Object oriented implementation of a second-order optimization method

L. F. D. Bras¹ & A. F. M. Azevedo¹
¹Civil Engineering Department, Faculty of Engineering, University of Porto, Portugal.

Abstract

A structural optimization problem may be formulated as a single nonlinear program (NLP) and solved with a second-order method. This class of methods requires first and second derivatives of the objective function and of the inequality and equality constraints. When the size of the problem is large, derivative calculation may become tedious and error prone. Automatic differentiation (AD) techniques provide exact values for the derivatives without user intervention. In this paper the object oriented implementation of a Lagrange-Newton optimization algorithm is presented. An object oriented parser is used to interpret all the functions that describe the NLP. First and second derivatives are calculated with AD techniques using Rall numbers and their associated overloaded operators. A shape optimization example is presented to illustrate the proposed techniques.

1 Introduction

Object oriented programming techniques provide higher-level tools to manipulate information. Before coding a numerical algorithm classes must be developed in order to implement the basic operations that will be required at a later stage. These classes are composed of sets of variables that store the information related to the class, and operations that safely manipulate that data. Vectors, sparse matrices, functions and Rall numbers [1] are examples of classes and addition, multiplication, evaluation, entry removal and partial derivation are examples of operations. The computer code that is described later required the development of a large number of basic classes. Object oriented techniques, such as inheritance, polymorphism and templates [2], played an important role in the
organization of the relationship between classes and provided significant economies in terms of development time and code extension. The higher-level algorithm utilizes the basic classes and implements new ones in order to perform its task.

2 Nonlinear programming

In the present work, structural optimization problems are formulated as a single nonlinear program (NLP), whose general form is shown in eqn (1).

\[
\begin{align*}
\text{Min.} & \quad f(x_1,\ldots,x_n) \\
\text{subject to} & \quad g_j(x_1,\ldots,x_n) \leq 0 \rightarrow g_j(x_1,\ldots,x_n) + s_j^2 = 0 \quad (j=1,\ldots,m) \\
& \quad h_k(x_1,\ldots,x_n) = 0 \quad (k=1,\ldots,p)
\end{align*}
\]

Squared slack variables are used to allow for the replacement of all the inequality constraints by equality constraints. The NLP (1) is solved by the Lagrange-Newton method using the necessary condition \( \nabla L = 0 \), where \( L \) is the Lagrangian. The resulting system of \( n + 2 m + p \) nonlinear equations (2) is solved by the Newton method [3].

\[
\begin{align*}
\frac{\partial f}{\partial x_i} + \sum_{j=1}^{m} \lambda_j^g \frac{\partial g_j}{\partial x_i} + \sum_{k=1}^{p} \lambda_k^h \frac{\partial h_k}{\partial x_i} &= 0 \quad (i=1,\ldots,n) \\
2 \lambda_j^g s_j &= 0 \quad (j=1,\ldots,m) \\
g_j + s_j^2 &= 0 \quad (j=1,\ldots,m) \\
h_k &= 0 \quad (k=1,\ldots,p)
\end{align*}
\]

In each Newton iteration \( q \) the following system of linear equations has to be solved

\[
H \{X^{q-1}\} \Delta X^q + \nabla L \{X^{q-1}\} = 0
\]

In eqn (3) \( H \) is the Hessian matrix and \( X \) is a vector containing all the variables \((x_i, s_j, \lambda_j^g, \lambda_k^h)\). The current solution is updated with

\[
X^q = X^{q-1} + \alpha^q \Delta X^q
\]

where \( \alpha^q \) is the line search parameter. Scaling techniques are used to improve the robustness and efficiency of the iterative process (see Section 6). The sparsity pattern of the Hessian matrix \( H \) is exploited in order to save storage and
unnecessary operations. With this approach, optimization problems with thousands of independent variables have already been solved [4].

