NPL
National Physical Laboratory

NPL REPORT MS 15

Finite element optimisation and uncertainty toolbox test results

L. Wright, X.-S. Yang

MAY 2012

National
Measurement
System

Finite element optimisation and uncertainty toolbox test results

L. Wright, X.-S. Yang.
Materials Division

ABSTRACT

This technical note reports the results of a set of tests of software toolboxes for optimisation and uncertainty evaluation using finite element models.

ISSN 1754-2960

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

Approved on behalf of NPLML by Alistair Forbes, Science Area Leader for Data Analysis and Uncertainty Evaluation .

CONTENTS

1 INTRODUCTION

Finite element (FE) analysis is commonly used for design tasks, both within NPL and in most sectors of industry. Most design tasks involve creating an object that has a specific performance under a set of well-defined physical conditions, with a set of constraints controlling physical parameters such as material properties, physical dimensions, etc. Design processes often include other goals such as weight or cost reduction, and other constraints such as manufacturability conditions. In general, such problems can be cast as optimisation problems, often with multiple conflicting objectives.

The awareness of the effects of variability and uncertainty on simulation results is increasing amongst FE users. Uncertainty evaluation is becoming more common, particularly in areas such as aerospace, automotive and civil engineering where meeting safety criteria is a key part of the development process. Another area of growing importance is robust optimisation, which combines optimisation and uncertainty evaluation to identify designs that are near-optimal and remain feasible when the effects of uncertainty are taken into account.

The use of FE models within optimisation and uncertainty tasks throws up several challenges. Most FE modelling is carried out using off-the-shelf software packages that do not offer source code access. The "black box" nature of such packages means that derivative information about the model is generally unavailable. The lack of derivative information limits the choice of optimisation algorithms and makes determination of sensitivity coefficients, required for the Guide to the expression of Uncertainty in Measurement (GUM) [16] approach to uncertainty evaluation, difficult.

FE models are frequently computationally expensive which makes numerical approximation of derivatives inconvenient. The computational expense also makes random sampling, as described in supplement 1 of the GUM [15], an unsuitable method for uncertainty evaluation. These challenges make the choice of suitable methods for optimisation and uncertainty evaluation difficult.

Recent NPL work has investigated optimisation algorithms and sampling methods suitable for use with FE models [4-8]. This work included the implementation of various algorithms as well as advice on good practice. The original intention was to collect these algorithms into a usable toolbox that could be interfaced with FE packages and used in real-world design problems. In preparation for this, a search was carried out for existing toolboxes that had similar aims.

The search identified several tools with strong ties to a particular FE package (including iSight for Abaqus and Design Modeler for Ansys) and several generic tools. Trial copies were obtained of two of the generic tools, OptiY and modeFRONTIER, for testing. These packages were chosen because trial versions were available in a form that could easily be integrated with NPL's desktop computing system and with existing FE models.

This technical note reports the nature of the tests, the results and general observations, and draws conclusions about the viability of the proposed NPL toolbox. The work here has focused on the optimisation capabilities. Optimisation algorithms can be more challenging to implement efficiently than sampling algorithms so it was thought that the quality of optimisation algorithms would be a good indicator of the quality of the rest of the functionality of the software.

2 TEST DEFINITIONS

Two test problems were selected. Both problems have been examined in previous work [5-7], so models were immediately available and the expected values for optimisation were well understood. The aim of the tests was to assess the usability of the software and to test the accuracy and efficiency of its algorithms. The usability was assessed via user feedback rather than any formal metrics. The accuracy and efficiency of the various algorithms was assessed by recording the number of function evaluations taken for optimisation algorithms to converge, and the proportion of times they found the correct solution when the algorithm was started repeatedly from different (randomly generated) points.

All tests were posed as optimisation problems that minimised the difference between a set of model results and target values. The test definitions consisted of three components: a model (finite element or finite volume, including mesh specification), a set of quantities that were allowed to vary (the input parameters) and a set of results of interest and target values for those results.

It is believed that both problems described here have unique solutions and smooth response surfaces, although mesh issues may introduce local roughness to the Abaqus problem in particular. The results of the tests should be read with the smoothness and uniqueness in mind: previous work [5] has shown that different search algorithms are better choices for different classes of problem.

2.1 ELLIPTICAL MEMBRANE TEST PROBLEM IN ABAQUS

The first test problem model simulated an elliptical membrane with a confocal hole under a uniform outward pressure load. The model used a plane stress formulation since there are no out of plane forces. This example was drawn from a NAFEMS benchmark [9], and was studied extensively as part of earlier work [5]. Figure 1 shows the basic geometry, loading and boundary conditions of the problem. The results of interest used in the original work were a set of stresses and displacements at each of the corner points of the domain, and the input parameters were the Young's modulus and Poisson's ratio of the ellipse material, and the horizontal semi-axes of the inner and outer ellipses, marked as *a* and *b* in figure 1.

