# Inverse identification of parameters of numerical models

PROFORM project - internal report

Igor Grešovnik
February 2005

**1 EXPERIMENTAL DETERMINATION OF MODEL PARAMETERS ..... 2**

1.1 IMPORTANCE OF PARAMETER IDENTIFICATION ..... 2
1.2 INVERSE IDENTIFICATION OF MODEL PARAMETERS ..... 3

**2 NUMERICAL OPTIMISATION TECHNIQUES ..... 5**

2.1 INTRODUCTION ..... 5
2.2 HEURISTIC MINIMISATION METHODS AND RELATED PRACTICAL PROBLEMS ..... 6
2.3 SIMPLEX METHOD ..... 10
2.4 BASIC MATHEMATICAL BACKGROUND ..... 15
*2.4.1 Basic Notions ..... 17*
*2.4.2 Conditions for Unconstrained Local Minima ..... 21*
*2.4.3 Desirable Properties of Algorithms and Notion of Conjugacy ..... 22*
2.5 NEWTON-LIKE METHODS ..... 27
*2.5.1 Quasi-Newton Methods ..... 29*
*2.5.2 Invariance Properties ..... 34*
2.6 CONJUGATE DIRECTION METHODS ..... 37
*2.6.1 Conjugate Gradient Methods ..... 38*
2.7 FURTHER REMARKS ..... 42

# 1 EXPERIMENTAL DETERMINATION OF MODEL PARAMETERS

## *1.1 Importance of parameter identification*

When we want to use numerical simulation as a decision support tool for analysis and optimization, the produced results must be accurate and reliable enough. In order to satisfy this basic requirement, we must possess a physical model that adequately describes the phenomena in question and numerical tools capable of reproducing approximations that are in good agreement with physical models.

Beside the laws that are regarded basic physical principles, such physical models can include simplified description of complex systems that can be derived from more basic principles, or are just assumed on the basis of experiments. An example of this are basic principles of thermodynamics, which were confirmed experimentally long before their validity could be anticipated by statistical thermodynamics, which starts from somehow more fundamental description as macroscopic models do. Although statistical thermodynamics can state macroscopic relations only in terms of averaging over microscopic states, for systems with large degrees of freedom deviations are small enough that we can consider continuous models valid in most practical situations.

In many practical situations we are forced to bridge large gaps between fundamental principles and physical models that are useful for simulation. As an example, the ideal gas equation can be derived by treating gas molecules as colliding rigid bodies whose radius is much smaller than average free path, and it is accurate enough in given situations. When it is not, it may be extremely difficult to derive a single point of the state equation, even if the dependence of two- and multi-molecular potential could be exactly calculated. In this case, fundamental assumptions on sole existence of state equation together with some regularity

assumptions can be supplemented by experimental data in order to build a physical model used for, say, simulation of a jet engine stage.

The remaining task is to experimentally determine enough points on the state surface in order to build its accurate enough interpolation. A problem arises when some fundamental components of the physical model can not be measured directly, as can be the case with viscosity. Most typically these components are related to material properties, but in simplified models they can also include boundary conditions (e.g. heat flux through the engine wall) or unknown but constant influence from system neighbourhood. These models must be established from experiments by using indirect techniques.

## *1.2 Inverse identification of model parameters*

Indirect techniques involves inference of model parameters on the basis of measuring something else in a controlled experiments. To make this possible, we must first insure that measurement outcome uniquely depends on parameters to be determined, i.e.

$$\mathbf{y}^{(m)} = \mathbf{f}(\mathbf{a}), \tag{1.1}$$

$\mathbf{y}^{(m)}$ being measurements and **a** unknown parameters of the model. This dependence is calculated by numerical simulation of the actual experiment that incorporates the physical model whose parameter we are trying to determine. Model parameters would in principle be obtained by solving the above equation, i.e. effectively applying inverse of **f** to measurement data (hence the name inverse techniques). Needless to say, this would require precise consistence of our model (including numerical calculation) with the physical reality and the ability to sample measured data exactly. None of this is true in practice and we can at most find estimation of parameters that is statistically the best according to measurements.

In order to accomplish this task, we define a measure of inconsistency of the model assuming given parameters with the experimental data. The estimate of model parameters is obtained by minimizing this discrepancy over all possible parameter values. Most often we define the discrepancy measure in the least square sense, i.e.

$$F(\mathbf{a}) = \chi^2(\mathbf{a}) = \sum_{i=1}^{m} \left[ \frac{y_i^{(m)} - y_i(\mathbf{a})}{\sigma_i} \right]^2, \tag{1.2}$$

thus

$$\mathbf{a}^* = \arg\min F(\mathbf{a}), \tag{1.3}$$

In (1.0, $y_i(\mathbf{a})$ represent the values of corresponding measurements $y_i$, calculated by using numerical model of the experiment assuming specific values for model parameters. Such a definition has a statistical background. If measurements are distributed normally with corresponding standard deviations $\sigma_i$ and the model exactly represent reality then $\mathbf{a}^*$ maximizes the likelihood that actual parameters are equal to $\mathbf{a}^*$. The distribution of values of $F$ is the chi-square distribution of order $\nu = m - n$ where $m$ is the number of measurements and $n$ the number of unknown parameters, with mean value $\nu$ and standard deviation $\sqrt{2\nu}$. This fact can serve statistical validation of the model itself by repeating the experiment. In practice we have to deal with imperfect models, and providing additional degrees of freedom in model vector **a** can provide means of fitting the model to observations in lack of physical arguments. This must be undertaken with extreme caution because everything can be fitted by sufficiently loose model, but such a model looses the ability of prediction and has no sense. To avoid this, system (1.0 must be sufficiently over-determined by capturing enough independent empirical information.

Simulated measurements $y_i(\mathbf{a})$ are defined implicitly through solution of model equations with a model ultimately defined by **a**. A software architecture that will enable the solution of the parameter identification problems defined e.g. as (1.0 is proposed in[1]-[2]. The computational shell *Inverse*[3] is constructed to enable incorporation of simulation environment in such a scheme. Beside a good numerical model, reliable algorithms for solving the resulting minimization problems are significant for successful practical application. Basics of such algorithms are outlined in the following sections.

# 2 NUMERICAL OPTIMISATION TECHNIQUES

## *2.1 Introduction*

In general, optimisation problems can be stated as problems of minimisation of some function of the design parameters **x**, subjected to certain constraints, i.e.:

$$\begin{aligned} &\textit{minimise} && f(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^n \\ &\textit{subject to} && c_i(\mathbf{x}) = 0, \ i \in E \\ &\textit{and} && c_j(\mathbf{x}) \geq 0, \ j \in I, \end{aligned} \tag{2.1}$$

where $f(\mathbf{x})$ is the objective function and $c_i(\mathbf{x})$ and $c_j(\mathbf{x})$ are constraint functions[1]. Design parameters are also referred to as optimisation variables. The second line of (2.0 represents the equality constraints of the problem and the third line represents the inequality constraints. We have introduced two index sets, set $E$ of the equality constraint indices and set $I$ of the inequality constraint indices. The above problem is also referred to as the general nonlinear problem. Most of optimisation problems can be expressed in this form, eventually having multiple objective functions in the case of several conflicting design objectives.

Points **x'**, which satisfy all constraints, are called feasible points and the set of all such points is called the feasible region. A point $\mathbf{x}^*$ is called a constrained local minimiser (or local solution of the above problem) if there exists some neighbourhood $\Omega$ of $\mathbf{x}^*$ such that $f(\mathbf{x}^*) \leq f(\mathbf{x}')$ for all feasible points $\mathbf{x}' \in \Omega, \mathbf{x}' \neq \mathbf{x}^*$. Such a point is called a strict local minimiser if the < sign is applied in place of $\leq$; a slightly stronger definition of isolated local minimiser, which requires the minimiser to be the only local minimiser in some neighbourhood. Furthermore, **x*** is called the global solution or global constrained minimiser if $f(\mathbf{x}^*) \leq f(\mathbf{x}')$ for all feasible points **x'**. This means that a global minimiser is the local solution with the least value of *f*.

Since the objective and constraint functions are in general nonlinear, the optimisation problem can have several constrained local minimisers $\mathbf{x}^*$. The goal of

[1] Number of optimisation variables will be denoted by *n* throughout chapter 2.

optimisation is of course to comply with the objective as much as possible, therefore the identification of the global solution is the most desirable. However, this problem is in general extremely difficult to handle. Actually there is no general way to prove that some point is a global minimiser. At best some algorithms are able to locate several local solutions and one can then take the best one of these. These methods are mostly based on some stochastic search strategy. Location of problem solutions is of a statistical nature, which inevitably leads to an enormous number of function evaluations needed to locate individual solutions with satisfactory accuracy and certainty. These methods are therefore usually not feasible for use with costly numerical simulations and are not included in the scope of this work. Currently the most popular types of algorithms for identifying multiple local solutions are the simulated annealing algorithms and genetic algorithms, briefly described in [12].

## *2.2 Heuristic Minimisation Methods and Related Practical Problems*

In the subsequent text the unconstrained problem is considered, namely

$$\textit{minimise} \qquad f(\mathbf{x}), \qquad \mathbf{x} \in \mathbb{R}^n \tag{2.2}$$

Throughout this chapter it is assumed that $f$ is at least a $\mathbb{C}^2$ function, i.e. twice continuously differentiable with respect to $\mathbf{x}$. Every local minimum is a stationary point of $f$, i.e. a point with zero gradient[4]:

$$\nabla f(\mathbf{x}^*) = \mathbf{g}(\mathbf{x}^*) = \mathbf{g}^* = 0 . \tag{2.3}$$

Minimisation can therefore be considered as a solution of the above equation, which is essentially a system of nonlinear equations for gradient components

$$g_i(\mathbf{x}) = \frac{\partial f(\mathbf{x})}{\partial x_i} = 0 , \; i = 1,\ldots n . \tag{2.4}$$

This is essentially the same system that arises in finite element simulation[37] and can be solved by the standard Newton method, for which the iteration is

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \left(\nabla \mathbf{g}^{(k)}\right)^{-1} \mathbf{x}^{(k)} . \tag{2.5}$$

The notation $\mathbf{g}^{(k)} = \mathbf{g}\left(\mathbf{x}^{(k)}\right)$ is adopted throughout this work.

The method is derived from the Taylor series[33],[35] for **g** about the current estimate $\mathbf{x}^{(k)}$:

$$\mathbf{g}\left(\mathbf{x}^{(k)} + \delta\right) = \mathbf{g}^{(k)} + \nabla\mathbf{g}^{(k)}\delta + O\left(\|\delta\|^2\right) \tag{2.6}$$

Considering this as the first order approximation for **g** and equating it to zero we obtain the expression for step $\delta$ which should bring the next estimate close to the solution of (2.2)[1]:

$$\nabla\mathbf{g}^{(k)}\delta = -\mathbf{g}^{(k)}.$$

By setting $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \delta$ we obtain the above Newton Iteration.

The Newton method is known to be rapidly convergent[5], but suffers for a lack of global convergence properties, i.e. the iteration converges to the solution only in some limited neighbourhood, but not from any starting point. This is the fundamental reason that it is usually not applicable to optimisation without modifications. The problem can usually be elegantly avoided in simulations, either because of some nice physical properties of the analysed system that guarantee global convergence, or by the ability of making the starting guess arbitrarily close to the equilibrium point where the equations are satisfied. This is, for example, exploited in the solution of path dependent problems where the starting guess of the current iterate is the equilibrium of the previous, and this can be set arbitrarily close to the solution because of the continuous nature of the governing equations. Global convergence can be ensured simply by cutting down the step size, if necessary.

In practice, this is usually not at all case in optimisation. The choice of a good starting point typically depends only on a subjective judgment where the solution should be, and the knowledge used for this is usually not sufficient to choose the starting point within the convergence radius of Newton's method, especially due to the complex non-linear behaviour of *f* and consequently **g**. Modifications to the method must therefore be made in order to induce global convergence[2], i.e. convergence from any starting guess.

[1] Notation $\mathbf{g}(\mathbf{x}) = \nabla f(\mathbf{x})$, $f^{(k)} = f\left(\mathbf{x}^{(k)}\right)$, $\mathbf{g}^{(k)} = \mathbf{g}\left(\mathbf{x}^{(k)}\right)$, etc. will be generally adopted throughout this text.

[2] Herein the expression global convergence is used to denote convergence to a local solution from any given starting point. In some of the literature this expression is used to denote convergence to a global solution.

One such modification arises from considering what properties the method must have in order to induce convergence to the solution. The solution $\mathbf{x}^*$ must be a limiting point of the sequence of iterations. This means that the distance between the iterates and the solution tends towards zero, i.e.

$$\lim_{k\to\infty} \left\| \mathbf{x}_k - \mathbf{x}^* \right\| = 0 . \tag{2.7}$$

This is satisfied if the above norm is monotonically decreasing and if the sequence has no accumulation point other than $\mathbf{x}^*$. When considering the minimisation problem and assuming that the problem has a unique solution, the requirements for a decreasing norm can be replaced (because of continuity of *f*) by the requirement that $f^{(k)}$ are monotonically decreasing. By such consideration, a basic property any minimisation algorithm should have, is the generation of descent iterates so that

$$f^{(k+1)} < f^{(k)} \quad \forall k . \tag{2.8}$$

This is closely related to the idea of line search, which is one of the elementary ideas in construction of minimisation algorithms. The idea is to minimise *f* along some straight line starting from the current iterate. Many algorithms are centered on this idea, trying to generate a sequence of directions along which line searches are performed, such that a substantial reduction of *f* is achieved in each line search and such that, in the limit, the rapid convergence properties of Newton's method are inherited.

