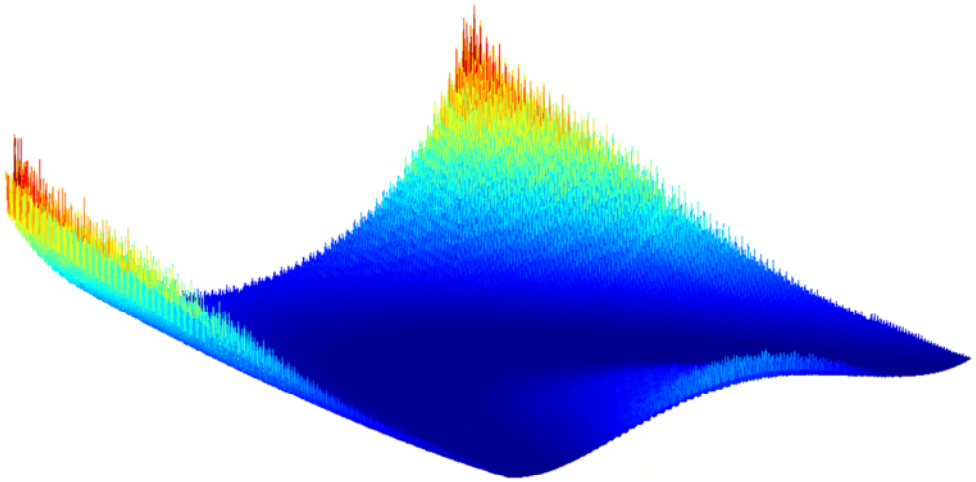
EPSRC
Engineering and Physical Sciences Research Council

InFoMM
Industrially Focused Mathematical Modelling

EPSRC Centre for Doctoral Training in Industrially Focused Mathematical Modelling



Derivative-Free Optimisation for Data Fitting

Lindon Roberts

UNIVERSITY OF OXFORD

nag®

Contents

1. Introduction

Background

Optimisation refers to the problem of finding a set of inputs for which a function (known as the objective) is maximised or minimised.

Optimisation problems are one of the most common types of mathematical problems to arise from industrial applications. We may wish to find the sale price which maximises revenues or the design which maximises efficiency; or perhaps find the controls which minimise the risk of a process failing. In scientific and other research disciplines, similar problems arise: we may wish to find the configuration of a physical system with minimal energy; or the parameter values which best fit observed data.

There are many techniques which have been developed for solving different types of optimisation problems. The most appropriate method for any particular situation is dependent on many things, such as whether the problem is constrained in any way (e.g. our sales price cannot be negative) or whether we desire a local or global solution; that is, if the solution just needs to be the best of all inputs near the chosen solution, or the best of all possible inputs. In general, constrained problems are harder than unconstrained, and global optima are harder to find than local optima. Our interest is in studying how to find consider local solutions to unconstrained problems, since efficient methods exist for these problems.

The derivative of a function tells us how fast the function is changing locally. It can also tell us in which direction we should look to find higher or lower function values, and if we are at a local maximum or local minimum.

Most of the standard optimisation techniques require the ability to calculate derivatives of the objective function (and functions defining constrains, if applicable). While this is often achievable, there are also many important situations where it is not. One important situation is when the function evaluation is inaccurate. This may arise, for instance, from a Monte Carlo simulation (a model whose output is the aggregation of many random trials), or where a model incorporates uncertainty in some inputs. Another such situation is when function evaluations are computationally expensive, so standard methods for estimating derivatives, such as finite differences, are not feasible. The requirements of such applications give rise to derivative-free optimisation (DFO), a class of optimisation algorithms which do not require or use derivative information.

The Numerical Algorithms Group (NAG) is an international mathematical software and technology company. One of its main products is the NAG Library, a software package which provides routines to solve many mathematical problems. Its optimisation routines are an important part of the library for customers in finance, energy, and other industries. Our aim is to investigate how NAG can improve its support for derivative-free optimisation, and in particular the use of DFO to solve least-squares problems.

