NASA Contractor Report 181927

ICASE Report No. 89–46

ICASE

NUMERICAL EXPERIENCE WITH A CLASS OF ALGORITHMS FOR NONLINEAR OPTIMIZATION USING INEXACT FUNCTION AND GRADIENT INFORMATION

Richard G. Carter

Contract No. NAS1–18605
June 1989

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, Virginia 23665–5225

Operated by the Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

Numerical Experience With a Class of Algorithms For Nonlinear Optimization Using Inexact Function and Gradient Information

Richard G. Carter *
Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, VA 23665

Abstract

For optimization problems associated with engineering design, parameter estimation, image reconstruction, and other optimization/simulation applications, low accuracy function and gradient values are frequently much less expensive to obtain than high accuracy values. We investigate the computational performance of trust region methods for nonlinear optimization when high accuracy evaluations are unavailable or prohibitively expensive, and confirm earlier theoretical predictions than the algorithm is convergent even with relative gradient errors of 0.5 or more. The proper choice of the amount of accuracy to use in function and gradient evaluations can result in orders-of-magnitude savings in computational cost.

*This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-18605 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23665.

1 Introduction

Consider the nonlinear optimization problem

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &x \in \Re^n \end{aligned} \tag{1}$$

where the function f has gradient ∇f, with ∇f assumed to be Lipschitz continuous. We are concerned with numerically solving this problem when function and gradient values are not known exactly.

Problems of this nature frequently occur in engineering design, parameter estimation, and many other situations. Consider, for example, the design of a heat sink for transferring excess heat away from an electronic component. Given the geometry of the sink (expressed, perhaps, as the spacing, thickness, and length of each cooling fin) and the heat flux from the component, one can mathematically model the temperature distribution in the sink and the surrounding medium by a system of partial differential equations. If we wish to find the design geometry which minimizes the time-averaged temperature of the component, we must numerically solve this system of PDE's at each iteration of the optimization algorithm to determine a value for the objective function f. Furthermore, a value for the gradient of f must be computed at each iteration either through successively perturbing each component of x and recomputing f to obtain a finite difference approximation, or through directly solving a larger system of PDEs. Clearly, exact function and gradient values are not attainable, and the computational expense of any approximation at a given iteration increases very rapidly as the required accuracy is increased. Let h be the discretization mesh size selected and m_d be the number of spatial dimensions in the PDE, so that the number of elements in the discretization is given by $k = o((1/h)^{m_d})$. If the order of the solution method is given by error $= o(h^{m_o})$, and the computational expense for the linear algebra associated with the problem is CPU $= o(k^{m_l})$, we have

$$\text{CPU/iteration} = o(error^{-l}) \tag{2}$$

for $l = \frac{m_d m_l}{m_0}$. Although this estimate is admittedly crude, it seems to hold for many applications and indicates that computational expense per iteration can rise *extremely* rapidly as more accurate solutions are required. In our example, if a full three-dimensional model of the sink is being used with a direct linear algebra solver and an $o(h^2)$ solution method, we would have $m_d = 3\,, m_l = 3\,, m_0 = 2$; hence l would be 3 and a thousand-fold increase in computational time would be needed to increase the accuracy of any given approximation by one digit. Problems involving systems of ODEs tend to be more benign with much smaller values of $l(\frac{1}{4}$ to $\frac{1}{10})$, but computational expense still increases geometrically with accuracy.

Since low accuracy function and gradient evaluations can be orders of magnitude less expensive than high accuracy evaluations, it behooves us to consider optimization algorithms that do not require the maximum possible accuracy at each iteration. Trust region methods are a natural candidate for investigation because of their reputation for robustness and efficiency. A number of authors have established global convergence results for trust region methods using inexact gradients ([12], [3], [17]), and inexact function evaluations have also been treated [4]. In this paper, we investigate the numerical behavior of the algorithms presented in [3] and [4] in order to answer the following questions.

1. How much error can we allow in our evaluations before the algorithm fails? Does this level agree with theoretical predictions?

2. The performance of the algorithm will certainly decrease when less accurate evaluations are used. How fast does performance degrade and how problem-dependent is the rate of degradation?

3. How does this lessened performance balance with the greatly decreased computational cost associated with less accurate evaluations?

4. How well do the techniques suggested in [3] for estimating and controlling gradient error work in practice?

The remainder of this paper is organized as follows.

In Section 2, we present the trust region algorithm and review the conditions on permissible levels of error established in [3]. These conditions depend on some of the parameters of the trust region method but are remarkably relaxed: typically relative errors in the gradient of 0.5 or more are permissible. In Section 3, we examine the performance of the algorithm on the set of standard test problems from Moré, Garbow, and Hillstrom [13] when synthetically generated errors are added to the gradients at each iteration. Our results confirm the theoretical predictions for the algorithm, and we note that the number of iterations required by the algorithm tends to increase exponentially with the relative error induced in the gradient. When balanced against (2), however, we find that allowing low accuracy evaluations is still attractive. In Section 4, we examine the performance of the algorithm on a parameter identification problem found in the literature in order to confirm our results without resorting to synthetically induced errors or invoking (2). We also verify a technique for estimating and controlling error when the gradient is approximated by finite differences. Section 5 summarizes our results.

2 The trust region algorithm and permissible error in function and gradient evaluations

The trust region method for solving (1) generates a sequence of iterates$\{x_k\}$ by approximately solving a sequence of constrained quadratic model problems. Each local quadratic model is of the form

$$\Psi_k(x_k + s) = f_k + g_k^T s + \frac{1}{2} s^T B_k s \tag{3}$$

where f_k is an approximation to $f(x_k)$, g_k is an approximation to the gradient $\nabla f(x_k)$, and the symmetric matrix $B_k \in \Re^{n \times n}$ is an approximation to the Hessian matrix $\nabla^2 f(x_k)$. At each iteration, we take $x_{k+1} = x_k + s_k$, where s_k is an approximate solution to the *trust region subproblem*

$$\begin{array}{ll} \text{minimize} & \Psi_k(x_k + s) \;\; s/t \; \|D_k s\| \leq \Delta_k. \\ s \in \Re^n & \end{array} \tag{4}$$

The positive scalar Δ_k is known as the *trust radius*, and the nonsingular matrix $D_k \in \Re_N$ is the *scaling* or *preconditioning* matrix (often taken to be a fixed diagonal matrix). At each iteration, Δ_k is adjusted so that the ball $\|D_k s_k\| \leq \Delta_k$ represents the region over which we expect (3) to adequately model the function f.

A number of techniques are available for computing an approximate solution to (4). An excellent survey of the main classes of these methods can be found in [12]. In our computations, we chose to use an *optimal locally constrained*, or *OLC* technique [6]. Similarly, a number of techniques can be used to generate the Hessian approximation $\{B_k\}$, but we selected the popular BFGS secant update

$$B_{k+1} = B_k + \frac{(g_{k+1} - g_k)\,(g_{k+1} - g_k)^T}{(g_{k+1} - g_k)^T s_k} - \frac{B_k s_k s_k^T B_k^T}{s_k^T B_k s_k} \tag{5}$$

when $(g_{k+1} - g_k)^T s_k \geq 10^{-6}(g_{k+1} - g_k)^T(g_{k+1} - g_k)$, and $B_{k+1} = B_k$ otherwise. Since g_k is only an approximation to $\nabla f(x_k)$, [4] and [3] suggest that upper bounds of the form

$$\|B_k\| \leq c_1 \tag{6}$$

or

$$g_k^T B_k g_k \leq c_1 g_k^T g_k \tag{7}$$

be directly enforced for some appropriately large c_1. This could be easily done by using the replacement operation

$$B_k := \min\{1, \frac{c_1}{\|B_k\|}\}B_k \tag{8}$$

at each iteration. Although bounds such as (6) and (7) are needed to establish convergence results, in practice we found (8) unnecessary.

A simple version demonstrating the salient features of the trust region algorithm is as follows.

Algorithm (1): The trust region method.

Let the constants $0 < \eta_1 < \eta_2 < \eta_3 < 1$ be prespecified. Select an initial guess $x_0 \in \Re^N$ and an initial trust radius Δ_0. Compute f_0 and g_0, and compute or initialize B_0. *For* $k = 0, 1, \ldots$ *until* "convergence" *do*:

(a) Determine an approximate solution s_k to problem (4).

(b) Calculate the predicted function reduction.

$$\text{pred}_k(s_k) = -g_k^T s_k - \frac{1}{2} s_k^T B_k s_k \tag{9}$$

and the computed function reduction

$$\text{cred}_k(s_k) = f_k - f_{k+1}. \tag{10}$$

If necessary, recompute f_{k+1} and/or f_k to greater accuracy.

(c) Compute the ratio

$$\rho_k = \frac{\text{cred}_k(s_k)}{\text{pred}_k(s_k)}. \tag{11}$$

(d) If $\rho_k < \eta_1$, then the step is unacceptable. Set $\Delta_k := \frac{1}{10}\Delta_k$ and return to (a).

(e) If $\eta_1 \leq \rho_k < \eta_2$, then set $\Delta_{k+1} = \frac{1}{2}\Delta_k$,

Else if $\eta_3 < \rho_k \leq (2 - \eta_3)$, then set $\Delta_{k+1} = 2\Delta_k$,

Else set $\Delta_{k+1} = \Delta_k$.

(f) Set $x_{k+1} = x_k + s_k$, compute g_{k+1}, and compute or update B_{k+1}.

End loop

End algorithm

Typical values for the step acceptance/trust radius update parameters are $\eta_1 = 0.001$, $\eta_2 = 0.1$, and $\eta_3 = 0.75$. Notice that no step is accepted unless $\rho_k \geq \eta_1$, and that the trust radius is never reduced unless $\rho_k \leq \eta_2$. Further notice that g_k is only computed once per major iteration.

Two conditions are required of the approximate function values. Define

$$\text{ared}_k(s_k) = f(x_k) - f(x_k + s_k). \tag{12}$$

We then require

$$|\text{ared}_k(s_k) - \text{cred}_k(s_k)| \geq \zeta_{f,1}\ \text{pred}_k(s_k) \tag{13}$$

and

$$|\text{ared}_k(s_k) - \text{cred}_k(s_k)| \leq \zeta_{f,2}\ |\text{cred}_k(s_k)| \tag{14}$$

for some constants $\zeta_{f,1}$ and $\zeta_{f,2}$. A stronger variation of these conditions that is typically more practical is

$$|f_{k+1} - f(x_{k+1})| + |f_k - f(x_k)| \leq \zeta_{f,1}\ \text{pred}_k(s_k) \tag{15}$$

and

$$|f_{k+1} - f(x_{k+1})| + |f_k - f(x_k)| \leq \zeta_{f,2}\ |\text{cred}_k(s_k)| \tag{16}$$

Assuming error estimates are available for $|f_k - f(x_k)|$, [3] suggests that (15) and (16) be enforced by using the following procedure in place of step (b) of Algorithm (1).

Procedure 2

Let $\alpha \in (0,1)$ be prespecified. Given x_k, s_k, Ψ_k, and an estimate for $|f_k - f(x_k)|$, do the following.

(i) Calculate $\text{pred}_k(s_k)$ and $\text{emax} = \zeta_{f,1}\text{pred}_k(s_k)$.

(ii) If necessary, recompute f_k to greater accuracy so that $|f_k - f(x_k)| \leq \alpha\,\text{emax}$.

(iii) Compute f_{k+1} so that $|f_{k+1} - f(x_{k+1})| \leq (1-\alpha)\text{emax}$.

(iv) Compute $\text{cred}_k(s_k)$. If (16) is satisfied, then exit procedure, else reduce emax and return to (ii).

