IP011817-2

TUNCONSTRUCT

Technology Innovation in Underground Construction

Integrated Project

Thematic Priority 3, NMP

Deliverable 1.3.2.1: Specifications for software to determine sensitivities for optimization of the design of underground construction as part of IOPT

Due date of deliverable: May 2006
Actual submission date: June 8th 2006

Start date of project: September 1, 2005 Duration: 48 months

Lead contractor: C3M
Author: Igor Grešovnik

Revison 3

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination level		
PU	Public	X
PP	Restricted to other programme participants (including Commission services)	
RE	Restricted to a group specified by the consortium (including Commission services)	
CO	Confidential, only for members of the consortium (including Commission services)	

INDEX

1 Abstract

The purpose of this deliverable is to specify methods and software to compute sensitivities as integral part of optimisation procedures that will be applied in TUNCONSTRUCT. Within SP1, applications of sensitivities will be focused on Back analysis of geological condition, Upscaling of laboratory data and the development and application of optimization procedures.

The sensitivity analyses can be performed in two ways. The first possibility is numerical derivation of response functions. Numerical differentiation is subject to numerical noise, which is inevitable in complex numerical computations. Application of approximation based sensitivity analysis procedures is intended for alleviation of problems related to instability of numerical differentiation in presence of noise. This approach is presented in combination with the approximation based optimisation techniques because close integration with optimisation concept is envisaged. It is anticipated that optimization techniques based on successive approximation of the response functions will be mainly used in TUNCONSTRRUCT, where derivatives of the successive approximated problems will be used in the minimization procedures.

The second possibility is to provide sensitivities directly as a result of numerical simulation, for which analytical differentiation of the numerical model with respect to design parameters must be performed. In this case, sensitivity analysis cannot be treated independent of the direct analysis and must be incorporated within the analysis code. Basic concepts of analytical differentiation of numerical models are presented in this deliverable.

Main objective of back-analyses in TUNCONSTRUCT is inverse reconstruction of geological conditions at the tunnel site, which includes material parameters for the surrounding rock mass, stress state before excavation and orientation of geological layers. In this document main features of the sensitivity procedures are demonstrated on a synthetic example that will be used for development and validation of procedures for back analysis as well as to demonstrate how sensitivities with respect to shape, material parameters and boundary conditions can be evaluated in an optimisation scheme.

Parameterisation according to various requirements has been prepared and the response functions for back-analysis problems defined. The sensitivity analysis for different scenarios was performed by analytical differentiation of the finite element model. The problem was set up in the simulation environment that enables automatic generation of the finite element code by utilising symbolical derivation of the element level expressions and ready incorporation of the generated code in the global simulation framework.

Considering manpower resources and software available in the TUNCONSTRUCT consortium it is evident that analytical differentiation methods are feasible only in conjunction with AceGen system, which includes symbolic system for automatic generation of finite element codes.

For sensitivity analyses where objective and constraint functions are evaluated by other solvers (EKATE, BEFE++, FLAC3D, ELFEN) numerical differentiation methods will be applicable. Precise specifications how these solvers could be integrated in sensitivity and

optimization procedures are described in Section 6. Note that necessary conditions to apply inverse and optimization procedures are parametric description of the direct problem, reliable evaluations of responses for different sets of parameters and provision of objective/constraint functions according to the specification. For sensitivity analyses in TUNCONSTRUCT it is required that direct models are set up by the partners who must ensure that their direct problems can be run automatically based on parameterized data from UCIS.

Section 7 contains a description of a test case for validation of back-analysis procedures using a synthetic test case with assumed conditions at a tunnel construction site.

2 Outline of the document

This deliverable gives specifications for the software that computes numerical derivatives (sensitivities) within optimisation procedures applicable in TUNCONSTRUCT for the solution of upscaling and back analysis problems.

Firstly, a general mathematical basis and the computational methods suitable for the solution of optimization problems in connection with numerical models are described in Sections **Error! Reference source not found.** - 5. The precise specifications for the integration of simulation software with optimization modules is provided in Section 6. A test case has been set up in Section 7 to demonstrate actual realization of sensitivity procedures related to shape and material parameters as well as to parametrised boundary conditions. Finally, an outline of the planned integration within TUNCONSTRUCT and the links with other software developed within TC is provided in Section 8.

3 Introduction

We consider the case where optimisation problems are defined by response functions that are based on results of numerical analysis (e.g. finite element) of the considered system. The response functions include the objective function to be minimised and constraint functions that define the admissible sets of designs. Optimisation or design parameters specify the trial design (in the case of optimization) or assumed parameters of the physical model (in the case of inverse analysis), and determine the input for numerical analysis of the considered system. Derivatives of the response functions with respect to design parameters are used in gradient-based optimisation algorithms.

Mathematically, we state an optimization problem as

$$\begin{array}{lll} \text{minimize} & f(\mathbf{x}), & \mathbf{x}\in\mathbb{R}^n \\ \text{subject to} & c_i(\mathbf{x})=0,\ i\in E & \\ \text{and} & c_j(\mathbf{x})\ge 0,\ j\in I\ , & \\ \text{where} & l_k\le x_k\le u_k,\ k=1,2,...,n & \end{array} \tag{1}$$

In the above equation, *f* is the *objective* function, c_i and c_j are *constraint* functions and l_k and u_k are upper and lower parameter bounds. The second and third line of the equation are referred to as equality and inequality constraints, respectively. We will collectively refer to *f*, c_i and c_j as *response functions*.

Analytical differentiation of finite element models is described first in a generalised way in Section 4. When available, analytical differentiation represents the most efficient and also reliable method for providing gradient information used in optimisation procedures. Implementation of derivatives in an existing finite element model is usually time consuming. This can be significantly alleviated by automatic generation of code based on symbolic derivation of expressions. A finite element system utilizing this approach is outlined at the end of Section 4.

Whether "analytical" derivatives are available or not will, among the others, depend on economical factors. Here the basic question is whether efficiency gains achieved by analytical differentiation outweigh the necessary implementation effort. When "analytical sensitivities" of the finite element model are not available, a logical approach would be to perform numerical differentiation of the response calculated by the direct model. The simplest and cheapest is the finite difference method where calculation of gradient of a response function includes perturbing parameters one by one and evaluating the response (including the complete finite element simulation) at the perturbed parameters. In order to calculate the response gradient, this method requires at least one additional evaluation of the response for each design parameter.

The main problem of numerical differentiation is related to the accuracy of the calculated derivatives, which critically depends on the step size used for parameter perturbation. For smooth functions calculated with unlimited precision, the increasing of the step size in general reduces the accuracy of the numerical derivatives because of increasing deviation of the function from its first order Taylor approximation. Because of limited precision, the step size can not be arbitrarily reduced and there exists an optimal step size for which precision of numerical derivatives is the highest. The optimal step depends on computational precision and higher order derivatives of the function. Because it varies with parameters and its computation would require extra effort, an often approach in practice is to guess a compromise step size that gives "reasonably good performance".

Finding an acceptable compromise for step size is often possible when the computational precision is high and there is a large range of step sizes that are sufficiently small with respect to function characteristics, but well above the limit where finite precision would cause substantial errors in numerical derivatives. When the evaluation of response functions involves complex numerical simulations, functions usually contain a high level of numerical noise that makes choice of an acceptable step size for numerical differentiation very difficult. In such cases, we can attempt to apply optimisation algorithms that don't require gradient information. However, these algorithms are usually significantly less efficient than those that are gradient-based.

Another approach is based on application of function approximation techniques. We begin with the assumption that the "true" response functions are smooth and therefore gradient-based optimisation algorithms could be efficiently applied to the problem. When we calculate the functions numerically, a certain level of noise is added to the "true" functions. One consequence of this is that the functions can not be evaluated with arbitrary precision, which also affect the precision to which the optimisation problem can be solved. Another consequence is of practical characters: the performance of numerical techniques such as optimisation and differentiation that would work (hypothetically, of course) fine on the “true” response may be completely deteriorated on the calculated response.

The first effect can only be treated by increasing accuracy of numerical computation, which increases computational time and can be achieved only to a limited extent. The approach envisaged to fight the second effect is to apply numerical algorithms not directly to noisy numerical response, but to smooth approximation of the response built on basis of its systematic sampling.

The sampling strategy and type of approximation used will depend on the purpose for which the response is used (e.g. optimisation). When there is no particular known model that would be especially suitable for description of the response functions, a weighted least squares approximation with low order monomial basis functions would be an intuitive choice. This rests on the same theoretical basis as the polynomial interpolation models, which is intrinsic also to most standard numerical differentiation procedures and many optimisation and other numerical methods – the Taylor expansion. The important distinction of approximation models as compared to interpolation is that the former can used an arbitrary number of samples (response evaluations) that are at least as large as the number of basis functions. Sampled function values are therefore not fit exactly, and this provides a mechanism to prevent large random fluctuations superimposed on the “true” response from spoiling completely the picture of the response (something we use implicitly when applying numerical differentiation or optimisation techniques). We can imagine this as a means of “levelling out” the noise. For “there is no free lunch” in the real world, the mechanism has its cost in evaluation of additional samples that are necessary to improve the image of the response, and its true value is in providing good control mechanisms for improving the response approximation. The spatial distribution of samples is technically much less important when using approximation models. This makes easy to add additional samples or to re-use existing samples for improvement of the accuracy of the approximation.

With regard to numerical differentiation, using approximation models preserves the dilemma about the choice of the step size, which corresponds to the choice of the size of the sampling region. When the size is too large the combination of the basis function can not approximate the “true response”, and when the size is too small, the approximation can not be stabilised against effects of random fluctuations. Control mechanisms provided by the weighting least squares approximation can be used for adaptive adjustment of the step size. Oversampling (i.e. calculation of the response functions in more points than minimum necessary for the approximation) provides means of estimation of accuracy of approximation, which governs the decision of step enlargement or reduction. Weighting samples according to their distance from the point of evaluation (e.g. of approximate response and its gradient) provide a mechanism of changing effective step size without completely discarding already calculated samples that fall out of the selected sampling region.

In accordance with the “no free lunch” paradigm, adaptive schemes for approximation based numerical differentiation can provide more reliable method on account of additional computational cost. In view of improving computational efficiency, we must consider numerical differentiation in combination with the purpose for which the computed gradients are used, e.g. for enabling gradient based optimisation techniques. The idea is to integrate numerical differentiation with optimisation in such a way that the excess computation required for adaptive step adaptation is incorporated in computational effort needed for the solution of the optimisation problem.

This idea is supported by the way in which efficient optimisation algorithms actually operate. E.g., the Newton's method[1] for unconstrained optimisation utilises the function value and its first *and second* derivatives in each step. Practical obstacles for application include the fact that starting solving optimisation problems within the convergence radius of the method is a rather exceptional situation in practice, and that computation of second derivatives is usually too expensive numerically instable. The quasi-Newton methods that are derived from the Newton's method are widely considered as the most efficient methods (especially the BFGS) for local unconstrained optimisation. These methods start with the gradient descent direction and utilise the line search prototype algorithm in order to ensure global convergence. In successive line searches, the approximation of second order derivatives[2] is gradually built up in the scheme resembling the finite difference numerical differentiation. However, the approximation is very rough since it initially uses large step sizes dictated by the line search. Its accuracy starts to improve only in the vicinity of the minimum where quadratic model is good approximation, taking in this way the advantage of the good local convergence properties of the Newton's method. In the area where good local quadratic approximation of the function would not contribute to algorithm performance, the time is not wasted in attempt of its construction.

Similar ideas are used for approximation based optimisation and sensitivity analysis. For better integration of approximation models, the restricted step prototype algorithm rather than line search is utilised as a framework for achieving global convergence. A the beginning, the attempt to maintain as large step size as possible is being made, rather than trying to make a good local approximation of the response. This corresponds to the large size of the sampling region and increased stability with respect to noise. Rather than by accuracy of approximation, the step size is controlled by success of the optimisation step measured by reduction of the value of the minimised function. Close to the minimum, good order of response approximation will contribute to good local convergence as far as this is permitted by the level of noise. When accuracy limit defined by the level of noise is achieved, this is detected on the basis of accuracy probing. Different kinds of breakdown (such as capturing in fictive minima that are in fact fluctuations) are prevented by stabilisation achieved by more distant samples.

For better adjustment to the above mentioned ideas, the *moving least squares* approximation model is considered. In this model, we allow that approximation coefficients are not constant but are parameter dependent. Their variation is defined by weighting functions, which assign weights to the samples according to relative distance from the point of evaluation. This gives raise to a local-global approximation that can adapt to the function over a large range of parameters as long as enough samples are available over the whole range, and behaves locally in a similar manner than the corresponding ordinary weighted least squares approximation.

Section 5 provides a brief description of approximation techniques that are used with approximation based numerical differentiation and optimisation techniques. Properties of the moving least squares approximation are demonstrated by instructive visual examples in

[1] This method is in some sense an equivalent to the Newton's method for equation solving , which is (usually with some modifications) widely used in engineering simulation techniques including the finite and boundary element methods, finite difference method and meshless techniques.

[2] In fact, approximation of inverse of the matrix of second derivatives is more commonly built up for efficiency reasons.

Section 5.4. The scope of discussion is limited on the case where only sampled response values are used. The approach can however be extended in order to incorporate eventual gradients that are provided by analytical differentiation of the numerical model[3].

Approximation based optimisation techniques have a large potential in terms of efficiency in the presence of noise, possibility of parallelisation, re-use of calculated information, combination with other response interrogation techniques, reliability based optimisation, limited time optimisation and other areas that are of particular importance from the practical engineering point of view. For making good use of this potential, ability of quick adaptation of solution algorithms to particular problems and requirements is especially important. The goal will be to design a library and of building blocks for solution of individual problems in such a way that they can be easily combined in relatively complex algorithms. This requires well considered internal design, which will also reflects in the application interfaces for optimisation and numerical differentiation algorithms. The specifications of these interfaces is given in Section 6.

Finally, a test example that has been set up for validation of numerical techniques used in back-analysis of geological conditions at the tunnel construction site, is described in Section 7.

Section 8 has been added that treats matters related to this deliverable and the two future tasks of C3M within the project - the upscaling and back analysis problem.

4 Analytical Differentiation of the Finite Element Model

4.1 Finite Element Simulations

The aim of numerical simulations is to predict the behaviour of a system under consideration. In the finite element approach this is performed by solving a set of algebraic equations, which can be expressed in the residual form

$$\mathbf{R}(\mathbf{u}) = \mathbf{0}\,. \tag{2}$$

The above equations represent the discretised form of the governing equations including balance laws, constitutive equations, and initial and boundary conditions, which arise in mechanical, thermal, or electromagnetic problems. Unknowns **u** define approximate solution and are considered as the primary system response. System (2) represents a wide variety of problems and description of finite element techniques to solve particular problems are beyond the scope of the present description. This section is focused on basic aspects of sensitivity analysis for nonlinear problems, which is crucial for efficient optimisation procedures.

The system (2) can be solved by the Newton-Raphson method, in which the following iteration is performed:

[3] Similar as with classical optimization techniques, availability of analytical gradients would be of great benefit for the improvement of efficiency. Separate numerical differentiation in order to provide gradients would not make sense on the other hand, since it is already implicitly incorporated in the approach (which is done in such a way that stability problems are overcome).

$$\frac{d\mathbf{R}}{d\mathbf{u}}\left(\mathbf{u}^{(i)}\right)\delta\,\mathbf{u} = -\mathbf{R}\left(\mathbf{u}^{(i)}\right), \tag{3}$$

$$\mathbf{u}^{(i+1)} = \mathbf{u}^{(i)} + \delta\,\mathbf{u}\,. \tag{4}$$

The term $\mathbf{R}\left(\mathbf{u}^{(i)}\right)$ is referred to as the residual (or load) vector and the term $\frac{d\mathbf{R}}{d\mathbf{u}}\left(\mathbf{u}^{(i)}\right)$ is referred to as the tangent operator (or tangential stiffness matrix).

For time dependent problems the iteration scheme given by (3) and (4) is not sufficient since the state of the system at different times must be determined. Time is usually treated differently to the spatial independent variables. The time domain is discretised according to the finite difference scheme in which approximate states are evaluated for discrete times ${}^{(1)}t, {}^{(2)}t, \ldots, {}^{(M)}t$. Solution for intermediate times is usually linearly interpolated within the intervals $\left[{}^{(n)}t, {}^{(n+1)}t\right]$ and time derivatives of the time dependent quantities are approximated by finite difference expressions.

The approximate solution for the *n*-th time step is obtained by solution of the residual equations

$${}^{(n)}\mathbf{R}\left({}^{(n)}\mathbf{u}, {}^{(n-1)}\mathbf{u}\right) = \mathbf{0}\,, \tag{5}$$

which are solved for each time step (or increment) for ${}^{(n)}\mathbf{u}$ while ${}^{(n-1)}\mathbf{u}$ is known from the previous time step. Dependence on earlier increments (${}^{(n-2)}\mathbf{u}$, etc.) is possible when higher order time derivatives are present in the continuum equations. The system (5) can again be solved by the Newton-Raphson method in which the following iteration is performed[4]:

$$\frac{d\,{}^{(n)}\mathbf{R}}{d\,{}^{(n)}\mathbf{u}}\left({}^{(n)}\mathbf{u}^{(i)}\right)\delta\,\mathbf{u} = -{}^{(n)}\mathbf{R}\left({}^{(n)}\mathbf{u}^{(i)}\right), \tag{6}$$

$${}^{(n)}\mathbf{u}^{(i+1)} = {}^{(n)}\,\mathbf{u}^{(i)} + \delta\,\mathbf{u}\,. \tag{7}$$

The incremental scheme is not used only for transient but also for path dependent problems such as plasticity where constitutive laws depend on evolution of state variables, which inherently calls for an incremental approach. Material response is not necessarily time dependent and the time can be replaced by some other parameter, referred to as pseudo time. Treatment of path dependent material behaviour requires introduction of additional internal state variables, which serve for description of the history effect.

The state of a continuum system is often defined by two distinct fields, e.g. the temperature and displacement fields. Two sets of governing equations define the solution for both types of variables. When neither of these variables can be eliminated by using one set of

[4] The Euler backward integration scheme is considered here, but other schemes such as variable midpoint algorithms can also be incorporated.

equations, both sets must be solved simultaneously and the system is said to be coupled. The approximate solution is obtained by solving two sets of residual equations in each time step:

$$ {}^{(n)}\mathbf{R}\left({}^{(n)}\mathbf{u},{}^{(n-1)}\mathbf{u},{}^{(n)}\mathbf{v},{}^{(n-1)}\mathbf{v}\right)=\mathbf{0} \tag{8}$$

and

$$ {}^{(n)}\mathbf{H}\left({}^{(n)}\mathbf{u},{}^{(n-1)}\mathbf{u},{}^{(n)}\mathbf{v},{}^{(n-1)}\mathbf{v}\right)=\mathbf{0}\,. \tag{9}$$

Different solution schemes include either solution of both systems simultaneously in an iteration system, or solution of the systems separately for one set of variables while keeping the other set fixed; the converged sets of variables are in this case exchanged between the two systems.

4.2 Sensitivity Analysis

For the purpose of optimisation the notion of parametrisation is introduced. We want to change the setup of the considered system either in terms of geometry, constitutive parameters, initial or boundary conditions, or a combination of these. A set of design parameters $\Phi=\left[\phi_1,\phi_2,...,,\phi_n\right]$ is used to describe the properties of the system which can be varied. The equations which govern the system and therefore the numerical solution depend on the design parameters.

To construct an optimisation problem, certain quantities of interest such as the objective and constraint functions must be defined. For many optimisation algorithms the derivatives of these quantities with respect to the design parameters (i.e. sensitivities) are important. Evaluation of these derivatives is the subject of sensitivity analysis, which is introduced here in terms of basic formalism. For this purpose, let us consider a general function that is dependent on the design parameters which completely define the system of interest:

$$F(\Phi)=G\left(\mathbf{u}(\Phi),\Phi\right) \tag{10}$$

F is referred to as the response functional and appears as a term in the objective or constraint functions. *F* will be typically defined through a system response **u**, but it may in addition include explicit dependence on the design parameters, as is indicated by the right hand side of (10). One way of evaluating derivatives $dF/d\phi_i$ is using numerical approximation by the finite difference formula,

$$\frac{dF}{d\phi_k}\left(\phi_1,\phi_2,...,\phi_n\right)\approx\frac{F\left(\phi_1,...,\phi_{k-1},\phi_k+\Delta\phi_k,\phi_{k+1}...,\phi_n\right)-F\left(\phi_1,...,\phi_{k-1},\phi_k,\phi_{k+1}...,\phi_n\right)}{\Delta\phi_k}\,. \tag{11}$$

Evaluation of each derivative requires an additional evaluation of *F* at a perturbed set of design parameters, which includes numerical evaluation of the system response **u** at the perturbed parameters. More effective schemes, which are incorporated in a solution procedure for evaluation of the system response, are described below.

Derivation of (10) with respect to a specific design parameter $\phi = \phi_k$ [5] gives

$$\frac{dF}{d\phi} = \frac{\partial G}{\partial \mathbf{u}} \frac{d\mathbf{u}}{d\phi} + \frac{\partial G}{\partial \phi}. \tag{12}$$

Derivatives $\partial G / \partial \mathbf{u}$ and $\partial G / \partial \phi$ are determined explicitly by definition of the functional *F*. The main task of the sensitivity analysis is therefore evaluation of the term $d\mathbf{u}/d\phi$, which is an implicit quantity because the system response **u** depends on the design parameters implicitly through numerical solution of the governing equations.

