NPL
National Physical Laboratory

NPL REPORT MS 2

Sensitivity analysis, optimisation, and sampling methods applied to continuous models

Trevor Esward, Clare Matthews, Louise Wright, Xin-She Yang

November 2010

National
Measurement
System

NPL Report MS 2

Sensitivity analysis, optimisation, and sampling methods applied to continuous models

Trevor Esward, Clare Matthews, Louise Wright, Xin-She Yang
Mathematics and Scientific Computing Group

November 2010

ABSTRACT

This report describes the initial stages of work for the Software Support for Metrology programme project *Tools for Continuous Modelling and Simulation*. The aims of the project are to improve the understanding and use of continuous modelling packages and to build the confidence of model users in the quality of model results. We introduce the study of sensitivity analysis, optimisation, and sampling methods applied to finite element models. A key problem affecting the application of such techniques to continuous models and black box software is how to gain the most accurate information on the performance and sensitivity of the modelled system using the least amount of computational effort.

Three small finite element models were developed that cover a range of physical problems, complexities and interactions between model parameters. The models are introduced and we explain the mathematical background to the methods that were applied to analyse the problems, and summarise our main conclusions and recommendations.

NPL Report MS 2

ISSN 1754–2960

National Physical Laboratory,
Hampton Road, Teddington, Middlesex, United Kingdom TW11 0LW

We gratefully acknowledge the financial support of the UK Department for
Business, Innovation and Skills (National Measurement Office)

Approved on behalf of NPLML by Jonathan Williams,
Knowledge Leader for the Optical Technology and Scientific Computing team

Contents

1 Introduction

1.1 Scope

This report introduces key aspects of the Software Support for Metrology (SSfM) project *Tools for Continuous Modelling and Simulation*. The Software Support for Metrology programme is funded by the National Measurement Office (part of the Department for Business, Innovation and Skills).

The general aim of government-supported National Measurement System (NMS) programmes is to provide and develop, at the national level, an infrastructure that ensures measurement in the UK is valid, fit for purpose, consistent and internationally recognised. SSfM is a programme that underpins the NMS, focusing on the use of mathematics and computing in metrology. Its purpose is to achieve a balance between research and development, whilst also extending the range of techniques and applications available to meet the continually changing needs of metrology.

The results of the programme are promulgated through reports, published papers, best and good practice guides (available from the NPL website), newsletters, workshops, presentations and other awareness activities, and through the support that the SSfM team gives to other parts of the NMS. Solutions to problems specific to one field of measurement are generalised to provide techniques that are more widely applicable.

The purpose of the *Tools for Continuous Modelling and Simulation* project is to support the understanding and use of continuous modelling packages and to improve the confidence of model users in the quality of model results. We regard continuous modelling as encompassing modelling methods based on the solution of differential and partial differential equations, using such techniques as finite element, finite difference, boundary element and computational fluid dynamics methods. Continuous modelling is widely applied in many branches of science and engineering and the increased usability of software, especially arising from the development of pre- and post-processing tools linked to powerful modern computers, means that methods are readily accessible to non-expert modellers.

There are two common uses of continuous modelling within NMS programmes. The first is to design new equipment or experiments, and the second is to estimate unknown properties of existing devices or to assist in interpreting the results of experiments. In applications of the first type, the purpose of the model is to generate a design for a new product or piece of equipment with a particular performance. This application is particularly common in industry where continuous modelling, and particularly finite element (FE) modelling, is widely

used for product design. In applications of the second type, models of equipment, objects, or experiments require specification of model parameters, such as material or dimensional properties, that are not accurately known, but experimental data collected from the system being modelled may be available. For both these application types optimisation methods can be used to estimate values of model parameters to match model results to measured values or to produce a desired performance.

An understanding of the sensitivity of the model results to the various model parameters enables robust designs to be identified and allows the metrologist to identify and control the critical parameters without wasting time and effort controlling the unimportant ones.

1.2 Project outputs

There are four main outputs from the *Tools for Continuous Modelling and Simulation* project.

- The first output is this report, which sets out a general methodology for sensitivity analysis and optimisation applied to continuous models. It is designed to stand alone and its recommendations and conclusions to be understood without reference to other project outputs.
- A second report, *Sensitivity analysis, optimisation, and sampling methods applied to continuous models: three test cases*[6] describes how the sensitivity analysis and optimisation methodology described in this report was applied to three specific finite element problems. It demonstrates the methodology in action and gives much more detail than is included in the first report.
- Extensive additional numerical results from the three finite element problems analysed in the second report have been made available on the SSfM web site. The second report summarises the main numerical results and the information on the web site is provided for readers who wish to understand in more detail the analysis of the three test problems.
- Two case studies [7, 8] have been carried out in which optimisation has been applied to two challenging metrology problems that are currently the subject of research at NPL. The first [7] is a transient heat flow problem in which the thermal properties of solid materials are measured using a laser flash technique. The second [8] is the measurement of charge mobility in organic light-emitting diodes. Both problems are modelled using finite difference techniques and experimental results are in both cases used in the objective function in an optimisation process.

1.3 The audience for this report

The audience for this introductory report includes all users of continuous modelling, and, in particular, those who employ models as tools for design, who want to determine unknown model parameters by matching model results to measurement data, or who wish to associate uncertainties with model results (i.e., uncertainties due to errors in model parameters, rather than discretisation errors or other errors that may be associated with the choice of solution method).

The methods we advocate in this report are applicable to all forms of continuous modelling, not simply to finite element models. As explained above, we illustrate the use of the methods in both finite element and finite difference models.

1.4 Content and structure of report

This report sets out a general methodology that can be applied to design optimisation and sensitivity analysis. We provide specific advice on the topics set out below:

- the effect that mesh density may have not only on the accuracy of results but also the sensitivity of results to model parameters. It is important to recognise that if the effects of changes in geometrical parameters are being investigated as part of an optimisation or sensitivity analysis, this may make the interpretation of mesh density effects more difficult.
- how to choose model parameters, how to assess their effects on model results, and how to assess their importance through sensitivity indices;
- how to choose an optimisation algorithm in the light of information from the sensitivity analysis, how to make good choices of initial values of model parameters and how to define limits for these values to improve convergence and accuracy;
- the benefits of normalising model parameters to avoid anomalous behaviour that may be present in some black box optimisation algorithms and how to define appropriate convergence criteria (taking into account measurement uncertainty and fitness for purpose of the results);
- how to obtain useful results from small sample sizes when performing sensitivity analyses.

The advice we provide is general in nature and we do not discuss topics that are specific to individual continuous modelling software packages.

The structure of this report is as follows. In section 1.5 we explain briefly our approach to uncertainty analysis in continuous modelling and follow this in section 1.6 with an introduction to the methodology that we have adopted in our study, including comments on software package-specific issues that we omit from our investigation. Section 2 introduces the sensitivity analysis, sampling and optimisation methods that were employed in our work and the subsequent two sections, 3 and 4, explain in more detail the mathematical background to the methods. Our main conclusions and recommendations are summarised at the end of the report in section 5.

1.5 Uncertainty analysis and continuous modelling

Reliable evaluation of uncertainties associated with the estimates of output quantities of a model according to the recommendations of the *Guide to the expression of uncertainty in measurement* (GUM) [22] is usually implemented in terms of an explicitly stated relationship between the input (influence) quantities and the output quantities that can be manipulated analytically. Such a relationship is not normally explicitly known for *black box* software such as most continuous modelling packages, so an alternative approach must be used. In addition, the GUM makes certain assumptions (linearisation of the model, applicability of the central limit theorem) that may not be appropriate for a given model, and which can be difficult to test. Simulations using Monte Carlo methods (MCM) [21] are employed to determine the sensitivity of model results to model parameters and to determine the uncertainty associated with the model results. Monte Carlo methods require a large number of evaluations of the model with varying model parameters and the computational expense of this process is often prohibitive, so alternative approaches to sampling are considered in the work we describe below.

Following a literature search to identify the methods most likely to be of interest, the first stages of our work have compared different sensitivity indices and different sampling and optimisation methods by applying them to three finite element problems. These were chosen for their small number of elements and consequent short run-times, so that repeated optimisation runs could be carried out in a reasonable amount of time. Problems with a fully-defined analytical solution were preferred so that the sensitivity indices for the analytical and finite element problems could be compared. However, analytical solutions were only available for two of the three finite element problems that we studied.

1.6 What this document does not do

All finite element models described in this work have been implemented within SOFEA [25], a finite element toolbox written in Matlab [23]. One of the reasons for this choice was the ease with which this finite element code can be interfaced with the wide range of optimisation algorithms available within Matlab.

This document does not provide advice on wrapping algorithms around specific continuous modelling software packages, nor does it explain how to automate repetitive tasks in specific packages. The majority of popular continuous modelling packages now offer automated running by means of a scripting language, but the languages and approaches vary too much between software packages for generic advice to be useful.

Similarly, we do not give advice about individual implementations of optimisation algorithms. Two different implementations of one algorithm have been tested during the work, and the variation in results that was observed illustrates that implementation can make a difference to performance. Since there are a wide range of packages available and our experience of these packages applied to finite element models is limited, no package-specific advice is given here.

For further assistance on such topics, consult either the appropriate software manual or the software suppliers. The authors of this report are happy to investigate specific examples but cannot offer general advice beyond what is given in the deliverables listed in section 1.2.

2 Sensitivity analysis and optimisation: an introduction

In this section we introduce the optimisation and sensitivity methods we used to study the SOFEA-based finite element models. We begin by explaining the terminology that will be employed in this report.

