Structural Optimization 7, 1–19 © Springer-Verlag 1994

*Review Papers*

# Numerical comparison of nonlinear programming algorithms for structural optimization*

**K. Schittkowski**

Mathematisches Institut, Universität Bayreuth, D-95440 Bayreuth, Germany

**C. Zillober**

IWR, Universität Heidelberg, D-69120 Heidelberg, Germany

**R. Zotemantel**

CAP debis Division Industrie, D-80992 München, Germany

**Abstract** For FE-based structural optimization systems, a large variety of different numerical algorithms is available, e.g. sequential linear programming, sequential quadratic programming, convex approximation, generalized reduced gradient, multiplier, penalty or optimality criteria methods, and combinations of these approaches. The purpose of the paper is to present the numerical results of a comparative study of eleven mathematical programming codes which represent typical realizations of the mathematical methods mentioned. They are implemented in the structural optimization system MBB-LAGRANGE, which proceeds from a typical finite element analysis. The comparative results are obtained from a collection of 79 test problems. The majority of them are academic test cases, the others possess some practical *real life* background. Optimization is performed with respect to sizing of trusses and beams, wall thicknesses, etc., subject to stress, displacement, and many other constraints. Numerical comparison is based on reliability and efficiency measured by calculation time and number of analyses needed to reach a certain accuracy level.

## 1 Introduction

The design of a mechanical structure is often based on the requirement to *optimize* a suitable criterion to obtain a better design according to the criterion chosen, and to retain feasibility subject to the constraints that must be satisfied. The more complex the structure, the more difficult is the empirical iterative refinement *by hand* based on successive analysis.

In the last ten years, the finite element analysis of a large number of software systems was extended by optimization modules, see e.g. Hörnlein and Schittkowski (1993) for a review. In all cases, the underlying mechanical design problem is modelled and described in abstract terms, so that a mathematical nonlinear programming problem of the following form is formulated:

$$\min f(x),$$

$$x \in R^n : \; g_j(x) \geq 0, \quad j = 1, \ldots, m, \quad x_\ell \leq x \leq x_u. \qquad (1)$$

We may imagine, for example, that the objective function describes the weight of a structure that is to be minimized subject to sizing variables, and that the constraints impose limitations on structural response quantities, e.g. upper bounds for stresses or displacements under static loads. Many other objectives or constraints can be modelled in a way so that they fit into the above general frame.

Although permitted by most of the algorithms under investigation, we do not add equality constraints to the mathematical model. The structural design test problems used for the computational analysis, possess only inequality constraints in form of lower and/or upper bounds for some nonlinear functions.

Basically we can distinguish between two different classes of optimization methods. One class was developed independently from the special type of structural design application we are considering now, and can be specified as follows:

- sequential linear programming methods,
- penalty methods,
- multiplier methods,
- generalized reduced gradient methods, and
- sequential quadratic programming methods.

Each implementation of a method in one of these subclasses requires additional decisions on a special variant or parameter selections, so that different codes of the same group may have completely different performances in practice. Moreover, there exist combinations of the fundamental strategy making it even more difficult to classify nonlinear programming algorithms. Comparative studies of codes for the general model have been performed in the past (see, e.g. Colville 1968; Asaadi 1973; Sandgren 1977; Sandgren and Ragsdell 1982; Schittkowski 1980). They proceed either from randomly generated test examples (Schittkowski 1980), or are based on artificial or simple application problems of the kind described by Hock and Schittkowski (1981) and Schittkowski (1987).

The second class of numerical methods is more related to structural design optimization and basically consists of two subclasses,

- optimality criteria methods, and
- convex approximation methods.

*The research project was sponsored by the Deutsche Forschungsgemeinschaft under research contract DFG-Schi 173/6-1

Corresponding algorithms have been implemented and tested entirely for solving structural design optimization problems, and the codes are either unable or at least inefficient to act as general purpose optimizers. Comparative results are found, for example, in the papers by Eason and Fenton (1972, 1974) and Arora and Belegundu (1985).

Our intention is to report the results of an extensive comparative study of structural optimization codes, where the analysis is based on a finite element formulation and where we consider only the classical *sizing* situation, i.e. we do not test the performance of algorithms on shape or topology optimization problems.

The FE-analysis is performed by the software system MBB-LAGRANGE (see Kneppe *et al.* 1987; Zotemantel 1993). Apart from the nine optimization algorithms included in the official version, two additional methods are added to the system, i.e. certain variants of convex approximation methods. The codes represent all classes of algorithms mentioned above. Most of the methods have been developed, implemented, and tested outside of the MBB-LAGRANGE environment, and are taken over from external authors.

To conduct the numerical tests, 79 design problems have been collected. Most of them are *academic*, i.e. more or less simple design problems found in the literature. The remaining ones possess some practical *real life* background from project work or are suitable modifications to act as benchmark test problems for the development of the software system. In all situations, we minimize the weight of a structure subject to displacement, stress, strain, buckling, dynamic and other constraints. Design variables are sizing variables, e.g. cross-sectional areas of trusses and beams.

The purpose of the comparative tests is to evaluate efficiency and reliability of nonlinear programming algorithms when applied to structural optimization, in a quantitative manner. We count the number of problems solved with respect to a given final accuracy, and the corresponding calculation times and numbers of FE-analyses, i.e. function and gradient evaluations. Numerical performance is evaluated with respect to different accuracy levels. To be able to compare mean values with respect to different sets of test problems solved by a specific code subject to a given accuracy, a special priority theory is adapted (cf. Saaty 1980).

In Section 2 some features of the software system MBB-LAGRANGE are summarized and, in particular, the structural finite element model is outlined. Information on the basic idea behind the nonlinear programming algorithms and some details about the special implementation are presented in Sections 3 and 4. In Section 5 we describe the test problems and present a table of characteristic data. The test procedure and numerical results are summarized in Section 6, followed by a discussion and some conclusions in Section 7.

## 2 The structural optimization system MBB-LAGRANGE

MBB-LAGRANGE is a computer aided structural design system which allows the optimization of structural systems. It is based on the finite element technic and mathematical programming. The optimization model is characterized by the design variables and many different types of restrictions. The following design variables are available: element thicknesses, cross-sections, concentrated masses, and fiber angles. In addition to isotropic, orthotropic, and anisotropic applications, the analysis and optimization of composite structures is among the most important features. The design can be restricted with respect to statics (e.g. stresses, buckling), dynamics (natural frequencies, dynamic response) and aeroelastics (efficiencies, flutter speed). The general aim is to minimize the structural weight with MBB-LAGRANGE which has a wide range of optimization strategies and a modular architecture.

The objective function $f(x)$ is the weight of a structure, but it is also possible to take other linear objective functions into account. In addition it is possible to optimize problems with several objective functions, e.g. weight and stresses. Special multiobjective optimization techniques can be applied in these cases.

The design variables can be divided into three types.

1. Sizing variables, i.e.
   - cross-sectional areas for trusses and beams,
   - wall thicknesses for membrane and shell elements, and
   - laminate thicknesses for every single layer in composite elements.
2. Balance masses.
3. Angles of layers for composite elements.

Constraints must be defined in the form of inequality restrictions as follows:

$$g(x) := 1 - \frac{r_{\text{act}}(x)}{r_{\text{allow}}} \geq 0, \tag{2}$$

where $r$ denotes one of the allowed constraints, e.g. stress constraint. The constraints specify the feasible domain of the structure and allow realistic manufacturing, for example, by gage constraints. The choice of a suitable combination of constraints depends on the physical model. The following restrictions may be formulated in MBB-LAGRANGE:

- displacements
- stresses
- strains
- buckling (critical stresses, wrinkling)
- local compressive stresses
- aeroelastic efficiencies
- flutter speed
- natural frequencies
- dynamic responses
- eigenmodes
- weight
- bounds for the design variables (gages)

The objective function, the design variables, and the constraints describe the optimization model. The highly modular program architecture allows to distinguish between three main software concepts, namely

(i) optimization algorithm,
(ii) structural model (structural response and gradients), and
(iii) optimization model as the link between (i) and (ii).

The optimization model has some additional numerical features. All model functions and variables are scaled internally to stabilize the numerical algorithms. Moreover, the user is allowed to reduce the number of design variables by a procedure called *variable linking*, i.e. by linking certain structural variables into one design variable as implied by the structure itself, the loading conditions or manufacturing requirements. From the mathematical point of view a transformation of the form $x = a + At$ can be defined with a linking matrix $\mathbf{A}$, the structural variables $t$ and the design variables $x$. It is also possible to fix elements which means these values do not change during the optimization process.

The structural and sensitivity analyses are based on the finite element method. Modules for the following calculations are included:

- static
- (local) buckling
- natural frequencies
- dynamic responses (frequency, transient, random)
- (stationary) aeroelastic
- flutter

It is possible to treat homogeneous materials with isotropic, orthotropic and anisotropic behaviour as well as composite materials. The element library contains all important element types:

- truss elements
- beam elements
- membrane elements (triangle, quadrilateral)
- shell elements (triangle, quadrilateral)
- some special elements (e.g. spring elements)

Since the evaluation of gradients is the most expensive part of an optimization process, the efficient computation of derivatives is emphasized and three different ways of obtaining gradients are included in MBB-LAGRANGE; by

(i) numerical difference formulae,
(ii) analytical formulae, and
(iii) semi-analytical formulae.

The most efficient way is to derive analytical formulae for the sensitivity analysis. In sizing problems the derivatives with respect to design variables are analysed and implemented directly, an essential assumption for solving large scale problems. For geometry variables, however, a semi-analytical formula was used to obtain gradients, see the paper by Hörnlein (1986) for details.

MBB-LAGRANGE has some special features for space and aircraft design, which is considered to be the main domain of application. Therefore, it is essential to allow aeroelastic and flutter calculations, including the corresponding constraint formulation. A wide range of dynamic functions is also available. For buckling problems it is possible to handle isotropic and composite materials, also local stability of sandwich structures (wrinkling). In some cases the so-called system identification is useful, i.e. the evaluation of the location of model imperfection by taking measured data from modal tests. A powerful way to reduce the weight in composite structures is to define layer angles as well as layer thicknesses as design variables (varying of layer angles).

## 3 Mathematical optimization strategies

In this section we outline the mathematical methods behind the nonlinear programming codes of the MBB-LAGRANGE system, which are used for performing the subsequent numerical tests. Since the mathematical background is found in text books and references cited (see, e.g. Gill *et al.* 1981; Papalambros and Wilde 1988), we give only a very brief outline on the basic methodology.

To simplify the notation, we omit a separate treatment of upper and lower bounds for the design variables in this section. They can be now considered as part of the general inequality constraints, but are handled separately in the numerical codes discussed in the subsequent section.

The most important tool for understanding the optimization, is the so-called Lagrange function

$$L(x, u) := f(x) - \sum_{j=1}^{m} u_j g_j(x)\,,$$

which is defined for $x \in \mathbb{R}^n$ and $u = (u_1, \ldots, u_m)^T$, and which describes a linear combination of the objective function and the constraints. The coefficients $u_j, j = 1, \ldots, m$, are called the Lagrange multipliers of problem (1).

Now we are able to formulate optimality criteria, which are needed to understand the methods to be described. To be able to formulate necessary conditions, we need an assumption called *constraint qualification* which means that for a feasible $x$, the gradients of active constraints, i.e. the set $\{\nabla g_j(x) : g_j(x) = 0\}$, are linearly independent.

**Theorem:** *Let $f$ and $g_j$ for $j = 1, \ldots, m$ be twice continuously differentiable functions, $x^*$ be a local minimizer of (1) and the constraint qualification be satisfied in $x^*$. Then there is a $u^* \in \mathbb{R}^m$ so that the following conditions are satisfied:*

(a) $u_j^* \geq 0$ for $j = 1, \ldots, m$, $\quad \nabla_x L(x^*, u^*) = 0$,
$u_j^* g_j(x^*) = 0$ for $j = 1, \ldots, m$,

(b) $s^T \nabla_x^2 L(x^*, u^*) s \geq 0$ for all $s \in \mathbb{R}^n$,
with $\nabla g_j(x^*)^T s = 0$ and $g_j(x^*) = 0$.

The condition, that the gradient of the Lagrange function vanishes at an optimal solution is called the *Kuhn-Tucker-condition* of (1). In other words, the gradient of $f$ is a linear combination of gradients of active constraints

$$\nabla f(x^*) = \sum_{i=1}^{m} u_j^* \nabla g_j(x^*)\,. \tag{3}$$

The complementary slackness condition $u_j^* g_j(x^*) = 0$ together with the feasibility of $x^*$ guarantees, that only the active constraints, i.e. the interesting ones, contribute a gradient in the above sum. Either a constraint is satisfied by equality or the corresponding multiplier value is zero.

The Kuhn-Tucker condition can be computed within an optimization algorithm, if suitable multiplier estimates are available, and serves as a stopping condition. However, the second order condition (b) can be evaluated numerically only if second derivatives are available. The condition is required in the optimality criteria to be able to distinguish between a stationary point and a local minimizer.

### 3.1 Optimality criteria methods

For the optimal design of structures there exist a couple of algorithms that are based on the above conditions and are therefore called the optimality criteria methods in engineering sciences (Berke and Khot 1974).