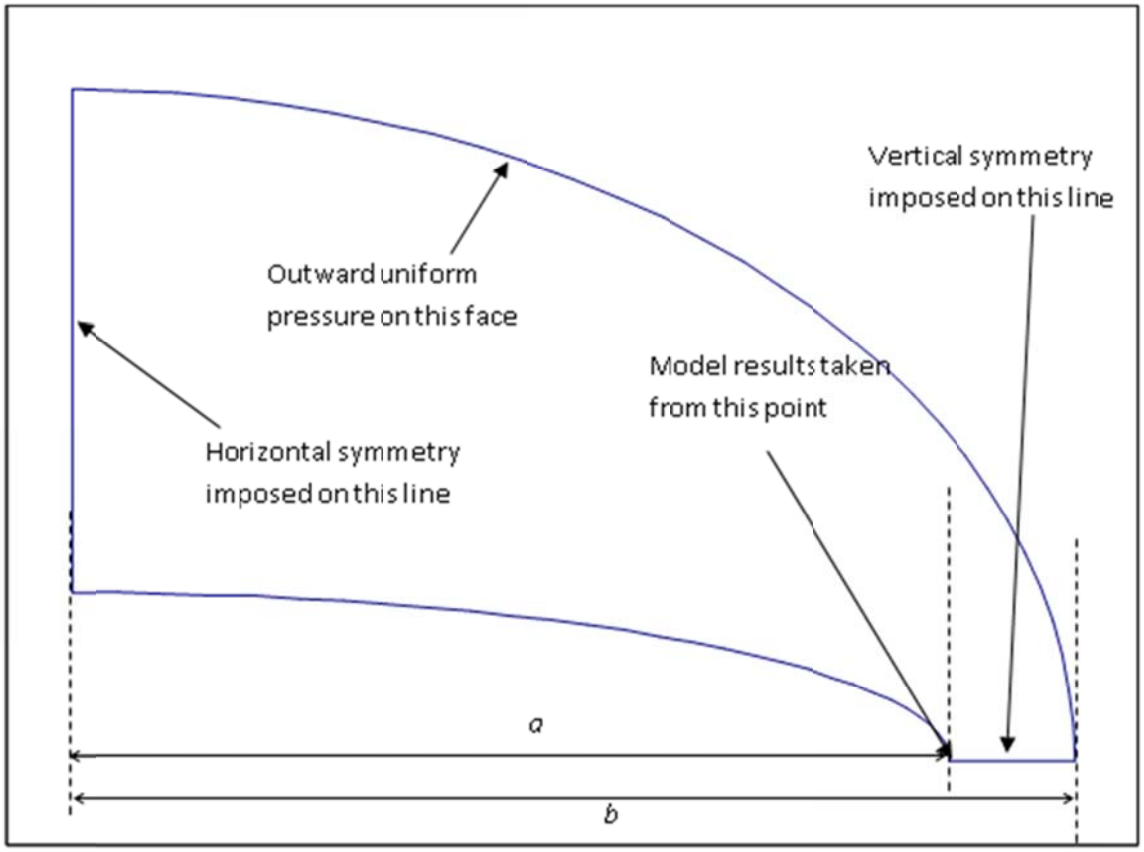

Figure 1: Sketch of the first test problem.

Earlier work had shown that the model results were most sensitive to changes in the semi-axes and least sensitive to changes in the material properties. The dependence of the model results on the semi-axes was strongly nonlinear, making the associated optimisation a challenging problem. The test problem therefore fixed values of the material properties (setting Young's modulus at 210 GPa and Poisson's ratio at 0.3) and allowed the semi-axes to vary.

The model was implemented as a finite element model within Abaqus. Since NPL only has one licence for the Abaqus pre-processor, the model was written such that the optimisation could be carried out via alterations to the text input (.inp) file rather than via use of the pre-processor interface. The changes in semi-axis values were implemented by using Abaqus' parameter shape change capability, which allows the user to define nodal positions as a weighted sum of positions with the weights being the parameter values.

The use of semi-axes complicated the modelling for several reasons. The most obvious problem is that if the parameters are unconstrained then a physically impossible shape might result if $b < a$. This problem was avoided by adding a constraint that $b - a > 0.25$ m. This constraint also avoided the ellipse becoming too thin, to avoid excessive stress values. The constraint could be imposed within both of the software packages under test.

It was decided to restrict the geometry so that the inner and outer boundaries were confocal ellipses. This choice makes the parameterisation of the geometry within the model input file considerably simpler and allows the geometry to be defined completely by two parameters. This restriction defines the vertical semi-axes in terms of the horizontal semi-axes. It is possible to use simple arithmetic functions within an Abaqus input file to impose confocality, but the resulting input file is somewhat complicated to set up. Ideally the problem would be avoided by parameterisation of the geometry within, and subsequent use of, the Abaqus pre-processor, but this approach was not possible due to licence availability.

A problem that always occurs when using geometric input parameters for a problem using numerical approximation methods to solve the main model is that of mesh definition. If a fixed mesh is used then some combinations of values of the input parameters can lead to a severely distorted mesh and hence inaccurate results. If automatic meshing is used it can be difficult to ensure that the variation in mesh density from model to model does not have a significant effect on model results. The method used to implement this model as an optimisation problem meant that a fixed mesh was the better option. The semi-axes were bounded to ensure that mesh distortion was not excessive, so that 1.75 m $< a <$ 5.75 m and 2 m $< b <$ 6.25 m.

The mesh density was determined by the maximum number of shape change parameters Abaqus allows, which is 32. The restriction on the mesh density meant that the results of the optimisation would not be consistent with those in the previous work, so new target values were determined for the problem using the restricted mesh. Since the test problem involves only two input parameters, a unique solution to the associated optimisation problem can be defined by using two model results in the objective function. The model results chosen were the stress σ_{yy} and the displacement u_x at the point shown in figure 1.