End procedure

The permissible level of error in the gradient evaluations can be characterized in two different ways. The preferable condition is

$$\frac{\|D_k^{-1}e_k\|}{\|D_k^{-1}g_k\|} \leq \zeta_g \tag{17}$$

for some constant ζ_g, with

$$e_k = g_k - \nabla f(x_k). \tag{18}$$

Under appropriate assumptions on f and $\{D_k\}$, equation (17) leads to the strong global convergence result $\lim_{k\to\infty}\|g_k\| = \lim_{k\to\infty}\|\nabla f(x_k)\| = 0$. The weaker convergence result $\liminf_{k\to\infty}\|g_k\| = 0$ can be obtained using the condition

$$\frac{(D_k^{-1}e_k)^T\,(D_k^{-1}g_k)}{(D_k^{-1}g_k)^T\,(D_k^{-1}g_k)} \leq \zeta_g \tag{19}$$

If each scaling matrix D_k is taken to be the identity, (17) and (19) become simply

$$\frac{\|e_k\|}{\|g_k\|} \leq \zeta_g \tag{20}$$

and

$$\frac{e_k^T g_k}{g_k^T g_k} \leq \zeta_g. \tag{21}$$

Finally, we must specify the allowed values for the error bounds $\zeta_{f,1}$,$\zeta_{f,2}$ and $\zeta_{f,3}$. These values are given by the inequalities

$$\zeta_g + \zeta_{f,1} < 1 - \eta_2 \tag{22}$$

and

$$0 \leq \zeta_{f,2} < 1, \tag{23}$$

with $\zeta_g \geq 0$ and $\zeta_{f,1} \geq 0$. These limits are *remarkably* generous. If η_2 (the parameter controlling trust radius reduction) is 0.1, we could select $\zeta_{f,1} = 0.05, \zeta_{f,2} = 0.99$, and $\zeta_g = 0.8$ — less than one significant bit of accuracy in the components of the gradient approximation.

3 Algorithm Performance on Moré-Garbow-Hillstrom Test Problems With Synthetically Induced Errors

The Moré-Garbow-Hillstrom test set [13] contains eighteen"typical" optimization problems. These problems are all algebraically defined, and thus function and gradient values are both inexpensive to compute and available to high accuracy. In order to test the effects of gradient error, we synthetically induced a random error into each gradient computed. More specifically, we computed a vector w_k with each component selected from a uniform distribution on [-1,1] and then set $g_k = \nabla f(x_k) + e_k$ with

$$e_k = 100 * w_k * \|\nabla f(x_k)\|/2^m, \tag{24}$$

where m is the smallest positive integer for which (20) is satisfied. For our first set of tests no errors were induced in the computation of function values.

our optimization code, we used an implementation of the Dennis-Schnabel routines an OLC step computation technique. The model Hessians B_k were computed using dard BFGS procedure (5). We emphasize that no special techniques were used to for gradient error when computing B_k, nor were safeguards such as (8) enforced. ue for η_2 in the optimization code was 0.1, so that the theory in [4] and [3] suggests r limit of 0.9 for ζ_g.

ach problem in the test set, we considered twenty different values of ζ_g ranging to 0.95. For each value of ζ_g, we ran the algorithm with a number [1] of different tions for the random number generator used to compute e_k values, and tabulated mum, median, and maximum number of iterations required to converge to a local er. In all, over 5000 test cases were run on a network of SUN 3/50 workstations in AN 77 double precision.

es (1) through (5) show plots of our results for selected problems. Let $K(\zeta_g)$ denote ber of iterations required for convergence as a function of relative gradient error. tical axis in each plot represents the natural log of $K(\zeta_g)$, while the three traces in t represent the observed minimum, median, and maximum values.

es (1) through (3) are typical of most of our test cases, in which performance exponentially:

$$K(\zeta_g) \approx K(0)\exp(b\zeta_g), \tag{25}$$

with observed decay coefficients b ranging from roughly 2 to 6 for the various problems. Figures (4) and (5) represent anomalous cases where $\ln(K(\zeta_g))$ has significant variation from linearity at small and large values of ζ_g.

Although the exponential performance degradation given by (25) is a telling argument

[1]Depending on the computational expense associated with a problem and the observed variability of results, between 5 and 100 test cases were run for each value of ζ_g.

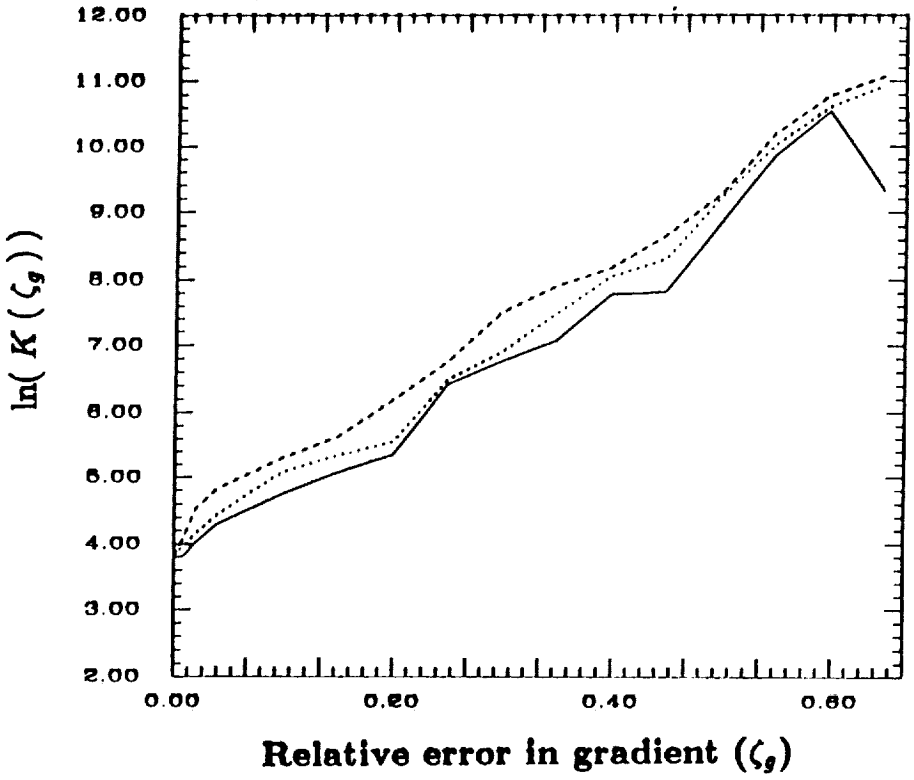

Figure 1: Number of iterations versus ζ_g for the Watson Test Function.

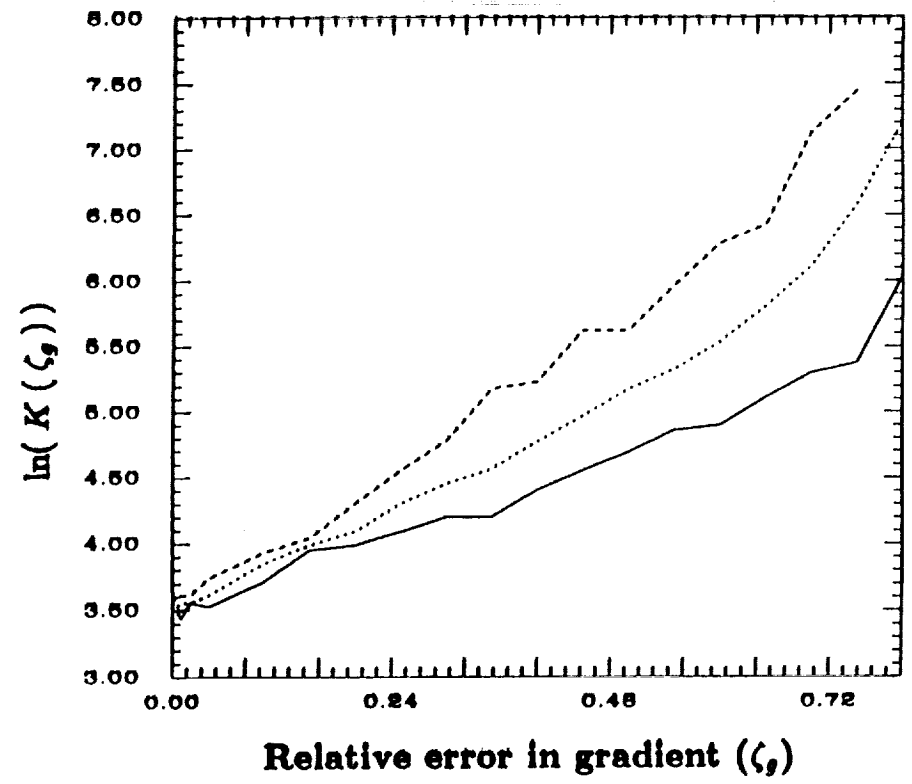

Figure 2: Number of iterations versus ζ_g for the Brown and Dennis Test Function.

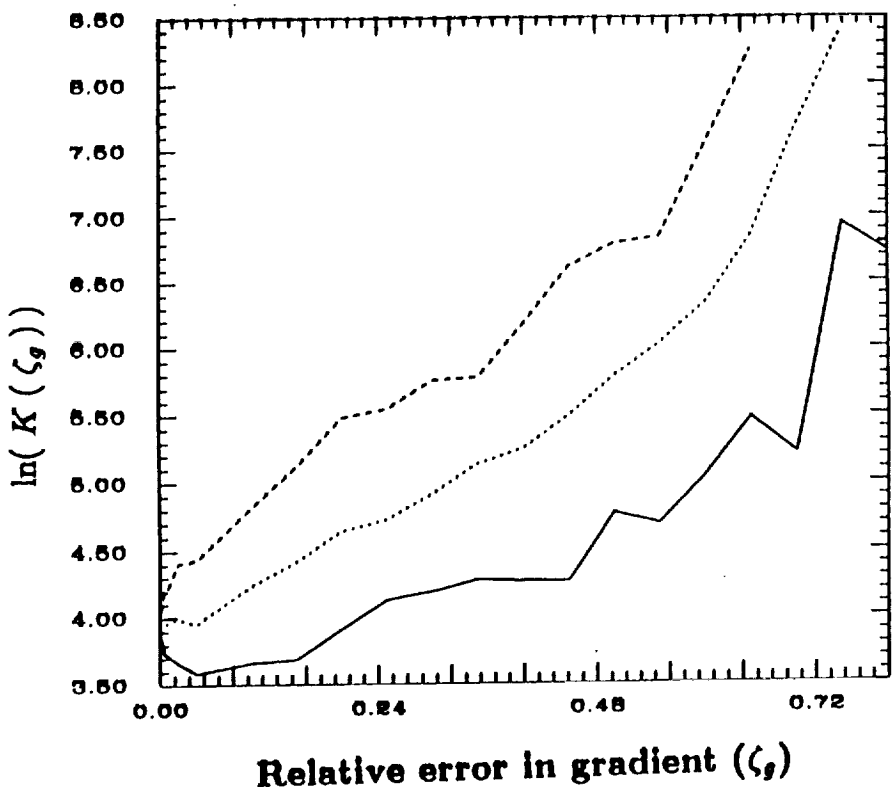

Figure 3: Number of iterations versus ζ_g for the Extended Powell Singular Test Function.

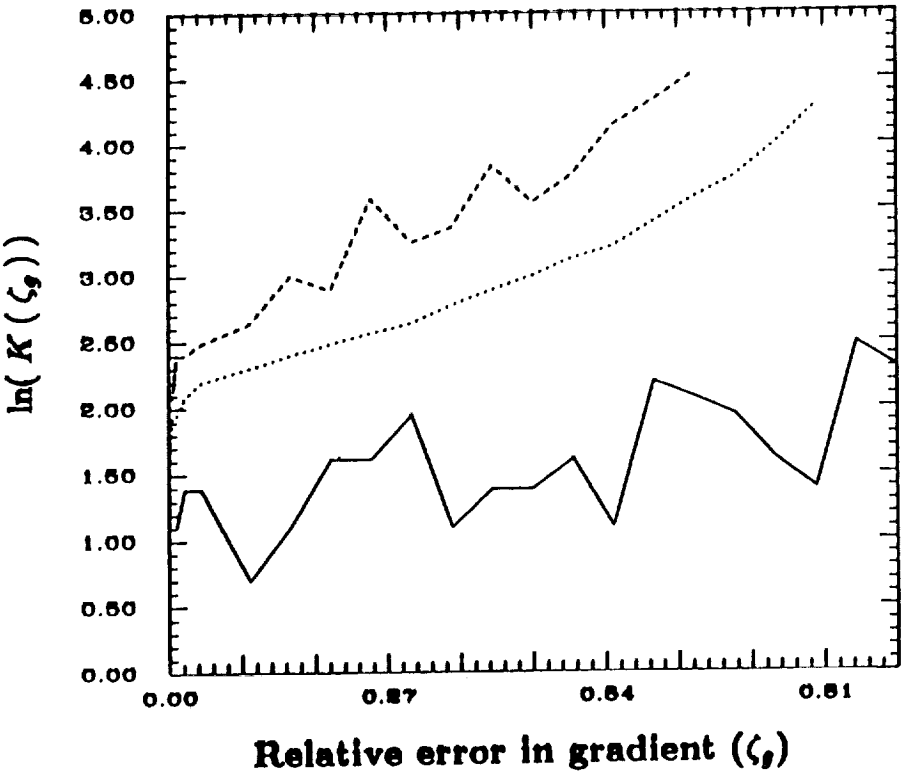

Figure 4: Number of iterations versus ζ_g for the Gaussian Test Function.