Least-squares problems are one of the most common optimisation problems. These problems have a particular mathematical structure, where the objective function can be written as the sum of the squares of other functions. They are most commonly encountered in data fitting, where we wish to find a set of model parameters which minimise the (square of the) difference between the model predictions and some observations. There are many data fitting problems where DFO is appropriate, such as in finance and climate modelling, where model evaluations are expensive and/or noisy. Lastly, mathematically-speaking, it is no different to minimise or maximise a function (since we can just change the sign of the objective). Since we will focus on least-squares minimisation, we will always assume are trying to minimise the objective.

Derivative-free optimisation (DFO) is important when function evaluations are expensive and/or inaccurate.

Our investigation of derivative-free methods for least-squares problems will proceed as follows. In Section 2, we introduce a class of derivative-free methods, and show how one class of derivative-based optimisation algorithms can be modified to cope without derivative information. We will then see how to adapt these general-purpose derivative-free methods to least-squares problems. In Section 3, we will compare the performance of several optimisation algorithms on least-squares problems, demonstrating the utility of DFO algorithms generally, and specifically their adaptation to least-squares problems. In Sections 4 and 5, we discuss the implications of these results for NAG and provide directions for future research.

2. Derivative-Free Optimisation

Many classical methods for optimising a function require the derivative of the objective function to be known or easily estimated. Derivative-based methods start with a (possibly poor) estimate of the optimal value – provided by the user – and use the objective and its derivatives at that point to build a polynomial model to approximate the objective function. This model is generally more accurate closer to the current estimate. Since the model is polynomial, it is generally easy to minimise. By doing this, we get a new point where we expect the objective value to be lower – this becomes our next solution estimate. We repeat this process until the derivative of the objective gets close to zero (i.e. we are close to a stationary point – which could be a maximum or a minimum). A famous and popular method, called Newton's method, uses a quadratic approximation as a model.

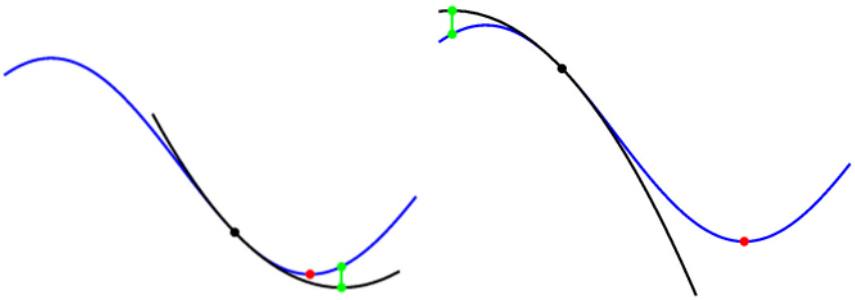


Figure 1: Illustration of the use of Newton's method to approximate an objective function (blue) in order to find a minimum (red). The black curves represent the models of the objective function, using the objective value and its derivatives at a current estimate of the solution (black point). Attempting to minimise the models produces new estimates (green points showing predicted and actual objective values). On the left, starting at a point close to the minimum gives a good new estimate. On the right, approximating a different point gives a bad new estimate, which actually has a larger objective value.

In Figure 1, we show two examples of Newton's method for the same objective function, starting from different initial estimates. In the first case, the quadratic model gives a new point which is much closer to the true minimum than the initial estimate. However, in the second case, attempting to minimise the model (by looking for a stationary point) instead finds a maximum, and gives a new estimate which is worse than the initial estimate. This common phenomenon, where Newton's method fails to find a minimum, is well-known.

To build an algorithm which converges from any starting point, we modify classical methods by constraining the size of each step we take.

To address this issue, we use what are called trust region methods. In these methods, we build a model of the objective, but instead of trying to minimise globally (by finding a stationary point), we instead try to minimise within a fixed neighbourhood of the current estimate. This reflects the fact that our approximations are only accurate near this estimate. By carefully adjusting the size of this neighbourhood as we progress, we can produce an algorithm which will converge to a solution regardless of the initial estimate we choose.

Derivative-based to derivative-free

Trust region methods provide a good basis from which to build derivative-free optimisation algorithms. If we can find a way to construct a model without using derivative information, then we can replace this step in a trust region method. As long as our new method produces a sufficiently good approximation to the function at every step, the resulting algorithms will inherit the same guarantee of convergence as the derivative-based case.