2.1 Terminology

Consider a mathematical model of the form

$$\mathbf{y} = \mathbf{g}(\mathbf{x}; \mathbf{a}), \tag{1}$$

where $\mathbf{y} = \{y_k; k = 1, 2, \ldots, K\}^{\mathrm{T}}$, $\mathbf{x} = \{x_n; n = 1, 2, \ldots, N\}^{\mathrm{T}}$ and $\mathbf{a} = \{a_j; j = 1, 2, \ldots, J\}^{\mathrm{T}}$. The values of the a_j are known and fixed. The values of the x_n can vary from one model evaluation to another. The x_n are called **model parameters**. The y_k, which may be a subset of a larger set of results of the model, are called the **results of interest**.

The purpose of a sensitivity analysis is to determine how much a variation in a model parameter affects a result of interest. Locally this is an estimate of $\partial y_k / \partial x_n$, the **sensitivity coefficient**, but in this work the sensitivity is usually estimated globally using **sensitivity indices**.

The aim of the optimisation process is to minimise the difference between the results of interest y_k and a set of **target results** $Y_k, k = 1, 2, \ldots, K$, by altering the model parameters. The approach is to minimise an **objective function**, typically of one of the following two forms (expressed in terms of absolute and relative differences, respectively):

$$\sum_{k=1}^{K} (Y_k - g_k(\mathbf{x}; \mathbf{a}))^2, \tag{2}$$

$$\sum_{k=1}^{K} \left(1 - \frac{g_k(\mathbf{x}; \mathbf{a})}{Y_k}\right)^2. \tag{3}$$

Other forms of objective function can be used, such as weighted sums of squares or the maximum difference between the results of interest and the model results, but only the forms above have been used in the work in this project.

The values of the model parameters that minimise the objective function are called the **optimal solution**. The solution that an algorithm identifies as the optimal

solution is called the **converged solution**. The problems reported here have been designed so that the minimum value of the objective function is zero, and the values of the model parameters that produce this minimum are known. This set of model parameters is called the **reference solution**. In practice it is unlikely that the minimum value of an objective function will be zero since a perfect match between model results and measured data is unlikely to occur.

The sampling methods used for uncertainty analysis are rated on the accuracy of their estimates of the mean, standard deviation, and distribution function associated with the results of interest. Assessment of this accuracy requires a **reference distribution**. These distributions have been assembled by carrying out a large-scale calculation using random sampling to generate sets of model parameters.

2.2 General features of the SOFEA problems and their analysis

As mentioned in section 1.6, the SOFEA problems were chosen for their short running time, and in two cases problems with known analytical solutions have been selected. The analytical solutions are useful for determination of reference sensitivity indices.

The approach we adopted is described in detail below.

2.2.1 Identifying and setting up the SOFEA problems

1. Identify a suitable problem, preferably with a known or analytical solution, and develop a finite element model of the problem.

2. Choose a set of quantities (finite element model results, or quantities calculable from those results) that will be the **results of interest** for the model.

3. Choose a set of **model parameters** (geometric parameters, material properties, features of the loading or boundary conditions) that will be varied during the test.

4. Choose values of the model parameters to act as a **reference solution**, and define a probability distribution and an interval of values to be associated with each parameter. The probability distribution is required to evaluate the uncertainties associated with the model results, and the interval of values is required to limit the space of possible solutions during the optimisation process.

2.2.2 Defining the reference mesh, target results and objective function

1. Run the finite element model repeatedly, using the reference solution as model parameter values, varying the mesh between runs. Identify a **reference mesh** that generates a well-converged solution in a short running time. Here, well-converged means that when the mesh density is increased, the model results do not change significantly.

2. Run the finite element model using the reference mesh and the reference solution and extract the results of interest to act as **target results**. The target results are assumed to be known exactly since they are model results associated with a specified set of values of the model parameters.

3. Define an objective function for optimisation based on the sum of squares of the relative differences between the values of the results of interest for a given set of values of the model parameters and the target results. The purpose of the optimisation is to use the target results to determine estimates of the reference solution. This is an inverse problem. The target results in the problems examined here have been chosen to be non-zero, thus the use of relative differences is a suitable approach. If the target results were approximately zero then absolute differences would be more suitable.

2.2.3 Model evaluation

1. Evaluate the model a large number of times ($O(10^5)$), making random draws from the probability distributions associated with each model parameter to choose the parameter values. For each model evaluation calculate the results of interest and hence the value of the objective function. Use these values to construct a **reference distribution** for each of the results of interest and the objective function. Calculate the mean and standard deviation of each of the reference distributions. The distributions will be used as a reference to evaluate the performance of sampling methods using smaller sample sizes.

2. Using a large sample size (typically $O(10^5)$), calculate the sensitivity indices of each of the results of interest and the objective function to each of the model parameters. These indices will be used as reference values to investigate the performance of the sensitivity indices calculated with smaller sample sizes. Information about the specific sensitivity indices that were employed is given in section 3.

3. Run a series of optimisation jobs that minimise the objective function by varying the model parameters, subject to a set of constraints that include, but are not confined to, the intervals of values limiting the model parameters as mentioned above in section 2.2.1, step 4. Use different algorithms, and for

each algorithm used, run the algorithm 50 times starting from different randomly-generated initial values of the model parameters (subject to the limits imposed on the parameters). Keep a record of how the parameter values and objective function value evolve as each run progresses. Note any extra evaluations required to set the problem up (e.g., some parameters used within an algorithm may be calculated from a small number of initial model evaluations).

2.2.4 Analysis of results

1. Compare the results of the optimisation runs using different algorithms, comparing the average number of model evaluations taken for the algorithm to converge and the percentage of times that the algorithm converged to the reference solution.

2. Carry out a set of small-scale sensitivity calculations for each sensitivity index, using the same sample sizes throughout. Run each sample size at least ten times for each index.

3. Compare the small-scale sensitivity calculations to the large-scale reference values calculated in step 2 of section 2.2.3.

4. Carry out a set of small-scale sampling runs using sample sizes chosen to make the sampling methods straightforward to implement, using the same sample sizes for each method. Run at least ten times with each sampling method. For each run, calculate the mean, standard deviation, and distribution of all quantities considered in the reference distribution calculations.

5. Compare the results of the small-scale sampling tests. Compare the calculated means and standard deviations to those calculated from the reference distributions. Compare the calculated distributions to the reference distributions. Consider the variability of the results of each method within the ten tests for each sample size, and the variations in accuracy and repeatability with sample size.

2.3 Limitations of our methodology

There are several points to note regarding the procedure outlined above. The problems are all defined such that there is a known optimal solution (the reference solution defined above), and all optimisation algorithms have been run repeatedly with different starting points. If the optimal solution is known, then the optimisation algorithms can be assessed on their ability to find that solution. In a

real problem, the optimal solution will not be known and so the same assessment cannot easily be made. It is important to investigate how the starting point affects convergence because a good optimisation algorithm should converge to the optimal solution and should not be over-sensitive to the starting point. Whilst a good starting point is likely to improve the speed of convergence, ideally it should not affect the values to which the algorithm converges.

The problems have (where possible) been chosen to have analytical solutions, but the analytical solutions are not used to generate the target results. The test of an optimisation algorithm requires a known, and preferably unique, solution to a minimisation problem. This is best achieved for our purposes by choosing a set of parameters and a mesh to generate the target results. The analytical solution is not the solution to the finite element model, although the solution to the finite element model should converge to the analytical solution as the mesh density is increased. Use of the analytical solution would lead to convergence to a different set of values of the model parameters, and possible non-uniqueness problems. In general, the target results have been chosen to guarantee a unique solution to the optimisation problem for the model parameters under consideration.

This distinction between analytical and finite element solutions raises an important point regarding applying optimisation methods to finite element models of real situations: the mesh used for the model must give a converged solution for the results of interest (i.e., if the mesh density is increased, the solution should not change appreciably). Tests have shown that the mesh convergence affects not only the values of the results of interest, but also the sensitivity of those values to variations in the model parameters.

Figure 1 shows an example of this effect. The results shown were generated using three different meshes of the linear elasticity problem described in the report *Sensitivity analysis, optimisation, and sampling methods applied to continuous models: three test cases*. The simplest mesh had 1678 linear elements and required calculation of 1860 degrees of freedom. The second mesh had 6712 linear elements (four times as many) and 7074 degrees of freedom, and the final mesh had 1678 quadratic elements and 7074 degrees of freedom. Each mesh was used to calculate the tangential stress at a particular point for eight different values of Poisson's ratio, and the results are plotted for each mesh against Poisson's ratio. Results for an extremely fine mesh (not shown here) indicate that the stress should be independent of Poisson's ratio.

The figure shows that the variation of the result with Poisson's ratio appears to depend on element type and mesh density. This variation with Poisson's ratio could lead to incorrect conclusions about sensitivity indicces, and to difficulties with convergence of optimisation routines. The improved performance of the quadratic elements is probably due to the elliptical geometry of the problem: a

