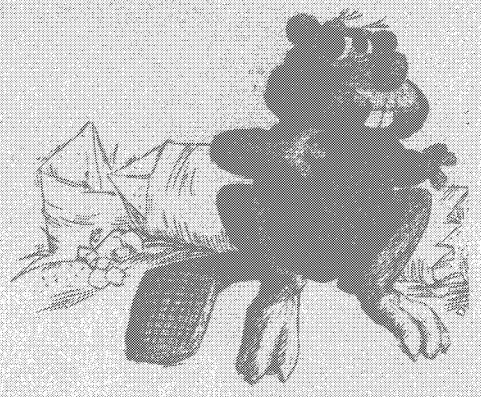
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT



A Proposal for a Standard Data Input Format for Large-Scale Nonlinear Programming Problems

A.R. Conn, N.I.M. Gould
P.L. Toint
Ph.

CS-89-61

March 1990

A Proposal for a Standard Data Input Format for Large-Scale Nonlinear Programming Problems.

Andrew R. Conn †, Nicholas I. M. Gould ‡ and Philippe L. Toint §

Abstract

We propose a standard data input format for large-scale nonlinear programming problems. Our intention is to allow users to concentrate their efforts in modelling nonlinear phenomena rather than having to tailor their problems to one of many input formats in use at present. In addition it will allow designers of nonlinear programming algorithms to anticipate the input that they are likely to receive. The standard is intended to be fully compatible with the MPS linear programming standard currently in widespread use.

† Department of Combinatorics and Optimization, University of Waterloo, Ontario, Canada.
The research of this author was supported in part by NSERC grant A8639.

‡ Computer Science and Systems Division, A.E.R.E. Harwell, Oxford, England.

§ Department of Mathematics, Facultés Universitaires ND de la Paix, B-5000, Namur, Belgium.

March 1990

1.1 Introduction

The mathematical modelling of many real-world applications involves the minimization or maximization of a function of unknown parameters or variables. Frequently these parameters have known bounds; sometimes there are more general relationships between the parameters. When the number of variables is modest, say up to ten, the input of such a problem to an optimization procedure is usually fairly straightforward. Unfortunately many applications areas now require the solution of optimization problems with thousands of variables; in this case merely the input of the problem data is extremely time-consuming and prone to error. Moreover, the mathematical programming community is only now designing algorithms for solving problems of this scale.

This work was motivated directly by the difficulties the authors were experiencing entering test examples to their large-scale nonlinear optimization package (SBMIN/LANCELOT, in preparation, see Conn, Gould and Toint, 1988a,b, 1990, Conn, Gould, Lescrenier and Toint, 1987). It soon became apparent that if others were to be encouraged to carry out similar tests and even enticed to use our software, the process of specifying problems had to be considerably simplified. Thus we were inevitably drawn to provide a preliminary version of what is described in the present paper: a standard data input format (SDIF) for nonlinear programming problems, together with an appropriate translator from the input file to the form required by the authors' minimization software. While understandably reflecting our views and experience, the present proposal is intended to be broadly applicable.

During the subsequent (and successive) stages of development of these preliminary ideas, various important considerations were discussed. These strongly influenced the present proposal.

- There are many reasons for proposing a standard input format. The most obvious one is maybe the increased consistency in coding nonlinear programming problems, and the resulting improvement in code reliability. As every problem is treated in a similar and standardized way, it is more difficult to overlook certain aspects of the problem definition. The providing of an SDIF file for a given problem also allows some elementary (and very often helpful) automatic error and consistency checking.

- A further advantage of having a standard input format is the long awaited possibility of having a portable testbed of meaningful problems. Moreover, a testbed that can be expected to grow. The authors soon experienced the daunting difficulties associated with specifying large scale problems – not only the difficulty of writing down the specification correctly but also the actual coding (and frequent re-coding) of a particular problem which often results in non-trivial differences between the initial and final data. These differences could be a major obstacle to valid comparisons between competing optimization codes. By contrast, having an SDIF file allows simple and unambiguous data transfer via diskette, tape or electronic mail. The success of the NETLIB and Harwell/Boeing problem collections for linear programming and sparse linear algebra (see, Gay, 1985, and Duff, Grimes and Lewis, 1989) is a good recommendation for such flexibility. The formality required by the SDIF approach may admittedly appear formidable for very simple problems, but is soon repaid when dealing with more complex ones.