An additional complication which limits the applicability of Newton's method is that the second derivatives of the objective function (i.e. first derivatives of its gradient) are required. These are not always directly available since double differentiation of numerical models is usually a much harder problem than single differentiation. Alternatively the derivatives can be obtained by straight numerical differentiation using small perturbation of parameters, but in many cases this is not applicable because numerical differentiation is very sensitive to errors in function evaluation[34],[36], and these can often not be avoided sufficiently when numerical models with many degrees of freedom are used. Furthermore, even if the Newton method converges, the limiting point is only guaranteed to be a stationary point of *f*, but this is not a sufficient condition for a local minimum, since it includes saddle points, which are stationary points but are not local minimisers.

The most simple algorithm that incorporates the idea of line search is sequential minimisation of the objective function in some fixed set of *n* independent directions in each iterate, most elementarily parallel to the coordinate axes. The requirement for *n* independent directions is obvious since otherwise the algorithm could not reach any point in $\mathbb{R}^n$. The method is called the alternating variables method and it seems to be adequate at a first glance, but turns to be very inefficient

and unreliable in practice. A simple illustration of the reasons for this is that the algorithm ignores the possibility of correlation between the variables. This causes the search parallel to the current search direction to destroy completely the property that the current point is the minimiser in previously used directions. This leads to oscillatory behaviour of the algorithm as illustrated in Figure 2.1.

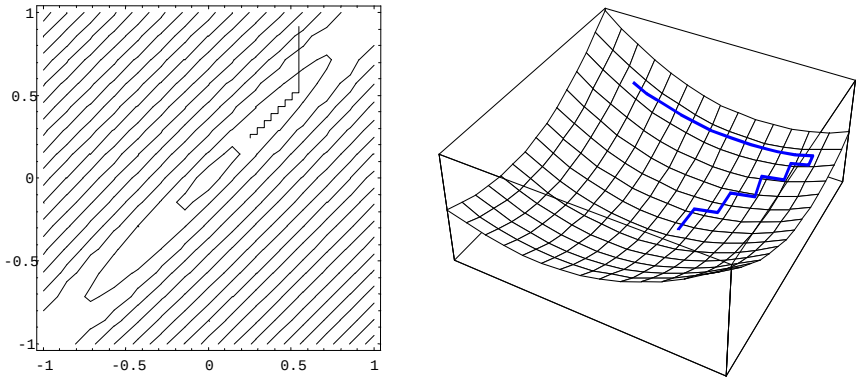


**Figure 2.1:** Oscillatory behaviour, which is likely to occur when using sequential minimisation in a fixed set of directions.

Another readily available algorithm is sequential minimisation along the current direction of the gradient of *f*. Again this seems to be a good choice, since the gradient is the direction of the steepest descent, i.e. the direction in which *f* decreases most rapidly in the vicinity of the starting point. With respect to this, the method is called the steepest descent method. In practice, however, the method suffers for similar problems to the alternating variables method, and the oscillating behaviour of this method is illustrated in Figure 2.2. The theoretical proof of convergence exists, but it can also be shown that locally the method can achieve an arbitrarily slow rate of linear convergence[4].

The above discussion clearly indicates the necessity for a more rigorous mathematical treatment of algorithms. Indeed the majority of the up-to-date algorithms have a solid mathematical background[4]-[10], [29] and partially the aim of this section is to point which are the most important features in the design of fast and reliable algorithms.

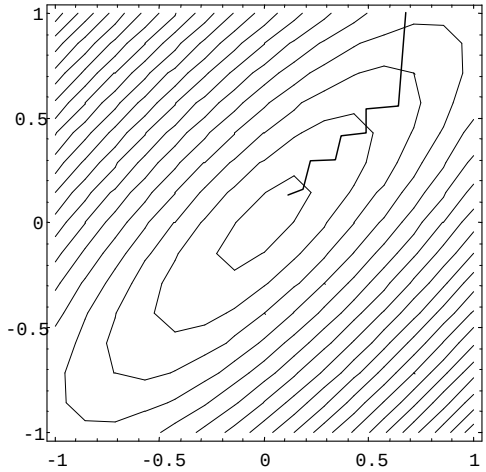


**Figure 2.2:** Oscillatory behaviour, which can occur when performing sequential line searches along the steepest descent directions.

## *2.3 Simplex Method*

One minimisation method that does not belong within the context of the subsequent text is the simplex method[15], [29],[4]. It has been known since the early sixties and could be classed as another heuristic method since it is not based on a substantial theoretical background.

The simplex method neither uses line searches nor is based on minimisation of some simplified model of the objective function, and therefore belongs to the class of direct search methods. Because of this the method does not compare well with other described methods with respect to local convergence properties. On the other hand, for the same reason it has some other strong features. The method is relatively insensitive to numerical noise and does not depend on some other properties of the objective function (e.g. convexity) since no specific continuity or other assumptions are incorporated in its design. It merely requires the evaluation of function values. Its performance in practice can be as satisfactory as any other non-derivative method, especially when high accuracy of the solution is not required and the local convergence properties of more sophisticated methods do not play so important role. In many cases it does not make sense to require highly accurate solutions of optimisation problems, because the obtained results are inevitably inaccurate with respect to real system behaviour due to numerical modeling of the system (e.g.

discretisation and round-off errors or inaccurate physical models). These are definitely good arguments for considering practical use of the method in spite of the lack of good local convergence results with respect to some other methods.

The simplex method is based on construction of an evolving pattern of *n+1* points in $\mathbb{R}^n$ (vertices of a simplex). The points are systematically moved according to some strategy such that they tend towards the function minimum. Different strategies give rise to different variants of the algorithm. The most commonly used is the Nelder-Mead algorithm described below. The algorithm begins by choice of *n+1* vertices of the initial simplex $(\mathbf{x}_1^{(1)}, \ldots, x_{n+1}^{(1)})$ so that it has non-zero volume. This means that all vectors connecting a chosen vertex to the reminding vertices must be linearly independent, e.g.

$$\exists \lambda_i \neq 0 \Rightarrow \sum_{i=1}^{n} \lambda_i \left( \mathbf{x}_{i+1}^{(1)} - \mathbf{x}_1^{(1)} \right) \neq 0 .$$

If we have chosen $\mathbf{x}_1^{(1)}$, we can for example obtain other vertices by moving, for some distance, along all coordinate directions. If it is possible to predict several points that should be good according to experience, it might be better to set vertices to these points, but the condition regarding independence must then be checked.

Once the initial simplex is constructed, the function is evaluated at its vertices. Then one or more points of the simplex are moved in each iteration, so that each subsequent simplex consists of a better set of points:

**Algorithm 2.1:** The Nelder-Mead simplex method.

After the initial simplex is chosen, function values in its vertices are evaluated: $f_i^{(1)} = f\left(\mathbf{x}_i^{(1)}\right), i = 1, \ldots, n+1$.

Iteration *k* is then as follows:

1. **Ordering step:** Simplex vertices are first reordered so that $f_1^{(k)} \leq f_2^{(k)} \leq \ldots \leq f_{n+1}^{(k)}$, where $f_i^{(k)} = f\left(\mathbf{x}_i^{(k)}\right)$.
2. **Reflection step:** The worst vertex is reflected over the centre point of the best *n* vertices ($\overline{\mathbf{x}}^{(k)} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i^{(k)}$), so that the reflected point $\mathbf{x}_r^{(k)}$ is

$$\mathbf{x}_r^{(k)} = \overline{\mathbf{x}}^{(k)} + \left( \overline{\mathbf{x}}^{(k)} - \mathbf{x}_{n+1}^{(k)} \right)$$

Evaluate $f_r^{(k)} = f\left(\mathbf{x}_r^{(k)}\right)$. If $f_1^{(k)} \leq f_r^{(k)} < f_n^{(r)}$, accept the reflected point and go to 6.

3. **Expansion step:** If $f_r^{(k)} < f_1^{(k)}$, calculate the expansion

$$\mathbf{x}_e^{(k)} = \overline{\mathbf{x}}^{(k)} + 2\left(\mathbf{x}_r^{(k)} - \overline{\mathbf{x}}^{(k)}\right)$$

and evaluate $f_e^{(k)} = f\left(\mathbf{x}_e^{(k)}\right)$. If $f_e^{(k)} < f_r^{(k)}$, accept $\mathbf{x}_e^{(k)}$ and go to 6. Otherwise accept $\mathbf{x}_r^{(k)}$ and go to 6.

4. **Contraction step:** If $f_r^{(k)} \geq f_n^{(k)}$, perform contraction between $\overline{\mathbf{x}}^{(k)}$ and the better of $\mathbf{x}_{n+1}^{(k)}$ and $\mathbf{x}_r^{(k)}$. If $f_r^{(k)} < f_{n+1}^{(k)}$, set

$$\mathbf{x}_c^{(k)} = \overline{\mathbf{x}}^{(k)} + \frac{1}{2}\left(\mathbf{x}_r^{(k)} - \overline{\mathbf{x}}^{(k)}\right)$$

(this is called the outside contraction) and evaluate $f_c^{(k)} = f\left(\mathbf{x}_c^{(k)}\right)$. If $f_c^{(k)} \leq f_r^{(k)}$, accept $\mathbf{x}_c^{(k)}$ and go to 6.
If in contrary $f_r^{(k)} \geq f_{n+1}^{(k)}$, set

$$\mathbf{x}_c^{(k)} = \overline{\mathbf{x}}^{(k)} - \frac{1}{2}\left(\overline{\mathbf{x}}^{(k)} - \mathbf{x}_{n+1}^{(k)}\right)$$

(inside contraction) and evaluate $f_c^{(k)}$. If $f_c^{(k)} < f_{n+1}^{(k)}$, accept $\mathbf{x}_c^{(k)}$ and go to 6.

5. **Shrink step:** Move all vertices except the best towards the best vertex, i.e.

$$\mathbf{v}_i^{(k)} = \mathbf{x}_1^{(k)} + \frac{1}{2}\left(\mathbf{x}_i^{(k)} - \mathbf{x}_1^{(k)}\right), i = 2, ..., n+1,$$

and evaluate $f_i^{'(k)} = f\left(\mathbf{v}_i^{(k)}\right), i = 2, ..., n+1$. Accept $\mathbf{v}_i^{(k)}$ as new vertices.

6. **Convergence check:** Check if the convergence criterion is satisfied. If so, terminate the algorithm, otherwise start the next iteration.

Figure 2.3 illustrates possible steps of the algorithm. A possible situation of two iterations when the algorithm is applied is shown in Figure 2.4. The steps allow the shape of the simplex to be changed in every iteration, so the simplex can adapt to the surface of *f*. Far from the minimum the expansion step allows the simplex to move rapidly in the descent direction. When the minimum is inside the simplex, contraction and shrink steps allow vertices to be moved closer to it.

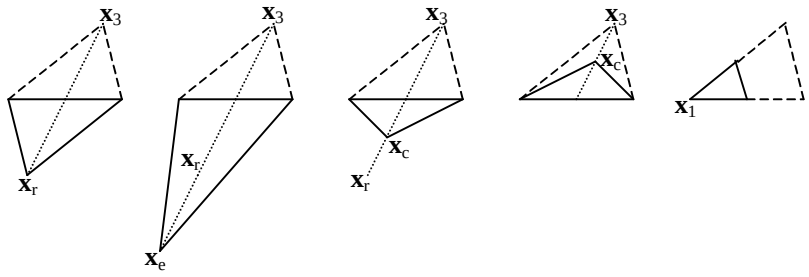


**Figure 2.3:** Possible steps of the simplex algorithm in two dimensions (from left to right): reflection, expansion, outside and inside contraction, and shrink.

