On the Geometry Phase in Model-Based Algorithms for Derivative-Free Optimization

Giovanni Fasano* José Luis Morales† Jorge Nocedal‡

March 14, 2008

Abstract

A numerical study of model-based methods for derivative-free optimization is presented. These methods typically include a geometry phase whose goal is to ensure the adequacy of the interpolation set. The paper studies the performance of an algorithm that dispenses with the geometry phase altogether (and therefore does not attempt to control the position of the interpolation set). Data is presented describing the evolution of the condition number of the interpolation matrix and the accuracy of the gradient estimate. The experiments are performed on smooth unconstrained optimization problems with dimensions ranging between 2 and 15.

*Dipartamento di Matematica Applicata Università di Venezia. Venezia, Italy. INSEAN-Italian Ship Model Basin, Rome, Italy. This author thanks Programma di Ricerca INSEAN 2007-2009.

†Departamento de Matemáticas, Instituto Tecnológico Autónomo de México, México. This author was supported by Asociación Mexicana de Cultura AC and CONACyT-NSF grant J110.388/2006.

‡Department of Electrical Engineering and Computer Science, Northwestern University, USA. This author was supported by National Science Foundation grant CCF-0514772 and Department of Energy grant DE-FG02-87ER25047-A004.

1 Introduction

Model-based methods have proved to be effective techniques for solving unconstrained optimization problems for which derivatives are not available [15, 2, 3, 4, 16, 10, 21, 1, 7, 14, 6]. Some of these methods construct quadratic models of the objective function by interpolating a set of function values, and compute steps by minimizing the models inside a trust region. For the model to be well defined, the interpolation points must be *poised* [5, 16], meaning that they must be compatible with the interpolation conditions imposed on them. In practice, it is common for the interpolation points to lose poisedness as the iteration progresses, and this is reflected in an increasing ill conditioning in the interpolation system. To prevent difficulties, a *geometry* phase is included in contemporary model-based algorithms; it typically replaces one or more of the interpolation points by other points that are selected with the exclusive goal of improving the position of the interpolation set.

In this paper we consider the effect of dispensing with the geometry phase altogether. We find, to our surprise, that omitting the geometry phase does not seem to harm the efficiency and robustness of the iteration, and that the simple algorithm presented in this paper is competitive with two established model-based algorithms that include a geometry phase. Although high ill conditioning in the interpolation system is observed during the course of the iteration — and the quality of model is often poor at regular intervals — the overall progress achieved by the method appears to be quite satisfactory. The conditioning of the interpolation system grows initially, but then oscillates during much of the run and tends to increase slowly (except possibly very near the solution). We conjecture that a self-correction mechanism may be at play that prevents the conditioning to grow steadily and causes the algorithm to fail.

The observations made in this paper are based on experiments involving 60 smooth test problems with dimensions ranging between 2 and 15. Our results are relevant to the design of algorithms because the geometry phase adds substantial complexity to the algorithm.

2 A Simple Model-Based Algorithm

We limit our discussion to the unconstrained optimization problem

$$\min_{x\in\mathbb{R}^n} f(x), \tag{2.1}$$

where f is a scalar-valued function. Although most model-based methods for derivative-free optimization are able to handle simple types of constraints, the unconstrained setting suffices for our investigation.

We consider an algorithm that approximates the objective function f by means of a quadratic model m_c. To define the model, we assume that at the current iterate x_c we have a set of interpolation points $Y_c = \{y^1, y^2, ..., y^q\}$, with $y^i \in \mathbb{R}^n$, $i = 1, 2, \ldots, q$, where

$$q = \tfrac{1}{2}(n+1)(n+2). \tag{2.2}$$

We assume that x_c is an element of this set and that no point in Y_c has a lower function value than x_c. We write the quadratic model in the form

$$m_c(x_c + p) = d_c + g_c^T p + \tfrac{1}{2} p^T G_c p, \tag{2.3}$$

and determine the scalar d_c, the vector $g_c \in \mathbb{R}^n$, and the symmetric matrix $G_c \in \mathbb{R}^{n\times n}$ by imposing the interpolation conditions

$$m_c(y^\ell) = f(y^\ell), \quad \ell = 1, 2, \ldots, q. \tag{2.4}$$

Since there are $\frac{1}{2}(n+1)(n+2)$ coefficients in the model (2.3), the interpolation conditions (2.4) represent a square system of linear equations in the coefficients d_c, g_c, G_c. If we choose the interpolation points $y^1, y^2, \ldots, y^q$ so that this system is nonsingular, the model m_c will be uniquely determined (we follow Conn, Scheinberg and Vicente [5] and say that points with this property are *poised*).

