

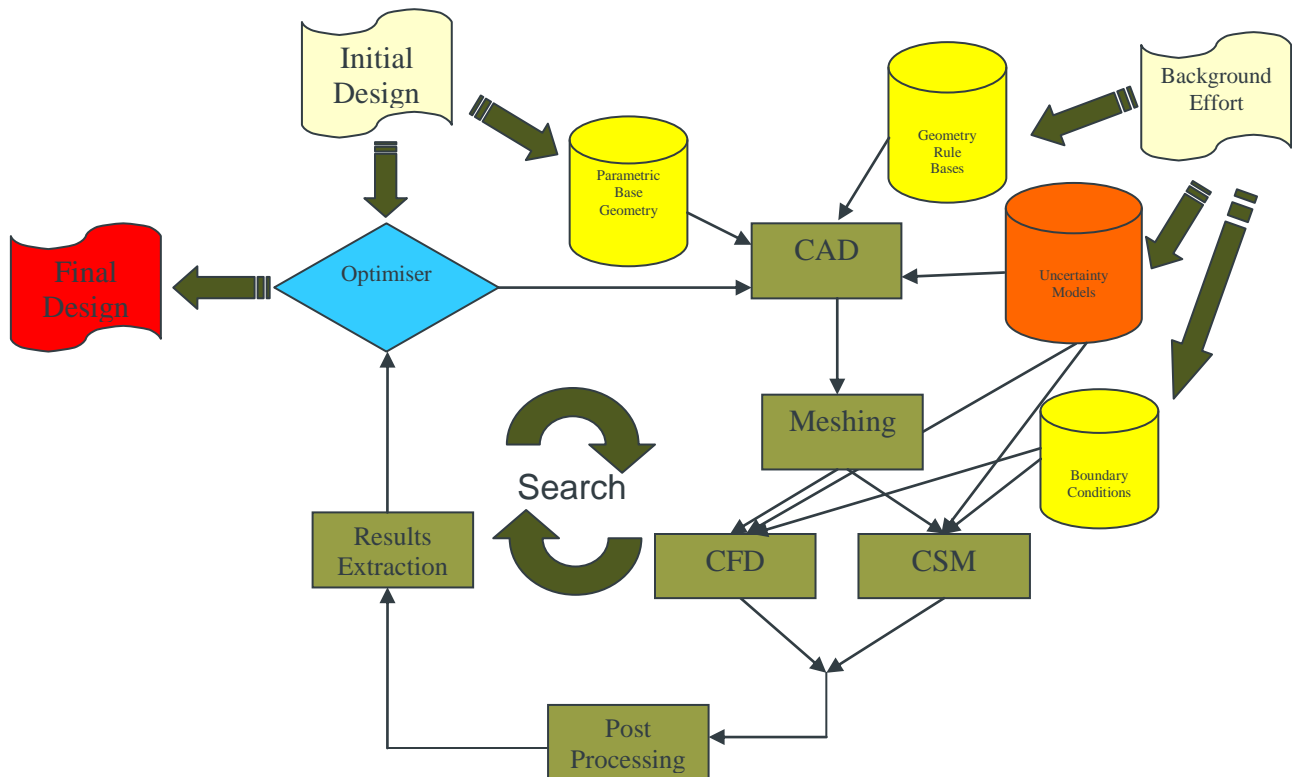
An Overview of Computational Methods in Engineering Design

Andy Keane

Director of the Rolls-Royce University Technology Centre for Computational Engineering at the University of Southampton, UK

Computationally led engineering design is based around the ability to predict the performance of a product before it is built, using simulations of various kinds.

Such simulations vary enormously but in much of the mechanical and aerospace engineering world they involve the specification of the design's geometry followed by the application of various methods stemming from the fields of Computational Fluid Dynamics (CFD) or Computational Structural Mechanics (CSM). In the majority of cases the techniques used involve some form of discretization of the problem being dealt with, commonly referred to as “meshing”, followed by finite difference, finite volume or finite element solution of the relevant equations describing the physics of interest, bearing in mind any problem specific boundary conditions. Having then solved the (often many millions) of resulting algebraic equations, some form of post-processing and results extraction is normally required to gain an overview of the performance quantities of interest such as drag, maximum stress, life, etc.