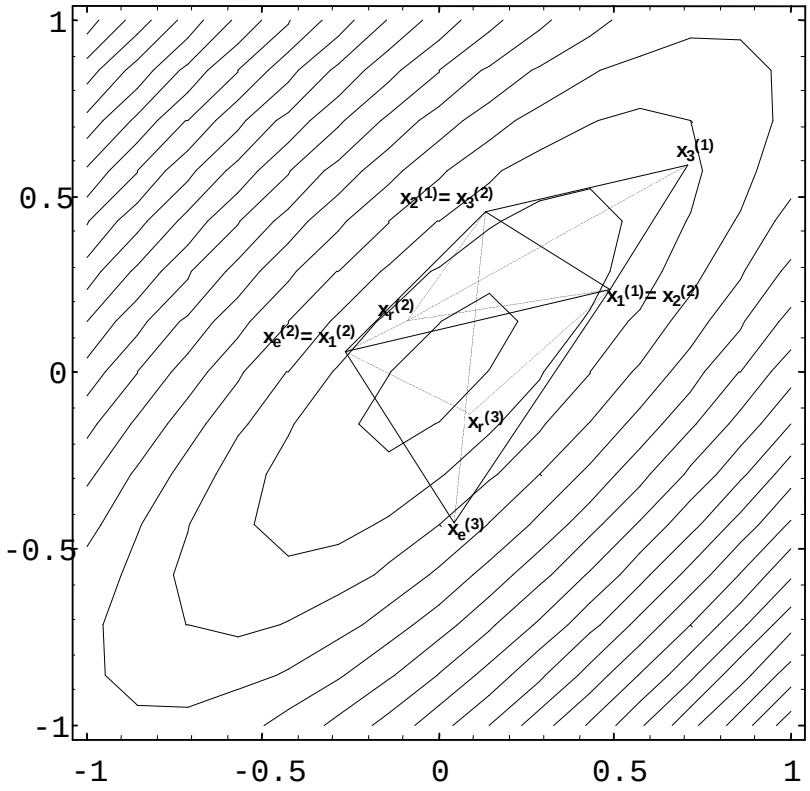


**Figure 2.4:** Example of evolution of the simplex.

There are basically two possibilities for the convergence criterion. Either that function values at vertices must become close enough or the simplex must becomes small enough. It is usually best to impose both criteria, because either of them alone can be misleading.

It must be mentioned that convergence to a local minimum has not been proved for the Nelder-Mead algorithm. Examples have been constructed for which the method does not converge[15]. However, the situations for which this was shown are quite special and unlikely to occur in practice. Another theoretical argument against the algorithm is that it can fail because the simplex collapses into a subspace, so that vectors connecting its vertices become nearly linearly dependent. Investigation of this phenomenon indicates that such behaviour is related to cases when the function to be minimised has highly elongated contours (i.e. ill conditioned Hessian). This is also a problematic situation for other algorithms.

The Nelder-Mead algorithm can be easily adapted for constrained optimisation. One possibility is to add a special penalty term to the objective function, e.g.

$$f'(\mathbf{x}) = f(\mathbf{x}) + f_{n+1}^{(1)} - \sum_{i \in I} c_i(\mathbf{x}) + \sum_{i \in I} \left| c_j(\mathbf{x}) \right|, \tag{2.9}$$

where $f_{n+1}^{(1)}$ is the highest value of $f$ in the vertices of the initial simplex. Since subsequent iterates generate simplices with lower values of the function at vertices, the presence of this term guarantees that whenever a trial point in some iteration violates any constraints, its value is greater than the currently best vertex. The last two sums give a bias towards the feasible region when all vertices are infeasible. The derivative discontinuity of the terms with absolute value should not be problematic since the method is not based on any model, but merely on comparison of function values. A practical implementation is similar to the original algorithm. $f$ is first evaluated at the vertices of the initial simplex and the highest value is stored. Then the additional terms in (2.0 are added to these values, and in subsequent iterates $f$ is replaced by $f'$.

Another variant of the simplex method is the multidirectional search algorithm. Its iteration consists of similar steps to the Nelder-Mead algorithm, except that all vertices but the best one are involved in all operations. There is no shrink step and the contraction step is identical to the shrink step of the Nelder-Mead algorithm. Possible steps are shown in Figure 2.5. The convergence proof exists for this method[15], but in practice it performs much worse than the Nelder-Mead algorithm. This is due to the fact that more function evaluations are performed at each iteration and that the simplex can not be adapted to the local function properties as well as the former algorithm. The shape of the simplex can not change, i.e. angles between it edges remain constant (see Figure 2.5). The multidirectional search algorithm is better suited to parallel processing because $n$ function evaluations can always be performed simultaneously.

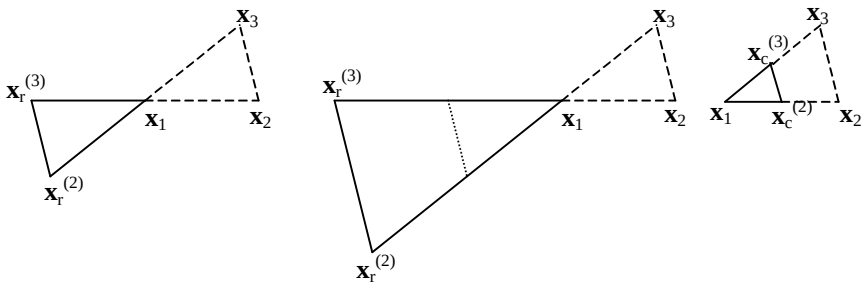


**Figure 2.5:** possible steps in the multidirectional search algorithm: reflection, expansion, and contraction.

## *2.4 Basic Mathematical Background*

Construction of optimisation methods described further in this section is based on some model of the objective function and constraints. Such treatment of the problem arises to a large extent from the fact that locally every function can be developed into a Taylor series[33] about any point $x'$:

$$f\left(x'+h\right)=\sum_{n=0}^{\infty}\frac{h^n}{n!}f^{(n)}\left(x'\right), \tag{2.10}$$

where $f^{(n)}(x)=\frac{\partial^n}{\partial x^n}f(x)$ and $n!=1\cdot 2\cdot 3\cdot ...\cdot n$. This expression itself does not have a significant practical value. A more important fact is that

$$\lim_{n\to\infty}R_n(h)=0 \tag{2.11}$$

and

$$\lim_{h\to 0}R_n(h)=0, \tag{2.12}$$

where

$$R_n(h) = f(x' + h) - S_n(h) \tag{2.13}$$

and

$$S_n(h) = \sum_{i=0}^{n} \frac{h^n}{n!} f^{(n)}(x'). \tag{2.14}$$

This means that if we use only a few terms in the Taylor series, the error that we make tends to zero both when we increase the number of terms without limit for some fixed *h*, and when we take a fixed number of terms and decrease the step *h* towards zero. This follows from the result[33]

$$R_n(h) = \frac{h^{n+1}}{(n+1)!} f^{(n+1)}(x' + \theta h), 0 < \theta < 1. \tag{2.15}$$

The above equation also holds if function *f* is only $\mathbb{C}^{n+1}$. This means that every sufficiently smooth function can be locally approximated by a simple polynomial function, which is sometimes more convenient for theoretical treatment than the original function.

A similar development is possible for a function of *n* variables[33]:

$$\begin{aligned} &f(x_1' + h_i, x_2' + h_2, ..., x_n' + h_n) = f(x_1', x_2', ..., x_n') + \\ &\sum_{i=1}^{m} \frac{1}{i!} \left( h_1 \frac{\partial}{\partial x_1} + h_2 \frac{\partial}{\partial x_2} + ... + h_n \frac{\partial}{\partial x_n} \right)^i f(x_1, x_2, ..., x_n) + , \\ &R_m(h_i, h_2, ..., h_n) \end{aligned} \tag{2.16}$$

where

$$\begin{aligned} &R_m(h_1, ..., h_n) = \frac{1}{(n+1)!} \left( h_1 \frac{\partial}{\partial x_1} + ... + h_n \frac{\partial}{\partial x_n} \right)^{m+1} \\ &f(x_1 + \theta_1 h_1, ..., x_n + \theta_n h_n), \quad 0 < \theta_i < 1, \ i = 1, ..., n \end{aligned} \quad . \tag{2.17}$$

In view of the beginning of this discussion, we can consider numerical optimisation as the estimation of a good approximation of the optimisation problem solution on the basis of limited information about the function, usually objective and constraint function values and their derivatives in some discrete set of points. The goal is to achieve satisfactory estimation with as little function and derivative

evaluations as possible. Now we can use the fact that general functions can be locally approximated by simpler functions. Besides, functions of simple and known form (e.g. linear or quadratic) are completely described by a finite number of parameters. If we know these parameters, we know (in principle) all about the function, including minimising points.

There exists a clear correspondence between the above considerations and the design of optimisation algorithms. One thing to look at when constructing algorithms is how they perform on simple model functions, and proofs of local convergence properties based to a large extent on properties of the algorithms when applied to such functions[4]-[10].

Heuristically this can be explained by considering a construction of a minimisation algorithm in the following way. Use function values and derivatives in a set of points to build a simple approximation model (e.g. quadratic), which will be updated when new information is obtained. Consider applying an effective minimisation technique adequate for the model function. Since the model approximates the function locally, some information obtained in this way should be applicable to making decision where to set the next iterate when minimising the original function. In the limit, when the iterates approach the minimum, the model function should be increasingly better approximation and minima of the successively built models should be good guesses for the subsequent iterates.

In fact many algorithms perform in a similar manner. The difference is usually that models are not built directly, but the iterates are rather constructed in such a way that the algorithm has certain properties when applied to simple functions, e.g. termination in a finite number of steps. This ensures good local convergence properties. In addition some strategy must be incorporated which ensures global convergence properties of the algorithm. The remainder of this section will consider some mathematical concepts related to this. First, some basic notions will be introduced, and then some important algorithmic properties will be discussed.

### 2.4.1 Basic Notions

Quadratic model functions are the most important in the study of unconstrained minimisation. This is because the Taylor series up to quadratic terms is the simplest Taylor approximation that can have an unconstrained local minimum. Keeping the terms up to the second order in (2.0 gives the following expression for a second order Taylor approximation:

$$f\left(\mathbf{x}^{'}+\mathbf{h}\right)\approx f\left(\mathbf{x}^{'}\right)+\mathbf{h}^{T}\nabla f\left(\mathbf{x}^{'}\right)+\frac{1}{2}\mathbf{h}^{T}\left[\nabla^{2} f\left(\mathbf{x}^{'}\right)\right]\mathbf{h}\,, \tag{2.18}$$

where

$$\nabla f(\mathbf{x}) = \mathbf{g}(\mathbf{x}) = \left[ \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_n} \right]^T_{\mathbf{x}}$$

is the function gradient and

$$\nabla^2 f(\mathbf{x}) = \mathbf{G}(\mathbf{x}) = \left(\nabla \nabla^T\right) f(\mathbf{x})$$

is the Hessian matrix[1] of the function, i.e. matrix of function second derivatives,

$$\left[\nabla^2 f(\mathbf{x})\right]_{ij} = \mathbf{G}_{ij}(\mathbf{x}) = \frac{\partial^2 f}{\partial x_i \partial x_j}(\mathbf{x}). \tag{2.19}$$

Notation $\mathbf{g}(\mathbf{x}) = \nabla f(\mathbf{x})$ and $\mathbf{G}(\mathbf{x}) = \nabla^2 f(\mathbf{x})$ will be used throughout this text.

The idea of a line in $\mathbb{R}^n$ is important. This is a set of points

$$\mathbf{x} = \mathbf{x}(\alpha) = \mathbf{x}' + \alpha \mathbf{s}, \tag{2.20}$$

where $\alpha \in \mathbb{R}$ is a scalar parameter, $\mathbf{x}'$ is any point on the line and **s** is the direction of the line. **s** can be normalised, e.g. with respect to the Euclidian norm, i.e. $\sum_{i=1}^{n} s_i^2 = 1$.

It is often useful to study how a function defined in $\mathbb{R}^n$ behaves on a line. For this purpose, we can write

$$f(\alpha) = f(\mathbf{x}(\alpha)) = f(\mathbf{x}' + \alpha \mathbf{s}). \tag{2.21}$$

From this expression we can derive direction derivative of *f*, i.e. derivative of the function along the line:

[1] In standard notation Operator $\nabla^2 = \Delta = \nabla^T \nabla = \sum_{i=1}^{n} \frac{\partial^2}{\partial x_i^2}$ is the Laplace operator. However, in most optimisation literature this notation is used for the Hessian operator, and so is also used in this text.

$$\frac{df(\alpha)}{d\alpha} = \sum_i \frac{dx_i}{d\alpha}\frac{\partial f}{\partial x_i} = \sum_i s_i \frac{\partial f}{\partial x_i} = (\nabla f(\mathbf{x}(\alpha)))^T \mathbf{s}\,.$$

This can be written as

$$\frac{df}{d\alpha} = \frac{df}{d\mathbf{s}} = \nabla f^T \mathbf{s}\,. \tag{2.22}$$

In a similar way the curvature along the line is obtained:

$$\frac{d^2 f(\alpha)}{d\alpha^2} = \frac{d}{d\alpha}\frac{df}{d\alpha} = \frac{d}{d\alpha}\sum_{i=1}^{n} s_i \frac{\partial f}{\partial x_i} =$$
$$\sum_{i=1}^{n} s_i \sum_{j=1}^{n} \frac{dx_j}{d\alpha}\frac{\partial^2 f}{\partial x_i \partial x_j} = \sum_{i=1}^{n}\sum_{j=1}^{n} s_i s_j \frac{\partial^2 f}{\partial x_i \partial x_j}$$

and so

$$\frac{d^2 f}{d\alpha^2} = \frac{d^2 f}{d\mathbf{s}^2} = \mathbf{s}^T (\nabla^2 f)\mathbf{s}\,. \tag{2.23}$$

A general quadratic function can be written in the form

$$q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \mathbf{G}\mathbf{x} + \mathbf{b}^T \mathbf{x} + c\,, \tag{2.24}$$

where $\mathbf{G}$ is a symmetric constant matrix, $\mathbf{b}^T$ a constant vector and $c$ a constant scalar. The gradient of this function is

$$\nabla q(\mathbf{x}) = \mathbf{G}\mathbf{x} + \mathbf{b} \tag{2.25}$$

and the Hessian matrix is

$$\nabla^2 q(\mathbf{x}) = \mathbf{G}\,, \tag{2.26}$$

where the rule for gradient of a vector product

$$\nabla(\mathbf{u}^T \mathbf{v}) = (\nabla \mathbf{u}^T)\mathbf{v} + (\nabla \mathbf{v}^T)\mathbf{u};\ \ \mathbf{u} = \mathbf{u}(\mathbf{x}),\ \mathbf{v} = \mathbf{v}(\mathbf{x})$$

was applied.