Once m_c has been formed, we compute a step p_c by approximately solving the trust-region subproblem

$$\min_p \; m_c(x_c + p), \quad \text{subject to} \quad \|p\|_2 \le \Delta_c, \tag{2.5}$$

for some trust-region radius $\Delta_c > 0$. The complete algorithm (which is a simplification of the method described in [10]) is specified as follows.

Algorithm I
Choose the parameters α, $\beta \in (0,1)$, $\gamma < 1$, an initial trust region radius $\Delta_c > 0$, and an initial guess x_c. Select an initial interpolation set $Y_c = \{y^1, y^2, \ldots, y^q\}$ that is poised. (We assume that $x_c \in Y_c$ and that $f(x_c) \le f(y)$ for all $y \in Y_c$).
Repeat until a convergence test is satisfied

1. Form the model (2.3) that interpolates the function f on the interpolation set Y_c;

2. Compute p_c by (approximately) solving subproblem (2.5);

3. Set $\text{ared}(p_c) = f(x_c) - f(x_c + p_c)$ and $\text{pred}(p_c) = m_c(x_c) - m_c(x_c + p_c)$;

4. Update Δ_c:
   If $\text{ared}(p_c) > \alpha\,\text{pred}(p_c)$, set $\Delta_+ = \gamma\Delta_c$,
   else set $\Delta_+ = \beta\Delta_c$.

5. Find a point in Y_c that is farthest from the current iterate (break ties arbitrarily):
   $y^{\ell_{\text{out}}} = \arg\max_{y\in Y_c} \|y - x_c\|_2$;

6. **If** $f(x_c + p_c) < f(x_c)$ (successful iteration)
   Update the current iterate, and include $x_c + p_c$ in the interpolation set, discarding $y^{\ell_{\text{out}}}$:
   a. $x_+ = x_c + p_c$
   b. $Y_+ = \{x_+\} \cup Y_c \setminus \{y^{\ell_{\text{out}}}\}$.
   else (unsuccessful iteration)
   If the new trial point is not further from the current iterate than $y^{\ell_{\text{out}}}$,
   admit it to the interpolations set, discarding $y^{\ell_{\text{out}}}$; otherwise, discard the new trial point:
   c. $x_+ = x_c$
   d. $Y_+ = \begin{cases} \{x_c + p_c\} \cup Y_c \setminus \{y^{\ell_{\text{out}}}\} & \text{if } \|y^{\ell_{\text{out}}} - x_c\|_2 \ge \|(x_c + p_c) - x_c\|_2 \\ Y_c & \text{otherwise.} \end{cases}$

7. $x_c \leftarrow x_+, \;\; Y_c \leftarrow Y_+, \;\; \Delta_c \leftarrow \Delta_+$.

End

Note that Algorithm I does not include a geometry phase, i.e., it does not attempt to ensure the poisedness of the interpolation set in any way. We use the method of Moré and Sorensen [12] to compute a solution of the problem (2.5). In our experiments, we set $\alpha = 0, \beta = .75$ and $\gamma = 1.5$.

The initial point x_c in Algorithm I is selected by the user; the rest of the points in the initial set Y_c are chosen as the set of vertices and mid-points of the simplex defined by (see [17, 18])

$$y^i = x_c + \Delta_c \, e_i, \qquad i = 1, \ldots, n, \tag{2.6}$$

where the initial trust region radius is set to $\Delta_c = 0.5$, and e_i denotes the ith coordinate vector.

Algorithm I is not as economical as other model-based methods for derivative free optimization because it recomputes the model m_c at every iteration and forms and factors the coefficient matrix in the linear system (2.4) afresh at every iteration. This causes the workload of each iteration to be of order $O(n^6)$. The linear algebra cost could be reduced to $O(n^4)$ by expressing the model m_c using Lagrange polynomials and updating the model and its factorization at every iteration [17]. One could reduce the work even further by implementing the Powell-Frobenius [19, 18] approach that only imposes $O(n)$ interpolation conditions. However, since the goal of this paper is to study the effect of omitting the geometry phase, we choose to work with the simplest possible algorithm.

3 Numerical Tests

We describe the results of numerical tests comparing Algorithm I with the NEWUOA software that implements the Powell-Frobenius method [20, 21] and QPOLY, a code developed specifically for this study [11]. NEWUOA and QPOLY are model-based trust region methods for derivative-free optimization that include a geometry phase.