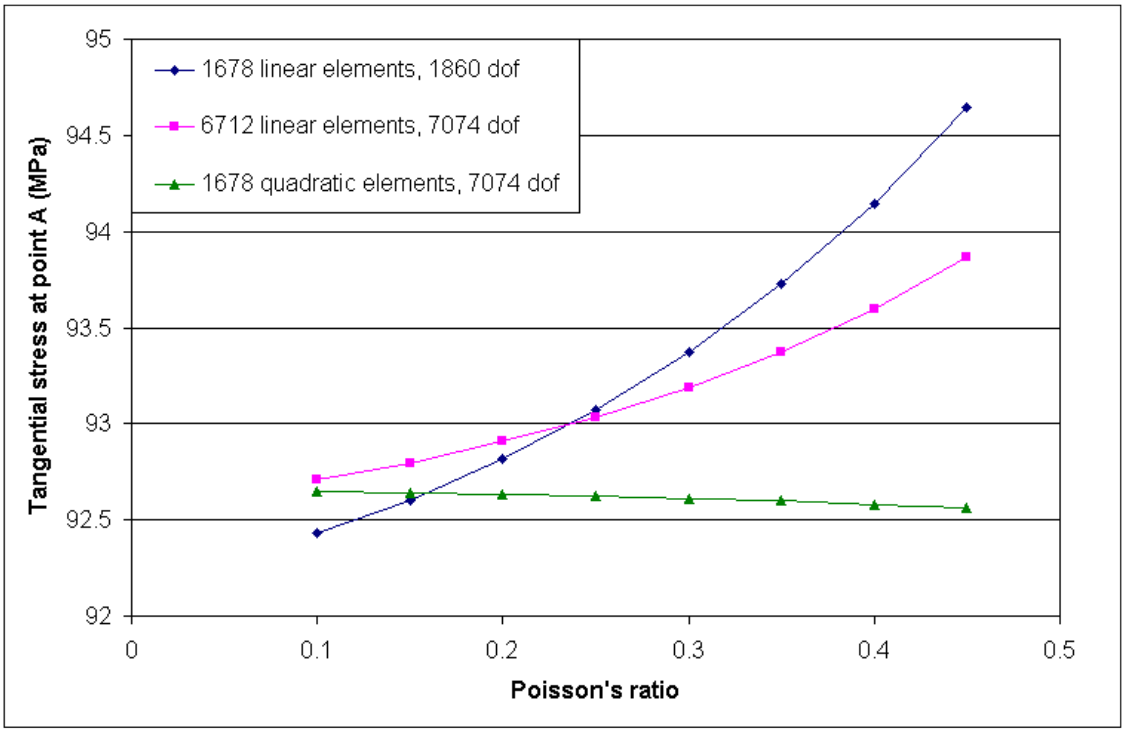

Figure 1: Comparison of results of different meshes for various values of Poisson's ratio. 'Dof' denotes degrees of freedom, related to the mesh density.

stress concentration forms at the point of interest, and quadratic elements are better able to capture this behaviour.

The main use of the analytical solution is to enable comparisons of the sensitivity indices calculated from the model results and those calculated from the analytical solution. Figure 1 suggests that comparison may not be straightforward for a mesh that does not lead to converged results, but the results will still be of interest.

The mesh for each problem has been defined by fixing the number of elements in each spatial direction and by defining elements to be equally-spaced (minor modifications to this method have been made for the thermal problem described in *Sensitivity analysis, optimisation, and sampling methods applied to continuous models: three test cases*, since the results are required at a specific internal point). This approach leads to potential problems when geometric parameters are being varied. The main problem is that the variation in element size will cause a variation in discretisation error, which then affects the objective function surface. It is likely that this approach would be the easiest to use in real-world problems and so it is useful to consider the problems that the approach generates.

2.4 Sensitivity analyses

For each SOFEA problem, three methods have been used to analyse quantitatively the sensitivity of the results of interest to the model parameters. The Sobol and Jansen methods are variance-based methods ([19], chapter 8). The Sobol method calculates the proportion of the total variation of a model result that is due to a particular model parameter. Indices should therefore take a value between zero and one. Jansen sensitivity indices are calculated in a similar manner and should also take a value between zero and one. The total effect of each model parameter on the model results is considered using a total sensitivity index (TS). For both sets of indices, the larger the value the more important the parameter. Morris's one-at-a-time (OAT) method ([19], chapter 4) is a global OAT design index, meaning that model parameters are allowed to vary over the entire input space rather than only around nominal values.

All three methods are described in detail in section 3. A separate document is available for download from the SSfM website that includes a complete list of results for all three problems.

The number of model evaluations required to calculate Sobol and Jansen indices depends on two factors: the number N of model parameters, and an integer M which defines how many sets of results are used in the calculation of the indices. The model is evaluated M times using randomly-generated sets of model parameters. Each set of parameters is perturbed one parameter at a time, requiring N model evaluations for each of the M sets, and the differences in the results of interest caused by each perturbation are used to calculate the indices. Throughout the small case studies, reference values for the sensitivity indices were generated using $M = 10^4$. The same $M(N + 1)$ model run results can be used to calculate the Sobol and Jansen indices as the two methods differ only in the processing of the generated results.

The Morris OAT method has a similar computational efficiency to the Sobol and Jansen methods as $M(N + 1)$ model runs are required to provide a full set of indices based on a sample size of M, for all N model parameters. The method calculates an estimated elementary effect for each model parameter using a difference quotient relation.

The ratio value will be greatly affected by the relative scales of the model parameter and results of interest. To produce a coherent set of sensitivity indices, once the results of interest have been obtained from the model, the model parameters are rescaled before being used to calculate the elementary effects. Model parameters are scaled such that each takes values within the interval [0, 1]. The indices lie within a K-dimensional unit hypercube, where K is the number of results of interest. When multiple elementary effects are calculated for each model

parameter (i.e. $M > 1$), the mean and standard deviation of the distribution indicate the relative importance of the various parameters. A high mean value implies a parameter important for first-order effects, while a high standard deviation implies significant interaction effects or non-linear behaviour.

For each test problem an initial sensitivity analysis was carried out in the form of a linear regression on data generated by a large scale Monte Carlo simulation. This analysis reused the data that had been used to generate the reference distributions employed to assess the sampling methods and did not involve the calculation of sensitivity indices. The analysis was carried out to identify any parameters that have a negligible effect on the results of interest, relative to other model parameters. Such parameters could then be omitted from any subsequent analysis, and the effects of the remaining, more important, model parameters were visualised using scatter plots.

Linear, interaction and quadratic regression models have been applied to the Monte Carlo data generated during the creation of the reference distributions for each result of interest of each test problem, and the mean square error recorded as a measure of fit. A quadratic model provided the fit with the lowest mean square error in all cases. Although the quality of fit to the regression model varies with each result of interest, the purpose of the regression analysis is to identify model parameters that have little effect on a result. In such a case, this will be carried through into the regression model and highlighted as little variation with the relevant parameter in the scatter plots. No quantitative information is taken from the regression analysis.

Scatter plots provide a simple and intuitive method for analysing sensitivity of an result variable to one or more model variables. Three-dimensional (3D) scatter plots or surface plots can be used to visualise the sensitivity of model results to two different model parameters simultaneously, and therefore highlight any interactions between these parameters. Using a scale to assign colour to each point on the scatter plot according to its value is an alternative, two-dimensional (2D) method to using 3D surface height to indicate value. This method can be extended to 3D and allows the analysis of three model parameters simultaneously, with each parameter being represented by one of the three axes and the colour of each plotted point representing the value at that point in 3D space. Two examples of scatter plots are shown in figures 2 and 3. The function

$$u = x + y + z \tag{4}$$

was calculated for 1 000 randomly-generated values of x, y and z, and the value of u determines the colour of the points in the plot. The 3D plot in figure 2 illustrates that all three variables have an effect on u because there is variation in colour along each of the three axes. The 2D plot in figure 3 shows u plotted against x and y. The presence of diagonal bands in the 2D plot could suggest that

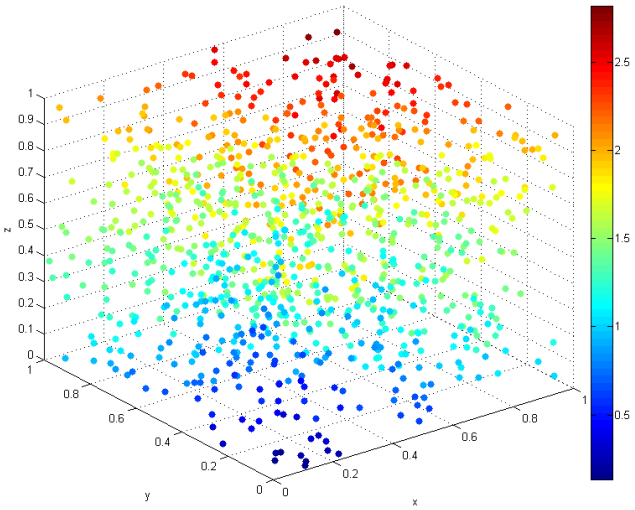

Figure 2: Three-dimensional scatter plot of $u = x + y + z$.

u depends on $x + y$, but would require further careful examination.

With an increasing number of model parameters it becomes increasingly difficult to visualise the effect of all parameters on the model results, and so identifying parameters that have little effect on an result and effectively reducing this number is a useful initial exercise.

2.5 Optimisation

2.5.1 General formulation

In general, all optimisation problems can be written as

$$\begin{aligned} &\text{minimise}_{\boldsymbol{x}\in\Re^N}\ f(\boldsymbol{x}), \qquad \boldsymbol{x} = (x_1, ..., x_N)^{\mathrm{T}}, \\ &\text{subject to } \phi_i(\boldsymbol{x}) = 0, \qquad (i = 1, 2, ..., N_{eq}), \\ &\qquad\qquad \psi_j(\boldsymbol{x}) \leq 0, \qquad (j = 1, 2, ..., N_{ineq}), \end{aligned} \tag{5}$$

where $f(\boldsymbol{x})$ is the objective function to be optimised and $\boldsymbol{x}$ is the vector of the design or independent variables [9] (model parameters). Constraints can be either equalities or inequalities. The functions ϕ and ψ can define simple bound constraints or linear or nonlinear functions of the parameters.

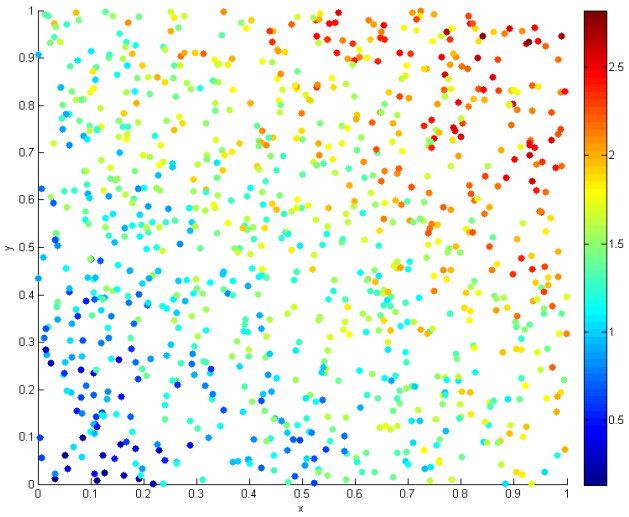

Figure 3: Two-dimensional scatter plot of $u = x + y + z$ plotted against x and y.