3 Automatic differentiation

First and second order derivatives of the functions \( f, g \) and \( h \) are required in order to calculate the components of eqn (3). The robustness and convergence rate of the Newton algorithm are highly dependent on the precision of these derivatives. When numerical differentiation is used, each variable has to be shifted. The size of this perturbation has a significant influence in the precision of the derivative and a suitable value for its size may be difficult to estimate [5]. The calculation of the Hessian matrix requires a large number of perturbations and function evaluations. To avoid errors associated with numerical differentiation, the user of the optimization code might be obliged to supply derived functions. When the size of the NLP is large, this technique becomes cumbersome and error prone. A possible solution to these problems is the development of code that applies the rules of differentiation. This technique is called automatic differentiation (AD) and is difficult to implement, due to the complexity of the algorithms involved. Object oriented programming techniques provide tools that relieve the task of programming the differentiation of user-supplied functions. In the present work, an expression parser is used to decompose each function into operands and operators (see Section 5). The operators are redefined in order to apply the rules of differentiation.

4 Rall numbers

A Rall number is a set of information that contains the numerical value of an operand, the numerical value of its gradient and the numerical value of its Hessian. When an operator is applied to an operand or to a pair of operands, the differentiation rules are used in order to perform the necessary tasks that lead to the correct evaluation of the gradient and Hessian of the result of the operation. In the present work a Rall number is implemented as a C++ class [1]. Its data members and the multiply operator are shown in the following example involving the functions \( f(x_1, x_2) \) and \( g(x_1, x_2) \) (see Table 1). Eqns (5) and (6) correspond to the evaluation of the first and second derivatives of the product of a pair of functions.

\[
\frac{\partial}{\partial x_1} (fg) = \frac{\partial f}{\partial x_1} g + f \frac{\partial g}{\partial x_1}
\]

\[
\frac{\partial^2}{\partial x_1 \partial x_2} (fg) = \frac{\partial^2 f}{\partial x_1 \partial x_2} g + \frac{\partial f}{\partial x_1} \frac{\partial g}{\partial x_2} + \frac{\partial f}{\partial x_2} \frac{\partial g}{\partial x_1} + f \frac{\partial^2 g}{\partial x_1 \partial x_2}
\]
When a Rall number is a constant, its data members are initialized with the following values (see Table 1):

\[ x = \text{constant value}; \quad v = [0,0]; \quad m = [[0,0],[0,0]] \]

When a Rall number represents one of the existing variables \( (x_1 \text{ or } x_2 \text{ in this example}) \), its initialization becomes:

\[ x = \text{value of } x_1; \quad v = [1,0]; \quad m = [[0,0],[0,0]] \]

or

\[ x = \text{value of } x_2; \quad v = [0,1]; \quad m = [[0,0],[0,0]] \]

**Table 1:** Abridged definition and implementation of the class `CRall`.

```cpp
class CRall {
    double x; // Operand value
    double v[2]; // df/dx1, df/dx2
    double m[2][2]; // d2f/dxi dxj

public:
    CRall CRall::operator* (const CRall & g) const {
        CRall t;
        t.x = x * g.x;
        t.v[0] = v[0]*g.x + x*g.v[0];
        t.v[1] = v[1]*g.x + x*g.v[1];
        t.m[0][0] = m[0][0]*g.x + v[0]*g.v[0] + v[0]*g.v[0] + x*g.m[0][0];
        t.m[0][1] = m[0][1]*g.x + v[1]*g.v[1] + v[1]*g.v[1] + x*g.m[0][1];
        t.m[1][0] = m[1][0]*g.x + v[0]*g.v[0] + v[0]*g.v[0] + x*g.m[1][0];
        t.m[1][1] = m[1][1]*g.x + v[1]*g.v[1] + v[1]*g.v[1] + x*g.m[1][1];
        return t;
    }
};
```

All the other operations involving Rall numbers are implemented in a similar manner (e.g., addition, subtraction, division, exponentiation, trigonometric functions, logarithm). In the current version of the optimization software, vectors and matrices are stored in sparse arrays.