Let us first consider *steady state problems* where the approximate system response can be obtained by solution of a single set of non-linear equations (2). Since the system is parametrised, these equations depend on the design parameters and can be restated as

$$\mathbf{R}\left(\mathbf{u}(\Phi), \Phi\right) = \mathbf{0}. \tag{13}$$

This equation defines implicit dependence of the system response on the design parameters and will be used for derivation of formulae for implicit sensitivity terms.

In the *direct differentiation method* the term $d\mathbf{u}/d\phi$ is obtained directly by derivation of (13) with respect to a specific parameter ϕ, which yields

$$\frac{\partial \mathbf{R}}{\partial \mathbf{u}} \frac{d\mathbf{u}}{d\phi} = -\frac{\partial \mathbf{R}}{\partial \phi}. \tag{14}$$

This set of linear equations must be solved for each design parameter in order to obtain the appropriate implicit term $d\mathbf{u}/d\phi$. This term is then substituted into (12) in order to obtain the derivative of the functional *F* with respect to that parameter. The equation resembles (3), which is solved iteratively to obtain the approximate system response. According to this analogy, (14) is often referred to as a pseudoproblem for evaluation of the implicit sensitivity terms, and the right-hand side $-\partial \mathbf{R}/\partial \phi$ is referred to as the pseudoload. As opposed to (3), (14) is solved only once at the end of the iterative scheme, because the tangent operator $\partial \mathbf{R}/\partial \mathbf{u}$ evaluated for the converged solution **u** (where equations (13) are satisfied) must be taken into account for evaluation of sensitivities. If the system of equations (3) is solved by decomposition of the stiffness matrix, then the decomposed tangent stiffness matrix from the last iteration can be used for solution of (14), which means that the additional computational cost includes only back substitution. Evaluation of derivatives with respect to each design parameter therefore contributes only a small portion of computational cost required for solution of (13) as opposed to the finite difference scheme, where evaluation of the derivative with respect to each parameter requires a complete solution of (13) for the corresponding perturbed design.

An additional task in the case of analytical differentiation is evaluation of the load vector $-\partial \mathbf{R}/\partial \phi$. It requires explicit derivation of the finite element formulation (more

[5] Index *k* is suppressed in order to simplify the derived expressions.

precisely the formulae for evaluation of element contributions to the stiffness matrix) with respect to design parameters, which must be incorporated in the numerical simulation.

An alternative method for evaluation of sensitivities is the *adjoint method*. In this method the implicit term $d\mathbf{u}/d\phi$ is eliminated from (12). An augmented functional

$$\tilde{F}(\Phi) = G\big(\mathbf{u}(\Phi),\Phi\big) - \lambda^T(\Phi)\mathbf{R}\big(\mathbf{u}(\Phi),\Phi\big) \tag{15}$$

is defined, where λ is the vector[6] of Lagrange multipliers, which will be used for elimination of implicit sensitivity terms. In the converged solution, $\widetilde{F} = F$ because $\mathbf{R} = 0$. Differentiation of (15) with respect to a specific design parameter ϕ yields

$$\frac{d\widetilde{F}}{d\phi} = \frac{\partial G}{\partial \mathbf{u}}\frac{d\mathbf{u}}{d\phi} + \frac{\partial G}{\partial \phi} - \left(\frac{d\lambda}{d\phi}\right)^T \mathbf{R} - \lambda^T\left(\frac{\partial \mathbf{R}}{\partial \mathbf{u}}\frac{d\mathbf{u}}{d\phi} + \frac{\partial \mathbf{R}}{d\phi}\right). \tag{16}$$

Since $\mathbf{R} = 0$ by (13) and $\dfrac{\partial \mathbf{R}}{\partial \mathbf{u}}\dfrac{d\mathbf{u}}{d\phi} + \dfrac{\partial \mathbf{R}}{d\phi} = 0$ by (14), we have

$$\frac{dF}{d\phi} = \frac{d\widetilde{F}}{d\phi}. \tag{17}$$

The terms in (15) which include implicit derivatives are

$$\frac{\partial G}{\partial \mathbf{u}}\frac{d\mathbf{u}}{d\phi} - \lambda^T\frac{\partial \mathbf{R}}{\partial \mathbf{u}}\frac{d\mathbf{u}}{d\phi} = \left(\frac{\partial G}{\partial \mathbf{u}} - \lambda^T\frac{\partial \mathbf{R}}{\partial \mathbf{u}}\right)\frac{d\mathbf{u}}{d\phi} \tag{18}$$

These terms are eliminated from (16) by defining λ so that the term in round brackets in (18) is zero. This is achieved if λ solves the system

$$\left(\frac{\partial \mathbf{R}}{\partial \mathbf{u}}\right)^T \lambda = \left(\frac{\partial G}{\partial \mathbf{u}}\right)^T. \tag{19}$$

System (19) is referred to as the adjoint problem for the adjoint response λ with the adjoint load $\left(\partial G/\partial \mathbf{u}\right)^T$. Once multipliers λ are evaluated, the derivative of *F* with respect to a specific parameter ϕ is obtained as

$$\frac{dF}{d\phi} = \frac{d\widetilde{F}}{d\phi} = \frac{\partial G}{\partial \phi} - \lambda^T\frac{\partial \mathbf{R}}{\partial \phi}. \tag{20}$$

The adjoint method requires the solution of the adjoint problem (19) for each response functional *F*. It is efficient when the number of response functionals is small compared to the number of design parameters.

[6] Vectors denoted by Greek letters are not typed in bold, but it should be clear from the context when some quantity is a vector and when scalar.

A similar approach can be adopted for *transient problems* where sensitivities are evaluated within the incremental solution scheme. As for steady state problems the dependence on the design parameter is taken into account in the discretised governing equations (5):

$$ {}^{(n)}\mathbf{R}\left({}^{(n)}\mathbf{u}(\phi),{}^{(n-1)}\mathbf{u}(\phi),\phi\right)=\mathbf{0}\,. \tag{21}$$

It will be assumed that the response functional is defined through the response at the final time ${}^{(M)}t$:

$$ F(\Phi)=G\left({}^{(M)}\mathbf{u}(\Phi),\Phi\right). \tag{22}$$

Derivation with respect to the parameter ϕ yields

$$ \frac{dF}{d\phi}=\frac{dG}{d^{(M)}\mathbf{u}}\frac{D^{(M)}\mathbf{u}}{d\phi}+\frac{\partial G}{\partial \phi}\,. \tag{23}$$

In the *direct differentiation method* the implicit derivative is obtained directly by derivation of (21), which yields (after setting the increment index to *M*)

$$ \frac{\partial^{(M)}\mathbf{R}}{\partial^{(M)}\mathbf{u}}\frac{d^{(M)}\mathbf{u}}{d\phi}=-\left(\frac{\partial^{(M)}\mathbf{R}}{\partial^{(M-1)}\mathbf{u}}\frac{d^{(M-1)}\mathbf{u}}{d\phi}+\frac{\partial^{(M)}\mathbf{R}}{\partial\phi_i}\right) \tag{24}$$

The pseudoload on the above equation contains the sensitivity of the response evaluated in the previous step. By applying the direct differentiation procedure back in time we see that the system

$$ \frac{\partial^{(n)}\mathbf{R}}{\partial^{(n)}\mathbf{u}}\frac{d^{(n)}\mathbf{u}}{d\phi}=-\left(\frac{\partial^{(n)}\mathbf{R}}{\partial^{(n-1)}\mathbf{u}}\frac{d^{(n-1)}\mathbf{u}}{d\phi}+\frac{\partial^{(n)}\mathbf{R}}{\partial\phi_i}\right) \tag{25}$$

must be solved for $d^{(i)}\mathbf{u}/d\phi$ after each time step (i.e. for *i*=1, 2, …, *M*) after convergence of the iteration (6) and (7), while the derivative of the initial condition $d^{(0)}\mathbf{u}/d\phi$ needed after the first increment is assumed to be known.

In the *adjoint method* the implicit terms are again eliminated by the appropriate definition of the Lagrange multipliers. The augmented functional is defined by combination of (22) and (21) for all increments:

$$ F(\Phi)=G\left({}^{(M)}\mathbf{u}(\Phi),\Phi\right)-\sum_{n=1}^{M}{}^{(n)}\lambda(\Phi)^{T(n)}\mathbf{R}\left({}^{(n)}\mathbf{u}(\Phi),{}^{(n-1)}\mathbf{u}(\Phi),\Phi\right) \tag{26}$$

Again $F = \tilde{F}$ follows from (21) and $\frac{dF}{d\phi} = \frac{d\tilde{F}}{d\phi}$ follows from (21) and (25). Derivation of (26) yields after rearrangement and some manipulation

$$\begin{aligned}
&\frac{dF}{d\phi} = \frac{d\tilde{F}}{d\phi} = \frac{\partial G}{\partial \phi} - \sum_{n=1}^{M} {}^{(n)}\lambda^T \frac{\partial^{(n)}\mathbf{R}}{\partial \phi} - {}^{(1)}\lambda^T \frac{d^{(1)}\mathbf{R}}{d^{(0)}\mathbf{u}} \frac{d^{(0)}\mathbf{u}}{d\phi} - \\
&- \sum_{n=1}^{M-1} {}^{(n)}\lambda^T \frac{\partial^{(n)}\mathbf{R}}{\partial^{(n)}\mathbf{u}} \frac{d^{(n)}\mathbf{u}}{d\phi} + {}^{(n+1)}\lambda^T \frac{\partial^{(n+1)}\mathbf{R}}{\partial^{(n)}\mathbf{u}} \frac{d^{(n)}\mathbf{u}}{d\phi} - \qquad , \\
&- {}^{(M)}\lambda^T \frac{\partial^{(M)}\mathbf{R}}{\partial^{(M)}\mathbf{u}} \frac{d^{(M)}\mathbf{u}}{d\phi} + \left(\frac{\partial G}{\partial^{(M)}\mathbf{u}} \right)^T \frac{d^{(M)}\mathbf{u}}{d\phi}
\end{aligned} \tag{27}$$

where the first line contains explicit terms and the other two lines contain implicit terms which must be eliminated.

Elimination of implicit terms from (27) is achieved by solution of the following set of adjoint problems for the Lagrange multiplier vectors:

$$\begin{aligned}
&\left(\frac{\partial^{(M)}\mathbf{R}}{\partial^{(M)}\mathbf{u}} \right)^T {}^{(M)}\lambda = \frac{\partial G}{\partial^{(M)}\mathbf{u}}, \\
&\left(\frac{\partial^{(n)}\mathbf{R}}{\partial^{(n)}\mathbf{u}} \right)^T {}^{(n)}\lambda = -\left(\frac{\partial^{(n+1)}\mathbf{R}}{\partial^{(n)}\mathbf{u}} \right)^T {}^{(n+1)}\lambda,\ n = M-1, M-2, \ldots, M-1
\end{aligned} \quad . \tag{28}$$

Once this is done, the functional derivative is obtained from

$$\frac{dF}{d\phi} = \frac{d\tilde{F}}{d\phi} = \frac{\partial G}{\partial \phi} - \sum_{n=1}^{M} {}^{(n)}\lambda^T \frac{\partial^{(n)}\mathbf{R}}{\partial \phi} - {}^{(1)}\lambda^T \frac{d^{(1)}\mathbf{R}}{d^{(0)}\mathbf{u}} \frac{d^{(0)}\mathbf{u}}{d\phi} \tag{29}$$

Since the equations (28) are evaluated in the reverse order as the tangent operators, the complete problem must be solved before the sensitivity analysis can begin. This requires storage of converged (and possibly decomposed) tangent operators from all increments. The adjoint analysis may still be preferred when the number of the design parameters is significantly larger than the number of response functionals.

A similar derivation can be performed for coupled systems (i.e. equations (6) and (7)). The procedure is outlined e.g. in [2] and [3]. In the direct method sensitivity of one field is expressed in terms of the sensitivity of another, which gives the dependent and the independent pseudoproblem. In the adjoint methods, two sets of Lagrange multipliers must be introduced, one for each corresponding equation. Two adjoint problems are solved for each set of multipliers for each increment, otherwise the procedure is the same as for non-coupled problems.

Sensitivity analysis increases the complexity of the simulation code. One complication comes at the global level where the assembled problem is solved in the incremental/iterative scheme. Solution of the adjoint or pseudoproblems must be included in the scheme, which includes assembling of pseudoloads from element terms. This is followed by appropriate substitutions in order to evaluate the complete sensitivities. An additional complication in the adjoint method is that the converged tangent operators must be stored for increments, since solution of the adjoint problems is reversed in time. In this level the additional complexity can be relatively easily kept under control if the programme structure is sufficiently flexible. The number of necessary updates in the code which is primarily aimed for solution of the direct problem is small and the additional complexity in the programme flow chart is comparable to the complexity of the original flow chart.

A more serious problem is the complexity which arises on the element level, where element terms of the pseudoloads are evaluated, i.e. derivatives of the residual with respect to design parameters. The code should be able to evaluate the pseudoload for any parametrisation that might be used, which can include shape, material, load parameters, etc. Implementation of a general purpose solution code which could provide response sensitivities for any possible set of parameters turns out to be a difficult task. It must be taken into account that such a code must include different material models and finite element formulations and that derivation of the process of evaluation of element residual terms with respect to any of the possible parameters can be itself a tedious task. Another complication which should not be overlooked is the evaluation of the terms $\partial G/\partial \mathbf{u}$. Although these are regarded explicit terms, for complex functionals their evaluation is closely related to the numerical model and can include spatial and time integration and derivation of quantities dependent upon history parameters, with respect to the primary response **u**.

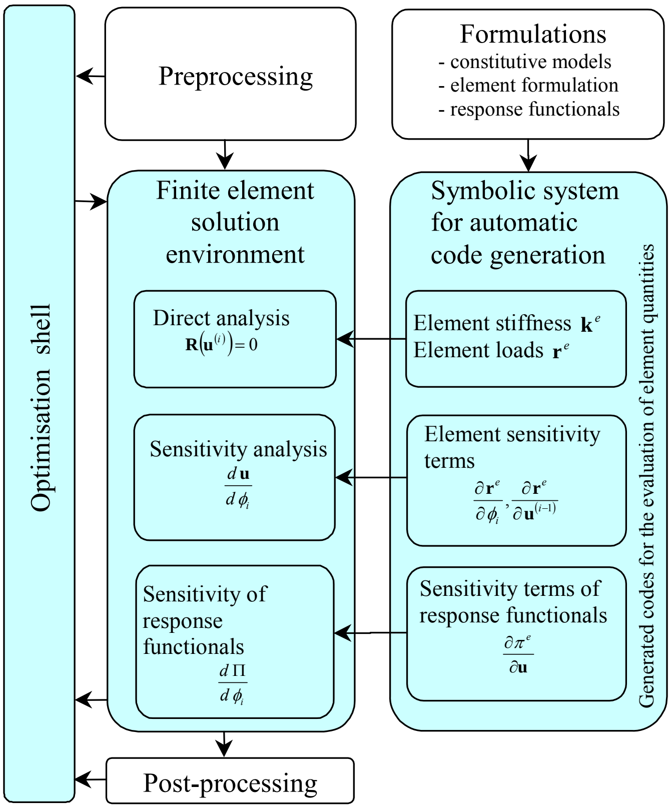

Optimisation shell
Preprocessing
Formulations
- constitutive models
- element formulation
- response functionals
Finite element solution environment
Symbolic system for automatic code generation
Direct analysis
$\mathbf{R}(\mathbf{u}^{(i)})=0$
Element stiffness $\mathbf{k}^e$
Element loads $\mathbf{r}^e$
Sensitivity analysis
$\frac{d\mathbf{u}}{d\phi_i}$
Element sensitivity terms
$\frac{\partial \mathbf{r}^e}{\partial \phi_i}, \frac{\partial \mathbf{r}^e}{\partial \mathbf{u}^{(i-1)}}$
Sensitivity of response functionals
$\frac{d\Pi}{d\phi_i}$
Sensitivity terms of response functionals
$\frac{\partial \pi^e}{\partial \mathbf{u}}$
Generated codes for the evaluation of element quantities
Post-processing

Figure 1: Outline of an optimization system utilizing symbolic code generation.

The reasons outlined above, use of symbolic systems for automatic generation of element level code is very beneficial. Use of such systems enables implementation of new finite element formulations and physical models in times drastically shorter than would be needed for manual development. Functionals which are used in optimisation and the necessary sensitivity terms can be defined on abstract mathematical level where the basic formulation of the numerical model is defined. These definitions can be readily adjusted to new types of problems, because the necessary derivations are performed by the symbolic systems and the appropriate computer code is generated automatically. The system for automatic code generation is connected with a flexible solution environment framework (referred to as the finite element driver[11], [12]) into which the generated code can be readily incorporated. The complexity which would arise in a static simulation code applicable for sensitivity analysis in general problems, is avoided to a large extent.

5 Approximation Based Sensitivity and Optimisation

5.1 Introduction

Numerical computation of parameter derivatives is usually based on polynomial interpolation of function response. The finite difference scheme from equation (11) is an example of such interpolation based sensitivity computation. The major problem with such schemes is that they are instable in the presence of noise, which is an inevitable companion in complex numerical computations.

Another approach is based on approximation of the response function. In this approach, the function is sampled in a sufficient number of points (with respect to the chosen basis functions) around the point of evaluation to calculate the approximation coefficients. Function gradient is then approximated by the gradient of the approximation. Matters of concern are the choice of the sampling region and positions of sampling points in order to maximise the accuracy, choice of basis functions and eventually the number of excessive sampling points that stabilise the approximation in the presence of noise in sampled data.

As mentioned in Section 2, this kind of numerical differentiation will typically be coupled with other analysis tools such as optimisation techniques. In this case the accuracy of numerical derivatives themselves may be of minor concern. The ultimate target in this case is efficiency of the optimisation procedure. Calculation of derivatives of the approximation appears as a tasks that is a part of solution of the approximated optimisation problem with additional step restriction constraints, an inner sub-problem that is solved in iterative scheme. For these reasons, adjustment of sampling strategy or weighting functions and other approximation related issues will be more subject to optimisation than to differentiation.

In this section the approximation methods that will be used in approximation based sensitivity analysis and optimisation are outlined. The concept of weighted least squares approximation is described first, with details exposed for quadratic polynomial approximation. In Section 5.3, the concept of moving least squares (MLS) approximation is introduced. This type of approximation is suitable for use in optimisation techniques due to the ability of fitting the response over large parameter ranges, as opposed to the ordinary least squares whose character is local. A qualitative demonstration of the MLS character is provided in Section 5.4.