We see that a quadratic function has a constant Hessian and its gradient is an affine function of **x**. As a consequence, for any two points the following equation relating the gradient in these points is valid:

$$\nabla q(\mathbf{x}^{''}) - \nabla q(\mathbf{x}^{'}) = \mathbf{G}(\mathbf{x}^{''} - \mathbf{x}^{'}). \tag{2.27}$$

If **G** is nonsingular, a quadratic function has a unique stationary point ($\nabla q(\mathbf{x}^{'}) = 0$):

$$\mathbf{x}^{'} = -\mathbf{G}^{-1}\mathbf{b}, \tag{2.28}$$

which is also a minimiser if **G** is positive definite (see section 2.4.2). Taylor development about the stationary point gives another form for a quadratic function

$$q(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{x}^{'})^{T}\,\mathbf{G}\,(\mathbf{x} - \mathbf{x}^{'}) + c^{'}, \tag{2.29}$$

where $c^{'} = c - \frac{1}{2}\mathbf{x}^{'T}\mathbf{G}\mathbf{x}^{'}$.

In this text a term linear function[1] will be used for functions of the form

$$l(\mathbf{x}) = \mathbf{a}^{T}\mathbf{x} + b, \tag{2.30}$$

where $\mathbf{a}^{\mathrm{T}}$ is a constant vector and *b* a constant scalar. Such functions have a constant gradient

$$\nabla l(\mathbf{x}) = \mathbf{a} \tag{2.31}$$

and zero Hessian

$$\nabla^{2} l(\mathbf{x}) = 0. \tag{2.32}$$

[1] Mathematically this is an affine function. Linear functions are those[33] for which $f(a\mathbf{x} + b\mathbf{y}) = af(\mathbf{x}) + bf(\mathbf{y})$ for arbitrary **x** and **y** in the definition domain and for arbitrary constants *a* and *b*. Affine functions are those for which $f(\mathbf{x}) - c$ is a linear function, where *c* is some constant. However, in the optimisation literature affine functions are often referred to simply as linear and this is also adopted in this text.

### 2.4.2 Conditions for Unconstrained Local Minima

Consider first a line through some point $\mathbf{x}^*$, i.e. $\mathbf{x}(\alpha) = \mathbf{x}^* + \alpha\mathbf{s}$. Let us define a scalar function of parameter $\alpha$ using values of function $f$ on this line as $f(\alpha) = f(\mathbf{x}(\alpha))$. If $\mathbf{x}^*$ is a local minimiser of $f(\mathbf{x})$, then 0 is clearly a local minimiser of $f(\alpha)$. From the Taylor expansion for a function of one variable about 0 then it follows[4] that $f$ has zero slope and non-negative curvature at $\alpha = 0$. This must be true for any line through $\mathbf{x}^*$, and therefore for any **s**. From (2.0 and (2.0 it then follows

$$\mathbf{g}^* = 0 \tag{2.33}$$

and

$$\mathbf{s}^T\mathbf{G}^*\mathbf{s} \geq 0 \quad \forall\mathbf{s}, \tag{2.34}$$

where the following notation is used: $f^* = f(\mathbf{x}^*)$, $\mathbf{g}(\mathbf{x}) = \nabla f(\mathbf{x})$, $\mathbf{g}^* = \mathbf{g}(\mathbf{x}^*)$, $\mathbf{G}(\mathbf{x}) = \nabla^2 f(\mathbf{x})$, and $\mathbf{G}^* = \mathbf{G}(\mathbf{x}^*)$. This notation will be used through this text, and similarly $f(\mathbf{x}^{(k)}) = f^{(k)}$, etc.

Since (2.0 and (2.0 are implied by assumption that $\mathbf{x}^*$ is a local minimiser of *f*, these are necessary conditions for $\mathbf{x}^*$ being a local minimiser. (2.0 is referred to a first order necessary condition and (2.0 as a second order necessary condition. This condition states that the Hessian matrix is positive semi-definite in a local minimum.

The above necessary conditions are not at the same time sufficient, i.e. these conditions do not imply $\mathbf{x}^*$ to be a local minimiser. Sufficient conditions can be stated in the following way[4]:

**Theorem 2.1:**

*Sufficient conditions for a strict and isolated local minimiser* $\mathbf{x}^*$ *of f are that f has a zero gradient and a positive definite Hessian matrix in* $\mathbf{x}^*$*:*

$$\mathbf{g}^* = 0 \tag{2.35}$$

and

$$\mathbf{s}^T\mathbf{G}^*\mathbf{s} > 0 \quad \forall \mathbf{s} \neq 0 \tag{2.36}$$

There are various ways how to check the condition (2.0. The most important for practical purposes are that[30],[32] **G** is positive definite, the Choleski factors of the $LL^T$ decomposition exist and all diagonal elements $l_{ii}$ are greater than zero, and the same applies for diagonal elements $d_{ii}$ of the $LDL^T$ decomposition. This can be readily verified on those algorithms which solve a system of equation with the system matrix **G** in each iteration, since one of these decompositions is usually applied to solve the system.

Some algorithms do not evaluate the Hessian matrix. These can not verify the sufficient conditions directly. Sometimes these algorithms check only the first order condition or some condition based on the progress during the last few iterations. It can usually be proved that under certain assumptions iterates still converges to a local minimum. Algorithms should definitely have the possibility of termination in a stationary point, which is not a minimum (usually in a saddle point with indefinite Hessian matrix). Some algorithms generate subsequent approximations of the Hessian matrix, which converge to the Hessian in the limit when iterates approach a stationary point. The condition can then be checked indirectly on the approximate Hessian. More details concerning this will be outlined in the description of individual algorithms.

### 2.4.3 Desirable Properties of Algorithms and Notion of Conjugacy

A desired behaviour of an optimisation algorithm is that iterates move steadily towards the neighbourhood of a local minimser, then converge rapidly to this point and finally that it identifies when the minimiser is determined with a satisfactory accuracy and terminates.

Optimisation algorithms are usually based on some model and on some prototype algorithm. A model is some approximation (not necessarily explicit) of the objective function, which enables a prediction of a local minimiser to be made.

A prototype algorithm refers to the broad strategy of the algorithm. Two basic types are the restricted step approach and the line search approach, described in detail in the subsequent sections. There it will be also pointed out that the ideas of prototype algorithms are usually closely associated with global convergence.

Local convergence properties of an algorithm describe its performance in the neighbourhood of a minimum. If we define the error of the *k*-th iterate

$$\mathbf{h}^{(k)} = \mathbf{x}^{(k)} - \mathbf{x}^{*}, \tag{2.37}$$

it may be possible to state some limit results for $\mathbf{h}^{(k)}$. An algorithm is of course convergent if $\mathbf{h}^{(k)} \to 0$. If a limit

$$\lim_{k\to\infty} \frac{\left\|\mathbf{h}^{(k+1)}\right\|}{\left\|\mathbf{h}^{(k)}\right\|^{p}} = a \tag{2.38}$$

exists where $a > 0$ is some constant, then we say that the order of convergence is *p*. This definition can also be stated in terms of bounds if the limit does not exist: the order of convergence is *p* if

$$\frac{\left\|\mathbf{h}^{(k+1)}\right\|}{\left\|\mathbf{h}^{(k)}\right\|^{p}} \le a \tag{2.39}$$

for some constant $a > 0$ and for each *k* greater than some $k_{lim}$. An important cases are linear or first order convergence

$$\frac{\left\|\mathbf{h}^{(k+1)}\right\|}{\left\|\mathbf{h}^{(k)}\right\|} \le a \tag{2.40}$$

and quadratic or second order convergence

$$\frac{\left\|\mathbf{h}^{(k+1)}\right\|}{\left\|\mathbf{h}^{(k)}\right\|^{2}} \le a\,. \tag{2.41}$$

The constant *a* is called the rate of convergence and must be less than 1 for linear convergence. Linear convergence is only acceptable if the rate of convergence is small. If the order and rate are 1, the convergence is sublinear (slower than all linear convergence). This would be the case if $\left\|\mathbf{h}^{k}\right\| = 1/k$.

When the order is 1, but the rate constant is 0, the convergence is superlinear (faster than all linear convergence), i.e.

$$\lim_{k\to\infty} \frac{\left\|\mathbf{h}^{(k+1)}\right\|}{\left\|\mathbf{h}^{(k)}\right\|} = 0\,. \tag{2.42}$$

Successful methods for unconstrained minimisation converge superlinearly.

Many methods for unconstrained minimisation are derived from quadratic models. They are designed so that they work well or exactly on a quadratic function. This is partially associated with the discussion of section 2.4.1: since a general function is well approximated by a quadratic function, the quadratic model should imply good local convergence properties. Because the Taylor series about an arbitrary point taken to quadratic terms will agree to a given accuracy with the original function on a greater neighbourhood than the series taken to linear terms, it is preferable to use quadratic information even remote from the minimum.

The quadratic model is most directly used in the Newton method (2.3), which requires the second derivatives. A similar quadratic model is used in restricted step methods. When second derivatives are not available, they can be estimated in various ways. Such quadratic models are used in the quasi-Newton methods.

Newton-like methods (Newton or quasi-Newton) use the Hessian matrix or its approximation in Newton's iteration (2.3). A motivation for this lies in the Dennis-Moré theorem, which states that superlinear convergence can be obtained if and only if the step is asymptotically equal to that of the Newton-Raphson method[4].

The quadratic model is also used by the conjugate direction methods, but in a less direct way. Nonzero vectors $\mathbf{s}^{(1)}, \mathbf{s}^{(2)}, ..., \mathbf{s}^{(n)}$ are conjugate with respect to a positive definite matrix **G**, when

$$\mathbf{s}^{(i)^T}\mathbf{G}s^{(j)} = 0 \quad \forall i \neq j \,. \tag{2.43}$$

Optimisation methods, which generate such directions when applied to a quadratic function with Hessian **G**, are called conjugate direction methods. Such methods have the following important property[4]:

**Theorem 2.2:**

*A conjugate direction method terminates for a quadratic function in at most n exact line searches, and each* $\mathbf{x}^{(k)}$ *is a minimiser of that function in the set*

$$\left\{ \mathbf{x}; \mathbf{x} = \mathbf{x}^{(1)} + \sum_{j=1}^{k} \alpha_j \mathbf{s}^{(j)} \,, \alpha_j \in \mathbb{R} \right\} \tag{2.44}$$

The above theorem states that conjugate direction methods have the property of quadratic termination, i.e. they can locate the minimising point of a quadratic function in a known finite number of steps. Many good minimisation algorithms can generate the set of conjugate directions, although it is not possible to state that

superlinear convergence implies quadratic termination or vice versa. For example, some successful superlinearly convergent Newton-like methods do not possess this property.

It is useful to further develop the idea of conjugacy in order to gain a better insight in what it implies. We can easily see that $\mathbf{s}^{(i)}$ are linearly independent. If for example $\mathbf{s}^{(j)}$ was a linear combination of some other vectors $\mathbf{s}^{(k)}$, e.g.

$$\mathbf{s}^{(j)} = \sum_{k \neq j} \beta_k \mathbf{s}^{(k)},$$

multiplying this with $\mathbf{s}^{(j)T}\mathbf{G}$ would give

$$\mathbf{s}^{(j)T}\mathbf{G}\mathbf{s}^{(j)} = 0,$$

which contradicts the positive definiteness of **G**.