The values of a and b chosen as the true values are $a = 2$ m, $b = 3.25$ m. The target values of the results of interest associated with these values of a and b were $\sigma_0 = 7.897 \times 10^7$ Pa and $u_0 = -1.026 \times 10^{-4}$ m. These values are clearly of vastly different orders of magnitude, and so the objective function for the problem was chosen to be a sum of squares of relative differences, i.e.

$$\sqrt{\left(\frac{\sigma_{yy} - \sigma_0}{\sigma_0}\right)^2 + \left(\frac{u_x - u_0}{u_0}\right)^2}$$

A Python script was used to automate the extraction of the results from Abaqus Viewer (the Abaqus post-processor) without opening the GUI. The model therefore required an input file, a batch file to control the running of the job and of Viewer, and a Python script.

2.2 LASER FLASH TEST PROBLEM IN MATLAB

The second test problem was a model of the laser flash thermal diffusivity experiment on a layered sample. The model simulates the transient heat flow within a layered sample. The thermophysical properties of all of the layers are known, with the exception of the thermal conductivity of one layer. The sample is heated by a short pulse of energy, assumed to be uniform over the sample surface, from a laser of unknown power density. The sample loses heat via radiation (the material emissivities are known) and via conduction due to the sample contacting part of the experimental apparatus. The thermal bond between the sample and the apparatus is characterised by a parameter of unknown value. The system used in the model is shown in figure 2. A full definition of the equations and boundary conditions defining the problem can be found in report MS4 [6,7].

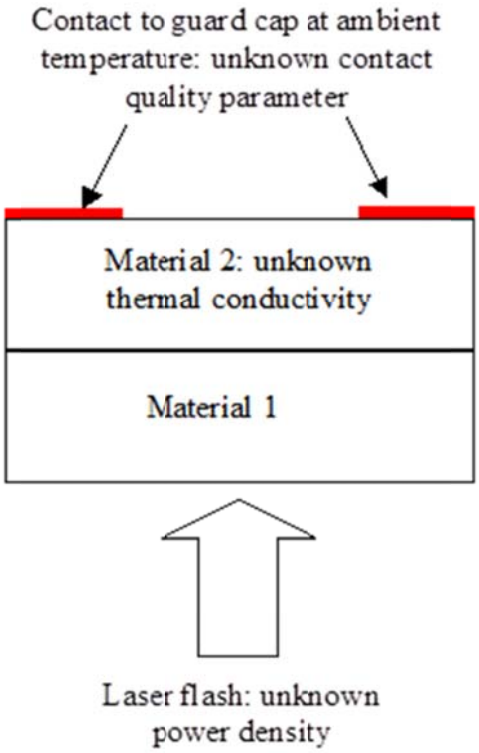

Figure 2: Sketch of geometry and boundary conditions within the model of the laser flash experiment.

The model is implemented as a finite volume model within Matlab. The contact between the sample and the apparatus is modelled as a boundary condition based on the assumption that the temperature of the apparatus remains fixed. The model has three unknown input parameters: the thermal conductivity of one layer λ, the laser power density I, and the thermal bond quality parameter β (units W m^{-2} K^{-1}) for the contact between the sample and the apparatus. The original work using this model also investigated the emissivity of the rear face of the sample [6], but the work showed that the model results were not sufficiently sensitive to emissivity for a value to be identified. Hence the work reported here fixed ithe emissivity value and concentrated on the other three parameters. The outputs of the model are a set of values of the temperature at the centre of the rear face of the sample at regular time intervals.

Earlier work has shown that the model inputs are typically of the following orders of magnitude: $\lambda \sim 1$ W m^{-1} K^{-1}, $I \sim 10^8$ W m^{-2}, $\beta \sim 10^6$ W m^{-2} K^{-1}, with the last of these values being an upper bound rather than an expected order of magnitude. Optimisation algorithms are generally most efficient when their input values are of the same order of magnitude, so the model inputs I and β were

normalised by factors of 10^8 and 10^6 respectively (this approach had proved successful in earlier work). The normalised model inputs were constrained to lie within the range $[10^{-5}, 10]$.

The optimisation problem minimises the difference between the model outputs and a set of measurement data gathered from a real sample. The measurement data are temperature differences rather than absolute temperature values, the difference being expressed relative to the controlled ambient temperature at which the measurements were made. The temperature differences lie within the range 0-5 K. Earlier work showed that the use of relative differences led to misleading results because the values close to 0 dominated the fitting, so the objective function used in the work reported here minimises the sum of squares of the absolute differences between the measured values and the model results.

The experimental data set was sampled to produce a representative subset of 101 points. The sampling was necessary to satisfy constraints on vector size within one of the tested packages. The resampling meant that the optimal solution for the test problem in this work and the problem studied in earlier work were slightly different. In order that the optimal solution was known, the test problem was solved using Matlab's optimsation algorithm lsqnonlin, an implementation of the Levenburg-Marquardt algorithm. The optimal values found were $\lambda = 3.602$ W m^{-1} K^{-1}, $I = 1.665 \times 10^8$ W m^{-2}, and $\beta = 2.030 \times 10^4$ W m^{-2} K^{-1} .

Both packages tested had a direct interface with Matlab, meaning that only the Matlab file defining the model was required to run the optimisation.