The most common derivative-free method for building models is to use interpolation. That is, suppose that instead of having a single point estimate of the objective and its derivatives, we have a cluster of points where the objective value is known. If we have the right number of points, we can find a polynomial passing through each of those points. If the geometry of cluster is good, there are theoretical guarantees that an interpolating function is a good approximation to the objective (Conn, Scheinberg & Vicente, 2009).

Therefore using this technique within a trust region framework gives us good candidate derivative-free routines.

Examples of a trust region method using derivative-based models and interpolation-based derivative-free models are shown in Figure 2. Both approaches start from the same (black) point as in the right-hand plot in Figure 1, where Newton's method failed to produce a sensible new estimate. By constraining the minimisation of the model to within the shaded trust region, we avoid the problem of incorrectly maximising the model instead of minimising. In both cases we get a new solution estimate (shown in green) which is an improvement over the original estimate. As we would hope, the derivative-free (interpolated) model gives a good approximation to the objective throughout the trust region.

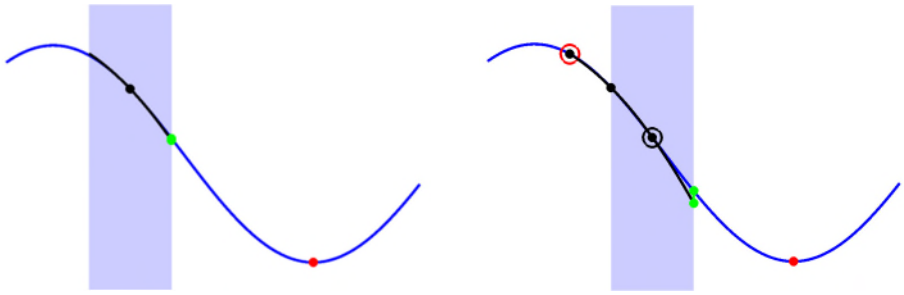


Figure 2: Illustration of trust-region methods for the example in Figure 1 for which Newton's method fails. The objective function (blue) is approximated by a quadratic model (black). Minimising the model within the trust region (shaded blue region) gives the next solution estimate (green points showing predicted and actual objective values). On the left, derivative information is used to construct the same model as in Figure 1. On the right, the model is interpolated from three function values (black points), corresponding to the original starting estimate and the ends of the trust region of the left-hand plot. The trust region is then centred at the sample point with the lowest function value (black circle). Both methods produce a good new solution estimate. In the interpolation case, the new point replaces the point with the highest objective value (red circle).

Adaptation to least-squares problems

We have introduced a framework for the derivative-free optimisation of general functions. However, it is common practice when producing mathematical software to include as much structure of the problem as possible. In general this leads to substantially better performance.

Least-squares problems require finding the minimum of an objective which is the sum of squares of several different functions. Such problems arise frequently in data fitting.

In the case of least-squares problems, the only extra piece of information we have is that our objective can be decomposed into the sum of squares of several different functions. A natural way to exploit this structure is to build models of each constituent function individually, rather than for the full objective function. When we wish to find our next solution estimate, we can construct a single model of the full objective by summing the squares of each constituent model.

This technique is well-known in the derivative-based case. The famous Gauss-Newton method for least-squares comes from applying Newton's method to a model built from linear (derivative-based) models of each constituent function. This same approach, but within a trust region framework, gives another famous least-squares algorithm, the Levenberg-Marquardt method (Nocedal & Wright, 2006).

However, the application of this insight to derivative-free methods has only been considered recently (Zhang, Conn & Scheinberg, 2010). The approach is the same as before – we construct interpolation-based models of each constituent function, which can be combined to build a model for the full objective. In the derivative-free case, we can allow either linear or quadratic models for each constituent function. However, we restrict the model for the full objective to be at most quadratic, which is a common approximation.

We have implemented this derivative-free algorithm by extending NAG's derivative-free solver to include individual constituent models. We believe this least-squares extension is not available in any other commercial software package.