We can use vectors $\mathbf{s}^{(j)}$ as basis vectors and write any point as

$$\mathbf{x} = \mathbf{x}^{(1)} + \sum_{i=1}^{n} \alpha_i \mathbf{s}^{(i)}. \tag{2.45}$$

Taking into account this equation,(2.0 and conjugacy, the quadratic function from the theorem can be written as[1]

$$q(\alpha) = \frac{1}{2}\left(\mathbf{x} - \mathbf{x}^*\right)^T \mathbf{G}\left(\mathbf{x} - \mathbf{x}^*\right) = \frac{1}{2}\left(\alpha - \alpha^*\right)\mathbf{S}^T\mathbf{G}\mathbf{S}\left(\alpha - \alpha^*\right). \tag{2.46}$$

We have ignored a constant term in (2.0, which has no influence on further discussion, and written the minimiser $\mathbf{x}^*$ of *q* as

$$\mathbf{x}^{(*)} = \mathbf{x}^{(1)} + \sum \alpha_i^* \mathbf{s}^{(i)},$$

and **S** is a matrix whose columns are vectors $\mathbf{s}^{(i)}$. Since $\mathbf{s}^{(i)}$ are conjugate with respect to **G**, the product $\mathbf{S}^T\mathbf{G}\mathbf{S}$ is a diagonal matrix with diagonal elements $d_i$, say, and therefore

[1] Notation $\alpha = [\alpha_1, \alpha_2, ..., \alpha_n]^T$ is used. Vectors denoted by Greek letters are not typed in bold, but it should be clear from the context when some quantity is vector and when scalar.

$$q(\alpha) = \frac{1}{2}\sum_{i=1}^{n}\left(\alpha_i - \alpha_i^*\right)^2 d_i \,. \tag{2.47}$$

We see that conjugacy implies a coordinate transformation from **x**-space to $\alpha$-space in which **G** is diagonal. Variables in the new system are decoupled from the point of view that $q(\alpha)$ can be minimised by applying successive minimisations in coordinate directions, which results in a minimiser $\alpha^*$ corresponding to $\mathbf{x}^*$ in the **x** space. A conjugate direction method therefore corresponds to the alternating variable method applied in the new coordinate system. Enforcing conjugacy overcomes the basic problem associated with the alternating variable method, i.e. the fact that minimisation along one coordinate direction usually spoils earlier minimisations in other directions, which is the reason for oscillating behaviour of the method shown in Figure 2.1. Since a similar problem is associated with the steepest descent method, conjugacy can be successfully combined with derivative methods.

A side observation is that eigenvectors of **G** are orthogonal vectors conjugate to **G**. A quadratic function is therefore minimised by exact minimisation along all eigenvectors of its Hessian. Construction of the conjugate direction methods will show that there is no need to know eigenvectors of **G** in order to take advantage of conjugacy, but it is possible to construct conjugate directions starting with an arbitrary direction.

Another important issue in optimisation algorithms is when to terminate the algorithm. Since we can not check directly how close to the minimiser the current iterate is, the test can be based on conditions for a local minimum, for example

$$\left\|\mathbf{g}^{(k)}\right\| \le \varepsilon \,, \tag{2.48}$$

where $\varepsilon$ is some tolerance. Sometimes it is not easy to decide what magnitude to choose for $\varepsilon$, since a good decision would require some clue about the curvature in the minimum. The above test is also dependent on the scaling of variables. Another difficulty is that it can terminate in a stationary point that is not a minimum. When second derivative information is available, it should be used to exclude this possibility.

When the algorithm converges rapidly, tests based on differences between iterates can be used, e.g.

$$\left|x_i^{(k)} - x_i^{(k+1)}\right| \le \varepsilon_i \;\forall i \tag{2.49}$$

or

$$f^{(k)} - f^{(k+1)} \leq \varepsilon . \tag{2.50}$$

These tests rely on a prediction how much at most *f* can be further reduced or **x** approached to the minimum.

The test

$$\frac{1}{2}\mathbf{g}^{(k)T}\mathbf{H}^{k}\mathbf{g}^{(k)}, \tag{2.51}$$

where **H** is the inverse Hessian or its approximation, is also based on predicted change of *f*.

Finally, the possibility of termination when the number of iterations exceeds some user supplied limit is a useful property of every algorithm. Even when good local convergence results exist for a specific algorithm, this is not necessarily a guarantee for good performance in practice. Function evaluation is always subjected to numerical errors and this can especially affect algorithmic performance near the solution where local convergence properties should take effect.

## *2.5 Newton-like Methods*

Newton-like methods are based on a quadratic model, more exactly on the second-order Taylor approximation (equation ) of $f(\mathbf{x})$ about $\mathbf{x}^{(k)}$. The basic ideas around this were explained in sections 2.2 and 2.4 and will be further developed in this section.

In section 2.2 Newton's method was derived from the solution of the system of equations

$$\nabla\mathbf{g}(\mathbf{x}) = 0 ,$$

where the iteration formula was derived from the first order Taylor's approximation of **g**(**x**), giving iteration formula (2.3). Two problems related with direct application of the method were mentioned there, i.e. lack of global convergence properties and explicit use of the second order derivative information regarding the objective functions. Some general ideas on how to overcome these problems were outlined in

section 2.4 and will be further developed in this section for algorithms, which in principle stick with the basic idea of Newton's method.

In order to take over and develop the ideas given in section 2.4, let us start from the second order Taylor approximation of *f* itself, developed around the current iterate:

$$f\left(\mathbf{x}^{(k)}+\delta\right)\approx q^{(k)}(\delta)= f^{(k)}+\mathbf{g}^{(k)T}\delta+\frac{1}{2}\delta^{T}\mathbf{G}^{(k)}\delta\,. \tag{2.52}$$

Using the results of section 2.4, the stationary point of this approximation is a solution of a linear system of equations

$$\mathbf{G}^{(k)}\delta=-\mathbf{g}^{(k)}\,. \tag{2.53}$$

It is unique if $\mathbf{G}^{(k)}$ is non-singular and corresponds to a minimiser if $\mathbf{G}^{(k)}$ is positive definite. Newton's method is obtained by considering $\delta^{(k)}$ as solution of the above equation and setting the next guess to $\mathbf{x}^{(k)}+\delta^{(k)}$. The *k*-th iteration of Newton's method is then

1. Solve (2.0 for $\delta^{(k)}$,
2. Set $\mathbf{x}^{(k+1)}=\mathbf{x}^{(k)}+\delta^{(k)}$.

This is well defined as a minimisation method only if $\mathbf{G}^{(k)}$ is positive definite in each iteration, and this can be readily checked if for example LDLT decomposition is used for solution of (2.0. However, even if $\mathbf{G}^{(k)}$ is positive definite, the method may not converge from any initial guess, and it can happen that $\left\{f^{(k)}\right\}$ do not even decrease.

Line search can be used to eliminate this problem. The solution of (2.0 then defines merely the search direction, rather than correction $\delta^{(k)}$. The correction is then obtained by line minimisation, and such a method is called Newton's method with line search. The direction of search is

$$\mathbf{s}^{(k)}=-\mathbf{G}^{(k)^{-1}}\mathbf{g}^{(k)}\,. \tag{2.54}$$

If $\mathbf{G}^{(k)}$ and hence its inverse are positive definite, this defines a descent direction. If $\mathbf{G}^{(k)}$ is not positive definite, it may be possible to make a line search in $\pm\mathbf{s}^{(k)}$, but the relevance of searching in $-\mathbf{s}^{(k)}$ is questionable because this is not a direction towards a stationary point of $q(\delta)$. Furthermore, the method fails if any $\mathbf{x}^{(k)}$ is a saddle point of *f*. This gives $\mathbf{s}^{(k)}=0$, although $\mathbf{x}^{(k)}$ is not a minimiser of *f*.

One possibility of how to overcome this problem is to switch to the steepest descent direction whenever $\mathbf{G}^{(k)}$ is not positive definite. This can be done in conjunction with the angle criterion to achieve global convergence.

Minimising in the steepest descent directions can lead to undesired oscillatory behaviour where small reductions of *f* are achieved in each iteration. This happens because second order model information is ignored, as shown in section 2.4.3. The alternative approach is to switch between the Newton and steepest descent direction in a continuous way, controlling the influence of both through some adaptive weighting parameter. This can be achieved by adding a multiple of the unit matrix to $\mathbf{G}^{(k)}$ so that the search direction is defined as

$$\left(\mathbf{G}^{(k)} + \nu \mathrm{I}\right)\mathbf{s}^{(k)} = -\mathbf{g}^{(k)}. \tag{2.55}$$

Parameter $\nu$ is chosen so that $\mathbf{G}^{(k)} + \nu \mathrm{I}$ is positive definite. If $\mathbf{G}^{(k)}$ is close to positive definite, a small $\nu$ is sufficient and the method therefore uses the curvature information to a large extent. When large values of $\nu$ are necessary, the search directions becomes similar to the steepest descent direction $-\mathbf{g}^{(k)}$.

This method still fails when some $\mathbf{x}^{(k)}$ is a saddle point, and the second order information is not used in the best possible way. Further modification of the method incorporates the restricted step approach in which minimisation of the model quadratic function subjected to length restriction is minimised.

## 2.5.1 Quasi-Newton Methods

In the Newton-like methods discussed so far the second derivatives of *f* are necessary and substantial problems arise when the Hessian matrix of the function is not positive definite. The second derivatives of $\mathbf{G}^{(k)}$ can be evaluated by numerical differentiation of the gradient vector. In most cases it is advisable that after this operation $\mathbf{G}$ is made symmetric by $\mathbf{G} = \frac{1}{2}\left(\overline{\mathbf{G}} + \overline{\mathbf{G}}^{T}\right)$, where $\overline{\mathbf{G}}$ is the finite difference approximation of the Hessian matrix. However, evaluation of $\mathbf{G}$ can be unstable in the presence of numerical noise, and it is also expensive, because quadratic model information built in the previous iterates is disregarded.

The above mentioned problems are avoided in so called quasi-Newton methods. In these methods $\mathbf{G}^{(k)^{-1}}$ are approximated by symmetric matrices $\mathbf{H}^{(k)}$, which are updated from iteration to iteration using the most recently obtained information. Analogous to Newton's method with line search, line minimisations are performed in each iteration in the direction

$$\mathbf{s}^{(k)} = -\mathbf{H}^{(k)}\mathbf{g}^{(k)}. \tag{2.56}$$

By updating approximate $\mathbf{G}^{-1}$ rather than **G**, a system of equations is avoided and the search direction is obtained simply by multiplication of the gradient vector by a matrix. An outline of the algorithm is given below:

**Algorithm 2.2:** General quasi-Newton algorithm.

Given a positive definite matrix $\mathbf{H}^{(1)}$, the *k*-th iteration is:

1. Calculate $\mathbf{s}^{(k)}$ according to (2.0.
2. Minimise *f* along $\mathbf{s}^{(k)}$, set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{s}^{(k)}$, where $\alpha^{(k)}$ is a line minimum.
3. Update $\mathbf{H}^{(k)}$ to obtain $\mathbf{H}^{(k+1)}$.

If no second derivative information is available at the beginning, $\mathbf{H}^{(1)}$ can be any positive definite matrix, e.g. $\mathbf{H}^{(1)} = \mathbf{I}$. The inexact line search strategy can be used in line 2. If $\mathbf{H}^{(k)}$ is positive definite, the search directions are descent. This is desirable and the most important are those quasi-Newton methods, which maintain positive definiteness of $\mathbf{H}^{(k)}$.

The updating formula should explicitly use only first derivative information. Repeated updating should change arbitrary $\mathbf{H}^{(1)}$ to a close approximation of $\mathbf{G}^{(k)-1}$ The updating formula is therefore an attempt to augment the current $\mathbf{H}^{(k)}$ with second derivative information gained in the current iteration, i.e. by evaluation of $f$ and $\nabla f$ at two distinct points. In this context equation (2.0, which relates the Hessian matrix of a quadratic function with its gradient in two distinct points, requires attention.

Let us write

$$\delta^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \tag{2.57}$$

and

$$\gamma^{(k)} = \mathbf{g}^{(k+1)} - \mathbf{g}^{(k)}. \tag{2.58}$$

Using the Taylor series of **g** about $\mathbf{x}^{(k)}$ gives a relationship similar to (2.0, i.e.

$$\gamma^{(k)} = \mathbf{G}^{(k)}\delta^{(k)} + o\left(\left\|\delta^{(k)}\right\|\right). \tag{2.59}$$

The updating formula should therefore correct $\mathbf{H}^{(k+1)}$ so that the above relation would hold approximately with $\mathbf{H}^{(k+1)^{-1}}$ in place of $\mathbf{G}^{(k)}$. This gives the so called quasi-Newton condition, in which the updating formula must satisfy

$$\mathbf{H}^{(k+1)}\gamma^{(k)} = \delta^{(k)}. \tag{2.60}$$

Since this condition gives only one equation, it does not uniquely define the updating formula and permits various ways of updating **H**. One possibility is to add a symmetric rank one matrix to $\mathbf{H}^{(k)}$, i.e.

$$\mathbf{H}^{(k+1)} = \mathbf{H}^{(k)} + \mathbf{u}\mathbf{u}^{T}. \tag{2.61}$$

Substituting this into (2.0 gives

$$\mathbf{H}^{(k)}\gamma^{(k)} + \mathbf{u}\mathbf{u}^{T}\gamma^{(k)} = \delta^{(k)}. \tag{2.62}$$

Since $\mathbf{u}^{(T)}\gamma^{(k)}$ is a scalar, matrix multiplication is associative and multiplication with a scalar is commutative, **u** must be proportional to $\delta^{(k)} - \mathbf{H}^{(k)}\gamma^{(k)}$. Writing

$$\mathbf{u} = a\left(\delta^{(k)} - \mathbf{H}^{(k)}\gamma^{(k)}\right)$$

and inserting this into (2.0 gives $a = 1\Big/\sqrt{\left(\delta^{(k)} - \mathbf{H}^{(k)}\gamma^{(k)}\right)^{T}\gamma^{(k)}}$ and therefore

$$\mathbf{H}^{(k+1)} = \mathbf{H}^{(k)} + \frac{\left(\delta^{(k)} - \mathbf{H}^{(k)}\gamma^{(k)}\right)\left(\delta^{(k)} - \mathbf{H}^{(k)}\gamma^{(k)}\right)^{(T)}}{\left(\delta^{(k)} - \mathbf{H}^{(k)}\gamma^{(k)}\right)^{(T)}\gamma^{(k)}}. \tag{2.63}$$

This formula is called the rank one updating formula according to the above derivation.