In NEWUOA, the number of interpolation points can be chosen between $2n+1$ and $\frac{1}{2}(n+1)(n+2)$; the remaining degrees of freedom are absorbed by a minimum Frobenius norm update. In our experiments, we use $\frac{1}{2}(n+1)(n+2)$ interpolation points so that NEWUOA uses the same amount of information as Algorithm I and QPOLY. We developed QPOLY as a simplification of NEWUOA; it does not include as many precautions to control roundoff errors and to mitigate the effects of noise. Our main reason for developing QPOLY is so that this study is not based solely on comparisons with NEWUOA—a package that is undergoing continuous development. (In §3.1 we also report comparisons with the WEDGE software package [10], but that algorithm attempts to control poisedness of the interpolation set in a different manner than most model-based algorithms.)

Algorithm I uses a monomial basis to define the quadratic interpolant, whereas NEWUOA and QPOLY employ Lagrange polynomials that yield substantial savings in computation; see [20, 18]. Algorithm I moves the reference point in the definition of the quadratic model at every iteration (it is the current iterate x_c), whereas NEWUOA and QPOLY change the reference point in the definition of the Lagrange basis only a few times throughout the whole iteration. Therefore Algorithm I has a slight numerical advantage because displacing the reference point limits rounding errors. We do not believe, however, that this fact alters the observations made below.

We tested the three codes on the 60 problems from the CUTEr collection listed in Table 1. All problems are unconstrained and the number of variables varies between 2 and 15. Table 1 reports the name of the problem, the number of variables, the number of function evaluations required by

the KNITRO package [22] (using second derivatives) and the optimal objective value computed by KNITRO to approximately 14 digits of accuracy. We use the results of KNITRO as a reference for our experiments.

name	n	fev	f^*	name	n	fev	f^*
allinitu	4	9	5.74438491032034e+00	freuroth	10	11	1.01406407257452e+03
arglinb	10	2	4.63414634146338e+00	genhumps	5	64	9.31205762089110e−33
arglinc	8	2	6.13513513513513e+00	gulf	3	26	5.70816776659866e−29
arwhead	15	7	5.32907051820075e−15	hairy	2	39	2.00000000000000e+01
bard	3	8	8.21487730657899e−03	hatfldd	3	22	6.61511391864778e−08
bdqrtic	10	11	1.82811617535935e+01	hatflde	3	15	4.43440070723924e−07
beale	2	9	1.03537993810258e−30	helix	3	17	1.81767515239766e−28
biggs3	3	11	3.49751055496115e−25	hilberta	10	2	1.51145573593758e−20
biggs6	6	76	5.49981608181981e−16	himmelbf	4	9	3.18571748791125e+02
box2	2	7	3.32822794031215e−23	himmelbg	2	8	1.17043537660229e−27
box3	3	9	1.85236429640516e−20	jensmp	2	10	1.24362182355615e+02
brkmcc	2	4	1.69042679196450e−01	kowosb	4	19	3.07505603849238e−04
brownal	10	8	1.49563496755546e−16	mancino	10	4	1.24143266331958e−19
brownden	4	9	8.58222016263563e+04	maratosb	2	8	−1.00000006249999e+00
chebyquad	8	19	1.75843686283896e−03	mexhat	2	5	−4.01000000000000e−02
chrosen	15	19	1.21589148855346e−19	morebv	10	4	1.85746736253704e−24
cragglvy	10	15	1.88656589666311e+00	nasty	2	2	1.53409170790554e−72
cube	2	44	5.37959996529976e−25	osborneb	11	21	4.01377362935478e−02
denschnd	3	28	2.15818302178292e−04	palmer1c	8	2	9.75979912629838e−02
denschne	3	22	1.29096866601748e−18	palmer3c	8	2	1.95376385131058e−02
denschnf	2	7	6.51324621983021e−22	palmer5c	6	2	2.12808666605511e+00
dixmaanc	15	14	1.00000000000000e+00	palmer8c	8	2	1.59768063470262e−01
dixmaang	15	28	1.00000000000000e+00	power	10	2	6.03971630559837e−31
dixmaani	15	15	1.00000000000000e+00	rosenbr	2	29	3.74397564313947e−21
dixmaank	15	18	1.00000000000000e+00	sineval	2	68	7.09027697800298e−20
dixon3dq	10	2	2.95822839457879e−31	singular	4	20	6.66638187151797e−12
dqdrtic	10	2	5.91645678915759e−29	sisser	2	20	1.06051492721772e−12
engval1	2	8	0.00000000000000e+00	vardim	10	15	1.59507305257139e−26
engval2	3	21	0.00000000000000e+00	yfitu	3	46	6.66972048929030e−13
expfit	2	9	2.40510593999058e−01	zangwil2	2	2	−1.82000000000000e+01

Table 1: Test Problems

To compare the three algorithms for various levels of accuracy in the solution, we terminated the runs when a prescribed number of correct significant figures in f was attained, or when a limit of 15,000 function evaluations was reached. We use three levels of accuracy: 3, 6 and 9 significant figures, and we report results where all three algorithms converged to the same objective value. The results are summarized in Figures 1-3 using the logarithmic performance profiles described in [8]; they compare the number of function evaluations needed to achieve the desired accuracy in f.