5.2 Weighted Least Squares in Function Approximation

We have values of some function $f(\mathbf{x})$ in *m* points:

$$f(\mathbf{x}_i) = y_i\ ,\ \ i = 1,...,m\ . \tag{30}$$

We would like to evaluate coefficients of linear combination of *n* functions $f_1(\mathbf{x})$, …, $f_n(\mathbf{x})$

$$y(\mathbf{x}) = a_1 f_1(\mathbf{x}) + a_2 f_2(\mathbf{x}) + ... + a_n f_n(\mathbf{x}) = \sum_{j=1}^{n} a_j f_j(\mathbf{x}), \tag{31}$$

such that

$$y(\mathbf{x}_i) \approx y_i\ \ \forall i = 1, ..., m\ , \tag{32}$$

i.e. we want that the linear approximation (or approximation) agrees as much as possible with values of $f(\mathbf{x})$ in all points $\mathbf{x}_i$. We look for the best agreement in the weighted least squares sense, i.e. we minimize the function

$$\phi(\mathbf{a}) = \sum_{k=1}^{m} w_k^2 \left(y(\mathbf{x}_k) - y_k\right)^2 = \sum_{k=1}^{m} w_k^2 \left(\left(\sum_{j=1}^{n} a_j f_j(\mathbf{x}_k)\right) - y_k\right)^2 . \tag{33}$$

with respect to parameters of approximation a_i. **w** is the *m*-dimensional vector of weights, which weight significance of points $\mathbf{x}_i$. Minimum is the stationary point of $\phi(\mathbf{a})$ where

$$\frac{d\,\phi(\mathbf{a})}{d\,a_k} = 0\ \ \forall\, k = 1,...,n\ . \tag{34}$$

Derivatives of $\phi(\mathbf{a})$ are

$$\frac{d\,\phi(\mathbf{a})}{d\,a_i} = 2\sum_{k=1}^{m}\left(w_k^2 \left(\sum_{j=1}^{n} a_j f_j(\mathbf{x}_k) - y_k \right) f_i(\mathbf{x}_k)\right) \tag{35}$$

Equation (34) therefore gives the following system of equations for unknown coefficients a_j:

$$\sum_{j=1}^{n} a_j \sum_{k=1}^{m}\left(w_k^2 f_j(\mathbf{x}_k) f_i(\mathbf{x}_k)\right) = \sum_{k=1}^{m}\left(w_k^2 y_k\ f_i(\mathbf{x}_k)\right),\ \ i = 1, ..., n \tag{36}$$

Coefficients **a** can therefore be obtained by solving the linear system of equations

$$\mathbf{Ca} = \mathbf{d}\,, \tag{37}$$

where

$$C_{ij} = \sum_{k=1}^{m} w_k{}^2 f_i(\mathbf{x}_k) f_j(\mathbf{x}_k) \tag{38}$$

and

$$d_i = \sum_{k=1}^{m} w_k{}^2 f_i(\mathbf{x}_k)\, y_k \tag{39}$$

We can write

$$\mathbf{C} = \mathbf{A}^T \mathbf{A} \tag{40}$$

and

$$\mathbf{d} = \mathbf{A}^T \mathbf{b}, \tag{41}$$

where

$$A_{ij} = w_i f_j(\mathbf{x}_i) \tag{42}$$

and

$$\mathbf{b}_i = w_i y_i. \tag{43}$$

From (31) we can see that

$$y(\mathbf{x}_i, \mathbf{a}) = \sum_{k=1}^{n} a_k f_k(\mathbf{x}_i). \tag{44}$$

and therefore

$$\frac{d\, y(\mathbf{x}_i, \mathbf{a})}{d\, a_k} = f_k(\mathbf{x}_i) = \frac{A_{ik}}{w_i} \tag{45}$$

We see that

$$\frac{A_{ik}}{w_i} = \frac{d\, y(\mathbf{x}_i, \mathbf{a})}{d\, a_k} \tag{46}$$

Sometimes we define matrix $\mathbf{X}$ so that

$$X_{ij} = \frac{d\, y(x_j, \mathbf{a})}{d\, a_i} = f_i(\mathbf{x}_j) = \sigma_j A_{ji} \,. \tag{47}$$

5.2.1 Quadratic Approximation

The quadratic approximation has the following general form:

$$q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \mathbf{G}\mathbf{x} + \mathbf{b}^T \mathbf{x} + c \,, \tag{48}$$

where **G** is a symmetric constant symmetric matrix (G_{ij}=G_{ji}), $\mathbf{b}^T$ a constant vector and *c* a constant scalar. The gradient of this function is

$$\nabla q(\mathbf{x}) = \mathbf{G}\mathbf{x} + \mathbf{b} \tag{49}$$

and its Hessian matrix is

$$\nabla^2 q(\mathbf{x}) = \mathbf{G} \,, \tag{50}$$

We must calculate 1 coefficient of the constant term (*c*), *N* coefficients of linear terms and $N \cdot (N+1)/2$ coefficients of quadratic terms. We will map the coefficients and functions in the following way:

$$\begin{aligned}
&a_1 = c \\
&a_{k+1} = b_k \,, k = 1,2,\ldots,N \\
&a_{k+N+1} = G_{1,k} \,, k = 1,2,\ldots,N \\
&a_{k+2N} = G_{2,k} \,, k = 2,3,\ldots,N \\
&a_{k+3N-2} = G_{3,k} \,, k = 3,4,\ldots,N \\
&a_{k+4N-5} = G_{4,k} \,, k = 4,5,\ldots,N \\
&\ldots \\
&a_{(N+1)\cdot(N+2)/2} = G_{N,N}
\end{aligned} \tag{51}$$

and

$$
\begin{aligned}
&f_1(\mathbf{x}) = 1 \\
&f_{k+1}(\mathbf{x}) = x_k \ , k = 1,2,...,N \\
&f_{N+2}(\mathbf{x}) = \frac{1}{2}{x_1}^2 \ , \quad f_{k+N+1}(\mathbf{x}) = x_1\,x_k \quad , k = 2,3,...,N \\
&f_{2N+2}(\mathbf{x}) = \frac{1}{2}{x_2}^2, \quad f_{k+2N}(\mathbf{x}) = x_2\,x_k, \ k = 3,4,...,N \\
&f_{3N+1}(\mathbf{x}) = \frac{1}{2}{x_3}^2, \quad f_{k+3N-2}(\mathbf{x}) = x_3\,x_k, \ k = 4,5,...,N \\
&... \\
&f_{(N+1)(N+2)/2}(\mathbf{x}) = \frac{1}{2}{x_n}^2
\end{aligned}
\tag{52}
$$

We obtain the coefficients a_1, a_2, … by constituting and solving the system of equations (37) according to (38) and (39). If the dimension of the space is N then we have $(N+1)(N+2)/2$ equations to solve.

Example: basis functions in two dimensions

We have

$$
q(x,y) = \tfrac{1}{2}G_{11}x^2 + \tfrac{1}{2}G_{22}y^2 + G_{12}x\,y + b_1x + b_2y + c\,, \tag{53}
$$

therefore

$$
\begin{aligned}
&a_1 = c, \quad f_1(x,y) = 1 \\
&a_2 = b_1, \ f_2(x,y) = x \\
&a_3 = b_2, \ f_3(x,y) = y \\
&a_4 = G_{11}, f_4(x,y) = \tfrac{1}{2}x^2 \\
&a_5 = G_{12}, f_5(x,y) = x\,y \\
&a_6 = G_{22}, f_5(x,y) = \tfrac{1}{2}y^2
\end{aligned}
\tag{54}
$$

5.3 Moving Least Squares (MLS)

Let us have a set of m points

$$
\mathbf{x}_i \in \Omega, \ i = 1, 2, ..., m \tag{55}
$$

$\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, …, $\mathbf{x}_m$ and values of the function $f(\mathbf{x})$ in these points

$$
y_i = f(\mathbf{x}_i), \ i = 1,2,...,m\,. \tag{56}
$$

The interval of the points is large so that we can not approximate well all the data with a single linear approximation of the form (31). We want to construct a smooth approximation that will closely approximate the data in a wide domain. This can be achieved by approximate $f(\mathbf{x})$ on the domain Ω by an approximation of the form

$$y(\mathbf{x}) = a_1(\mathbf{x})f_1(\mathbf{x}) + a_2(\mathbf{x})f_2(\mathbf{x}) + \ldots + a_n(\mathbf{x})f_n(\mathbf{x}) = \sum_{i=1}^{n} a_i(\mathbf{x})f_i(\mathbf{x}). \qquad (57)$$

The approximation is similar to (31), except that the coefficients depend on **x**. We want to find such $a_1(\mathbf{x})$, $a_2(\mathbf{x})$, …, $a_n(\mathbf{x})$ $y(\mathbf{x})$ is smooth and it well approximates the given data. We construct the approximation as follows.

For a certain point $\mathbf{x}_0 \in \Omega$ we define the function $y_l(\mathbf{x}_0, \mathbf{x}) = \sum_{i=1}^{n} a_i(\mathbf{x}_0) f_i(\mathbf{x})$. We require that the coefficient functions $a_i(\mathbf{x})$ are such that for each $\mathbf{x}_0$, $y_l(\mathbf{x}_0, \mathbf{x})$ will closely approximate the values in those points that are close to $\mathbf{x}_0$. We will therefore calculate the coefficients $a_i(\mathbf{x}_0)$ by the weighted least square approximation of the data, with large weights for points close to $\mathbf{x}_0$ and small weights for points that are more distant from $\mathbf{x}_0$. According to (37), (38) and (39), $a_i(\mathbf{x}_0)$ will be calculated according to the formula

$$\mathbf{C}\mathbf{a}(\mathbf{x}_0) = \mathbf{d},$$

where

$$C_{ij} = \sum_{k=1}^{m} {w_{0k}}^2 f_i(\mathbf{x}_k) f_j(\mathbf{x}_k)$$

and

$$d_i = \sum_{k=1}^{m} {w_{0k}}^2 f_i(\mathbf{x}_k)\, y_k$$

We can compute coefficients $a_i(\mathbf{x})$ in a similar manner for any point $\mathbf{x} \in \Omega$. If we want the approximation generated in this way to be smooth in **x**, then weights corresponding to sampling points must also smoothly depend on **x**. We achieve this by making the weights smooth functions of **x**, and we ensure a good approximation of sampling values by making the weights fall when the distance of **x** from the corresponding sampling points grow.

Therefore, in each point $\mathbf{x} \in \Omega$ we calculate the approximation according to (37), where the coefficient $a_i(\mathbf{x})$ are calculated by the solution of the equation

$$\begin{aligned} &\mathbf{C}\,\mathbf{a}(\mathbf{x}) = \mathbf{d} \\ &\mathbf{a}(\mathbf{x}) = [a_1(\mathbf{x}), a_2(\mathbf{x}), \ldots, a_n(\mathbf{x})] \end{aligned}, \qquad (58)$$

with the following coefficients of the system matrix **C**:

$$C_{ij}(\mathbf{x}) = \sum_{k=1}^{m} w_k(\mathbf{x})^2 f_i(\mathbf{x}_k) f_j(\mathbf{x}_k) \tag{59}$$

and the following components of the right-hand side vector **d**:

$$d_i(\mathbf{x}) = \sum_{k=1}^{m} w_k(\mathbf{x})^2 f_i(\mathbf{x}_k) y_k \; . \tag{60}$$

In some cases we will define all $w_k(\mathbf{x})$ as

$$w_k(\mathbf{x}) = w(\mathbf{x} - \mathbf{x}_k), \tag{61}$$

where for example $w(\mathbf{x}) = e^{-\left(\|\mathbf{x}\|_2^2/d\right)^2}$. By making the weights assigned to sampling points fall quickly enough with the distance from the distance, the influence of the sampling points is localized, i.e. limited to some neighborhood of the points (Figure 2). We can choose weighting functions that decay more slowly at large distances, or design $w(\mathbf{x})$ with compact support such that $w(\mathbf{x}) = 0$ for $\|\mathbf{x}\|_2 > r_0$. In this case, samples that are too distant from the evaluation point do not have any influence on the value of approximation at that point. In general, the 2-norm will be replaced with more general norm $\|\mathbf{x}\|_{\mathbf{A}} = \mathbf{x}^T \mathbf{A}\, \mathbf{x}$ in order accommodate the approximation to eventual drastic anisotropy of function behavior in different directions.

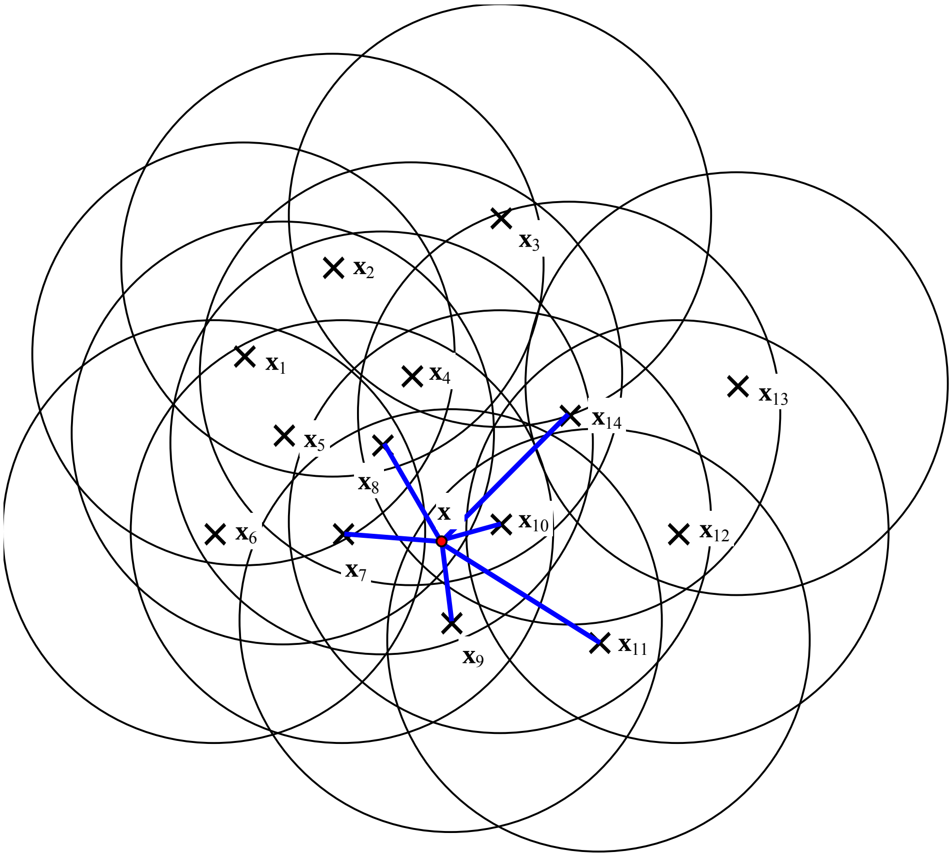


Figure 2: Construction of a diffuse approximation in a given point - sampling points with their influence ranges.

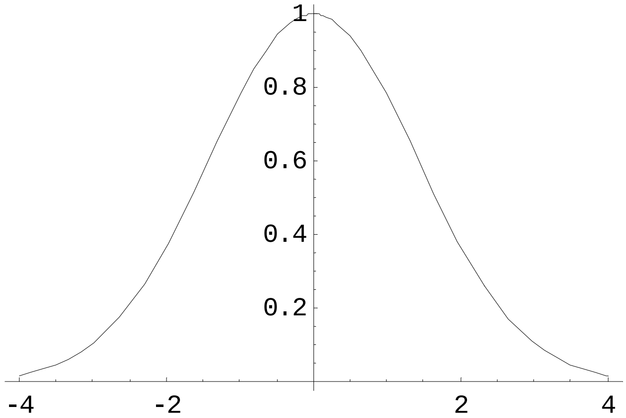


Figure 3: A possible choice for a one-dimensional weighting function *w*.

5.4 Demonstration: MLS approximation of a Noisy Function

5.4.1 Approximation of a Function of One Variable

In this section, approximation of a function of a single variable is demonstrated at different noise levels and sample points. The moving least squares approximation with quadratic polynomial basis functions (i.e. 1, x and x^2) was used for approximation of the function

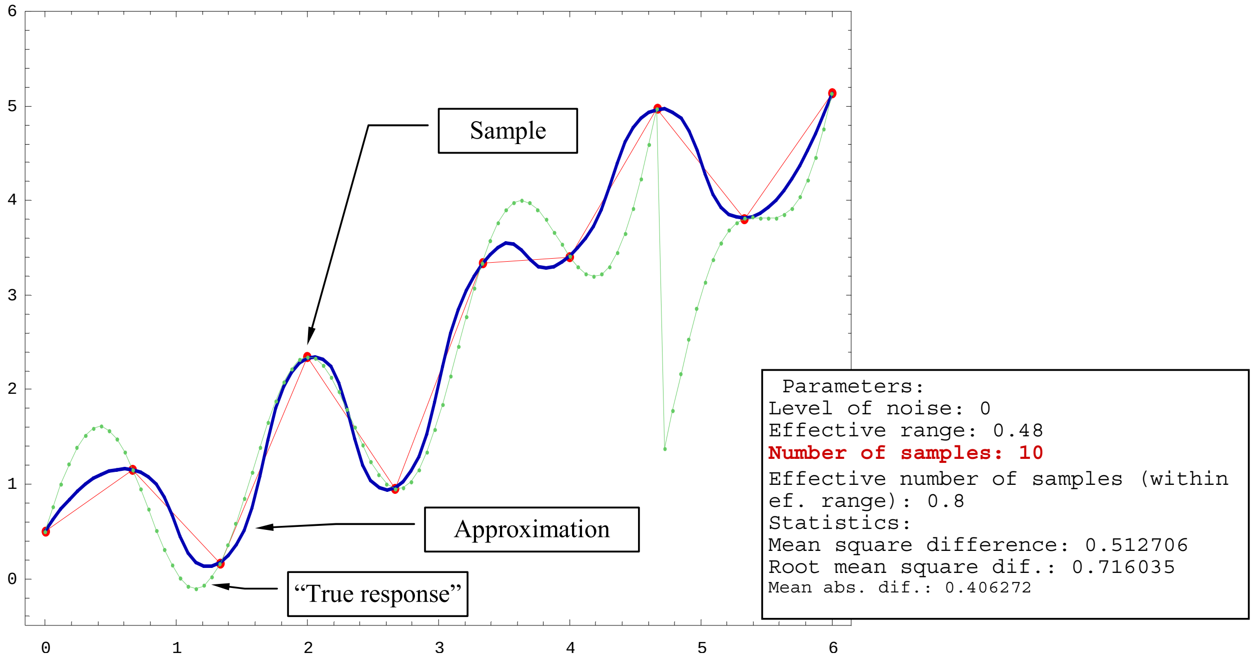
$$f(x) = \sin(4\,x) + 0.5 * e^{x/2} + \begin{cases} 0; & x < x_j \\ \Delta f_j\,; & x \geq x_j \end{cases} \quad . \tag{62}$$

on the interval (0,6), with x_j=4.7 and Δf_j=-4. The function is difficult to approximate with low order polynomials. Characteristics of approximation are shown with regard to numbers of samples on basis of which the approximation is calculated and level of noise that is added to the samples. Sampled values were calculated as

$$f^{(m)}(x) = f(x) + R\,(\mathrm{Rnd}() - 0.5), \tag{63}$$

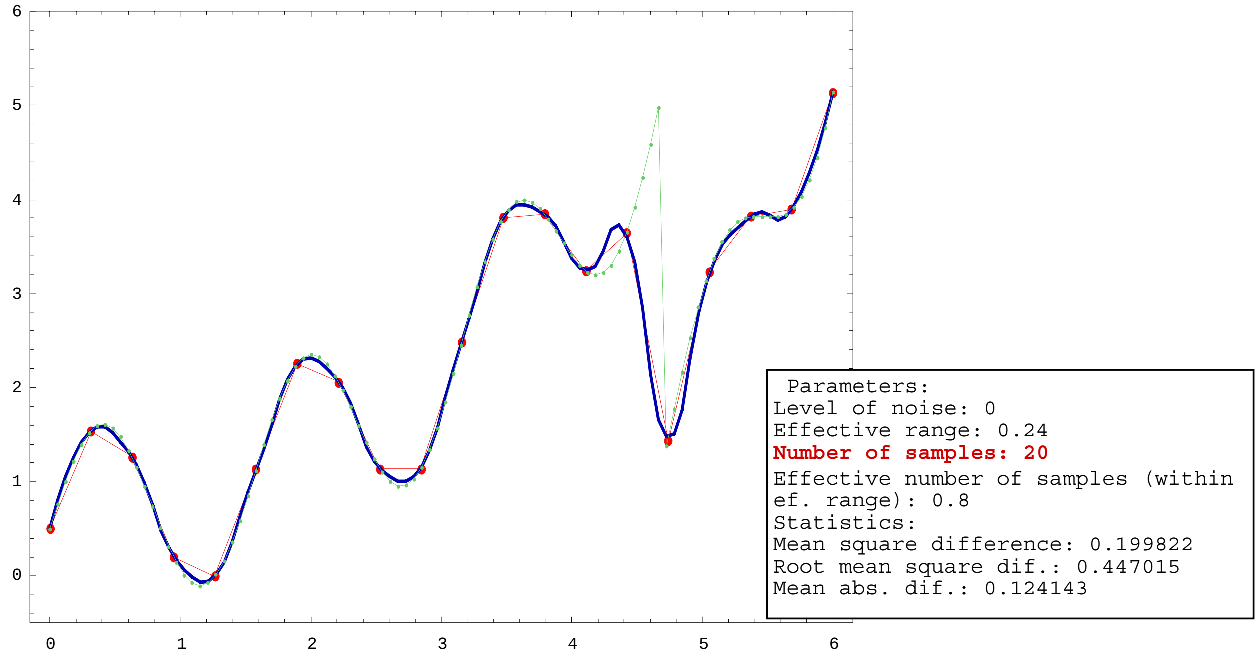
where Rnd() is a function that produces uniform random numbers between 0 and 1 and R is the chosen noise level. According to the discussion in Section 2, the model function defined by (62) is regarded as the "true response" to be approximated, and (62) defines the values of the response we can actually sample (i.e. measure or calculate) and which contain some level of spatially uncorrelated random noise. In addition, a jump discontinuity is added at x_j in order to observe the transition behavior of the approximation at the points where the "true response" itself is discontinuous. The study is illustrated by figures below where the values of approximation parameters are specified in graphs.