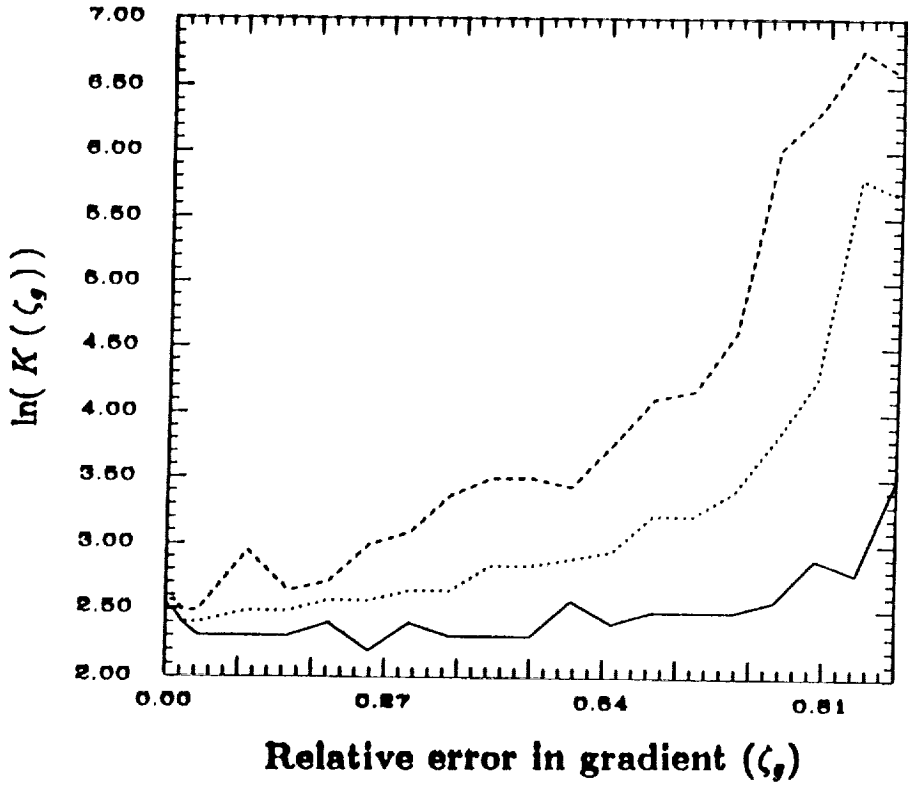

Figure 5: Number of iterations versus ζ_g for the Trigonometric Test Function.

against using low accuracy gradients if high accuracy evaluations are obtainable without greatly increased computational expense, low accuracy evaluations are still attractive in cases where the computational expense increases rapidly with increasing accuracy. Suppose, for instance, that

$$\text{CPU/iteration} \approx c_2 \zeta_g^{-l} \tag{26}$$

for some constants c_2 and l. The total computational expense for solving a given problem will then be proportional to $\zeta_g^{-l} K(\zeta_g)$. Figures (6) through (10) show the predicted total computational cost of the median curves for $K(\zeta_g)$ in figures (1) through (5) for the values $l = \frac{1}{2}, 1$, and 2 (with each curve normalized so that the minimum value is 1.0).

Notice that each curve of total computational cost increases very rapidly as $\zeta_g \to 0$ or $\zeta_g \to 1$, and has a relatively large, flat minima. This behavior holds for both the "typical" cases (figures (6) through (8)) and the anomalous cases (figures (9) and (10)). Interestingly,

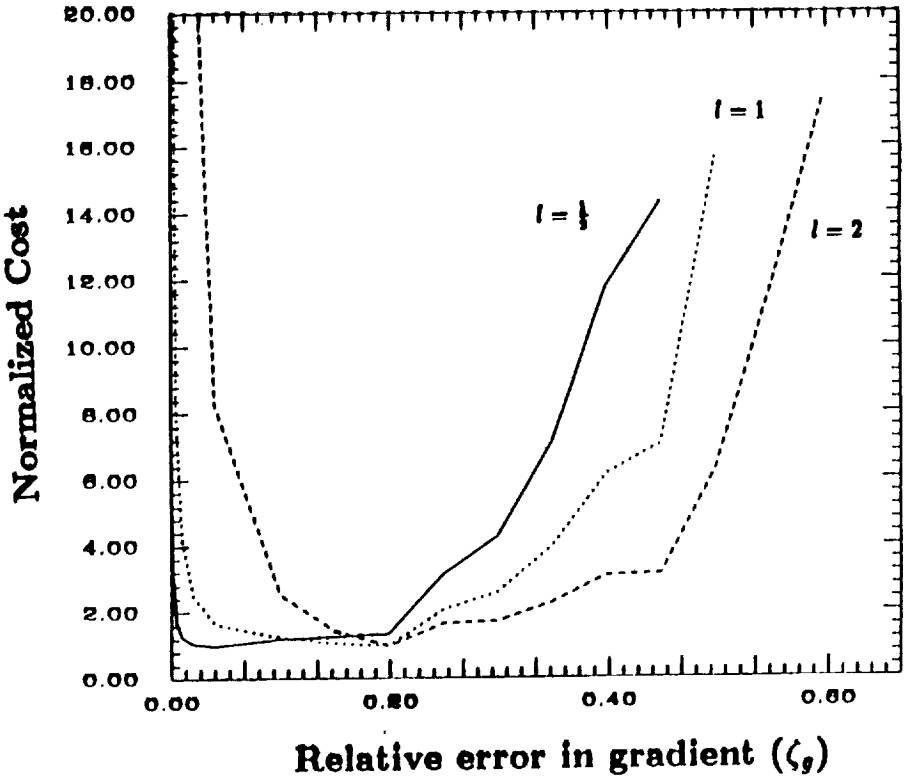

Figure 6: Computational cost profiles for the Watson Test Function.

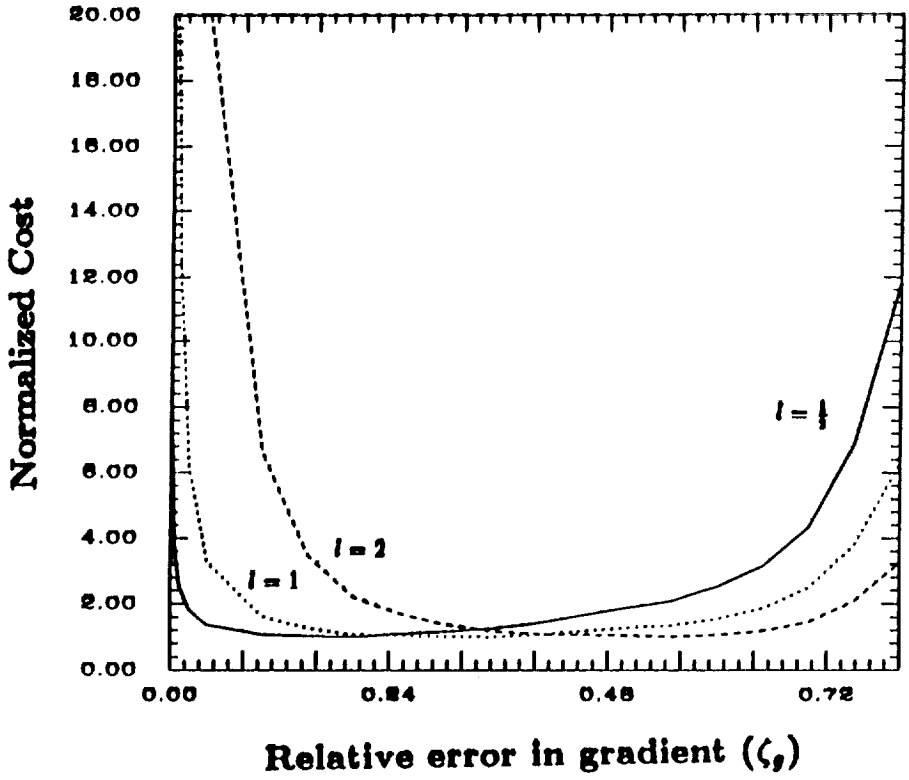

Figure 7: Computational cost profiles for the Brown and Dennis Test Function.

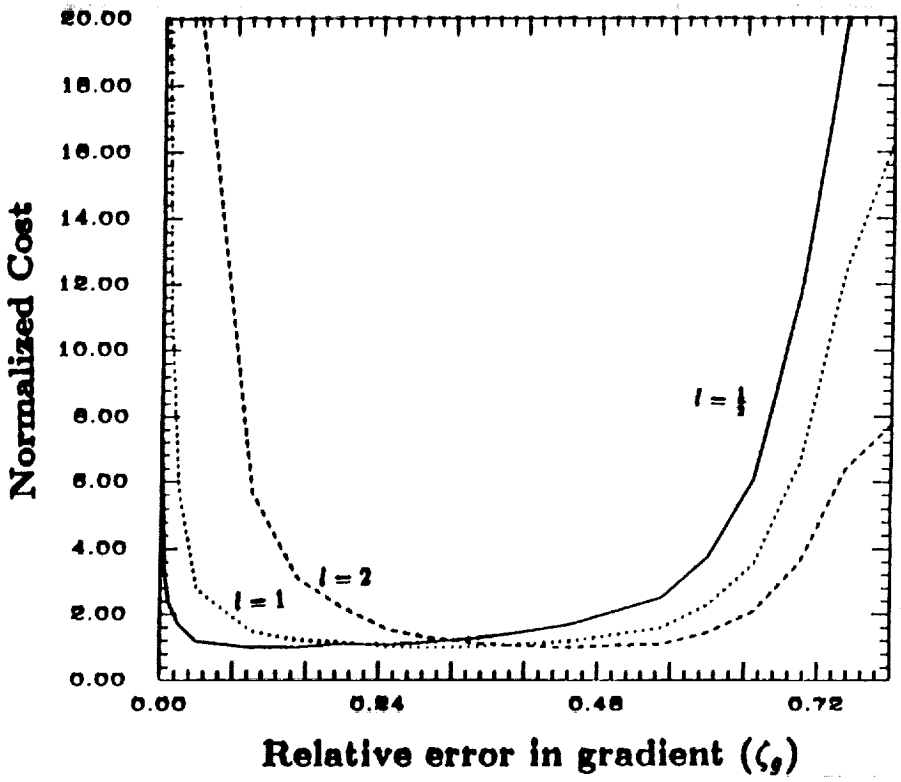

Figure 8: Computational cost profiles for the Extended Powell Singular Test Function.

