The Devil is in the Detail: Hints for Practical Optimisation

T M Christensen, A S Hurn
School of Economics and Finance, Queensland University of Technology

K A Lindsay
Department of Mathematics, University of Glasgow.

September 17, 2008

Abstract

Finding the minimum of an objective function, such as a least squares or negative log-likelihood function, with respect to the unknown model parameters is a problem often encountered in econometrics. Consequently, students of econometrics and applied econometricians are usually well-grounded in the broad differences between the numerical procedures employed to solve these problems. Often, however, relatively little time is given to understanding the practical subtleties of implementing these schemes when faced with ill-behaved problems. This paper addresses some of the details involved in practical optimisation, such as dealing with constraints on the parameters, specifying starting values, termination criteria and analytical gradients, and illustrates some of the general ideas with several instructive examples.

Keywords
gradient algorithms, unconstrained optimisation, generalised method of moments.

JEL Classification C13, C63

Corresponding author
Stan Hurn
School of Economics and Finance
Queensland University of Technology
Brisbane, 4001, Australia
email s.hurn@qut.edu.au

1 Introduction

An important class of problem in econometric modelling is the unconstrained minimisation problem

$$\min_{\theta} \; f(x, \theta)\,, \tag{1}$$

where x is the observed data and θ is a vector of model parameters to be chosen to minimise the objective function $f(x, \theta)$. Typically, $f(x, \theta)$, the function to be minimised, will be either a (weighted) sum of squares or a negative log-likelihood function. In most realistic scenarios no closed-form expression for $f(x, \theta)$ (or its partial derivatives) can be constructed and so the minimisation process is almost invariably achieved by numerical methods. There are two broad flavours of minimisation algorithm, namely, procedures which rely on function values alone and algorithms that use the derivatives of the function. As the problems typically encountered in econometrics leads to objective functions that are at least twice differentiable, this paper will focus only on gradient-based algorithms.

At a superficial level, solving the minimisation problem is straightforward, usually involving only the choice of preferred econometric programming language like `MATLAB`, `GAUSS`, `OX` or `RATS`, or more interactive software packages like `EViews`. These packages usually provide reliable default settings for the user-supplied information required by a minimisation routine. It is therefore possible to solve advanced problems without ever inquiring into the meaning of many of the concepts involved in the minimisation process. An undesirable side effect of this is that the details of numerical optimisation are very often not included in introductory or even advanced courses in econometrics. Consequently, econometricians are adept at describing the similarities and differences of the typical iteration of variants of Newton-type algorithms, but are less proficient in navigating the murky waters of the subtle differences between truncation and rounding error, the extent of the loss of accuracy when computing numerical derivatives as opposed to using analytical derivatives and how to choose termination criteria optimally.

The aim of this paper, therefore, is to elucidate some of the devil in the detail lurking behind successful practical optimisation by emphasising that numerical minimsation requires a sequence of many floating-point operations performed with finite precision. To understand under what conditions an algorithm can fail and assess whether or not its final output represents an acceptable solution, it is desirable to have at least some appreciation of the more intricate aspects of the steps in the optimisation process.

The rest of the paper is structured as follows. Section 2 provides a brief summary of the basic structure of a gradient algorithm. Sections 3 to 6 describe many of the decisions that need to be made in any minimisation procedure and the criteria that have to be set to regulate the quality of the output. A practical example, based on Generalised Method of Moments (GMM) estimation is given in Section 7. The appendix describes two commonly used test problems that are also used in some of the examples in the paper. All the computation in the paper is done using `MATLAB` and all relevant code and data may be downloaded at `http://www.ncer.edu.au/data` or `http://www.eap-journal.com`.

2 A Quick Recap on Gradient Algorithms

As noted earlier, the decision to minimise the objective function $f(x, \theta)$ using an algorithm based on gradients assumes *a priori* that all its first and second derivatives exist. The gradient vector $G(\theta)$ and Hessian matrix $H(\theta)$ of the function $f(x, \theta)$ are defined respectively by

$$G(\theta) = \frac{\partial f(x,\theta)}{\partial \theta}\,, \qquad H(\theta) = \frac{\partial^2 f(x,\theta)}{\partial \theta \partial \theta'}\,. \tag{2}$$

Minima of the objective function occur at parameter values for which the gradient is zero and the associated Hessian matrix is positive definite. The estimator, $\widehat{\theta}$, of the parameter vector θ therefore satisfies the condition

$$G(\widehat{\theta}) = 0\,. \tag{3}$$

Given a guess $\widehat{\theta}_k$ which is assumed to be near the optimal value θ at which a minimum is attained, the Taylor series expansion of $G(\widehat{\theta})$ about $\widehat{\theta}_k$ is

$$G(\widehat{\theta}) = G(\widehat{\theta}_k + (\widehat{\theta} - \widehat{\theta}_k)) = G(\widehat{\theta}_k) + H(\widehat{\theta}_k)(\widehat{\theta} - \widehat{\theta}_k) + O(\widehat{\theta} - \widehat{\theta}_k)^2\,.$$

By replacing $G(\widehat{\theta})$ in equation (3) by the previous expression and ignoring all terms of order two and above, it follows that

$$\widehat{\theta} \simeq \widehat{\theta}_k - H(\widehat{\theta}_k)^{-1} G(\widehat{\theta}_k)\,.$$

Based on this result, the next guess for $\widehat{\theta}$ is

$$\widehat{\theta}_{k+1} = \widehat{\theta}_k - H(\widehat{\theta}_k)^{-1} G(\widehat{\theta}_k)\,. \tag{4}$$

The term $-H(\widehat{\theta}_k)^{-1} G(\widehat{\theta}_k)$ is usually referred to as a "full Newton step" which would normally be taken in the close vicinity of the minimum. Further away from the minimum, the full Newton step is not guaranteed to reduce the value of the function and a smaller step may be required and therefore the convention is to use the adjusted update scheme

$$\widehat{\theta}_{k+1} = \widehat{\theta}_k - \alpha_k H(\widehat{\theta}_k)^{-1} G(\widehat{\theta}_k)\,, \tag{5}$$

where α_k controls the step size and is chosen to ensure that the function is reduced at each iteration. The value of α_k is chosen by a one-dimensional minimisation procedure, known as line-searching, which is embedded in each iteration of the outer algorithm.[1]

Once the minimum has been located and parameter values determined, the standard errors of these parameters are usually found by taking the square root of the main diagonal of the inverse of the estimated Hessian matrix. Assuming that the shape of the objective function in the vicinity of the minimum is approximately parabolic, the Hessian matrix at the minimum is guaranteed to be a positive definite matrix and its value gives the curvature of the objective function at the minimum. Larger Hessian matrices correspond to sharper curvatures of the

[1] Line-searching is an important topic in its own right, but will not be examined in detail in this paper. It is important to note, however, that the efficacy of line-searching depends on the accuracy with which the objective function can be resolved in the direction of the line-search. If it is impossible to resolve the function to the requested accuracy, then the minimisation algorithm will fail in the line search and return an appropriate error flag.

objective function at the minimum, and consequently more precise estimates of the values of the optimal parameters.

All gradient-based algorithms employ the general iterative scheme in equation (5) and differ only in respect of their approximation of the Hessian matrix at each iteration. A Newton-Raphson algorithm computes the Hessian matrix directly, the Method of Scoring uses the Information matrix (negative of the expected value of the Hessian matrix) and the Berndt, Hall, Hall and Hausman (BHHH) algorithm (Berndt *et al.*, 1974) approximates the Hessian by the outer product of the gradient vector. Another important class of gradient methods are the quasi-Newton algorithms. Instead of computing the Hessian matrix these algorithms build up an approximation of it from successive iterations.

In theory there is little to choose between these algorithms because in the vicinity of a minimum each should enjoy quadratic convergence, which means that

$$\|\widehat{\theta}_{k+1} - \theta\| < \kappa \|\widehat{\theta}_k - \theta\|^2 \qquad \kappa > 0\,.$$

In other words, if $\widehat{\theta}_k$ is accurate to 2 decimal places, then it is anticipated that $\widehat{\theta}_{k+1}$ will be accurate to 4 decimal places and that $\widehat{\theta}_{k+2}$ will be accurate to 8 decimal places and so on. In choosing an algorithm, however, there are a few practical considerations to bear in mind.