For a quadratic function with positive definite Hessian the rank one method terminates in at most $n$+1 steps with $\mathbf{H}^{(n+1)} = \mathbf{G}^{-1}$, provided that $\delta^{(1)}, ..., \delta^{(n)}$ are independent and that the method is well defined[4]. The proof does not require exact line searches. Also the so called hereditary property can be established, i.e.

$$\mathbf{H}^{(i)}\gamma^{(j)} = \delta^{(j)}, \quad j = 1, 2, ..., i-1. \tag{2.64}$$

A disadvantage is that in general the formula does not maintain positive definiteness of $\mathbf{H}^{(k)}$ and the dominator in (2.0 can become zero.

Better formulas can be obtained by allowing the correction to be of rank two. This can always be written[32],[34] as

$$\mathbf{H}^{(k+1)} = \mathbf{H}^{(k)} + \mathbf{u}\mathbf{u}^T + \mathbf{v}\mathbf{v}^T . \tag{2.65}$$

Using this in the quasi-Newton condition gives

$$\delta^{(k)} = \mathbf{H}^{(k)}\gamma^{(k)} + \mathbf{u}\mathbf{u}^T\gamma^{(k)} + \mathbf{v}\mathbf{v}^T\gamma^{(k)} . \tag{2.66}$$

**u** and **v** can not be determined uniquely. A straightforward way of satisfying the above equation is to set **u** proportional to $\delta^{(k)}$ and **v** proportional to $\mathbf{H}^{(k)}\gamma^{(k)}$. By solution of the equation separately for both groups of proportional vectors the Davidon – Fletcher - Powell or DFP updating formula is obtained:

$$\mathbf{H}_{DFP}^{(k+1)} = \mathbf{H} + \frac{\delta\delta^T}{\delta^T\gamma} - \frac{\mathbf{H}\gamma\gamma^T\mathbf{H}}{\gamma^T\mathbf{H}\gamma} . \tag{2.67}$$

Indices *k* have been omitted for the sake of simplicity (this approach will be adopted through this section) and the symmetry of **H** is used.

Another rank two updating formula can be obtained by considering updating and approximating **G** instead of $\mathbf{G}^{-1}$. Let us write $\mathbf{B}^{(k)} = \mathbf{H}^{(k)-1}$ and consider updating $\mathbf{B}^{(k)}$ in a similar way as $\mathbf{H}^{(k)}$ was updated according to the DFP formula. We require that the quasi-Newton condition (2.0 is preserved. This was true for the DFP formula, but now we are updating inverse of $\mathbf{H}^{(k)}$, therefore, according to (2.0, $\gamma^{(k)}$ and $\delta^{(k)}$ must be interchanged. This gives the formula

$$\mathbf{B}_{BFGS}^{(k+1)} = \mathbf{B} + \frac{\gamma\gamma^T}{\gamma^T\delta} - \frac{\mathbf{B}\delta\delta^T\mathbf{B}}{\delta^T\mathbf{B}\delta} . \tag{2.68}$$

We however still want to update $\mathbf{H}^{(k)}$ rather than $\mathbf{B}^{(k)}$, because a solution of system of equations is in this way avoided in the quasi-Newton iteration. The following updating formula satisfies $\mathbf{B}_{BFGS}^{(k+1)}\mathbf{H}_{BFGS}^{(k+1)} = \mathbf{I}$:

$$\mathbf{H}_{BFGS}^{(k+1)} = \mathbf{H} + \left(1 + \frac{\gamma^T\mathbf{H}\gamma}{\delta^T\gamma}\right)\frac{\delta\delta^T}{\delta^T\gamma} - \left(\frac{\delta\gamma^T\mathbf{H} + \mathbf{H}\gamma\delta^T}{\delta^T\gamma}\right). \tag{2.69}$$

This is called the Broyden – Fletcher – Goldfarb – Shanno or BFGS updating formula.

The BFGS and the DFP formula are said to be dual or complementary because the expressions for $\mathbf{B}^{(k+1)}$ and $\mathbf{H}^{(k+1)}$ in one are obtained by interchanging $\mathbf{B} \leftrightarrow \mathbf{H}$ and $\gamma \leftrightarrow \delta$ in the other. Such duality transformation preserves the quasi-Newton condition. The rank one formula is self-dual.

The DFP and BFGS updating formula can be combined to obtain the so called Broyden one-parameter family of rank two formulae:

$$\mathbf{H}_{\phi}^{k+1} = (1-\phi)\mathbf{H}_{DFP}^{(k+1)} + \phi\,\mathbf{H}_{BFGS}^{(k+1)}. \tag{2.70}$$

This family includes the DFP and BFGS and also rank 1 formula. The quasi-Newton method with a Broyden family updating formula has the following properties[4]:

1. For a quadratic function with exact line searches:
   - The method terminates in at most *n* iterations with $\mathbf{H}^{(n+1)} = \mathbf{G}^{-1}$.
   - Previous quasi-Newton conditions are preserved (hereditary property (2. 0).
   - Conjugate directions are generated, and conjugate gradients when $\mathbf{H}^{(0)} = \mathbf{I}$.
2. For general functions:
   - The method has superlinear order of convergence.
   - The method is globally convergent for strictly convex functions if exact line searches are performed.

The Broyden family updates maintain positive definiteness of $\mathbf{H}_{\phi}^{(k+1)}$ for $\phi \geq 0$.

Global convergence has also been proved for the BFGS method with inexact line searches, applied to a convex objective function[4]. The BFGS method with inexact line searches converges superlinearly if $\mathbf{G}^{(*)}$ is positive definite.

The BFGS method also shows good performance in numerical experiments. The method is not sensitive to exactness of line searches, in fact it is a generally accepted opinion that inexact line searches are more efficient with the BFGS method than near exact line searches. The contemporary optimisation literature[4],[7] suggests the BFGS method as preferable choice for general unconstrained optimisation based on a line search prototype algorithm.

### 2.5.2 Invariance Properties

It is important to study how optimisation algorithms perform when affine transformation of variables is made, i.e.

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{a}, \tag{2.71}$$

where **A** is nonsingular. This is a one-to-one mapping with inverse transformation

$$\mathbf{x} = \mathbf{A}^{-1}(\mathbf{y} - \mathbf{a}).$$

*f* can be evaluated either in **x** space (denoted by $f_x(\mathbf{x})$) or in **y** space (denoted by $f_y(\mathbf{y}) = f_x(\mathbf{A}^{-1}(\mathbf{y} - \mathbf{a}))$).

Applying the chain rule for derivation in **x** space gives

$$\frac{\partial}{\partial x_i} = \sum_{k=1}^{n} \frac{\partial y_k}{\partial x_i} \frac{\partial}{\partial y_k} = \sum_{k=1}^{n} \left(\mathbf{A}^T\right)_{ik} \frac{\partial}{\partial y_k}, \tag{2.72}$$

therefore $\nabla_x = \mathbf{A}^T \nabla_y$ and so

$$\mathbf{g}_x = \mathbf{A}^T \mathbf{g}_y. \tag{2.73}$$

Applying the gradient operator to the above equation then gives $\nabla_x \mathbf{g}_x{}^T = \mathbf{A}^T \nabla_y \mathbf{g}_y{}^T \mathbf{A}$, i.e.

$$\mathbf{G}_x = \mathbf{A}^T \mathbf{G}_y \mathbf{A}. \tag{2.74}$$

The notation $\mathbf{g}_y = \nabla_y f_y$, etc. was used, so that for example

$$\left[\mathbf{G}_y\right]_{ij} = \frac{\partial^2 f_y}{\partial y_i \partial y_j}.$$

The following theorem[4] applies to Newton-like methods:

**Theorem 2.3:**

*If* $\mathbf{H}^{(k)}$ *transforms under transformation (2.0 as*

$$\mathbf{H}_x^{(k)} = \mathbf{A}^{-1}\mathbf{H}_y^{(k)}\mathbf{A}^{-T} \quad \forall k\,, \tag{2.75}$$

*then a Newton-like method with fixed step* $\alpha^{(k)}$ *is invariant under the transformation (2.0. A method is also invariant if* $\alpha^{(k)}$ *is determined by tests on* $f^{(k)}$, $\mathbf{g}^{(k)T}\mathbf{s}^{(k)}$ *or other invariant scalars.*

Transformation (2.0 in the above theorem is obtained by inverting (2.0, since $\mathbf{H}^{(k)}$ approximate $\mathbf{G}^{(k)}$ in the quasi-Newton methods.

We see that the steepest descent method (treated as quasi-Newton method with $\mathbf{H}^{(k)} = \mathbf{I}$) is not invariant under transformation (2.0 because $\mathbf{I}$ does not transform correctly. Modified Newton methods are also not invariant because $\mathbf{G} + \nu\mathbf{I}$ does not transform correctly when $\nu > 0$.

For a quasi-Newton method to be invariant, $\mathbf{H}^{(1)}$ must be chosen so as to transform correctly (as (2.0) and the updating formula must preserve the transformation property (2.0. Therefore, if $\mathbf{H}^{(1)} = \mathbf{I}$ is chosen, then invariance does not hold. $\mathbf{H}^{(1)} = \mathbf{G}\left(\mathbf{x}^{(1)}\right)^{-1}$ transforms correctly and therefore this choice does not affect invariance.

In order to show that a specific updating formula preserves the transformation property (2.0, we must show that $\mathbf{A}\mathbf{H}_x^{(k)}\mathbf{A}^T = \mathbf{H}_y^{(k)}$ (which is (2.0 pre-multiplied by $\mathbf{A}$ and post-multiplied by $\mathbf{A}^T$) which implies $\mathbf{A}\mathbf{H}_x^{(k+1)}\mathbf{A}^T = \mathbf{H}_y^{(k+1)}$. Let us do this for the DFP formula

$$\mathbf{H}_x^{(k+1)} = \mathbf{H}_x + \frac{\delta_x\delta_x^T}{\delta_x^T\gamma_x} - \frac{\mathbf{H}_x\gamma_x\gamma_x^T\mathbf{H}_x}{\gamma_x^T\mathbf{H}_x\gamma_x}\,. \tag{2.76}$$

We will pre-multiply the above equation by $\mathbf{A}$ and post-multiply it by $\mathbf{A}^T$ and use relations $\mathbf{A}\delta_x = \delta_y$ following from (2.0 and $\gamma_x = \mathbf{A}^T\gamma_y$ following from (2.0. We will consider individual terms in equation (2.0.

The first term on the right-hand side of (2.0 gives, after multiplication, $\mathbf{H}_y^{(k)}$ by assumption. Consider then the denominator of the second term:

$$\delta_x^T\gamma_x = \delta_x^T\mathbf{A}^T\mathbf{A}^{-T}\gamma_x = \left(\mathbf{A}\delta_x\right)^T\gamma_y = \delta_y^T\gamma_y\,,$$

the denominator is invariant. The numerator after multiplication gives

$$\mathbf{A}\delta_x\delta_x{}^T\mathbf{A}^T = \delta_y\delta_y{}^T,$$

so the second term transforms correctly. Consider the denominator of the third term:

$$\gamma_x{}^T\mathbf{H}_x\gamma_x = \gamma_x{}^T\mathbf{A}^{-1}\mathbf{A}\mathbf{H}_x\mathbf{A}^T\mathbf{A}^{-T}\gamma_x = \left(\mathbf{A}^{-T}\gamma_x\right)^T\mathbf{H}_y\mathbf{A}^{-T}\gamma_x = \gamma_y{}^T\mathbf{H}_y\gamma_y,$$

the denominator is invariant under transformation. The numerator after multiplication is

$$\mathbf{A}\mathbf{H}_x\gamma_x\gamma_x{}^T\mathbf{H}_x\mathbf{A}^T = \mathbf{A}\mathbf{H}_x\mathbf{A}^T\mathbf{A}^{-T}\gamma_x\gamma_x{}^T\mathbf{A}^{-1}\mathbf{A}\mathbf{H}_x\mathbf{A}^T = \mathbf{H}_y\gamma_y\left(\mathbf{A}^{-T}\gamma_x\right)^T\mathbf{H}_y = \mathbf{H}_y\gamma_y\gamma_y{}^T\mathbf{H}_y,$$

so the third term is also transformed correctly. $\mathbf{A}\mathbf{H}_x^{(k+1)}\mathbf{A}^T = \mathbf{H}_y^{(k+1)}$ is valid since

$$\mathbf{A}\mathbf{H}_x^{(k+1)}\mathbf{A}^T = \mathbf{H}_y + \frac{\delta_y\delta_y{}^T}{\delta_y{}^T\gamma_y} - \frac{\mathbf{H}_y\gamma_y\gamma_y{}^T\mathbf{H}_y}{\gamma_y{}^T\mathbf{H}_y\gamma_y}$$

and this is the DFP formula in the **y** space.