- Of course, the SDIF format should cover a large part of the practical optimization problems that users may want to specify. Explicit provision should be made not only for unconstrained problems, but also for constraints of different types and complexity: simple bounds on the variables, linear and/or nonlinear equations and inequalities should be handled without trouble. Special structure of the problem at hand is also a mandatory part of an SDIF file. For example, the structure of least-squares problems must be described in an exploitable form. Sparsity of relevant matrices and partial separability of involved nonlinear functions must be included in the standard problem

description when they are known. Finally, the special case of systems of nonlinear equations should also be covered.

· The existence and worldwide success of the MPS standard input format for linear programming must be considered as a *de facto* basis for any attempt to define an SDIF for nonlinear problems. The number of problems already available in this format is large, and many nonlinear problems arise as a refinement of existing linear ones whose linear part and sparsity structure are expected to be described in the MPS format. It therefore seems reasonable to require that an SDIF for nonlinear programming problems should conform to the MPS format. We were thus led to chose a standard that corresponds to MPS, augmented with additional constructs and structures, thus allowing nonlinearity, and the general features that we wished, to be described properly.

· The requirement of compatibility with the MPS format has a number of consequences, not all of which are pleasant. The first one is that the new SDIF must be based on fixed format for the SDIF file. Indeed, blanks are significant characters in MPS, when they appear in the right data fields, and cannot be used a general separators for free format input in any compatible system. The second one is the *a priori* existence of a style for keywords and overall layout of the problem description, style which is not always ideally suited to the description of nonlinear problems. Thus our present proposal accepts these limitations.

· The SDIF should not be dependent on a specific operating system and/or manufacturer. In this respect, it must avoid relying on tools that may be excellent but are too specific (yacc and lex, for example). This of course does not prevent any implementation of an SDIF interpreter to use whatever facilities are locally available.

The authors are very well aware of the shortcomings of the SDIF approach when compared to more elaborate modelling languages (see, for example, GAMS (Brooke, Kendrick and Meeraus, 1988), AMPL (Fourer, Gay and Kernighan, 1987) and OMP (Decker, Louveaux, Mortier, Schepens and Looveren, 1987)). These probably remain the best way to allow easy and error free input of large problems. However, we contend that there is at present no language in the public domain which satisfactorily handles the nonlinear aspects of mathematical programming problems. While the advent of a tool of this nature is very much hoped for, it nevertheless seems necessary to provide something like the SDIF now. This (we hope, intermediate) step is indeed crucial for the development and comparison of algorithms for solving the large scale nonlinear problems, without which a more elaborate tool would be meaningless anyway. The SDIF for nonlinear problems may also be considered as a first attempt to specify the minimal structures that should be present in a true modelling language for such problems. It is also of interest to develop a relatively simple input format, given that researchers developing new optimization methods may have to implement their own code for translating the SDIF file into a form suitable for their algorithms. At this level, some compromise between completeness and simplicity seems necessary. Finally, the existence and availability of modelling languages for linear programming for a number of years has not yet made the MPS format irrelevant.

Hence, the reader should be aware that what sometimes appear as unnecessarily restrictive "features" of the proposed standard are often direct consequences of the considerations outlined above.