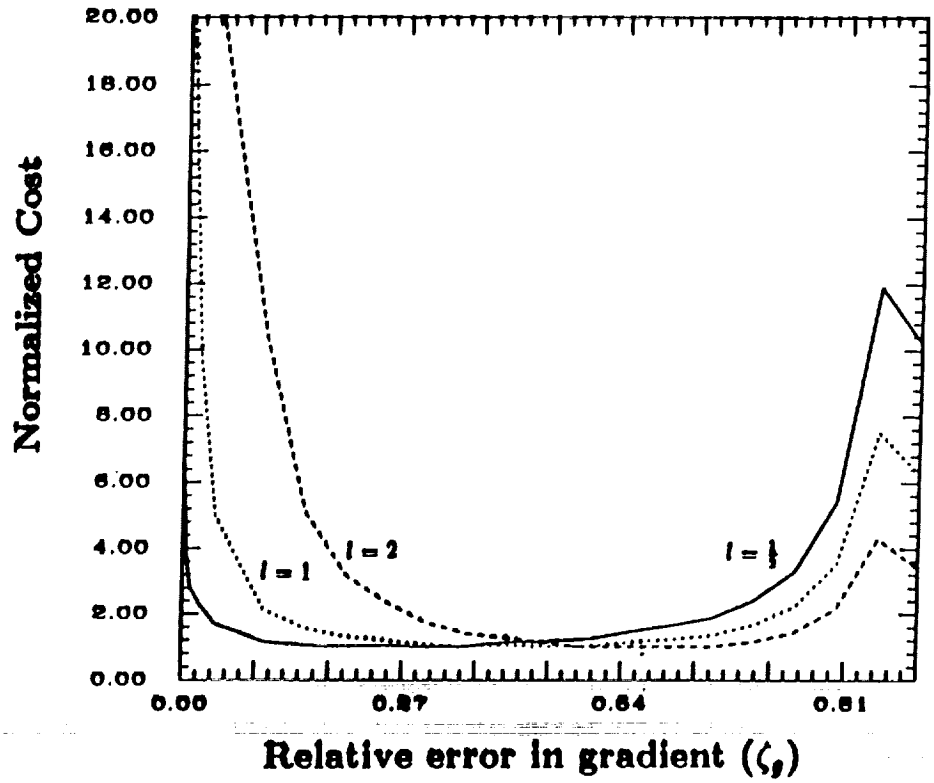

Figure 9: Computational cost profiles for the Gaussian Test Function.

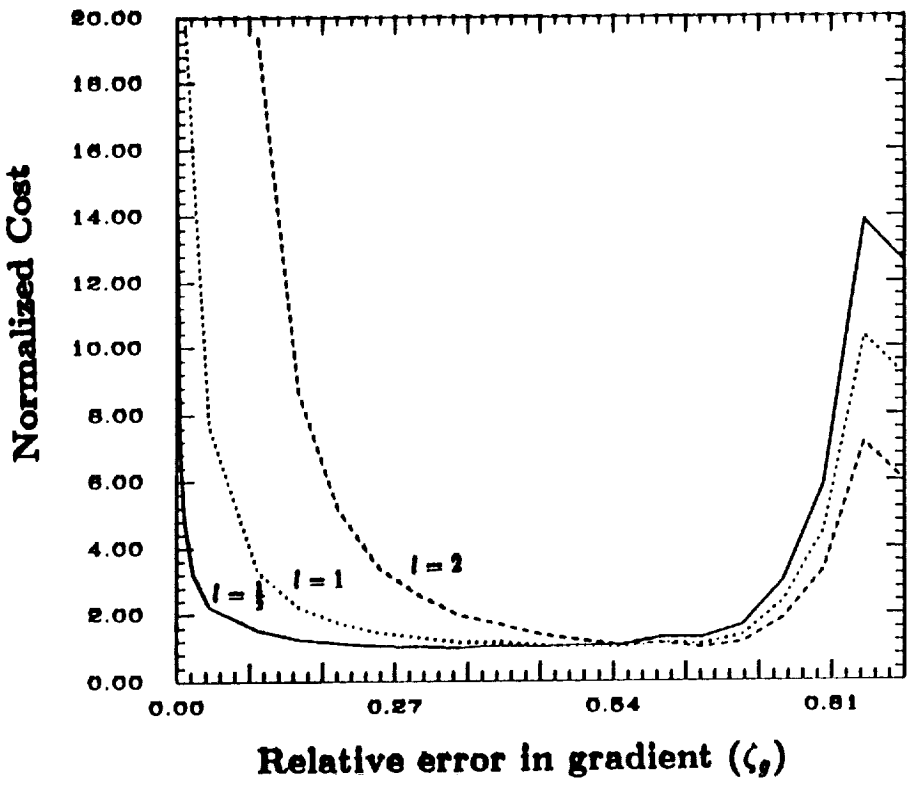

Figure 10: Computational cost profiles for the Trigonometric Test Function.

combining the idealizations (25) and (26) into

$$\text{CPU} \approx K(0)c_2\zeta_g^{-l}\exp(b\zeta_g) \tag{27}$$

yields the theoretical "best" value

$$\zeta_g^* = \frac{l}{b}. \tag{28}$$

Using the observed "typical" value $b = 4$ yields the rule of thumb choice

$$\zeta_g^* = \frac{l}{4}, \tag{29}$$

which worked quite well for all of our test problems.

A number of variations on these numerical tests were also tried. Rather than (26) we also considered the idealization

$$\text{CPU/iteration} \approx c_2 \zeta_g^{-l} + c_3, \tag{30}$$

where c_3 represents a fixed "overhead" cost per iteration. For moderate values of c_3, the character of the overall computational cost behavior remained unchanged. Errors in function values were also considered with ζ_g fixed at 0.1. The algorithm proved to be quite insensitive to these errors for $\zeta_{f,1} < 0.5$. Indeed, the algorithm works in practice even if $\zeta_{f,1} > 1$ provided the *average* value of $|\text{ared}_k(s_k) - \text{cred}_k(s_k)| \ / \ \text{pred}_k(s_k)$ is sufficiently less than $1 - \eta_2 - \zeta_g$. As pointed out in [4], this is a very reasonable result since function values are only used to update the trust radii, and a mistake at any given iteration will not cause the algorithm to fail.

4 Algorithm performance on a parameter identification problem

The test problems of the last section are widely recognized as representing "typical " optimization problems, and because they were algebraically defined, we were able to run an enormous number of test cases to examine the ranges of possible behavior in the presence of errors. It should be remembered, however, that our interpretation of these results rests both on the character of the synthetic noise added to each gradient evaluation and on idealization (26). In order to verify our results, we also tested our algorithm on the following parameter identification problem from [10] and [11].

Consider the accidental release of the radioactive gas tritium into an enclosure surrounding a nuclear reactor. The tritium will react with water vapor in the containment to produce other tritium-based species via

$$T_2 + H_2O \rightleftharpoons HT + HTO, \tag{31}$$

and some of the HTO may be adsorped into the surface of the containment. This adsorped tritium species represents a significant clean-up problem.

Given the reaction rate constant in (31) and the adsorption and release rates of HTO on the surfaces of the containment, the physical problem can be modeled by a system of four coupled initial value ODEs:

$$Y'(t;x) = h(t, Y(t;x)), Y(0;x) = Y_0, \tag{32}$$

with the components $Y : \Re \times \Re^3 \rightarrow \Re^4$ being species concentrations. Unfortunately, the rate constants are not directly measurable. Maroni *et al* performed an experiment in which a known amount of tritium was introduced into a small enclosure and the total tritium concentration $Y_1 + Y_2 + Y_3$ was measured at m discrete time points. The rate constants $x_1, x_2,$ and x_3 can then be estimated by minimizing $f : \Re^4 \rightarrow \Re$ with

$$f(x) = \frac{1}{2}\sum_{i=1}^{m}\left[\sum_{j=1}^{3} Y_j(t_i, x) - O_i + x^4\right]^2 /(O_i)^2, \tag{33}$$

where O_i is the observed experimental concentration

$$O_i = Y_1(t_i) + Y_2(t_i) + Y_3(t_i), \tag{34}$$

and x^4 is an additional variable representing an unknown experimental bias in the instrument for measuring (34).

Equation (33) is a classical inverse problem. Note that each function evaluation for a given iterate x_k involves the numerical solution of four coupled ODEs. Gradient values can be computed via finite differences, or by the numerical solution of a system of sixteen coupled ODEs. Although the latter technique is usually preferable in practice, we used the more difficult approach of estimating g_k by finite differences so that we could investigate a techniques suggested in [3] for estimating and controlling gradient error.

Our numerical experiments with (33) were designed as follows. In order to approximate f at a given point x_k, (32) was solved using ODEPACK [7], which uses an adaptive solution technique. An important feature of ODEPACK is that it allows a desired level of accuracy (either absolute or relative) to be prespecified for each component of Y. In order to achieve

an accuracy of, say, ε_k in $f(x_k)$, we specified a desired accuracy of $\alpha_{k-1}\varepsilon_k$ in each component of Y, where α_{k-1} was the amount of accuracy lost due to cancellation in evaluating (33) at the last iteration. This simple procedure worked remarkably well: the actual error in f_k was typically in the range $[1/10\varepsilon_k, 2\varepsilon_k]$ in our preliminary tests of this technique.

Each gradient was initially approximated by a central difference formula using $2n$ extra function evaluations, where each function evaluation was computed with a specified desired relative accuracy of $\overline{\varepsilon}_k$. Denote this approximation $\overline{g}_k$. We then computed a more accurate estimate of the directional derivative of f in the direction $\overline{g}_k$ (or $(D_k^T D_k)^{-1}\overline{g}_k$ if the scaling matrix is not the identity) as suggested in [3], using the formula

$$\overline{d}_k = \frac{1}{2\delta_k}(f(x_k + \delta_k \overline{g}_k) - f(x_k - \delta_k \overline{g}_k)). \tag{35}$$

Each function evaluation in (35) was computed with a specified desired relative accuracy of $\overline{\varepsilon}_k/10$. The perturbation length δ_k was taken to be

$$\delta_k = \left(\frac{\overline{\varepsilon}_k}{10}\right)^{\frac{1}{3}} |f_k|/g_k^T g_k \,, \tag{36}$$

a value expected to perturb two-thirds of the accurate digits of f. Using (18) and (35), we can then estimate the error term in (21) via

$$\frac{e_k^T \overline{g}_k}{\overline{g}_k^T \overline{g}_k} = 1 - \frac{\nabla f(x_k)^T \overline{g}_k}{\overline{g}_k^T \overline{g}_k} \approx 1 - \frac{\overline{d}_k}{\|\overline{g}_k\|^2}. \tag{37}$$

If this error was significantly larger than the desired gradient error level ζ_g, then $\overline{\varepsilon}_k$ was decreased before the next iteration; if it was significantly smaller $\overline{\varepsilon}_k$ was immediately decreased and $\overline{g}_k$ was recomputed (in practice this seldom occurred except at the first gradient computation .) Figure (11) shows the agreement between actual and requested gradient error. Even with the simple procedures used to adjust $\overline{\varepsilon}_k$, the error estimate given by (35) and (37) allows us to control, with reasonable certainty, the level of accuracy in $\overline{g}_k$.

The approximation $\overline{g}_k$ can be further improved at no additional cost by setting

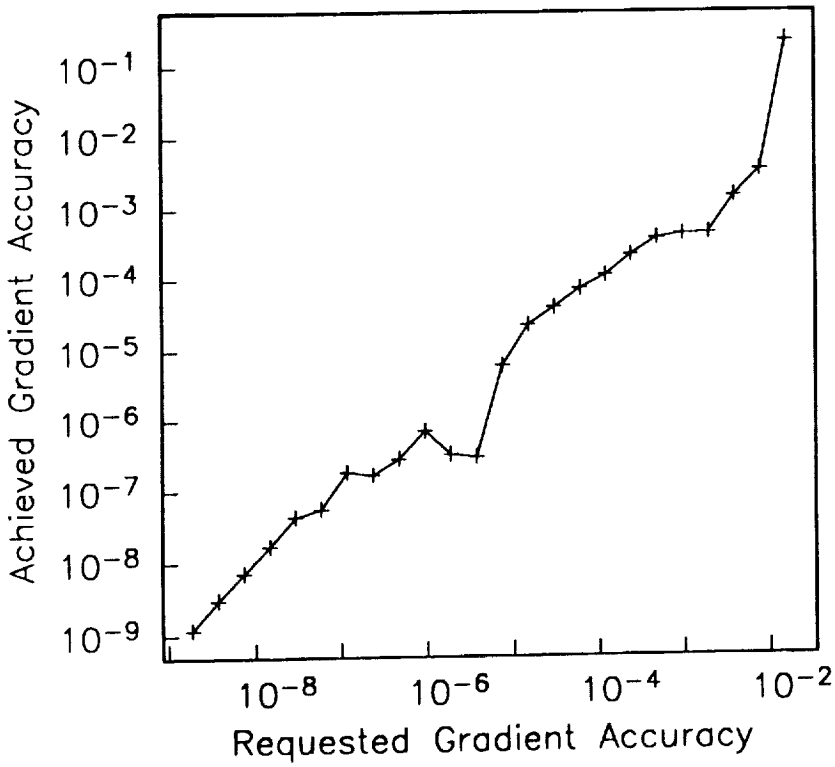

Figure 11: Actual versus Commanded error in the gradient for the Tritium Parameter Estimation Problem.

$$g_k = \frac{\overline{d}_k}{\|\overline{g}_k\|^2}\overline{g}_k \tag{38}$$

so that

$$\frac{e_k^T g_k}{g_k^T g_k} = 1 - \frac{\nabla f(x_k)^T \overline{g}_k}{\overline{d}_k}. \tag{39}$$

Figure (12) shows the CPU time required to achieve a given level of accuracy using (38) in addition to the previously discussed procedure for adjusting $\overline{\varepsilon}_k$. We see that computational expense increases geometrically as accuracy increases. [2]

Given these methods of evaluating f_k and g_k to some specified accuracy, we recorded the

[2]The idealised cost profile (26) yields a very close fit to this plot if l is taken to be $\frac{1}{10}$. However, tests with different values of x_k showed that better empirical form this problem is

$$\text{CPU/gradient evaluation} \approx c_2\|g_k\|^{-1}\zeta_g^{-1/10}.$$

Nonetheless, the rule-of-thumb choice (29) with $l = \frac{1}{10}$ proved to be close to the optimal selection in our numerical tests.