## 5 Expression parser

When a computer program needs to evaluate an expression, the most straightforward strategy is to hard code the expression before the compilation of the module. Whenever the expression has to be modified, a new compilation is required. The alternative of coding an expression parser has also some disadvantages, such as code complexity and a decreased performance. An object oriented programming language (e.g., C++) can lighten the task of coding and reusing complex algorithms, by means of supplying tools that allow for an higher
abstraction level. Examples of such techniques are operator overloading, templates, inheritance and polymorphism [2].

The expression parser described in this Section is based on the work of Rogers [6]. Some enhancements and new features were added, such as the support for the most common intrinsic functions (e.g., \( \sin, \cos, \sqrt{\cdot}, \log, \text{pow} \)) and the implementation of scaling techniques (see Section 6).

Three types of entities can be extracted from an expression: constants, variables and operators. According to the priority rules of the parentheses and operators, all the objects are inserted in a binary tree (see Figure 1). A symbol table is an independent object where the names of all the variables and their values are stored. The evaluation of the root object causes a postorder traversal of the binary tree. When a variable is found, the supplied symbol table is searched and the corresponding value is extracted. These operations produce the result of the evaluation of the whole expression and are exemplified with eqn (7) (see also Figure 1).

\[
\frac{x_1 + 8 x_2}{(6 - x_3^2)} \tag{7}
\]

Figure 1: Binary tree obtained from eqn (7).

In the tree shown in Figure 1, the terminal nodes are replaced with Rall numbers (see Section 4). The operations between Rall numbers are subject to the same priority rules that are used in the evaluation of the expression. Since the Rall numbers also operate with the gradient and Hessian of the function, their numerical values are simultaneously evaluated. In order to avoid the systematic manipulation of all the variables that are present in an optimization problem, a data member is used to store the list of variables that need to be considered in each function.

Figure 2 shows the hierarchy diagram of the classes that implement the behavior of all the expression components. The base class \( \text{Exp\_Obj} \) defines the common interface of all the derived classes. Polymorphism is used to facilitate
the manipulation of the expression objects, regardless of their specified type or functionality.

During the construction of the binary tree, the following tasks are performed: read a token, check the type of the token and add the token to a stack. Variables and literals are added to the expression stack and operators are added to the operator stack. Each operator and each parenthesis has its priority level and is processed accordingly.

![Hierarchy diagram of the expression parser.](image)

Figure 2: Hierarchy diagram of the expression parser.

### 6 Scaling

In a mathematical program several types of variables appear in the objective function and constraints. It is common to have variables and function values in different units, exhibiting dissimilar orders of magnitude (e.g., Young's modulus and rotation angle). When this happens the optimization algorithms may experience numerical instabilities, slow convergence or even a global failure. These problems are caused by round off errors, due to the limited precision of the computer calculations and due to a high condition number in some matrices. Two simple techniques may be implemented to alleviate these problems, namely
variable scaling and constraint normalization. Variable scaling can be performed by means of the replacement of each variable \(x_i\) by the product \(Z_i y_i\) where \(Z_i\) is the scaling factor (usually the initial value of \(x_i\)), and \(y_i\) is the new variable in the mathematical program, having a unitary initial value. Constraint normalization consists in the multiplication of the objective function and each constraint by a constant, whose value sets the initial Euclidean norm of the gradient equal to one. These techniques are illustrated with the nonlinear program (8).

\[
\begin{align*}
\text{Min.} & \quad 2000 \ x_i \\
\text{subject to} & \\
-x_i + 200 + x_3^2 & = 0 \\
x_2 - 0.2 + x_3^2 & = 0 \\
-10x_i x_2 + 500 & = 0
\end{align*}
\]

(8)

Assuming the initial solution \((500, 0.1, \sqrt{300}, 0.1)\), and applying the aforementioned techniques, (8) is replaced by the NLP (9) [3].

\[
\begin{align*}
\text{Min.} & \quad y_1 \\
\text{subject to} & \\
-0.640y_1 + 0.256 + 0.384 y_2^2 & = 0 \\
0.447y_2 - 0.894 + 0.447 y_2^2 & = 0 \\
-0.707y_1 y_2 + 0.707 & = 0
\end{align*}
\]

(9)

The scaled version of the mathematical program is numerically more stable and has a faster convergence [3]. The solution of the original NLP (8) can be recovered from the solution of (9). The implementation of the scaling of the variables implied the inclusion of a new data member in each entry of the symbol table in order to store the scaling factor. The constraint normalization factor is stored in a data member of the class BaseExpression (see Figure 2). The computation of derivatives has to take into account these modifications of the expressions.