One of the approaches developed in the past is called *the stress ratio method* which is applicable to problems with stress constraints only. In this case we have separable constraint functions of the form

$$g_j(x) = \bar{s}_j - s_j(x_j)\,,$$

where $s_j$ denotes the stress in the $j$-element and $\bar{s}_j$ an upper bound. In the case of a statically determined structure, all these constraints are active, leading to a trivial solution of the optimization problem.

The technique is extended easily to the case of multiple load cases, additional bounds for design variables and non-determined structures.

### 3.2 Penalty methods

Penalty methods belong to the first attempts to solve constrained optimization problems satisfactorily. The basic idea is to construct a sequence of unconstrained optimization problems and to solve them by any standard minimization method, so that the minimizers of the unconstrained problems converge to the solution of the constrained one.

To construct the unconstrained problems, so-called penalty terms are added to the objective function which *penalize* $f(x)$ whenever the feasible region is left. A factor $r_k$ controls the degree of penalizing $f$. Proceeding from a sequence $\{r_k\}$ with $r_k \to \infty$ for $k = 0, 1, 2, \ldots$, penalty functions can be defined, for example, by

$$p_e(x, r) := f(x) + r_k \sum_{i=1}^{m} \{\min[0, g_j(x)]\}^2\,, \tag{4}$$

or

$$p_b(x, r) := f(x) + \frac{1}{r_k} \sum_{i=1}^{m} \log[g_j(x)]\,, \tag{5}$$

or

$$p_i(x, r) := f(x) + \frac{1}{r_k} \sum_{i=1}^{m} \frac{1}{g_j(x)}\,. \tag{6}$$

The first penalty function allows violation of constraints and is called an external one. The subsequent two are barrier functions, i.e. they can only work with feasible iterates.

The unconstrained nonlinear programming problems are solved by any standard technique, e.g. a quasi-Newton search direction combined with a line search. However, the line search must be performed quite accurately due to the steep, narrow valleys created by the penalty terms.

There exists a large variety of other proposals and combinations of them (e.g. Fiacco and McCormick 1968; Lootsma 1971). The main disadvantage of penalty type methods is that the condition number of the Hessian matrix of the penalty function increases when the parameter $r_k$ becomes too large (Murray 1967). It should be noted, however, that penalty methods became quite attractive again in recent years either in connection with Karmarkar-type interior point methods (e.g. Powell 1992), or with second derivatives (e.g. Broyden and Attia 1988).

### 3.3 Multiplier methods

Multiplier or augmented Lagrangian methods try to avoid the disadvantage of penalty algorithms, i.e. that too large penalty parameters lead to ill-conditioned unconstrained subproblems. Thus the objective function is augmented by a term including information about the Lagrangian function.

One of the first proposals was made by Powell (1969) and later extended by Fletcher (1975) to inequality constraints

$$\psi_r(x, v) := f(x) + \frac{1}{2} \sum_{i=1}^{m} r_j [g_j(x) - v_j]_-^2\,, \tag{7}$$

where $a_- := \min(0, a)$ for $a \in \mathbb{R}$, $v \in \mathbb{R}^m$, and $r \in \mathbb{R}^m$. Multipliers are approximated by $r_j v_j$.

A similar augmented Lagrangian function was proposed by Hestenes (1969) for equality and by Rockafellar (1974) for inequality constraints,

$$\phi_r(x, v) := f(x) - \sum_{j=1}^{m} \begin{cases} [v_j g_j(x) - \frac{1}{2} r_j g_j(x)^2], & \text{if } g_j(x) \le v_j / r_j \\ \frac{1}{2} v_j^2 / r_j, & \text{otherwise} \end{cases} \tag{8}$$

After solving an unconstrained minimization problem with one of the above objective functions, the muliplier estimates are updated according to certain rules, for example, by

$$\bar{v}_j := v_j - \min[g_j(x), v_j]\,,$$

in the first case or

$$\bar{v}_j := v_j - \min[r_j g_j(x), v_j],$$

in the second case for $j = 1, \ldots, m$. If there is no sufficient reduction of the constraint violation, then the penalty parameter vector is increased as well, typically by a constant factor. More details are found in the literature cited, or in the work by Pierre and Lowe (1975), Schuldt (1975) or Bertsekas (1976).

The unconstrained subproblems are solved more or less in the same way as in penalty methods. A search direction is computed successively by a quasi-Newton technique, and a one-dimensional line search is performed, until convergence criteria are satisfied.

### 3.4 Sequential linear programming methods

Particularly for design optimization, sequential linear programming or SLP methods, are quite powerful due to the special problem structure and, in particular, due to numerical limitations that prevent the usage of higher order methods in some cases. The idea is to approximate the nonlinear problem (1) by a linear one to obtain a new iterate. Thus the next iterate $x_{k+1} = x_k + d_k$ is formulated with respect to solution $d_k$ of the following linear programming problem:

$$\begin{aligned} &\min \nabla f(x_k)^T d\,, \\ d \in \mathbb{R}^n :\ &\nabla g_j(x_k)^T d + g_j(x_k) \ge 0\,, \quad j = 1, \ldots, m\,, \\ &\| d \| \le \delta_k\,. \end{aligned} \tag{9}$$

The principle advantage is that the above problem can be solved by any standard linear programming software. Additional bounds for the computation of $d_k$ are required to avoid bad estimates particularly at the beginning of the algorithm,

when the linearization is too inaccurate. The bound $\delta_k$ must be adapted during the algorithm. One possible way is to consider the so-called exact penalty function

$$p(x, r) := f(x) + \sum_{i=1}^{m} r_j |\min[0, g_j(x)]|, \tag{10}$$

defined for each $x \in \mathbb{R}^n$ and $r = (r_1, \ldots, r_m)^T$. Moreover, we need its first order Taylor approximation given by

$$p_a(x, d, r) := f(x) + \nabla f(x)^T d + \sum_{i=1}^{m} r_j |\min[0, g_j(x) + \nabla g_j(x)^T d]|. \tag{11}$$

Then we consider the quotient of the actual and predicted change at an iterate $x_k$ and a solution $d_k$ of the linear programming subproblem

$$q_k := \frac{p(x_k, r) - p(x_k + d_k, r)}{p(x_k, r) - p_a(x_k, d_k, r)},$$

where the penalty parameters are predetermined and must be sufficiently large, e.g. larger than the expected multiplier values at an optimal solution. The $\delta_k$-update is then performed by

$$\delta_{k+1} := \begin{cases} \delta_k/\sigma, & \text{if } q_k < \rho_1 \\ \delta_k \sigma, & \text{if } q_k > \rho_2 \\ \delta_k, & \text{otherwise} \end{cases}.$$

Here $\sigma > 1$ and $0 < \rho_1 < \rho_2 < 1$ are constant numbers. Some additional safeguards are necessary to be able to prove convergence (e.g. Lasdon *et al.* 1983; Fletcher and de la Maza 1987).

### *3.5 Sequential quadratic programming methods*

Sequential quadratic programming or SQP methods are the standard general purpose algorithms for solving smooth nonlinear optimization problems under the following assumptions.

- The problem is not too big.
- The functions and gradients can be evaluated with sufficiently high precision.
- The problem is smooth and well-scaled.

The mathematical convergence and the numerical performance properties of SQP methods are very well understood now and have been published in so many papers that only a few can be mentioned here [see, e.g. Stoer (1985) or Spellucci (1993) for a review]. Theoretical convergence has been investigated by Han (1976, 1977), Powell (1978a, 1978b), Schittkowski (1983), for example, and the numerical comparative studies of Schittkowski (1980) and Hock and Schittkowski (1981) show their superiority over other mathematical programming algorithms under the above assumptions.

The key idea is to approximate also second-order information to obtain a fast final convergence speed. Thus we define a quadratic approximation of the Lagrangian function $L(x, u)$ and an approximation of the Hessian matrix $\nabla_x^2 L(x_k, u_k)$ by a so-called quasi-Newton matrix $\mathbf{B}_k$. Then we have the subproblem

$$\min \frac{1}{2} d^T B_k d + \nabla f(x_k)^T d,$$

$$d \in \mathbb{R}^n : \ \nabla g_j(x_k)^T d + g_j(x_k) \geq 0, \quad j = 1, \ldots, m. \tag{12}$$

Instead of trust regions or move limits, respectively, as for SLP methods, the convergence is ensured by performing a line search, i.e. a step length computation to accept a new iterate $x_{k+1} := x_k + \alpha_k d_k$ for an $\alpha_k \in [0, 1]$ only if $x_{k+1}$ satisfies a descent property with respect to a solution $d_k$ of (12). Following the approach of Schittkowski (1983), for example, we need also a simultaneous line search with respect to the multiplier approximations called $v_k$ and define $v_{k+1} := v_k + \alpha_k(u_k - v_k)$ where $u_k$ denotes the optimal Lagrange multiplier of the quadratic programming subproblem (12).

The line search is performed with respect to a merit function

$$\psi_k(\alpha) := \phi_{r_k}[x_k + \alpha d_k, v_k + \alpha(u_k - v_k)],$$

where $\phi_r(x, v)$ is a suitable exact penalty or augmented Lagrangian function, for example, of the type (10) or (8), respectively.

We should note here that also other concepts, i.e. other merit functions are found in the literature. Then we initiate a subiteration starting with $\alpha = 1$ and perform a successive reduction combined with a quadratic interpolation of $\psi_k(\alpha)$ until, for the first time, a stopping condition of the form

$$\psi_k(\alpha) \leq \psi_k(0) + \mu\alpha\psi_k'(0),$$

is satisfied, where we must be sure that $\psi_k'(0) < 0$, of course. To guarantee this condition, the penalty parameter $r_k$ must be evaluated by a special formula which is not repeated here.

The update of the matrix $\mathbf{B}_k$ can be performed by standard techniques known from unconstrained optimization. In most cases, the BFGS-method is applied, a numerically simple rank-2 correction starting from the identity or any other positive definite matrix. Only the differences $x_{k+1} - x_k$, $\nabla_x L(x_{k+1}, u_k) - \nabla_x L(x_k, u_k)$ are required. Under some safeguards it is possible to guarantee that all matrices $\mathbf{B}_k$ are positive definite.

One of the most attractive features of SQP methods is the superlinear convergence speed in the neighbourhood of a solution given by

$$\| x_{k+1} - x^* \| \leq \gamma_k \| x_k - x^* \|,$$

where $\gamma_k$ is a sequence of positive numbers converging to zero and $x^*$ an optimal solution.

### *3.6 Generalized reduced gradient methods*

By introducing artificial slack variables, the original nonlinear programming problem is converted easily into a problem with nonlinear equality constraints und lower bounds for the slack variables only. Thus we proceed from a slightly more general problem

$$\min \overline{f}(z),$$

$$z \in \mathbb{R}^{\overline{n}} : \ \overline{g}_j(z) = 0, \quad j = 1, \ldots, m,$$

$$z_\ell \leq z \leq z_u, \tag{13}$$

where $z := (x, y)$, $\overline{n} = n + m$, $\overline{f}(z) := f(x)$, $\overline{g}_j(z) = g_j(x) - y_j$; for $j = 1, \ldots, m$.

As in linear programming, variables $z$ are classified into basic and non-basic ones (Wolfe 1967). In our situation we can use $y$ for the initial basic and $x$ for the initial non-basic variables. By now defining

$$\overline{g}(z) := [\overline{g}_1(z), \ldots, \overline{g}_m(z)]^T ,$$

we try to satisfy the system of equations $\overline{g}(z) = 0$ for all possible iterates. Let $y(x)$ be a solution of this system with respect to given variables $x$, i.e. $\overline{g}[x, y(x)] = 0$, and let $F(x) := \overline{f}[x, y(x)]$ be the so-called reduced objective function.

Starting from a feasible iterate and an initial set of basic variables, the algorithm performs a search step with respect to the free variables, for example, by a conjugate gradient or a quasi-Newton method. If the new iterate violates constraints, then it will be projected onto the feasible domain by a Newton-type technique. If necessary, a line search is performed also combined with a restoration phase to obtain feasible iterates.

When changing an iterate it might happen that a basic variable $y$ violates a bound. In this case the corresponding variable leaves the basic and another one enters it.

For evaluating a search direction in the reduced space, we need the gradient of the reduced objective function $F(x)$ with respect to the non-basic variables $x$, which is computed from

$$\nabla F(x) =$$

$$\nabla_x \overline{f}[x, y(x)] - \nabla_x \overline{g}[x, y(x)]^T \nabla_y g[x, y(x)]^{-T} \nabla_y \overline{f}[x, y(x)] .$$

The situation is more complicated, when we have to consider also bounds for the non-basic variables. For details see the work of Abadie (1978), Lasdon and Waren (1978), Lasden *et al.* (1978) or Schittkowski (1986). Generalized reduced gradient methods can be easily extended to problems with special structure in the constraints or very large problems. Moreover, they are related to sequential quadratic programming methods and there exist combinations of both approaches (Parkinson and Wilson 1986; Schittkowski 1985a, b). The last papers also outline the relationship to penalty and multiplier methods.

### *3.7 Sequential convex programming methods*

Sequential convex programming or convex approximation (CA) methods, respectively, have been developed in particular by Fleury (1979, 1986), and Svanberg (1987) extended this approach. Their key motivation was to implement an algorithm that is particularly designed for solving mechanical structural optimization problems. Thus their domain of application is somewhat restricted to a special problem type.

The key idea is to use a convex approximation of the original problem (1) instead of a linear or quadratic one, and then to solve the resulting nonlinear subproblem by a specifically designed algorithm that takes advantage of the simplified problem structure. Consequently CA methods are only useful in cases where the evaluation of function and gradient values is much more expensive than the internal computations to solve the subproblem.