5.4.1.1 Approximation without noise (demonstrates accommodation):



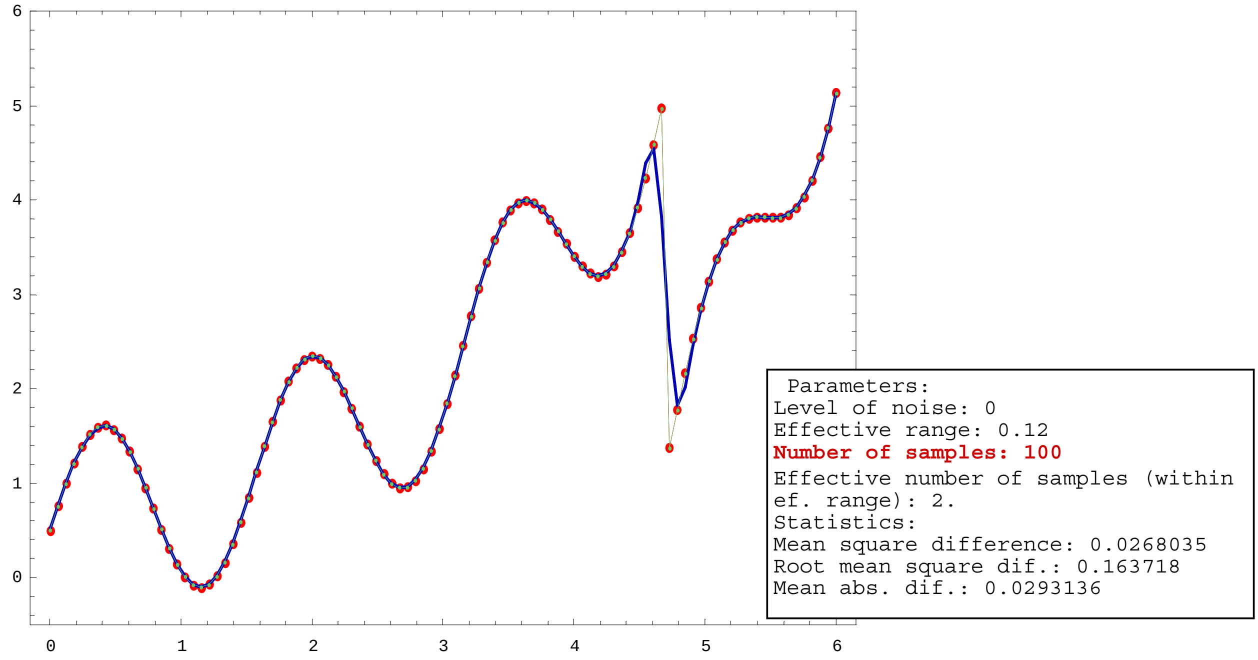
Parameters:
Level of noise: 0
Effective range: 0.48
Number of samples: 10
Effective number of samples (within ef. range): 0.8
Statistics:
Mean square difference: 0.512706
Root mean square dif.: 0.716035
Mean abs. dif.: 0.406272

Figure 4: Very few sampling points with regard to function variation; pure approximation but some basic features are visible.



```
 Parameters:
Level of noise: 0
Effective range: 0.24
Number of samples: 20
Effective number of samples (within
ef. range): 0.8
Statistics:
Mean square difference: 0.199822
Root mean square dif.: 0.447015
Mean abs. dif.: 0.124143
```

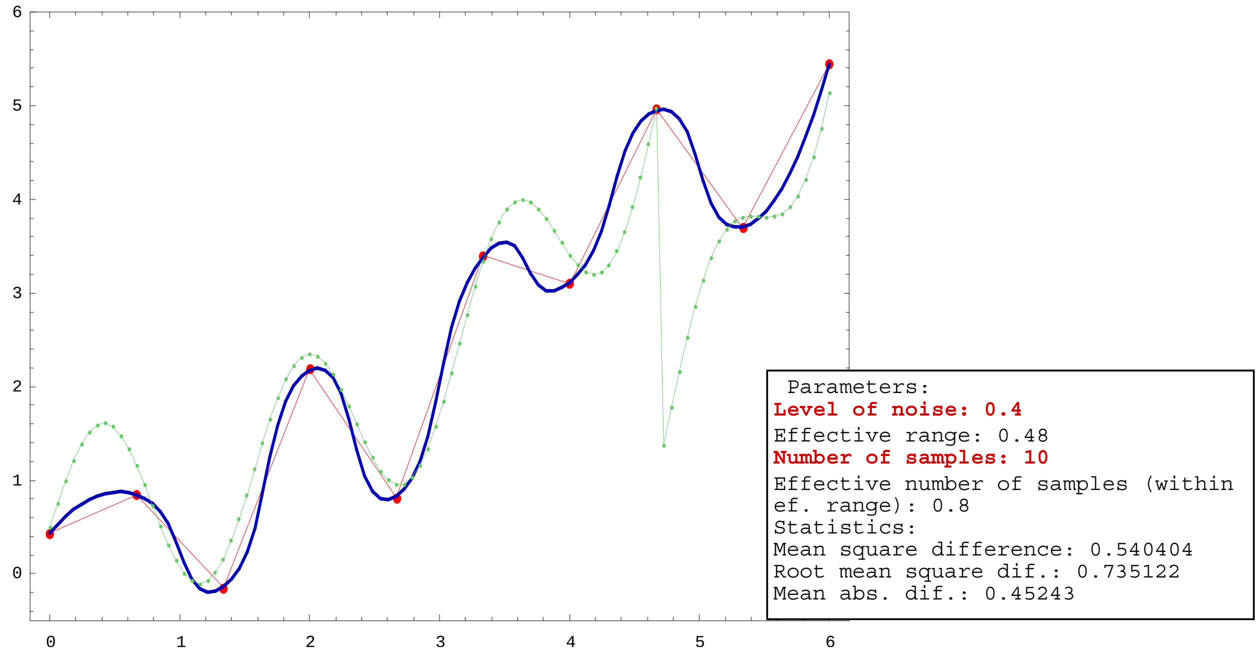
Figure 5: Small number of samples (20), but approximation is pretty good due to quadratic basis, except near the jump.



```
 Parameters:
Level of noise: 0
Effective range: 0.12
Number of samples: 100
Effective number of samples (within
ef. range): 2.
Statistics:
Mean square difference: 0.0268035
Root mean square dif.: 0.163718
Mean abs. dif.: 0.0293136
```

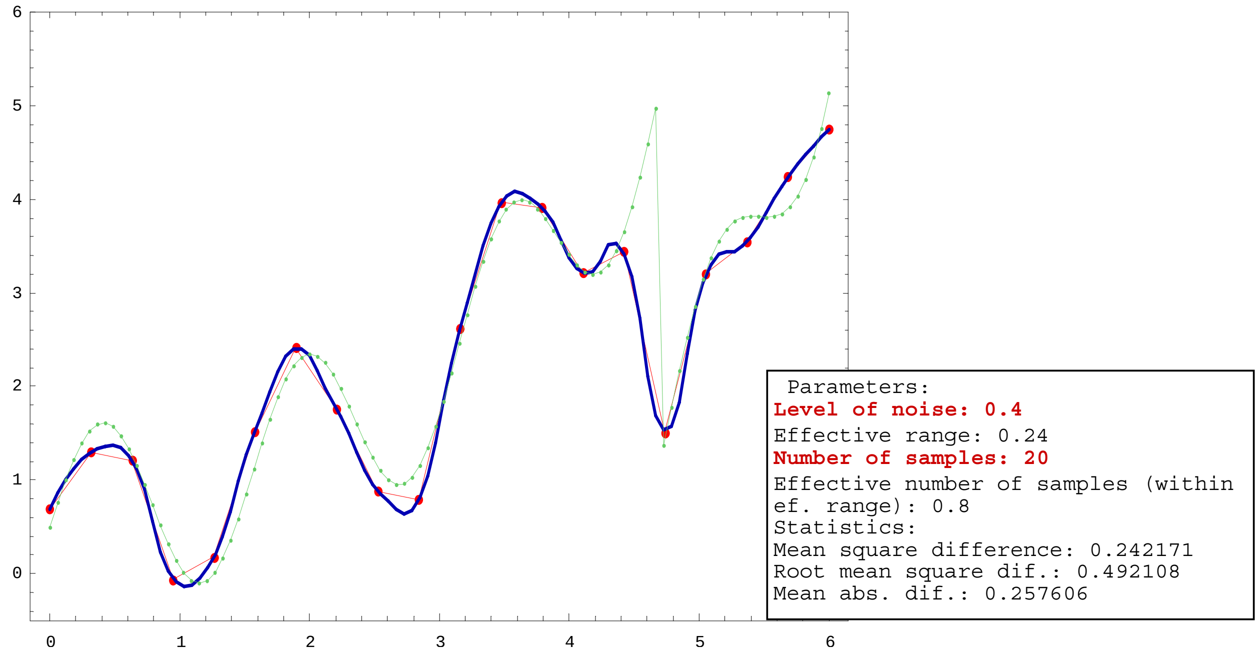
Figure 6: Dense sampling, good approximation over whole interval.

5.4.1.2 Moderate noise



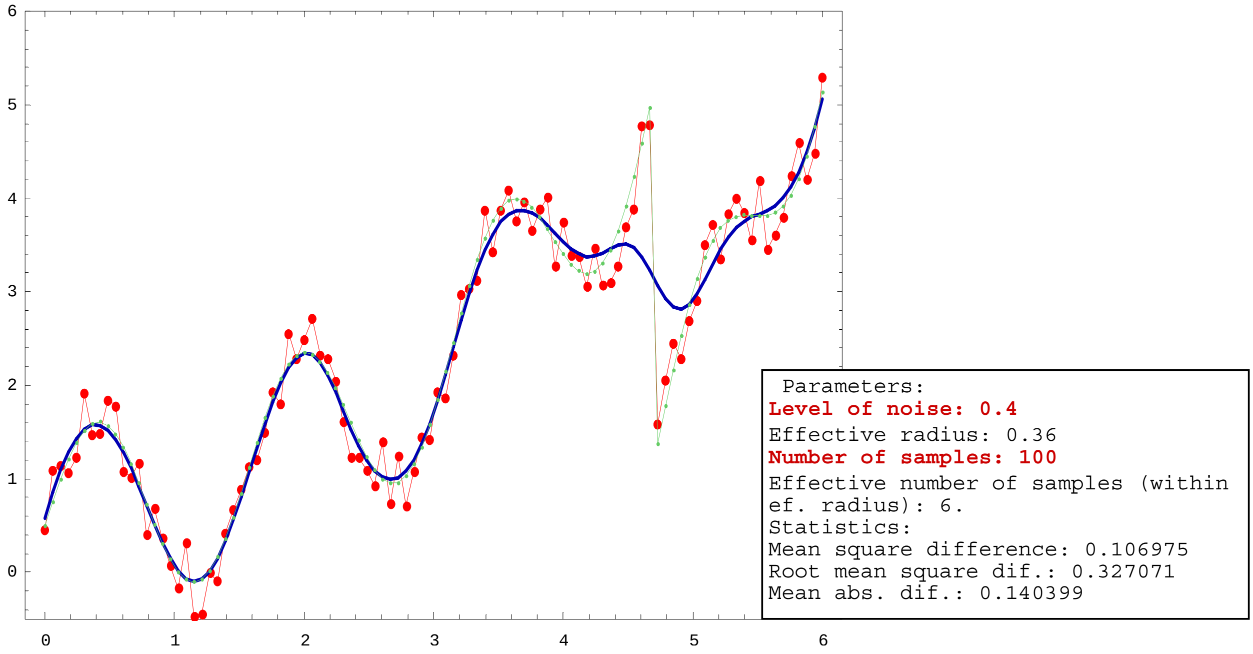
```
 Parameters:
Level of noise: 0.4
Effective range: 0.48
Number of samples: 10
Effective number of samples (within
ef. range): 0.8
Statistics:
Mean square difference: 0.540404
Root mean square dif.: 0.735122
Mean abs. dif.: 0.45243
```

Figure 7: Bad approximation due to scarce sampling, effect of noise is secondary (noise level: 0.4).



```
 Parameters:
Level of noise: 0.4
Effective range: 0.24
Number of samples: 20
Effective number of samples (within
ef. range): 0.8
Statistics:
Mean square difference: 0.242171
Root mean square dif.: 0.492108
Mean abs. dif.: 0.257606
```

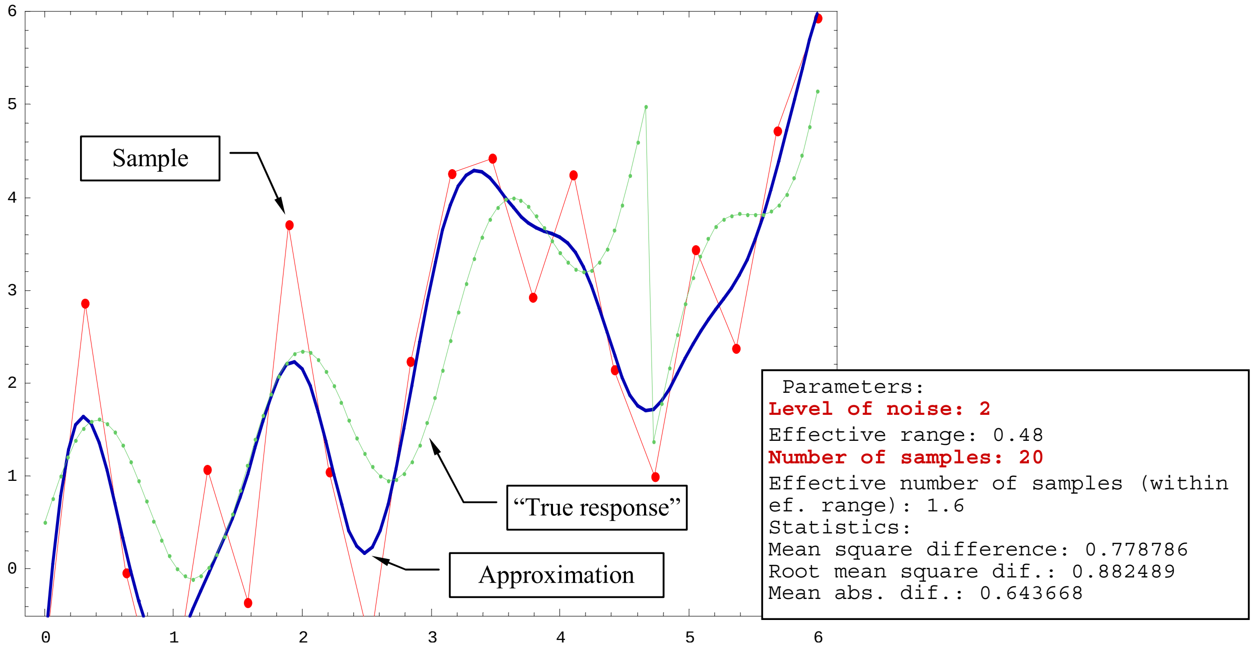
Figure 8: Effect of noise is expressed (cf. Figure 5).



```
 Parameters:
Level of noise: 0.4
Effective radius: 0.36
Number of samples: 100
Effective number of samples (within
ef. radius): 6.
Statistics:
Mean square difference: 0.106975
Root mean square dif.: 0.327071
Mean abs. dif.: 0.140399
```

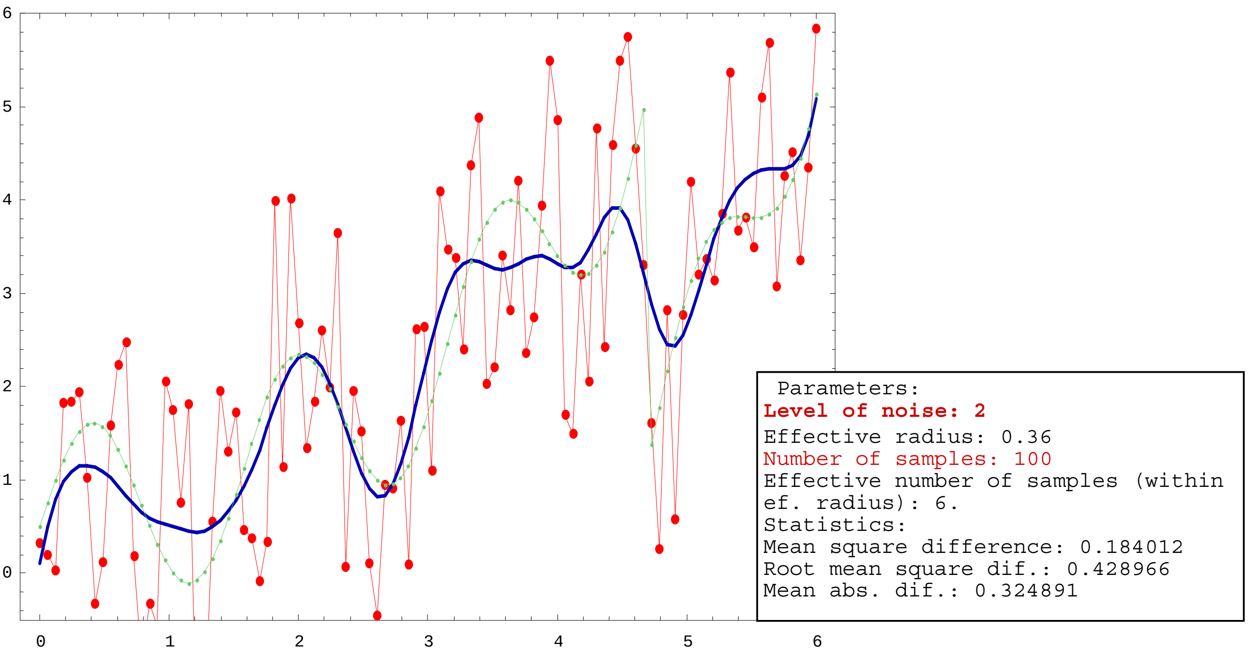
Figure 9: Effect of noise is leveled out a little by taking more samples (cf. Figure 8).

Severe noise



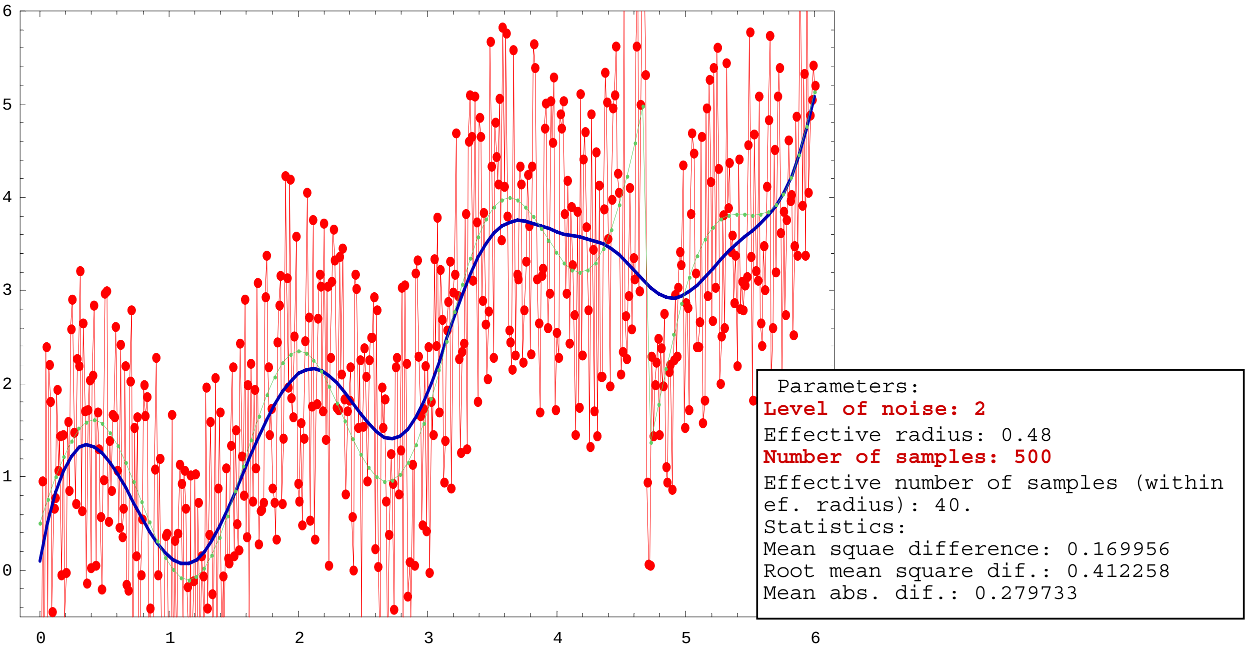
```
 Parameters:
Level of noise: 2
Effective range: 0.48
Number of samples: 20
Effective number of samples (within
ef. range): 1.6
Statistics:
Mean square difference: 0.778786
Root mean square dif.: 0.882489
Mean abs. dif.: 0.643668
```

Figure 10.



```
 Parameters:
Level of noise: 2
Effective radius: 0.36
Number of samples: 100
Effective number of samples (within
ef. radius): 6.
Statistics:
Mean square difference: 0.184012
Root mean square dif.: 0.428966
Mean abs. dif.: 0.324891
```

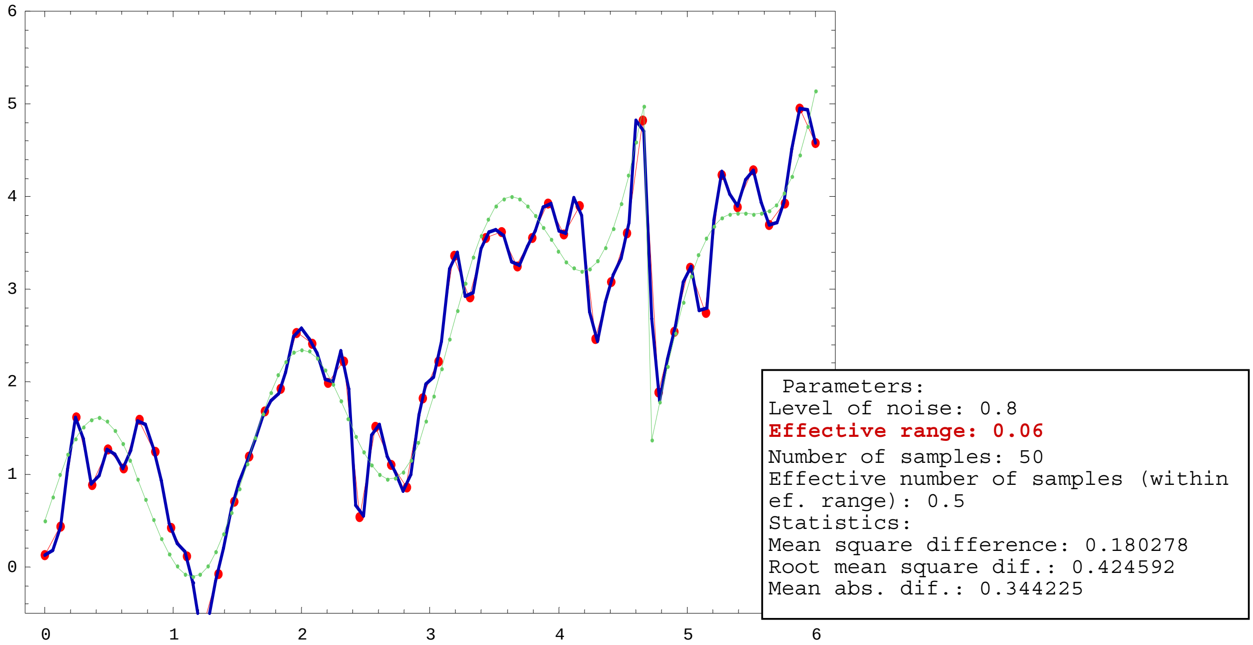
Figure 11.



```
 Parameters:
Level of noise: 2
Effective radius: 0.48
Number of samples: 500
Effective number of samples (within
ef. radius): 40.
Statistics:
Mean squae difference: 0.169956
Root mean square dif.: 0.412258
Mean abs. dif.: 0.279733
```