7 Numerical example

A simple shape optimization problem is presented in order to illustrate the main characteristics of the proposed algorithm. It consists on the minimization of the weight of a two bar truss, subjected to a single load case (see Figure 3) [7]. Dead load is not considered and buckling may be ignored due to the fact that both bars are under tension. The nonlinear program describing the optimization problem is defined in (10).
Min. \( w(x_1, x_2) = C_1 x_1 \sqrt{1 + x_2^2} \)

subject to

\[
\sigma_1(x_1, x_2) = C_2 \sqrt{1 + x_2^2} \left( \frac{8}{x_1} + \frac{1}{x_1 x_2} \right) \leq 1
\]

\[
\sigma_2(x_1, x_2) = C_2 \sqrt{1 + x_2^2} \left( \frac{8}{x_1} - \frac{1}{x_1 x_2} \right) \leq 1
\]

\(0.2 \leq x_1 \leq 4.0; \quad 0.1 \leq x_2 \leq 1.6\)

Figure 3: Shape optimization of a two bar truss.

In the nonlinear program (10), \( C_1 = 1.0 \) and \( C_2 = 0.124 \). These values are not modified during the iteration process. The initial values of \( x_1 \) and \( x_2 \) are 1.5 cm\(^2\) and 0.5 m respectively. Table 2 shows the syntax of the input file of the optimization program, containing the description of the nonlinear program (10). The last four lines declare the type of each variable, its initial value and its name.

Table 2: Input file of the optimization program.

```
# Main title
Shape optimization of a two bar truss

# N. of eq. constr.; N. of ineq. constr.
0 6

# Objective Function
C1*x1*sqrt(1+x2^2);

# Allowable stress - bar 1
C2*sqrt(1+x2^2)*(8/x1+1/x1/x2)-1;
```

Table 2 (cont.): Input file of the optimization program.

```
# Allowable stress - bar 2
C2*sqrt(1+x2^2)*(8/x1-1/x1/x2)-1;
# Minimum x1
-x1+0.2;
# Maximum x1
x1-4.0;
# Minimum x2
-x2+0.1;
# Maximum x2
x2-1.6;
# N. of variables
4
SUBSTITUTED, 1.000, C1;
SUBSTITUTED, 0.124, C2;
INDEPENDENT, 1.5, x1;
INDEPENDENT, 0.5, x2;
```

Figure 4 shows the iteration history of the objective function whose optimal value is \( w = 1.509 \). The intermediate solutions exhibiting objective function values that are smaller than the optimum correspond to infeasible points. Since the Lagrange-Newton method is based on the search of a saddle point of the Lagrangian, there is no guarantee that the optimum be approached from the interior of the feasible region. The optimal value of the independent design variables is \( x_1 = 1.412 \text{ cm}^2 \) and \( x_2 = 0.377 \text{ m} \).
Several optimization problems described in the books of Arora [8] and Azevedo [3] were solved with this computer code and the results were compared with the solutions obtained with other computer programs. Since a good agreement was always obtained, the proposed algorithm and the complex C++ coding have been validated.

8 Conclusion

The work presented in this paper consists on the combination of a robust second-order optimization method, an object oriented parser and an automatic differentiation algorithm based on operator overloading and Rall numbers. Some preliminary numerical experimentation indicates that the code is very versatile, i.e., its adaptation to new types of optimization problems is very easy. Code maintenance and the implementation of alternative numerical techniques are facilitated by the object oriented design and its inherent features, such as polymorphism and inheritance. Some work is still needed to improve the global efficiency of the optimization process.