Let us consider e.g. the objective function $f(x)$. By inverting suitable variables, we obtain the convex approximation of $f(x)$ in the neighbourhood of an $x_k \in I\!R^n$ by

$$f_k(x) := f(x_k) + \sum_{i \in I_k^+} \frac{\partial}{\partial x_i} f(x_k)(x_i - x_i^k) - \sum_{i \in I_k^-} \frac{\partial}{\partial x_i} f(x_k)(1/x_i - 1/x_i^k)(x_i^k)^2 , \qquad (14)$$

where $x = (x_1, \ldots, x_n)^T$ and $x_k = (x_1^k, \ldots, x_n^k)^T$ and where

$$I_k^- := \left\{ i : 1 \le i \le n, \ \frac{\partial}{\partial x_i} f(x_k) \le 0 \right\} ,$$

$$I_k^+ := \left\{ i : 1 \le i \le n, \ \frac{\partial}{\partial x_i} f(x_k) > 0 \right\} .$$

The reason for inverting design variables in the above way is that stresses and displacements are exact linear functions of the reciprocal linear homogeneous sizing variables in the case of a statically determined structure. Moreover, numerical experience shows that also in other cases, convex linearization is applied quite successfully in practice, in particular in shape optimization, although a mathematical motivation cannot be given in this case.

In a similar way, reciprocal variables are introduced for the inequality constraints, where we have to change the signs to obtain a concave function approximation, but, on the other hand, a convex feasible region of the subproblem. The corresponding index sets are denoted by $I_{k,j}^+$ and $I_{k,j}^-$ for $j = 1, \ldots, m$.

After some reorganisation of constant data, we obtain a convex subproblem of the following form:

$$\min \sum_{i \in I_k^+} f_i^k x_i - \sum_{i \in I_k^-} f_i^k / x_i ,$$

$$x \in I\!R^n : \sum_{i \in I_{k,j}^-} g_{i,j}^k x_i - \sum_{i \in I_{k,j}^+} g_{i,j}^k / x_i + \overline{g}_{i,j}^k \ge 0 ,$$

$$j = 1, \ldots, m , \qquad (15)$$

where $f_k^i$ and $g_k^i$ are the parameters of the convex approximation (14) with respect to objective function and constraints.

The solution to the above problem then determines the next iterate $x_{k+1}$. We do not investigate here the question how the mathematical structure of the subproblem can be exploited to obtain an efficient solution algorithm for solving. As long as the problem is not too big, we may assume without loss of generality that (15) is solved by any standard nonlinear programming technique.

To control the degree of convexification and to adjust it with respect to the problem to be solved, Svanberg (1987) introduced so-called moving asymptotes $U_i$ and $L_i$ to replace $x_i$ and $1/x_i$ by

$$\frac{1}{x_i - L_i} , \quad \frac{1}{U_i - x_i} ,$$

where $L_i$ and $U_i$ are given parameters, which can also be adjusted from one iteration to the next. The algorithm is called the method of moving asymptotes. The larger flexibility allows a better convex approximation of the problem and thus a more efficient and robust solution.

Numerical experience shows that both variants are very efficient. However, there are no additional safeguards to stabilize the algorithm as, for example, done for sequential quadratic programming methods. When starting from an inappropriate initial design, it may happen that the algorithm as described above, does not converge.

To overcome this drawback, Zillober (1993a) added a line search procedure to the standard convex approximation method, similar to the approach used in sequential quadratic programming. In this case, it is possible to prove a global convergence theorem based on very weak assumptions.

## 4 Nonlinear programming codes

One of the reasons for using the software system MBB-LAGRANGE for the FE-analysis, was the highly modular program architecture facilitating the inclusion of new optimization algorithms. In addition to nine nonlinear programming codes that are part of the official system, two further codes have been added to MBB-LAGRANGE for the purpose of this comparative study of variations of convex approximation methods.

By the subsequent comments, some additional features of the algorithms and special implementation details are outlined. To identify the optimization codes we take over the notation of the MBB-LAGRANGE documentation.

SRM: The stress ratio code belongs to the class of optimality criteria methods and is motivated by statically determined structures. The algorithm is applicable to problems only with stress constraints, consists of a simple update formula for the design variables, and does not need any gradient information.

IBF: The inverse barrier function method is an implementation of a penalty method as described in the previous section, subject to the penalty function (6). Thus one needs a feasible design to start the algorithm. The unconstrained minimization is performed with respect to a quasi-Newton update (BFGS) and an Hermite interpolation procedure for the line search. It is recommended to perform only a relatively small number of iterations, e.g. 5 or 10, and to start another cycle by increasing the penalty parameter through a constant factor.

MOM: Proceeding from the same unconstrained optimization routine as IBF, a sequential unconstrained minimization technique is applied. The method of multipliers uses the augmented Lagrangian function (8) for the subproblem and the corresponding update rules for the multipliers. Both methods, i.e. IBF and MOM, have a special advantage when evaluating gradients of the objective function in the subproblem. The inverse of the stiffness matrix obtained by a decomposition technique is multiplied only once with the remaining part of the gradient, not in each restriction as required for most of the subsequent methods.

SLP: The sequential linear programming method was implemented by Kneppe (1985). The linear subproblem is solved by a simplex method. So-called move limits are introduced to prevent cycling and iterates too far away from the feasible area. They are reduced in each iteration by the formula $\delta_{k+1} = \delta_k/(1+\delta_k)$ and an additional cubic line search is performed as soon as cycling is observed.

RQP1: The first recursive or sequential quadratic programming code is the subroutine NLPQL of Schittkowski (1985, 1986). Subproblems are solved by a dual algorithm based on a routine written by Powell (1983). The augmented Lagrangian function (8) serves as a merit function and BFGS-updates are used for the quasi-Newton formula. The special implementation of NLPQL is capable of solving also problems with very many constraints (Schittkowski 1992), and is implemented in MBB-LAGRANGE in reverse communication. The idea is to save as much working memory as possible by writing optimization data on a file during the analysis, and by saving analysis data during an optimization cycle.

RQP2: This is the original sequential quadratic programming code VMCWD of Powell (1978a) with the merit function (10). Also in this case, the BFGS-update is used internally together with a suitable modification of the penalty parameter.

GRG: The generalized reduced gradient code was implemented by Bremicker (1986). During the line search an extrapolation is performed to follow the boundary of active constraints closer. The Newton-algorithm for projecting non-feasible iterates during the line search onto the feasible domain, uses the derivative matrix for the very first step. Subsequently a rank-1-quasi-Newton formula of Broyden is updated.

QPRLT: To exploit the advantages of SQP and GRG methods, a hybrid method was implemented by Sömer (1987). Starting from a feasible design, a search direction is evaluated by the SQP-approach, i.e. by solving a quadratic programming subproblem. This direction is then divided into basic and non-basic variables, and a line search very similar to the generalized reduced gradient method GRG is performed.

CONLIN: This is the original implementation of Fleury (1989), where a convex and separable subproblem is generated as outlined in Section 3. In particular, only variables belonging to negative partial derivatives are inverted. The nonlinear subproblem is solved by a special dual method.

SCP: The sequential convex programming method was implemented by Zillober (1993b) and added to the MBB-LAGRANGE-system for the purpose of this comparative study. The algorithm uses moving asymptotes and a line search procedure for stabilization with respect to the merit function (8).

MMA: The code is a reimplementation of the original convex approximation method of Svanberg (1987) with moving asymptotes. As for CONLIN and SCP, the subproblems are solved by a special dual approach. The adaption of moving asymptotes is described by Zillober (1993b).

## 5 Test problems

The success of any comparative numerical study of optimization codes depends mainly on the quality of test problems. In the present case all results have been obtained by 79 test examples, which were collected over the years of the development of the structural optimization system MBB-LAGRANGE.

Thus there was no basic strategy or any selection rule to find *good* design examples. Most of the problems (about 70%) are more or less academic problems, some are found in the literature, some represent real life structures. In fact the 79 test examples do not represent independent problems, since some of them are variants of one and the same underlying design model. There exist 50 different, independent models as marked in the second column in Table 1. Most test cases are modifications of academic or real life problems with the intention to test certain options of the program code of MBB-LAGRANGE. Moreover, since all test examples are to be solvable by all available optimization algorithms, the structure size, i.e. number of elements and degrees of freedom, is relatively small compared to real life applications.

To describe some characteristic properties of the problems we distinguish between five categories (the following numeration refers to the group numbers).

(1) There are seven real life structures or models derived from them, from the design of space and aircrafts.

- A bracket link as a small part of the Ariane spacecraft (no. 2), a structure with several load cases and stress contraints.
- The swept wing of a Boeing aircraft as a composite structure (no. 3), without and with layer angle variation.
- A coolplate of a space structure (no. 13), with stress constraints and one lower frequency bound.
- The tank floor of a combat aircraft (no. 17).
- A center spar of the training aircraft JPATS (no. 24), a composite structure with failure criteria constraints.
- The propulsion module of a satellite structure (no. 32), a model only with frequency constraints and a special mode control option, with the intention to find a feasible design.
- A bulkhead of a combat aircraft (no. 41), i.e. a structure sensitive to buckling, the so-called Grumman wingbox (no. 20).

(2) Eight examples are taken from the literature or other publications. These are a simple plate supported at four points (no. 1) (Fleury *et al.* 1984), the Boeing swept wing (no. 3), various publications, a three-bar truss (no. 15), a cantilever plate with different constraints in dynamics (no. 16), see NASTRAN-manual (1985), a plane frame work (no. 19) as an example for system identification (ESA model), the supporting beam of a crane structure (no. 27) (Schwarz 1981), and the wellknown ten-bar truss (no. 43).

(3) Most problems are test cases to verify certain aspects of the analysis, e.g. element types. Some examples of this group of 40 problems are to test bar elements (no. 4, 5, 6), shell elements (no. 1, 13, 20, 31, 47, 49), solid elements (no. 23, 42), the buckling analysis (no. 37, 48, 50), multipoint constraints (no. 30), different coordinate systems (no. 2, 14), and special elements (no. 11, 21, 38).

(4) Other problems check MBB-LAGRANGE with respect to the optimization model, i.e. constraints and design variables. There are examples for layer angle variation (no. 3, 39), point masses as design variables (no. 25, 34), buckling constraints (no. 18, 37, 48, 50), wrinkling constraints (no. 18), local stress constraints (no. 9), frequency response constraints (no. 5, 16, 46), time response constraints (no. 16), and manufacturing constraints (no. 12).

(5) The remaining examples are developed to test the different optimization strategies in MBB-LAGRANGE (nos. 1, 15, 26, 31, 43).

We believe that the present set of test cases is representative at least for small or medium size structural designs. It is also important to note that we do not want to test the analysis part of an FE-system. Instead, the response of optimization routines when applied to solve structural optimization problems is to be investigated.

In the subsequent tables, we classify some characteristic data of the test structures under investigation. For reference reasons, we use the original notation as determined by the engineers implementing and testing MBB-LAGRANGE.

Table 1 gives an impression of the size of the analysis and optimization model, and presents the following information:

**Table 1.** Information on model structure

| No. | Model | Test example | NET | NOD | NLY | NLC | NDT | NSV | NDV | NDG |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | APLATE2 | 49 | 64 | 1 | 1 | 175 | 49 | 7 | 51 |
| 2 | 1 | APLCON | 49 | 64 | 1 | 1 | 175 | 49 | 32 | 51 |
| 3 | 2 | ARIANB | 310 | 357 | 1 | 5 | 2136 | 310 | 24 | 1020 |
| 4 | 3 | B4SIZE | 130 | 88 | 4 | 1 | 240 | 520 | 72 | 522 |
| 5 | 3 | B4SIZELA | 130 | 88 | 4 | 1 | 240 | 520 | 83 | 522 |
| 6 | 4 | BALKEN2 | 2 | 4 | 1 | 2 | 10 | 2 | 2 | 4 |
| 7 | 5 | BARBIG | 9 | 11 | 1 | 0 | 54 | 9 | 9 | 91 |
| 8 | 6 | BARDISP | 3 | 4 | 1 | 1 | 12 | 3 | 2 | 3 |
| 9 | 6 | BARDYN2 | 3 | 4 | 1 | 1 | 12 | 3 | 3 | 4 |
| 10 | 6 | BARDYN5 | 3 | 4 | 1 | 1 | 12 | 3 | 3 | 4 |
| 11 | 7 | BARMASS | 8 | 10 | 1 | 1 | 48 | 8 | 8 | 1 |
| 12 | 8 | BAROF9 | 7 | 4 | 1 | 1 | 18 | 7 | 7 | 7 |
| 13 | 9 | BUST | 8 | 15 | 1 | 2 | 54 | 8 | 8 | 16 |
| 14 | 9 | BUSTB | 8 | 9 | 1 | 2 | 30 | 8 | 8 | 16 |
| 15 | 9 | BUSTM | 4 | 6 | 1 | 2 | 8 | 4 | 4 | 8 |
| 16 | 9 | BUSTQ | 8 | 15 | 1 | 2 | 54 | 8 | 8 | 16 |
| 17 | 9 | BUSTQM | 2 | 6 | 1 | 2 | 8 | 2 | 2 | 4 |
| 18 | 9 | BUSTT | 10 | 15 | 1 | 2 | 54 | 10 | 10 | 20 |
| 19 | 10 | CANE | 40 | 44 | 1 | 1 | 120 | 40 | 10 | 41 |
| 20 | 11 | CELAS4 | 22 | 29 | 1 | 1 | 87 | 18 | 2 | 18 |
| 21 | 12 | COMPKRAG | 4 | 7 | 4 | 2 | 10 | 16 | 10 | 44 |
| 22 | 13 | COOLPLATE | 177 | 180 | 1 | 1 | 971 | 177 | 8 | 177 |
| 23 | 14 | CORDS | 63 | 35 | 4 | 1 | 87 | 135 | 8 | 96 |
| 24 | 15 | DREI | 3 | 4 | 1 | 2 | 2 | 3 | 3 | 6 |
| 25 | 15 | DREIDISP | 3 | 4 | 1 | 2 | 2 | 3 | 2 | 6 |
| 26 | 15 | DREISLP | 3 | 4 | 1 | 2 | 2 | 3 | 3 | 6 |
| 27 | 16 | DYNPLT | 63 | 60 | 1 | 1 | 250 | 63 | 10 | 157 |
| 28 | 17 | EFA2CLAG | 492 | 385 | 1 | 1 | 1859 | 492 | 54 | 153 |
| 29 | 18 | F1F3 | 24 | 28 | 9 | 1 | 52 | 100 | 13 | 67 |
| 30 | 19 | GARTEUR1 | 83 | 79 | 1 | 0 | 231 | 83 | 13 | 21 |
| 31 | 20 | GBOX | 97 | 89 | 1 | 1 | 473 | 97 | 6 | 98 |
| 32 | 21 | GENTEST | 3 | 4 | 1 | 2 | 18 | 1 | 1 | 2 |
| 33 | 22 | GRADPLA | 4 | 9 | 1 | 2 | 41 | 4 | 4 | 10 |
| 34 | 23 | HEXA | 9 | 52 | 1 | 1 | 148 | 9 | 6 | 13 |
| 35 | 24 | JPATS16 | 252 | 214 | 2 | 1 | 406 | 744 | 144 | 416 |
| 36 | 25 | KALA3_1 | 9 | 9 | 4 | 1 | 18 | 75 | 37 | 40 |
| 37 | 26 | KRAG | 4 | 7 | 3 | 2 | 10 | 7 | 7 | 14 |
| 38 | 26 | KRAGBA | 8 | 10 | 4 | 2 | 16 | 12 | 10 | 26 |
| 39 | 26 | KRAGBAD | 8 | 10 | 4 | 2 | 16 | 12 | 10 | 25 |
| 40 | 26 | KRAGBAM | 8 | 10 | 4 | 2 | 16 | 12 | 10 | 20 |
| 41 | 26 | KRAGDYN | 8 | 10 | 4 | 2 | 16 | 12 | 8 | 27 |
| 42 | 26 | KRAGIBF | 4 | 7 | 3 | 2 | 10 | 7 | 7 | 14 |

**Table 1.** Continued

| No. | Model | Test example | NET | NOD | NLY | NLC | NDT | NSV | NDV | NDG |
|---|---|---|---|---|---|---|---|---|---|---|
| 43 | 26 | KRAGMAN | 4 | 7 | 4 | 2 | 10 | 16 | 10 | 44 |
| 44 | 26 | KRAGMOM | 4 | 7 | 3 | 2 | 10 | 7 | 7 | 14 |
| 45 | 27 | KRAN0 | 54 | 20 | 1 | 1 | 112 | 54 | 54 | 54 |
| 46 | 27 | KRAN1 | 54 | 20 | 1 | 1 | 112 | 54 | 54 | 55 |
| 47 | 28 | LAGTEST | 40 | 36 | 1 | 1 | 158 | 40 | 8 | 40 |
| 48 | 10 | MCANE | 40 | 44 | 1 | 1 | 120 | 40 | 40 | 41 |
| 49 | 29 | MOMENT | 8 | 9 | 1 | 2 | 48 | 8 | 8 | 16 |
| 50 | 30 | MPC | 18 | 29 | 1 | 1 | 113 | 18 | 2 | 18 |
| 51 | 31 | MPLATE | 6 | 10 | 1 | 2 | 26 | 6 | 6 | 12 |
| 52 | 31 | MPLATEH | 6 | 10 | 1 | 2 | 26 | 6 | 6 | 12 |
| 53 | 32 | OPM | 208 | 133 | 1 | 0 | 516 | 208 | 11 | 2 |
| 54 | 33 | PLATTE | 10 | 16 | 4 | 1 | 27 | 40 | 10 | 41 |
| 55 | 34 | POINTMASS | 8 | 10 | 1 | 0 | 48 | 8 | 8 | 1 |
| 56 | 35 | PUNCHTEST | 50 | 31 | 4 | 1 | 60 | 86 | 20 | 86 |
| 57 | 36 | QUARTOPLT | 4 | 9 | 1 | 4 | 10 | 4 | 4 | 20 |
| 58 | 37 | QUATRIA | 39 | 49 | 8 | 1 | 84 | 92 | 35 | 99 |
| 59 | 38 | RBAR | 25 | 29 | 1 | 1 | 134 | 25 | 2 | 25 |
| 60 | 39 | SCHEIT210 | 8 | 9 | 4 | 1 | 15 | 32 | 4 | 32 |
| 61 | 40 | SPANT | 20 | 15 | 1 | 1 | 26 | 20 | 13 | 20 |
| 62 | 41 | SPSLP | 414 | 196 | 1 | 1 | 389 | 414 | 108 | 414 |
| 63 | 41 | SPSLPB | 414 | 196 | 1 | 1 | 389 | 414 | 108 | 415 |
| 64 | 42 | T3D066 | 9 | 52 | 1 | 1 | 148 | 9 | 6 | 13 |
| 65 | 43 | TBDYN | 10 | 6 | 1 | 0 | 8 | 10 | 10 | 1 |
| 66 | 44 | TEMPER2 | 16 | 25 | 1 | 1 | 18 | 16 | 12 | 12 |
| 67 | 43 | TENBAR | 10 | 6 | 1 | 1 | 8 | 10 | 10 | 10 |
| 68 | 43 | TENCON | 10 | 6 | 1 | 1 | 8 | 10 | 10 | 10 |
| 69 | 43 | TENMOM | 10 | 6 | 1 | 1 | 8 | 10 | 10 | 10 |
| 70 | 43 | TENRQP | 10 | 6 | 1 | 1 | 8 | 10 | 10 | 10 |
| 71 | 43 | TENSRM | 10 | 6 | 1 | 1 | 8 | 10 | 10 | 10 |
| 72 | 2 | TESTCORD1 | 173 | 207 | 1 | 2 | 1017 | 173 | 11 | 176 |
| 73 | 2 | TESTCORD4 | 173 | 207 | 1 | 2 | 1027 | 173 | 11 | 176 |
| 74 | 45 | TESTNAS0 | 30 | 18 | 4 | 2 | 72 | 42 | 30 | 84 |
| 75 | 46 | TESTPARDYN | 9 | 11 | 1 | 0 | 54 | 9 | 9 | 91 |
| 76 | 47 | TR4X4 | 32 | 25 | 1 | 1 | 56 | 32 | 32 | 32 |
| 77 | 48 | TRIOM | 10 | 18 | 1 | 1 | 32 | 10 | 5 | 13 |
| 78 | 49 | TSHELL3 | 1 | 3 | 1 | 2 | 3 | 1 | 1 | 2 |
| 79 | 50 | TUBE | 60 | 32 | 4 | 1 | 72 | 132 | 8 | 98 |

NET: net size, i.e. number of finite elements

NOD: number of nodes

NLY: number of layers in case of composite elements (maximum value)

NLC: number of load cases

NDT: number of degrees of freedom

NSV: number of structural variables

NDV: number of design variables

NDG: total number of constraints (without bounds for variables)

Table 2 summarizes some information on the type of the constraints. The following data are listed:

NDG: total number of constraints

NGS: number of stress constraints

NGD: number of displacement constraints

NGB: number of buckling constraints

NGF: number of frequency constraints

NGV: number of eigenvector constraints

NGM: number of manufacturing constraints

NGR: number of frequency response constraints

NGT: number of time response constraints

FID: feasible initial design (0 - yes, 1 - no)

## 6 Numerical comparative results

In this section we describe the test procedure and summarize some of the numerical results achieved. All tests have been performed on a VAX 6000-510 running under VMS, at the Computing Centre of the University of Bayreuth. The numerical codes are implemented in double precision FORTRAN. Two additional optimization routines, SCP and MMA, are added to the FE-system MBB-LAGRANGE for the purposes of the comparative study.

The intention behind our tests is to apply all optimization routines to all test examples listed in the previous section. To evaluate the results achieved, we need some information about the optimal solution, since the difference from the minimal weight of a test structure and the corresponding constraint violation serves as a measure for the accuracy of an actual iterate.

Thus we have to compute an optimal solution for each test case as accurately as possible. The most reliable codes were executed with a very small termination tolerance and a large number of iterations, until we got a stable and reliable solution. Test examples that did not lead to a clear solution point, because of too many different local minimizers, have not been included in our set of test problems.

Having now an accepted reference value, it is possible to define whether an actual iterate $x$ is sufficiently close to the optimal solution $x^*$ subject to a given tolerance $\varepsilon > 0$ or not. For each function or gradient evaluation during a test run, we store the corresponding objective function value $f(x)$ and the maximum constraint violation

$$r(x) := \max\{|\min[0, g_j(x)]| : j = 1, \ldots, m\},$$

together with some further data for analysis number and calculation time.

Now we are able to evaluate the performance of an algorithm subject to a given accuracy level $\varepsilon$. We sum up the performance criterion, i.e. calculation time or number of function and gradient evaluations, until for the first time the conditions

$$f(x) \le f(x^*)(1+\varepsilon), \quad r(x) \le \varepsilon \tag{16}$$

are satisfied. We should note here that the constraint functions are scaled internally by the analysis procedure of MBB-LAGRANGE.

Moreover, there are some reasonable upper bounds for the number of iterations, and we must be aware of the fact that there are situations where a code is unable to find at least one solution in a test problem class within the given accuracy level and the maximum number of iterations.

When trying to evaluate the performance of an optimization algorithm, we are immediately faced with the following difficulties.

- The number of successful test runs is too small to prepare a statistical analysis particularly for small accuracy levels $\varepsilon$ and special subsets of test examples. Also there is no chance of finding the probability distribution of our performance criteria.
- When we evaluate only mean values over all test runs, we penalize the more reliable codes. Since the poor ones are often unable to find a solution of the more difficult examples, they avoid a large number of iterations needed to attain a solution in these cases.

- When we evaluate mean values only with respect to the test examples which could be solved successfully by all algorithms, we obtain a too small test set, containing moreover only the unimportant *toy* problems. Also it is possible that we obtain empty test sets.

Thus we need another approach to evaluate the performance of an algorithm in a suitable way, when their qualification varies drastically as in our case. One possible attempt is the priority theory of Saaty (1980), which was used by Schittkowski (1980) and Lootsma (1981). In these cases, it is tried to compare non-measurable quantities of optimization codes, e.g. ease of use. We apply now the same idea to measurable data such as calculation time and number of function and gradient evaluations.

For the purpose of our performance evaluation, we exploit Saaty's priority theory in its simplest form. Imagine that there are unknown weights or priorities $w_1, \ldots, w_n$, where $n$ is the number of optimization codes we want to investigate, which are positive and which satisfy

$$\sum_{i=1}^{n} w_i = 1 .$$

These weights are now to characterize one of our performance criteria, say calculation time. It is very easy to see that the matrix

$$\mathbf{A} := (w_i/w_j)_{i,j=1,n} ,$$

is of rank one and has only one eigenvalue $n$ with eigenvector

$$\mathbf{w} = (w_1, \ldots, w_n)^T ,$$

i.e.

$$\mathbf{A}\mathbf{w} = n\mathbf{w} .$$

Moreover the row sums of $\mathbf{A}$, i.e. $w_i \sum_{i=1}^{n} 1/w_j$ are multiples of the vector $\mathbf{w}$.

The idea motivating our approach is now that we are unable to obtain appropriate estimates for the desired priorities $w_i$ directly, but we can compute estimates for the relative performance weights $w_i/w_j$ in the following way. Let $p_i^k$ denote a performance index, e.g. number of function evaluations, for optimization algorithm $i$ and test example $k$, where $i = 1, \ldots, n$, $k = 1, \ldots, m$. Now $n$ denotes the number of algorithms that we want to compare, and $m$ the number of test examples in our test set under consideration. Moreover, we denote by $I_{ij}(\epsilon)$ the indices of all test problems, which are solved successfully by algorithms $i$ and $j$ subject to the error tolerance $\epsilon$ as defined by (16). Then

$$r_{ij} := \frac{\sum_{k \in I_{ij}} p_i^k}{\sum_{k \in I_{ij}} p_j^k} \tag{17}$$

is considered to be an estimate for $w_i/w_j$ and we use the normalized row sum of matrix $(r_{ij})$ as an estimate $\overline{w}_i$ for $w_i$, i.e.

$$\overline{w}_i := \frac{\sum_{j=1}^{n} r_{ij}}{\sum_{i,j=1}^{n} r_{ij}} . \tag{18}$$

Since the numerical figures for the performance criterion calculation time differ drastically, we use the geometric mean in this case to estimate the relative weights as above.