(1) Because the Information matrix is the expected value of the negative Hessian matrix, it is problem specific and typically is not easy to compute. Consequently, the Method of Scoring is largely of theoretical interest.

(2) Close to the minimum Newton-Raphson converges quadratically but further away from the minimum, the Hessian matrix may not be positive definite and this may cause the algorithm to become unstable. This has led to a strategy in which the the starting values are refined by means of an algorithm based solely on function evaluations, such as the simplex algorithm (Nelder and Mead, 1965), before switching into the Newton-Raphson procedure.

(3) BHHH provides an estimate of the Hessian that is guaranteed to be positive definite and is therefore a popular choice in econometrics.

(4) The current consensus is that quasi-Newton algorithms that construct approximations to the Hessian matrix over successive iterations are the preferred choice. Specifically, the Hessian update of the BFGS algorithm (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970) is particularly robust and therefore most optimisation toolboxes will provide BFGS as an option, very often as the default choice.

3 Imposing Constraints

Although the minimisation problem formulated in equation (1) is an unconstrained one, few problems in econometrics are truly unconstrained, since often the specification of the problem dictates that certain parameters are required to lie within certain intervals. For example, the

parameters of autoregressive or moving-average time-series models must lie within the unit interval for the processes to be stationary. The parameters in an unconstrained minimisation algorithm, however, must be able to take all possible values. Once iteration commences, the user cannot control the values the algorithm selects for parameters. Thus if an unconstrained algorithm is used in the minimisation of an objective function for which the parameters are necessarily constrained, there is no mechanism to prevent the algorithm attempting to calculate the objective function for values of parameters which fail to satisfy the constraint conditions even although the true minimum is attained for values of the parameters for which all constraints are satisfied.

One approach is to abandon unconstrained optimisation and adopt the formal framework of constrained optimisation. Constrained minimisation procedures, however, are often difficult to formulate and to implement, and also suffer from slower convergence. In short, constrained minimisation algorithms are less robust than unconstrained algorithms. Another approach, often easy to implement, is for the user to specify the problem in such a way that an unconstrained algorithm will nevertheless choose model parameters that always satisfy the constraint conditions. This outcome can be accomplished by means of a bijective mapping from the true parameter space of the model into a user-defined unconstrained parameter space. The choice of mapping function will inevitably be nonlinear, because its role is to transform a finite or semi-infinite interval into an infinite interval.

Consider the case of estimating a single parameter θ where $\theta \in (a, b)$. The key idea is to transform the parameter θ by means of a non-linear bijective mapping, $\phi = g(\theta)$, between the constrained interval (a, b) and the real line. Thus each and every value of ϕ corresponds to a unique value of θ satisfying the desired constraint, and obtained by applying the inverse transform $\theta = g^{-1}(\phi)$. In other words, when the minimsation algorithm returns $\widehat{\phi}$, the value of ϕ that minimises the objective function, the associated estimate of θ is given by $\widehat{\theta} = g^{-1}(\widehat{\phi})$. Some useful one-dimensional transformations, their associated inverse functions and the gradients of the transformations are presented in Table 1. Note, however, that the application of the bijective mapping alters the scale of the variable being transformed and therefore its gradient. This has important implications for the minimisation algorithm if analytical derivatives are to be used, a point taken up in Section 6.

Constraint	Transform $\phi = g(\theta)$	Inverse Transform $\theta = g^{-1}(\phi)$	Jacobian $d\phi/d\theta$
$(0, \infty)$	$\phi = \log \theta$	$\theta = e^{\phi}$	$\frac{dg}{d\theta} = \frac{1}{\theta}$
$(-\infty, 0)$	$\phi = \log(-\theta)$	$\theta = -e^{\phi}$	$\frac{dg}{d\theta} = \frac{1}{\theta}$
$(0, 1)$	$\phi = \log\left(\frac{\theta}{1-\theta}\right)$	$\theta = \frac{1}{1+e^{-\phi}}$	$\frac{dg}{d\theta} = \frac{1}{\theta(1-\theta)}$
$(0, \pi)$	$\phi = \log\left(\frac{\theta}{\pi-\theta}\right)$	$\theta = \frac{\pi}{1+e^{-\phi}}$	$\frac{dg}{d\theta} = \frac{\pi}{\theta(\pi-\theta)}$
(a, b)	$\phi = \log\left(\frac{\theta-a}{b-\theta}\right)$	$\theta = \frac{b+ae^{-\phi}}{1+e^{-\phi}}$	$\frac{dg}{d\theta} = \frac{b-a}{(\theta-a)(b-\theta)}$
$(-1, 1)$	$\phi = \operatorname{atanh}(\theta)$	$\theta = \tanh(\phi)$	$\frac{dg}{d\theta} = \frac{1}{1-\theta^2}$
$(-1, 1)$	$\phi = \frac{\theta}{1-\lvert\theta\rvert}$	$\theta = \frac{\phi}{1+\lvert\phi\rvert}$	$\frac{dg}{d\theta} = \frac{1}{(1-\lvert\theta\rvert)^2}$
$(-1, 1)$	$\phi = \tan\left(\frac{\pi\theta}{2}\right)$	$\theta = \frac{2}{\pi}\tan^{-1}\phi$	$\frac{dg}{d\theta} = \frac{\pi}{2}\sec^2\left(\frac{\pi\theta}{2}\right)$

Table 1: Some useful transformations for imposing constraints on θ.

Self-evidently the choice of mapping used will be problem specific, but it is useful to note the following.

(1) The log transform is particularly useful in problems where the parameters are required to be positive, as will become clear in Section 7. Note that in many cases not only $d\phi/d\theta$ is required but also $d\theta/d\phi$ and in the case of the log transform, the latter is simply the parameter θ itself.

(2) A popular mapping used to constrain a parameter to the interval $(-1, 1)$ is

$$\phi = \frac{\theta}{1-|\theta|}\,.$$

This is fundamentally different to all of the other mappings listed in Table 1 in the respect that its second derivative is not defined at $\theta = 0$.

The price to be paid for the convenience of using an unconstrained minimisation algorithm on what is essentially a constrained problem is that the Hessian matrix returned by the algorithm cannot be used immediately to obtain the standard errors of the model parameters simply by taking the square roots of the diagonal elements of the inverse Hessian matrix. In this situation, there are two ways of obtaining standard errors for the parameters.

(1) A straightforward way to compute standard errors is to express the objective function in terms of the true parameters θ. The gradient vector and Hessian matrix can then be computed numerically at the minimum using the estimated values of the parameters. Because no searching is involved and the perturbation of the optimal parameters used in the computation of the finite differences is small, the probability of numerical instability is much reduced.

(2) Alternatively, the relationship between H_θ, the Hessian matrix with respect to the parameters of the model and H_ϕ, the Hessian matrix evaluated at the mapped parameters, can be constructed using the chain rule

$$\frac{\partial^2 f}{\partial\theta\partial\theta'} = \sum_{i=1}^{n} \frac{\partial f}{\partial\phi_i}\frac{\partial^2\phi_i}{\partial\theta\partial\theta'} + \sum_{i=1}^{n}\sum_{r=1}^{n} \frac{\partial\phi_i}{\partial\theta}\frac{\partial^2 f}{\partial\phi_i\partial\phi_r}\frac{\partial\phi_r}{\partial\theta'}\,. \tag{6}$$

At the minimum, the first term on the right hand side of this identity is zero because the gradient of the objective function with respect to the parameters ϕ is zero. It therefore follows from the identity (6) that

$$H_\theta = J' H_\phi J\,,$$

where

$$J = \begin{bmatrix} \frac{\partial\phi_1}{\partial\theta_1} & \frac{\partial\phi_1}{\partial\theta_2} & \frac{\partial\phi_1}{\partial\theta_3} & \cdots & \frac{\partial\phi_1}{\partial\theta_n} \\ \frac{\partial\phi_2}{\partial\theta_1} & \frac{\partial\phi_2}{\partial\theta_2} & \frac{\partial\phi_2}{\partial\theta_3} & \cdots & \frac{\partial\phi_2}{\partial\theta_n} \\ \frac{\partial\phi_3}{\partial\theta_1} & \frac{\partial\phi_3}{\partial\theta_2} & \frac{\partial\phi_3}{\partial\theta_3} & \cdots & \frac{\partial\phi_3}{\partial\theta_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial\phi_n}{\partial\theta_1} & \frac{\partial\phi_n}{\partial\theta_2} & \frac{\partial\phi_n}{\partial\theta_3} & \cdots & \frac{\partial\phi_n}{\partial\theta_n} \end{bmatrix},$$

is the Jacobian matrix of the transformation from θ to ϕ, in this instance evaluated at the minimum of the objective function. In the particular case in which each constrained parameter is transformed individually, the Jacobian is a diagonal matrix. Computing standard errors by scaling the Hessian in this way is known as the Delta Method.