References

Coordination process in decomposed optimal design problems

C. Martins¹ & J. Guillot²
Instituto PIAGET, Portugal
Laboratoire de Génie Mécanique de Toulouse, France

Abstract

The optimal design of a mechanical system consists in seeking "The best solution". This solution will have to satisfy as far as possible the optimisation criteria set by the designer. The difficulty occurs when the optimisation problem to be treated includes a great number of design variables and/or a great number of constraint functions. This disadvantage is overcome by making a systematic decomposition of the initial optimal design problem[4]. Once decomposed, the difficulty then lies in the management of these subsystems underlinked with multidirectional interactions: the non-hierarchical problems. The authors have proposed various Coordination Strategies which depends on the links type such as linking variables exclusively, linking constraints functions or an association of linking variables and constraints functions.

For each one, an algorithm has been created for the Coordination Strategy and a mathematical program (DOT) are used to solve each subsystem. In order to demonstrate these approaches, we included, for each case, an application of non-linear optimization problems.

Introduction

The main interest of the decomposition method lies in the rapidity of resolving the subsystems. The difficulty consists in the management of these subsystems. In fact, the decomposition of optimal design problems takes various forms. For example for some the subsystems are independents, for others it is necessary to organise the linking variables exclusively and others still may relate to the arrangement of the
linking constraint functions. Finally it is possible to encounter the association of linking variables and constraints. It is for these multiple reasons that we have carried out various collaboration procedures in order to determine the optimal values of the design variables in the shortest possible time. In the final section, conclusions are drawn and proposals for future research are suggested.

2 Coordination Process

As we said before, there are four classes of linking systems and we go now to propose various Coordination Strategies which depend on the nature of the links between the subsystems.

2.1 Independent subsystems

This case represents perhaps the most ideal one in decomposition, but it is necessary all the same to explain its mode of resolution. Grouping of subsystems into a matrix representation [2], having neither linking variables nor linking constraints (fig. 1) is the most favourable situation but also the rarest since there is no interaction between the sub-groups.

All the sub-problems are thus independent and are resolved by processing the various subsystems (fig. 2). In the present case, it is sufficient to solve the multiple subsystems independently in order to obtain the optimum for each sub-problem and to regroup them to acquire the optimal solution of the design problem. In this type of system, the optimal solution is always the sum of the various optimal solutions. With this in mind, we have studied the example suggested by Wagner [7] relating to a marketing strategy problem. The primary formulation of the optimisation problem is as follows:

\[
\min f = \sum_{i=1}^{5} f_i \\
\text{with} \\
f_1 = -16.212 - 3.937 \cdot X_2^{0.91} \cdot X_1^{1.31} \\
f_2 = 2.1 \times 10^4 \cdot X_1^{-0.95} \cdot X_6^{-0.68} \cdot X_8^{-0.34} \cdot X_7^{-0.28} \\
f_3 = 3.05 \times 10^3 \cdot X_1^{0.18} \cdot X_1^{1.76}
\]
\[ f_4 = 4.6 \times 10^{-3} X_5^{-0.9} X_{14}^{-1.1} \]
\[ f_5 = 5.3 \times 10^{-3} X_6^{-0.76} X_{10}^{-1.12} \]

subject to:
\[ g_1: x_1 + x_2 - 0.35 \leq 0 \quad \text{and} \quad g_{10,11}: 0.2 \leq x_5 \leq 0.35 \quad \text{and} \quad g_{20,21}: 0.3 \leq x_{10} \leq 0.8 \]
\[ g_2: x_3 + x_4 - 0.35 \leq 0 \quad \text{and} \quad g_{12,13}: 0.1 \leq x_6 \leq 1.35 \]
\[ g_3: x_3 + x_4 + x_5 - 0.6 \leq 0 \quad \text{and} \quad g_{14,15}: 0.6 \leq x_7 \leq 1.3 \]
\[ g_4: x_9 - x_{15} - 2 \leq 0 \quad \text{and} \quad g_{16,17}: 0.6 \leq x_8 \leq 1.3 \]
\[ g_5: x_6 + x_2 - 0.35 \leq 0 \quad \text{and} \quad g_{18,19}: 0.8 \leq x_9 \leq 1.25 \]
\[ g_{i=1,4}: 0.1 - x_i \leq 0, \quad i=1,...,4 \]