As we have already mentioned, structure is an integral and significant aspect of large-scale problems. Structure is often equated with sparsity; indeed the two are closely linked when the problem is linear. However, sparsity is not the most important phenomenon associated with a nonlinear function; that role is played by invariant subspaces. The *invariant subspace* of a function $f(\mathbf{x})$ is the set of all vectors $\mathbf{w}$ for which $f(\mathbf{x}+\mathbf{w})=f(\mathbf{x})$ for all possible vectors $\mathbf{x}$. This phenomenon encompasses function sparsity. For instance, the function

$$f(x_1, x_2, \ldots, x_{1000}) = x_{500}^2$$

has a gradient and Hessian matrix each with a single nonzero, has an invariant subspace of dimension 999, and is, by almost any criterion, sparse. But the function

$$f(x_1, x_2, \ldots, x_{1000}) = (x_1 + \ldots\ldots + x_{1000})^2$$

has a completely dense Hessian matrix but still has an invariant subspace of dimension 999, the set of all vectors orthogonal to a vector of ones. The importance of invariant subspaces is that nonlinear information is not required for a function in this subspace. We are particularly interested in functions which have large (as a percentage of the overall number of variables) invariant subspaces. This allows for efficient storage and calculation of derivative information. The penalty is, of course, the need to provide information about the subspace to an optimization procedure.

A particular objective function $F(\mathbf{x})$ is unlikely to have a large invariant subspace itself. However, many reasonably behaved functions may be expressed as a sum of *element* functions, each of which does have a large invariant subspace. This is certainly true if the function is sufficiently differentiable and has a sparse Hessian matrix (see, Griewank and Toint, 1982). Thus, rather than storing a function as itself, it pays to store it as the sum of its elements; the elemental representation of a particular function is by no means unique and there may be specific reasons for selecting a particular representation. Specifying Hessian sparsity is also supported in the present proposal, but we believe that it is more efficient and also much easier to specify the invariant subspaces directly.

In this paper, we shall consider two types of nonlinear programming problem; the data required for the two is identical in form and the actual optimization problem solved is left to the optimization procedure.

The first problem is the minimization or maximization of an objective function of the form

$$F(\mathbf{x}) = \sum_{i=1}^{n_g} g_i\left(\sum_{j \in J_i} w_{i,j} f_j(\mathbf{x}_j) + \mathbf{a}_i^T \mathbf{x} - b_i \right), \qquad \mathbf{x} = (x_1, x_2, \ldots, x_n) \tag{1.1.1}$$

within the "box" region

$$l_i \le x_i \le u_i, \quad 1 \le i \le n \tag{1.1.2}$$

and where either bound on each variable may be infinite. The univariate functions g_i are known as *group functions*. The argument

$$\sum_{j \in J_i} w_{i,j} f_j(\mathbf{x}_j) + \mathbf{a}_i^T \mathbf{x} - b_i$$

is known as the i–th *group*. The functions f_j, $j = 1, \ldots, n_e$, are called *nonlinear* elements functions. They are functions of the problem variables $\mathbf{x}_j$, where the $\mathbf{x}_j$ are either small subsets of $\mathbf{x}$ or such that f_j has a large invariant subspace for some other reason. Each index set J_i is a subset of $\{1, \ldots\ldots, n_e\}$. The constants $w_{i,j}$ are known as *weights*. Finally, the function $\mathbf{a}_i^T \mathbf{x} - b_i$ is known as the *linear* element for the i–th group.

The second problem is the minimization or maximization of an objective function of the form

$$g_o\left(\sum_{j \in J_o} w_{o,j} f_j(\mathbf{x}_j) + \mathbf{a}_o^T \mathbf{x} - b_o \right), \qquad \mathbf{x} = (x_1, x_2, \ldots, x_n) \tag{1.1.3}$$

within the "box" region (1.1.2) and where the variables are required to satisfy the extra conditions

$$g_i\left(\sum_{j \in J_i} w_{i,j} f_j(\mathbf{x}_j) + \mathbf{a}_i^T \mathbf{x} - b_i \right) = 0 \quad (i \in I_E) \tag{1.1.4}$$

and

$$0 \left\{\begin{matrix}\leq\\ \geq\end{matrix}\right\} g_i\Big(\sum_{j\in J_i} w_{i,j} f_j(\mathbf{x}_j) + \mathbf{a}_i^T\mathbf{x} - b_i\Big) \left\{\begin{matrix}\leq\\ \geq\end{matrix}\right\} r_i, \quad (i\in I_I) \tag{1.1.5}$$

for some index sets I_E and I_I and (possibly infinite) values r_i. Again the functions g_i are known as *group functions*, although it it more common to call those in (1.1.4) equality constraint functions, those in (1.1.5) inequality constraint functions and that in (1.1.3) the objective.