Similarly the preservation of (2.0 can be proved for all updating formulas in which the correction is a sum of rank one terms constructed from vectors $\delta$ and $\mathbf{H}\gamma$, multiplied by invariant scalars. Such versions are the BFGS formula and hence all Broyden family formulas.

The Broyden family (including BFGS and DFP) algorithms are therefore invariant under the affine transformation of variables (2.0, provided that $\mathbf{H}^{(1)}$ is chosen so as to transform correctly, i.e. as (2.0. However, even if $\mathbf{H}^{(1)}$ is not chosen correctly, after *n* iterations we have $\mathbf{H}^{(n+1)} \approx \mathbf{G}^{(n+1)^{-1}}$, which is transforms correctly. The method therefore becomes close to the one in which invariance is preserved.

Invariance to an affine transformation of variables is a very important algorithmic property. Algorithms which have this property, are less sensitive to situations in which **G** is ill-conditioned, since an implicit transformation which transforms **G** to the unity matrix **I** can be introduced, which does not change the method. Algorithms that are not invariant, i.e. the steepest descent or the alternating variables method, can perform very badly when the Hessian is ill-conditioned.

When using methods which are not invariant, it can be advantageous to find a linear transformation which improves the conditioning of the problem[17].

If columns of $\mathbf{A}$ are eigenvectors of **G**, then **G** is diagonalised when transformation (2.0 is applied. Conditioning can be achieved by additional scaling of variables, i.e. by multiplication with a diagonal matrix. This approach is however not applicable in practice because it is usually difficult to calculate eigenvectors of **G**. For positive definite **G** the same effect is achieved by using Choleski factors of **G** as the transformation matrix. $\mathbf{G}_x = \mathbf{A}^T\mathbf{A}$ gives

$$\mathbf{G}_y = \mathbf{A}^{-T}\mathbf{G}_x\mathbf{A}^{-1} = \mathbf{A}^{-T}\mathbf{A}^T\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}.$$

It is often possible to improve conditioning just by scaling the variables. In this case **A** is chosen to be a diagonal matrix so that $\mathbf{A}^2$ estimates $\mathbf{G}_x$ in some sense. $\mathbf{G}_y = \mathbf{A}^{-T}\mathbf{G}_x\mathbf{A}^{-1}$ (from (2.0) is required to be close to the unity matrix in some sense. It can be required, for example, that $\left[\mathbf{G}_y\right]_{ii} = 1 \forall i$. It is usually not necessary to explicitly perform the scaling, but **I** can be replaced in the methods by a suitable diagonal matrix. For example, the modified Newton method can be improved by using $\mathbf{G} + \nu\mathbf{A}^{-2}$ in place of $\mathbf{G} + \nu\,\mathrm{I}$.

## *2.6 Conjugate Direction Methods*

Optimisation algorithms described in this section are based on the result given in Theorem 2.2, which associates conjugacy and exact line searches with quadratic termination. These algorithms rely on an idealized assumption that exact line searches are performed. This is possible for a quadratic function, but not in general. By using interpolation in the line search algorithm, it is still possible to locate a local minimum up to a certain accuracy, and this approach is used in practice with the conjugate direction methods. An argument which justifies this is that in the close neighbourhood of a minimum, quadratic interpolations of the objective functions will enable the line minimum to be located almost exactly, so that the inexact nature of the line search algorithm will not spoil local convergence properties, which are theoretically based on the assumption of exact line search.

In section 2.6.1 derivative based conjugate direction methods are described. The described methods generate conjugate directions when they are applied to a quadratic function.

## 2.6.1 Conjugate Gradient Methods

Conjugate gradient methods begin with line search along

$$\mathbf{s}^{(1)} = -\mathbf{g}^{(1)} \tag{2.77}$$

and then generate search directions $\mathbf{s}^{(k+1)}, k \geq 1$ from $-\mathbf{g}^{(k+1)}$, so that they are conjugate to $\mathbf{s}^{(1)}, ..., \mathbf{s}^{(k)}$ with respect to the Hessian matrix $\mathbf{G}$ when a method is applied to a quadratic function.

For a quadratic function it follows from (2.0 that

$$\gamma^{(k)} = \mathbf{G}\delta^{(k)} \; \forall k \,, \tag{2.78}$$

where $\gamma^{(k)} = \mathbf{g}^{(k+1)} - \mathbf{g}^{(k)}$ and $\delta^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$, as usual. Conjugacy conditions (2.0 can therefore be written as

$$\mathbf{s}^{(i)^T} \gamma^{(j)} = 0 \quad j \neq i \tag{2.79}$$

since $\gamma^j = \mathbf{G}\delta^{(j)} = \mathbf{G}\alpha^{(j)}\mathbf{s}^{(j)}$. The last expression is a consequence of the fact that $\mathbf{x}^{(j+1)}$ is obtained by a line search performed from $\mathbf{x}^{(j)}$ along $\mathbf{s}^{(j)}$.

The above equation can be used to prove an important property. First we can see that

$$\mathbf{s}^{i^T} \mathbf{g}^{(i+1)} = 0 \;\; \forall i \,, \tag{2.80}$$

because exact line searches are used. By using the above equation and (2.0 we obtain

$$\begin{aligned} &\mathbf{s}^{(i)^T} \mathbf{g}^{(k+1)} = \\ &\mathbf{s}^{(i)^{(T)}} \left( \mathbf{g}^{(k+1)} - \mathbf{g}^{(k)} + \mathbf{g}^{(k)} - \mathbf{g}^{(k-1)} + ... - \mathbf{g}^{(i+1)} + \mathbf{g}^{(i+1)} \right) = , \\ &\mathbf{s}^{(i)^{(T)}} \left( \gamma^{(k)} + \gamma^{(k)} + ... \gamma^{(i+1)} + \mathbf{g}^{(i+1)} \right) = 0 \;\; \forall i, k > i \end{aligned} \tag{2.81}$$

This means that $\mathbf{g}^{(k+1)}$ is orthogonal to all search directions of previous steps:

$$\mathbf{s}^{(i)T}\mathbf{g}^{(k+1)} = 0 \quad \forall k, i \le k \,. \tag{2.82}$$

This is actually the result of Theorem 2.2.

In the Fletcher-Reeves method $\mathbf{s}^{(k+1)}$ is obtained from $-\mathbf{g}^{(k+1)}$ by the extended Gramm-Schmidt orthogonalisation[30],[32] with respect to $\gamma^{(i)}$, $j \le k$, in order to satisfy conjugacy conditions (2.0. We can write[1]

$$\mathbf{s}^{(k+1)} = -\mathbf{g}^{(k+1)} + \sum_{j=1}^{k} \beta^{(j)}\mathbf{s}^{(j)} \,. \tag{2.83}$$

Multiplying the transpose of the above equation by $\gamma^{(i)}$ gives

$$\mathbf{s}^{(k+1)^T}\gamma^{(i)} = 0 = -\mathbf{g}^{(k+1)^T}\gamma^{(i)} + \beta^{(i)}\mathbf{s}^{(i)^T}\gamma^{(i)}, \tag{2.84}$$

where (2.0 was taken into account. It follows that

$$\beta^{(i)} = \frac{\mathbf{g}^{(k+1)^T}\gamma^{(i)}}{\mathbf{s}^{(i)^T}\gamma^{(i)}} = \frac{\mathbf{g}^{(k+1)^T}\left(\mathbf{g}^{(i+1)} - \mathbf{g}^{(i)}\right)}{\mathbf{s}^{(i)^T}\left(\mathbf{g}^{(i+1)} - \mathbf{g}^{(i)}\right)}. \tag{2.85}$$

It follows from construction of $\mathbf{s}^{(k)}$ ((2.0 and (2.0) that vectors $\mathbf{g}^{(1)}, ..., \mathbf{g}^{(k)}$ and $\mathbf{s}^{(1)}, ..., \mathbf{s}^{(k)}$ span the same subspace. Therefore, since $\mathbf{g}^{(k+1)}$ is orthogonal to the subspace spanned by $\mathbf{s}^{(1)}, ..., \mathbf{s}^{(k)}$ due to (2.0, it is also orthogonal to vectors $\mathbf{g}^{(1)}, ..., \mathbf{g}^{(k)}$, i.e.

$$\mathbf{g}^{(i)^T}\mathbf{g}^{(k+1)} = 0 \quad \forall k, i \le k \tag{2.86}$$

We see that only $\beta^{(k)} \neq 0$ and that

$$\beta^{(k)} = \frac{\mathbf{g}^{(k+1)^T}\left(\mathbf{g}^{(k+1)} - \mathbf{g}^{(k)}\right)}{\mathbf{s}^{(k)^T}\left(\mathbf{g}^{(k+1)} - \mathbf{g}^{(k)}\right)} = -\frac{\mathbf{g}^{(k+1)^T}\mathbf{g}^{(k+1)}}{\mathbf{s}^{(k)^T}\mathbf{g}^{(k)}} \tag{2.87}$$

[1] The derivation of the Fletcher-Reeves method was found to be not completely clear in some optimisation literature and is therefore included herein.

The denominator of the above equation can be obtained by substituting $\mathbf{s}^{(k)}$ by (2.0 with decreased indices and taking into account that only $\beta^{(k-1)}$ is non-zero, together with the established orthogonality properties:

$$\mathbf{s}^{(k)T}\mathbf{g}^{(k)} = \left(-\mathbf{g}^{(k)} + \beta^{(k-1)}\mathbf{s}^{(k-1)}\right)^T \mathbf{g}^{(k)} = -\mathbf{g}^{(k)T}\mathbf{g}^{(k)}.$$

Now we have

$$\beta^{(k)} = \frac{\mathbf{g}^{(k+1)T}\mathbf{g}^{(k+1)}}{\mathbf{g}^{(k)T}\mathbf{g}^{(k)}}. \tag{2.88}$$

The obtained results can be summarized in the following way:

**Theorem 2.4:**

*The Fletcher-Reeves method with exact line searches terminates for a quadratic function at a stationary point* $\mathbf{x}^{m+1}$ *after* $m \le n$ *iterations. In addition, the following results hold for* $1 \le i \le m$*:*

$$\mathbf{s}^{(i)T}\mathbf{G}\mathbf{s}^{(j)} = 0; \quad j = 1, 2, ..., i-1 \text{ (conjugate directions)}, \tag{2.89}$$

$$\mathbf{g}^{(i)T}\mathbf{g}^{(j)} = 0; \quad j = 1, 2, ..., i-1 \text{ (orthogonal gradients)} \tag{2.90}$$

*and*

$$\mathbf{s}^{(i)T}\mathbf{g}^{(i)} = -\mathbf{g}^{(i)T}\mathbf{g}^{(i)} \text{ (descent conditions)}. \tag{2.91}$$

The termination must occur in at least *n* iterations because in the opposite case $\mathbf{g}^{(n+1)} \neq 0$ would contradict the result that gradients are orthogonal.

When applied to a quadratic function with positive definite **G**, the Fletcher-Reeves method turns to be equivalent to the Broyden family of methods if $\mathbf{H}^{(1)} = \mathbf{I}$, the starting point is the same and exact line searches are performed in both methods[4],[7]. For non-quadratic functions line relatively accurate line search is recommended. Resetting the search direction to the steepest descent direction periodically after every *n* iterations is generally an accepted strategy in practice. When compared with quasi-Newton methods, conjugate gradient methods are less efficient and less robust and they are more sensitive to the accuracy of the line search algorithm. Methods with resetting are globally convergent and exhibit n-step superlinear convergence, i.e.

$$\lim_{k\to\infty}\frac{\left\|\mathbf{x}^{(k+n)}-\mathbf{x}^{*}\right\|}{\left\|\mathbf{x}^{(k)}-\mathbf{x}^{*}\right\|}=0 \tag{2.92}$$

Some other formulas may be used instead of (2.0. Examples are the conjugate descent formula

$$\beta^{(k)}=\frac{\mathbf{g}^{(k+1)^T}\mathbf{g}^{(k+1)}}{\mathbf{g}^{(k)^T}\mathbf{s}^{(k)}} \tag{2.93}$$

and the Polak-Ribiere formula

$$\beta^{(k)}=\frac{\left(\mathbf{g}^{(k+1)}-\mathbf{g}^{(k)}\right)^T\mathbf{g}^{(k+1)}}{\mathbf{g}^{(k)^T}\mathbf{g}^{(k)}}. \tag{2.94}$$

Considering the derivation of the Fletcher-Reeves method, it can be seen that these formulas are equivalent to the Fletcher-Reeves formula when applied to quadratic functions with exact line searches. The conjugate descent formula has a strong descent property that $\mathbf{s}^{(k)^T}\mathbf{g}^{(k)}<0$ if $\mathbf{g}^{(k)}\neq 0$. The Polak-Ribiere formula is recommended when solving large problems[4].