This particular optimisation problem includes 15 design variables and 31 constraint functions and an overall solution is likely to be complex. To determine the optimal solution, we start with the decomposition process [4]. This mechanism enables us to obtain 4 independent subsystems which are represented in figure 3.

This parcelling out is carried out while carefully avoiding consideration of the constraints limiting the design variables. These constraints will be subsequently used
in the resolution of the various independent subsystems. The results of the collaboration process are proposed in table 1.

The values of the design variables have been determined in the case of a global resolution of the problem and in the case of a decomposed resolution thus enabling us to compare the results.

It will be noticed that these results are similar and correspond to the values obtained by Wagner. Only the computing time is different: decomposition obtains a time gain of about 15% for this type of example.

In conclusion, the decomposition method slightly reduces resolution time while determining correctly the optimal values of the design variables.

2.2 Non-Hierarchical Coordination

In the case of non-hierarchical problems (fig. 4), there are multidirectional interactions between the subsystems which lead to the problem of system management.

To deal with this, coordination algorithms have been set up to process the organisation of the non-hierarchical problems.

![Diagram of 4 subsystems with linking variables](image)

**Figure 4**: Non-Hierarchical Problem.

2.2.1 Linking variables

The main difficulty appears with the presence of variables binding the various subsystems (Fig. 5). Indeed, until now it has not been possible to deal with such problems of organisation of the various groups present.

![Diagram of variable types](image)

**Figure 5**: Problem with linking variables.

The problems have thus been arranged in a coherent process in order to resolve them effectively and to provide the optimal solution for the overall design problem (fig. 6).
This method is based on a simple idea according to which the optimal solution to a global problem is more easily obtained by introducing linking variables with optimal values into the various subsystems. Accordingly, we begin with a decomposed system into which the linking variables are incorporated. An iterative process is then applied where the linking variables depend on three parameters: Lagrange multipliers and the partial derivative value of the objective and constraint functions.

The test of convergence relates to the objective function values for two successive iterations. Once this criterion is validated, the optimal solution of the design problem is obtained.

Let us apply this process to an example taken from [1]. It concerns the optimal design of a pressure vessel. The objective is to minimise the construction cost while
respecting the constraints envisaged by the Pressure Vessel Code (1962) and the ASME Boiler.
The mathematical model to optimise is as follows:

\[
\min \quad f = \sum_{i=1}^{4} f_i = 0.662 R \cdot t_s \cdot L + 1.777 R^2 \cdot t_h + 1.58 t_s^2 \cdot L + 19.84 R \cdot t_s^2
\]

subject to:

\[
\begin{align*}
g_1 &= 0.0193 \cdot R - t_s \leq 0 \\
g_2 &= 413000 - R^2 \cdot L \leq 0 \\
g_3 &= 0.00417 \cdot L - 1 \leq 0 \\
g_4 &= 0.131 \cdot R - t_h \leq 0 \\
g_5 &= 0.1 - L \leq 0 \\
g_6 &= 0.1 - t_s \leq 0 \\
g_7 &= 0.1 - t_h \leq 0 \\
g_8 &= 0.1 - R \leq 0 
\end{align*}
\]

by applying the decomposition process, we obtain two independent subsystems with their corresponding locals variables:

\[
\begin{align*}
GV_1 &= \{t_s, L\}, \\
GF_1 &= \{g_1, g_2, g_3, g_5, g_6\} \quad \text{and} \quad GV_2 = \{t_h\}, \\
GF_2 &= \{g_4, g_7\} 
\end{align*}
\]

as well as a variable binding the two subsystems: \( R \).

By using the coordination process, the optimal values of the design variables can be obtained with a very short computing time (table 2).