3 THE PACKAGES

Two packages were tested: OptiY [18] and modeFRONTIER [17]. These packages were chosen because they were general and not associated with a particular FE tool, and because free trial copies were available. The package DAKOTA [19] from Sandia National Laboratories is a well-established alternative that was not tested. This package includes optimisation and uncertainty quantification tools, runs from a command-line interface, and can be interfaced with general packages for generation of model results. The package was not tested due to time constraints, but is worth evaluating in future as it is free to download and looks extensive. Other packages were not included because free versions were not available, there was a lack of supplier response, or there were strong links to a single FE package (since the intention is to test generic tools).

3.1 MODEFRONTIER

To quote its website [17], "modeFRONTIER is a multidisciplinary and multi-objective software written to allow easy coupling to any computer aided engineering tool". The software was originally developed as part of an EU funded project in the late 90s, and the first commercial version of the software was released in 1999. The software is distributed world-wide by local offices, including a UK office. The website includes case studies across a wide range of industries, including automotive, aerospace, biotech, and general manufacturing.

The package includes interfaces to many CAE front ends, including Abaqus, Ansys and common CAD packages such as Solidworks, as well as interfaces to Excel and Matlab and tools to handle general text file interfaces, DOS commands, and various types of script, and calculator formulae.

The optimisation tools include robust optimisation and metamodelling tools as well as standard and advanced algorithms, and the user can extend the range of tools using their own Matlab, Scilab or Octave algorithms. Preliminary tools for initial explorations include a wide variety of design of experiments (DOE) methods, from fully factorial to stratified sampling methods, and the optimisation algorithms include classical gradient based algorithms, genetic algorithms and other nature-derived algorithms such as particle swarm optimisation [11], and simulated annealing methods [12]. The work

reported here used Nelder-Mead simplex search [13], Levenburg-Marquardt [2,3], and BFGS [1] methods from the classical algorithms, and genetic algorithms (GA) [14].

The software is entirely GUI driven. A picture of a typical project workflow is shown in figure 3. The icons represent files, software packages or actions, and can be linked by dragging from the input and output points on each icon. The workflow shown is the most complicated one we tested, but the case studies on the modeFRONTIER website illustrate that far more complicated processes can be investigated.

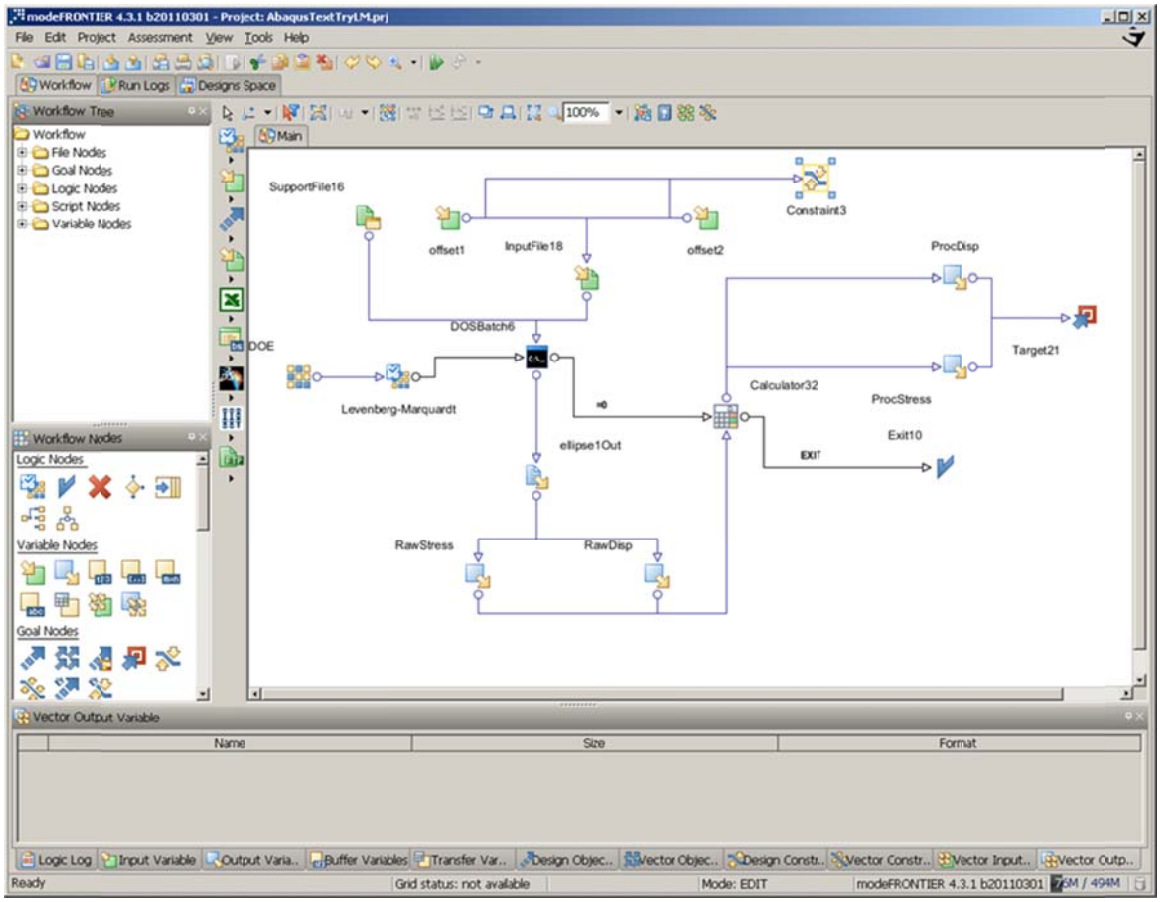

Figure 3: A typical modeFRONTIER project workflow.