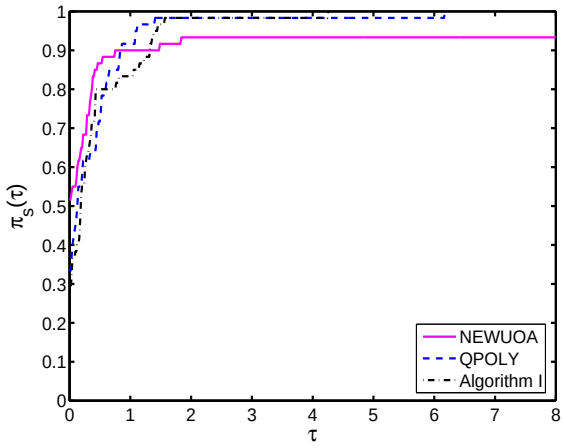

Figure 1: 3 digits of accuracy in final function value.

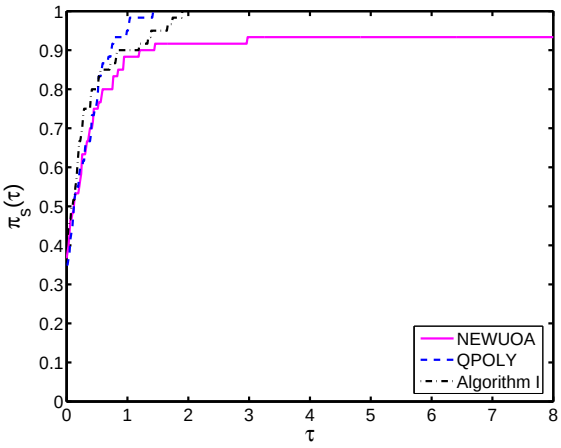

Figure 2: 6 digits of accuracy in final function value.

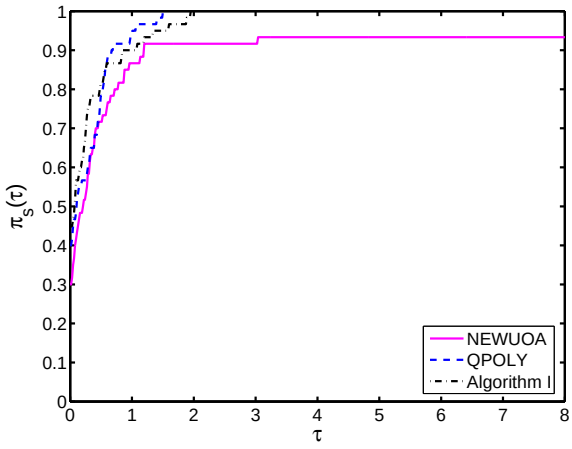

Figure 3: 9 digits of accuracy in final function value.

Note that the three algorithms perform similarly in these tests, both in terms of robustness and in terms of number of function evaluations. This is surprising, as we expected Algorithm I to be often slow (or even fail) due to a lack of control in the quality of the model. The slight disadvantage of NEWUOA in terms of robustness reflects its inability to solve some of the `palmer` problems within the limit of $15,000$ function evaluations. NEWUOA is, however, able to achieve the desired accuracy for these problems if this limit is increased to $100,000$ function evaluations.

Although the overall performance of Algorithm I is quite good, this does not mean that the interpolation points generated during the iteration remain well poised. On the contrary, the interpolation system normally becomes ill-conditioned after only a few iterations; yet the algorithm is able to make steady progress toward the solution. To illustrate the behavior of the algorithm in some detail, in the rest of the paper we provide data about the evolution of the condition number, the quality of the gradient and the rate of success of the iteration.

3.1 Conditioning of the Interpolation System

If we define $d_c = f(x_c)$ in (2.3), we remove one unknown and the interpolation conditions (2.4) can be expressed as the $(q-1) \times (q-1)$ linear system

$$Mx = b, \tag{3.1}$$

where x contains the elements of g_c and G_c. M is a matrix whose l-th row is given by

$$\left((s^\ell)^T, \{s_i^\ell s_j^\ell\}_{i<j}, \left\{ \tfrac{1}{\sqrt{2}} (s_i^\ell)^2 \right\}_{i=1,\ldots,n} \right)^T, \qquad \ell = 1, \ldots, q-1,$$

with $s^\ell = y^\ell - x_c$, and the right hand side vector b is given by

$$b = [f(y^1) - f(x_c), \cdots, f(y^{q-1}) - f(x_c)]^T;$$

see [10].