This fundamental process is illustrated by the gold and bright-yellow coloured boxes in the diagram above. In this case the geometry is specified in a parametric Computer Aided Design (CAD) package controlled by a separate rule-base, although for some purposes other ways of describing the design of interest may be preferable.

To allow for a rapid consideration of various design alternatives it is highly desirable to automate this central analysis process as far as possible. This requires two slightly different aspects to be considered. First, there needs to be some mechanism whereby alternative geometries can be specified – a so called “parameterization” of the geometry. Secondly there must be some way of managing the sequence of tasks required – this is often referred to as the “workflow”. The workflow itself needs to be dealt with at various levels of granularity. At the topmost level it simply needs to reflect the arrows in the diagram saying in which order tasks are to be carried out. But clearly it will also need to set out where any data-files are to be stored and which computers and codes are to be invoked. At a more detailed level it will also need to contain explicit instructions for the running of each code invoked. These detailed instructions need setting out with much care as the codes will, by definition, be used to analyse a range of geometries and what may work well in one case, may fail for a slightly changed geometry. In fact a very effective and simple test of any automated workflow is to first check that some base or initial design will happily pass through and then to make a small set of random perturbations to the geometry and see how many of these will also solve correctly. This “visit to the gym” is a wise health-check of any automated and parametric design chain.

Once the necessary health-check has been cleared it then becomes possible to carry out a series of design improvement processes. Perhaps the simplest, but often overlooked, approach is to test out the base design against variations in operating conditions, or to allow for likely uncertainties in manufacture or wear. This can be carried out by making a series of perturbations that reflect likely variances and represents a simple extension to the “gym test”. Such a process will allow the designers to understand how robust their design is and give insight into its sensitivity to various possible unknowns. These perturbations are reflected by the orange element in the diagram.

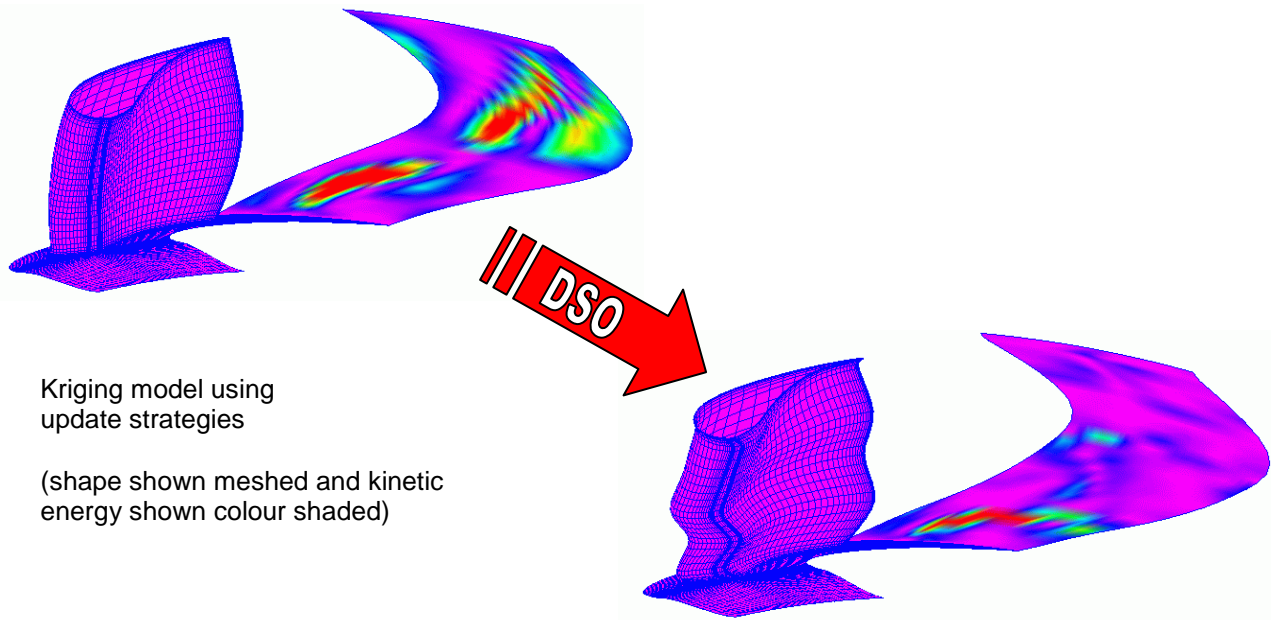
Following on from a simple robustness study of the base design we come to formal design improvement schemes. These all hinge around the basic idea of Design Search and Optimization (DSO). DSO is a branch of applied mathematics and computer science involved with the manipulation of design inputs (the variables in the parametric CAD model for example) so as to improve specified goals subject to any given constraints (the results extracted in the final gold box of the diagram). There are a wide range of approaches that can be adopted for such work and these are summarized in many books on this subject (see for example “Computational Approaches for Aerospace Design: The Pursuit of Excellence” by Keane and Nair, www.southampton.ac.uk/~aerodesn).