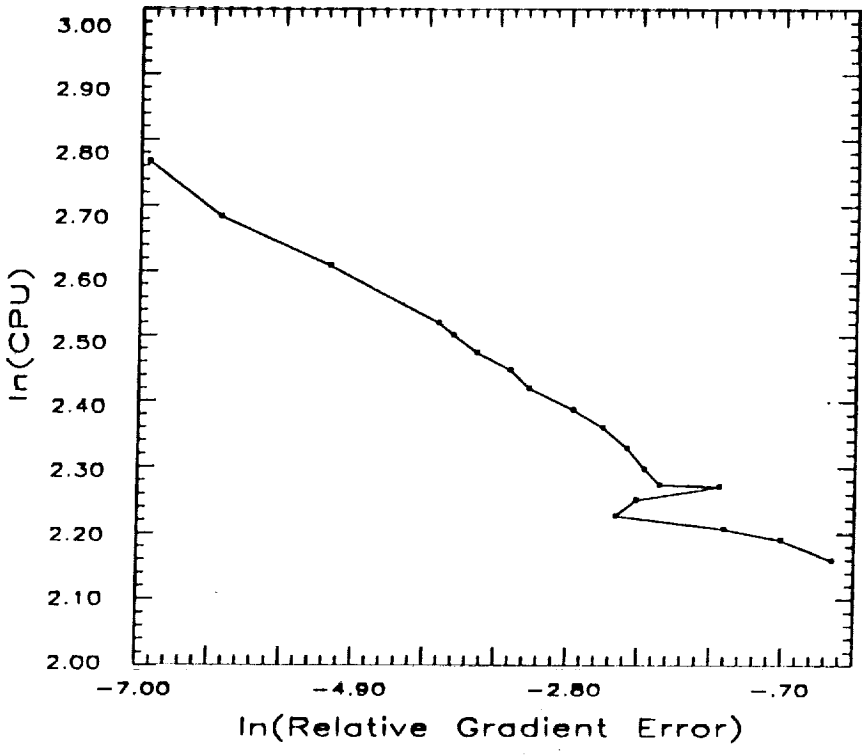

Figure 12: Solution time required for one gradient evaluation in Tritium parameter identification problem.

computational time required to solve (33) for a number of different values of ζ_g, and for the following 3 cases.

1. Each f_k value was computed to high relative accuracy (10^{-8}), and each g_k value was computed as previously described *including* the correction (38).

2. Each f_k was computed to high relative accuracy (10^{-8}), and each g_k was computed as described previously but *without* doing correction (38).

3. Each f_k was computed using Procedure 2 with $\zeta_{f,1} = 0.1$ and $\zeta_{f,2} = 0.99$, and each g_k value was computed as previously described *including* correction (38).

A somewhat different implementation of the trust region method was used rather than the Dennis-Schnabel code used in the last section. First, we included nonnegativity constraints on the first three components of x to be consistent with the physics of the problem. This

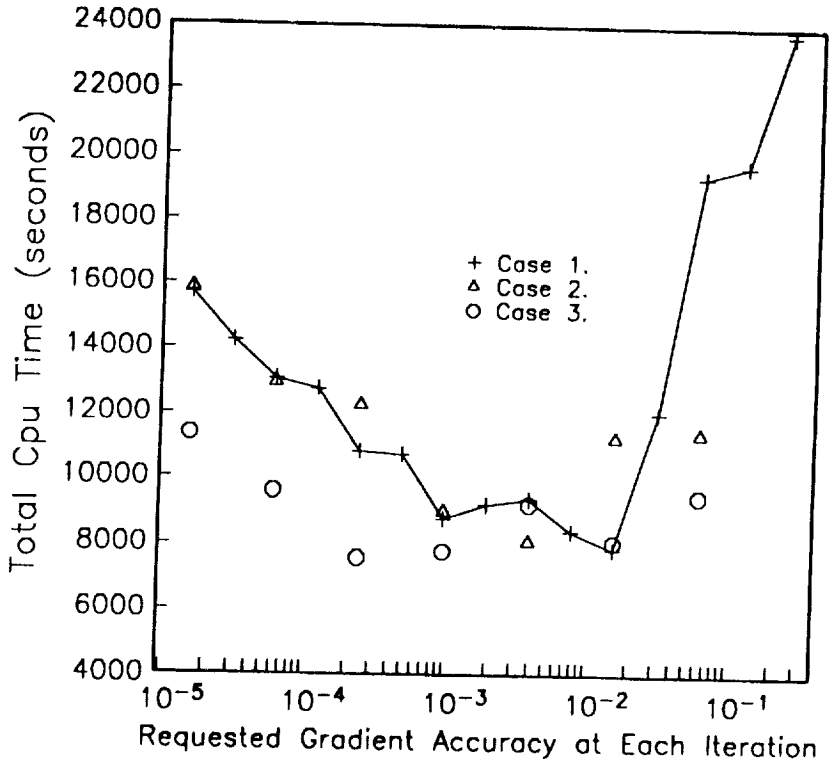

Figure 13: Solution time for numerical optimization of the Tritium parameter identification problem.

was done by replacing the ellipsoidal trust region in (4) with the rectangular trust region $\|D_k s\|_\infty \leq \Delta_k$, and by using a quadratic programming code to exactly solve the trust region subproblem subject to the nonnegativity constraints on x. Second, we used the Hessian safeguarding techniques proposed in [2] in addition to the standard BFGS update (5). Third, we used a simplified stopping criteria for comparison purposes. Each test case was terminated when $f_k - f^* \leq \frac{1}{100}(f_0 - f^*)$, where f^* was the optimal value of f.

Figure 13 shows the results of our tests.

Case 1 was tested for 15 different values of ζ_g. Note that the total computational time required is less than 8000 seconds for $\zeta_g = 0.15$, but rises to almost 16000 and 24000 seconds for $\zeta_g = 1.5 \times 10^{-8}$ and $\zeta_g = 0.25$, respectively. Fewer data were collected for cases 2 and 3, but note that the correction (38) appears to make little difference to the algorithm when $e_k^T g_k / g_k^T g_k$ is small. On the other hand, using Procedure 2 rather than computing each f_k to a fixed accuracy of 10^{-8} resulted in a moderately faster algorithm.

In addition to the above 3 cases, a number of other numerical tests were made. Rather than keeping ζ_g fixed throughout the algorithm, we tried setting

$$\zeta_g^{k+1} = \begin{cases} 1/10 & \text{if } k = 0 \\ max\{\zeta_g^k/2, 10^{-5}\} & otherwise, \end{cases} \tag{40}$$

or conversely

$$\zeta_g^{k+1} = \begin{cases} 10^{-5} & \text{if } k = 0 \\ min\{2\zeta_g^k\ ,\ 1/10\} & otherwise. \end{cases} \tag{41}$$

The idea behind (40) is to try to obtain the fast local convergence properties of the BFGS method when accurate gradients are available, while the idea behind (41) is to avoid highly accurate gradient approximations near the solution where they are likely to be most expensive. Interestingly, both of these approaches performed similarly, requiring 8887 and 10647 seconds, respectively.

Although the correction (38) appears to be of little use when $e_k^T g_k / g_k^T g_k$ is already small, it does appear to be useful in preventing the algorithm from failing due to occasionally encountering highly inaccurate gradient evaluations. In tests where a large synthetic error was added to the gradient approximation every p iterations, the algorithm was much more robust when (38) was used (although the algorithm did still have problems involving convergence to a point with $g_k = 0$ and $\nabla f(x_k) \neq 0$, as predicted in [4].) In a similar vein, the gradient accuracy test given by (35) and (37) should be useful in verifying the accuracy of analytically derived gradients.

5 Summary

We have examined the numerical behavior of trust region algorithms for nonlinear optimization when function and gradient values are not computed exactly. This class of algorithms has proven remarkably robust, and can be successfully implemented even with very large errors in the function and gradient evaluations.

In a large number of tests using standard test problems with synthetically induced gradient errors, we observed that the algorithm performance, as measured by the number of iterations required for convergence, tends to degrade exponentially as the relative gradient error increases. This is a telling argument for using accurate evaluations provided they can be obtained at reasonable expense. For many optimization/simulation problems, however, the computational expense of these evaluations rises sharply with increasing accuracy, and low accuracy evaluations are again attractive. A good choice for the amount of relative gradient error allowed in the algorithm can result in orders-of-magnitude savings in computational cost. If (26) holds, then the choice $\zeta_g = l/4$ was nearly optimal for all of our test problems.

Using a parameter estimation problem based on the numerical solution of a system of ODEs, we tested a technique for estimating and controlling the amount of error in a gradient approximation. This technique was very successful when used in conjunction with the "user specified accuracy" feature in the numerical differential equation solver ODEPACK. Actual computational costs for various values of relative gradient error were examined to confirm the behavior observed in the test problems with synthetically induced errors.

References

[1] T.M. Apostol. *Mathematical Analysis.* Addison-Wesley, Reading, Massachusetts, 1957.

[2] R.G. Carter. Safeguarding Hessian approximations in trust region algorithms. Technical Report TR87-06, Rice University, Dept. of Mathematical Sciences, Revised October 1988.

[3] R.G. Carter. On the global convergence of trust region algorithms using inexact gradient information. Technical Report TR87-12, Rice University, Dept. of Mathematical Sciences, Revised April 1989.

[4] R.G. Carter. Numerical optimization in Hilbert space using inexact function and gradient information. Technical Report 89-45 , Institute for Computer Applications in Science and Engineering, June 1989.

[5] J.E. Dennis Jr. and R.B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

[6] D.M. Gay. Computing optimal locally constrained steps. *SIAM J. Sci. Statist. Comput.*, 2:186–197, 1981.

[7] A.C. Hindmarsh. ODEPACK, a systematized collection of ODE solvers. In *Scientific Computing*, R. S. Stepleman et al, editor, pages 55–64. North-Holland, Amsterdam, 1983.

[8] J.D. Lambert. *Computational methods in ordinary differential equations.* John Wiley and Sons, New York, 1973.

[9] J.N. Lyness. Remarks about performance profiles. Technical Memorandum 369, Applied Mathematics Division, Argonne National Laboratory, 1981.

[10] V.A. Maroni, R.H. Land, and M. Minkoff. TSOAK-M1: A computer code used to determine tritium reaction/adsorption/release parameters from experimental results of air-detritiation tests. Report ANL-79-82, Argonne National Laboratory, 1979.

[11] M. Minkoff. Approaches to optimization/simulation problems. *Appl. Numer. Math.*, 3:453–466, 1987.

[12] J.J. Moré. Recent developments in algorithms and software for trust region methods. In *Mathematical Programming: State of the Art*, A. Bachem, M. Grőtschel, and B.Korte, editors, pages 258–287. Springer Verlag, Berlin, 1983.