Another possibility for conjugate gradient methods is to use symmetric projection matrices in the calculation of $\mathbf{s}^{(k+1)}$, which annihilate vectors $\gamma^{(k)}, ..., \gamma^{(k)}$:

$$\mathbf{s}^{(k)}=-\mathbf{P}^{(k)}\mathbf{g}^{(k)},\ k=1, 2, ..., n. \tag{2.95}$$

Initially

$$\mathbf{P}^{(1)}=\mathbf{I} \tag{2.96}$$

and subsequent $\mathbf{P}^{(k)}$ are updated as

$$\mathbf{P}^{(k+1)}=\mathbf{P}^{(k)}-\frac{\mathbf{P}^{(k)}\gamma^{(k)}\gamma^{(k)^T}\mathbf{P}^{(k)}}{\gamma^{(k)^T}\mathbf{P}^{(k)}\gamma^{(k)}}. \tag{2.97}$$

Again this method is equivalent to other described methods for quadratic functions. When applied to general functions, $\mathbf{P}^{(i)}$ must be reset to $\mathbf{I}$ every $n$ iterations since $\mathbf{P}^{(n+1)} = 0$. The method has the descent property $\mathbf{s}^{(k)T}\mathbf{g}^{(k)} \leq 0$, but has a disadvantage that matrix calculations are required in each iteration.

## *2.7 Further Remarks*

In the present chapter some of the basis of nonlinear programming is outlined. This knowledge is important for understanding the practical requirements for implementation of the algorithmic part in the optimisation shell. The literature cited in this chapter is mostly related to the mathematical and algorithmic background of optimisation and less to practical implementation (except references [6], [11] and [29]). Some implementation aspects are stressed in the next chapter within a larger framework of the optimisation shell. The need for hierarchical and modular implementation, which is stated there, is partially based on the heterogeneity of optimisation algorithms evident from the present chapter.

In practice it is not always obvious which algorithm to use in a given situation. This depends first of all on the case being solved. Although the theory can offer substantial support for making the judgment, most of the literature on optimisation methods recognize the significance of numerical experimentation alongside the theoretical development. This implies a significant aspect that was borne in mind during development of the optimisation shell. The shell should not only include a certain number of algorithms, but also provide an open framework for incorporation of new algorithms and testing them on simple model functions as well as on practical problems.

Many issues important for engineering practice were not taken into account. One of them is handling multiple conflicting optimisation criteria, i.e. solving the problem stated as

$$\begin{aligned} &\textit{minimise} && [f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_m(\mathbf{x})] \\ &\textit{subject to} && \mathbf{x} \in \Omega . \end{aligned} \tag{2.98}$$

A common approach is to weight the individual criteria, which leads to the problem

$$\textit{minimise} \qquad f(\mathbf{x}) = w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x}) + ... + w_m f_m(\mathbf{x})$$

$$\text{subject to} \quad \mathbf{x} \in \Omega, \tag{2.99}$$

where $w_1, \ldots, w_m$ are positive weighting coefficients. The problem which arises is how to choose these coefficients. The choice is made either on the basis of experience or in an iterative process where optimisation is performed several times and coefficients are varied on the basis of the optimisation results.

Sometimes it is more convenient to designate one criterion as a primary objective and to constrain the magnitude of the others, e.g. in the following way:

$$\begin{aligned} &\text{minimise} && f_1(\mathbf{x}) \\ &\text{subject to} && f_2(\mathbf{x}) \le C_2, \\ & && \ldots \\ & && f_m(\mathbf{x}) \le C_m, \\ & && \mathbf{x} \in \Omega. \end{aligned} \tag{2.100}$$

This approach suffers for a similar defect as weighting criteria, i.e. the solution depends on the choice of coefficients $C_2, \ldots, C_m$. Attempts to overcome this problem lead to consideration of Pareto optimality[12],[17] and solution of the min-max problem[12],[23].

Another important practical issue is optimisation in the presence of numerical noise. Most of the methods considered in this chapter are designed on the basis of certain continuity assumptions and do not perform well if the objective and constraint functions contain a considerable amount of noise. This can often not be avoided due to complexity of the applied numerical models and their discrete nature (e.g. adaptive mesh refinement in the finite element simulations).

A promising approach to optimisation in the presence of noise incorporates approximation techniques[38],[39]. In this approach successive low order approximations of the objective and constraint functions are made locally on the basis of sampled function values and/or derivatives. This leads to a sequence of approximate optimisation subproblems. They refer to minimisation of the approximate objective functions subject to the approximate constraints and to additional step restriction, which restricts the solution of the subproblem to the region where the approximate functions are adequate. The subproblems are solved by standard nonlinear programming methods. For approximations more data is usually sampled than the minimum amount necessary for determination of the coefficients of the approximate functions, which levels out the effect of noise. A suitable strategy

must be defined for choosing the limits of the search region and for the choice of sampling points used for approximations (i.e. the plan of experiments)[38].

A common feature of all methods mentioned in this chapter is that they at best find a local solution of the optimisation problem. There are also methods which can (with a certain probability) find the global solution or more than one local solution at once. The most commonly used are simulated annealing[29],[12],[17] and genetic algorithms[12],[17]. Most of these methods are based on statistical search, which means that they require a large number of function evaluations in order to accurately locate the solution. This makes them less convenient for use in conjunction with expensive numerical simulations, except in cases where global solutions are highly desirable. Use of these techniques can also be suitable for finding global solutions of certain optimisation problems which arise as sub-problems in optimisation algorithms and in which the objective and constraint functions are not defined implicitly through a numerical simulation.

## *References:*


[1] M. Dutko, *Software architectural, functional and design specifications*, PROFORM project interim report, 2004.

[2] Rodič,T. & Grešovnik,I., *A computer system for solving inverse and optimization problems*.- International Journal for Computer Aided Engineering and Software 15 / 6-7, 1998, 893-907.J.Baldyga, M.Jasinska, Effects of fluid motion and mixing on particle agglomeration and coating during precipitation. Chemical Engineering Science, 2005, (Article in press).

[3] *Optimization Shell Inverse*, electronic document at http://www.c3m.si/inverse/ , maintained by the Centre for Computational Continuum Mechanics, Ljubljana.

[4] R. Fletcher, *Practical Methods of Optimization (second edition)*, John Wiley & Sons, New York, 1996).

[5] E. J. Beltrami, *An Algorithmic Approach to Nonlinear Analysis and Optimization*, Academic Press, New York, 1970

[6] J. E. Dennis (Jr.), R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, SIAM, Philadelphia, 1996.

[7] D. P. Bertsekas, *Nonlinear Programming (second edition)*, Athena Scientific, Belmont, 1999.

[8] *Mathematical Optimization*, electronic book at http://csep1.phy.ornl.gov/CSEP/MO/MO.html , Computational Science Education Project, 1996.

[9] A. V. Fiacco, G. P. McCormick, *Nonlinear Programming – Sequential Unconstrained Minimisation Techniques*, Society for Industrial and Applied Mathematics, Philadelphia, 1990.

[10] D. P. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods*, Athena Scientific, Belmont, 1996.

[11] J. L. Nazareth, *The Newton – Cauchy Framework – A Unified Approach to Unconstrained Nonlinear Minimisation*, Springer – Verlag, Berlin, 1994.

[12] A. D. Belgundu, T. R. Chandrupatla: *Optimization Concepts and Applications in Engineering*, Prentice Hall, New Jersy, 1999.

[13] P. E. Gill, W. Murray, M. H. Wright, *Practical Optimization*, Academic Press, London, 1981.

[14] M. J. D. Powell (editor), *Nonlinear Optimization – Proceedings of the NATO Advanced Research Institute, Cambridge, July 1981*, Academic Press, London, 1982.

[15] M. H. Wright, *Direct Search Methods: Once Scorned, Now Respectable*, in D. F. Griffiths and G. A. Watson (eds.), Numerical Analysis 1995 (Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis, p.p. 191 – 208, Addison Wesley Longman, Harlow, 1996.

[16] K.G. Murty, *Linear Complementarity, Linear and Nonlinear Programming*, Helderman-Verlag, 1988.

[17] S.R. Singiresu, *Engineering Optimization – Theory and Practice (third edition)*, John Wiley & Sons, New York, 1996.

[18] E. Panier, A. L. Tits, *On Combining Feasibility, Descent and Superlinear Convergence in Inequality Constrained Optimization*, Mathematical Programming, Vol. 59 (1993), p.p. 261 - 276.

[19] C. T. Lawrence, A. L. Tits, *Nonlinear Equality Constraints in Feasible Sequential Quadratic Programming*, Optimization Methods and Software, Vol. 6, 1996, pp. 265 - 282.

[20] J. L. Zhou, A. L. Tits, *An SQP Algorithm for Finely Discretized Continuous Minimax Problems and Other Minimax Problems With Many Objective Functions*, SIAM Journal on Optimization, Vol. 6, No. 2, 1996, pp. 461 - 487.

[21] P. Armand, J. C. Gilbert, *A piecewise Line Search Technique for Maintaining the Positive Definiteness of the Matrices in the SQP Method*, Research Report No. 2615 of the "Institut national de recherche en informatique et en automatique", Rocquencourt, 1995.

[22] C. T. Lawrence, A. L. Tits, *Feasible Sequential Quadratic Programming for Finely Discretized Problems from SIP*, in R. Reemtsen, J.-J. Ruckmann (eds.): Semi-Infinite Programming, in the series Nonconcex Optimization and its Applications. Kluwer Academic Publishers, 1998.

[23] J. L. Zhou, A. L. Tits, *Nonmonotone Line Search for Minimax Problems*, Journal of Optimization Theory and Applications, Vol. 76, No. 3, 1993, pp. 455 - 476.

[24] J. F. Bonnans, E. Panier, A. L. Tits, J. L. Zhou, *Avoiding the Maratos Effect by Means of a Nonmonotone Line search: II. Inequality Problems - Feasible Iterates*, SIAM Journal on Numerical Analysis, Vol. 29, No. 4, 1992, pp. 1187-1202.

[25] C. T. Lawrence, J. L. Zhou, A. L. Tits, *User's Guide for CFSQP Version 2.5: A C Code for Solving (Large Scale) Constrained Nonlinear (Minimax) Optimization Problems, Generating Iterates Satisfying all Inequality Constraints*, Institute for Systems Research, University of Maryland, Technical Report TR-94-16r1, 1997.

[26] *The FSQP Home page*, electronic document at http://www.isr.umd.edu/Labs/CACSE/FSQP/fsqp.html , maintained by the Institute for Systems Research, University of Maryland.

[27] H. J. Greenberg, *Mathematical Programming Glossary*, electronic document at http://www.cudenver.edu/~hgreenbe/glossary/glossary.html , 1999.

[28] *Optimization Frequently Asked Questions*, electronic document at http://www-unix.mcs.anl.gov/otc/Guide/faq/ , maintained by Robert Fourer, The Optimization Technology Center.

[29] W.H. Press, S.S. Teukolsky, V.T. Vetterling, B.P. Flannery, *Numerical Recipies in C – the Art of Scientific Computing*, Cambridge University Press, Cambridge, 1992.

[30] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[31] L. N. Trefethen, D. Bau, *Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[32] B. Jacob, *Linear Algebra*, W. H. Freeman and Company, New York, 1990.

[33] I. N. Bronstein, K. A. Smendljajew, G. Musiol, H. Mühlig, *Taschenbuch des Mathematik (second edition - in German)*, Verlag Harri Deutsch, Frankfurt am Main, 1995.

[34] I. Kuščer, A. Kodre, H. Neunzert, *Mathematik in Physik und Technik (in German)*, Springer - Verlag, Heidelberg, 1993.

[35] E. Kreyszig, *Advanced Engineering Mathematics (second edition)*, John Wiley & Sons, New York, 1993.

[36] Z. Bohte, *Numerične metode*, Društvo matematikov, fizikov in astronomov SRS, Ljubljana, 1987.

[37] K. J. Bathe, *Finite Element Procedures*, p.p. 697-745, Prentice Hall, New Jersey, 1996.

[38] F. van Keulen, V. V. Toropov, *Multipoint Approximations for Structural Optimization Problems with Noisy Response Functions*, electronic document at http://www-tm.wbmt.tudelft.nl/~wbtmavk/issmo/paper/mam_nois2.htm.

[39] J. F. Rodriguez, J. E. Renaud, *Convergence of Trust Region Augmented Lagrangian Methods Using Variable Fidelity Approximation Data*, In: WCSMO-2 : proceedings of the Second World Congress of Structural and Multidisciplinary Optimization, Zakopane, Poland, May 26-30, 1997. Vol. 1, Witold Gutkowski, Zenon Mroz (editors), 1st ed., Lublin, Poland, Wydawnictwo ekoincynieria (WE), 1997, pp. 149-154.