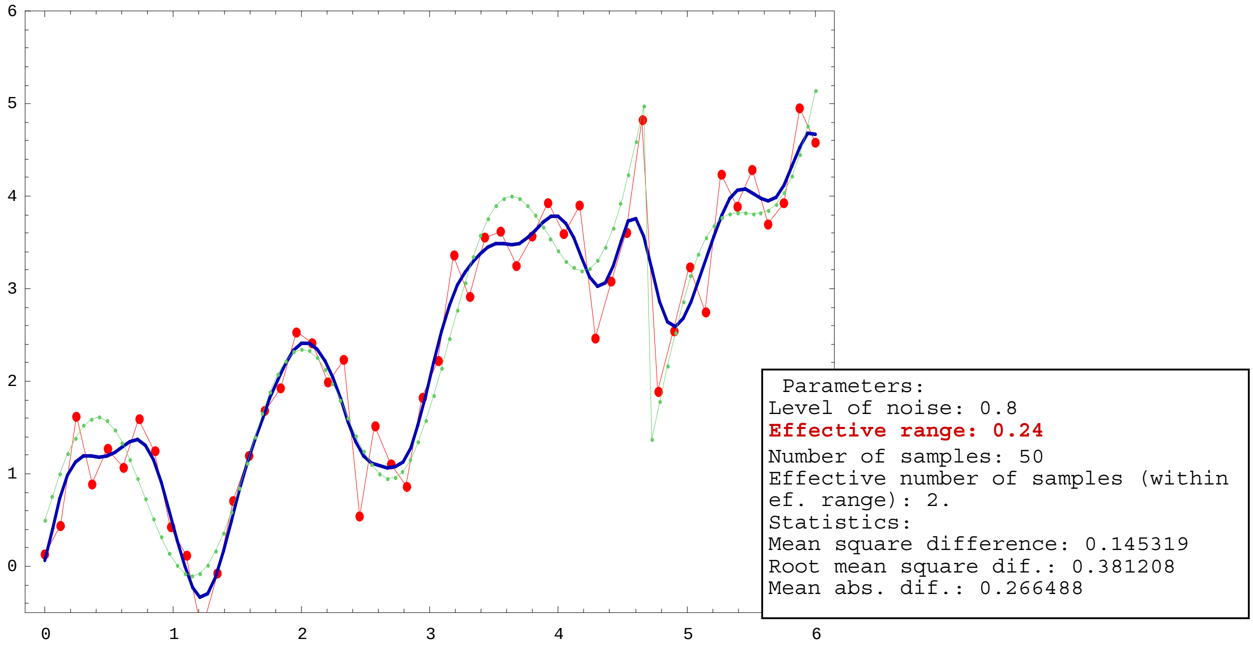
Figure 12: Amplitude of noise is large compared to span of features that can be approximated by basis functions. Improvement with oversampling becomes slow.

Effect of effective range of weighting functions



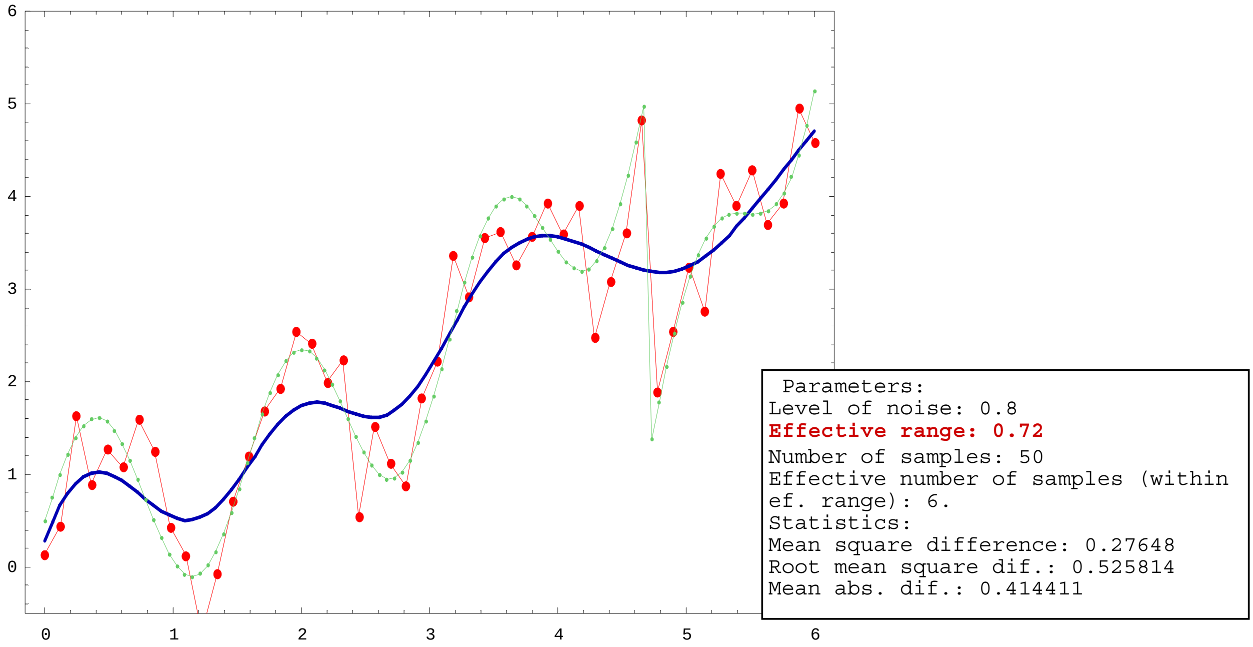
```
 Parameters:
Level of noise: 0.8
Effective range: 0.06
Number of samples: 50
Effective number of samples (within
ef. range): 0.5
Statistics:
Mean square difference: 0.180278
Root mean square dif.: 0.424592
Mean abs. dif.: 0.344225
```

Figure 13: Small range, approximation degenerates to interpolation.



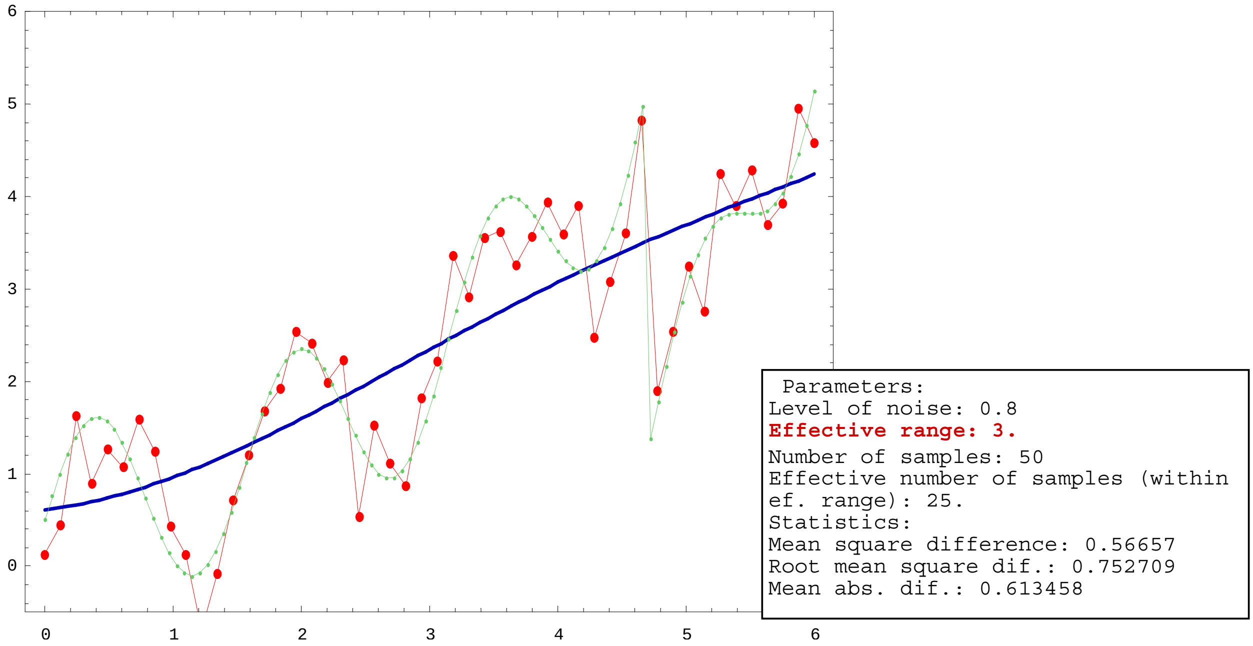
```
 Parameters:
Level of noise: 0.8
Effective range: 0.24
Number of samples: 50
Effective number of samples (within
ef. range): 2.
Statistics:
Mean square difference: 0.145319
Root mean square dif.: 0.381208
Mean abs. dif.: 0.266488
```

Figure 14: Range close to optimal; stabilization with respect to noise effect is good, model function can still be followed.



```
 Parameters:
Level of noise: 0.8
Effective range: 0.72
Number of samples: 50
Effective number of samples (within
ef. range): 6.
Statistics:
Mean square difference: 0.27648
Root mean square dif.: 0.525814
Mean abs. dif.: 0.414411
```

Figure 15: Too large range, high frequency components of the model function are not recapitulated well.



```
 Parameters:
Level of noise: 0.8
Effective range: 3.
Number of samples: 50
Effective number of samples (within
ef. range): 25.
Statistics:
Mean square difference: 0.56657
Root mean square dif.: 0.752709
Mean abs. dif.: 0.613458
```

Figure 16: Effective range too large, details of the model function are pretty smeared. However, this feature can be useful at the initial stage of optimization procedures.

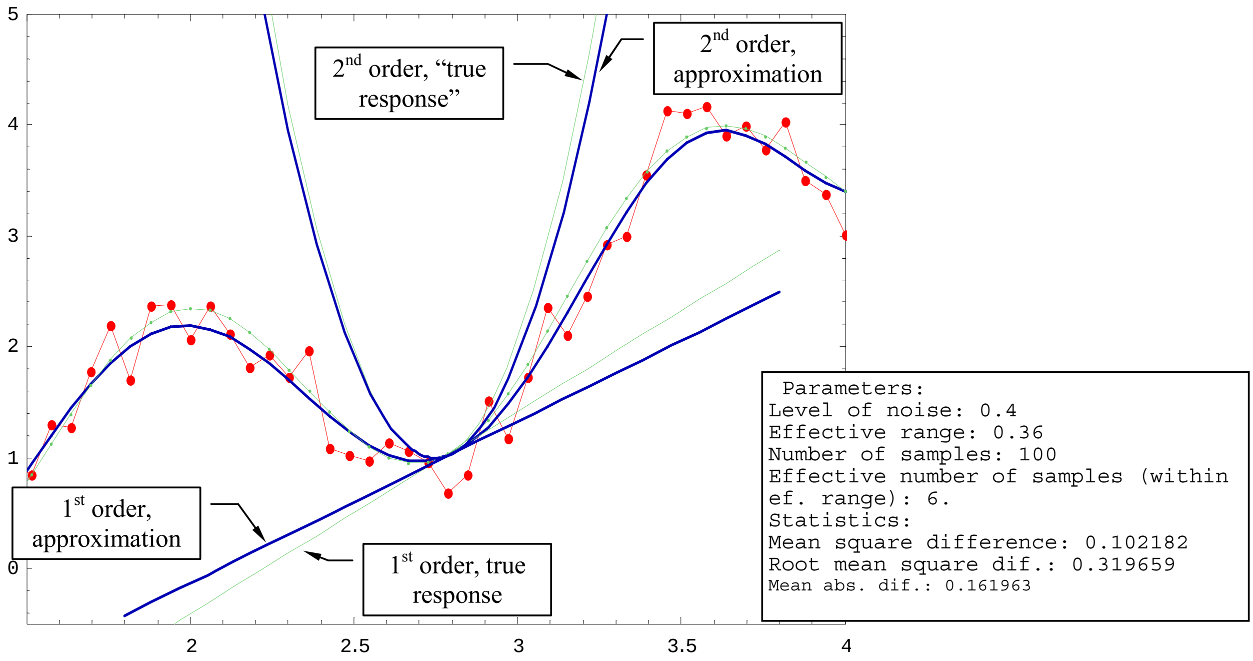


Figure 17: Derivative approximation. First and second order Taylor expansion of the model function and its approximation are shown.

Two Dimensional Demonstration

This example was taken from two-parametric optimisation of a forming operation. The objective and constraint functions sampled on a grid of 15x15 points are shown in Figure 18. The substantial level of noise was preventing application of efficient optimisation techniques. Therefore, response functions were approximated on the basis of sampled values (Figure 19) and an approximated optimisation problem was solved where the original response functions were replaced by the corresponding approximations Figure 20. Beside facilitating solution of the optimisation problem, such approximations are useful for visualisation of the response, which can give additional insight on the impact of the design parameters on system performance.

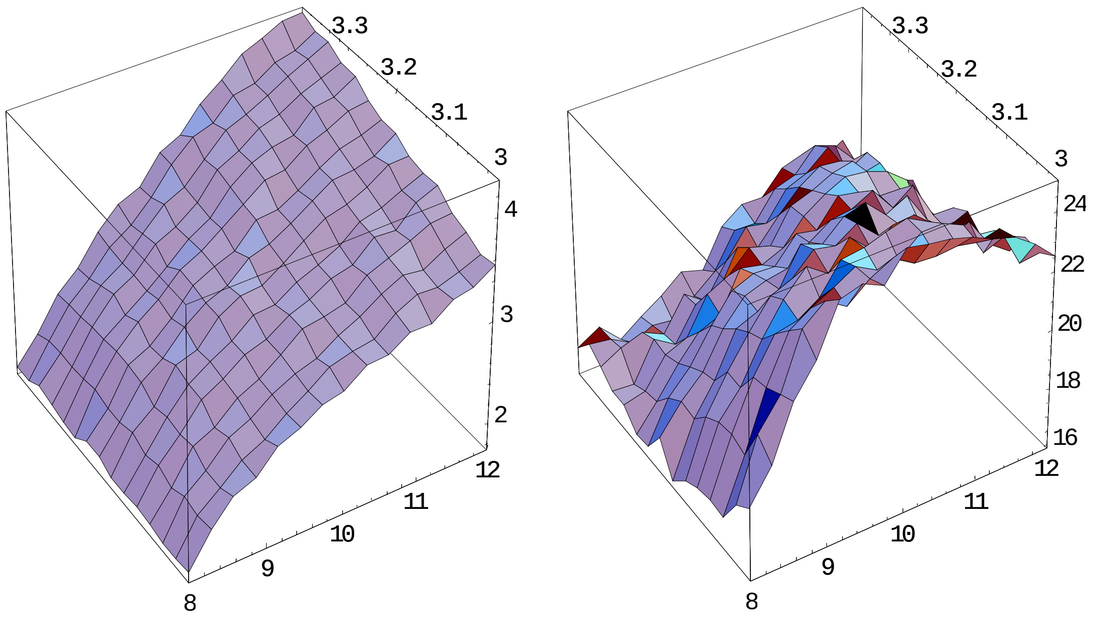


Figure 18: Sampled objective (left) and constraint function (right).

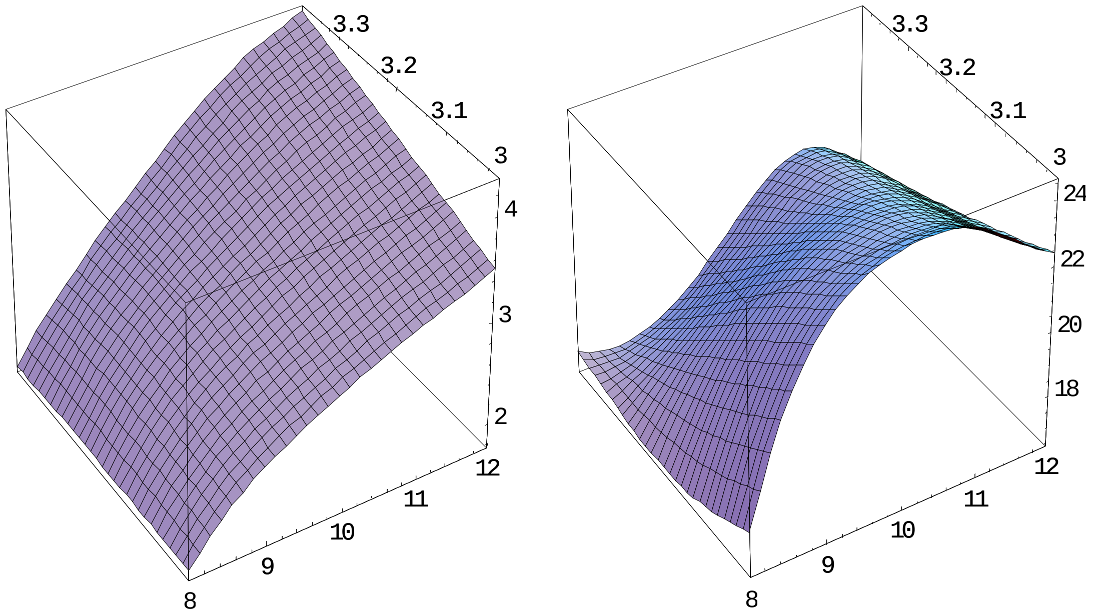


Figure 19: MLS approximation of functions from Figure 18.

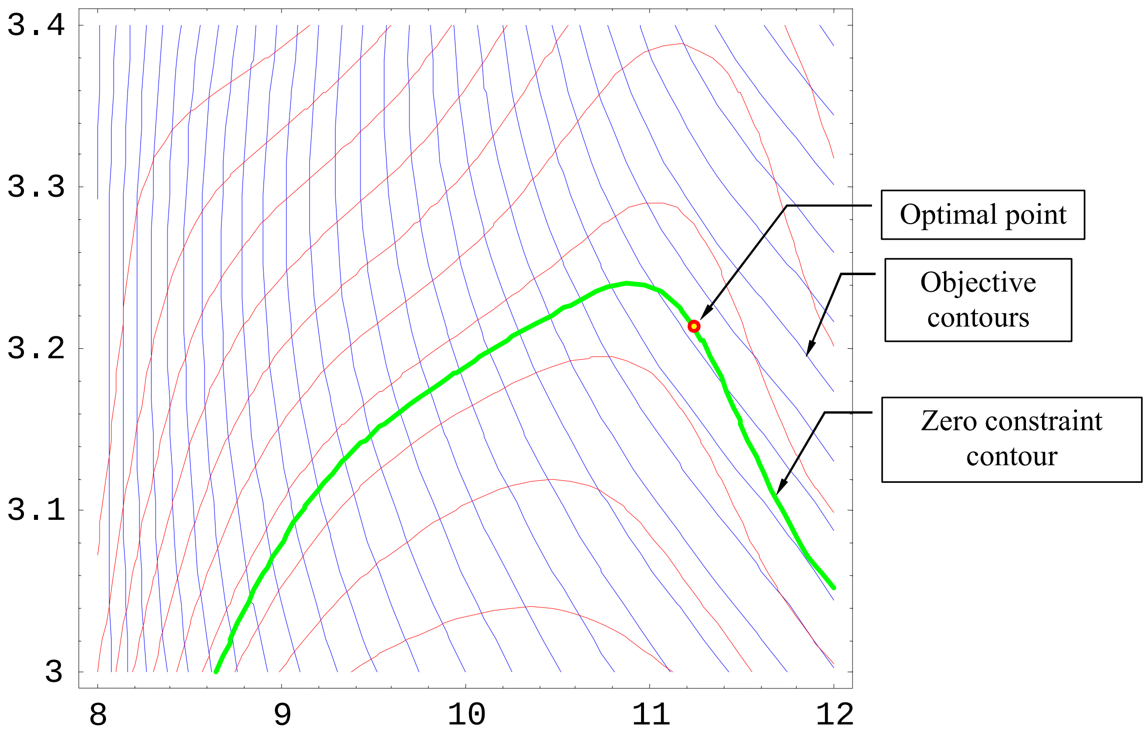


Figure 20: Contours of the MLS approximation of functions from Figure 18, with solution of the approximated problem marked.

6 Software specification

6.1 Software Framework Used by C3M

For solving inverse and optimization problems, C3M is developing a software environment that enables efficient incorporation of new material models, analytical sensitivity analysis and optimization. By this system, finite elements formulations and material laws are defined at symbolic level. The element stiffness, loads and corresponding element sensitivity routines are then generated by the symbolic mechanics system, which also generates the routines that can be readily incorporated in the global finite element environment.