For a design problem, the function f is often a critical parameter that affects the cost or performance of the object being designed such as the total mass of the object. For a problem determining estimates of unknown parameters associated with an existing device, the objective function f is typically expressed as a sum of squares of differences between measured data and model results or quantities derived from model results. In the work reported here, during the optimisation procedure the measured data are assumed to be exact.

The constraints in such problems often fall into two classes. One class is generally chosen to ensure that the design performs effectively, for example, the temperature within a device may need to be less than some critical value everywhere, or the peak stress may need to be lower than some failure value for the material. The other class of constraints is typically direct limits on parameter values. These limits may be due to manufacturing considerations, e.g., some parts may have a minimum thickness for practical applications. Alternatively they may be due to the underlying physics, e.g., Poisson's ratio must lie between 0 and 0.5.

2.5.2 Optimisation of black-box functions

The formulation given in section 2.5.1 is generic. How difficult the problem is to solve, how much computational effort will be required, and which algorithms will be most suitable for solving it, depend on (amongst other factors) the nature of f and the way f is calculated. The work described here addresses the difficulties that

occur when f includes results generated from a black box function, with particular focus on results from finite element models.

In such cases, the objective function $f(\boldsymbol{x})$ cannot be written as an explicit function of the model parameters. The relationship between the results and the model parameters is deterministic but unknown. This lack of explicit knowledge about the objective function makes choosing a suitable algorithm difficult because different algorithms are suitable for minimising objective functions with different characteristics. The range of algorithms is also restricted by the lack of derivative information available when a black box function is used.

Ideally information about the objective function and the constraints should be extracted so that some idea of the *landscape* of the objective function and its dependence on the model parameters can be developed. In particular, it is useful to identify whether the function includes discontinuities and whether it has multiple local minima. Optimisation usually becomes more difficult if there are discontinuities associated with the objective function. Functions with multiple minima can lead to the algorithm converging to a local solution rather than the required global minimum. A function with no local minima and a single global minimum is called **unimodal**.

For black-box optimisation, no information is available from the initial formulation. Unless extra model runs are carried out prior to starting the optimisation, any information about the landscape (called the **response surface**) can only be obtained dynamically during the optimisation process, and such information is always incomplete. This restriction makes it extremely difficult to choose the right algorithm before starting the optimisation. An alternative approach is to carry out a sensitivity analysis to obtain some of this information, and choose the optimisation algorithm based on the results.

2.5.3 Evaluations and comparison

The computational effort associated with an optimisation problem consists of two main components: the evaluation time of the objective function, and the time taken for the optimisation algorithm to update the estimates of the model parameters based on the objective function values. In general, the execution of the optimisation algorithm itself is almost instantaneous, whereas the vast majority of finite element analyses are not. The problems chosen for the work reported here typically run in less than a minute, but most models of real-world problems take considerably longer than that to run. Hence the computational cost is determined by the number of function evaluations. Any optimisation algorithm that can significantly reduce the number of function evaluations required to find a good solution is a good choice.

To compare different algorithms it is necessary to choose an appropriate measure for comparing algorithm performance. There are two different important measures: the objective function value obtained after a fixed number of function evaluations, and the number of evaluations taken to obtain a given function value. The natural choice of measure in our case is the number of function evaluations, as it is related to the computional effort and thus better demonstrates the performance of the optimisation algorithm.

Since the problems used in the work reported here have been designed to have known reference solutions, it is also possible to consider the success rate of the algorithms. If the algorithms are started repeatedly from randomly-chosen starting points then the success rate can be evaluated by calculating the percentage of runs which converge to the correct (reference) solution.

2.5.4 Algorithms used and parameters required

A typical optimisation algorithm starts from an initial estimate of the model parameters, generates a series of new estimates, and continues this process until either the new estimates become very close to one another, or until the objective function values associated with the new estimates differ by very little (this process is called **convergence**, and the rules governing when the algorithm stops are called **stopping criteria** or **convergence criteria**). The difference between algorithms lies in the methods they use to produce the new estimates of the model parameter values.

We have studied a range of algorithms. All the algorithms are described more fully in section 4 but a summary of some key features is given here.

Traditional algorithms use deterministic methods to generate the new estimates, often based on approximation of gradients. The traditional algorithms used in this work included the Nelder-Mead simplex method [15], the trust region Levenberg-Marquardt method [5, 4], and two implementations of sequential quadratic programming [10, 9, 11, 18, 17]. Where necessary, these algorithms are abbreviated as NM, LM, and SQP, respectively.

More recent developments have included a range of optimisation algorithms inspired by biology and nature that typically use probabilistic methods to keep seemingly suboptimal sets of model parameter values in case they contain useful features not included in model parameter values that have a lower objective function value. Such methods typically do not seek to approximate derivatives or response surfaces. The methods examined here are simulated annealing (SA) [14] and particle swarm optimisation (PSO) [2, 13, 12].

As mentioned above, traditional algorithms require an initial estimate of the model parameters to start the algorithm and a set of convergence criteria to stop it. In some cases other parameters controlling the behaviour of the algorithm are defined within the implementation. For instance, if the algorithm seeks to approximate the derivatives of the objective function using finite differences, it is likely that the step size used for the approximation will be hard-coded into the implementation rather than being a user-defined parameter.

Definition of the convergence criteria is a key point, since a poor definition can lead to premature convergence or to unnecessary local searches. In most cases, little is known about the function that is being minimised. For instance it is not generally possible to state what the minimum value of the function is, and so it is not reasonable to stop when the algorithm has found a point that has a particular function value. The problems described here have known minima (a function value of zero, obtained at the reference solution), but this is not the case for real-world applications.

Minima are stationary points where the gradient of the objective function is zero, and so it is reasonable to stop the algorithm if the gradient falls below some value, although this may lead to identification of a local minimum rather than the global minimum. For black-box functions the gradient would have to be determined via finite differences or some other approximation method. An alternative approach is to stop the algorithm if consecutive function evaluations differ by less than some small value since this suggests that the search has reached a stationary point.

It is also reasonable to stop the algorithm if the new estimates are sufficiently close to the current estimates. If the algorithm is no longer looking in new areas of the model parameter space, the search has effectively converged. The user may also want to stop the algorithm owing to time constraints, an option that is typically implemented by stopping the algorithm after it has evaluated the objective function a sufficiently large number of times.

Each of these criteria require the definition of *sufficiently close, sufficiently small* or *sufficiently large number of function evaluations*. The majority of algorithms implemented within packages such as Matlab [23] and the Numerical Algorithms Group (NAG) library [24] have default values for these values, but these values may not be ideal for all problems.

As an example, consider the Nelder-Mead algorithm as implemented in Matlab. This implementation includes four stopping criteria that can be changed by the user: a maximum number of iterations, a maximum number of function evaluations, a tolerance on the function value (TolFun), and a tolerance on the model parameter values (TolX). The tolerances are applied as upper bounds on changes in value, so that if $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ are consecutive values of the model

parameters and f_i and f_{i+1} are the corresponding function values, the algorithm will terminate if $|\mathbf{x}_i - \mathbf{x}_i|$ is less than TolX or if $|f_i - f_{i+1}|$ is less than TolFun. The documentation states that *sometimes* these tolerances are used as relative bounds, so that stopping occurs if $|\mathbf{x}_i - \mathbf{x}_{i+1}| < \text{TolX}(1 + |\mathbf{x}_i|)$ and similarly for f, but does not state when *sometimes* is.

The metaheuristic algorithms SA and PSO both require additional parameters. The SA algorithm controls the efficiency and the rate of convergence via the cooling rate or cooling schedule. As described in more detail later in this report, there are three parameters associated with the cooling schedule: the initial temperature T_0, the rate of cooling α, and the final temperature T_f. The initial temperature is essentially related to the initial acceptance probability. For a given acceptance probability, the initial temperature can be determined and so for most problems, T_0 is a fixed but unknown constant. The final temperature T_f is related to one of the stopping criteria. The cooling rate $\alpha \in [0, 1]$ affects the rate of convergence significantly as explained in detail in section 4. For the work discussed in this report, a geometric cooling rate $T_n = \alpha T_{n-1}$ was used, with two different values of α: $\alpha = 0.95 \sim 0.975$ (slow cooling) and $\alpha = 0.8 \sim 0.9$ (fast cooling).

The progress of a PSO run is controlled by two learning parameters α and β, which define the strength of the randomization. In the standard version of PSO, the values used are typically $\alpha \approx \beta \approx 2$. The randomness can be reduced as the algorithm proceeds, by introducing an extra control parameter γ such that the randomness in $[0, 1]$ is reduced to $[0, \gamma^t]$ where $\gamma < 1$ and t is a quantity that increases as the number of iterations increases. It is expected that this approach should improve the rate of convergence. In all the test problems two versions of PSO have been used, one with $\gamma = 1$ (PSO 1), and one with $\gamma = 0.9$ (PSO 2).

Another important parameter in the use of PSO is the population size n. Test runs were carried out using the linear elastic test problem of *Sensitivity analysis, optimisation, and sampling methods applied to continuous models: three test cases* for various values of n between 10 and 50. It was found that for this problem $n = 20$ to 40 produced the best results in terms of convergence. Using more particles (larger values of n) may mean that the search space is explored more intensively, but it also increases the overall number of function evaluations, and hence slows down the overall optimisation process. If too few particles (say, $n < 10$) are used, the search space cannot be explored efficiently due to the lack of parallel agents. There is some trade-off, and so $n = 20$ has been used in all the work reported here.