[13] J.J. Moré, B.S. Garbow, and K.E. Hillstrom. Fortran subroutines for testing unconstrained optimization software. *TOMS*, 7:136–140, 1981.

[14] J.J. Moré, B.S. Garbow, and K.E. Hillstrom. Testing unconstrained optimization software. *TOMS*, 7:17–41, 1981.

[15] M.J.D. Powell. A new algorithm for unconstrained optimization. In *Nonlinear Programming*, J.B. Rosen, O.L. Mangasarian, and K. Ritter, editors, pages 31–65. Academic Press, London, 1970.

[16] G.A. Schultz, R.B. Schnabel, and R.H. Byrd. A family of trust-region-based algorithms for unconstrained minimization with strong global convergence properties. *SIAM J. Numer. Anal.*, 22:47–67, 1985.

[17] Ph. L. Toint. Global convergence of a class of trust region methods for nonconvex minimization in Hilbert space. Technical Report 87/6, Department of Mathematics, Facultés Universitaires ND de la Paix, Namur, Belgium, 1987.

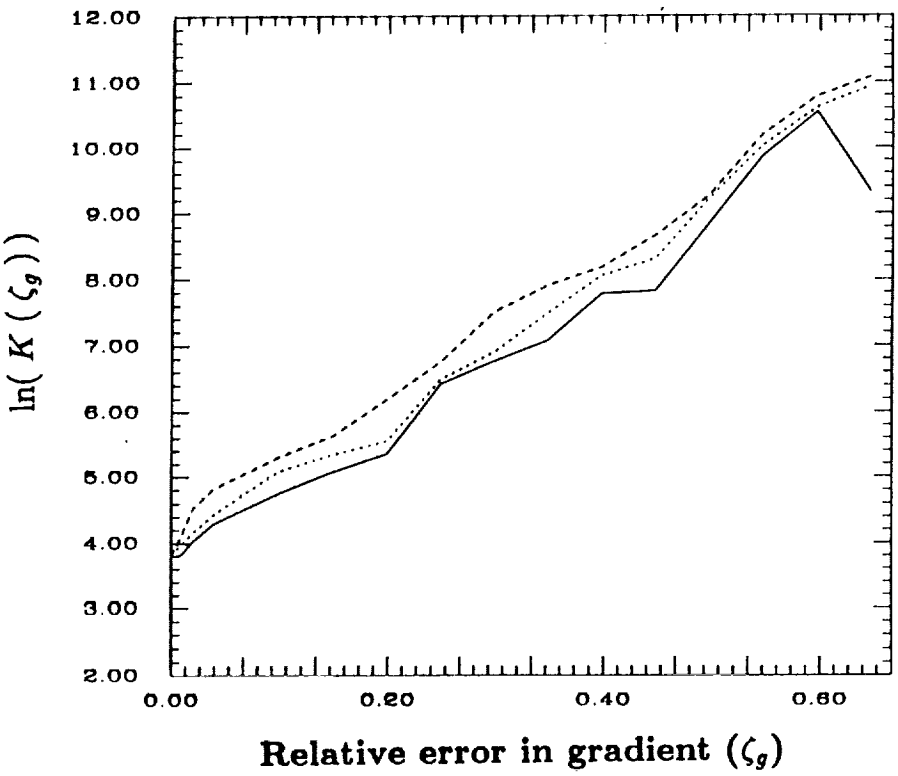

Figure 1: Number of iterations versus ζ_g for the Watson Test Function.

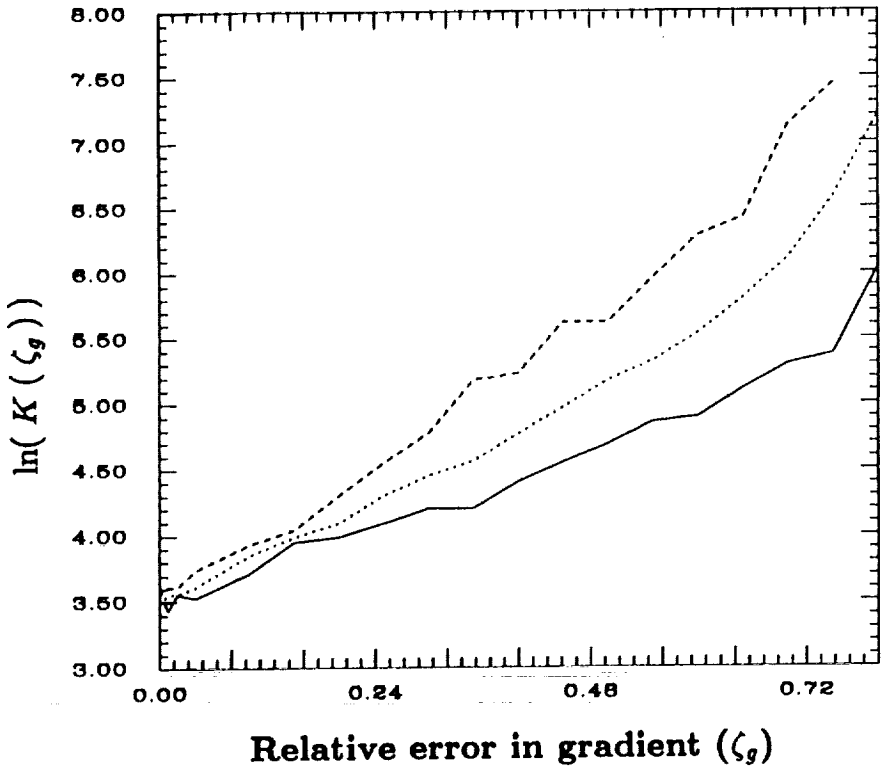

Figure 2: Number of iterations versus ζ_g for the Brown and Dennis Test Function.

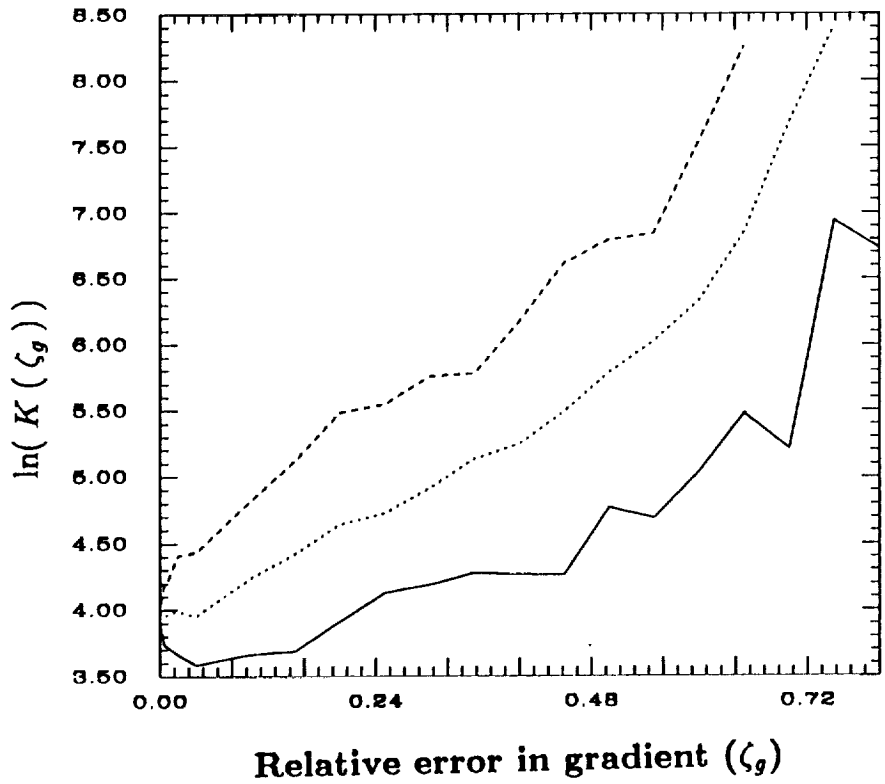

Figure 3: Number of iterations versus ζ_g for the Extended Powell Singular Test Function.

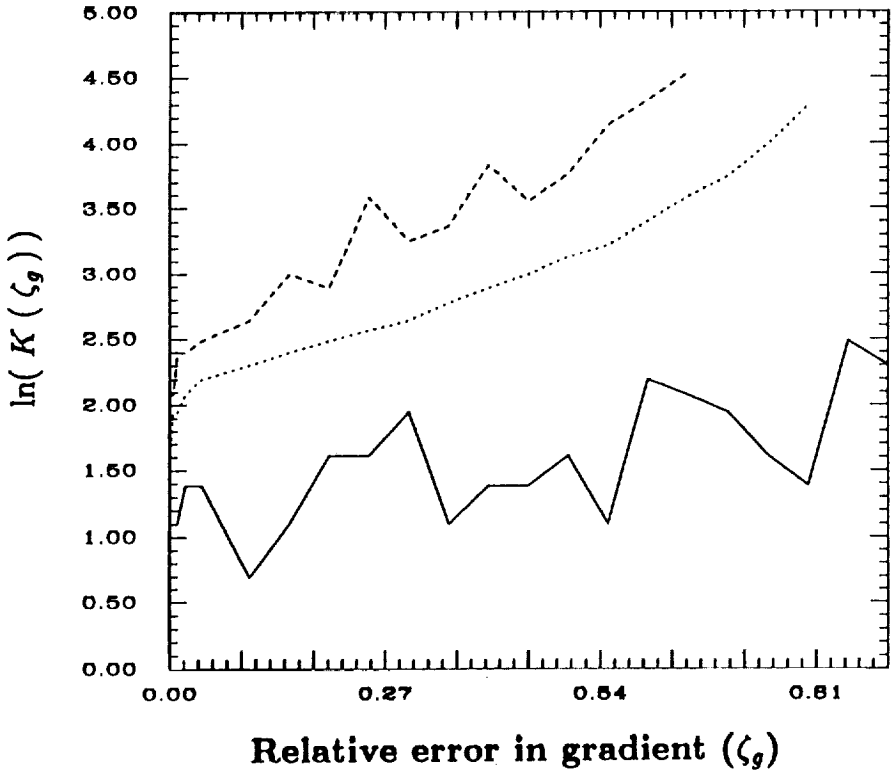

Figure 4: Number of iterations versus ζ_g for the Gaussian Test Function.

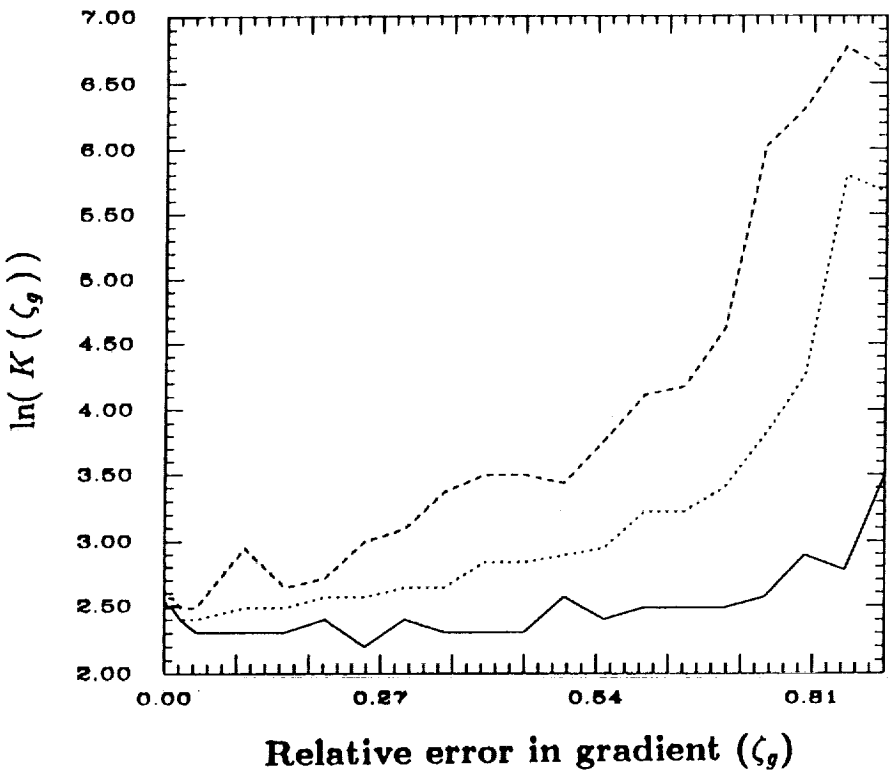

Figure 5: Number of iterations versus ζ_g for the Trigonometric Test Function.

against using low accuracy gradients if high accuracy evaluations are obtainable without greatly increased computational expense, low accuracy evaluations are still attractive in cases where the computational expense increases rapidly with increasing accuracy. Suppose, for instance, that

$$\text{CPU/iteration} \approx c_2 \zeta_g^{-l} \tag{26}$$

for some constants c_2 and l. The total computational expense for solving a given problem will then be proportional to $\zeta_g^{-l} K(\zeta_g)$. Figures (6) through (10) show the predicted total computational cost of the median curves for $K(\zeta_g)$ in figures (1) through (5) for the values $l = \frac{1}{2}, 1$, and 2 (with each curve normalized so that the minimum value is 1.0).