4 Starting Values

All numerical algorithms require a starting value, $\widehat{\theta}_0$, for the parameter vector. If the objective function is globally concave, that is every chord of the objective function lies above the function, then there is a unique global minimum and the algorithm will converge to this global minimum for any initial choice of values for the parameters.

It is usually a good idea to check the computation of the objective function at the proposed starting values as this can be a useful diagnostic device. Two different types of error can occur. The first error is a failure to compute the objective function resulting in "NaN" being returned. Errors of this type usually occur as the result of a request to either divide by zero, compute the logarithm of zero or compute the exponential of a very large positive/negative number resulting in overflow/underflow. The second type of error occurs when the objective function has a complex value when computed at the initial point (or any other point). This happens because programs such as `MATLAB` and `GAUSS` do not fail when asked to take the logarithm or power of a negative number, but instead return a complex number. In these situations it is likely that a programming error has occurred in which a logarithm of a negative number has been taken or a negative number has been raised to a non-integer power.

If the computation of the objective function is well behaved for the chosen starting values, but fails as soon as minimisation algorithm is called, then is is likely that there has been a violation of an overlooked parameter constraint. As pointed out in Section 3, this is likely to occur if an unconstrained algorithm is used to minimise an objective function for which the parameters are constrained.

Previous empirical work of a similar nature may provide reasonable starting values for the model parameters, but it is not wise to assume that the convergence of the algorithm for a single choice of starting values provides a reliable indicator that a global minimum has been obtained. Consider for example the Judge model, described in the Appendix, in which the parameters are to be estimated by a simple nonlinear least squares fit. For a random choice of starting locations (β_1, β_2) chosen in the square $[-4, 4] \times [-4, 4]$ the global minimum of the Judge function will be identified approximately $55\% - 60\%$ of the time.[2] At the very least this observation suggests that the algorithm must be run for *more than one choice of starting parameters* to help guard against the isolation of a local as opposed to a global minimum.

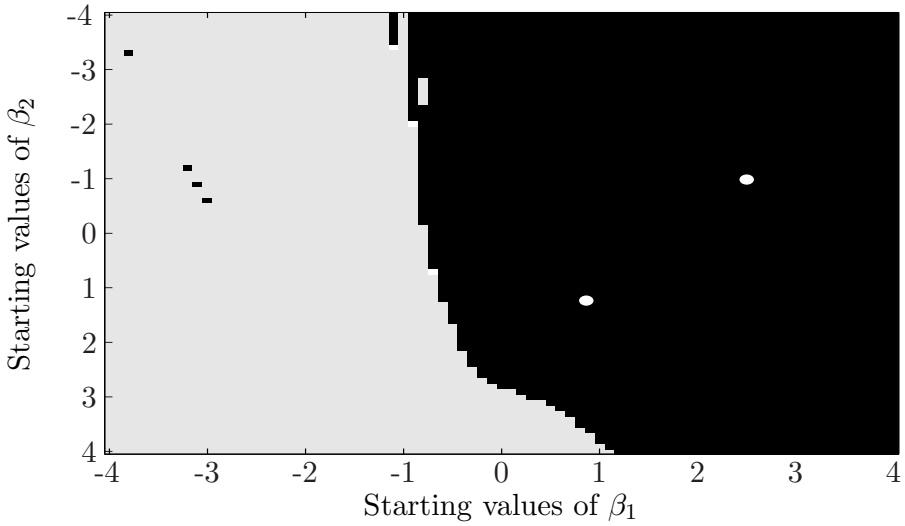

Figure 1: The pattern of convergence for different combinations of starting values for the parameters of the Judge model. The darker regions correspond to starting values that converged to the global minimum (lower white dot) whereas the lighter regions correspond to starting values that converged to the local minimum (upper white dot).

Figure 1 illustrates the convergence pattern for the Judge function for a grid of starting points (β_1, β_2) in the square $[-4, 4] \times [-4, 4]$. The result of this experiment indicates that it may sometimes be insufficient to make *minor changes* to starting values in order to test the robustness of a minimum. For example, starting points (β_1, β_2) with $\beta_1 < -1$ almost always converge to the local minimum irrespective of the choice of starting value for β_2. The reason for this stems from the behaviour of the surface of the function as can be seen in Figure 4 in the Appendix. There is a ridge approximate lying along the line $\beta_1 = -1$ and negative starting values for this parameter will mean that there is a strong probability that the algorithm will slide down the ridge in the direction of local minimum. The lesson to be learned here is that repeated starting values chosen at random offer a better chance of avoiding convergence to a local minimum than

[2]The BFGS algorithm was run 100 times with all the user specified parameters taking their default values.

a strategy which tests robustness of a proposed global minimum by simply making small changes to the original starting values of the parameter vector.

Counter intuitively, Figure 1 indicates that starting parameters close to the local minimum also appear to converge to the global minimum. The reason for this relates to the tolerance set on the gradient, a topic which is explored in the next section.[3]

5 Termination Criteria

All minimisation algorithms that revolve around an iterative procedure require stopping or termination criteria that will cause the algorithm will stop when one or more of these criteria are satisfied. Two straightforward termination criteria impose a maximum number of iterations or a maximum number of function evaluations and require no further explanation. More tricky to deal with are stopping criteria based on the change of the value of the objective function, the absolute value of its gradient and the change in the parameter values generated by an iteration. These criteria are specified in terms of user-supplied tolerances. An iterative algorithm will terminate when any one of these stopping criteria is satisfied. In order to set meaningful values for these user-specified tolerances it is first necessary to explore their meaning in more detail.

5.1 Tolerances

Tolerance defined on function value

Let $f(\widehat{\theta}_k)$ and $f(\widehat{\theta}_{k+1})$ denote the values of the function at iterations k and $(k+1)$. The change in the value of the function as the result of performing the $(k+1)^{th}$ iteration is regarded as small when

$$|f(\widehat{\theta}_{k+1}) - f(\widehat{\theta}_k)| < \tau_F\big(1 + |f(\widehat{\theta}_k)|\big)\,, \tag{7}$$

where τ_F is a user-supplied error tolerance for measuring changes in the value of the objective function. If $|f(\widehat{\theta}_k)|$ is less than one, then τ_F behaves as an absolute error for changes in the value of the function. On the other hand, if $|f(\widehat{\theta}_k)|$ is large, then τ_F behaves as a relative error, comparing changes in the value of the function with the value of the function itself. For this reason, τ_F is called a mixed-error tolerance and the algorithm will terminate whenever a condition like equation (7) is satisfied.

Tolerance defined on the gradient

The notion of a tolerance is more difficult to define in the context of vector-valued functions such as the gradient of a function with respect to its parameters. In this case, size is measured by means of a norm. In minimisation algorithms there are two popular choices of norm; the **Infinity norm** and the L^2 **norm**.

[3]Setting an undemanding tolerance for convergence on the gradient will result in the algorithm converging to the local minimum for starting values close to it.

Infinity norm: The infinity norm of the gradient vector is defined to be the maximum absolute value of the components of the gradient vector and is given by

$$\| G(\theta) \|_\infty = \max_i \left| \frac{\partial f(x,\theta)}{\partial \theta_i} \right| . \tag{8}$$

L^2 **norm:** The L^2 norm corresponds to the conventional definition for the length of a vector and for the gradient vector is given by

$$\| G(\theta) \|_2 = \sqrt{G(\theta)' G(\theta)} . \tag{9}$$

Since a smooth function achieves a minimum where the gradient of the function with respect to its parameters is zero, a gradient-based stopping condition will therefore require that a norm of the gradient vector is suitably small, that is

$$\| G(\widehat{\theta}_{k+1}) \| < \tau_G , \tag{10}$$

where τ_G is a user-supplied tolerance for the gradient.

