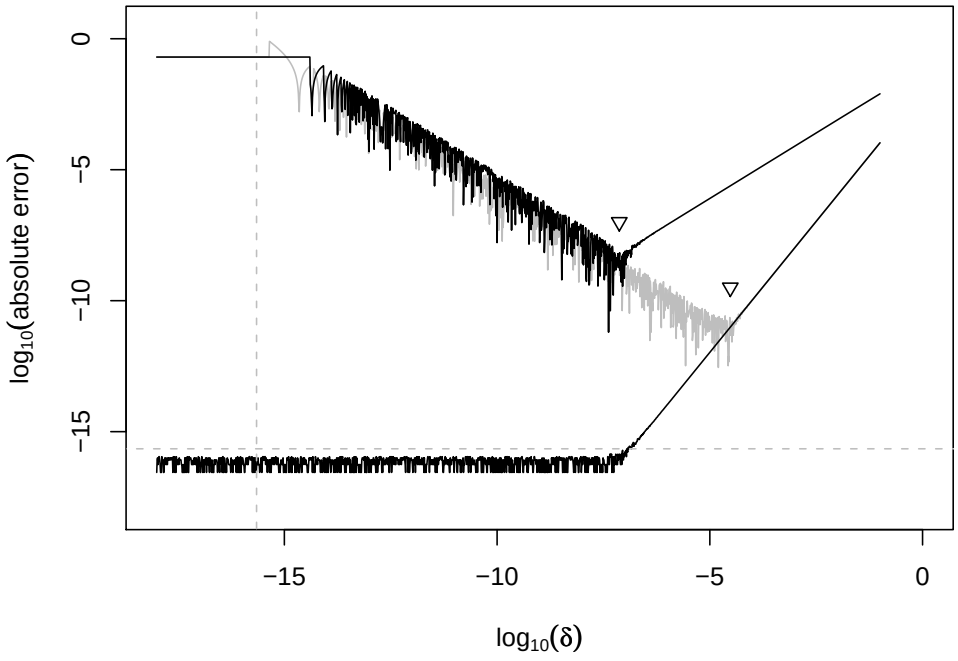

Figure 2:

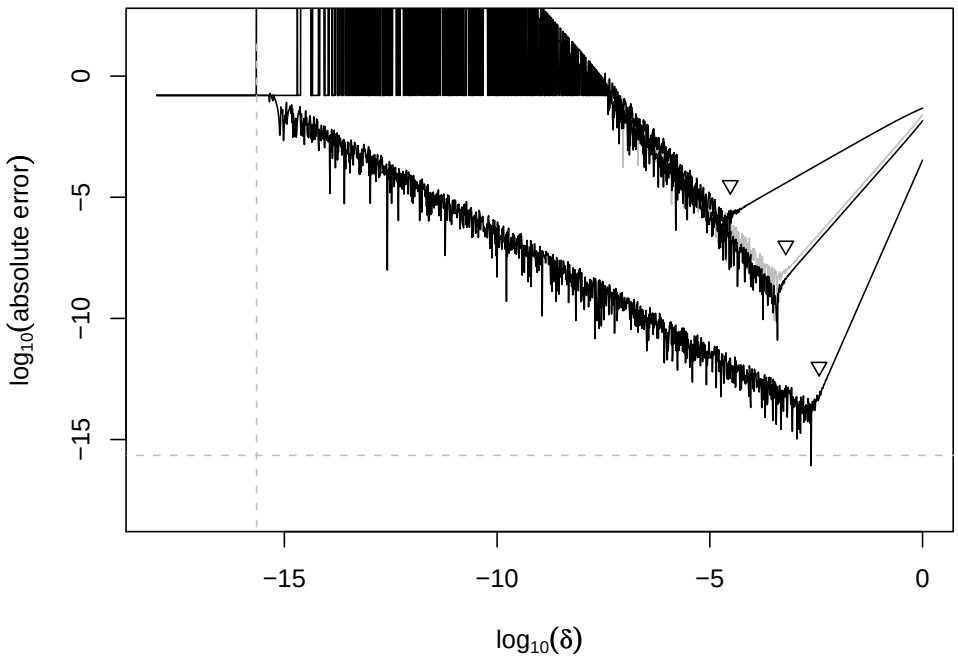

Figure 3:

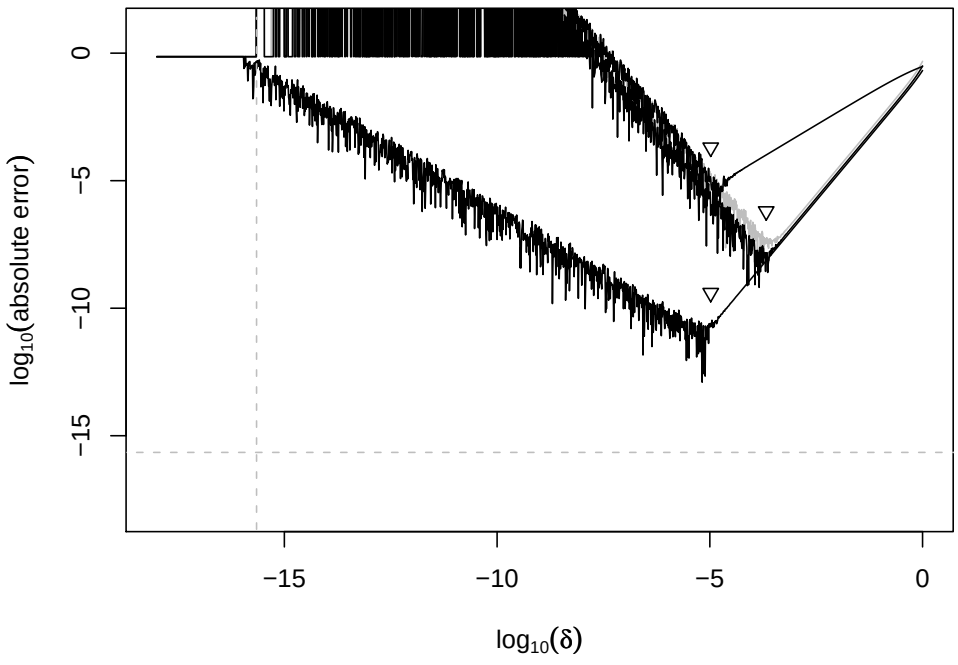

Figure 4:

```
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
```