As discussed by Conn, Scheinberg and Vicente [5], the matrix M should be scaled because its condition number grows as the interpolation points cluster near x_c — even if the interpolation set Y_c is well poised. The appropriate scaling matrix is given by

$$D = \begin{bmatrix} rI_n & 0 \\ 0 & r^2 I_{q-n-1} \end{bmatrix},$$

where r is the radius of smallest ball centered at x_c that contains all points in Y_c.

When computing the model (2.3) in Algorithm I, we replace (3.1) by the scaled system

$$\hat{M}\hat{x} = b, \quad \text{with} \quad \hat{M} = MD, \quad \hat{x} = D^{-1}x. \tag{3.2}$$

In Figures 4—7 we plot the logarithm of the condition number of $\hat{M}$ as a function of the iteration number, for four representative problems in the test set. A circle indicates that the step gave a reduction in the objective function and a '+', that it did not. The algorithms were terminated when 9 significant digits in f were obtained.

Note from these figures that the condition number of $\hat{M}$ grows rapidly in the first few iterations, but increases only modestly during the rest of the iteration (except sometimes towards the very end). Observe also that there is a fairly regular oscillation in the condition number, and that steps that decrease the objective (circles) are correlated with the lower range of condition values; i.e. the circles tend to lie on the lower part of the graph. Therefore, there appears to be a self-regulating mechanism that prevents the condition number of $\hat{M}$ from growing steadily, but we have not been able to provide an analytical explanation for it.

An examination of the runs also shows that when a step yields a higher value of the objective function, the trust region is typically reduced, which forces the interpolation points to cluster. After a few unproductive steps, the conditioning of $\hat{M}$ decreases slightly and the algorithm is able to make progress. Therefore unproductive steps seem to play a role similar to the geometry phase used in other methods, although it exerts a very weak control on the poisedness of the interpolation set.

Let us now compare the conditioning of the interpolation systems generated by Algorithm I and by the WEDGE software package [10]. WEDGE implements a model-based method that ensures

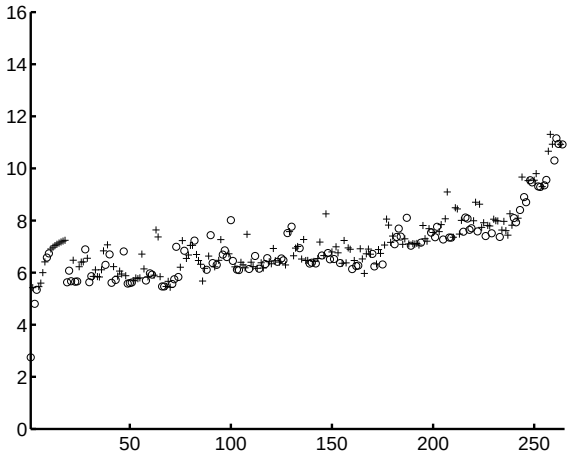

Figure 4: $\log_{10} \operatorname{cond}(\hat{M})$ vs. number of iterations for problem `arwhead`.

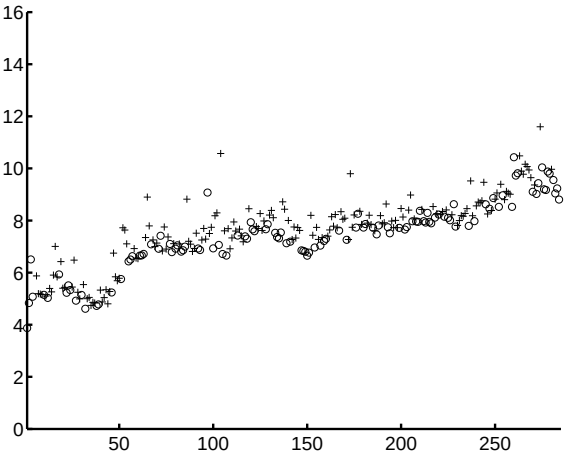

Figure 5: $\log_{10} \operatorname{cond}(\hat{M})$ vs. number of iterations for problem `brownal`.

the poisedness of the interpolation set by including an additional constraint in the step generation subproblem (2.5). Table 2 reports some statistics about the conditioning of the scaled interpolation matrix $\hat{M}$ for Algorithm I and WEDGE. We compare the two codes on the 60 test problems given in Table 1, using the three stopping conditions discussed in the previous section.

digits in f	3	6	9	3	6	9
WEDGE	12	81	134	12086	17959	20527
Algorithm I	106	177	262	11609	13447	14272

Table 2: Total number of iterations in which the condition number of $\hat{M}$ exceeded 10^{12} (first 3 columns) and total number of function evaluations (last 3 columns) to achieve 3, 6 and 9 digits of accuracy in f.