3. Comparison of Solvers

In the previous section, we introduced several different optimisation techniques, covering general and least-squares problems. Some required derivative information, whereas others were derivative-free. Now we will compare how they perform on a set of test problems. The different solvers that were tested are:

- **Gauss-Newton** method for least-squares, as implemented in the NAG Library (Gill & Murray, 1978).
- **BOBYQA** (Powell, 2009), a state-of-the-art trust region-based DFO solver for general problems, available in the NAG Library (as routine E04JC);
- **DFO-basic** (Conn, Scheinberg & Toint, 1998), a simpler trust region-based DFO solver for general problems;
- MATLAB routine **fminsearch**, a DFO solver for general problems which uses the famous Nelder-Mead algorithm (Nelder & Mead, 1964); and
- **E04JC-LS**, our new extension of NAG's BOBYQA to least-squares problems.

For the derivative-based routine (Gauss-Newton), derivatives were estimated using finite differencing. This technique estimates derivatives by measuring the change in function values between two nearby points.

Testing Framework

To test the different solvers, a standard test set of 53 least-squares problems was used (Moré and Wild, 2009). Since we are interested in derivative-free solvers, it is important to compare the solvers in a context for which DFO is appropriate. To this end, the evaluation of each objective function optionally included a small amount of random noise.

Also, as previously mentioned, DFO is particularly important when function evaluation is expensive compared to the cost of the optimisation routine itself. Therefore we are interested in examining how well the solvers perform for a given budget of function evaluations. In this context, we cannot estimate how close to a true minimum we are, since that requires derivative information. As a result, we usually do not expect solvers to produce very high accuracy solutions within a limited computational budget.

The testing environment and performance measures were chosen to reflect typical use cases of derivative-free optimisation.

To this end, our performance measure is the fraction of the test problems for which a solver can produce a sufficiently small objective value in a given number of function evaluations (Moré and Wild, 2009). Since it is harder to optimise problems with more unknowns, the number of function evaluations is measured in units which grow with the problem dimension (the number of unknowns), where one unit corresponds to the number of function evaluations required to estimate the full derivative of a function at a single point.

Performance for noiseless problems

The performance of the solvers for noiseless problems is shown in Figure 3a. The new solver E04JC-LS performs substantially better than all other solvers. Most interestingly in this case is that it outperforms Gauss-Newton, which is an appropriate method in this situation, as the objective functions are all noiseless. The considerable outperformance is because we are measuring performance within a constrained computational budget, and DFO solvers are designed for this situation. Also, E04JC-LS is a more appropriate method because the required level of accuracy for these problems is relatively low compared to what Gauss-Newton could provide using derivative information. These results provide evidence that DFO solvers are preferable when we have a small computational budget and only desire low solution accuracy, even when the objective function is noiseless.

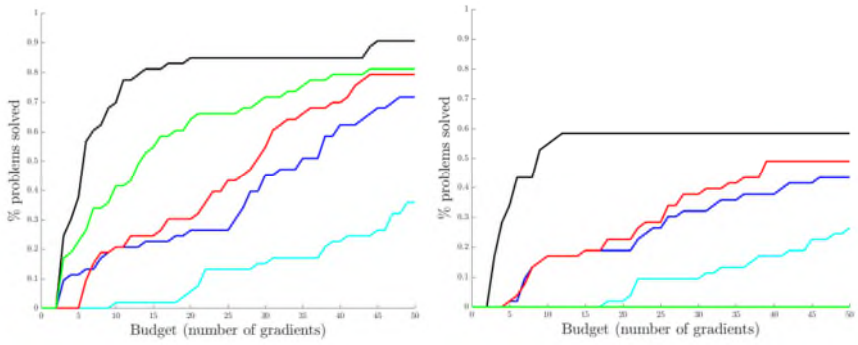


Figure 3: Comparison of solvers for (a; left) noiseless and (b; right) noisy problems. The plot shows the proportion of the 53 test problems solved for a given computational budget (higher values are better). The solvers are E04JC-LS (black), Gauss-Newton (green), BOBYQA (red), DFO-basic (dark blue) and Nelder-Mead/fminsearch (light blue). The budget (x-axis) is measured in units of 'gradients'; one 'gradient' is the number of function evaluations required to estimate the derivative of the objective one point. This measure allows for a fair comparison between problems of different sizes.