Tolerance on change in parameters

Suppose the minimisation algorithm has reached a point for which each iteration changes the values of the parameters by an insignificant amount. When this situation occurs, the convention is to accept that the algorithm has converged to a local or global minimum. Let τ_X be a user-supplied tolerance for the change in values of the parameters over an iteration. A change in the value of the parameters is usually regarded as insignificant whenever

$$\| \widehat{\theta}_{k+1} - \widehat{\theta}_k \| < \tau_X (1 + \| \widehat{\theta}_k \|) , \tag{11}$$

where $\| \cdot \|$ is either the infinity or L^2 norm.

5.2 Choice of values for tolerances

The arithmetic used by computers has finite precision. Most econometric packages implemented on a desktop PC usually attain a maximum precision of 16 decimal places in their specification of real numbers. Press *et al.* (1992, p410) argue that τ_F can be set at machine precision or slightly above, while τ_X should not be set smaller than the square root of machine precision, that is, approximately 8 decimal places.

The basis for this argument stems from the fact that in the vicinity of a minimum the objective function exhibits parabolic behaviour. For example, in one dimension in the vicinity of a minimum at $\theta = 0$ the function value behaves approximately like

$$f - f_{\min} = a\theta^2 \qquad a > 0 . \tag{12}$$

It is clear therefore that resolution of the function to tolerance τ_F is associated with a resolution of θ to order $\sqrt{\tau_F}$. Furthermore, equation (12) may be interpreted as a forward difference

approximation to the gradient of the function and this suggests that the gradient of the function can also only be resolved to accuracy $\sqrt{\tau_F}$. In Section 6, however, the point will be made that a different way of computing numerical derivatives, namely, central differences, allows greater resolution of the gradients.

Setting τ_F to machine precision presupposes that the function can be computed to machine accuracy. This is only likely to be the case if the function is given by a closed form expression, such as the Rosenbrock function, described in the Appendix. By contrast, the Judge nonlinear regression model has an objective function defined as a sum of squares which is more computationally intensive and therefore likely to be less accurate. In most practical situations machine precision is unlikely be achieved in the computation of the objective function and the choice of tolerance will be more problem dependent and is therefore more subjective than the Press *et al.* (1992) rules of thumb suggest.

5.3 Implementation

Note that individual software packages use different variants of the termination conditions outlined previously thereby obscuring the meaning of some tolerances. For example, the optimisation toolbox in `MATLAB` provides the function `fminunc` for unconstrained minimisation problems in which the user supplies the tolerances `TolFun` and `TolX`. The sequence of tests for convergence is as follows.

(1) The infinity norm of the gradient is tested at the initial parameter values and the algorithm terminates if

$$\|G(\widehat{\theta}_0)\|_\infty < \texttt{TolFun}\,.$$

(2) At each iteration, the algorithm tests for convergence on the gradient and terminates if

$$\|G(\widehat{\theta}_k)\|_\infty < \texttt{TolFun}(1 + \|G(\widehat{\theta}_0)\|_\infty)\,.$$

(3) At each iteration, the algorithm tests for convergence on the change in parameters and terminates if

$$\left\| \frac{\widehat{\theta}_{k+1} - \widehat{\theta}_k}{1 + \| \widehat{\theta}_k \|} \right\| < \texttt{TolX}\,.$$

(4) Finally, at each iteration the number of iterations and number of function evaluations are tested against the set maxima.[4]

However, it would seem that `MATLAB's fminunc` function treats `TolFun` as a tolerance on the gradient and *not* as a true tolerance on the function. This must be kept in mind, when choosing its value, an issue which is returned to later.

An empirical example illustrating the effects of changing the tolerances on the results reported by `fminunc` for the Rosenbrock function are given in Table 2. For each choice of tolerances, the

[4] In general, the number of function evaluations will exceed the number of iterations due to line searching and the computation of gradients.

function is minimised for random starting values for $(\beta_1, \beta_2) \in [-2, 2] \times [-2, 2]$. Note that the seed of the random number generator is set so that the same starting values are used for each set of tolerances.

Table 2: Convergence results using the BFGS algorithm to minimise the Rosenbrock Function with derivatives calculated numerically.

$\widehat{\beta}_1$	$\widehat{\beta}_2$	Function Value	Exit Flag	Number of Iterations	Function Evaluations
Tolerances:- $\tau_X = 0.5 \times 10^{-4}$ and $\tau_F = 0.5 \times 10^{-4}$					
0.9288	0.8628	0.0051	1	17	66
0.9914	0.9831	0.0001	1	23	96
0.9119	0.8315	0.0078	1	18	84
0.9834	0.9674	0.0003	1	4	15
0.9992	0.9984	0.0000	1	19	78
Tolerances:- $\tau_X = 0.5 \times 10^{-6}$ and $\tau_F = 0.5 \times 10^{-6}$					
1.0000	1.0000	0.0000	1	23	87
0.9993	0.9986	0.0000	1	25	102
0.9999	0.9998	0.0000	1	24	105
1.0001	1.0002	0.0000	1	13	45
1.0000	1.0000	0.0000	1	21	84
Tolerances:- $\tau_X = 0.5 \times 10^{-8}$ and $\tau_F = 0.5 \times 10^{-8}$					
1.0000	1.0000	0.0000	1	24	90
1.0000	1.0000	0.0000	1	28	111
1.0000	1.0000	0.0000	1	26	111
1.0000	1.0000	0.0000	1	15	51
1.0000	1.0000	0.0000	1	22	87
Tolerances:- $\tau_X = 0.5 \times 10^{-8}$ and $\tau_F = 0.5 \times 10^{-12}$					
1.0000	1.0000	0.0000	1	26	96
1.0000	1.0000	0.0000	-2	29	120
1.0000	1.0000	0.0000	2	27	114
1.0000	1.0000	0.0000	-2	16	72
1.0000	1.0000	0.0000	-2	23	90

Values of exit flag:

1 – Magnitude of gradient smaller than the specified tolerance
2 – Change in parameter was smaller than the specified tolerance
−2 – Line search cannot find an acceptable point in the current search direction

A number of interesting conclusions emerge from this simple illustration.

(1) When the tolerances `TolFun` and `TolX` take the relatively non-exacting values of 0.5×10^{-4}, the algorithm indicates successful convergence on the gradient but it is clear that iteration has ended prematurely. Neither the estimated parameters nor the function values are near their optimal values in any of the 5 repetitions of the experiment. The fact that the minimum of the Rosenbrock function lies in a fairly flat valley (see Figure 3 in the Appendix) means that the gradient criterion imposed by an inexacting specification of

TolFun can be satisfied for values of the parameters not close to those at which the actual minimum is attained.

(2) For tolerances that are almost identical to the MATLAB default tolerances of 1×10^{-6} the algorithm performs much better, but there are still discrepancies from the true values of the parameters in some repetitions of the experiment.

(3) For tolerances of 0.5×10^{-8} the true minimum is found with good accuracy on each occasion. At this point TolX is at the maximum value recommended by Press *et al.* (1992) but TolFun is still significantly greater than machine precision.

(4) When TolFun is increased to 0.5×10^{-12} the algorithm is seen not to converge on the gradient condition. This supports the previous contention that TolFun is in fact a gradient tolerance and should therefore be of the same order as the square root of machine precision. Note that the algorithm fails in the line search on three occasions returning a flag value of -2. This failure occurs because there is a conflict between TolFun and TolX. On the one hand the gradient condition cannot be met with any reliability since it demands too much accuracy, while on the other hand TolX is also at the limit of what can reasonably be achieved. The line search algorithm is being called under circumstances in which the function cannot be resolved accurately enough for regular termination of the line search algorithm. Simply specifying a less demanding value of TolX will result in a regular termination with an exit flag of 2.

To conclude, setting TolX at the square root of machine precision is probably too demanding for most problems and the MATLAB default value of 1.0×10^{-6} seems a reasonable starting point. It is clear from these experiments, that TolFun is not a tolerance defined on the value of the function but rather on value of its gradient. A useful strategy may be to decrease the value of TolFun until the algorithm is terminated by satisfying the TolX criterion. In this way both criteria are potentially achievable at the optimum.

6 Derivatives

If an analytical expression for the gradient of the objective function is available, then the gradient may be computed to the same accuracy as the function. Unfortunately most objective functions encountered in applications do not have a closed form expression for the gradient of the objective function and therefore all derivatives required by the minimisation algorithm must be calculated numerically, and very often numerical derivatives are the default option in minimisation routines. Whenever an analytical derivative can be computed, however, it is beneficial to do so both in terms of speed and accuracy.