As expected, WEDGE controls the condition number of the interpolation system better than Algorithm I, and an examination of the results shows that WEDGE produces longer sequences of accepted steps. However, Algorithm I consistently requires a lower number of function evaluations because its successful steps are more productive. Table 2 gives cumulative results; performance profiles [8] would also show a clear advantage of Algorithm I over WEDGE. This was another unexpected finding of our experiments.

It may seem surprising from Table 2 that WEDGE permits the conditioning of the linear system to exceed 10^{12}. Although that algorithm can be implemented to exert better control over the position of the interpolation points, it was observed in [10] that a loser control (relaxing the"wedge constraint") results in better performance.

3.2 Quality of the Gradient

We have seen that for most iterations of Algorithm I, the points in the interpolation set Y_c are not well poised. The analysis given in [5] indicates that in this case the gradient and Hessian estimates

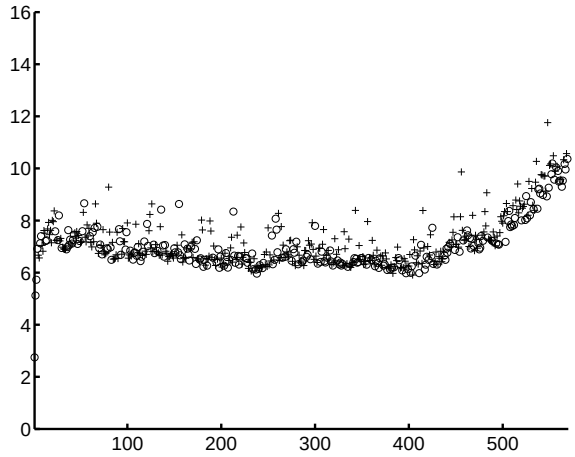

Figure 6: $\log_{10} \mathrm{cond}\,(\hat{M})$ vs. number of iterations for problem `dixmaank`.

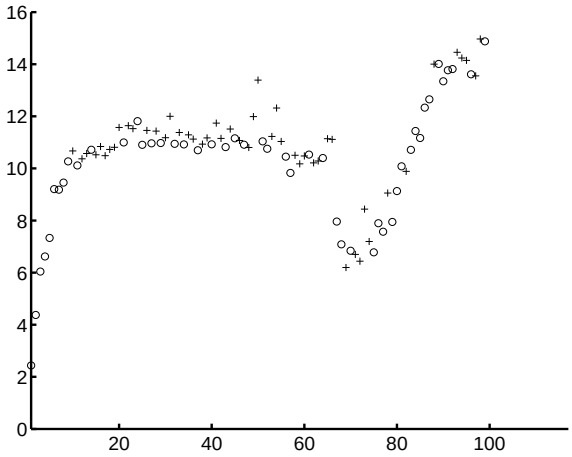

Figure 7: $\log_{10} \mathrm{cond}\,(\hat{M})$ vs. number of iterations for problem `mancino`.

g_c, G_c in (2.3) can be expected to be poor, but to our knowledge their accuracy has not been reported in the literature. We could try to use the analytical error bounds given in [5], but they depend on unknown Lipschitz constants (and may not always be tight). Therefore we compute the error in the gradient estimate (which plays a more important role than G_c) numerically. We compare our estimate g_c with the exact gradient g_c^* computed by AMPL [9].

We first test whether $-g_c$ is a descent direction. Figures 8-9 plot the cosine of the angle between g_c and the g_c^*, i.e.,

$$\cos\theta_c \equiv \frac{g_c^T g_c^*}{\|g_c\|_2 \|g_c^*\|_2},$$

as a function of the iteration number, for two representative problems from our test set. Note from these figures that although the estimated gradient g_c is a descent direction in the majority of the iterations, the descent property is not achieved consistently since $\cos\theta_c$ changes sign frequently.

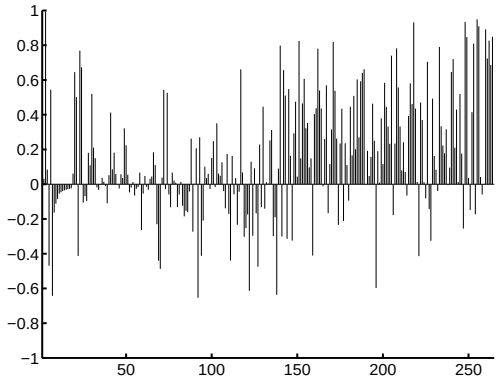

Figure 8: $\cos\theta_c$ vs. iteration number for problem `arwhead`.