2.6 Sampling methods

The use of sampling methods allows the uncertainties associated with black-box model results to be evaluated. The work reported here used three methods to evaluate the uncertainties associated with the results of interest for each finite element problem. The work examined how the sample size affects the accuracy of the estimates of the mean, standard deviation and distribution function (DF).

The three sampling methods tested are random, Latin hypercube and importance sampling ([19], chapter 6). Random sampling uses values drawn at random from the distribution function of each model parameter with no restrictions. Latin hypercube sampling involves dividing the DF of each model parameter into intervals of equal probability. A value is then selected at random within each interval, and values for each of the model parameters are grouped together randomly to create the sampling points. For importance sampling, the entire sample space is divided into strata. For the three finite element problems, equal probability strata were selected, and a sample point selected at random within each stratum.

The latter two methods ensure better coverage of the sample space when the sample size is small. The sampling methods are described in more detail in section 3. A separate document is available for download from the SSfM website that includes a complete list of all sampling result statistics for all three problems.

3 Sampling and sensitivity methods

A good reference for the methods and techniques described in this section is [19], which includes worked examples of practical applications as well as the supporting theory.

3.1 Random sampling

Random sampling, also known as Monte Carlo simulation, is based on random sampling from the distributions of model parameters. For a model based on N model parameters, the vector of model parameters $(x_1, \ldots, x_N)$ is formed by sampling randomly from the joint probability distribution function (pdf) of the model parameters. For a sample size M, M such vectors are sampled independently from the joint pdf. The model is evaluated at each sampled vector.

Results of the model evaluations can be used to calculate uncertainties associated with for model results, and to illustrate the effect of each model parameter on the results. Random sampling produces unbiased estimates of the mean, variance and distribution function of model results (section 6.3.4, [19]), and is also straightforward to implement. It is therefore a preferred sampling method if a large sample size is feasible. With a small sample size, random sampling may be inefficient as several sample points may lie very close together in the sample space and some regions may be missed altogether.

3.2 Latin hypercube sampling

In Latin hypercube sampling, the interval for each of the N model parameters is divided into k distinct sub-intervals, giving a total of k^N partitions. For the case studies examined here, regions of equal size were created. For each parameter, a total of M/k random samples are randomly drawn in each of the k subintervals, giving a total of M samples of each parameter. Input vectors are then created by randomly selecting, without replacement, a sample point for each model parameter and combining these into an N-vector.

Latin hypercube sampling ensures better coverage of the sample space. For small sample sizes this can become particularly important given that extremes of model results may often occur at the extremes of the model parameter values, and these may often be undersampled with pure random sampling. In many applications, it is these extreme results that are of most interest or concern.

Latin hypercube sampling tends to produce more repeatable estimates of the mean and distribution functions of model results than random sampling. Estimates of the mean and distribution function are unbiased, however Latin hypercube sampling produces a biased estimate of variance (section 6.3.4, [19]). Latin hypercube sampling also performs better than random sampling when the model results are dominated by only a few parameters.

Adjusting the value of k, and therefore M/k, affects the degree of randomness in the sampling. At one extreme, with $k = 1$, this gives purely random sampling. At the opposite extreme, with $k = M$, this can lead to a very small region over which each sample is "randomly" chosen, with essentially the only randomness then coming from the grouping of the N model parameters. Throughout the work reported in *Sensitivity, optimisation, and sampling methods applied to continuous models: three test problems* [6], either one or two samples per region were used.

3.3 Importance sampling

Similarly to Latin hypercube sampling, importance sampling involves the division of the sampling space into distinct subregions. The domain of the N model parameters is divided into a total of k regions. For the case studies examined here, regions of equal size were created. A total of M/k N-dimensional points are sampled randomly in each region, according to the joint pdf of the model parameters. This creates a total of M N-vectors.

As described in section 3.2, importance sampling ensures a more representative cover of the sample space, particularly for a small sample size.

Importance sampling allows greater control and sophistication than Latin hypercube sampling, through the implementation of non-uniform regions and region probabilities. For example, importance sampling can be used to ensure the inclusion of sample points that have a low probability of occurence, but a high consequence in terms of model results. However, without detailed a priori knowledge of the distribution of model parameters and the model function itself, it can be difficult to design the relevant regions and probabilities.

3.4 Sobol sensitivity measure

The Sobol sensitivity measure is a variance based method and produces first-order sensitivity indices S_i describing the fractional contribution of parameter x_i to the variance of a model result, second-order sensitivity indices S_{ij}, measuring the effects of the interaction of x_i and x_j on the results, and higher order measures.

The sum of the first-order indices and all higher-order indices, over all model parameters, is equal to one. Sobol sensitivity analysis can also calculate the total sensitivity index TS_i, which gives the total effect of model parameter x_i on the variance of the model result, that is, the sum of the first-order effects and all higher-order, interaction effects involving x_i. For the case studies examined here, the total sensitivity indices were calculated. A comparison of first-order and total sensitivity indices can reveal information on the co-dependence of the various model parameters. However, calculating the first-order effects in addition to the total sensitivity indices effectively doubles the number of model evaluations required.

For an index based on M inital samples and N model parameters, the calculation of a set of either first-order or total sensitvity indices requires $M(N+1)$ model evaluations. Two groups of randomly-sampled sets of model parameter values are generated, each group having M members. These groups are called $\mathbf{x}^{a,j}, j = 1, 2, \ldots, M$ and $\mathbf{x}^{b,j}, j = 1, 2, \ldots, M$.

Let the vector $\mathbf{x}^{a,j}_{\sim i}$ be equal to $\mathbf{x}^{a,j}$ with model parameter $x_i^{a,j}$ removed, and let $f(\mathbf{x}^{a,j}_{\sim i}, x_i^{b,j})$ be the function value evaluated with $x_i^{a,j}$ replaced by $x_i^{b,j}$. For calculation of total sensitvity indices, the values $f(\mathbf{x}^{a}_{\sim i}, x_i^{a})$ and $f(\mathbf{x}\sim i^{a}, x_i^{b}), i = 1, 2, \ldots, N$ are calculated for $j = 1, 2, \ldots M$. The indices are then given by

$$TS_i = 1 - D_{\sim i}/D,$$

where

$$D = \frac{1}{M}\sum_{j=1}^{M} f^2(x^{a,j}) - f_0^2,$$

$$D_{\sim i} = \frac{1}{M}\sum_{j=1}^{M} f(x^{a,j}) f(x^{a,j}_{\sim i}, x_i^{b,j}) - f_0^2,$$

$$f_0 = \frac{1}{M}\sum_{j=1}^{M} f(x^{a,j}).$$

Calculation of the first-order sensitivity indices evaluates the function values $f(\mathbf{x}^{b}_{\sim i}, x_i^{a,j})$ rather than $f(\mathbf{x}^{a,j}_{\sim i}, x_i^{b,j})$. Therefore, the total sensitivity measure compares model results where one model parameter has been altered, while the first-order measure compares results where one model parameter has remained the same.

The Sobol total sensitivity indices should lie in the interval [0, 1]. This is based on the assumption that $0 < 1 - D_{\sim i}/D < 1$ using the definitions above. This assumption leads to the inequality $\sum f(x^a) f(x^a_{\sim i}, x_i^b) < \sum f^2(x^a)$ for the condition that the sensitivity index is greater than zero. For model parameters that have

very little influence on the model results, $f(x^a_{\sim i}, x^b_i) \approx f(x^a)$, and the resulting sensitivity index can compute to a (very small) negative value. For a small sample size, the overall sum on the left hand side of the inequality above can be influenced by a single summand. If the substitution of x^b_i into vector of model parameters $x^a_{\sim i}$ causes a significantly large increase in the value of the model result, this can lead to $\sum f(x^a) f(x^a_{\sim i}, x^b_i) > \sum f^2(x^a)$, and again, a sensitivity index less than zero. In general, it should be quite straightforward to identify the reason for a negative sensitivity index. If the magnitude of the negative value is very small (say, < 0.1), this is likely due to a lack of sensitivity to the respective model parameter. If the magnitude of the negative value is large, this implies a too small sample size, where the index is being skewed by a few extreme model values.

Similarly, while in general the inequality $1 - D_{\sim i}/D < 1$ will hold, for a small sample size there is the possibility that a sensitivity index greater than one will be produced. As $D \geq 0$ must hold, a sensitivity index greater than one results from a negative value for $D_{\sim i}$, or

$$\frac{1}{M}\sum_{j=1}^{M} f(x^a) f(x^a_{\sim i}, x^b_i) < \left(\frac{1}{M}\sum_{j=1}^{M} f(x^a)\right)^2.$$

3.5 Jansen sensitivity measure

The Jansen method is similar to the Sobol method described above. The only difference is that the Jansen method calculates the squared difference of $f(x^a)$ and $f(x^a_{\sim i}, x^b_i)$ rather than the product, so that

$$TS_i = D_i/D,$$

where

$$D_i = \frac{1}{2M}\sum_{j=1}^{M} (f(x^{a,j}) - f(x^{a,j}_{\sim i}, x^{b,j}_i))^2,$$

and D is as defined above. As this results in a sum of squares calculation, unlike the Sobol method the Jansen indices cannot take negative values. However, for small sample sizes the indices can take values greater than one.

With a small sample size, both Sobol and Jansen sensitivity measures can be highly influenced by extreme model result values. Both methods are based on model evaluations at vectors of model parameters with a single model parameter adjusted. Therefore, to reduce overall computation times, the same model results were used for both measures, the only difference being the way in which the results were processed to produce a sensitivity measure. This reuse of results means that for the small sample size tests, both measures would be affected by any extreme model result values.