6.1 Numerical derivatives

A numerical derivative is obtained by approximating the derivative of the objective function by the slope of a chord in the vicinity of the point. For example, a Taylor series expansion of the

function $f(x)$ about the point $x = a$ gives

$$\frac{f(a+h) - f(a-h)}{2h} = f'(a) + O(h^2) \tag{13}$$

for suitably small values of h. The central difference approximation of $f'(a)$ is obtained from (13) by ignoring the terms represented by $O(h^2)$ to obtain

$$f'(a) \approx \frac{f(a+h) - f(a-h)}{2h}\,, \tag{14}$$

where the right hand side of equation (14) is the difference between the values of the function at $x = a + h$ and $x = a - h$ divided by the change in x, that is, the gradient of the chord of the function centred at $x = a$. Computation of the derivative of a function by numerical differencing unavoidably introduces two sources of error.

(1) The error introduced by approximating a derivative by the gradient of a chord is called *truncation error* simply because the error arises as a result of terminating or truncating the infinite Taylor series expansion of $(f(a + h) - f(a - h))/2h$ at its first term, namely $f'(a)$. Expression (14) most accurately mimics the value of $f'(a)$ when h is close to zero and least accurately approximates $f'(a)$ when h is large. In particular, the size of this discrepancy is $O(h^2)$, that is, it grows like a quadratic function of h.

(2) Finite precision arithmetic means that all computations of the value of a function are subject to *rounding error*. If f can be computed to machine precision, ϵ, then the contribution of rounding error when estimating $f'(a)$ using formula (14) is $O(\epsilon/h)$.

Without providing explicit details of the analysis, it can be demonstrated that the sum of the truncation error and rounding error arising in the central difference approximation to the derivative of a function is minimised when $h = O(\epsilon^{1/3})$ and the associated minimum error is $O(\epsilon^{2/3})$. This is to be compared to order ϵ when the derivative can be computed analytically. We remark that when $f'(a)$ is estimated by a forward difference approximation, the estimate of the gradient is accurate to $O(\epsilon^{1/2})$ which is significantly larger that the comparable error arising in the use of the central approximation, but of course, the forward difference scheme does involve one less function evaluation.[5]

Consider the numerical computation of the derivative of $f(x) = \sqrt{x}$ at the point $x = 1$. Table 3 demonstrates the error in computing the derivative for various choices of h. It is clear that both large and small values of h give inferior estimates than intermediate values but that the central difference is everywhere superior. Notably, the best estimate for the forward difference scheme gives 8 decimal places of accuracy ($\approx \epsilon^{1/2}$), while the central difference approximation gives 11 decimal places of accuracy ($\approx \epsilon^{2/3}$).

[5]It should be noted that the current version of `fminunc` uses forward differences only.

Table 3: Rounding and truncation error arising in the computation of the derivative of $f(x) = \sqrt{x}$ at $x = 1$ by numerical differentiation using forward and central differences.

h	Error (Forward Difference)	Error (Central Difference)
$1.0e-002$	-0.001243788791105	0.000006250273449
$1.0e-003$	-0.000124937539036	0.000000062500006
$1.0e-004$	-0.000012499376034	0.000000000624445
$1.0e-005$	-0.000001249996828	0.000000000003276
$1.0e-006$	-0.000000124941224	0.000000000014378
$1.0e-007$	-0.000000011920520	0.000000000291934
$1.0e-008$	-0.000000003038735	0.000000002512380
$1.0e-009$	0.000000041370185	0.000000041370185
$1.0e-010$	0.000000041370185	0.000000041370185
$1.0e-011$	0.000000041370185	0.000000041370185
$1.0e-012$	0.000044450291171	0.000044450291171
$1.0e-013$	-0.000399638918680	0.000155472593633
$1.0e-014$	-0.011501869164931	-0.005950754041805
$1.0e-015$	-0.055910790149937	-0.000399638918680

6.2 Implementation

Consider again the empirical example involving the use of `MATLAB's` function `fminunc` to minimise the Rosenbrock function. In this case, the analytical derivatives given in equation (23) are used in the minimisation. The same simulation exercise as reported in Table 2 is now repeated using analytical gradients and the results reported in Table 4. There are three observations to be made.

(1) The number of function evaluations required to achieve convergence is significantly reduced because it is no longer necessary to compute the derivatives of the function numerically. This also significantly speeds up the computation.

(2) The increased accuracy of the gradient computation allows the tolerance `TolFun` to be significantly increased without compromising the convergence of the algorithm.

(3) In this particular example, `TolFun` can actually be set at machine precision without an error flag being returned, indicating that the gradient is being computed to machine precision.

The robust conclusion is therefore that if at all possible, analytical gradients should be used in function minisation.

Table 4: Convergence results using the BFGS algorithm to minimise the Rosenbrock Function with derivatives calculated analytically.

$\widehat{\beta}_1$	$\widehat{\beta}_2$	Function Value	Exit Flag	Number of Iterations	Function Evaluations
Tolerances:- $\tau_X = 0.5 \times 10^{-4}$ and $\tau_F = 0.5 \times 10^{-4}$					
0.9287	0.8625	0.0051	1	17	22
0.9914	0.9831	0.0001	1	23	32
0.9119	0.8315	0.0078	1	18	28
0.9834	0.9674	0.0003	1	4	5
0.9991	0.9984	0.0000	1	19	26
Tolerances:- $\tau_X = 0.5 \times 10^{-6}$ and $\tau_F = 0.5 \times 10^{-6}$					
1.0000	1.0000	0.0000	1	23	29
0.9993	0.9987	0.0000	1	25	34
0.9999	0.9998	0.0000	1	24	35
1.0001	1.0002	0.0000	1	13	15
1.0000	1.0000	0.0000	1	21	28
Tolerances:- $\tau_X = 0.5 \times 10^{-8}$ and $\tau_F = 0.5 \times 10^{-8}$					
1.0000	1.0000	0.0000	1	24	30
1.0000	1.0000	0.0000	1	28	37
1.0000	1.0000	0.0000	1	26	37
1.0000	1.0000	0.0000	1	15	17
1.0000	1.0000	0.0000	1	22	29
Tolerances:- $\tau_X = 0.5 \times 10^{-12}$ and $\tau_F = 0.5 \times 10^{-12}$					
1.0000	1.0000	0.0000	1	26	32
1.0000	1.0000	0.0000	1	30	39
1.0000	1.0000	0.0000	1	28	39
1.0000	1.0000	0.0000	1	18	20
1.0000	1.0000	0.0000	1	24	31

Values of exit flag:

1 – Magnitude of gradient smaller than the specified tolerance.

7 Putting Theory into Practice: GMM Estimation

The Cox-Ingersoll-Ross (1985) model for the short term interest rate $X(t)$ at time t is

$$dX = \alpha(\beta - X)\,dt + \sigma X^{\gamma}\,dW \tag{15}$$

where dW is an increment of the standard Wiener process so $dW \sim N(0, dt)$. The parameters $\theta = \{\alpha, \beta, \sigma, \gamma\}$ are interpreted as follows: β is the level to which the rate reverts; α controls the speed of mean reversion, σ is a volatility parameter and γ is the so-called levels-effect parameter.[6] There are many techniques to estimate the parameters of this model and of stochastic differential

[6]Note that by specifying the model in this fashion these parameters must be non-negative to have a meaningful interpretation.

equations in general (see, for example, Hurn *et al.*, 2007). The technique used to estimate the CIR model (15) in this instance is Generalised Method of Moments (Hansen, 1982).