As can be seen, the data for either problem is of the same form. We need to specify the group functions, linear and nonlinear elements and the way that they all fit together.

In §1 of this paper, we explain how we propose to exploit the structure in problems of the form (1.1.1)–(1.1.5). We do this both in general and with respect to a number of examples. Details of the way in such structure may be expressed in a standard data input format follow in §2. The input of nonlinear information for element and group functions is covered in §3 and §4 respectively. The relationship to existing work is presented in §5 and conclusions drawn in §6. A further set of examples and the resulting input files are given in an appendix to the paper.

1.2 Problem, elemental and internal variables

A nonlinear element function f_j is assumed to be a function of the problem variables $\mathbf{x}_j$, a subset of the overall variables $\mathbf{x}$. Suppose that $\mathbf{x}_j$ has n_j components. Then one can consider the nonlinear element function to be of the *structural* form $f_j(v_1,\ldots,v_{n_j})$, where we assign $v_1=\mathbf{x}_{j1},\ldots,v_{n_j}=\mathbf{x}_{jn_j}$. The *elemental* variables for the element function f_j are the variables $\mathbf{v}$ and, while we need to associate the particular values $\mathbf{x}_j$ with $\mathbf{v}$, it is the elemental variables which are important in defining the character of the nonlinear element functions.

As an example, the first nonlinear element function for a particular problem might be

$$(x_{29}+x_3-2x_{17})\,e^{x_{29}-x_{17}}; \tag{1.2.1}$$

this is of the structural form

$$f_1(v_1,v_2,v_3)=(v_1+v_2-2v_3)\,e^{v_1-v_3}, \tag{1.2.2}$$

where we need to assign $v_1=x_{29}$, $v_2=x_3$ and $v_3=x_{17}$. For this example, there are three elemental variables.

The example may be used to illustrate a further point. Although f_1 is a function of three variables, the function itself is really only composed of *two* independent parts; the product of $v_1+v_2-2v_3$ with $e^{v_1-v_3}$, or, if we write $u_1=v_1+v_2-2v_3$ and $u_2=v_1-v_3$, the product of u_1 with e^{u_2}. The variables u_1 and u_2 are known as *internal* variables for the element function. They are obtained as *linear combinations* of the elemental variables. The important feature as far as an optimization procedure is concerned is that each nonlinear function involves as few internal variables as possible, as this allows for compact storage and more efficient derivative approximation.

It frequently happens, however, that a function does not have useful internal variables. For instance, another element function might have structural form

$$f_2(v_1,v_2)=v_1\sin v_2, \tag{1.2.3}$$

where for example $v_1=x_6$ and $v_2=x_{12}$. Here, we have broken f_2 down into as few pieces as possible; Although there are internal variables, $u_1=v_1$ and $u_2=v_2$, they are the same in this case as the elemental variables and there is no virtue in exploiting them. Moreover it can happen that although there are special internal variables, there are just as many internal as elemental variables and it therefore doesn't particularly help to exploit them. For instance, if

(1.2.4) $$f_3(v_1,v_2)=(v_1+v_2)\log(v_1-v_2),$$

where, for example, $v_1=x_{12}$ and $v_2=x_2$, the function can be formed as $u_1\log u_2$, where $u_1=v_1+v_2$ and $u_2=v_1-v_2$. But as there are just as many internal variables as elementals, it will not normally be advantageous to use this internal representation.

Finally, although an element function may have useful internal variables, the user may decide not to exploit them. The optimization procedure should still work but at the expense of extra storage and computational effort.