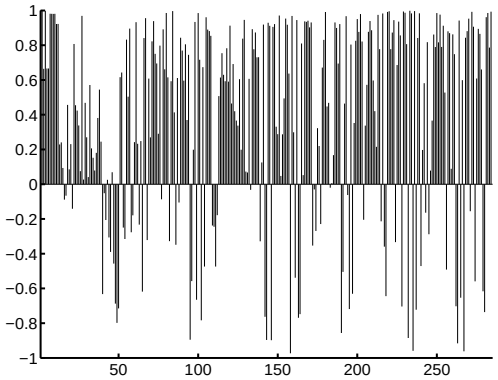

Figure 9: $\cos\theta_c$ vs. iteration number for problem `chrosen`.

Next, we measure the relative error in the gradient,

$$e_c = \|g_c - g_c^*\|_2 / \|g_c^*\|_2.$$

In Figures 10-11 we plot the quantity $-\log_{10} e_c$ as a function of the iteration number, for the same two problems as in Figures 8-9. Note that the relative error in the gradient is greater than 1 in the majority of the iterations (but there tends to be a slight improvement toward the end of the run). These results confirm that poorly poised interpolation points normally lead to highly inaccurate gradient approximations g_c. This in turn leads to the generation of many unproductive steps. We have observed that between 25% and 50% of the iterations are unsuccessful, in the sense that they produce a function value that is larger than $f(x_c)$. This may seem highly inefficient, but model-based methods invoke the geometry phase with a similar frequency.

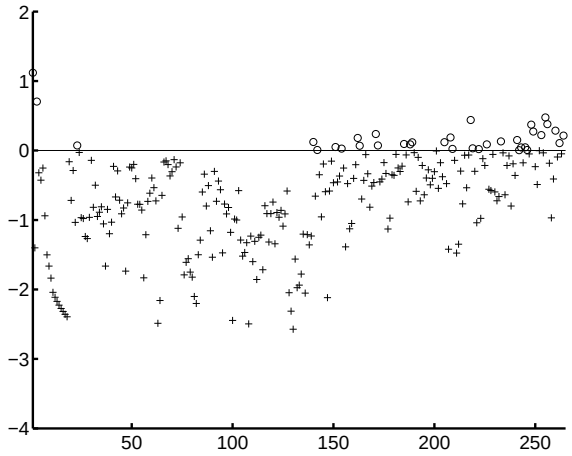

Figure 10: $-\log_{10} e_c$ vs. number of iterations for problem `arwhead`.

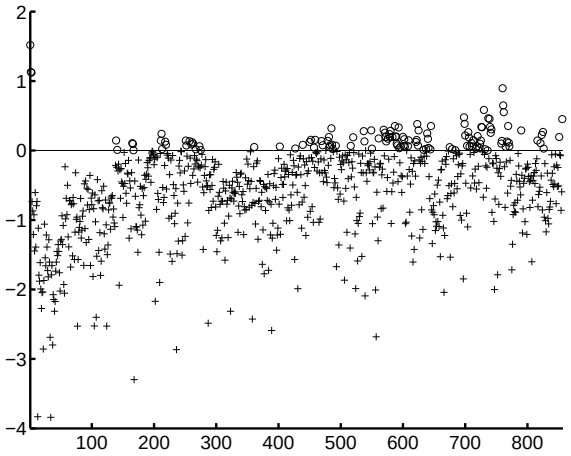

Figure 11: $-\log_{10} e_c$ vs. number of iterations for problem `chrosen`.

Improving the quality of the gradient approximation g_c without investing additional function evaluations would therefore be of great benefit to the algorithm. As a simple (idealized) experiment, we tested Algorithm I with one change: after the model (2.3) is computed, we replace the estimate g_c with the exact gradient g_c^*. (Since g_c is altered, but G_c is left unchanged the model (2.3) is no longer an interpolation model). We observed a dramatic improvement in performance; the percentage of rejected steps decreased significantly and the overall number of function evaluations dropped drastically.

4 Final Remarks

The numerical experiments reported in this paper suggest that a model-based algorithm that ignores the position of the interpolation points performs as well as two established methods that include a geometry phase. Although the interpolation systems become highly ill conditioned during the course of the iteration, this ill conditioning remains at tolerable levels and the algorithm is able to make steady progress toward the solution.

Our experiments suggest that a self-correcting mechanism may be at play: if the interpolation points are close to lying in a low dimensional subspace of $\mathbb{R}^n$, then the model becomes accurate

enough in that subspace and a point outside of it is generated. This cycle could then be repeated. Although it may not be possible to prove that this cyclic behavior always takes place, it may be possible to show that there is a tendency for it—but this remains an open question.