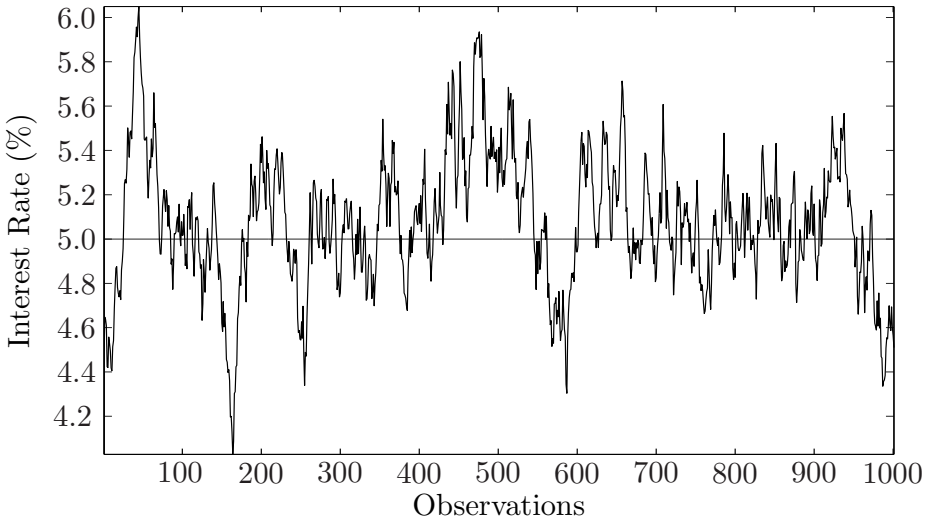

Figure 2: Artificial interest rate data generated from the CKLS model with parameters $\alpha = 0.05$, $\beta = 0.05$, $\sigma = 0.02$ and $\gamma = 0.75$.

7.1 Specification

Consider a sample of size $T + 1$, denoted $X_1, X_2, \ldots, X_{T+1}$. Moment conditions based on the discretisation of (15) following Chan *et al.* (1992) are

$$\begin{aligned} d_t^{(1)}(\theta) &= X_{t+1} - X_t - \alpha(\beta - X_t)\Delta \\ d_t^{(2)}(\theta) &= \left(d_t^{(1)}(\theta)\right)^2 - \sigma^2 X_t^{2\gamma}\Delta \\ d_t^{(3)}(\theta) &= d_t^{(1)}(\theta) X_t \\ d_t^{(4)}(\theta) &= d_t^{(2)}(\theta) X_t \,, \end{aligned} \tag{16}$$

where Δ represents the duration of the time interval at which X is sampled, here taken to have value 1 without any loss of generality.

GMM estimation is based on the fact that each moment condition has expectation zero.[7] When the expectation is replaced by the sample average of each moment condition, namely,

$$M(\theta) = \begin{bmatrix} M_1 \\ M_2 \\ M_3 \\ M_4 \end{bmatrix} = \begin{bmatrix} \frac{1}{T}\sum_{t=1}^{T} d_t^{(1)} \\ \frac{1}{T}\sum_{t=1}^{T} d_t^{(2)} \\ \frac{1}{T}\sum_{t=1}^{T} d_t^{(3)} \\ \frac{1}{T}\sum_{t=1}^{T} d_t^{(4)} \end{bmatrix},$$

the GMM estimator of θ is given by

$$\widehat{\theta} = \underset{\theta}{\operatorname{argmin}}\, M'(\theta)\Omega(\theta)M(\theta)\,, \tag{17}$$

[7] In this particular example, the number of moment conditions is equal to the number of parameters, so that the parameters are just identified. In general, this is not always the case and GMM also provides a way of testing any over identifying restrictions. This facet of estimation and testing is not pursued in this example.

where $\Omega(\theta)$ is a positive-definite weighting matrix. When $\Omega = I$, the identity matrix, the estimation will yield consistent, but not efficient, estimates of θ. To obtain both consistent and efficient estimates, Hansen (1982) suggests using the weighting matrix

$$\Omega(\theta) = \left[\frac{1}{T} D(\theta)' D(\theta)\right]^{-1}, \tag{18}$$

where

$$D(\theta) = \begin{bmatrix} d_1^{(1)} & d_1^{(2)} & d_1^{(3)} & d_1^{(4)} \\ d_2^{(1)} & d_2^{(2)} & d_2^{(3)} & d_2^{(4)} \\ \vdots & \vdots & \vdots & \vdots \\ d_T^{(1)} & d_T^{(2)} & d_T^{(3)} & d_T^{(4)} \end{bmatrix}. \tag{19}$$

Equations (17) and (18) can be used to estimate θ without any further input. An often overlooked fact, however, is that analytical derivatives are available for this particular problem. The matrix of derivatives of equations (16) with respect to the parameters α, β, σ and γ, in that order, is

$$-\begin{bmatrix} (\beta - X_t) & 2(\beta d_t^{(1)} - d_t^{(3)}) & (\beta X_t - X_t^2) & 2(\beta d_t^{(3)} - d_t^{(3)} X_t) \\ \alpha & 2\alpha d_t^{(1)} & \alpha X_t & 2\alpha d_t^{(3)} \\ 0 & 2\sigma X_t^{2\gamma} & 0 & 2\sigma X_t^{2\gamma+1} \\ 0 & 2\sigma^2 X_t^{2\gamma} \log X_t & 0 & 2\sigma^2 X_t^{2\gamma+1} \log X_t \end{bmatrix}. \tag{20}$$

Taking expectations of the matrix in expression (20) gives

$$J_\theta = -\begin{bmatrix} \beta - \mathrm{E}[X_t] & 2(\beta M_1 - M_3) & \beta \mathrm{E}[X_t] - \mathrm{E}[X_t^2] & 2(\beta M_3 - \mathrm{E}[d_t^{(3)} X_t]) \\ \alpha & 2\alpha M_1 & \alpha \mathrm{E}[X_t] & 2\alpha M_3 \\ 0 & 2\sigma \mathrm{E}[X_t^{2\gamma}] & 0 & 2\sigma \mathrm{E}[X_t^{2\gamma+1}] \\ 0 & 2\sigma^2 \mathrm{E}[X_t^{2\gamma} \log X_t] & 0 & 2\sigma^2 \mathrm{E}[X_t^{2\gamma+1} \log X_t] \end{bmatrix}, \tag{21}$$

where the expectations are understood to be replaced by sample averages in the computation.

There are two cases to consider in computing the analytic derivatives of the GMM objective function.

Identity weighting matrix In this case the gradient vector of the objective function is the (4×1) vector $2\, J_\theta\, M$.

Hansen weighting matrix In using the weighting matrix in equation (18), a choice must be made as to whether to iterate the weighting matrix or to set it at some constant value evaluated at the consistent parameter estimates obtained from an estimation based on the identity matrix. In this latter case, where Ω is constant, the gradient vector of the objective function is $2\, J_\theta\, \Omega\, M$.

7.2 Estimation

Prior to simply estimating the parameters of the interest rate model, the issues dealt with in Sections 3 to 6 are now addressed in a systematic way.

Constraints As the model is specified in equation (15), all the parameters must be positive in order for the model to have any meaning. Strictly speaking σ is the 'size' of the volatility and is normally required to be positive. In any analysis, σ will always occur as σ^2 or the volatility. At the outset, therefore a choice needs to be made as to whether to attempt to estimate the raw parameters or to enforce these constraints by means of the mapping strategy outlined earlier. For this example the mapping $\phi = \log \theta$ will be used.

Starting values Estimation results for the CKLS model will use two sets of starting values. The first set of starting values are the true parameters used to generate the simulated interest rate data, namely $\alpha = 0.05$, $\beta = 0.05$, $\sigma = 0.02$ and $\gamma = 0.75$. This, however, is artificial in the sense that starting values in a realistic problem require a little more work. In this instance, reasonable starting values could be those reported by Chan *et al.* (1992) for monthly data. As no guidance is available to aide in the choice of the starting value for the levels-effect parameter, γ, a sound option is to opt for the value 0.5 representing the well-known CIR model (Cox *et al.* 1985). The second set of starting values is therefore $\alpha = 0.02$, $\beta = 0.08$, $\sigma = 0.02$ and $\gamma = 0.50$.

Tolerances Because the `MATLAB` function `fiminunc` is used to estimate the parameters of the model, values for `TolFun` and `TolX` must be provided. To illustrate the effect of simply adopting the default values of these tolerances, estimation using numerical derivatives will use the value 1×10^{-6} for both. Estimation using analytical derivatives will set `TolFun` to be 1×10^{-16} (approximately machine accuracy) and `TolX` we be set at 1×10^{-8} reflecting the Press *et al.* (1992) rule of thumb.