Table 2: Decomposition Results.

<table>
<thead>
<tr>
<th></th>
<th>Global</th>
<th>Wagner</th>
<th>Martins &amp; Guillot</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f^* \cdot 10^{-4} )</td>
<td>2.268547</td>
<td>2.268547</td>
<td>2.268547</td>
</tr>
<tr>
<td>( R^* )</td>
<td>41.499518</td>
<td>41.499518</td>
<td>41.499518</td>
</tr>
<tr>
<td>( L^* )</td>
<td>239.80815</td>
<td>239.80815</td>
<td>239.80815</td>
</tr>
<tr>
<td>( t_s^* )</td>
<td>0.800941</td>
<td>0.8009407</td>
<td>0.8009407</td>
</tr>
<tr>
<td>( t_h^* )</td>
<td>5.436437</td>
<td>5.436437</td>
<td>5.436437</td>
</tr>
<tr>
<td>( time )</td>
<td>6.98 s</td>
<td>n.c.</td>
<td>4.66 s</td>
</tr>
</tbody>
</table>

These results are compared with the values determined by a global resolution of the optimisation problem and with the values determined by Wagner [7]. The latter has employed a method based on monotony of the functions and approximations using K-S functions introduced by Kreisselmeier and Steinhauser [3].

This new coordination process is characterised by the fact that it is not necessary to know the properties of monotony of the functions. It is appropriate for loosely limited optimal design problems. In the case of problems which include a very large number of linking variables, the process has a high response time, even higher than the resolution of the global problem.

2.2.2 Linking by constraint functions

Here, the sub-groups are linked by constraint functions: \( g_i(x_i) \), i.e. these functions depend on variables belonging to distinct subsystems (fig. 7).
The optimisation problem can be solved by applying of the process presented in Figure 8.

It concerns an iterative algorithm which initially determines the values of the variables belonging to the linking constraints \( (x_{gl}) \).
A second stage consists in using the linking constraints and determining the remaining design variables. If the test of convergence is not checked, the boundaries of the variables $x_{gl}$ are modified; the solutions interval is then reduced. These boundary variations depend on the Khun-Tucker conditions which gives the relation:

$$\frac{df}{dx_{gl}} = \frac{\partial f}{\partial x_{gl}} + \lambda \cdot \frac{\partial g_i}{\partial x_{gl}}$$

In the first case, if $\frac{df}{dx_{gl}} > 0$ then the upper limit takes the $x_{gl}^*$ value and the lower limit remains unchanged. In the second case (i.e. $\frac{df}{dx_{gl}} < 0$), it is the lower limit which takes the $x_{gl}^*$ value and the upper limit keeps its preceding value.

This collaboration method is applied successfully to an example based on a car disc brake design. The disc brake studied here is based on the model suggested by Siddall [5], which is composed of a caliper holding a hydraulic cylinder and piston assembly and our intention is to minimise the stopping time.

The mathematical model has the following form:

$$\begin{align*}
\min R(\bar{x}) &= \frac{30459.28}{D_p^2P_o^2} + 0.01a = f_1 + f_2 \\
\text{subject to:} & \quad g_1(\bar{x}) = D - 12 \leq 0 \\
& \quad g_2(\bar{x}) = \frac{d}{2} - \frac{D}{2} + R \leq 0 \\
& \quad g_3(\bar{x}) = \frac{d}{2} - R + 1.5 \leq 0 \\
& \quad g_4(\bar{x}) = \frac{D_p}{2} - R + 1.75 \leq 0 \\
& \quad g_5(\bar{x}) = \frac{\pi}{4} \frac{D_p^2P_o}{4R} - 400 \leq 0 \\
& \quad g_6(\bar{x}) = \frac{83.67}{D_p^2a} - 405 \leq 0 \\
& \quad g_7(\bar{x}) = \frac{\pi}{2} \frac{1}{2} P_o D_p^2 - 10500 \leq 0
\end{align*}$$

where, $R$, $d$, $D_p$, $a$, $P_o$, $D$ correspond to the lining center line radius, lining diameter, piston diameter, disc thickness, oil pressure, and outside disc diameter, respectively.