Another difficulty is that some nonlinear programming codes are only capable of solving certain subclasses of structural optimization problems, e.g. problems only with stress constraints (SRM) or only problems with a feasible initial design (IBF). Thus we consider the following subsets of test runs to evaluate the criteria as described:

| No. | Codes excluded | Test cases | Description |
|---|---|---|---|
| 1 | IBF, SRM | 79 | all problems |
| 2 | IBF | 44 | only stress constraints |
| 3 | IBF, SRM | 35 | only mixed constraints |
| 4 | SRM | 44 | feasible starting point |
| 5 | IBF, SRM | 35 | non-feasible starting point |

For the purpose of our comparative study, we evaluate the performance criteria

- calculation time in seconds;
- number of function evaluations, where an evaluation of objective and all constraints is counted as one function call; and
- number of gradient evaluations, i.e. evaluation of the gradient of objective function and of all active constraints, where active constraints are determined by the internal active set strategy of MBB-LAGRANGE.

These three criteria are computed by the modified priority theory as described. The resulting numerical figures cannot be interpreted as mean values for the performance item we

**Table 2.** Information on constraint types

| No. | Test example | NDG | NGS | NGD | NGB | NGF | NGV | NGM | NGR | NGT | FID |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | APLATE2 | 51 | 49 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | APLCON | 51 | 49 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | ARIANB | 1020 | 204 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | B4SIZE | 522 | 520 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | B4SIZELA | 522 | 520 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | BALKEN2 | 4 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 7 | BARBIG | 91 | 0 | 0 | 0 | 0 | 0 | 0 | 91 | 0 | 1 |
| 8 | BARDISP | 3 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | BARDYN2 | 4 | 3 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 10 | BARDYN5 | 4 | 3 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 11 | BARMASS | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 12 | BAROF9 | 7 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | BUST | 16 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | BUSTB | 16 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | BUSTM | 8 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16 | BUSTQ | 16 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17 | BUSTQM | 4 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | BUSTT | 20 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 19 | CANE | 41 | 40 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 20 | CELAS4 | 18 | 18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 21 | COMPKRAG | 44 | 16 | 0 | 0 | 0 | 0 | 12 | 0 | 0 | 1 |
| 22 | COOLPLATE | 178 | 177 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 23 | CORDS | 96 | 96 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 24 | DREI | 6 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 25 | DREIDISP | 6 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 26 | DREISLP | 6 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 27 | DYNPLT | 157 | 63 | 3 | 1 | 1 | 0 | 0 | 61 | 28 | 1 |
| 28 | EFA2CLAG | 153 | 108 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 29 | F1F3 | 67 | 56 | 0 | 11 | 0 | 0 | 0 | 0 | 0 | 1 |
| 30 | GARTEUR1 | 21 | 0 | 0 | 0 | 1 | 20 | 0 | 0 | 0 | 1 |
| 31 | GBOX | 98 | 97 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 32 | GENTEST | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 33 | GRADPLA | 10 | 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 34 | HEXA | 13 | 9 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 35 | JPATS16 | 416 | 416 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 36 | KALA3_1 | 40 | 36 | 3 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 37 | KRAG | 14 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 38 | KRAGBA | 26 | 12 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 39 | KRAGBAD | 25 | 12 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 40 | KRAGBAM | 20 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 2. Continued

| No. | Test example | NDG | NGS | NGD | NGB | NGF | NGV | NGM | NGR | NGT | FID |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 41 | KRAGDYN | 27 | 12 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 42 | KRAGIBF | 14 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 43 | KRAGMAN | 44 | 16 | 0 | 0 | 0 | 0 | 12 | 0 | 0 | 1 |
| 44 | KRAGMOM | 14 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 45 | KRAN0 | 54 | 54 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 46 | KRAN1 | 55 | 54 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 47 | LAGTEST | 40 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 48 | MCANE | 41 | 40 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 49 | MOMENT | 16 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 50 | MPC | 18 | 18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 51 | MPLATE | 12 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 52 | MPLATEH | 12 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 53 | OPM2 | 2 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |
| 54 | PLATTE | 41 | 40 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 55 | POINTMASS | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 56 | PUNCHTEST | 86 | 86 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 57 | QUARTOPLT | 20 | 4 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 58 | QUATRIA | 99 | 92 | 2 | 5 | 0 | 0 | 0 | 0 | 0 | 1 |
| 59 | RBAR | 25 | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 60 | SCHEIT210 | 32 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 61 | SPANT | 20 | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 62 | SPSLP | 414 | 414 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 63 | SPSLPB | 415 | 414 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 64 | T3D066 | 13 | 9 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 65 | TBDYN | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 66 | TEMPER2 | 12 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 67 | TENBAR | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 68 | TENCON | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 69 | TENMOM | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 70 | TENRQP | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 71 | TENSRM | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 72 | TESTCORD1 | 176 | 88 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 73 | TESTCORD4 | 176 | 88 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 74 | TESTNAS0 | 84 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 75 | TEST-PARDYN | 91 | 0 | 0 | 0 | 0 | 0 | 0 | 91 | 0 | 1 |
| 76 | TR4X4 | 32 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 77 | TRIOM | 13 | 10 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 1 |
| 78 | TSHELL3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 79 | TUBE | 98 | 96 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 1 |

are considering. They give an impression of the relative performance of an optimization code and must be interpreted in this way. Moreover, we want to evaluate some figures that measure the reliability of a code, i.e. any guess for the probability that an algorithm is capable to compute a solution subject to a given accuracy. Thus we display also the number of test problems that are not solved successfully by a code with respect to four different accuracy levels ranging from $\varepsilon = 0.01$ to $\varepsilon = 0.00001$.

The subsequent four figures display the results achieved in graphical form with respect to the set of all test runs. Figure 1 shows the numbers of unsolved problems solved with respect to four termination tolerances. The corresponding performance results calculation time, number of function and gradient evaluations are displayed in Figs. 2 to 4 for the final termination accuracy $\varepsilon = 0.01$.

For each of the five subsets of test runs defined above, two tables with information about the performance of the optimization algorithms are given. The first five tables show the robustness of the optimization codes, i.e. the number of problems that are not solved successfully subject to a given final accuracy. The subsequent five tables list the efficiency performance data calculation time and number of function and gradient evaluations, evaluated by the priority theory as outlined in the beginning. Again the priorities are computed for each of the five different test problem classes separately.

From the numerical results obtained, we can make the following observations:

(a) **Robustness**

(1) **All test problems** (Table 3)

The most robust implementation is the RQP1-code. For all tolerances it has the least number of unsolved problems. For $\varepsilon = 10^{-2}$ QPRLT is the second most reliable method, but with decreasing $\varepsilon$ it got almost the same number of failures as RQP2, SCP and MMA. For $\varepsilon = 10^{-2}$ the program SLP performs as reliably as SCP or MMA, but its relative robustness decreases for lower $\varepsilon$. The GRG-method performs a bit worse than QPRLT, SCP and MMA, especially for the largest $\varepsilon$. For lower termination tolerances, MOM is very unreliable and its usage cannot be recommended. Also CONLIN is quite unreliable and for lower $\varepsilon$ only MOM has more failures.

(2) **Stress/mixed constraints** (Tables 4/5)

When considering only problems with stress constraints, we observe that SLP, RQP2, SCP and MMA perform considerably better with respect to the percentage of unsolved problems than in test class no. 3, while GRG, QPRLT and RQP1, CONLIN do not seem to be sensitive to the type of constraints. On the other hand, this fact improves their relative performance with respect to test class no. 3. The SRM-method, which is only applicable in test problem class no. 2, is not very robust. Even for very low requirements on termination accuracy it solves less than one half of the test problems.

(3) **Feasible/infeasible initial designs** (Tables 6/7)

Apart from MOM, all methods are more robust when the initial design is feasible and considerably worse when it is infeasible. Their relative performance is not very different from that in the general case. The IBF-method, which is only considered in test class no. 4, is very unreliable and has the worst percentages of unsolved problems for all $\varepsilon$'s.

(b) **Efficiency**

(1) **Calculation time**

First of all, we have to mention, that there are no drastic differences between the computed weights for the four $\varepsilon$-values.

In test class no. 1, the most efficient method is CONLIN. It has the best weights followed by MMA and SLP, which do not differ significantly. The next group of algorithms with about the same efficiency scores consists of RQP2, QPRLT and SCP. GRG gave somewhat higher values. The RQP1-method requires a relatively large amount of calculation time, since the actual implementation writes and reads all intermediate analysis or optimization data, respectively, into temporary files to save memory.

The results in test class no. 3 are not very different from those in test class no. 1. When we consider the SRM-method in test class no. 2, we observe that SRM is the most efficient method concerning computation