A wide range of post-processing tools is available for interpreting the results. These tools include plotting tools, response surface methodology and meta-modelling tools, and tools for analysis of correlation. The work here has not investigated the full range of tools due to time constraints. An example screen shot of a visualisation tool is shown in figure 4. The figure shows, for a Latin Hypercube Sampling DOE analysis applied to the elliptical membrane problem defined in section 2:

- the pdfs of the feasible points for the input variables (upper two diagonal entries)
- the pdfs of the output variables (lower two diagonal entries)
- the correlation matrix (below the diagonal)
- scatter plots with each pair of variables (above the diagonal)

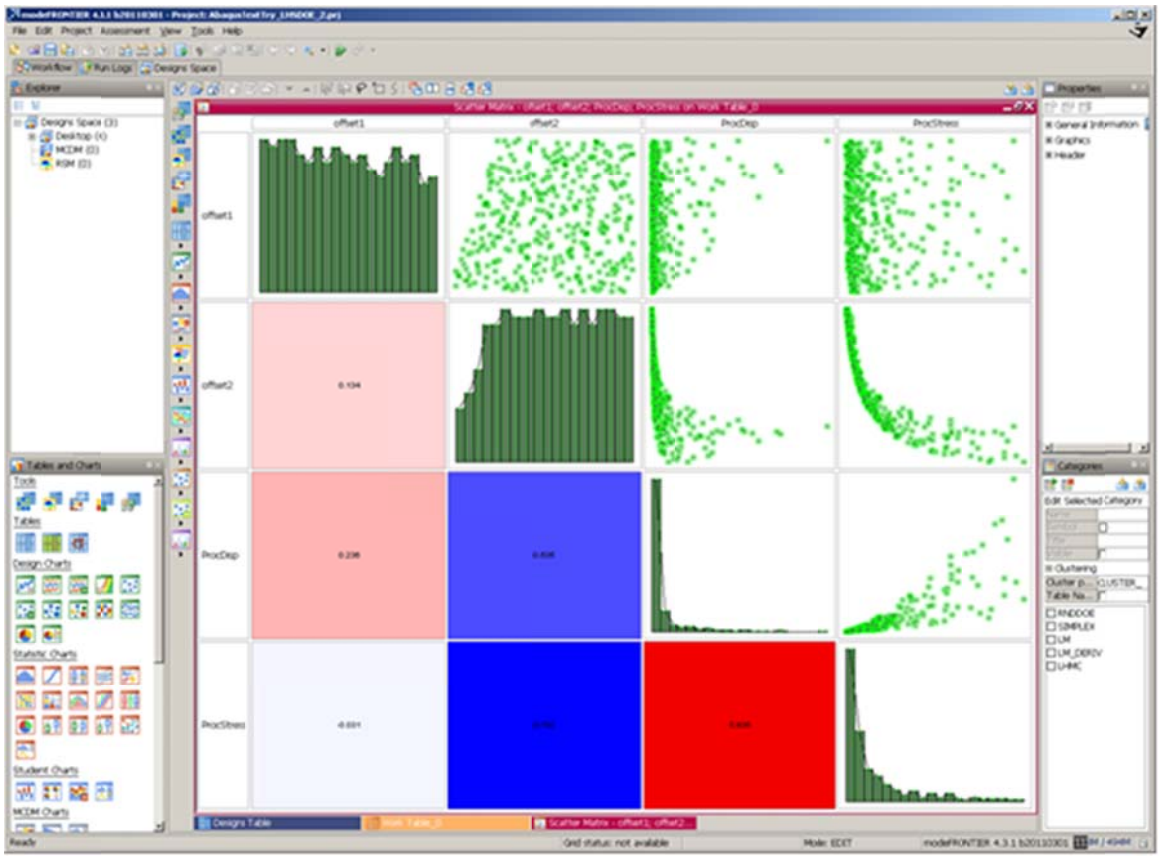

Figure 4: Example of a post-processing tool within modeFRONTIER.

3.2 OPTIY

OptiY [18] is an optimisation environment that can be used to carry out optimisation in combination with external simulators or a set of objective functions defined as subroutines and scripts, or even scripts of a black-box type. It also includes algorithms for probabilistic analysis (sensitivity, uncertainty evaluation, etc.). The company that develops and maintains the software is a spin-out company of TU Dresden. The first commercial version of the software was released in 2005. Its website lists a wide range of case studies, largely focussed on robust design and probabilistic assessment of life expectancy of products. Most of the examples use FE and CFD models and are from a range of industries.

The package is entirely GUI-driven. A picture of a typical screen is shown in figure 5. The different quantities and actions within a project are represented by boxes, and the linkings between the boxes represent the project flow.

The algorithms used in the testing were the Hooke-Jeeves pattern search [10], an evolutionary strategy (similar to a genetic algorithm), a grid-search method, a "simulation" method (essentially a completely random search across a space) and an adaptive response surface. The Hooke-Jeeves search, commonly used for local optimisation, is set as the "standard" method for lower dimensions (fewer than 10 variables) and the evolutionary algorithms are the standard settings for higher dimensions.

In addition to the optimisation capabilities, OptiY has robust optimisation capabilities (using Gaussian or uniform distributions for the input quantities) and design of experiment features, including Latin hypercube, Monte Carlo and Sobol samplings.