After the complete direct problem is set up in the above environment for the finite element analysis, the numerical analysis can be connected with the optimization program *Inverse* in order to perform parametric studies, inverse identification of material and other parameters or process optimization. Figure 21 shows a general scheme for solving optimization problems when evaluation of the response functions include finite element simulation of the system under consideration. The right-hand part of the scheme is performed by the simulation software while the left-hand side is performed by the optimization program *Inverse*.

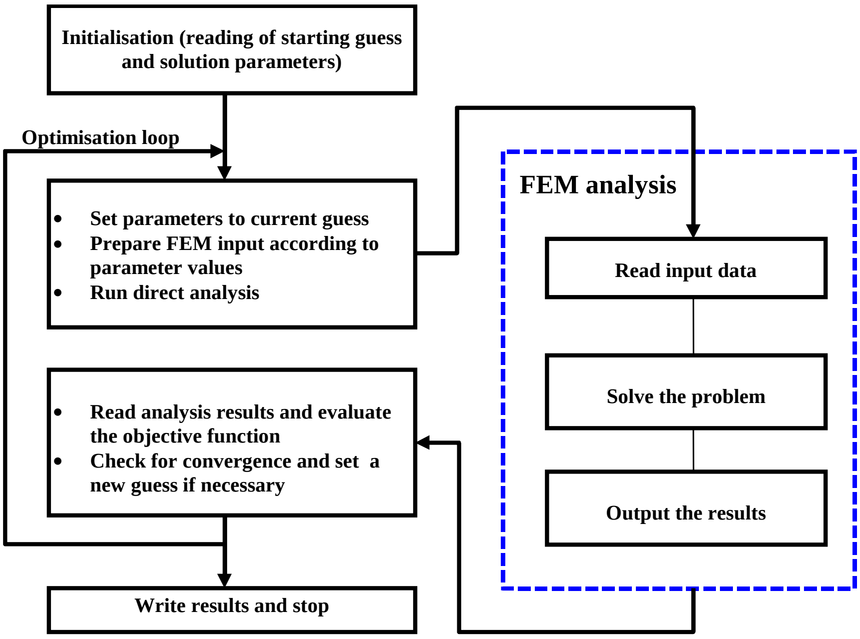

Initialisation (reading of starting guess and solution parameters)
Optimisation loop
• Set parameters to current guess
• Prepare FEM input according to parameter values
• Run direct analysis
• Read analysis results and evaluate the objective function
• Check for convergence and set a new guess if necessary
Write results and stop
FEM analysis
Read input data
Solve the problem
Output the results

Figure 21: Solution scheme for optimization problems.

Structure of the optimisation program is shown schematically in Figure 22. The software is centered around a command file interpreter that act as a user interface for accessing of program's functionality. The user defines the solution scheme for the optimisation problem in the command file that is interpreted by *Inverse*. Interpreter functions enable access to built-in functionality of the program including optimisation algorithms and interfacing utilities. In a special *analysis* block of the command file, user of the system defines how the response functions (i.e. objective and constraint functions) are computed. When the optimization problem is solved according to the scheme in Figure 21, includes use of utilities for interfacing the simulation software in order to prepare input for numerical analysis according to the current values of the design parameters and read the simulation results that are used for the evaluation of the response functions and eventually their derivatives with respect to design parameters. Link between optimisation algorithms and user definition of the response functions is established through pre-defined interpreter variables.

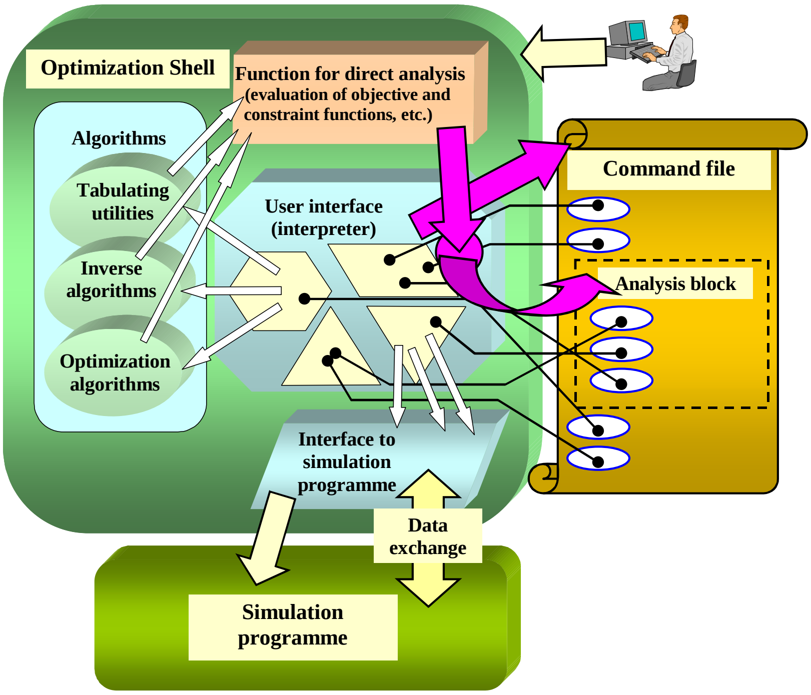


Figure 22: Outline of the optimization program *Inverse*.

In principle, *Inverse* can be used to utilise any simulation software for solving optimisation problems. A general file interface enables exchange of information with simulation environment through its input and output files. A direct interface may be constructed for a specific simulation code in order to improve control over performance of the numerical analysis through direct memory access to problem data and by access to the relevant functions. An example is direct interface with the commercial finite element system *Elfen*.

Additional documentation for *Inverse* is available at [6]-[9]. In particular, [9] can be used as quick introduction, while the download area[7] is a good starting point for using the program and includes references to further information.

For has direct access to the finite element driver[11],[12],[13], an interface with the symbolic system *Mathematica* has been built. The finite element driver is the environment that utilises the symbolically generated finite element code and has been used as simulation environment for the example described in Section 7, and is controlled by the *Mathematica* interpreter. The interface between *Inverse* and *Mathematica* functions on the interpreter-to-interpreter level. This means that interpreter of *Inverse* can be accessed through the *Mathematica* interpreter and vice versa.

6.2 Application programming interface for sensitivity and optimization procedures

The idea of approximation based optimisation and sensitivity analysis techniques has been outlined in Section 2 and approximation methods intended for use with these techniques were introduced in Section 5.

An example of approximation based optimization algorithm can be outlined by the following prototype algrithm:

1. Choose the center point $\mathbf{x}_0$ (initial guess) and the trust region parameter r.
2. Sample response functions in a chosen number of points contained in the sampling region $\Omega_{\mathbf{x}_i,r}$ centered around $\mathbf{x}_i$.
3. Build approximations of the objective and constraint functions.
4. Solve the problem with approximated objective and constraint functions, subjected in addition to step length constraints defined by r. Set a new $\mathbf{x}_i$ to the solution of this problem.
5. Update r according to algorithm progress and with respect to agreement of samples in the last iterate with approximations in previous iterates.
6. If not converged, repeat the procedure from step 2, otherwise post-process the collected data and finish with $\mathbf{x}_i$ as problem solution.

The primary design target is to spend as few function evaluations as possible for converging to the solution of the optimization problem with the requested accuracy, as well as to ensure stability in presence of noise. It is crucial for pursuance of this goal that sampling is performed economically. Usually less samples are calculated in each iteration than the minimum needed for the approximation, and are therefore combined with samples acquired in previous iterations. Attempt is also made to choose sampling points in a way that ensures maximal accuracy of approximations. In order to achieve that, a set of randomly generated sampling points can be generated and then improved by updating sampling point co-ordinates via the solution of a suitably defined optimization sub-problem.

There are several sub-problems that can be treated individually in such an algorithm, such as the problem of choice of the sampling points, approximation problem, approximated optimisation problem with step restriction, convergence check, problem of updating adaptive parameters. A library called *IoptLib* (investigative optimisation library) is being built to provide the basic utilities for solution of these sub-problems. The design of such a library should enable efficient combination of such utilities in order to form complex solution schemes, which can be quickly adapted to particular problems and requirements. For this purpose, the library specifies internal standards of how particular utilities exchange information, which includes definition of standard data formats and function prototypes where this turns beneficial. The library will be provided as free open source code written entirely in plain ANSI C, which will ensure high level of platform independency and enable easier exchange of ideas and solutions between researchers in the field of optimisation and users of optimisation algorithms.

The approximation based algorithms used in this project will be developed on the basis of *IoptLib*. The definition of user interfaces for accessing optimisation techniques will therefore be adopted in this specification. Accommodation to the standards provided by the library has been partially carried out in the optimization program *Inverse*, and specifications provided by the library are also reflected in the extensions to interface with *Mathematica* , which are provided in the Mathematica notebook *opt_inverse.nb* that can be downloaded from [7]. This interface is used in formulation of the optimisation procedures for the test case outlined in Section 7.

6.2.1 Standard Analysis Function

The core part of the interface specification is the definition of the *standard analysis function prototype*, which defines the form of functions that compute the system response at given values of the design parameters, cf. equation (1). Such functions are iteratively called by the algorithms for solution of the optimisation problems, and must be provided by the caller. The caller of an algorithms provides the algorithm with his or her particular definition of the optimisation problem to be solved.

Beside the obvious task of transferring the values of the design parameters to the analysis function and obtaining results after its execution, the function prototype should provide means of making precise definition of the response functions by additional “definition parameters” and easy utilization of existing definitions of response functions in definition of modified or derived problems. Above all, this requirement is important form the point of view of library design, however it also has applicability in the design of solution environments for optimization problems.

As illustration of the first requirement, one can define a generic analysis function for quadratic problems (i.e. with quadratic objective function and linear constraints) where coefficients of the quadratic and linear functions must be provided in order to precisely define the problem. In a similar way, one can provide coefficients and basis functions of approximations of the response functions in order to define the approximated problem, which is solved within the approximation-based optimisation scheme. In the restricted step approach is used in such a scheme, the approximated problem must in addition include the step restriction constraints, which are precisely specified in each iteration. If we have e.g. a generic analysis function based on approximated response (where coefficients and basis functions of approximations are provided as definition data) and another analysis function for definition of step restriction constraints (where the definition data consists e.g. of the centre of the restricted region and its size), then it must be possible to define a combined analysis function, which precisely defines the overall optimisation sub-problem (i.e. approximated original problem with added step restriction constraints) that must be solved within the approximation based optimisation scheme. In this way, the second requirement mentioned above is satisfied.

In order to fulfil the important requirements for the form of functions that perform response evaluation, the standart analysis prototype is defined (in C) as follows:

```
typedef
  int (*analysis_bas_f) (
      vector param,int *calcobj,double **addrobj,
      int *calcconstr,stack *addrconstr,
      int *calcgradobj,vector *addrgradobj,
      int *calcgradconstr,stack *addrgradconstr,void *cd);
```

This defines a function type with which the analysis functions in the libraries that call optimisation algorithm must be consistent. The meaning of function arguments is explained in Table 1.

Table 1: Meaning, types and dimension of arguments of the *standard analysis functions*. (type *analysis_bas_f*). In the table below, integer numbers *numparam* , *numconstraints* and *numobjectives* denote number of parameters, number of constraints and number of objective functions (only 0 or 1 are possible), respectively.

Argument	**Meaning**	**Remarks**
vector param	Vector of design parameters.	In general, it must be allocated with correct dimension, i.e. *numparam*.
Flag pointers	Input/output. Define what to evaluate and inform what has been evaluated.	Input/output. Pointer to non-zero value means that evaluation is requested, NULL or pointer to 0 means evaluation is not requested. Output (when evaluation is requested): if evaluation is requested then pointed value is set to 0 if evaluation or return of corresponding results could not be done or if the corresponding response is not defined in the problem corresponding to the analysis function.
int *calcobj	Objective function evaluation.	Requests evaluation of the objective function.
int *calcconstr	Constraint functions evaluation.	Requests evaluation of constraint functions (all in a package).
int *calcgradobj	Evaluation of gradient of the objective function.	Requests evaluation of the gradient of the objective functions.
int *calcgradconstr	Evaluation of gradients of constraint functions.	Requests evaluation of gradients of constraint functions.
Storage addresses	Define address for storage of calculated response	Output. For each type of response there is an argument specifying storage address. Arguments must not be NULL when evaluation of given response is requested (but may be NULL when it is not). Storage is allocated/reallocated by the analysis function when necessary and kept untouched when evaluation of corresponding response is not required. When a given kind of response is requested but it is not defined, the storage would be untouched, but corresponding flag would be set to 0.
double **addrobj	Objective function storage.	**addrobj is set to the value of the objective function. *addrobj is set to NULL when objective function is not defined.
stack *addrconstr	Storage for constraint functions.	Stack holds *numconstr* elements of type double *, which hold values of constraint functions.
vector *addrgradobj	Storage for objective function gradient.	Vector of dimension *numparam*, elementa are components of the objective function gradient.
stack *addrgradconstr	Storage for gradients of constraint functioins.	Stack of *numconstr* elements of type *vector*. Vectors are of dimension *numparam* and hold gradients of individual constraint functions.
Definition data	Additional exchange of information.	Intended for different roles: precise definition of analysis response (e.g. coefficients of quadratic objective functions), may be used for data transfer between the algorithm, analysis and user (state & requests), seamless upgrade of analysis (e.g. non-derivative analysis upgraded by numerical differentiation) etc.
void *cd		Input and/or output, not compulsory. Type and structure of the pointed data is arbitrary, it is interpreted within the analysis function. May be NULL when additional data is not necessary. Caller of the analysis function must know and obey the rules for type and layout of the pointed data, which are defined on the analysis side.
Info mode	When *calcobj*, *calcconstr*, *calcgradobj* and *calcgradconstr* are all NULL, the analysis	

All flag pointer arguments are NULL	function operates in **Info** mode. It does not evaluate anything, but checks all storage address arguments that are different than NULL and allocates or re-allocates the addressed storage if necessary in such a way that all the dimensions of the allocated storage are consistent with the problem defined by the analysis function (e.g. *addrgradconstr* will point to stack with *nconstr* vector elements of dimension *numparam*, provided that there are also constraints in the response).	
Return value (int)		0 if everything is OK, usually a negative error code of the calculation could not be performed correctly.

It must be mentioned that the specified form of standard analysis function also enables easy definition of wrapper functions that can be used to convert other forms of analysis functions to the standard form. This enables utilisation of external libraries that can not be modified by users and connection of provided functionality with the optimization algorithms that conform with *IOptLib* standards. For example, simulation environments may provide their own routines for evaluation of response functions which provide access to the definitions of the optimisation problems that are set up interactively in the graphical user interface of the simulation environment.

The return value and evaluation flags provide the means of complete control over what is evaluated and possible exceptional situations that may occur. Evaluation flags are input **and** output parameters and provide a valuable information to the calling optimisation algorithm. For example, an algorithm may be designed in such a way that they can try to avoid regions in the parameter space where the objective function can not be evaluated. In this case the return values of evaluation flags provide information that is as valuable as the values of the response functions themselves, while less sophisticated algorithm may choose not to utilise this information and only check the return value in order to detect exceptions.

The argument *cd* is a pointer of undetermined type and provides the possibility for specification of the additional definition data that precisely specifies how the computation of response functions is performed. For example, it can contains coefficients of quadratic and linear functions that define the quadratic programming problem (QP). This also provides means for combination of different objective and response functions, e.g. for combining the approximation of the original problem with restricted step constraints. In the latter case, *cd* would contain both function pointers (corresponding functions) and definition data pointers for approximate response and restricted step constraints.

The definition data is used only by the analysis function itself and by the code that sets up the precise definition of the problem (and must therefore be aware of the definition of the analysis function). This is the reason that use of the pointer of unspecified type (i.e. *void* * in C) will not cause any conflicts. In the case that definition data is necessary, definition of the optimization problem consists of both the analysis function **and** definition data, and both must be provided to the algorithm that is called to solve the problem. For the algorithm, the meaning of the definition data is irrelevant. The algorithm does not need to interpret the pointed data, but only passes the pointer to the analysis function. For the algorithm, *cd* is merely an address of the memory location where the analysis function that it calls will find the necessary data.

In some cases, analysis function will contain the complete definition of the problem without the need to provide supplemental definition data. In such cases, the NULL pointer can simply be passed to the analysis function (since it will be ignored anyway).

6.2.2 Calling of Optimisation Algorithms

Beside the algorithm specific data such as initial guess, tolerances, maximal number of iterations, a number of flags that define the behaviour of the algorithm and other data, the caller must provide the definition of the optimization problem to be solved. According to the present specification, this is done in the standard manner by passing the pointer to the analysis function and to the definition data. The algorithm will then call the analysis function and pass it the definition data pointer any time it will need to evaluate the response (i.e. objective and constraint functions) at different design parameters. Algorithms designed in compliance with the present specification will strictly obey this rule.

The function that calls such an algorithm may be defined in the following way:

```
int optimize (vector initial, double tolerance, int maxiterations, ...,
            vector *optimum, analysis_bas_f analysis, void *cd).
```

In this hypothetical example, *initial* is the vector of starting values of parameters (initial guess), *tolerance* is the parameter that defines when satisfactory convergence is achieved according to the applied criterion, *maxiterations* is the macimum number of iterations after which the algorithm stops even if convergence is not achieved, and *optimum* is the output argument through which the calculated optimal parameters are passed to the caller.

The last two arguments define the optimization problem that is solved by the algorithm. ***analysis*** is the pointer of the analysis function that is called by the algorithm to evaluate the response. ***cd*** is the pointer to definition data used by the analysis function that completely defines how the evaluation of response functions is performed, and is passed to the analysis function each time it is called by the algorithm.

6.2.3 Specification for Functions for Numerical Differentiation of the Response

As regards definition of the optimization problem (i.e. evaluation of the response functions), similar agreements apply as for optimization algorithms.

The result of numerical differentiation are gradients of the response, which are already accounted for in the definition of the analysis function prototype. The functions for numerical differentiation can therefore be regarded as functions that adds gradients to the original non-gradient response, for which it must perform additional evaluations of the non-gradient response at different parameters around the evaluation point. Therefore, the standard analysis function prototype can be conveniently used:

```
int numgradanalyse (
      vector param,int *calcobj,double **addrobj,
      int *calcconstr,stack *addrconstr,
      int *calcgradobj,vector *addrgradobj,
      int *calcgradconstr,stack *addrgradconstr,void *cd);
```

The definition of the original problem (for which derivatives are not provided) is contained in the *cd*, as well as eventual additional parameters required by the numerical differentiation algorithm. The precise structure of the data pointed to by *cd* will dependent on the method that is used. The agreement is, however, that the first two fields in the data pointed to by *cd* are the function pointer of the original (non-derivative) analysis function and the pointer to its definition data.

For example, in the finite difference method defined by the equation (11) is used, the additional parameters include the vector of step sizes for numerical differentiation for each of the design parameters. In this case, *cd* is of the following type:

```
typedef struct _numdifcd_fd {
    analysis_bas_f analysis;
    void *analysisdata;
    vector stepsiyes;
} *numdifcd_fd;
```

The last field of the structure is of the *vector* type declared as

```
typedef struct _vector {
    int d;      /* dimension (num. of comp.) */
    double * v; /* table of elements, STARTS WITH 1! */
} *vector;
```

6.3 Optimisation Related Integration Issues[7]

The present document introduced some general aspects of sensitivity analysis and provided specifications that are relevant for integration of the sensitivity and optimization module with simulation modules in order to perform optimization related tasks. A test case has been set up (Section 7) to demonstrate actual realization of such a task based on the provided specifications, and this section addresses some practical issues related to the specific tasks within the project where optimization procedures will be applied.

These tasks are divided to back analysis of geological conditions and to upscaling of rock mass properties from parameters that can be determined by laboratory tests to the model parameters that can be used at the full scale of the tunnel excavation area. The significance of these tasks is in improvement of methods for prediction of ground behaviour, for which a shift from rough and partially descriptive (qualitative) characterisation of rock mass behaviour to more accurate quantitative characterisation is necessary.

6.3.1 Solution scheme shown on an artificial back analysis example

The task of back analysis is to determine a set of unknown geological material and shape parameters and eventually the initial stress state, which apply to the rock mass around the excavation area. The unknown parameters should be determined on basis of on site measurements of the ground response (e.g. displacements after excavation). This can be done

[7] Although the title of this deliverable refers to a software to determine sensitivities, specifications given in section 6.2 also apply for optimization modules. It is optimization procedures that are actually important for tasks in TUNCONSTRUCT as currently envisaged, while numerical differentiation will eventually be integrated in the module and used implicitly (only developers will really be aware of it), as mentioned in the Introductory part of this document. We therefore consistently refer to optimization procedures rather than sensitivity, to make the specification more relevant from application point of view.

by solving an appropriate optimization problem, where a function is minimized that measures discrepancy between the actual measurements and equivalent quantities calculated with numerical model.

6.3.1.1 Analysis function

The problem is solved according to the scheme in Figure 21. Concerning the software architecture, the basic task at the simulation side is to provide a function (subroutine) for calculation of the discrepancy function. More generally, we will call this function the ***direct analysis function***. It acts as the function for evaluation of response functions from (1). Specification of function prototype is provided in Section 6.2.1. What this function typically contains is best explained on example (Figure 23). The example is purely hypothetical and unrealistic, but is chosen in order to demonstrate what must be provided on the simulation side.