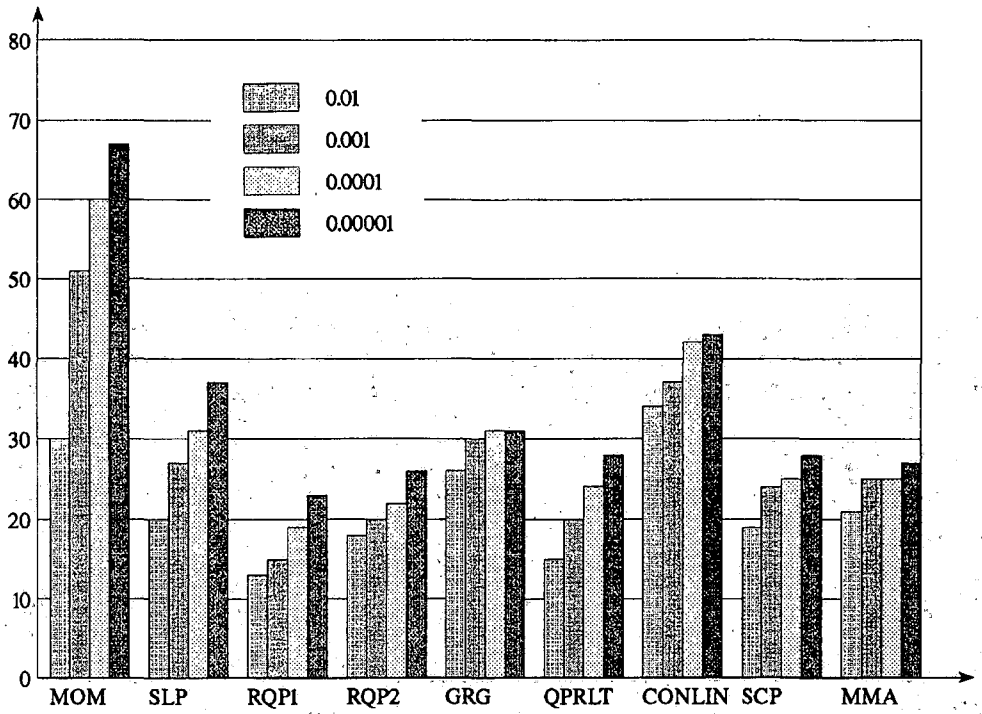


Fig. 1. Number of unsolved problems w.r.t. all test runs

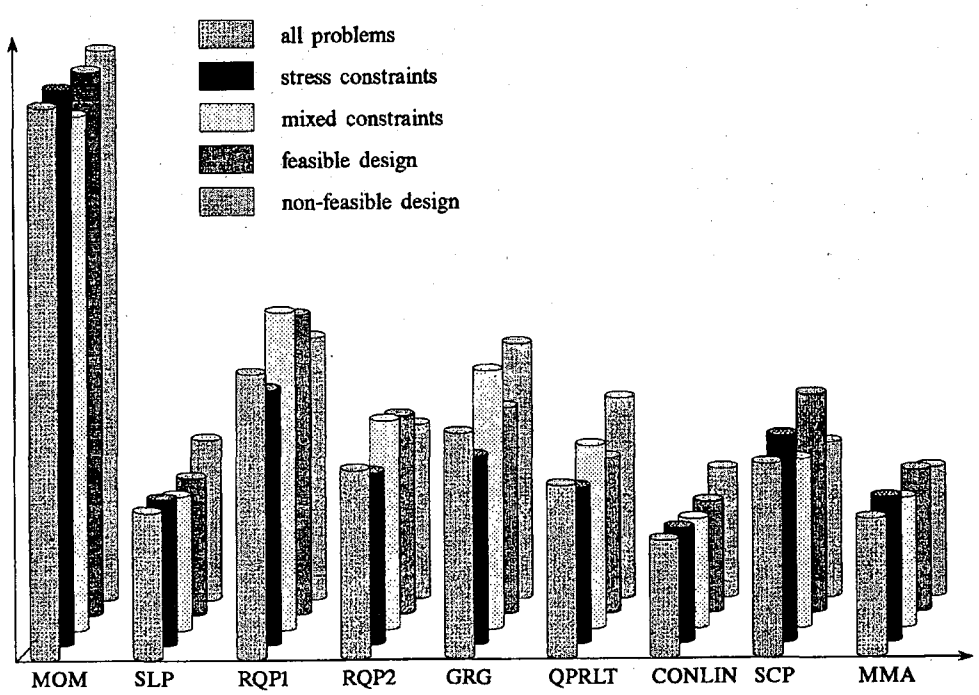


Fig. 2. Calculation time w.r.t. $\varepsilon = 0.01$

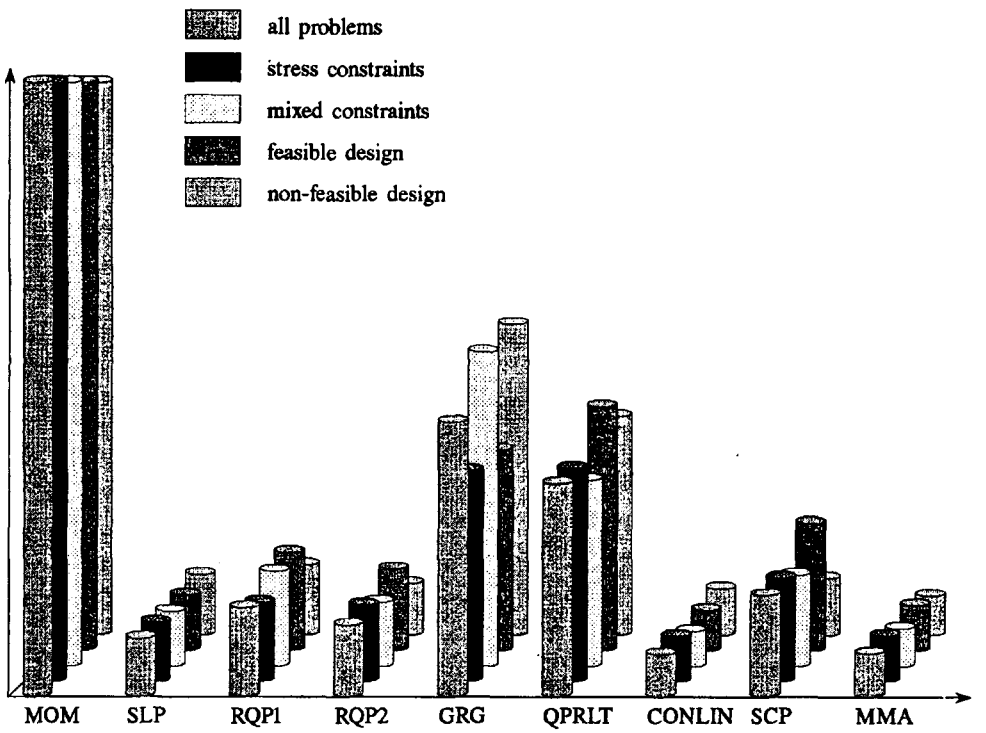


Fig. 3. Number of function evaluations w.r.t. $\epsilon = 0.01$

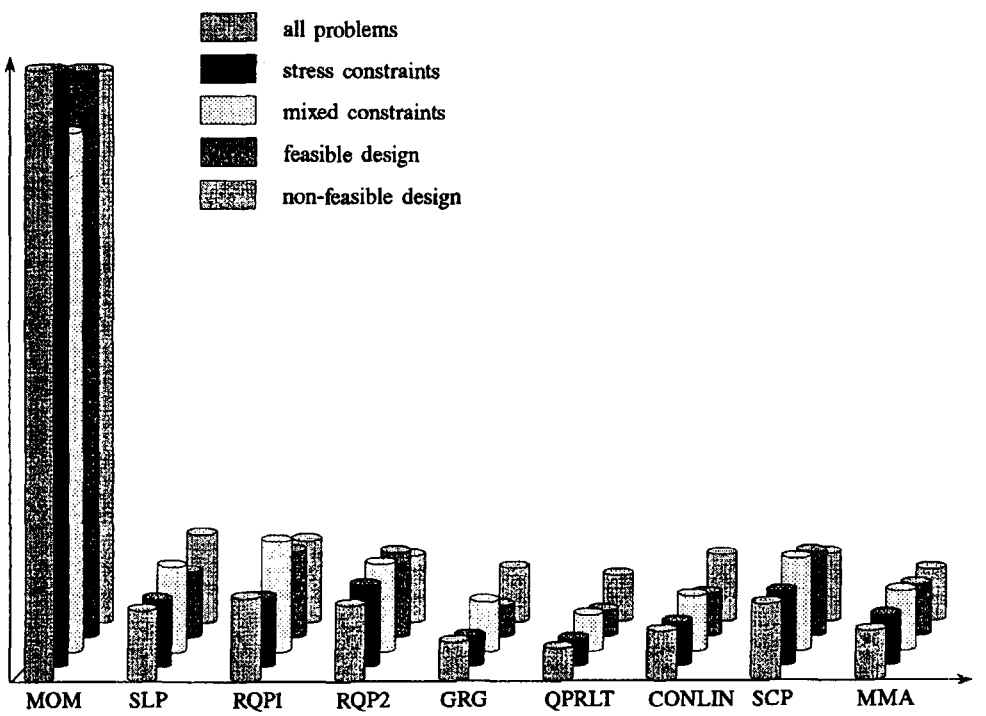


Fig. 4. Number of gradient evaluations w.r.t. $\epsilon = 0.01$

time and stress constraints. The relative performance of all other methods does not differ very much from that of test class no. 1. In test class no. 4 there are only slight differences in the relative values of the nine standard methods. The absolut values changed since the IBF–method is even worse than MOM, and the sum of the weights is normed to 100. For infeasible initial designs we see that QPRLT has higher values than in test class no. 1 and, on the other hand, the performance of SCP improves. The other methods have about the same scores in this test class.

**Table 3.** Number of unsolved problems for test class no. 1 (all 79 problems)

| Code | $\varepsilon = 0.01$ | | $\varepsilon = 0.001$ | | $\varepsilon = 0.0001$ | | $\varepsilon = 0.00001$ | |
|---|---|---|---|---|---|---|---|---|
| MOM | 30 | (38%) | 51 | (65%) | 60 | (76%) | 67 | (85%) |
| SLP | 20 | (25%) | 27 | (34%) | 31 | (39%) | 37 | (47%) |
| RQP1 | 13 | (16%) | 15 | (19%) | 19 | (24%) | 23 | (29%) |
| RQP2 | 18 | (23%) | 20 | (25%) | 22 | (28%) | 26 | (33%) |
| GRG | 26 | (33%) | 30 | (38%) | 31 | (39%) | 31 | (39%) |
| QPRLT | 15 | (19%) | 20 | (25%) | 24 | (30%) | 28 | (35%) |
| CONLIN | 34 | (43%) | 37 | (47%) | 42 | (53%) | 43 | (54%) |
| SCP | 19 | (24%) | 24 | (30%) | 25 | (32%) | 28 | (35%) |
| MMA | 21 | (27%) | 25 | (32%) | 25 | (32%) | 27 | (34%) |

**Table 4.** Number of unsolved problems for test class no. 2 (44 problems only with stress constraints)

| Code | $\varepsilon = 0.01$ | | $\varepsilon = 0.001$ | | $\varepsilon = 0.0001$ | | $\varepsilon = 0.00001$ | |
|---|---|---|---|---|---|---|---|---|
| SRM | 23 | (52%) | 24 | (55%) | 25 | (57%) | 26 | (59%) |
| MOM | 16 | (36%) | 29 | (66%) | 36 | (82%) | 42 | (95%) |
| SLP | 7 | (16%) | 10 | (23%) | 11 | (25%) | 13 | (30%) |
| RQP1 | 7 | (16%) | 7 | (16%) | 9 | (20%) | 9 | (20%) |
| RQP2 | 6 | (14%) | 7 | (16%) | 7 | (16%) | 10 | (23%) |
| GRG | 14 | (32%) | 14 | (32%) | 15 | (34%) | 15 | (34%) |
| QPRLT | 9 | (20%) | 12 | (27%) | 14 | (32%) | 16 | (36%) |
| CONLIN | 16 | (36%) | 17 | (39%) | 20 | (45%) | 20 | (45%) |
| SCP | 9 | (20%) | 10 | (23%) | 10 | (23%) | 11 | (25%) |
| MMA | 8 | (18%) | 9 | (20%) | 9 | (20%) | 9 | (20%) |

**Table 5.** Number of unsolved problems for test class no. 3 (35 problems only with mixed constraints)

| Code | $\varepsilon = 0.01$ | | $\varepsilon = 0.001$ | | $\varepsilon = 0.0001$ | | $\varepsilon = 0.00001$ | |
|---|---|---|---|---|---|---|---|---|
| MOM | 14 | (40%) | 22 | (63%) | 24 | (69%) | 25 | (71%) |
| SLP | 13 | (37%) | 17 | (49%) | 20 | (57%) | 24 | (69%) |
| RQP1 | 6 | (17%) | 8 | (23%) | 10 | (29%) | 14 | (40%) |
| RQP2 | 12 | (34%) | 13 | (37%) | 15 | (43%) | 16 | (46%) |
| GRG | 12 | (34%) | 16 | (46%) | 16 | (46%) | 16 | (46%) |
| QPRLT | 6 | (17%) | 8 | (23%) | 10 | (29%) | 12 | (34%) |
| CONLIN | 18 | (51%) | 20 | (57%) | 22 | (63%) | 23 | (66%) |
| SCP | 10 | (29%) | 14 | (40%) | 15 | (43%) | 17 | (49%) |
| MMA | 13 | (37%) | 16 | (46%) | 16 | (46%) | 18 | (51%) |

(2) **Number of function evaluations**

The computed weights seem to be more or less independent of $\varepsilon$ with one exception. The priority values for number of function evaluations in test class no. 1 differ strongly from those of the calculation time. SLP, RQP1, RQP2, CONLIN, SCP and MMA had a better score, where MOM, QPRLT and GRG had lower values. The relative ranking shows that CONLIN and MMA have the best weights. With some small differences in each case, SLP, RQP2, RQP1 and SCP follow. GRG and QPRLT have values much higher than those of the calculation time. The reason is that both methods need many function evaluations in their line–search to reach a feasible design in each iteration. The weights of MOM are again very bad.

In test classes no. 2 and 3 we do not observe significant differences to the results of test class no. 1. The performance of the SRM–method deteriorates with decreasing $\varepsilon$.

In test class no. 5 there is one remarkable difference. With decreasing $\varepsilon$ the performance indices of MOM improve more and more and finally are better than those of GRG and QPRLT. In test class no. 4 GRG increases with decreasing $\varepsilon$. The relative classification of the other methods is not very different from that in test class no. 1.

**Table 6.** Number of unsolved problems for test class no. 4 (44 problems only with feasible starting point)

| Code | $\varepsilon = 0.01$ | | $\varepsilon = 0.001$ | | $\varepsilon = 0.0001$ | | $\varepsilon = 0.00001$ | |
|---|---|---|---|---|---|---|---|---|
| IBF | 29 | (66%) | 35 | (80%) | 40 | (91%) | 40 | (91%) |
| MOM | 14 | (32%) | 33 | (66%) | 33 | (75%) | 38 | (86%) |
| SLP | 9 | (20%) | 13 | (30%) | 13 | (30%) | 16 | (36%) |
| RQP1 | 4 | (9%) | 6 | (14%) | 9 | (20%) | 10 | (23%) |
| RQP2 | 7 | (16%) | 9 | (20%) | 9 | (20%) | 12 | (27%) |
| GRG | 13 | (30%) | 13 | (30%) | 14 | (32%) | 14 | (32%) |
| QPRLT | 6 | (14%) | 10 | (23%) | 11 | (25%) | 13 | (30%) |
| CONLIN | 13 | (30%) | 15 | (34%) | 17 | (39%) | 18 | (41%) |
| SCP | 9 | (20%) | 12 | (27%) | 12 | (27%) | 14 | (32%) |
| MMA | 9 | (20%) | 12 | (27%) | 12 | (27%) | 13 | (30%) |

**Table 7.** Number of unsolved problems for test class no. 5 (35 problems only with non-feasible starting point)

| Code | $\varepsilon = 0.01$ | | $\varepsilon = 0.001$ | | $\varepsilon = 0.0001$ | | $\varepsilon = 0.00001$ | |
|---|---|---|---|---|---|---|---|---|
| MOM | 16 | (46%) | 22 | (63%) | 27 | (77%) | 29 | (83%) |
| SLP | 11 | (31%) | 14 | (40%) | 18 | (51%) | 21 | (60%) |
| RQP1 | 9 | (26%) | 9 | (26%) | 10 | (29%) | 13 | (37%) |
| RQP2 | 11 | (31%) | 11 | (31%) | 13 | (37%) | 14 | (40%) |
| GRG | 13 | (37%) | 17 | (49%) | 17 | (49%) | 17 | (49%) |
| QPRLT | 9 | (26%) | 10 | (29%) | 13 | (37%) | 15 | (43%) |
| CONLIN | 21 | (60%) | 22 | (63%) | 25 | (71%) | 25 | (71%) |
| SCP | 10 | (29%) | 12 | (34%) | 13 | (37%) | 14 | (40%) |
| MMA | 12 | (34%) | 13 | (37%) | 13 | (37%) | 14 | (40%) |

(3) **Number of gradient evaluations**

In this case, the weights do not depend heavily on $\varepsilon$. As GRG and QPRLT got relatively bad scores for the number of function evaluations, both methods improved their scores with respect to the number of gradient evaluations, since they do not use gradients for the restoration phase, i.e. the projection onto the feasible region. They are followed by CONLIN and MMA. SLP, RQP1, RQP2 and SCP have about the same weights and are a bit worse than the other four. As in the other cases, MOM is the most inefficient code.

In test problem classes no. 2 and 3 there are no significant differences to class no. 1 with respect to the

relative performance. The absolute values decrease in Table 9 only because the weights of MOM increase and the sum is normed. Only for SLP we observe a significant improvement for $\varepsilon = 10^{-4}$ in test class no. 3, probably generated by side effects of other codes with lower reliability and efficiency in this case. Since SRM does not use any gradient information at all, it got the best scores.