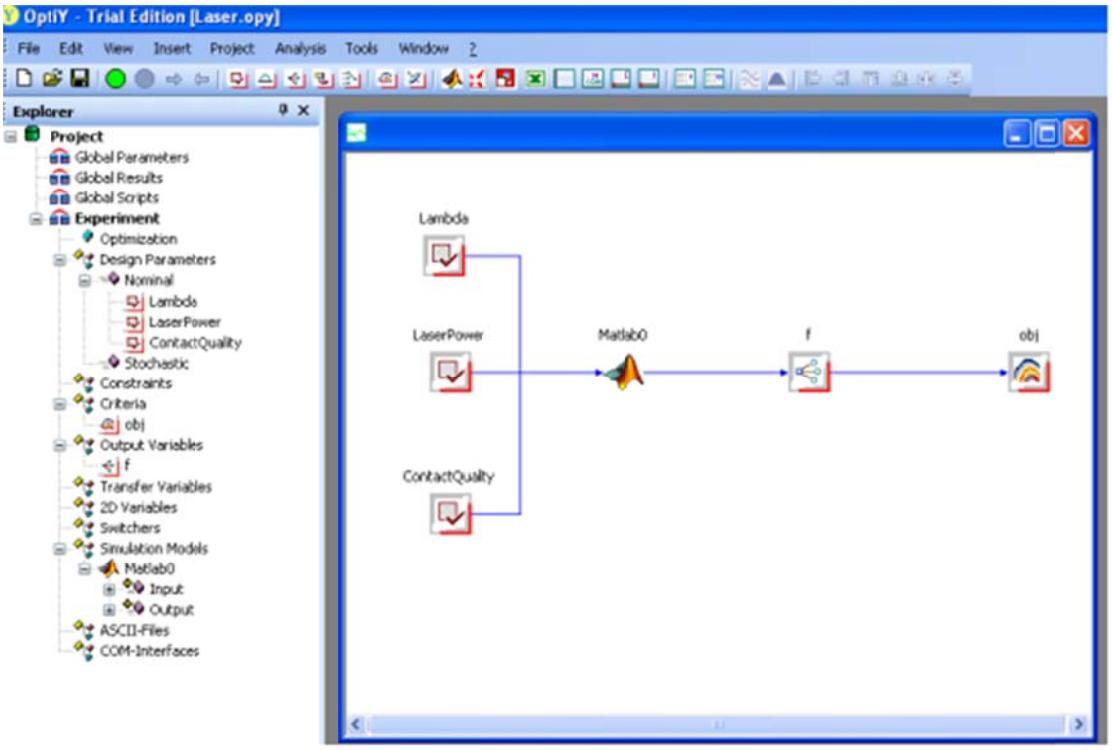

Figure 5: A typical screen-shot of an OptiY project.

The post-processing and visualisation options are restricted to comparatively basic graphs.

4 RESULTS

4.1 EASE OF USE

ModeFRONTIER defines its project flow through a set of linked icons as shown in figure 3. The project structure is easy to set up, and most of the icons are either intuitively simple to use or are well-described in the manual. Initially the user had some trouble identifying how to set up some linkages, particularly for the text-file case, but the support services helped explain the processes in more detail. If there had not been licensing concerns, the Abaqus interface within modeFRONTIER would have made the use of Abaqus simple (this has been demonstrated using a separate problem not reported here). The examples in the manual are also helpful, but in some cases the language is not easy to follow. Linking directly to Matlab was straightforward and the interface with the various text files was intuitive and easy to use.

OptiY similarly has a GUI driven by a visual representation of the project flow, shown in figure 5. It is relatively easy to set up a simple project flow using the available tools, although OptiY does not have a "click and link" feature unlike modeFRONTIER, and this lack can make building the correct links somewhat frustrating. Some of the edit functions are also frustrating, requiring use of clickable buttons where key strokes would be simpler (an example being the means of entering algebraic operators). Once the basic approach was understood, setting up the required structures for both jobs was straightforward. Linking to Matlab directly and to Abaqus via text files was straightforward. The examples on the OptiY website were a great help in setting up appropriate project structures.

Small tests of the modeFRONTIER sampling, robust optimisation and response surface tools have been run, but a properly-defined test has not been possible in either case due to time constraints. The tools are easy to use and to set up. The response surface was not successful due to poor user choices: the user needs to have an understanding of the options available if meaningful results are to be generated. The robust optimisation did not use a suitable test problem due to poor planning. No conclusions can be drawn about any of these aspects of modeFRONTIER.

As was stated in section 2, simple bounds were placed on the unknown parameters in all cases, and a (linear) constraint was included in the elliptical membrane job. Some of the algorithms attempted to evaluate out-of-bounds or unfeasible points, and in some cases this led the underpinning model to crash, but modeFRONTIER is sufficiently robust to return an error code for the crashed job and continue with the optimisation process. In general the error messages are helpful and informative.

The error reports in OptiY caused the user significant problems. Errors do not always cause error reports, and the contents of the error reports that do appear are not often helpful. The manual is of limited help with support for errors. Poor error message design is often a problem with early versions of software packages, because the error messages are largely there to assist the developers, who already know how the package works and so can interpret messages that can seem vague to less experienced users. Hopefully subsequent versions will address this issue.

The error catching process in OptiY is also not helpful to the inexperienced user. In the most serious cases, a job can be run even when objective functions or constraints are not defined correctly. The software removes the links relating to the badly-defined objects and runs the job anyway. The results of such a job run are meaningless and the process is a waste of time and computational resource.