The test problems used in this study are all smooth unconstrained optimization problems. Moré and Wild [13] study the performance of derivative-free methods on noisy, smooth, and piecewise-smooth problems. They test an implementation of the Nelder-Mead method, a pattern search method, and the model-based method in NEWUOA, and report that NEWUOA is overall the most effective method—in many cases by a very wide margin. These results and the developments reported by Powell [21] provide new impetus for further development of model based methods. This paper suggests that the role of the geometry phase should be re-examined. Perhaps it may be preferable to employ it only as a method of last resort; not as an integral part of the algorithm. Or it may be desirable to improve the geometry phase so that it exerts better control over the position of the interpolation points and yields faster convergence than the simple algorithm studied in this paper.

Algorithm I has been implemented in a MATLAB software package that can be obtained from the authors.

References

[1] F. V. Berghen and H. Bersini, *CONDOR, a new parallel, constrained extension of Powell's UOBYQA algorithm: experimental results and comparisons with the DFO algorithm*, Journal of Computational and Applied Mathematics, 181 (2005), pp. 157–175.

[2] A. R. Conn, K. Scheinberg, and P. L. Toint, *On the convergence of derivative-free methods for unconstrained optimization*, in Approximation Theory and Optimization: Tributes to M. J. D. Powell, A. Iserles and M. Buhmann, eds., Cambridge, England, 1997, Cambridge University Press, pp. 83–108.

[3] ——, *Recent progress in unconstrained nonlinear optimization without derivatives*, Mathematical Programming, Series B, 79 (1997), pp. 397–414.

[4] ——, *A derivative free optimization algorithm in practice*, Tech. Rep. TR98/11, Department of Mathematics, University of Namur, Namur, Belgium, 1998.

[5] A. R. Conn, K. Scheinberg, and L. Vicente, *Error estimates and poisedness in multivariate polynomial interpolation*, tech. rep., IBM T. J. Watson Research Center, 2006.

[6] ——, *Geometry of interpolation sets in derivative free optimization*, Mathematical Programming, Series A, 111 (2007), pp. 141–172.

[7] G. Deng and M. Ferris, *Adaptation of the UOBYQA algorithm for noisy functions*, in Proceedings of the 38th conference on Winter simulation, Winter Simulation Conference, 2006, pp. 312–319.

[8] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Mathematical Programming, Series A, 91 (2002), pp. 201–213.

[9] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*, Scientific Press, 1993. `www.ampl.com`.

[10] Marazzi, M. and J. Nocedal, *Wedge trust region methods for derivative free optimization*, Mathematical Programming, Series A, 91 (2002), pp. 289–305.

[11] J. L. Morales, *A trust region based algorithm for unconstrained derivative-free optimization*, tech. rep., Departamento de Matemáticas, ITAM, 2007.

[12] J. J. Moré and D. C. Sorensen, *Computing a trust region step*, SIAM Journal on Scientific and Statistical Computing, 4 (1983), pp. 553–572.

[13] J. J. Moré and S. Wild, *Benchmarking derivative-free optimization algorithms*, Tech. Rep. Preprint ANL/MCS-P1471-1207, Argonne National Laboratory, December 2007.

[14] R. Oeuvray and M. Bierlaire, *BOOSTERS, a derivative-free algorithm based on radial basis functions*, International Journal of Modelling and Simulation, (2008). to appear.

[15] M. J. D. Powell, *A direct search optimization method that models the objective and constraint functions by linear interpolation*, in Advances in Optimization and Numerical Analysis, Proceedings of the Sixth Workshop on Optimization and Numerical Analysis, Oaxaca, Mexico, S. Gomez and J. P. Hennart, eds., vol. 275, Dordrecht, The Netherlands, 1994, Kluwer Academic Publishers, pp. 51–67.

[16] ——, *Direct search algorithms for optimization calculations*, Acta Numerica, 7 (1998), pp. 287–336.

[17] ——, *UOBYQA: unconstrained optimization by quadratic approximation*, Mathematical Programming, 92 (2002), pp. 555–582.

[18] ——, *On the use of quadratic models in unconstrained minimization without derivatives*, Tech. Rep. DAMTP 2003/NA03, Department of Applied Mathematics and Theoretical Physics, Cambridge University, Cambridge, England, 2003.

[19] ——, *On trust-region methods for unconstrained minimization without derivatives*, Mathematical Programming, 97 (2003), pp. 605–623.

[20] ——, *Least Frobenius norm updating of quadratic models that satisfy interpolation conditions*, Mathematical Programming, 100 (2004), pp. 183–215.

[21] ——, *New developments of NEWUOA for minimization without derivatives*, Tech. Rep. DAMPT 2007/NA05, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, 2007.

[22] R. A. Waltz and T. D. Plantenga, Knitro 5.0 *User's Manual*, tech. rep., Ziena Optimization, Inc., Evanston, IL, USA, February 2006.