In general, there will be a linear transformation from the elemental variables to the internal ones. For example (1.2.2), we have

(1.2.5) $$\begin{pmatrix}u_1\\u_2\end{pmatrix}=\begin{pmatrix}1&1&-2\\1&0&-1\end{pmatrix}\begin{pmatrix}v_1\\v_2\\v_3\end{pmatrix}$$

while in (1.2.3), we have

(1.2.6) $$\begin{pmatrix}u_1\\u_2\end{pmatrix}=\begin{pmatrix}1&0\\0&1\end{pmatrix}\begin{pmatrix}v_1\\v_2\end{pmatrix}.$$

In general the transformation will be of the form

(1.2.7) $$\mathbf{u}=\mathbf{W}\mathbf{v}$$

and this transformation is *useful* if the matrix $\mathbf{W}$ has fewer rows than columns.

1.3. Element and group types

It is quite common for large nonlinear programming problems to be defined in terms of many nonlinear elements. It is also common that these elements, although using different problem variables, are structurally the same as each other. For instance, the function

(1.3.1) $$\sum_{i=1}^{n-1}(x_ix_{i+1})^i$$

naturally decomposes into the sum of $n-1$ group functions. Each group is a nonlinear element function v_1v_2 of the two elemental variables v_1 and v_2 evaluated for different pairs of problem variables. More commonly, the elements may be arranged into a few classes; the elements within each class are structurally the same. For example, the function

(1.3.2) $$\sum_{i=1}^{n-1}(x_ix_{i+1}+x_1/x_i)^i$$

naturally decomposes into the sum of $n-1$ group functions. Each group is the sum of two nonlinear elements v_1v_2 and v_1/v_2 again evaluated at different pairs of problem variables. A further common occurrence is the presence of elements which have the same structure, but which differ in using different problem variables *and other auxiliary parameters*. For instance, the function

(1.3.3) $$\sum_{i=1}^{n-1}(ix_ix_{i+1})^i$$

naturally decomposes into the sum of $n-1$ group functions. Each group is a nonlinear element $p_1v_1v_2$ of the single parameter p_1 and two elemental variables v_1 and v_2 evaluated for different values of the parameter and pairs of problem variables. Any two elements which are structurally the same are said to be

of the same *type*. Thus examples (1.3.1) and (1.3.3) use a single element type, where as (1.3.2) uses two types. When defining the data for problems of the form (1.1.1)–(1.1.5), it is unnecessary to define each nonlinear element in detail. All that is actually needed is to specify the characteristics of the element types and then to identify each f_j by its type and the indices of its problem variables and (possibly) auxiliary parameters.

The same principal may be applied to group functions. For example, the group functions that make up

$$\sum_{i=1}^{n-1} (x_i x_{i+1})^2 \tag{1.3.4}$$

have different arguments but are structurally all the same, each being of the form $g_i(\alpha)=\alpha^2$. As a slightly more general example, the group functions for

$$\sum_{i=1}^{n-1} i(x_i x_{i+1})^2 \tag{1.3.5}$$

have different arguments and depend upon different values of a parameter but are still structurally all the same, each being of the form $g(\alpha)=p_1\alpha^2$ for some parameter p_1. Any two group functions which are structurally the same are said to be of the same *type;* the structural function is known as the *group type* and its argument is the *group-type variable*. Once again, using group types makes the task of specifying the characteristics of individual group functions more straightforward. The group type $g(\alpha)=\alpha$ is known as the *trivial* type. Trivial groups occur very frequently and are considered to be the default type. It is then only necessary to specify non-trivial group types.

1.4. An example

We now consider the small example problem,

$$\text{minimize } F(x_1,x_2,x_3)\equiv x_1^2+x_2^4x_3^4+x_2\sin(x_1+x_3)+x_1x_3+x_2$$

subject to the bounds $-1\leq x_2\leq 1$ and $1\leq x_3\leq 2$. There are a number of ways of casting this problem in the form (1.1.1). Here, we consider partitioning F into groups as

$$\begin{array}{llll} (x_1)^2 & + (x_2x_3)^4 & + (x_2\sin(x_1+x_3)+x_1x_3+x_2) & \\ \uparrow & \uparrow & \uparrow & \\ \text{group 1} & \text{group 2} & \text{group 3} & \end{array} \tag{1.4.1}$$

Notice the following:

i. group 1 uses the non-trivial group function $g_1(\alpha)=\alpha^2$. The group contains a single *linear* element; the element function is x_1.