3.6 Morris one-at-a-time sensitivity measure

The Morris one-at-a-time (OAT) sensitivity measure is based around elementary effects of model parameters. The elementary effect of model parameter i at point $\mathbf{x}^j$ is:

$$E_i(\mathbf{x}^j) = \frac{f(x_1^j, \ldots, x_{i-1}^j, x_i^j + \Delta, x_{i+1}^j, \ldots, x_n^j) - f(\mathbf{x}^j)}{\Delta}$$

where Δ is a predefined value. A set of elementary effects are calculated for M randomly generated values of $\mathbf{x}^j$ for each of the model parameters and model results.

Three statistics are calculated from the set of elementary effects for each parameter and model result. The mean gives an impression of the overall sensitivity to a particular model parameter, while the standard deviation indicates whether there is non-linearity within the model, or co-dependence between various model parameters. A large standard deviation indicates that the influence of the respective model parameter is dependent on the value of the model parameter (non-linear) or the values of one or more remaining model parameters (interactions). The mean of the absolute value of elementary effects takes into account only the magnitude of the effect caused by a change in the model parameter. Comparing this with the general mean can also indicate non-linearity within the model. If the two values are very similar this implies that, for example, an increase in a particular model parameter will always lead to an increase in the model result.

For the Morris OAT measure, the relative values of the model parameters and results must be considered. Elementary effects can give a vast range of values depending on the relative scale of the particular model parameter being altered and result value being tested. Therefore, to produce a sensitivity measure that is more straightforward for interpretation and comparison, it is convenient to scale both the model parameters and result values to lie in the interval [0, 1]. While the specific choice of scaling constants will not have any effect on the relative position of sensitivity indices (the highest index will always be the highest etc.), it will affect the absolute values of the indices. Therefore, within each problem in this study, to make comparison between the small-scale and large reference scale tests possible, the same scaling constants were used for each model parameter and model result throughout. Input parameters described by a uniform distribtion were scaled to a standard uniform distribution. Input parameters described by other distributions were scaled according to the minimum and maximum values produced during the large-scale sampling. Each of the model results was scaled according to the minimum and maximum values produced during the large-scale test.

4 Optimisation methods

This section describes the various optimisation algorithms that have been used in the work. It aims to give technical details and information about further resources, but does not give full listings of implementations.

Optimisation methods can be divided into those that use gradient information about the objective function to guide the search for an optimal solution, and those that do not. In the former category the work has examined two different sequential quadratic programming methods and the Levenberg-Marquardt method (designed for minimisation of sums of squares). In the latter category the work has used the Nelder Mead algorithm and two stochastic search algorithms. There are two basic types of stochastic search algorithm: trajectory-based and population-based. Trajectory-based algorithms trace the progress of a single point moving on the objective function surface according to some set of rules, and population-based algorithms use the group and individual characteristics of a set of points to create the next set of points. The work described here has used one of each type. Simulated annealing (SA) is a trajectory-based algorithm, and particle swarm optimisation (PSO) is population-based.

Gradient methods generally use the Taylor expansion of the objective function

$$f(\boldsymbol{x} + \Delta\boldsymbol{x}) \approx f(\boldsymbol{x}) + \sum_{i=1}^{n} \Delta x_i \frac{\partial f}{\partial x_i} + \frac{1}{2} \sum_{i,j=1}^{n} \Delta x_i \Delta x_j \frac{\partial^2 f}{\partial x_i \partial x_j} \tag{6}$$

to choose a sequence of points that get closer and closer to the minimum. The vector whose ith entry is $\partial f/\partial x_i$ is called the gradient of the function f and is denoted ∇f.The matrix whose (i, j)th entry is $\partial^2 f/\partial x_i \partial x_j$ is called the Hessian matrix and is denoted H.

In general, gradient methods are also trajectory methods. Given a set of model parameter values, the algorithms have a method to determine the next approximation to the minimal solution. Gradient methods do not include random factors, whereas simulated annealing does, but both sets of methods generally follow the progress of a point across the objective function surface. The main step in many gradient methods is defining the search direction for the next step.

Metaheuristic algorithms do not use any geometric or gradient information to proceed. Instead they cover the search space by balancing two components: diversification and intensification [1]. For an algorithm to be efficient and effective, it must be able to generate a diverse range of solutions, including the potentially optimal solutions, so as to explore the whole search space effectively. As the search progresses it needs to intensify the search in the neighbourhood of optimal or near-optimal solutions so that convergence to an optimum can be achieved.

In order to preserve diversity, every part of the search space must be accessible, though not necessarily visited, during the search. Diversification is often implemented as a random component attached to a deterministic component in order to explore the space effectively and efficiently. Intensification uses past solutions to select the potentially good solutions via elitism or use of memory.

Any successful metaheuristic algorithm requires a good balance of these two seemingly contradictory components [1]. If the intensification is too strong, only a fraction of local space might be visited, and there is a risk of being trapped in a local optimum, as often occurs for the gradient-based search algorithms applied to multimodal problems. If the diversification is too strong, the algorithm will converge too slowly with solutions jumping around several potentially optimal solutions. Typically a search starts from a set of randomly or near-randomly generated initial guesses with a high level of diversity and as the search progresses the intensity increases and the diversity decreases as the algorithm homes in on the optimal solution. How quickly to alter the intensity and diversity is still a matter for investigation in many cases.

4.1 Sequential Quadratic Programming

Sequential quadratic programming (SQP) methods [9, 10, 11, 17, 18] solve constrained nonlinear optimisation problems. Instead of using derivative information to solve the minimisation problem directly, they solve a set of equations (the Karush-Kuhn-Tucker (KKT) equations) involving derivatives that are necessary and sufficient for a point's optimality (provided that the objective function and all the constraint functions are all convex).

If the general constrained optimisation problem is written as in (5) then the KKT equations state that the optimial point $\mathbf{x}^*$ obeys

$$\nabla f(\boldsymbol{x}^*) + \sum_{i=1}^{N_{eq}} \lambda_i \nabla \phi_i(\boldsymbol{x}^*) \quad + \quad \sum_{j=1}^{N_{ineq}} \mu_j \nabla \psi_j(\boldsymbol{x}^*) = 0, \tag{7}$$

$$\lambda_i \phi_i(\boldsymbol{x}^*) \quad = \quad 0,\ i = 1 \ldots N_{eq}, \tag{8}$$

$$\mu_j \quad \geq \quad 0,\ j = 1 \ldots N_{ineq}. \tag{9}$$

In the KKT equations, the λ_i and μ_j are Lagrange multipliers, unknown quantities determined during the optimisation process that enable the constraints to be included in the minimisation more directly. The Lagrangian function

$$L(\boldsymbol{x}, \lambda, \mu) = f(\boldsymbol{x}) + \sum_{i=1}^{N_{eq}} \lambda_i \phi_i(\boldsymbol{x}) + \sum_{j=1}^{N_{ineq}} \mu_j \psi_j(\boldsymbol{x}) \tag{10}$$

is used to form a series of quadratic programming subproblems. The kth quadratic subproblem is a minimisation problem of the form

$$\text{minimise}_{\mathbf{d}\in\Re^N} \ \frac{1}{2}\mathbf{d}^{\mathrm{T}} H_k \mathbf{d} + \nabla f(\boldsymbol{x}_k)^{\mathrm{T}}\mathbf{d} \tag{11}$$

$$\nabla\phi_i^{\mathrm{T}}(\boldsymbol{x}_k)\mathbf{d} + \phi_i(\boldsymbol{x}_k) = 0, \ i = 1,\ldots,N_{eq}, \tag{12}$$

$$\nabla\psi_j(\boldsymbol{x}_k)^{\mathrm{T}}\mathbf{d} + \psi_j(\boldsymbol{x}_k) \leq 0, \ j = 1,\ldots,N_{ineq}, \tag{13}$$

where $\mathbf{d}$ is a vector defining the next search direction. The matrix H_k is an approximation to the Hessian matrix of the Lagrangian function.

Various different methods are used to solve the subproblem. Both implementations used in this work use an active set method, which involves finding a feasible point that obeys the constraints and then finding a sequence of points that converge to the correct solution.

4.2 Levenberg-Marquardt

The Levenberg-Marquardt method [4, 5] solves problems that can be written in the form

$$\text{minimise}_{\boldsymbol{x}\in\Re^N} \ \sum_{i=1}^{m} [F_i(\boldsymbol{x})]^2 \tag{14}$$

The Jacobian matrix associated with this problem is a matrix whose (i,j)th entry is $\partial F_i/\partial x_j$.

The Levenberg-Marquardt method uses a linear approximation to the new estimate of the model parameter values and a damping coefficient to produce a method lying between the Gauss-Newton algorithm and the method of steepest descent. The search direction $\mathbf{d}$ for the kth iteration is the solution to the set of linear equations

$$(J^{\mathrm{T}}(\boldsymbol{x}_k)J(\boldsymbol{x}_k) + \lambda_k I)\mathbf{d} = -J^{\mathrm{T}}(\boldsymbol{x}_k)\mathbf{F}(\boldsymbol{x}_k), \tag{15}$$

where $\mathbf{F}(\boldsymbol{x}_k) = \{F_i(\boldsymbol{x}), i = 1,2,\ldots m\}^{\mathrm{T}}$. An alternative to this formulation that can improve convergence replaces the identity matrix I with the diagonal terms in $J^{\mathrm{T}}(\boldsymbol{x}_k)J(\boldsymbol{x}_k)$. The damping parameter λ_k is generally updated during the progress of the algorithm in order to combine efficiency and robustness. Once the step direction has been determined, a line search is carried out along the chosen direction to determine the best possible step size for the function minimisation.

4.3 Nelder-Mead

The Nelder-Mead algorithm [15] is a simplex search algorithm. For an optimsation using N model parameters, the algorithm has a group of $N+1$ points at every

step (a simplex in N-dimensional space). New points are created using three simple geometric procedures, and the new points replace the previous worst point in the set if they are improvements. If no new points are improvements then the entire simplex is shrunk around the current best point. The algorithm is regarded as having converged when the simplex shrinks below a certain size.