Another important conclusion from the results shown in Figure 3a is that the two solvers adapted to least-squares problems (E04JC-LS and Gauss-Newton) outperform the general-purpose solvers. This highlights the importance of exploiting problem structure when it is available. We also notice that the model-based DFO solvers (E04JC-LS, BOBYQA and DFO-basic) all outperform fminsearch, which uses the popular Nelder-Mead algorithm. Although Nelder-Mead is one of the oldest and perhaps the most well-known DFO algorithms, there is clear evidence that more modern techniques have resulted in better solvers.

Performance for noisy problems

Another situation where DFO is appropriate is when function evaluations are noisy. In Figure 3b, we show the performance of each solver when a small amount of random noise (of size ±0.1%) is added to each function evaluation. The first point of comparison with Figure 3a is that all algorithms perform worse when noise is added. This is to be expected, since the noise dilutes the information received from each function evaluation, of which there are few.

The derivative-based Gauss-Newton method fails to produce any results, as even small amounts of noise can dominate when estimating derivatives. By contrast, all the DFO solvers are still able to make progress, reflecting the robustness of interpolation as a model-building strategy compared to derivative-based approximation.

Our new solver outperforms all other solvers tested.

The new solver E04JC-LS still outperforms all other solvers, again providing further evidence that adapting the solution strategy to the underlying problem structure is beneficial. We also still see that the model-based DFO solvers outperform Nelder-Mead.

4. Discussion, Conclusions and Recommendations

There are many real-world optimisation problems where derivative information is unavailable, or prohibitively expensive to calculate. In particular, there are many least-squares problems for which derivative-free optimisation algorithms are a suitable alternative to classical methods.

Our newly implemented E04JC-LS algorithm has shown itself to be superior to many other solvers for least-squares problems when computational budget is the limiting factor in algorithm performance. It outperforms other methods designed for smooth least-squares problems when function evaluations are smooth by making better use of limited

resources, while retaining robustness to noisy functions. The adaptation of model-based DFO to least-squares problems has shown to substantially improve performance within this class of problems. The inclusion of E04JC-LS in the NAG Library will improve NAG's offering for an important class of problems.

Future research could improve the robustness of DFO solvers to noise and their ability to solve large-scale problems.

There are still several areas where DFO solvers could be improved. Firstly, DFO algorithms do not currently handle large numbers of variables to be optimised. There are many applications, such as data assimilation for weather forecasting, where this is crucial. Secondly, as seen by comparing Figure 3a with Figure 3b, even a small amount of noise causes a reasonable reduction in performance. Future research to improve DFO solvers in these situations would substantially improve their versatility, and therefore likely increase the (currently limited) use of DFO solvers in industrial applications.

5. Potential Impact

The new DFO solver adapted for least-squares problems will feature in a future release of the NAG Library. This new routine will improve NAG's offering in derivative-free algorithms. NAG will have the first commercially-available solver of this type.

The solver will help NAG's customers who wish to solve least-squares problems when function evaluations are noisy and/or expensive. One example is data fitting for financial models, where model evaluation takes the vast majority of the total computational time. The widespread use of Monte Carlo simulations also means that function evaluations may be noisy. As shown above, the new solver outperforms both classical least-squares algorithms and general-purpose DFO algorithms in this situation.

Future work on general-purpose and least-squares DFO algorithms could improve their robustness to noise and their ability to handle larger problems. This would increase the utility of such routines, driving increased usage in industry. The value of having good-quality DFO routines would therefore be even greater to companies offering mathematical software, such as NAG.

One example where future work could be beneficial to NAG's customers is in the energy sector. Oil companies need to decide the best drilling locations for new wells. To evaluate a given set of locations requires a complex and expensive simulation, spanning the entire life of the reservoir, and there is much uncertainty around the exact distribution of oil in the reservoir, potentially giving rise to noise.

Jan Fiala, a technical consultant at NAG, said: "*NAG, as a hi-tech software company, has to constantly look for gaps in the market and ways to improve our products to stay ahead of the competition. Collaboration with academia gives us direct access to the latest research which leads to enhancements in our offering. This project was highly beneficial to NAG; within a short period of time Lindon not only showed the advantages of the proposed solution but also developed a trial software implementation which will be integrated into our main product, the NAG Library. This allows us to move the solver faster to market. We look forward to the continuation of the project.*"