The block-angular structure obtained is the following:

- **SP-1**: $\{D_p, P_o\}$ with functions $\{f_1, g_7\}$
- **SP-2**: $\{R, d, a, D\}$ with functions $\{f_2, g_1, g_2, g_3, g_5, g_6\}$

and two coupling functions constraints: $g_4(R, D_p)$, $g_6(R, d, D_p, P_o)$.

The optimal solution is sought using the process presented above. We start by solving the first subsystem with an initial point identical to that chosen by Siddall $x_0 = \{4, 2.6, 2, 0.3, 500., 11.\}$. Then the values of variables $D_p^*$ and $P_o^*$ are introduced into the second subsystem which includes the constraints $g_4$ and $g_6$. If convergence is not satisfied, the boundary variables $D_p$ and $P_o$ are reduced. Each optimisation is carried out using software ADS [6].

The results obtained by starting with this collaboration process and the computing times are compared with a global resolution of the problem and the values determined by Siddall (table 3). Note that among the three results suggested, the col-
Collaboration method makes it possible to determine the optimum with a reduced computing time.

Table 3: Comparison of the results.

<table>
<thead>
<tr>
<th></th>
<th>Global Optimisation</th>
<th>Collaborative Optimisation</th>
<th>Siddall</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f^*(s.)$</td>
<td>4.5596</td>
<td>4.5563</td>
<td>4.5576</td>
</tr>
<tr>
<td>R (inches)</td>
<td>3.8546</td>
<td>4.0</td>
<td>3.9327</td>
</tr>
<tr>
<td>D (inches)</td>
<td>2.5997</td>
<td>2.6</td>
<td>2.5766</td>
</tr>
<tr>
<td>$D_p$ (inches)</td>
<td>1.9140</td>
<td>1.9051</td>
<td>1.935</td>
</tr>
<tr>
<td>a (inches)</td>
<td>0.29983</td>
<td>0.2</td>
<td>0.2729</td>
</tr>
<tr>
<td>$P_o$ (Psi)</td>
<td>480</td>
<td>467</td>
<td>460</td>
</tr>
<tr>
<td>D (inches)</td>
<td>11</td>
<td>11</td>
<td>10.47015</td>
</tr>
<tr>
<td><strong>Time</strong></td>
<td>1min. 21s</td>
<td>17.5s</td>
<td>nc</td>
</tr>
</tbody>
</table>

This new collaboration process is applicable to all the subsystems including linking constraints. It can reduce definition intervals of variable coupling functions until reaching the optimal solution: it is the "reduction of solutions area" envelope. Moreover, it is fast, easy to use and achieves satisfactory optimum variable values.

2.2.3 Association of linking variables and constraint functions

In this case, the decomposed problems contain a group of linking design variables and functions constraints (fig. 9).

The idea here is to implement a "pseudo-hierarchical" iterative coordination process. In this case, we consider a master problem which treats the optimal linking variables and then lots of sub-problems which determine the optimal design local variable values. The functions constraints play a role in checking the optimum (convergence test). If the test is concluded, optimal values are obtained. If not, an update of the linking variables is made reducing their definition interval (fig. 10). But this idea must be verified and also adjusted in order to obtain the optimal design variable values.
3 Conclusion

We note that the computing time, in the case of a decomposed problem, is lower than the time taken in the case of the complete problem resolution. The optimal values are close to the exact values. The optimisation problems for which the determination of the linking variables is a priori impossible can be solved by this decomposition process in a systematic and structured way, without knowledge of the mechanical system. Each coordination process implemented here efficiently solves the management problem of the subsystems. Subsequent research could lead to the design of a software making it possible to determine the optimal decomposition and to link it automatically with the appropriate coordination process. It would be also useful to direct research towards the study of optimal design problems including discrete and/or integer linking variables. The final result would be the creation an assisted design software.

References