ii. group 2 uses the non-trivial group function $g_2(\alpha)=\alpha^4$. The group contains a single *nonlinear* element; this element function is x_2x_3. The element function has *two* elemental variables, v_1 and v_2, say, (with $v_1=x_2$ and $v_2=x_3$) but there is no useful transformation to internal variables.

iii. group 3 uses the trivial group function $g_3(\alpha)=\alpha$. The group contains two *nonlinear* elements and a single *linear* element x_2. The first nonlinear element function is $x_2\sin(x_1+x_3)$. This function has *three* elemental variables, v_1, v_2 and v_3, say, (with $v_1=x_2$, $v_2=x_1$ and $v_3=x_3$), but may be expressed in terms of *two* internal variables u_1 and u_2, say, where $u_1=v_1$ and $u_2=v_2+v_3$. The second nonlinear element function is x_1x_3, which has two v_1 and v_2 (with $v_1=x_1$ and $v_2=x_3$) and is of the same type as the nonlinear element in group 2.

Thus we see that we can consider our objective function to be made up of three groups; the first and second are non-trivial (and of different types) so we will have to provide our optimization procedure with function and derivative values for these at some stage. There are three nonlinear elements, one from group two and two more from group three. Again this means that we shall have to provide function and derivative values for these. The first and third nonlinear element are of the same type, while the second element is a different type. Finally one of these element types, the second, has a useful transformation from elemental to internal variables so this transformation will need to be set up.

1.5. A second example

We now consider a different sort of example, the unconstrained problem,

$$\text{minimize } F(x_1,\ldots,x_{1000}) \equiv (\sum_{i=1}^{999} \sin(x_i^2 + x_{1000}^2 + x_1 - 1)) + \tfrac{1}{2}\sin(x_{1000}^2). \tag{1.5.1}$$

Once again, there are a number of ways of casting this problem in the form (1.1.1), but the most natural is to consider the argument of each sine function as a group – the group function is then $g_i(\alpha) = p_1 \sin\alpha$, $1 \le i \le 1000$, for various values of the parameter p_1. Each group but the last has two nonlinear elements, x_{1000}^2 and x_i^2 $1 \le i \le 999$ and a single linear element $x_1 - 1$. The last has no linear element and a single nonlinear element, x_{1000}^2. A single element type, v_1^2, of the elemental variable, v_1, covers all of the nonlinear elements.

Thus we see that we can consider our objective function to be made up of 1000 nontrivial groups, all of the same type, so we will have to provide our optimization procedure with function and derivative values for these at some stage. There are 1999 nonlinear elements, two from each group except the last, but all of the same type and again this means that we shall have to provide function and derivative values for these. As there is so much structure to this problem, it would be inefficient to pass the data group-by-group and element-by-element. Clearly, one would like to specify such repetitious structures using a convenient shorthand.

1.6. A final example

As a third example, consider the constrained problem in the variables $x_1, \ldots, x_{100}$ and y

$$\text{minimize } \tfrac{1}{2}((x_1 - x_{100})x_2 + y)^2 \tag{1.6.1}$$

subject to the constraints

$$x_1 x_{i+1} + (1 + 2/i)x_i x_{100} + y \le 0 \quad (1 \le i \le 99), \tag{1.6.2}$$

$$0 \le (\sin x_i)^2 \le \tfrac{1}{2} \quad (1 \le i \le 100), \tag{1.6.3}$$

$$(x_1 + x_{100})^2 = 1 \tag{1.6.4}$$

and the simple bounds

$$-1 \le x_i \le i \quad (1 \le i \le 100). \tag{1.6.5}$$

As before, there are a number of ways of casting this problem in the form (1.1.2)–(1.1.5). We chose to decompose the problem as follows:

i. the objective function group uses the non-trivial group function $g(\alpha)=\frac{1}{2}\alpha^2$. The group contains a single *linear* element; the element function is y. There is also a nonlinear element $(x_1-x_{100})x_2$. This element function has three elemental variables, v_1, v_2 and v_3, say (with $v_1=x_1$, $v_2=x_{100}$ and $v_3=x_2$); there is a useful transformation from elemental to internal variables of the form $u_1=v_1-v_2$ and $u_2=v_3$ and the element function may then be represented as u_1u_2.