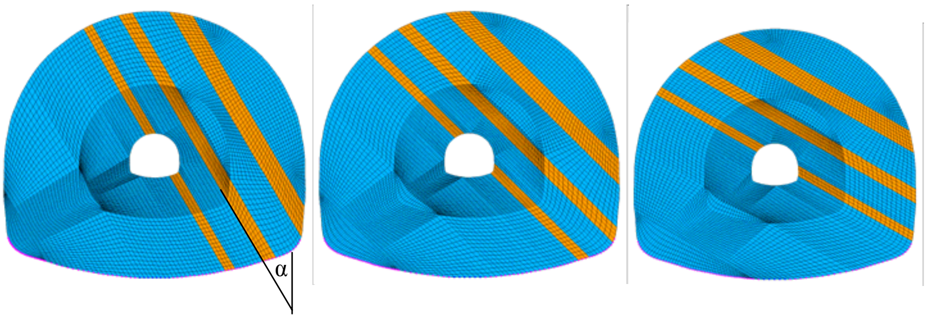


Figure 23: Numerical models for different inclination angles α.

Let us assume that two different kinds of rocks with known material properties and contact conditions between them are arranged in layers whose inclination angle is unknown. We measure displacements around the excavated area after excavation, and would like to estimate the unknown inclination angle on basis of these measurements. In order to perform the problem, we define the discrepancy function to be minimized, e.g.

$$f(\alpha)=\sum_{i=1}^{m}\left(d_i(\alpha)-d_i^{(m)}\right)^2 \quad , \tag{64}$$

Where $d^{(m)}$ are measured displacements in m points around the circumference of the cavity, $d_i(\alpha)$ are equivalent quantities calculated by the numerical model (Figure 23). We minimise f to find the parameter α at which the discrepancies between the actual and simulated measurements (in term of the above measure) are the smallest and assume that actual inclination angle of the layers must be something close to this value[8].

The body of the direct analysis function will comprise the following basic steps:

[8] Under specific circumstances, the mean value and expected error in parameters can be estimated

1. Take current optimization parameters (input arguments of the function; *α* in this case)
2. Prepare numerical model according to parameters (see Figure 23)
3. Run numerical simulation of the excavation process with this model
4. Extract the relevant quantities from simulation results (in this case, displacements $d_i\left(\alpha\right)$)
5. From measured data (which must be available somewhere, e.g in the file; in our case, measured displacements $d^{(m)}$) and from equivalent calculated data $d_i\left(\alpha\right)$, calculate the response functions and eventually their gradients (in our case the discrepancy function *f*)
6. Store the response functions in output arguments and return

Figure 24: Discretized model with design parameters that can be varied and their reference values corresponding to the picture.

It is also nice if the function checks for possible errors and returns the error code if something suspicious is detected that could corrupt the results. Input and output arguments should comply with specification provided in Section 6.2.1. In general, things will be a bit more complex for the following reasons:

- In general, there will be more than one parameters **x** (in our case, **x**={*α*}).
- Beside the objective function *f*(**x**), constraints may also be specified in the problem, in which case the analysis function must be able to calculate the constraint functions c_i(**x**).
- The analysis function may also provide gradients of the response functions
- Some optimization algorithms don’t need evaluation of all response functions at every step, and calculation of only part of the response may be significantly cheaper than calculation of everything. This is why evaluation flags are envisaged, which define what should be evaluated at a particular call.

Figure 26 contains a list of possible parameters to be identified in the described example. Notion of constraints can also be easily illustrated. In the present example, the finite element mesh can be too distorted when *α* exceeds some moderate range. Numerical model can give useless results in this case, which can in turn cause the optimisation procedure to fail. To prevent that, we can define that the parameter may not exceed a given range, e.g. $\alpha_l \le \alpha \le \alpha_r$, which can be done (c.f. (1)) by defining two constraint functions:

$$\begin{aligned} c_1\left(\{\alpha\}\right) &= \alpha_l - \alpha \\ c_2\left(\{\alpha\}\right) &= \alpha - \alpha_r \end{aligned} . \tag{65}$$

Many algorithms will relatively easily recover from strange results of the model[9] if it is clearly indicated that constraints are violated at given parameters. With regard to our case, actual inclination angle may lie outside the range defined by (65). In this case, the algorithm would probably converge to the closest point on the boundary of the allowed region, which would give us a chance to re-define the geometry in such a way that regular models would be produced over a different range, and run optimization problem again.

In general, constraint functions will not be explicit functions of parameters but will incorporate results of the numerical analysis (similarly as the objective function). For example, we may require that a measure the measure of maximal distortion of any element of the mesh after loading is below some accepted limit (for which all nodal displacements must be calculated).

Remark on parameterization:

Point 2 of the Figure 24 is referred to as parameterisation. Parameterization utilities and utilities for processing of results of numerical simulation (that enable definition of response functions on basis of the simulation results) are the two additional things that must be provided at the analysis side in order to utilize a simulation environment for solving optimization problems. They are tightly bound to the simulation environments since they depend on representation of data in a particular environment.

In practice, two different approaches can be used. In one approach a set of utilities that enable virtually any kind of parameterization is provided in advance. These utilities can be used by any user of the simulation environment independently of their developers for formulation of a large variety of problems. It must be mentioned however that the set of problems that can be covered in this way is inevitably limited. The approach can still be applied for well defined standard sets of problems.

In the second approach, parameterization is prepared on a case to case basis. This requires that when a user of the simulation environment wants to set up a new optimization problem, a support of the developer who prepares parameterization is provided.

6.3.1.2 Optimization functions

The task of the optimisation module is to solve the problem (1), for which some input is needed from the calling environment that requests the solution. Optimization module will provide a set of optimization algorithms implemented as functions (routines). Input of the algorithm (i.e., the corresponding function that contains its implementation) will consist of standard parameters such as the initial guess, required tolerances[10], etc., the inevitable definition of the problem (the previously described direct analysis function / routine). Output will consist of the optimal parameters and eventually the values of response functions at these parameters.

[9] Many optimization algorithms expect certain nice properties from the response functions they operate on, e.g. a given continuity class. As a rule of thumb, the more sophisticated and efficient the algorithm is, the more important the the response properties. And efficiency (in terms of number of function evaluations spent to obtain the solution with a given accuracy) gets quite important when a single evaluation of the response takes hours or even days.

[10] What this actually is will depend on the algorithm. There are a lot of different possibilities with respect to on what one can impose a tolerance, e.g. accuracy of optimal parameters, accuracy of function value, maximal size of the gradient norm, etc. Specifics will be provided in algorithm specifications.

An example algorithm function prototype is stated in Section 6.2.2. No uniform specification is provided for this because algorithms differ widely with respect to input and output parameters. If we tried to provide a standard function prototype, the argument list should account for all possible variants, which would bring little benefit for the price of uncountable possibilities for incorrect use. Each time a new algorithm will be provided for a particular purpose, it will come with exact specification of input/output arguments and probably a short description of a range of problems for which it is suitable[11]. Each algorithm separately will be linked with environment where it is used according to specification.

It is important to note that the algorithm will repeatedly call the direct analysis function that is provided through its input. The analysis function is provided by the calling environment that calls the optimization algorithm. The algorithm will call the analysis function autonomously (i.e. the calling environment will not have any influence on this).

Side remark:

Specification of the analysis function in Section 6.2.1 anticipates an additional specification parameter (denoted *cd*) that can be used to precisely specify (apart form optimization parameters **x**) how the response is evaluated. This data differs form optimisation parameters in that it is typically not changed during the optimization procedure, in fact the optimization algorithm has nothing to do with it[12]. This piece of data provides a possibility for the calling environment to communicate with its direct analysis function without the need for using global data (which is usually not an elegant solution). This data will be passed as input argument to the optimisation algorithm. The optimization algorithms will be obliged to pass the data to the analysis function each time it is called. The mechanism will be consistently implemented, but its use can be avoided (by simply not using the data in the analysis function).

In the solution scheme in Figure 21, the analysis function consists of the right-hand side of the figure plus the following steps:

- Prepare FEM input according to parameter values
- Run direct analysis
- Read analysis results and evaluate the objective function

Everything else is a part of the optimization algorithm. Typically, these two components will be incorporated in a broader environment that will enable manipulation of numerical models, definition of analysis functions and execution of optimization requests. Implementation of such environment (e.g. in terms of user interfaces) is not the subject of this document. Figure 25 shows

[11] Unfortunately, this will be a rather abstract and dry description not having anything to do with actual problems.

[12] This will not allways be the case, but this fact is relevant for nobody but developers of the optimizatiion module.

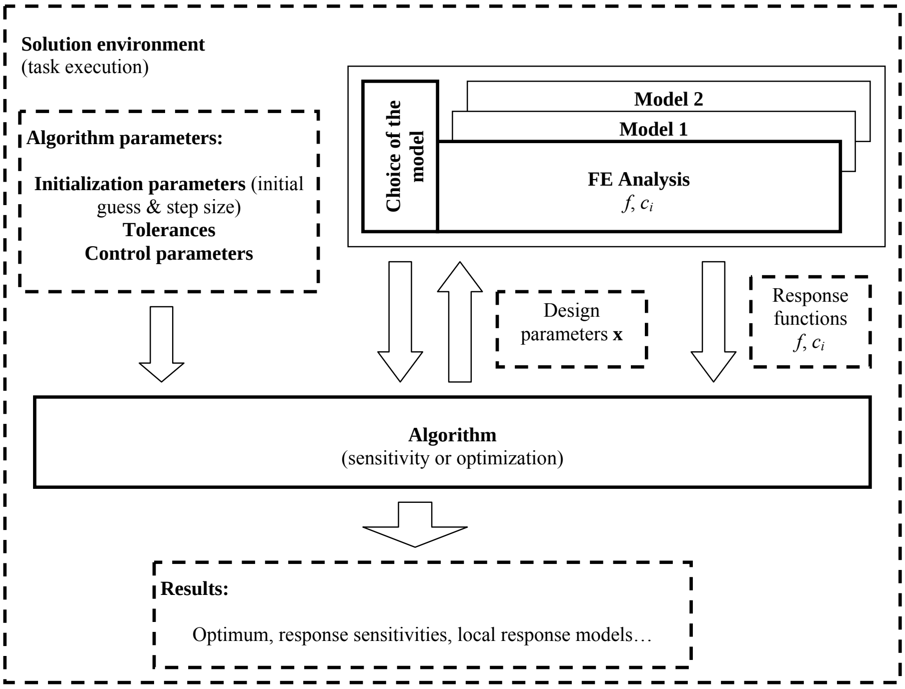


Figure 25: Function of an optimization algorithm within solution environment.

6.3.1.3 Putting things together, dependence on the specific case and organisation issues

The presented solution schemes utilize the "black box" concepts. The optimization algorithm uses the analysis function merely as an entity for calculation of the response functions without interfering with what these functions are. Similarly, the analysis function is just packed and passed to the algorithm, without questioning what the algorithm will do with the results. The algorithm represents merely a mathematical procedure for solving the problem while the analysis function contains the definition of the problem to be solved. It is up to designer of the analysis function to define which optimization problem should be solved e.g. for calculating the appropriate parameters in back analysis, and it is up to the algorithm to do the job in an efficient and reliable manner and spit out the results.

Before all this is a technical arrangement with a number of advantages. Exchange of information between the optimization module and simulation kernels is reduced to a minimum and standardised. This enables independent development of algorithm and of the analysis model and easy connection of those when things are finished.

In practice, the optimization part, the simulation part and the definition of the problem are interrelated. The particular problem may impose requirements on the optimization side that will call for some special algorithmic solutions (e.g. problematic handling of constraint violation). For efficient and successful solution of the problem, it is crucial that s suitable optimization algorithm is used[13]. On the other hand, details in definition of the response functions may have a great impact on performance of the optimization algorithm, e.g. because of affecting smoothness of the response.

[13] There is no such thing as optimization algorithm suitable for all kinds of problems.

Ideally, complex problems should therefore be solved by close co-operation of experts in different areas involved. Technically, the black box concept should still be advantageously applied as a technical instrument for making development and co-operation efficient. Collaboration should be practiced at a higher level where specific problems are analysed and discussed from different aspects before the solution schemes are defined and applied.

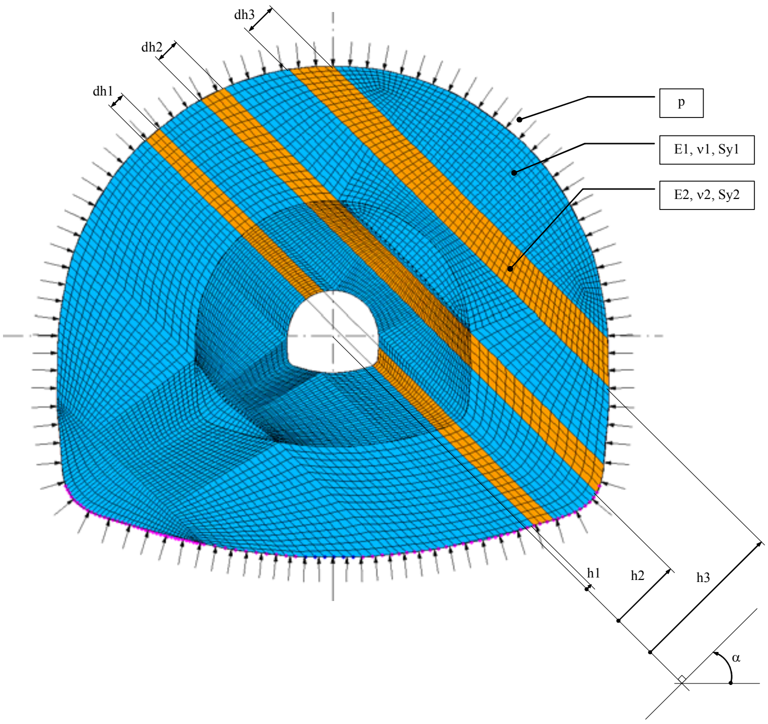
Common rules of collaborative work should be followed. This means that clear and concise specifications of work to be done & interactions are provided in advance, and in particular that each partner provides completed implementations on the side for which that partner is specialised, together with clearly described interfaces that require minimal effort when connecting different modules.

7 Test Case for Validation of Inverse and Sensitivity Procedures

In order to validate the sensitivity and back-analysis procedures, a synthetic case was set up with assumed conditions at a tunnel construction site. The simulation and analytical differentiation procedures (including parameterisation of the design) were set up in the finite element driver[11],[12]. This is a finite element environment that offers advantage of symbolic derivation of expressions and automatic generation of finite element code, together with very flexible environment for problem definition supported by the symbolic system *Mathematica*.

The test case consists of a 2D finite element model for computation of displacements after excavation of the tunnel. Figure 26 shows the discretization of the excavation area and parameterization of the model. Parameters of the model include geometric parameters (thicknesses, height and inclination of geological layers), material parameters for the two types of rocks (elastic modulus, Poisson coefficients and yield stress) and tractions on the border of the discretized region before excavation caused by the weight o fsurrounding rocs.

For back-analysis it is assumed that we can measure displacements of material points in the circumference of the excavated domain. The objective function is defined as sum of squared differences between the "measured" displacements and the corresponding displacements calculated by the numerical model at given trial parameter values. Actual parameters are obtained by minimisation of the objective function. Since this is a synthetic case that serves for validation of procedures, the "measured displacements" are obtained by numerical analysis of the model with the assumed (or reference) parameter values. In this way we know exactly which parameters must be obtained by the back analysis and can therefore verify the performance of back analysis procedures under specified conditions.



Shape parameters

- h1=0.3 m
- h2=3.3 m
- h3=7.0 m
- dh1=0.8 m
- dh2=1.1 m
- dh3=1.5 m
- α=(45.) Degree

Material parameters

- E1=10*10^9 N/m^2
- ν1=0.25
- Sy1=13*10^6 N/m^2
- E2=100*10^9 N/m^2
- ν2=0.1
- Sy2=14*10^6 N/m^2

Boundary tractions

- Pres=5*10^6 N/m^2

Figure 26: Discretized model with design parameters that can be varied and their reference values corresponding to the picture.

Sensitivity analysis has been performed by analytical differentiation of the numerical model. Derivatives of nodal displacements and the objective function with respect to all parameters have been calculated. Detailed results are in shown in the presentation for the workshop while in this document a few characteristic results are presented in figures below.

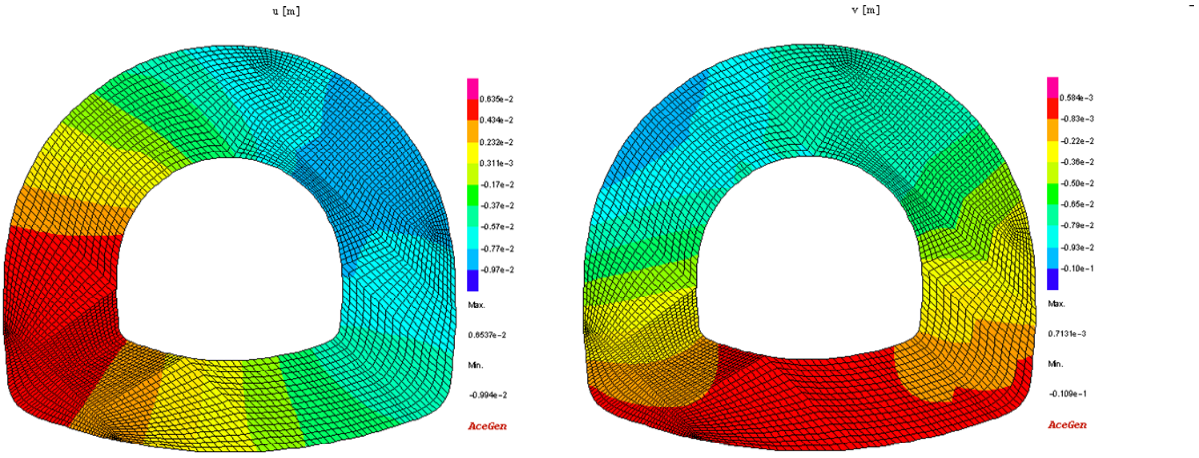


Figure 27: Displacements around the excavated area.

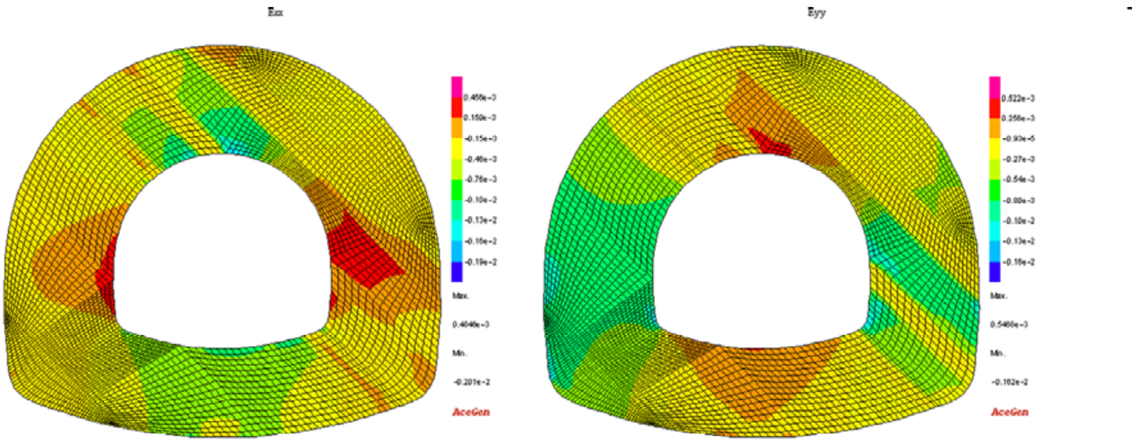


Figure 28: *xx* and *yy* components of strain.

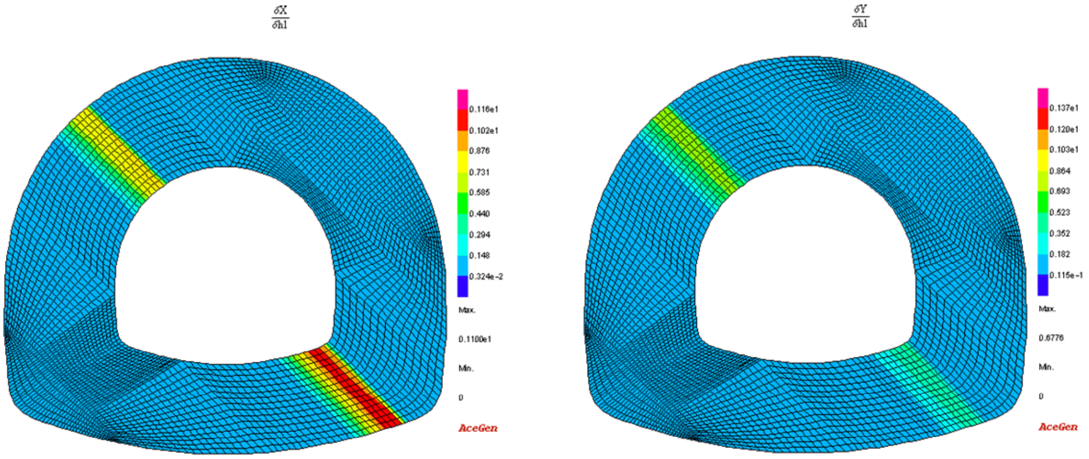


Figure 29: Derivatives of nodal co-ordinates with respect to height of the first layer, $h1$.