For the classes of feasible/infeasible initial design we cannot observe significant differences in the relative performance. For test class no. 4 IBF has the worst values for higher $\varepsilon$, but its weights decrease for lower $\varepsilon$. Vice versa, the same is valid for MOM. In test class no. 5, there is a decrease in the weights of MOM similar to the number of function evaluations, but its final weight is still very bad.

**Table 8.** Performance indices calculation time, number of function and gradient evaluations for test problem class no. 1 (all problems)

| Code | $\varepsilon = 0.01$ | $\varepsilon = 0.001$ | $\varepsilon = 0.0001$ | $\varepsilon = 0.00001$ |
|---|---|---|---|---|
| | 27.16 | 31.70 | 31.58 | 28.79 |
| MOM | 43.72 | 48.88 | 47.76 | 49.64 |
| | 55.67 | 61.55 | 63.20 | 54.68 |
| | 7.28 | 6.82 | 7.30 | 6.91 |
| SLP | 3.66 | 3.29 | 4.22 | 3.21 |
| | 6.62 | 5.84 | 7.25 | 6.85 |
| | 14.15 | 12.97 | 13.07 | 13.63 |
| RQP1 | 5.59 | 4.47 | 4.59 | 4.51 |
| | 7.69 | 6.15 | 5.75 | 7.43 |
| | 9.39 | 8.25 | 8.31 | 8.97 |
| RQP2 | 4.49 | 3.64 | 3.88 | 3.93 |
| | 7.05 | 5.38 | 5.23 | 6.89 |
| | 11.20 | 10.80 | 10.68 | 11.06 |
| GRG | 17.34 | 15.87 | 15.71 | 15.56 |
| | 3.62 | 3.03 | 2.71 | 3.41 |
| | 8.61 | 8.13 | 8.17 | 8.36 |
| QPRLT | 13.45 | 12.56 | 12.72 | 11.77 |
| | 3.02 | 2.66 | 2.53 | 2.96 |
| | 5.82 | 5.68 | 5.33 | 5.66 |
| CONLIN | 2.63 | 3.12 | 2.35 | 2.45 |
| | 4.59 | 5.18 | 3.49 | 4.60 |
| | 9.54 | 9.06 | 9.03 | 9.59 |
| SCP | 6.39 | 5.59 | 6.01 | 6.07 |
| | 7.11 | 5.94 | 5.65 | 7.43 |
| | 6.86 | 6.59 | 6.54 | 7.04 |
| MMA | 2.73 | 2.59 | 2.75 | 2.86 |
| | 4.64 | 4.26 | 4.19 | 5.75 |

## 7 Conclusions

Before drawing any conclusions, we have to note two important observations.

- All optimization codes are executed with one constant set of parameters and control options. An experienced user trying to approach a solution step by step, will certainly be able to tune the parameters depending on the model and data, and could, therefore, achieve much better results in practice. We do not investigate the question, whether one algorithm is more robust than another with respect to input tolerances.

**Table 9.** Performance indices calculation time, number of function and gradient evaluations for test problem class no. 2 (stress constraints)

| Code | $\varepsilon = 0.01$ | $\varepsilon = 0.001$ | $\varepsilon = 0.0001$ | $\varepsilon = 0.00001$ |
|---|---|---|---|---|
| | 3.96 | 4.04 | 4.20 | 4.62 |
| SRM | 3.20 | 3.94 | 4.45 | 5.19 |
| | 0.0 | 0.0 | 0.0 | 0.0 |
| | 27.31 | 34.40 | 37.72 | 36.91 |
| MOM | 44.33 | 53.62 | 53.73 | 54.03 |
| | 58.64 | 69.06 | 78.97 | 77.51 |
| | 7.24 | 6.16 | 6.08 | 6.00 |
| SLP | 3.69 | 2.63 | 3.00 | 2.72 |
| | 6.19 | 3.98 | 2.86 | 2.90 |
| | 12.64 | 10.89 | 10.67 | 10.72 |
| RQP1 | 4.96 | 3.35 | 3.53 | 3.49 |
| | 6.38 | 4.10 | 2.89 | 3.13 |
| | 8.56 | 7.13 | 6.94 | 7.06 |
| RQP2 | 4.80 | 3.19 | 3.42 | 3.32 |
| | 7.45 | 4.70 | 3.30 | 3.55 |
| | 5.68 | 5.49 | 4.95 | 5.01 |
| GRG | 2.83 | 3.02 | 2.48 | 2.42 |
| | 4.11 | 4.34 | 2.76 | 3.01 |
| | 9.35 | 9.06 | 8.57 | 8.63 |
| QPRLT | 13.44 | 12.56 | 13.09 | 13.23 |
| | 2.81 | 2.26 | 1.43 | 1.50 |
| | 7.78 | 6.83 | 6.35 | 6.39 |
| CONLIN | 13.31 | 10.36 | 8.66 | 8.19 |
| | 2.54 | 1.86 | 1.20 | 1.24 |
| | 10.28 | 9.31 | 8.50 | 8.50 |
| SCP | 6.55 | 4.95 | 5.21 | 4.94 |
| | 6.79 | 5.47 | 3.74 | 3.99 |
| | 7.21 | 6.68 | 6.01 | 6.16 |
| MMA | 2.88 | 2.36 | 2.42 | 2.46 |
| | 4.70 | 3.95 | 2.66 | 2.98 |

- We cannot exclude that some test examples possess other local solutions with larger objective function values. Whenever an algorithm approximates a local solution different from the reference design, an error will be reported. There is no attempt to identify this situation.

Both remarks explain, at least partially, why the total number of unsolved problems is relatively large for all nonlinear programming codes under investigation. Also the set of test problems does not consist only of *trivial* problems, although the problem sizes are relatively small. Thus it must be recommended to include as many different optimization algorithms in a structural design system as possible.

To summarize the most important conclusions, we distinguish between the seven different optimization strategies introduced in Section 3. However the conclusions are drawn from the numerical results obtained by the special implementations under consideration. Since for some of the optimization strategies considered, only one code was implemented, we must be careful when applying them to other situations.

*1. Optimality criteria methods*

The only algorithm of this class in the MBB-LAGRANGE system is the stress ratio method SRM. Even if we apply the code only to problems with stress constraints, more than 50% of all problems cannot be solved at all because of the heuristic iteration method without any additional safeguards guaranteeing convergence. On the other hand, SRM is the

**Table 10.** Performance indices calculation time, number of function and gradient evaluations for test problem no. 3 (mixed constraints)

| Code | $\varepsilon = 0.01$ | $\varepsilon = 0.001$ | $\varepsilon = 0.0001$ | $\varepsilon = 0.00001$ |
|---|---|---|---|---|
| | 25.29 | 25.56 | 24.96 | 26.05 |
| MOM | 44.67 | 44.16 | 42.83 | 47.30 |
| | 47.08 | 48.36 | 40.27 | 45.31 |
| | 6.66 | 7.26 | 8.30 | 6.20 |
| SLP | 3.42 | 3.79 | 6.06 | 3.30 |
| | 7.84 | 8.84 | 16.06 | 9.70 |
| | 15.67 | 15.45 | 15.24 | 15.13 |
| RQP1 | 5.97 | 5.26 | 5.32 | 5.14 |
| | 10.04 | 9.04 | 9.32 | 9.14 |
| | 10.34 | 9.72 | 9.54 | 10.03 |
| RQP2 | 4.02 | 3.71 | 3.90 | 4.07 |
| | 8.00 | 7.31 | 7.89 | 8.25 |
| | 12.79 | 12.38 | 11.96 | 11.71 |
| GRG | 19.87 | 19.38 | 16.35 | 15.17 |
| | 4.61 | 4.24 | 4.02 | 3.84 |
| | 9.08 | 9.50 | 9.94 | 9.59 |
| QPRLT | 11.73 | 12.90 | 15.08 | 13.41 |
| | 3.37 | 3.43 | 3.89 | 3.41 |
| | 5.40 | 5.27 | 4.96 | 5.39 |
| CONLIN | 2.20 | 2.67 | 1.71 | 2.04 |
| | 5.09 | 5.62 | 3.87 | 4.28 |
| | 8.37 | 8.54 | 8.65 | 9.13 |
| SCP | 5.74 | 5.67 | 6.11 | 6.76 |
| | 8.46 | 7.56 | 8.13 | 8.71 |
| | 6.40 | 6.32 | 6.46 | 6.76 |
| MMA | 2.37 | 2.44 | 2.66 | 2.81 |
| | 5.50 | 5.61 | 6.55 | 7.36 |

most efficient method in case of convergence particularly with respect to calculation time. We have to note here that the algorithm does not require any sensitivity analysis.

*2. Penalty methods*

The inverse barrier method IBF implemented in MBB-LAGRANGE, is unreliable and inefficient. Moreover, this version requires a feasible initial design thus restricting the domain of application. The usage of an inverse barrier method similar to the code IBF within the framework of a mechanical structural optimization system cannot be recommended.

*3. Multiplier methods*

Also in this case, the successive unconstrained optimization cycles require a large number of function and gradient evaluations. Only the reliablity is acceptable for very large termination tolerances. Again the conclusions about the method are vague since the results are obtained by only one implementation of a multiplier method.

*4. Sequential linear programming methods*

Although only one realization of a sequential linear programming method was tested, the results are very promising. Since there is no inheritance of round-off errors as for the more advanced numerical algorithms, the code is quite robust and efficient. Moreover, the implementation is simple provided that a *black box* solver for the linear programming subproblem is available. Also the method can be applied to solve large real life design problems successfully.

**Table 11.** Performance indices calculation time, number of function and gradient evaluations for test problem class no. 4 (feasible starting point)

| Code | $\varepsilon = 0.01$ | $\varepsilon = 0.001$ | $\varepsilon = 0.0001$ | $\varepsilon = 0.00001$ |
|---|---|---|---|---|
| | 32.57 | 29.29 | 24.25 | 24.87 |
| IBF | 56.17 | 55.89 | 60.48 | 56.49 |
| | 44.89 | 48.25 | 24.79 | 27.91 |
| | 18.24 | 21.69 | 27.41 | 26.61 |
| MOM | 19.03 | 22.99 | 22.98 | 27.45 |
| | 31.40 | 34.91 | 53.43 | 48.51 |
| | 4.61 | 4.45 | 4.72 | 4.62 |
| SLP | 1.54 | 1.17 | 1.07 | 0.93 |
| | 3.31 | 2.05 | 3.23 | 3.11 |
| | 9.99 | 9.48 | 9.25 | 9.18 |
| RQP1 | 2.72 | 1.98 | 1.53 | 1.41 |
| | 4.30 | 2.85 | 3.70 | 3.93 |
| | 6.64 | 6.07 | 6.00 | 6.18 |
| RQP2 | 2.28 | 1.73 | 1.37 | 1.30 |
| | 4.24 | 2.68 | 3.53 | 3.90 |
| | 6.92 | 3.98 | 3.70 | 3.72 |
| GRG | 6.78 | 1.33 | 0.78 | 0.74 |
| | 1.60 | 2.13 | 2.08 | 2.28 |
| | 5.22 | 7.58 | 7.49 | 7.53 |
| QPRLT | 5.58 | 6.46 | 5.06 | 5.09 |
| | 1.33 | 1.17 | 1.50 | 1.70 |
| | 3.72 | 5.00 | 4.98 | 4.85 |
| CONLIN | 1.12 | 4.31 | 3.54 | 3.45 |
| | 2.15 | 0.89 | 1.24 | 1.35 |
| | 7.31 | 7.46 | 7.35 | 7.47 |
| SCP | 3.53 | 2.98 | 2.32 | 2.29 |
| | 4.22 | 3.01 | 3.86 | 4.34 |
| | 4.78 | 5.01 | 4.85 | 4.96 |
| MMA | 1.27 | 1.14 | 0.88 | 0.86 |
| | 2.57 | 2.06 | 2.63 | 2.97 |

*5. Sequential quadratic programming methods*

As is also known from other comparative studies, sequential quadratic programming methods can be implemented in a very reliable and robust way. There are two drawbacks when applying them to solve structural optimization problems. First round-off errors may have a significant impact on the numerical performance, in particular when introduced by inexact numerical derivatives. Especially they prevent the superlinear local convergence speed. Moreover, they require some memory space for an approximation of the Hessian matrix and for the linearized constraints which is often not available for large real life structures when implemented in the standard way. However, for small or medium size problems as tested in the frame of the comparative study, the sequential quadratic programming algorithms are very efficient with respect to the number of function and gradient evaluations, and are only inferior to special purpose implementations exploiting model dependent features.