Notice that each curve of total computational cost increases very rapidly as $\zeta_g \to 0$ or $\zeta_g \to 1$, and has a relatively large, flat minima. This behavior holds for both the "typical" cases (figures (6) through (8)) and the anomalous cases (figures (9) and (10)). Interestingly,

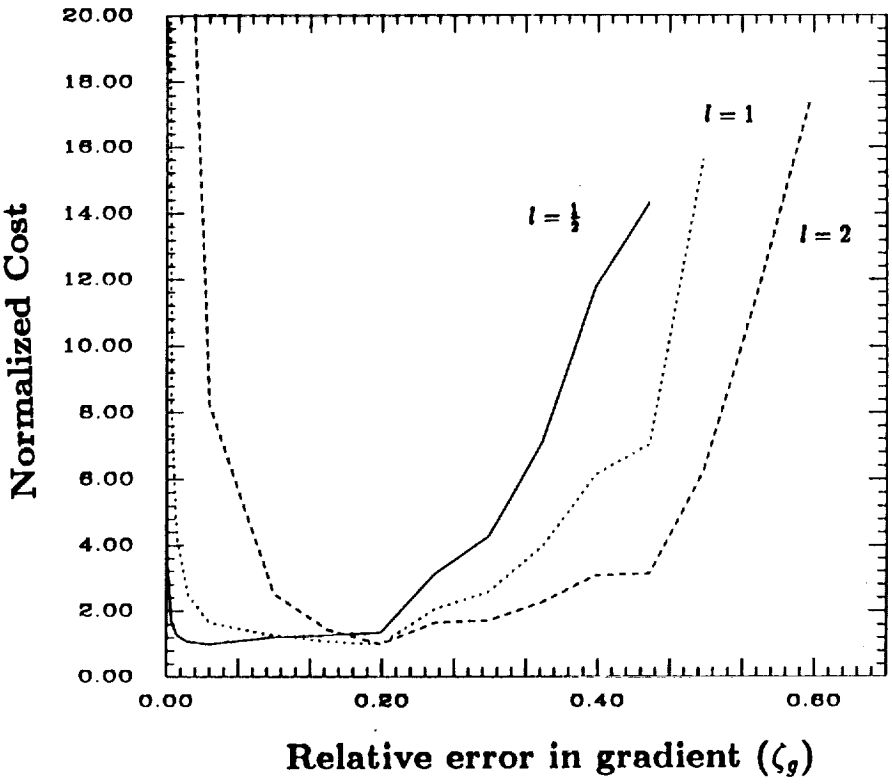

Figure 6: Computational cost profiles for the Watson Test Function.

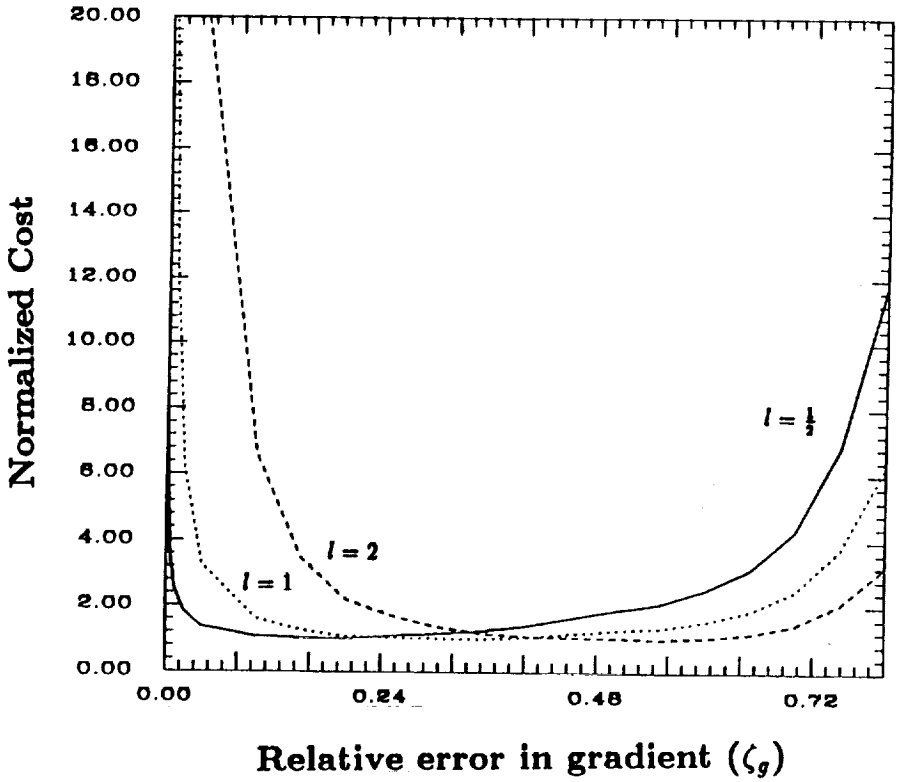

Figure 7: Computational cost profiles for the Brown and Dennis Test Function.

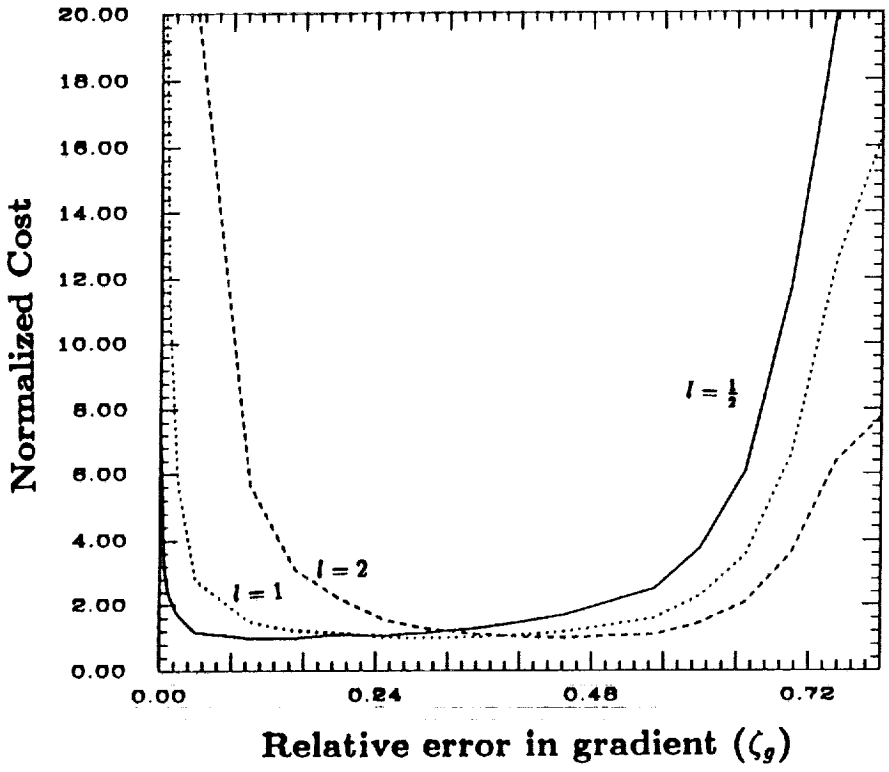

Figure 8: Computational cost profiles for the Extended Powell Singular Test Function.

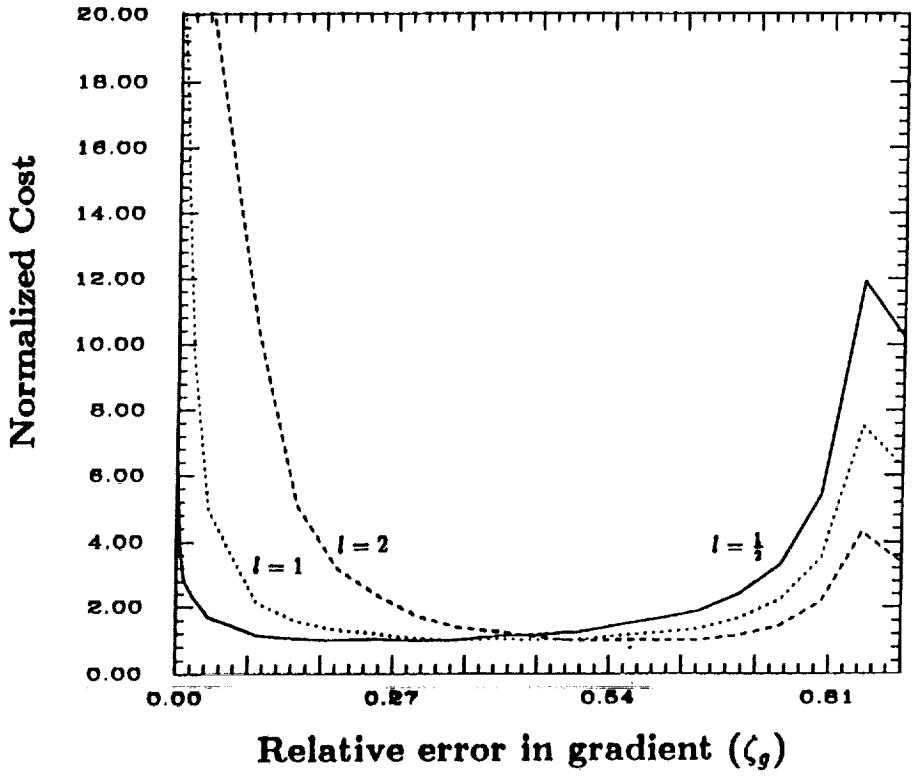

Figure 9: Computational cost profiles for the Gaussian Test Function.

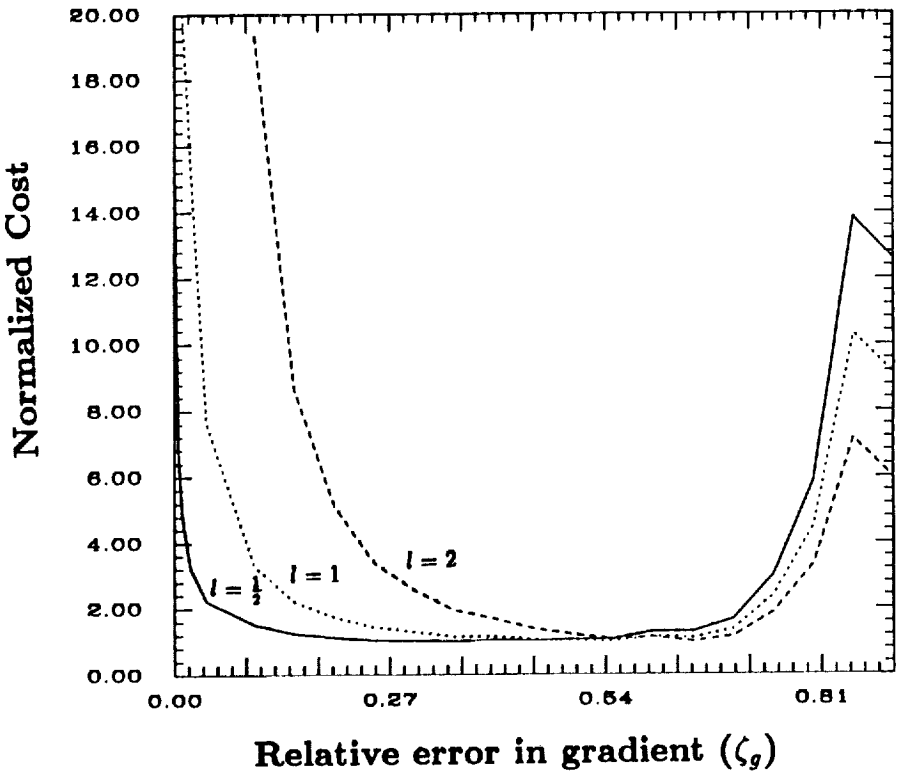

Figure 10: Computational cost profiles for the Trigonometric Test Function.

combining the idealizations (25) and (26) into

$$\text{CPU} \approx K(0)c_2\zeta_g^{-l}\exp(b\zeta_g) \tag{27}$$

yields the theoretical "best" value

$$\zeta_g^* = \frac{l}{b}. \tag{28}$$

Using the observed "typical" value $b = 4$ yields the rule of thumb choice

$$\zeta_g^* = \frac{l}{4}, \tag{29}$$

which worked quite well for all of our test problems.