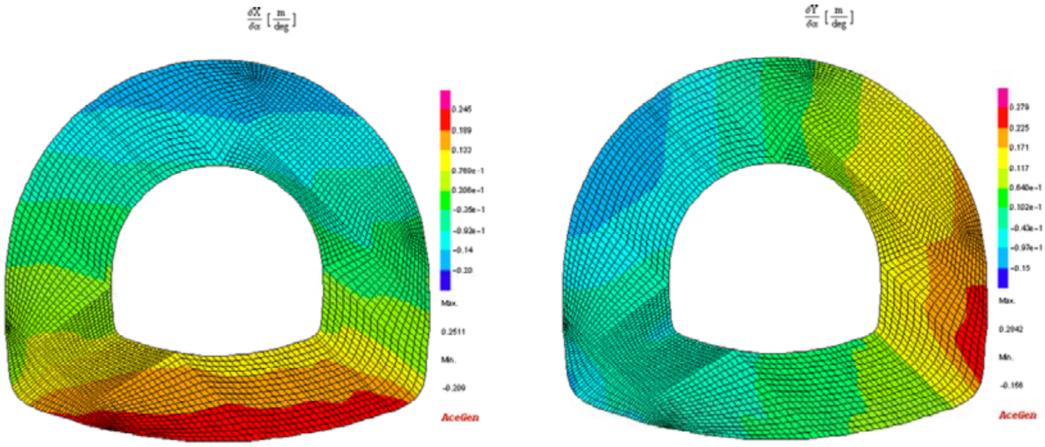


Figure 30: Derivatives of nodal co-ordinates with respect to inclination angle of layers, α.

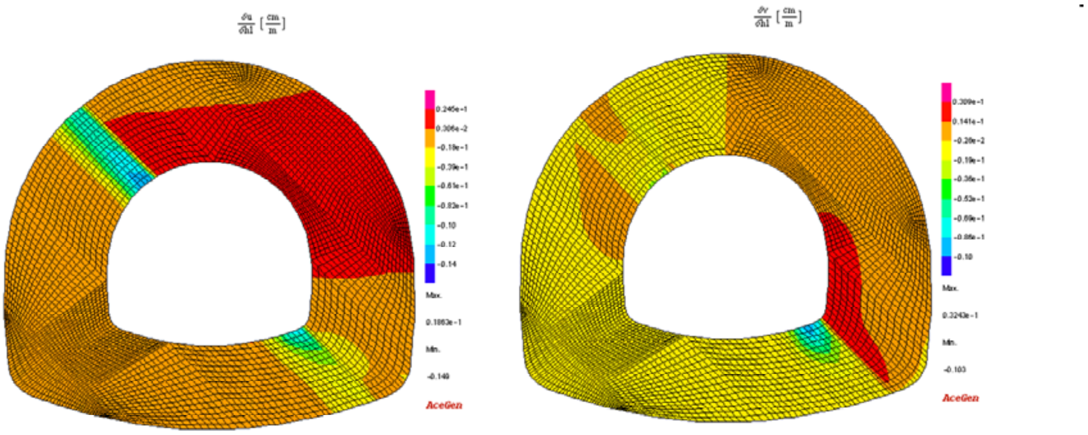


Figure 31: Derivatives of nodal displacements with respect to height of the first layer, $h1$.

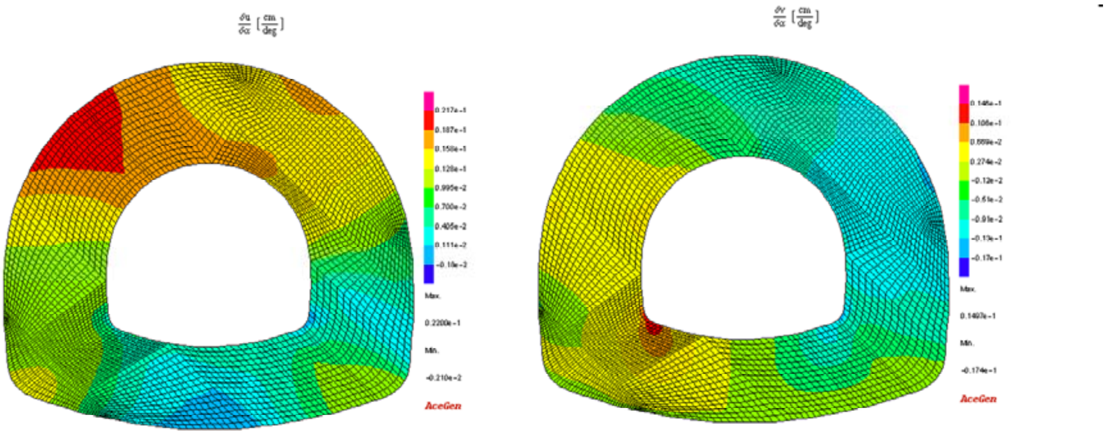


Figure 32: Derivatives of nodal displacement with respect to inclination angle of layers, α.

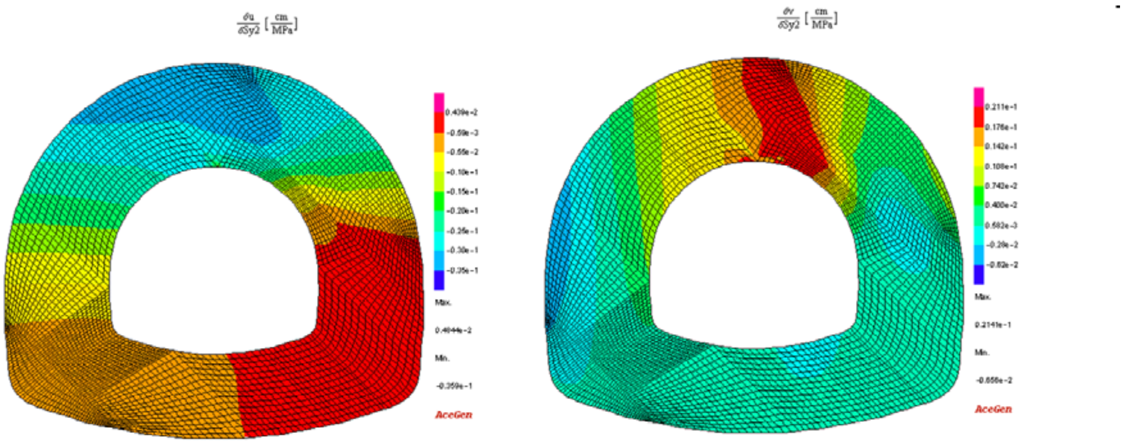


Figure 33: Derivatives of nodal displacement with respect to yield stress of rocks that form the layer.

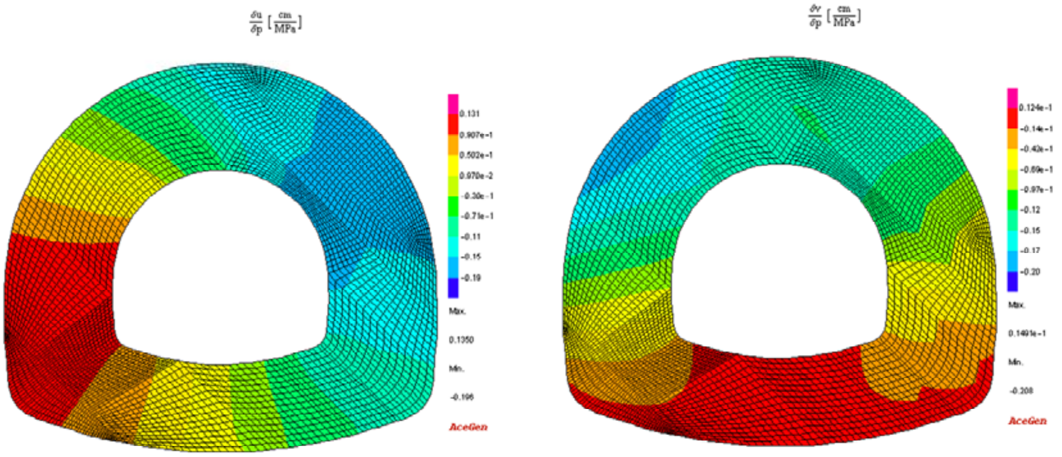


Figure 34: Derivatives of nodal displacement with respect to prescribed normal tractions.

8 Plans for Integration with the optimization related tasks within TUNCONSTRUCT

The present document introduced some general aspects of optimization and provided precise specifications (Section 6) to be used for integration of the optimization module with simulation modules in order to perform optimization related tasks. A test case has been set up (Section 7) to demonstrate actual realization of such a task based on the provided specifications, which can be used as a reference example for further work when other cases are set up.

This section addresses the specific tasks within the project where optimization procedures will be applied. It also specifies the optimization procedures to be developed and their application in TUNCONSTRUCT.

After discussions and review of practical situations in tunnel building, it has been established that application of automatic design optimization procedures would not be relevant for tunnel design. Because of an extensive variation of geological conditions and large amount of uncertainty in their knowledge, the most suitable design procedure is based on selection of different pre-defined construction methods, which are selected on basiss of general evaluation of geological conditions and possible system behaviour.

On the other hand, improvement of numerical models that can be used for predictions is of great importance. Estimation of initial stress state and ground behaviour is currently the most critical for accuracy of the models used to determine system behaviour. We have therefore decided to concentrate the optimization related activities on the problem of back analysis and upscaling of laboratory data.

8.1 Up-Scaling and homogenization of lab data using back analysis of reference projects

The currently used up-scaling systems (GSI, RMR) are purely empirical and establish a connection between merely descriptive values of the rock mass (joint roughness classes and weathering state, block size/joint spacing) and mechanical parameters for the rock mass. Such approach has the advantage of being fast, and since the mentioned up-scaling systems are widely used and also updated to correlate with the observed mechanical response, it can be generally said that they are able to determine the order of the magnitude of the mechanical rock mass parameters. Nevertheless, the downsides of the approach are clear:

a) non-transparent transition from the lab test results to the rock mass parameters (due to the empirical nature of the system);
b) omitting the influence of the discontinuity orientation
c) questionable applicability to highly heterogeneous rock mass types (e.g. flysch, etc.)
d) generally, a non-mechanical approach to the problem.

We decided to start with a quantitative approach, using numerical simulations with a variation of the geometrical and mechanical parameters. Finding a set of smeared parameters can be treated as an optimization problem, with the aim of finding a solution at which the discrepancy function has a minimum.

8.1.1 Current status

After coordinative talks with IRMT the work on the up-scaling and homogenization of the rock mass parameters, it has been concluded that the current technical abilities of C3M are best suited to address the issue of the highly heterogeneous rock masses (for example: flysch). Provided with geological site data, C3M will conduct a series of calculations to determine the sensitivities of the ground response with respect to the variation of the lithological and geometrical parameters (Figure 1.). With the aforementioned discrepancy function, a set of elasticity and strength parameters can be calculated, with the given condition that the solution is achieved when the discrepancy function is at its minimum.

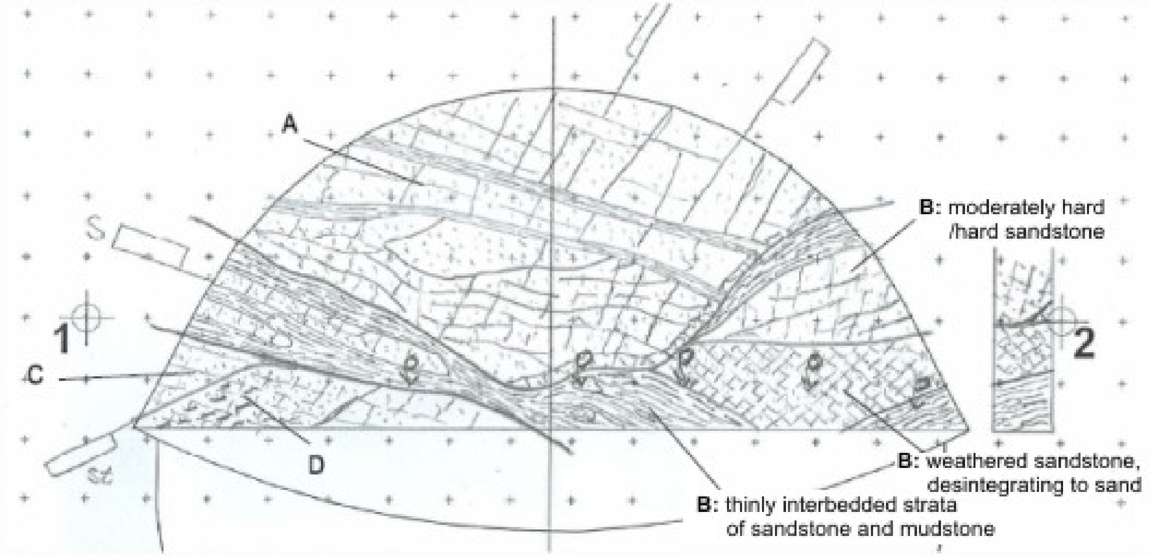

A
B: moderately hard /hard sandstone
1
2
C
D
B: weathered sandstone, desintegrating to sand
B: thinly interbedded strata of sandstone and mudstone

Figure 35: Geological composition of the excavated cross section.

For the task of pure back analysis of data from existent tunneling sites, a number of practical issues were discussed where members of C3M obtained valuable information from IMRT with regard to practical situations in tunnel construction. C3M provided feedback with respect to technical issues envisaged in back analysis (such as simulation software abilities, expectations for plausibility of back analysis in different situations with respect to geological structures and related uncertainties). Possible candidates for back analysis have been discussed such as spacing between geological layers, joint properties and initial stress, with emphasis of practical applicability in tunnel construction. The back analysis procedure has been roughly defined at conceptual level, while technical details and specific problems remain to be determined.

8.2 Development and application of optimization procedures to support tunnel design

Optimization methods based on successive approximation of the response functions will be developed for problems related to tunnel construction. The general framework for designing such algorithms has been outlined in Section 6.2. One of advantages of this framework is that it can be adapted to problems with large amount of noise. Since approximations are constructed by simultaneously considering response in a large number of sampled points, the effect of noise on algorithm progress is reduced. This effect is exposed in Section 5 through simple instructive demonstration of approximation properties with respect to noise[14]. Ability of handling noisy response is considered crucial for application to problems in tunnel construction. In simulation of system behaviour with accounting for inhomogeneous ground structure (which will be involved in back analysis & optimization), the source of noise is not only on the numerical side, but discontinuity of the response (with respect to optimization parameters) are inherently incorporated in physics of the system. Smearing procedures are possible at the level of calculation of response, but approaching the problem in this way can easily corrupt optimization results.

Schemes based on successive approximations allow for rich variability of implementations and enable good adaptation to specific problems. It is e.g. possible to build reliability based optimization procedures on these schemes[15], and it is possible to design algorithms that require gradients of response functions or not. This is important because we do not expect that main simulation codes used in TUNCONSTRUCT will provide analytical differentiation of numerical models (Section 4).

Approximateion based schemes have some technical features that are important for practical application. Approximations based on sampled response enable visualization and application of other analysis techniquees, algorithm restarts can be better supported and parallelization is easier.

Aside the advantages, there are many implementation details to be solved. The most significant for good algorithm performance is adaptive co-ordination of procedures for response sampling, determination of the restricted region and approximation strategy (choice of approximation approach and weights). For these components, universal solutions are not yet available. For determination of the sampling and restricted region in multidimensional cases, choice of orientation and scaling is the most important, and harmonization of approximation with these is crucial for algorithm stability. Beside mathematical

[14] Integration of response approximations in muti-dimensional optimization algorithms is a rather different thing, but the basic ideas can be grasped form these examples.

[15] By now, we have not identified problems where this would be necessary, but this can change in thefuture.

instrumentation that is applied to algorithmic design, a large degree of intuition and heuristics is always involved, and numerical experimentation plays a crucial role for confirmation of efficient approaches.

The optimization procedures will largely rely on the IOptLib (investigative optimization library)[11], which is being designed as a framework of efficient development and testing of algorithms, especially of approximation based techniques. The main emphasis is given on ability of easy combination of components for solving different tasks and flexibility of their modification, but a good testing environment and technical support for incorporation in solution environments are also accounted for.

A number of algorithm components (such as sampling and approximation) are already provided. Further work on components will concentrate on unification of procedures for scaling of the design space, validation procedures for support of adaptive schemes, general testing procedures and test examples. After this, components will be used to built variants of algorithms for basic testing, after which they will be ready for testing on applications. as new information on performance is available, refinements will continue and algorithms can be tailored to situations encountered during solution of practical problems. Swiftness of this process will depend on the amount of time available for these tasks, which we should define in the near future.

Beside the foreseen approximation based methods, application of other algorithms is possible in different situations. This can include e.g. the SQP or BFGS if analytical model when derivatives are available and the amount of noise is small[16], or methods derived from the Nelder-Mead method when derivatives are not available.

Optimization procedures suitable for design optimization (formulated according to equation (1) with continuum design parameters) can be based on the same framework than procedures for upscaling and back analysis. For optimization of design, treatment of constraints is more important.

In order to encourage awareness of importance of efficient collaborative approach (Section 6.3) and enable testing of analysis software incorporated in optimization scheme, it is intended that a template for simple, portable and safe way of linkage of optimization procedure with analysis cores will be provided for interested partners. This will be provide implementation of specifications for interaction with optimization procedures (Section 6) through simple file I/O operations and program execution, and will enable quick hard-coded implementations for solving optimization problems. With implementation of the analysis by re-running the simulation code each time, potential problems with improper memory cleaning and re-initialization will be avoided.

8.3 Application to specific tasks

First we envisage determination of material parameters of the ground model by up-scaling of the laboratory data. This procedure would yield material parameters for the homogenized model, based on comparison (and minimization of discrepancy) of the system

[16] Which is not expected other than for some special cases.

response (in terms of selected measurement data) calculated by the up-scaled homogeneous model and by the inhomogeneous model taking into account actual geological structure.

In the case of back analysis, the determination of model parameters would be done directly for the inhomogeneous model, based on in situ measurements. Initial stress, joint properties, intact rock properties and geometrical parameters such as joint spacing would be the subject of back analysis. Analysis software used for this has not yet been determined.

After testing the performance of back analysis procedures , it will be possible to link the optimization modules with simulation software developed in TUNCONSTRUCT. Specifications for integration of optimization utilities described in Section 6 should be used for this purpose.

9 Conclusion

Theoretical aspects and software implementation issues related to evaluation of sensitivities for upscaling and back-analyses are provided in this deliverable. Based on these considerations precise specification how solvers available in TUNCONSTRUCT could be integrated into sensitivity and optimisation procedures are described. It is pointed out that analytical differentiation methods are feasible in conjunction with AceGen system while numerical differentiation procedures will be available for other solvers (EKATE, BEFE++, FLAC3D, ELFEN). The necessary conditions to apply inverse and optimization procedures with these solvers are parametric description of the direct problem, reliable evaluation of responses for different sets of parameters and provision of objective/constraint functions according to the specification. The direct models will be set up by the partners who must ensure that their direct analysis software can be run automatically based on parameterized data sets.

10 References

[1] O. C. Zienkiewicz, R. Taylor, *The Finite Element Method*, Vol. 1, 2 (fourth edition), McGraw-Hill, London, 1991.

[2] P. Michaleris, D. A. Tortorelli, C. A Vidal, *Tangent Operators and Design Sensitivity Formulations for Transient Non-Linear Coupled Problems with Applications to Elastoplasticity,* Int. Jour. For Numerical Methods in Engineering, vol. 37, pp. 2471-2499, John Wiley & Sons, 1994.

[3] I. Doltsinis, T. Rodič, *Process Design and Sensitivity Analysis in Metal Forming*, Int. J. Numer. Meth. Engng., vol. 45, p.p. 661-692, John Wiley & Sons, 1999.

[4] J. Korelc, *Automatic generation of finite-element code by simultaneous optimalization of expressions*, Theoretical Computer Science, 187, p.p. 231-248, 1997.

[5] J. Korelc, *Symbolic Approach in Computational Mechanics and its Application to the Enhanced Strain Method*, Ph. D. thesis, Technische Hochschule Darmstadt, 1996.

[6] I. Grešovnik. "*Optimisation Shell Inverse*", electronic document at http://www.c3m.si/inverse/ , maintained by the Centre for Computational Continuum Mechanics, Ljubljana.

[7] I. Grešovnik. "*Download page for Inverse*", electronic document at http://www.c3m.si/inverse/download/ .

[8] I. Grešovnik. "*Inverse manuals*", electronic document at http://www.c3m.si/inverse/doc/man/ , maintained by the Centre for Computational Continuum Mechanics, Ljubljana.

[9] I. Grešovnik. "*Quick introduction to optimization shell Inverse*", electronic document at http://www.c3m.si/inverse/doc/other/index.html .

[10] I. Grešovnik. "*A General Purpose Computational Shell for Solving Inverse and Optimisation Problems - Applications to Metal Forming Processes*", Ph.D. thesis, available at http://www.c3m.si/inverse/doc/phd/index.html .

[11] I. Grešovnik. "*IOptLib*", http://www.c3m.si/igor/ioptlib/index.html .

[12] T. Šuštar, *FEM Driver*, electronic document at http://www.c3m.si/driver/ , Ljubljana, 2000.

[13] J. Korelc. Symbolic methods in numerical analysis. Electronic document at http://www.fgg.uni-lj.si/Symech/ .