Derivatives For this particular problem, analytical derivatives are available. Note however that when the minimisation is performed with scaled variables, the gradient of the objective function must be provided with respect to the scaled variables ϕ and not the model parameters θ. In other words, although J_θ is provided above and is easy to compute, it is J_ϕ that is required by the algorithm. In all cases, J_ϕ may be computed from J_θ by simply scaling the latter by the gradient of the mapping with respect to ϕ. In practice, this means post-multiplying J_θ by a matrix of gradients. The desirable characteristic of the logarithmic mapping is that the scaling matrix is diagonal and its elements are either 1 if no scaling has taken place, or the model parameter itself.

Table 5 reports the estimates of model parameters obtained when the identity matrix is used as the weighting matrix in the GMM objective function. Various combinations of untransformed (U) and transformed (T) parameters in combination with numerical (N) and analytical (A) derivatives are given.

Table 5: Parameter estimates of the CKLS model using simulated interest rate data for differing start values, tolerances and derivative computation. The true of the parameters are $\alpha = 0.05$, $\beta = 0.05$, $\sigma = 0.01$ and $\gamma = 0.75$. In panels 3 and 4 `TolFun` is 10^{-11} when numerical gradients are used and 10^{-16} when analytical gradients are used.

Protocol	$\widehat{\alpha}$	$\widehat{\beta}$	$\widehat{\sigma}$	$\widehat{\gamma}$	Iters	Fcn Evals	Flag
Start values:- True, `TolFun` $= 10^{-6}$, `TolX` $= 10^{-6}$							
U/N	0.049995	0.050768	0.019999	0.750000	2	25	1
U/A	0.049995	0.050768	0.019999	0.750000	2	5	1
T/N	0.050000	0.050000	0.020000	0.750000	0	5	1
T/A	0.050000	0.050000	0.020000	0.750000	0	1	1
Start values:- CKLS, `TolFun` $= 10^{-6}$, `TolX` $= 10^{-6}$							
U/N	0.000875	0.071863	0.019857	0.500009	4	35	1
U/A	0.000875	0.071863	0.019857	0.500009	4	7	1
T/N	0.018552	0.060642	0.019991	0.500328	2	40	1
T/A	0.018552	0.060642	0.019991	0.500328	2	8	1
Start values:- True, `TolFun` $= 10^{-11}/10^{-16}$, `TolX` $= 10^{-8}$							
U/N	0.049995	0.050768	0.009844	0.750605	17	105	1
U/A	0.063250	0.050768	0.009837	0.750605	42	46	2
T/N	0.049991	0.050768	0.017967	0.952874	18	125	1
T/A	0.063250	0.050768	0.017966	0.952961	43	50	1
Start values:- CKLS, `TolFun` $= 10^{-11}/10^{-16}$, `TolX` $= 10^{-8}$							
U/N	0.000006	0.071608	0.004753	0.500896	21	120	1
U/A	0.000006	0.071608	0.004753	0.500896	24	27	2
T/N	0.018292	0.050773	0.013877	0.863655	23	145	1
T/A	0.063250	0.050768	0.013871	0.866027	53	61	1

Protocol key:

U = untransformed parameters used in optimisation.
T = mapped parameters used in optimisation.
N = numerical derivatives used in optimisation.
A = analytical derivatives used in optimisation.

A number of conclusion emerge from consideration of these results.

(1) **Constraints:** It appears that the transformed parameter estimates are better behaved than the untransformed parameter estimates.

(i) The drift parameter α in the untransformed specification is driven to zero if it is not started at the optimal value. This effect is more pronounced at higher tolerances when it might realistically be expected that higher accuracy is achieved.

(ii) In the untransformed specification, the parameter γ is never significantly altered from its starting value, irrespective of the choice of starting value, tolerance or choice of derivative.

(iii) For the parameter β, which is known to be relatively easy to estimate, poor estimates are obtained for the untransformed specification even at demanding tolerances.

A tentative conclusion is that the transformed specification of the problem is to be preferred.

(2) **Tolerances:** Simply accepting the `MATLAB` default tolerances causes `fminunc` to converge too quickly with parameter estimates that are little changed (if at all) from their starting values. In the extreme case, the transformed specification of the model terminates without iteration if the true parameter values are used as starting values. For tolerances set using the guidelines established in Section 5, the results are improved in the sense that the algorithm performs a number of iterations and is seen to change the parameters from their intial values.

(3) **Gradients:** It is clear that most of function evaluations in the minimisation are used in the computation of gradients and there are significant savings to be made by providing analytical gradients. When more demanding tolerances are used, the transformed specification in conjunction with analytical gradients provides solid estimates of all the parameters except perhaps γ. The explanation for this is probably that the value of γ is an order of magnitude larger than any of the other parameters, indicating that the inefficiency of using the identity matrix for weighting.

Table 6 reports the estimates of model parameters obtained when the Hansen (1982) weighting matrix is used in the GMM objective function.

Table 6: Parameter estimates of the CKLS model using simulated interest rate data for differing start values, tolerances and derivative computation. The true values of the parameters are $\alpha = 0.05$, $\beta = 0.05$, $\sigma = 0.01$ and $\gamma = 0.75$. In panels 3 and 4 `TolFun` is 10^{-11} when numerical gradients are used and 10^{-16} when analytical gradients are used.

Protocol	$\widehat{\alpha}$	$\widehat{\beta}$	$\widehat{\sigma}$	$\widehat{\gamma}$	Iters	Fcn Evals	Flag
Start values:- True, `TolFun` $= 10^{-6}$, `TolX` $= 10^{-6}$							
N	0.063250	0.050768	0.010346	0.767549	43	285	1
A	0.063250	0.050768	0.010342	0.767416	42	55	1
Start values:- CKLS, `TolFun` $= 10^{-6}$, `TolX` $= 10^{-6}$							
N	0.063229	0.050769	0.011821	0.812266	27	180	1
A	0.063229	0.050769	0.011821	0.812266	27	36	1
Start values:- True, `TolFun` $= 10^{-11}/10^{-16}$, `TolX` $= 10^{-8}$							
N	0.063250	0.050768	0.010346	0.767536	34	225	-2
A	0.063250	0.050768	0.010345	0.767498	36	43	2
Start values:- CKLS, `TolFun` $= 10^{-11}/10^{-16}$, `TolX` $= 10^{-8}$							
N	0.063250	0.050773	0.010346	0.767543	39	235	2
A	0.063250	0.050768	0.010345	0.767498	35	38	2

Protocol key:

N = numerical derivatives used in optimisation.
A = analytical derivatives used in optimisation.

In each instance the weighting matrix is computed at the parameter estimates of the model obtained using the GMM objective function with the identity weighting matrix. In this experiment, only transformed parameters are used in combination with numerical (N) and analytical (A) derivatives. The parameter estimates obtained from the GMM objective function using the Hansen weighting matrix are significantly better than those obtained previously. There is also now little to choose between the accuracy of the parameter estimates provided by numerical as opposed to analytical derivatives, but the significant gain in terms of function evaluations remains. What is readily apparent, however, is the danger of simply accepting default tolerances in a minimisation problem. Starting with the true parameters all the parameters are well estimated, but in the more realistic scenario of using the CKLS starting values, it is clear that γ is very badly estimated. An unsuspecting user could easily be duped in accepting this value as the optimal estimate. With more demanding tolerances, however, all minimisations converge to the same parameter estimates independently of initial values and choice of derivative computation. Note, however, that the value of the flag indicates that convergence is now driven by `TolX` indicating that the more demanding tolerance on the gradients has not been met.

8 Conclusion

It has been argued here that nitty-gritty practical considerations, such as the treatment of constraints on parameters, choice of starting values, specification of termination criteria and the influence of analytical versus numerical gradients, can materially alter the final output of a minimisation algorithm. Self-evidently, these choices will also affect the computational effort required to achieve this final output. This paper has attempted to shed some light on these often overlooked but nonetheless important aspects of a numerical minimisation scheme.

References

Berndt, E., Hall, B., Hall, R. and Hausman, J. (1974). Estimation and inference in nonlinear structural models. *Annals of Social Measurement*. **3**, 653–665.

Broyden, C.G. (1970). The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematical Applications*, **6**, 76–90.

Chan, K. C., Karolyi, G. A., Longstaff, F. A. and Sanders, A. B. (1992). An Empirical Comparison of Alternative Models of the Short-Term Interest Rate. *Journal of Finance* 47(3), 1209–1227.

Cox, J. C., Ingersoll, J. E. and Ross, S. A. (1985). A Theory of the Term Structure of Interest Rates. *Econometrica*, 53(2), 385–407.