4.2 GENERAL DISCUSSION OF OPTIMISATION ALGORITHMS

More details of the algorithms will be given in the next section, but it is worth making some general points separately.

ModeFRONTIER requires a DOE set to be associated with each optimisation process. For "population" type algorithms such as genetic algorithms and particle swarm algorithms, this DOE set defines the initial population. For the simplex algorithm with n unknowns, the initial simplex is given by the first n+1 DOE set members. For algorithms requiring only an initial estimate, the algorithm is run m times for a DOE set with m points, using each point as an initial estimate. A random DOE set was defined in all cases.

It should be noted that this DOE approach is unlikely to be a wise choice for the simplex algorithm and is likely to have affected the convergence, but time did not permit rerunning of the relevant jobs. The Nelder-Mead Matlab implementation is generated by perturbing a single point in each of the parameter directions. It is not obvious which approach is best since the former is likely to give a simplex that spans the space more effectively, but the latter is more likely to create a geometrically sensible shape.

Within modeFRONTIER, the classical algorithms requiring gradients to identify a search direction use central differences to numerically approximate the required gradients. This approach is often more stable than use of forward differences, but can be computationally expensive. The approach can also strongly depend on the step size used to approximate the derivatives. No attempt was made to vary this during the tests: default options were used throughout. In general, algorithms converged according to internal but user-definable criteria. The exception was the GA, which ran for a user-specified number of function evaluations.

The choice of optimisation algorithms in OptiY is potentially somewhat limiting compared with the range available in modeFRONTIER, with none of the common gradient search methods implemented. As would be expected, the grid search and random search algorithms were not effective for optimisation problems since they are more DOE design approach than optimisation algorithm.
The constraint choices in OptiY are similarly limited: only simple bound constraints can be used, with no options for constraints involving more than one variable. This limitation made it impossible to test the elliptical membrane problem as defined. The problem was implemented without the constraint, in the expectation that any resulting problems with the FE model would not be so severe as to crash the entire optimisation process.

The stopping criteria for the optimisation algorithms in OptiY can be defined either as conditions on the change of variable or objective function values, or as a condition on the number of function evaluations. The second option is not true convergence, but can allow the user to obtain the best result possible in a fixed time. The software includes a "continue" button that restarts a stopped optimisation run from the current optimum.

4.3 ELLIPTIC MEMBRANE TEST RESULTS

The results of the tests using the elliptic membrane problem are shown in table 1. "Correct runs" here are those runs that converged successfully to the expected values of the input quantities. "Number of function evaluations" is the number of calls of the main model made before the algorithm converged or stopped. For some algorithms, in particular the GA, this is not a meaningful quantity since the user defines a stopping criterion based on the number of function evaluations. "St. dev." means standard deviation.

Table 1: Results of the optimisations using the elliptical membrane model.

Package & algorithm	Total number of runs	Number of correct runs	Average number of function evaluations	St. dev. of number of function evaluations
OptiY, Hooke-Jeeves	10	4	177	82
OptiY, Grid search	10	1	441	147
OptiY, Random search (simulation)	10	2	303	119
OptiY, Evolutionary algorithms	10	3	245	124
OptiY, Adaptive response	10	4	291	93
modeFRONTIER, BBFGS	10	8	135	68
modeFRONTIER, Levenburg-Marquardt	10	8	129	301
modeFRONTIER, Nelder-Mead	1	1	128	
modeFRONTIER, Genetic algorithm	1	1	1500	

The OptiY user noted that the success or failure of a given run strongly depended on the location of the initial estimate, whereas this seemed to have much less of an effect on the performance of the modeFRONTIER algorithms. In particular the Hooke-Jeeves pattern search algorithm is very good for local optimisation but less so for global optimisation.

It would be possible to obtain better results from the OptiY algorithms by using the "continue" button. Values quoted are for the first convergence of the various algorithms without use of continuation. A significant proportion of the points tested in the OptiY evolutionary algorithm were not feasible as they violated the constraint $b - a > 0.25$, thus wasting computational effort. The OptiY grid search and random search methods performed very poorly, but this is to be expected as they are little better than random guessing. These algorithms may be more suited to acting as local search techniques once the most likely region for existence of a minimum has been identified.

The figures for the modeFRONTIER Levenburg-Marquardt runs are heavily biased by a single run that got stuck without converging. If the figures for that run are ignored, the average number of function evaluations is 34 and the standard deviation is 21. Whilst the modeFRONTIER GA used

1500 evaluations, the population had largely converged to the correct solution after about 350 function evaluations.

In general these results suggest that the modeFRONTIER algorithms are more robust than those in OptiY, and that they require fewer function evaluations to reach a confirmed solution. It is likely that the OptiY routines would perform well on small-scale bounded local problems rather than large-scale global searches.

4.4 LASER FLASH TEST RESULTS

Extra optimisation runs were carried out using the Matlab implementations of the Nelder-Mead and Levenburg-Marquardt algorithms. These extra runs allow an independent assessment of the algorithms in OptiY and modeFRONTIER against what could be regarded as benchmark implementations, and were easy to carry out since the laser flash model is a Matlab routine. Table 2 shows the full results for all algorithms, using the same nomenclature as table 1.

Table 2: Results of the optimisations using the laser flash model.