In all cases the basic strategy is the same. First a series of designs are studied to build an initial “map” of the design space. This can be very local in nature, resulting from very small and systematic perturbations to gain local gradients of goals with respect to variables as in traditional hill-climbing methods through to the wide ranging, statistically designed surveys used in response

surface approaches. Having sampled the design space some kind of internalized or explicit understanding of how variables affect goals and constraints is constructed (a “map” if you will) which can then be used to suggest new design variants for analysis. These new designs are then analysed, in parallel if possible and if the available computing infrastructure permits, before updating the map so as to allow for improved understanding and hopefully further gains in performance. This process is represented by the “Search” term in the diagram and is controlled by the (blue) optimizer element.

There are many variations on how to conduct such searches and what to allow for when carrying them out. For example the approach followed will vary depending on whether there is a single clear and unambiguous goal being pursued (rare) or if there are multiple goals (much more likely). Approaches will also vary according to the nature of any constraints on results or limits on variables. It is also possible to retain uncertainty perturbations during search so as to study mean design behaviour as opposed to nominal and also to permit the explicit inclusion of variances in performance either as a goal or a constraint. Perhaps the most important consideration in DSO is the balance between the number of variables, the number of goals, the computational cost of evaluating a design and the available computing power. It is quite common for designers to wish to vary hundreds of variables over wide ranges with tens of goals using analysis codes that takes hours to solve while having access to only modest computational power. In such circumstances even with the most refined of parallel DSO studies compromises will need to be made. DSO methods using expensive codes tend to work best if no more than 20 variables are used, at least to begin with. It is also wise to start with modest ranges for these variables that prevent designs being too radical as this may cause either problems with meshing or invalidate the underlying assumptions behind the analysis chain. Using more than two goals can also cause problems in understanding results and it is often better to set any other goals in the form of constraints, for example light weight may be a goal but it can also be handled by setting a maximum acceptable weight as a limit. Similar approaches can be taken for cost or drag for example.

The following figure illustrates the results of a typical DSO study, carried out using the optimization toolkit incorporated in the dezineforce service, for Rolls-Royce. Here a guide vane design was optimized using a CFD code and 21 design variables at seven radial sections to reduce secondary kinetic energy in the outflows. This process was run in parallel using 20 processors and required 100 design evaluations, looping around the DSO process five times. This resulted in a 50% reduction in secondary kinetic energy, with a vane design that is not intuitive, and thus unlikely to have been arrived at by exercise of designer judgement alone. In this case the DSO approach used was a statistical design of experiments followed by a Kriging response surface model with updating, a strategy now commonly adopted by the world’s leading aerospace and F1 race car designers.



To conclude this brief introduction to DSO and design improvement it should be stressed that the word “Search” in DSO is included to make clear that DSO cannot guarantee to produce an “optimal” design – rather the process aims to *improve* designs given the resources available and the quality of the models and analysis processes. If the results coming from computational methods are not fully credible across the range of design being studied clearly the results of any DSO runs will be suspect. Optimizers are notoriously good at finding shortfalls in an analysis and exploiting them so as to “improve” the measures of merit being used. It is quite common to have to carefully monitor a DSO run to ensure that high quality results are being provided by the analysis chain. Sometimes design teams fail to include all their constraints when setting up problems and if they then simply wait for the DSO run to complete find that the resulting designs do not meet their desires. Equally, it is possible for a DSO process to push an analysis beyond the range of validity that it was created for. Even when all goes well the design team must take responsibility for their work and ensure they understand why any design improvements have been achieved so that they are sure that their products are fit for purpose. Despite these caveats it is however safe to say that DSO is now routinely used by many large companies and that it offers very great advantages in terms of both speed of design and quality of outputs.

The incorporation by dezineforce of DSO in its subscription-based service now brings DSO within the reach of small companies for the first time.