Fletcher, R. (1970). A new approach to variable metric algorithms. *Computer Journal*, **13**, 317–322.

Goldfarb, D. (1970). A family of variable metric methods derived by variational means. *Mathematics of Computation*, **24**, 23–26.

Hansen, L. P. (1982). Large Sample Properties of Generalized Method of Moments Estimators. *Econometrica*, 50(4), 1029–1054.

Hurn, A. S., Jeisman, J. I. and Lindsay, K. A. (2007). Seeing the Wood for the Trees: A Critical Evaluation of Methods to Estimate the Parameters of Stochastic Differential Equations. *Journal of Financial Econometrics*, 5(3), 390–455.

Judge, G., Hill, R.C., Griffiths, W.E., Lütkepohl, H. and Lee, T.C. (1985). *Introduction to the Theory and Practice of Econometrics*. John Wiley & Sons: New York.

Nelder, J.A. and Mead, R. (1965) A simplex method for function minimization. *Computer Journal*, **7**, 308–313.

Newey, W. K. and West, K. D. (1987). A Simple, Positive Semi-Definite, Heteroskedasticity and Autocorrelation Consistent Covariance Matrix. *Econometrica* 55(3), 703–708.

Press, W.H., Teukolsky, S.A., Vetterling, W.T. and Flannery, B.P. *Numerical Recipes in C*. Cambridge University Press.

Shanno, D.F. (1970). Conditional of quasi-Newton methods for function minimization. *Mathematics of Computation*, **24**, 647–657.

Appendix: Test problems

Rosenbrock function The Rosenbrock function

$$f(\beta_1, \beta_2) = 100(\beta_2 - \beta_1^2)^2 + (1 - \beta_1)^2 \tag{22}$$

is an artificial function that is often used to test the performance of numerical optimisation algorithms. The gradient of the function is given by

$$G(\beta) = \frac{\partial f(\beta)}{\partial \beta} = \begin{bmatrix} -400\beta_1(\beta_2 - \beta_1^2) - 2(1 - \beta_1) \\ 200(\beta_2 - \beta_1^2) \end{bmatrix}, \tag{23}$$

and the Hessian matrix is

$$H(\beta) = \frac{\partial^2 f(\beta)}{\partial \beta \partial \beta'} = \begin{bmatrix} 1200\beta_1^2 - 400\beta_2 + 2 & -400\beta_1 \\ -400\beta_1 & 200 \end{bmatrix}.$$

The Rosenbrock function, which is plotted in Figure 3, contains a long, curved valley with global minimum value zero at the point $(1, 1)$. It is clear from Figure 3 that the valley floor is almost flat and so gradient-based algorithms experience difficulty in isolating the minimum of the Rosenbrock function in the respect that they exhibit a tendency to stop prematurely if the termination conditions are not chosen appropriately.

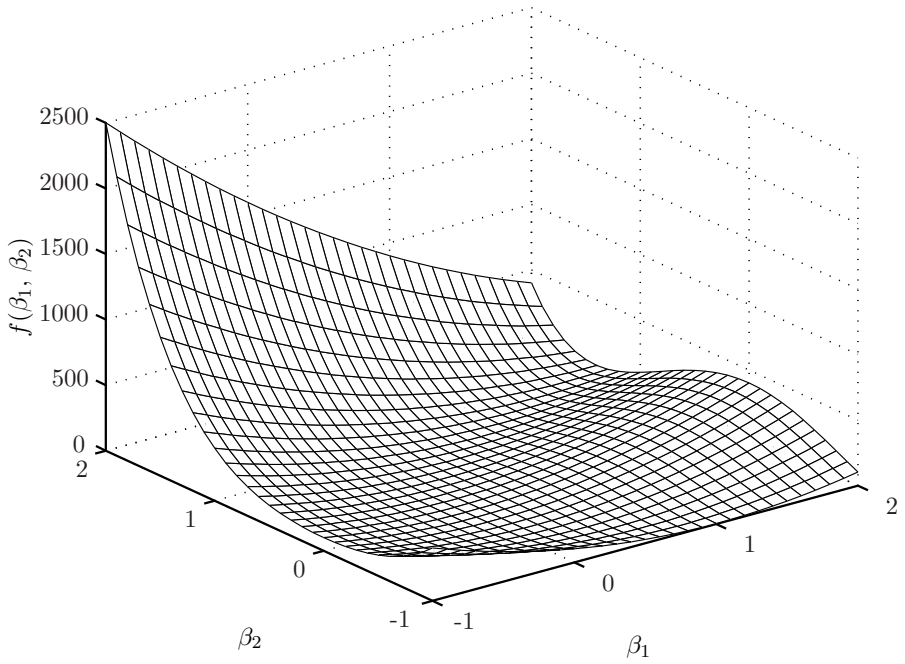

Figure 3: Surface of the Rosenbrock Test Function.

Judge model A useful test of a minisation routine is that provided by Judge *et al.* (1985, pp956-958) in the context of the nonlinear regression model

$$Y_t = \beta_1 + \beta_2 X_1 + \beta_2^2 X_2 + \epsilon_t \tag{24}$$

where Y_t, X_{1t} and X_{2t} are given in Table 7 and β_1 and β_2 are parameters to be estimated by a least squares fit.

N	Y	X_1	X_2	N	Y	X_1	X_2
1	4.284	0.286	0.645	11	2.106	0.936	0.142
2	4.149	0.973	0.585	12	1.428	0.889	0.296
3	3.877	0.384	0.310	13	1.011	0.006	0.175
4	0.533	0.276	0.058	14	2.179	0.828	0.180
5	2.211	0.973	0.455	15	2.858	0.399	0.842
6	2.389	0.543	0.779	16	1.388	0.617	0.039
7	2.145	0.957	0.259	17	1.651	0.939	0.103
8	3.231	0.948	0.202	18	1.593	0.784	0.620
9	1.998	0.543	0.028	19	1.046	0.072	0.158
10	1.379	0.797	0.099	20	2.152	0.889	0.704

Table 7: Data for the Judge *et al.* (1985) test function.

Figure 4 illustrates the resulting objective function which contains a local minimum at the point $(2.4986, -0.9826)$ with approximate function value 20.48234 and the global minimum at $(0.8648, 1.2357)$ with approximate function value 16.0817.

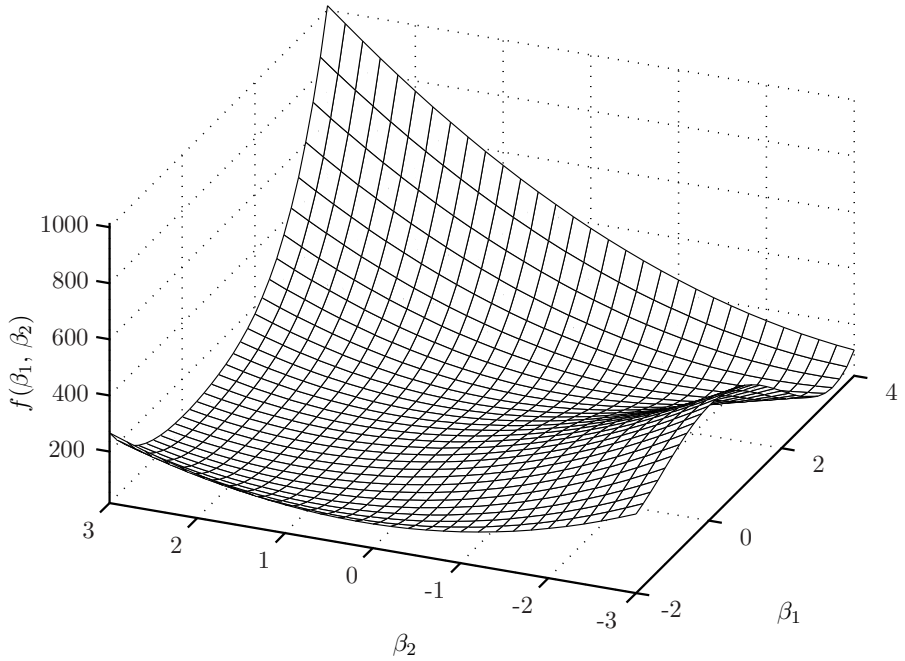

Figure 4: Surface of the Judge Test Function.