*6. Generalized reduced gradient methods*

The necessity to project a new iterate back to the feasible region as soon as a constraint is violated, requires additional function evaluations, since this procedure does not depend on gradients within the implementation of the codes tested. Thus the performance scores with respect to number of gradient calls are much better than the corresponding scores for function evaluations. On the other hand the generalized re-

**Table 12.** Performance indices calculation time, number of function and gradient evaluations for test problem no. 5 (non-feasible starting point)

| Code | $\epsilon = 0.01$ | $\epsilon = 0.001$ | $\epsilon = 0.0001$ | $\epsilon = 0.00001$ |
|---|---|---|---|---|
| | 27.19 | 33.54 | 23.73 | 19.72 |
| MOM | 45.59 | 51.09 | 24.94 | 20.59 |
| | 52.41 | 54.68 | 42.78 | 26.13 |
| | 7.95 | 7.73 | 9.73 | 7.83 |
| SLP | 3.86 | 3.78 | 9.12 | 6.35 |
| | 7.98 | 8.73 | 16.55 | 13.52 |
| | 12.97 | 12.10 | 14.04 | 15.51 |
| RQP1 | 4.45 | 3.79 | 5.93 | 7.55 |
| | 7.42 | 6.56 | 7.78 | 12.18 |
| | 8.67 | 7.73 | 8.91 | 10.22 |
| RQP2 | 3.30 | 2.76 | 4.48 | 5.83 |
| | 6.05 | 5.23 | 6.50 | 10.16 |
| | 12.64 | 10.73 | 11.87 | 12.44 |
| GRG | 19.75 | 15.58 | 21.54 | 22.99 |
| | 4.94 | 4.22 | 4.42 | 5.84 |
| | 9.95 | 9.66 | 11.41 | 11.90 |
| QPRLT | 13.90 | 14.35 | 21.58 | 20.84 |
| | 4.22 | 4.06 | 4.64 | 5.51 |
| | 6.46 | 5.68 | 6.14 | 7.06 |
| CONLIN | 2.99 | 3.04 | 3.24 | 4.71 |
| | 6.09 | 6.42 | 5.05 | 8.17 |
| | 7.72 | 7.01 | 7.63 | 8.12 |
| SCP | 3.65 | 3.28 | 5.37 | 6.22 |
| | 6.17 | 5.50 | 6.41 | 9.56 |
| | 6.44 | 5.82 | 6.52 | 7.21 |
| MMA | 2.51 | 2.33 | 3.79 | 4.92 |
| | 4.71 | 4.58 | 5.87 | 8.95 |

duced gradient method is very reliable in particular when some features similar to those of sequential quadratic programming algorithms are implemented. An additional advantage of these methods is that whenever an iteration is stopped, the final design is feasible.

*6. Sequential convex programming methods*

Since convex programming methods exploit special features of the underlying design model by inverting some variables, it can be expected that the resulting codes are more efficient than the general purpose optimizers discussed above. However, we obtain a reasonable reliablity only when combining the convex approximation with additional safeguards, i.e. moving asymptotes or line search. It is surprising that the method is also very efficient when applied to mixed constraints where the motivation cannot be justified as easily as only for stress constraints. Probably, in most of these cases the stress constraints also dominate. The code MMA is only slightly less reliable than SCP, although no line search is performed. Obviously we have a test problem set with initial designs which are relatively close to the optimal solution.

## References


Abadie, J. 1978: The GRG method for nonlinear programming. In: Greenberg, H. (ed.) *Design and implementation of optimization software.* Alphen an den Rijn: Sijthoff and Noordhoff

Arora, J.S.; Belegundu, A.D. 1985: A study of mathematical programming methods for structural optimization. Part II: Numerical results. *Int. J. Num. Meth. Engng.* **21**, 1601–1623

Asaadi, J. 1973: A computational comparison of some nonlinear programs. *Nonlinear Programming* **4**, 144–154

Berke, L.; Khot, N.S. 1974: Use of optimality criteria methods for large scale systems. *AGARD Lecture Series No. 70 on Structural Optimization*

Bertsekas, D.P. 1976: Multiplier methods: a survey. *Automatica* **12**, 133–145

Bremicker, M. 1986: Entwicklung eines Optimierungsalgorithmus der generalisierten reduzierten Gradienten. *Report, Forschungslaboratorium für angewandte Strukturoptimierung, Universität-GH Siegen*

Broyden, C.G.; Attia, N.F. 1988: Penalty functions, Newton's method, and quadratic programming. *J. Optimiz. Theory Appl.* **58**, 377–385

Colville, A.R. 1968: A comparative study on nonlinear programming codes. *IBM N.Y. Scientific Center Report 320–2949*

Eason, E.D.; Fenton, R.G. 1972: Testing and evaluation of numerical methods for design optimization. *Technical Publication Series UTME-TP 7204*, Department of Mechanical Engineering, University of Toronto

Eason, E.D.; Fenton, R.G. 1974: A comparison of numerical optimization methods for engineering design. *Trans. ASME* **96**, Series B, 196–200

Fiacco, A.V.; McCormick, G.P. 1968: *Nonlinear sequential unconstrained minimization techniques.* New York: John Wiley & Sons

Fletcher, R. 1975: An ideal penalty function for constrained optimization. In: Mangasarian, O.L.; Meyer, R.R.; Robinson, S.M. (eds.) *Nonlinear programming 2.* New York: Academic Press

Fletcher, R.; De La Maza, E.S. 1987: Nonlinear programming and nonsmooth optimization by successive linear programming. *Report No. Na/100*, Department of Mathematical Sciences, University of Dundee

Fleury, C. 1979: Structural weight optimization by dual methods of convex programming. *Int. J. Num. Meth. Engng.* **14**, 1761–1783

Fleury, C. 1986: Shape optimal design by the convex linearization method. In: Bennett, J.; Botkin, M. (eds.) *The optimum shape: automated structural design*, pp. 297–326. New York: Plenum Press

Fleury, C. 1989: CONLIN: an efficient dual optimizer based on convex approximation concepts. *Struct. Optim.* **1**, 81–89

Fleury, C.; Ramanathan, R.K.; Salema, M.; Schmitt, L.A. 1984: Access computer program for the synthesis of large structural systems. *NASA Langley Research Center Optimization Conf.*

Gill, P.E.; Murray, W.; Wright, M.H. 1981: *Practical optimization.* New York: Academic Press

Han, S.-P. 1976: Superlinearly convergent variable metric algorithms for general nonlinear programming problems. *Mathematical Programming* **11**, 263–282

Han, S.-P. 1977: A globally convergent method for nonlinear programming. *J. Optimiz. Theory Appl.* **22**, 297–309

Hestenes, M.R. 1969: Multiplier and gradient methods. *J. Optimiz. Theory Appl.* **4**, 303–320

Hock, W.; Schittkowski, K. 1981: Test examples for nonlinear programming codes. *Lecture Notes in Economics and Mathematical Systems* **187**. Berlin, Heidelberg, New York: Springer

Hörnlein, H.R.E.M. 1986: Sensitivitätsanalyse für verformungsabhängige Nebenbedingungen bei der Dimensionierung von FE-Strukturen. *Technical Note TN/S-0129-86*, MBB, Ottobrunn

Hörnlein, H.; Schittkowski, K. (eds.) 1993: *Software systems for structural optimization*. Basel: Birkhäuser

Kneppe, G.; Krammer, H.; Winkler, F. 1987: Structural optimization of large scale problems using MBB-LAGRANGE. *Proc. 5th World Cong. and Exhibition on FEM* (held in Salzburg, Austria)

Kneppe, G. 1985: Direkte Lösungsstrategien zur Gestaltsoptimierung von Flächentragwerken. *Reihe 1, No. 135*, Düsseldorf: VDI-Verlag

Lasdon, L.S.; Kim, N.-H.; Zhang, J. 1983: An improved successive linear programming algorithm. *Working Paper 8384-3-1*, Department of General Business, The University of Texas at Austin

Lasdon, L.S.; Ratner, M.W.; Waren, A.D. 1978: GRG2 user's guide. *Report, School of Business Administration, The University of Texas at Austin*

Lasdon, L.S.; Waren, A.D. 1978: Generalized reduced gradient software for linearly and nonlinearly constrained problems. In: Greenberg, H. (ed.) *Design and implementation of optimization software*. Alphen an den Rijn: Sijthoff and Noordhoff

Lootsma, F.A. 1971: A survey of methods for solving constrained minimization problems via unconstrained minimization. In: Lootsma, F.A. (ed.) *Numerical methods for nonlinear optimization*. New York: Academic Press

Lootsma, F.A. 1981: Fuzzy performance evaluation of nonlinear optimization methods, with sensitivity analysis and final scores. *J. Information and Optimization Sciences* **10**, 15–44

MacNeal-Schwendler Co. 1993: MSC-NASTRAN Handbook for Linear Analysis.

Murray, W. 1967: Ill-conditioning in barrier and penalty functions arising in constrained nonlinear programming. *Proc. Sixth Int. Symp. on Mathematical Programming* (held at Princeton University, N.J.)

Parkinson, A.; Wilson, M. 1986: Development of a hybrid SQP-GRG-algorithm for constrained nonlinear programming. *Proc. Design Engineering Technical Conf.* (held in Ohio)

Papalambros, P.Y.; Wilde, D.J. 1988: *Principles of optimal design*. Cambridge: Cambridge University Press

Pierre, D.A.; Lowe, M.J. 1975: *Mathematical programming via augmented Lagrangian. An introduction with computer programs*. Addison-Wesley

Powell, M.J.D. 1969: A method for nonlinear constraints in minimization problems. In: Fletcher, R. (ed.) *Optimization*. New York: Academic Press

Powell, M.J.D. 1978a: A fast algorithm for nonlinearly constrained optimization calculations. In: Watson, G.A. (ed.) *Numerical analysis*. Berlin, Heidelberg, New York: Springer

Powell, M.J.D. 1978b: The convergence of variable metric methods for nonlinearly constrained optimization calculations. In: Mangasarian, O.L.; Meyer, R.R.; Robinson, S.M. (eds.) *Nonlinear programming*. New York: Academic Press

Powell, M.J.D. 1983: On the quadratic programming algorithm of Goldfarb and Idnani. *Report DAMTP* **19**, University of Cambridge

Powell, M.J.D. 1992: Log barrier methods for semi-infinite programming calculations. *Report DAMTP* **11**, University of Cambridge

Rockafellar, R.T. 1974: Augmented Lagrange multiplier functions and duality in non-convex programming. *SIAM J. Control* **12**, 268–285

Saaty, T.L. 1980: *The analytic hierarchy process, planning, priority setting, resource allocation*. New York: McGraw-Hill

Sandgren, E. 1977: *The utility of nonlinear programming algorithms*. Ph.D. Thesis, Purdue University, West Lafayette, Indiana

Sandgren, E.; Ragsdell, K.M. 1982: On some experiments which delimit the utility of nonlinear programming methods for engineering design. *Mathematical Programming Study* **16**, 118–136

Schittkowski, K. 1980: Nonlinear programming codes. *Lecture Notes in Economics and Mathematical Systems* **183**. Berlin, Heidelberg, New York: Springer

Schittkowski, K. 1983: On the convergence of a sequential quadratic programming method with an augmented Lagrangian line search function. *Optimization, Mathematische Operationsforschung und Statistik* **14**, 197–216

Schittkowski, K. 1985a: On the global convergence of nonlinear programming algorithms. *J. Mech. Trans. Auto. Des.* **107**, 454–458

Schittkowski, K. 1985b: A unified outline of nonlinear programming algorithms. *J. Mech. Trans. Auto. Des.* **107**, 449–453

Schittkowski, K. 1985/86: NLPQL: a FORTRAN subroutine solving constrained nonlinear programming problems. *Annals of Operations Research* **5**, 485–500

Schittkowski, K. 1986: On the convergence of a generalized reduced gradient algorithm for nonlinear programming. *Optimization* **17**, 731–755

Schittkowski, K. 1987: More test examples for nonlinear programming codes. *Lecture Notes in Economics and Mathematical Systems* **282**. Berlin, Heidelberg, New York: Springer

Schittkowski, K. 1992: Solving nonlinear programming problems with very many constraints. *Optimization* **25**, 179–196

Schuldt, S.B. 1975: A method of multipliers for mathematical programming problems with equality and inequality constraints. *J. Optimiz. Theory Appl.* **17**

Schwarz, H.R. 1981: *FORTRAN-Programme zur Methode der Finiten Elemente*. Stuttgart: Teubner

Sömer, M. 1987: Aufstellen und Testen eines hybriden SQP-GRG Optimierungsalgorithmus. Studienarbeit, Institut für Mechanik und Regelungstechnik, Universität-GH-Siegen

Spellucci, P. 1993: *Numerische Verfahren der nichtlinearen Optimierung*. Basel: ISNM Birkhäuser

Stoer, J. 1985: Principles of sequential quadratic programming methods for solving nonlinear programs. In: Schittkowski, K. (ed.) *Computational mathematical programming*, pp. 165–208

Svanberg, K. 1987: Method of moving asymptotes — a new method for structural optimization. *Int. J. Num. Meth. Engng.* **24**, 359–373

Wolfe, P. 1967: Methods for linear constraints. In: Abadie, J. (ed.) *Nonlinear programming*. Amsterdam: North-Holland

Zillober, C. 1993a: A globally convergent version of the method of moving asymptotes. *Struct. Optim.* 6, 166–174

Zillober, C. 1993b: SCP — an implementation of two algorithms for solving nonlinear programming problems. *Report of the DFG* (German Science Foundation), Universität Bayreuth

Zotemantel, R. 1993: MBB-LAGRANGE: a computer aided structural design system. In: Hörnlein, H.; Schittkowski, K. (eds.) *Software systems for structural optimization.* Basel: Birkhäuser

*Received Aug. 9, 1993*

## Errata

Rozvany, G.I.N.; Sigmund, O.; Lewiński, T.; Gerdes, D.; Birker, T. 1993: Exact optimal structural layouts for non-self-adjoint problems.

*Struct. Optim.* 5, 204-206
in (4) $\cos(2\alpha_1)$ should read $-\cos(2\alpha_1)$
in (11) the last "-" sign should read "+".