Package & algorithm	Total number of runs	Number of correct runs	Average number of function evaluations	St. dev. of number of function evaluations
Matlab, Levenburg-Marquardt	10	9	87	51
Matlab, Nelder-Mead	10	3	285	114
OptiY, Hooke-Jeeves	10	5	191	79
OptiY, Grid search	10	2	574	261
OptiY, Random search (simulation)	10	2	418	145
OptiY, Evolutionary algorithms	10	4	289	192
OptiY, Adaptive response	10	3	323	117
modeFRONTIER, BBFGS	9	3	306	180
modeFRONTIER, Levenburg-Marquardt	10	10	161	17
modeFRONTIER, Nelder-Mead	2	1	141	
modeFRONTIER, Genetic algorithm	1	1	940	

It might be expected that the algorithms would require more function evaluations for this test problem than the elliptical membrane, because the elliptical membrane has two variables to be determined whereas the laser flash model has three. With the exception of the modeFRONTIER BBFGS algorithm this expectation seems not to be correct: the average number of function evaluations differs little between the two problems for a given algorithm.

Several of the BBFGS runs (four out of nine) got stuck in a valley of local minima where the results were insensitive to the value of β. This local behaviour led to large numbers of function evaluations before convergence and poor success rates, which may explain why the figures in tables 1 and 2 differ so much for that algorithm.

In the results above, the Matlab Levenburg-Marquardt figures are strongly affected by a single tun that converged very slowly to the wrong solution. If the average and standard deviation are calculated without the non-convergent run, the values are an average of 72 and a standard deviation of 17.

It should also be noted when comparing the two implementations of the Levenburg-Marquardt algorithm that the Matlab version uses forward differences to determine gradients where the modeFRONTIER version uses central differences. Hence for a problem with n variables, for each point visited Matlab will evaluate n+1 values and modeFRONTIER will evaluate 2n+1 values.

For this model, n=3 and so it would be expected that the ratio (Matlab average function evaluations)/(ModeFRONTIER average function evaluations) should be about $4/7 \approx 0.57$. The actual value is $87/161 \approx 0.54$, suggesting that the Matlab and modeFRONTIER algorithms are equally efficient in terms of points tested. It is likely that modeFRONTIER's use of central differences would bring strong stability advantages for problems with noisy response surfaces, but this hypothesis has not been tested.

In general the performance of the algorithms is similar to their performance applied to the elliptical membrane problem, suggesting that the response surfaces associated with the two problems share characteristics such as smoothness and absence of multiple local minima.

5 CONCLUSIONS

modeFRONTIER and OptiY fulfil the same functionality as the previously proposed FE toolbox. The structures of the two tools are very similar, using a GUI-driven approach based on a visual representation of project flow. The main difference between the two tools is that modeFRONTIER has a significantly wider range of functionality, probably due to a longer development time. modeFRONTIER offers a better range of algorithms, a more robust error-catching system, and a wider range of visualisation tools than OptiY. It has the added advantage of a dedicated UK team who offer excellent support for users.

The tools offered in OptiY seem best suited for local optimisation, and so could be of benefit when small changes to an existing design are required rather than wider-ranging step changes in a design.

NPL has recently learnt from discussions with suppliers that iSight, a tool with strong links to Abaqus, can also be used in conjunction with other packages.

The existence of these packages, and other similar ones, makes creation of a new toolbox technically unnecessary and commercially unviable.

The key next step is to decide whether there is a business case for buying any of these packages. We need to identify applications within NPL, particularly applications involving FE and other computationally expensive models, where optimisation and uncertainty evaluation are being carried out by hand, or are not being carried out at all due to lack of tools. This case will be developed by contacting FE users and asking if they would benefit from such a tool, and by attempting to assign a financial value to the time saved.

6 REFERENCES

[1] R. H. Byrd, P. Lu and J. Nocedal. A Limited Memory Algorithm for Bound Constrained Optimization (1995), SIAM Journal on Scientific and Statistical Computing, 16, 5, pp. 1190–1208.

[2] 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.

[3] 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.

[4] T. J. Esward, C. E. Matthews, L. Wright, and X.-S. Yang. Sensitivity analysis, optimisation, and sampling methods applied to continuous models. NPL Report MS 2, National Physical Laboratory, 2010.

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

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

[7] L. A. Chapman, C. E. Matthews, S. J. Roberts, L. Wright, and X.-S. Yang. Parameter estimation from Laser Flash Experiment Data. Chapter in Computation Optimization and Applications in Engineering and Industry, Springer-Verlag, 2011.

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

[9] G. A. O. Davies, R. T. Fenner, and R. W. Lewis. NAFEMS Background to Benchmarks. NAFEMS, Glasgow, 1993.

[10] R. Hooke, T. A. Jeeves. 'Direct search' solution of numerical and statistical problems. Journal of the Association for Computing Machinery 8 (2): 212–229 (1961) .

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

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

[13] 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.

[14] M. Mitchell. An introduction to genetic algorithms. MIT Press, Cambridge, Mass., USA, 1996.

[15] BIPM, IEC, IFCC, ILAC, ISO, IUPAC, IUPAP, and OIML. 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: 2008.

[16] BIPM, IEC, IFCC, ILAC, ISO, IUPAC, IUPAP, and OIML. Guide to the expression of uncertainty in measurement. Joint Committee for Guides in Metrology JCGM 100: 2008.

[17] http://www.esteco.com/home/mode_frontier/mode_frontier.html

[18] http://www.optiy.eu/default.htm

[19] http://dakota.sandia.gov/