A number of variations on these numerical tests were also tried. Rather than (26) we also considered the idealization

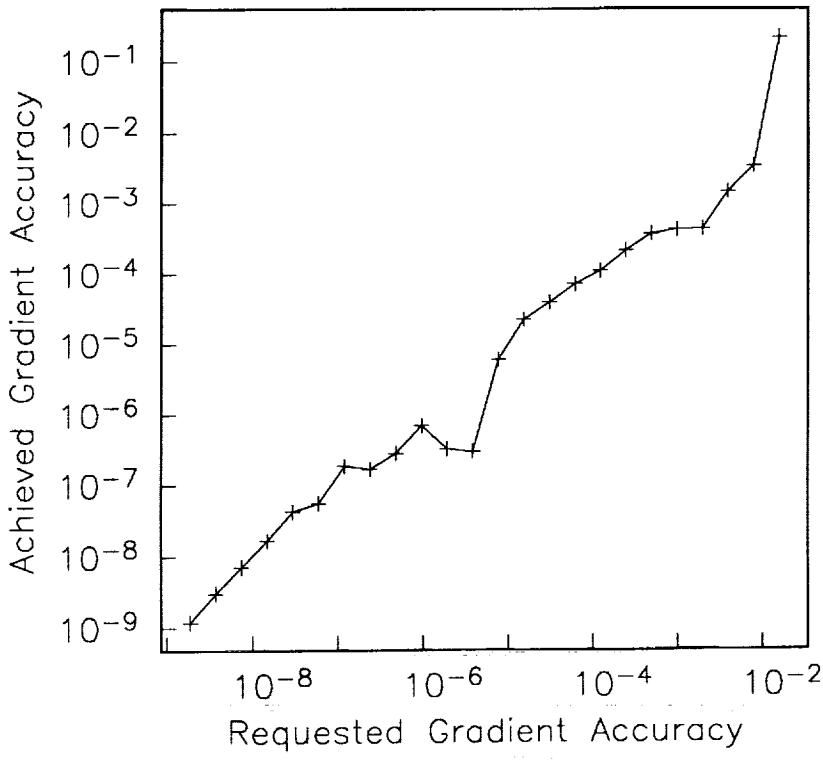

Figure 11: Actual versus Commanded error in the gradient for the Tritium Parameter Estimation Problem.

$$g_k = \frac{\overline{d}_k}{\|\overline{g}_k\|^2}\overline{g}_k \tag{38}$$

so that

$$\frac{e_k^T g_k}{g_k^T g_k} = 1 - \frac{\nabla f(x_k)^T \overline{g}_k}{\overline{d}_k}. \tag{39}$$

Figure (12) shows the CPU time required to achieve a given level of accuracy using (38) in addition to the previously discussed procedure for adjusting $\overline{\varepsilon}_k$. We see that computational expense increases geometrically as accuracy increases. [2]

Given these methods of evaluating f_k and g_k to some specified accuracy, we recorded the

[2]The idealized cost profile (26) yields a very close fit to this plot if l is taken to be $\frac{1}{10}$. However, tests with different values of x_k showed that better empirical form this problem is

$$\text{CPU/gradient evaluation} \approx c_2\|g_k\|^{-1}\zeta_g^{-1/10}.$$

Nonetheless, the rule-of-thumb choice (29) with $l = \frac{1}{10}$ proved to be close to the optimal selection in our numerical tests.

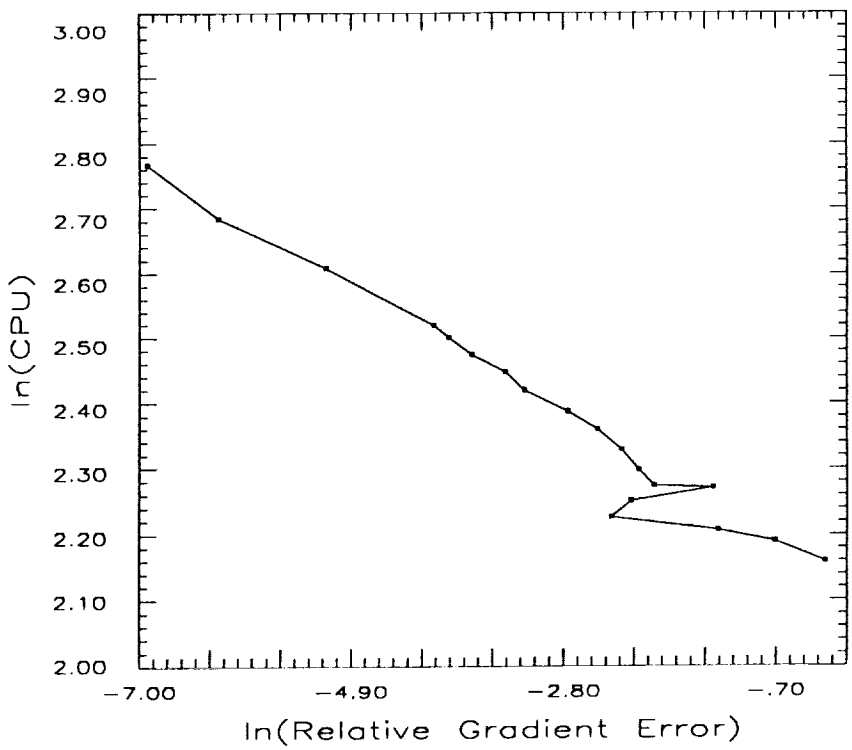

Figure 12: Solution time required for one gradient evaluation in Tritium parameter identification problem.

computational time required to solve (33) for a number of different values of ζ_g, and for the following 3 cases.

1. Each f_k value was computed to high relative accuracy (10^{-8}), and each g_k value was computed as previously described *including* the correction (38).

2. Each f_k was computed to high relative accuracy (10^{-8}), and each g_k was computed as described previously but *without* doing correction (38).

3. Each f_k was computed using Procedure 2 with $\zeta_{f,1} = 0.1$ and $\zeta_{f,2} = 0.99$, and each g_k value was computed as previously described *including* correction (38).

A somewhat different implementation of the trust region method was used rather than the Dennis-Schnabel code used in the last section. First, we included nonnegativity constraints on the first three components of x to be consistent with the physics of the problem. This

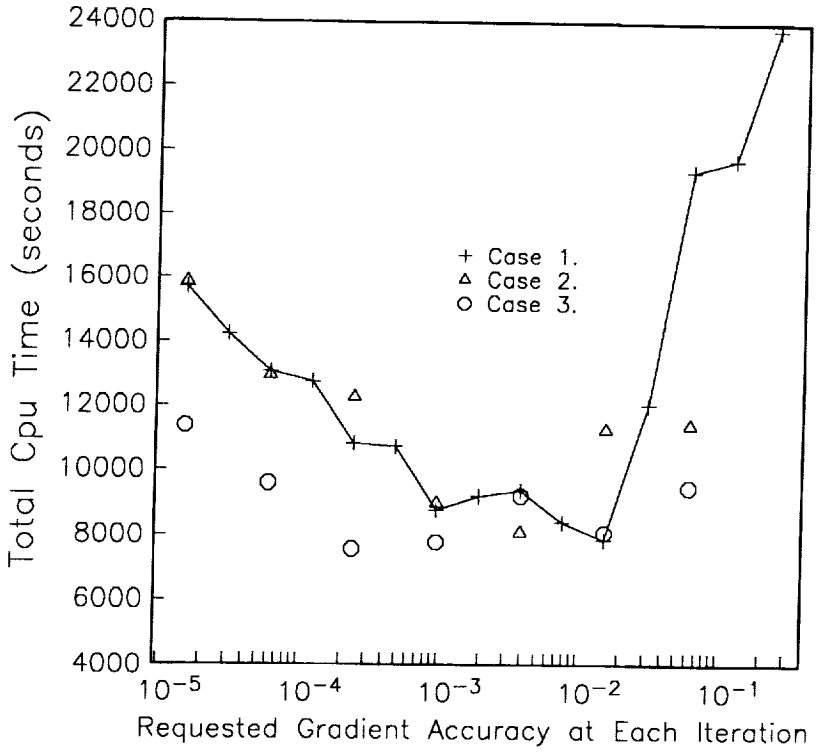

Figure 13: Solution time for numerical optimization of the Tritium parameter identification problem.

was done by replacing the ellipsoidal trust region in (4) with the rectangular trust region $\|D_k s\|_\infty \leq \Delta_k$, and by using a quadratic programming code to exactly solve the trust region subproblem subject to the nonnegativity constraints on x. Second, we used the Hessian safeguarding techniques proposed in [2] in addition to the standard BFGS update (5). Third, we used a simplified stopping criteria for comparison purposes. Each test case was terminated when $f_k - f^* \leq \frac{1}{100}(f_0 - f^*)$, where f^* was the optimal value of f.

Figure 13 shows the results of our tests.

Case 1 was tested for 15 different values of ζ_g. Note that the total computational time required is less than 8000 seconds for $\zeta_g = 0.15$, but rises to almost 16000 and 24000 seconds for $\zeta_g = 1.5 \times 10^{-5}$ and $\zeta_g = 0.25$, respectively. Fewer data were collected for cases 2 and 3, but note that the correction (38) appears to make little difference to the algorithm when $e_k^T g_k / g_k^T g_k$ is small. On the other hand, using Procedure 2 rather than computing each f_k to a fixed accuracy of 10^{-8} resulted in a moderately faster algorithm.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-181927 ICASE Report No. 89-46		

4. Title and Subtitle

NUMERICAL EXPERIENCE WITH A CLASS OF ALGORITHMS FOR NONLINEAR OPTIMIZATION USING INEXACT FUNCTION AND GRADIENT INFORMATION

5. Report Date

June 1989

6. Performing Organization Code

7. Author(s)

Richard G. Carter

8. Performing Organization Report No.

89-46

10. Work Unit No.

505-90-21-01

9. Performing Organization Name and Address

Institute for Computer Applications in Science and Engineering
Mail Stop 132C, NASA Langley Research Center
Hampton, VA 23665-5225

11. Contract or Grant No.

NAS1-18605

13. Type of Report and Period Covered

Contractor Report

12. Sponsoring Agency Name and Address

National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23665-5225

14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor:
Richard W. Barnwell

SIAM Journal on Control and Optimization

Final Report

16. Abstract

For optimization problems associated with engineering design, parameter estimation, image reconstruction, and other optimization/simulation applications, low accuracy function and gradient values are frequently much less expensive to obtain than high accuracy values. We investigate the computational performance of trust region methods for nonlinear optimization when high accuracy evaluations are unavialable or prohibitively expensive, and confirm earlier theoretical predictions than the algorithm is convergent even with relative gradient errors of 0.5 or more. The proper choice of the amount of accuracy to use in function and gradient evaluations can result in orders-of-magnitude savings in computational cost.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
nonconvex unconstrained optimization, inexact functions, inexact gradients, trust region methods, global convergence	64 - Numerical Analysis Unclassified - Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	36	

NASA FORM 1626 OCT 86