The three steps used to generate new points are easy to visualise in two dimensions using three points, but they can be generalised to n dimensional space. The steps are reflection, extension, and contraction. Suppose that the points in figure 4 are such that $f(A) < f(B) < f(C)$. The algorithm identifies the worst point (C) and calculates the average of A and B (X). The worst point is then reflected in the line perpendicular to the line joining C and X [assing through X, giving point D. If this point is a significant improvement to the objective function value (e.g. if it is the new best value), point E is generated by extending D further along the line between C and X. If D is not an improvement, then the point is contracted along the line between C and X so that it lies within the simplex at F. If none of D, E, and F have lower function values than C, then B and C move to G and H, respectively, and the process repeats.

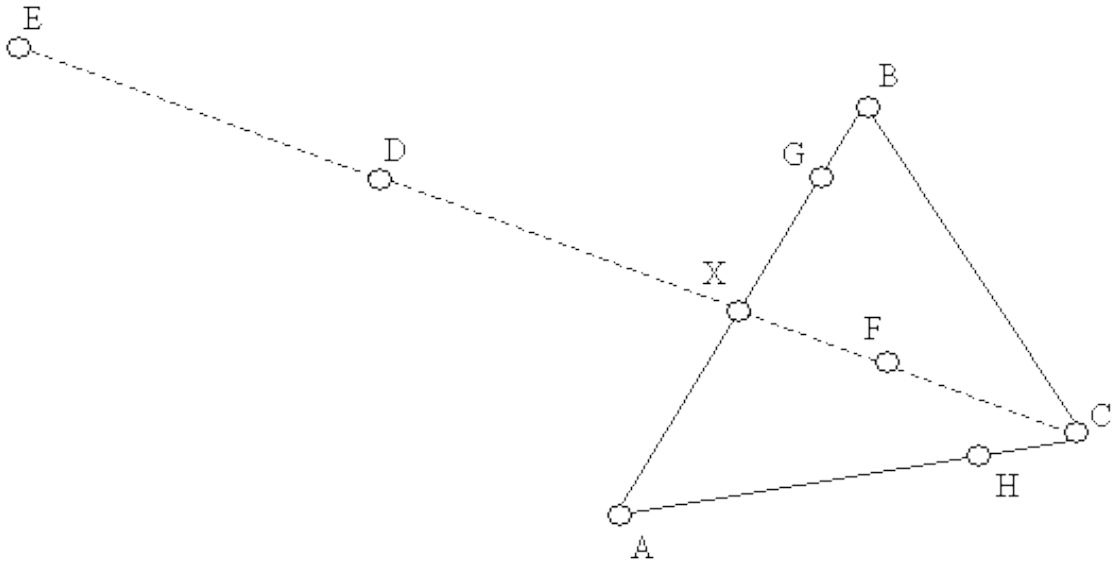

Figure 4: Sketch of the steps in the Nelder-Mead algorithm

4.4 Simulated Annealing

Simulated annealing is probably the best example of modern metaheuristic algorithms. The method was developed by Kirkpatrick, Gelatt and Vecchi in

1983 [14], inspired by the annealing process of metals during heat treatment and the Metropolis algorithms for Monte Carlo simulations.

The basic idea of the simulated annealing algorithm is similar to dropping some bouncing balls over a landscape. As the balls bounce and lose energy, they will settle down at some set of local minima. If the balls are allowed to bounce enough times and lose energy slowly enough, some of the balls will eventually fall into the globally lowest locations, and hence the global minimum will be reached. The algorithm can use many balls (parallel simulated annealing), or use a single ball to trace one trajectory, which is the standard implementation of simulated annealing. The energy of the balls is parameterised as a temperature which is gradually reduced as the algorithm progresses.

The optimisation process starts from an initial guess with a high temperature. A new guess is chosen at random and its objective function value is calculated. If the new objective function value is an improvement, the new point is accepted. If the new point is not an improvement, the point is retained with a probability

$$p = \exp[-\frac{\delta E}{k_B T}], \tag{16}$$

which is a Boltzmann-type probability distribution. Here T is the temperature of the system, while k_B is the Boltzmann constant and is taken to be 1 for simplicity. The energy difference δE is often related to the objective function $f(\boldsymbol{x})$ to be optimised. For minisation problems $\delta E = \delta f$ is often used.

The trajectory of the simulated annealing algorithm is a piecewise path, and this is a Markov chain as the new estimate of the parameter values only depends on the current estimate and the transition probability p.

The main difficulty associated with applying the algorithm is choosing the value of T throughout the optimisation process. If T is too low ($T \to 0$), then any $\delta E > 0$ (worse solution) will rarely be accepted as $p \to 0$. Consequently, the diversity of the solutions is limited. On the other hand, if T is too high ($T \to \infty$), the system is at a high-energy state, most new changes will be accepted, and the minima are not easily reached. So the temperature T is essentially controlling the balance of diversification and intensification. The change of T is called the cooling schedule.

There are two main categories of cooling schedules: the monotonically decreasing and non-monotonic. In most cases, monotonic cooling is used first, and if it does not work well then a non-monotonic schedule is recommended. One example of a monotonic schedule is the exponential cooling schedule, which takes the form

$$T(t) = T_f + (T_0 - T_f)e^{-\alpha t}, \tag{17}$$

where T_0 and T_f are the initial and final temperature respectively, t is a counter of

iterations (equivalent to time), and α is a cooling constant. However, this schedule requires the definition of the final temperature which cannot be determined easily.

For monotonic cooling, the geometric cooling schedule is by far the most widely used. The schedule defines T_t as

$$T(t) = T_0 \alpha^t, \tag{18}$$

or

$$T(t) = \alpha T(t-1), \tag{19}$$

where t is the time step or counter of iterations, T_0 is the initial temperature, and α is a cooling constant in the range of $(0, 1)$. The advantage of this schedule is that there is no need to determine the final temperature. The main disadvantage is that it can be difficult to choose an appropriate value for α. If a very small value of α is used, then there is a risk that the system may freeze too quickly and the solution might be trapped in some local optimum. If α is approaching 1, then the convergence can be unnecessarily slow. One possible solution is to use a non-monotonic cooling schedule so that the system can be elevated to a higher energy state when necessary, but if the temperature is raised too many times then the convergence is affected. These conflicting requirements demonstrate the difficulty in finding a balance between diversification and intensification.

Simulated annealing implements diversification via the random acceptance of solutions that would normally be rejected. This acceptance is controlled by the temperature T. As the algorithm proceeds, the decrease in temperature increases the intensification and decreases the diversification. For the algorithm to search efficiently, the optimal balance of diversification and intensification is required, and determining such a balance itself is an optimisation process. Fine tuning of parameters is often required to improve the efficiency of the algorithms for a particular problem. This illustrates that identifying the best parameters for efficient optimisation requires computational effort: there is "no free lunch" in any optimisation problem [20].

4.5 Particle Swarm Optimisation

Particle swarm optimization (PSO) was developed by Kennedy and Eberhart in 1995, based on the swarm behaviour exhibited in nature by fish and bird schooling, the so-called swarm intelligence [2, 12, 13]. This algorithm searches the space of model parameters by adjusting the trajectories of individual agents, called particles, as the piecewise path formed by positional vectors in a quasi-stochastic manner.

The algorithm starts with a randomly-generated set of initial positions $\boldsymbol{x}_i$ for the

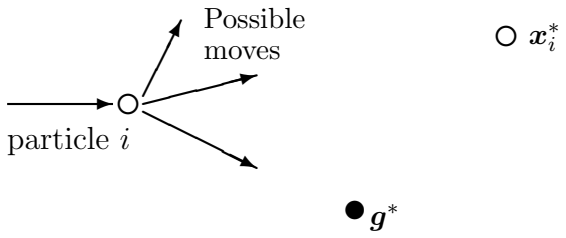

Figure 5: Representation of the typical motion of particles in PSO, moving towards the global best $\boldsymbol{g}^*$ and the current best $\boldsymbol{x}_i^*$ for each particle i.

particles. Each particle is assigned a velocity $\boldsymbol{v}_i$, which defines the location of the particle at the next time step. The velocity of each particle depends on the location of the current global optimum $\boldsymbol{g}^*$ and the location of the previous minimum value of the particle, $\boldsymbol{x}_i^*$. The particle is attracted to both of these minima, but the strength of the attraction is controlled by two random vectors $\boldsymbol{\epsilon}_1$ and $\boldsymbol{\epsilon}_2$ and two learning parameters α and β. The updating formula for the velocities is given by

$$\boldsymbol{v}_i^{t+1} = \boldsymbol{v}_i^t + \alpha\boldsymbol{\epsilon}_1 \odot [\boldsymbol{g}^* - \boldsymbol{x}_i^t] + \beta\boldsymbol{\epsilon}_2 \odot [\boldsymbol{x}_i^* - \boldsymbol{x}_i^t], \tag{20}$$

where the product $\boldsymbol{u} \odot \boldsymbol{v}$ is the componentwise inner product of two vectors. Each entry in $\boldsymbol{\epsilon}_1$ and $\boldsymbol{\epsilon}_2$ is drawn at random from the interval $[0, \gamma^t]$ where $\gamma < 1$ and t is the pseudo time of the iterations. In all the test problems γ has been fixed at either 1.0 or 0.9. The parameters α and β are the learning parameters or acceleration constants, which are commonly set to $\alpha \approx \beta \approx 2$. In theory $\boldsymbol{v}$ could take any value, but is typically limited by the size of the problem, often by forcing it to lie in some range $[0, \boldsymbol{v}_{max}]$.

The main step of the algorithm moves each particle to a new position determined by its old position and its velocity,

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \boldsymbol{v}_i^{t+1}, \tag{21}$$

and evaluates the objective function at each of the new positions. If a particle has reached the best function value in its trajectory, $\boldsymbol{x}_i^*$ is updated for that particle. If a particle has found a new global minimum then $\boldsymbol{g}^*$ is updated. This process repeats either until the global best has not improved for several iterations, or when some user-defined number of iterations has been exceeded.

The particle movement is schematically represented in figure 5 and the algorithm is summarised as pseudo code in figure 6.

There are many variations on the basic PSO algorithm as outlined above. One variation that has been found to improve convergence is the use of an inertia

Particle Swarm Optimization

begin

Objective function $f(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, ..., x_N)^T$

Generate initial locations $\boldsymbol{x}_i$ *and velocities* $\boldsymbol{v}_i$ *of* n *particles.*

while *(criterion)*

for *loop over all* n *particles and all* p *dimensions*

Generate new velocities $\boldsymbol{v}_i^{t+1}$ *using equation (20)*

Calculate new locations $\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \boldsymbol{v}_i^{t+1}$

Evaluate objective functions at new locations $\boldsymbol{x}_i^{t+1}$

Find the current best

end for

Find current best $\boldsymbol{x}_i^*$ *and current global best* $\boldsymbol{g}^*$

$t = t + 1$ *(pseudo time or iteration counter)*

end while

Output the results $\boldsymbol{x}_i^*$ *and* $\boldsymbol{g}^*$

end

Figure 6: Pseudo code of particle swarm optimization.

function $\theta(t)$ so that $\boldsymbol{v}_i^t$ is replaced by $\theta(t)\boldsymbol{v}_i^t$ where θ takes values between 0 and 1 [3]. In the simplest case, the inertia function can be taken as a constant, typically $\theta \approx 0.5 \sim 1.0$. A value of $\theta = 1$ has been used i nthis work, which is equivalent to the standard PSO algorithm. This choice is equivalent to introducing a virtual mass to stabilize the motion of the particles, which is expected to encourage the algorithm to converge more quickly.

PSO has some similarity with other population-based algorithms such as genetic algorithms. Both algorithms use multiple objects to search a space, and both retain the best solution from step to step. PSO is much simpler to implement because it does not use mutation or crossover operators to maintain diversification. Instead, it uses real-number randomness and global communication among the swarm particles. An additional simplification is that there is no encoding or decoding of the parameters into binary or real number strings as is required in genetic algorithms [16].

5 Conclusions

In this final section of the report we summarise the main conclusions and recommendations arising from the investigations into sensitivity analysis and optimisation, as applied to the test problems. Readers who wish to understand the background detail and numerical results that led to these conclusions are referred to the report on the three SOFEA finite element test problems.

We consider separately sensitivity analysis and optimisation.

5.1 Sensitivity analysis

1. Carrying out a sensitivity analysis of a model brings multiple benefits. It allows the key model parameters to be identified in advance of optimisation work, which reduces computational effort and often increases success rates. It also provides information about the smoothness and continuity of the objective function, which enables the user to choose a suitable optimisation algorithm.

2. Basing a regression model on the points from a sensitivity analysis can give sufficient qualitative information to identify which parameters can be neglected. Coloured scatter plots are a valuable tool for visualising the results of the full analysis and associated regression models.

5.1.1 Sensitivity indices

1. Sensitivity indices provide a good measure of global sensitivity, but it should be noted that this is not the same as local sensitivity and so may not be an accurate measure of local behaviour and dependencies. The choice of global or local measures needs to be based on what information is to be obtained. Choosing key parameters may require a global approach whereas an uncertainty analysis will require local information.

2. The three types of index examined in this work (Sobol total sensitivity indices, Jansen total sensitivity indices and Morris one-at-a-time (OAT) indices) gave consistent orderings of importance of the model parameters for all problems studied.

3. Of the three sensitivity indices, Morris OAT indices offer more information about the nature of the dependence of the model results on the parameter than the others, since the mean value of the indices provides an indication of the degree of linear dependence of the parameters and their standard

deviation is related to the amount of non-linear dependence (including interaction with other model parameters).

4. Sobol indices can sometimes be misleading as they can take negative values if the sample size is small or if the model parameter is insignificant. Jansen indices do not have this problem as they are computed from a sum of squares. Both indices can take values larger than one for small sample sizes.

5. For the test finite element problems investigated, a comparatively small number of model runs are required to compute indices that rank the model parameters correctly in order of importance. In general, the Morris OAT indices were better than the other indices for small sample size, and had better repeatability than the other indices over repeated runs.

6. For a small sample size, Latin hypercube sampling and importance sampling give better estimates of the mean, standard deviation, and cumulative distribution function of an result quantity than random sampling. The former methods also have better repeatability over multiple trials.

5.2 Optimisation

1. The performance of optimisation algorithms is very strongly affected by the nature of the objective function. The problems investigated have all had objective functions based on a sum of squares, and have mostly been smooth and unimodal. These features have meant that algorithms designed for smooth unimodal functions, such as the Levenburg-Marquardt algorithm, have performed well and those designed for more challenging problems, such as metaheuristic algorithms, have not been so efficient. The choice of algorithm should be guided by as much knowledge as possible of the objective function's characteristics.

2. As would be expected, problems with a small number of parameters converge more quickly than those with a large number of parameters and are often easier to solve (i.e., algorithms have better success rates on smaller problems). Similarly, constraining parameters, even if it is just by imposing simple limits on their values, reduces computational expense and increases success rates.

3. If using a black-box optimisation algorithm within a software package, it is worth simplifying the problem as much as possible by normalising all variables to be $O(1)$ as far as is possible. Black-box algorithms can exhibit unexpected behaviour (e.g., unnecessary repetition of evaluations) and can involve parameters that may not be rigorously defined and whose default values may not be immediately obvious (this is particularly true of parameters associated with convergence or stopping criteria).

4. The success rate of traditional algorithms can be strongly affected by the location of the initial guess used as a starting point. Metaheuristics are less prone to this problem because one of their key features is that they attempt to maintain a diverse search space throughout.

5. The convergence rate of metaheuristic algorithms can be improved by adjusting their internal parameters, but there are no set rules for choosing the best parameter values, and the choice of values is dependent on the nature of the objective function. Values that are good for a smooth unimodal function may restrict the search of a more complicated surface too rapidly.

References

[1] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptural comparison. *ACM Comput. Surv.*, 35:268–308, 2003.

[2] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, 1999.

[3] A. Chatterjee and P. Siarry. Nonlinear inertia variation for dynamic adaptation in particle swarm optimization. *Comp. Oper. Research*, 33:859–871, 2006.

[4] T. F. Coleman and Y. Li. An interior trust region approach for nonlinear minimization subject to bounds. *SIAM Journal on Optimization*, 6:418–485, 1996.

[5] T. F. Coleman and Y. Li. On the convergence of reflective Newton methods for large-scale nonlinear minimization subject to bounds. *Mathematical Programming*, 67:189–224, 1994.

[6] T. J. Esward, C. E. Matthews, L. Wright, X.-S. Yang. Sensitivity analysis, optimisation, and sampling methods applied to continuous models: three test cases. NPL Report MS 3, NPL, 2010.

[7] L. A. Chapman, C. E. Matthews, S. J. Roberts, L. Wright, X.-S. Yang. Tools for continuous modelling case study 1: the laser flash experiment. NPL Report MS 4, NPL, 2010.

[8] T. J. Esward, S. Knox, L. Wright. Tools for continuous modelling case study 2: organic light-emitting diodes. NPL Report MS 5, NPL, 2010.

[9] P.E. Gill, W. Murray, and M. H. Wright. *Practical optimization.* Academic Press Inc, 1981.

[10] P. E. Gill, W. Murray, and M. A. Saunders. An SQP algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 12:979–1006, 2002.

[11] S. P. Han. A globally convergent method for nonlinear programming. *Journal of Optimization Theory and Applications*, 22:297–309, 1977.

[12] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proc. of IEEE International Conference on Neural Networks*, pages 1942–1948. IEEE Press, 1995.

[13] J. Kennedy, R. Eberhart, and Y. Shi. *Swarm intelligence.* Academic Press, 2001.

[14] S. Kirkpatrick, C. D. Gellat, and M. P. Vecchi. Optimisation by simulated annealing. *Science*, 220:671–680, 1983.

[15] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright. Convergence properties of the Nelder-Mead simplex method in low dimensions. *SIAM Journal of Optimization*, 9:112–147, 1998.

[16] M. Mitchell. *An introduction to genetic algorithms*. MIT press, Cambridge, Mass., USA, 1996.

[17] M. J. D. Powell. *A fast algorithm for nonlinearly constrained optimization calculations*. Springer Verlag, 1978.

[18] M. J. D. Powell. The convergence of variable metric methods for nonlinearly constrained optimization calculations. In *Nonlinear Programming 3*. Academic Press, 1978.

[19] A. Saltelli, K. Chan, and E. M. Scott. *Sensitivity Analysis*. Wiley, 2001.

[20] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1:67–82, 1997.

[21] BIPM, IEC, IFCC, ILAC, ISO, IUPAC, IUPAP, AND OIML 2008 *Evaluation of measurement data — Supplement 1 to the "Guide to the expression of uncertainty in measurement" — Propagation of distributions using a Monte Carlo method* Joint Committee for Guides in Metrology JCGM 101

[22] BIPM, IEC, IFCC, ILAC, ISO, IUPAC, IUPAP, AND OIML 2008 *Evaluation of measurement data — Guide to the expression of uncertainty in measurement* Joint Committee for Guides in Metrology JCGM 100

[23] MATLAB `www.mathworks.co.uk/products/matlab/`

[24] NAG `http://www.nag.co.uk/`

[25] SOFEA `http://hogwarts.ucsd.edu/~pkrysl/sofea/publish.html`