ii. The next set of groups, inequality constraints, $x_1x_{i+1}+(1+2/i)x_ix_{100}+y\leq 0$ for $1\leq i\leq 99$ are of the form (1.1.5) with no lower bounds. Each uses the trivial group function $g(\alpha)=\alpha$ and contains a single *linear* element, y, and two *nonlinear* elements x_ix_{i+1} and $(1+2/i)x_ix_{100}$. Both nonlinear elements are of the same type, $p_1v_1v_2$, for appropriate variables v_1 and v_2 and parameter p_1 and there is no useful transformation to internal variables.

iii. The following set of groups, again inequality constraints, $0\leq(\sin x_i)^2\leq\frac{1}{4}$ for $1\leq i\leq 100$ are of the form (1.1.5) with both lower and upper bounds. Each uses the non-trivial group function $g(\alpha)=\alpha^2$ and contains a single *nonlinear* element of the type $\sin v_1$ for an appropriate variable v_1. Notice that the group types for these groups and for the objective function group are both of the form $g(\alpha)=p_1\alpha^2$, for some parameter p_1, and it may prove more convenient to use this form to cover both sets of groups.

iv. The last group, an equality constraint, $(x_1+x_{100})^2-1=0$ is of the form (1.1.4). Again, this group uses the trivial group function $g(\alpha)=\alpha$ and contains a single *linear* element, -1, and a single *nonlinear* element of the type $(v_1+v_2)^2$ for appropriate elemental variables v_1 and v_2. Once more, a single internal variable, $u_1=v_1+v_2$ can be used and the element is then represented as u_1^2.

Thus we see that we can consider our problem to be made up of 201 groups of two different types so we will have to provide our optimization procedure with function and derivative values for these at some stage. There are 200 nonlinear elements of four different types and again this means that we shall have to provide function and derivative values for these. As for the previous example, there is so much structure to this problem and it would be inefficient to pass the data group-by-group and element-by-element. Again, one needs to specify this repetitious structure using a convenient shorthand.

We now consider how to pass the data for these and other problems to an optimization procedure. In our description, we will concentrate on our third example; we will show how the input file might be specified for this example to motivate the overall structure of such a file and then follow this with the general syntax allowed.

2. The standard data input format

The data which defines a particular problem is written in a file in a standard format. It is intended that this data file is interpreted by an appropriate decoding program and converted into a format useful for input to an optimization package or program. The content of the file is specified line by line. As our format is intended to be compatible with the MPS linear programming format (MPS/360 Version 2, Linear and separable programming – users manual H20-0476-2, 1969), we preserve the MPS terminology and call these lines *cards*.

2.1 Introduction to the standard data input format

As we have just said, the data format is designed to be compatible with the MPS linear programming format. There are, however, extensions to allow the user to input nonlinear problems. The user must prepare an input file consisting of three types of cards:

1 Indicator cards, which specify the type of data to follow.

2 Data cards, which contain the actual data.

3 Comment cards.

Indicator cards contain a simple keyword to specify the type of data that follows. The first character of such cards must be in column 1; indicator cards are the only cards, with the exception of comment cards, which start in column 1. Possible indicator cards are given in Figure 2.1.

keyword	comments	presence	described in §
NAME		mandatory	2.2.1
	either		
GROUPS		mandatory	2.2.6
ROWS	synonym for GROUPS	mandatory	2.2.6
CONSTRAINTS	synonym for GROUPS		2.2.6
VARIABLES		mandatory	2.2.7
COLUMNS	synonym for VARIABLES		2.2.7
	or		
VARIABLES		mandatory	2.2.8
COLUMNS	synonym for VARIABLES		2.2.8
GROUPS		mandatory	2.2.9
ROWS	synonym for GROUPS		2.2.9
CONSTRAINTS	synonym for GROUPS		2.2.9
CONSTANTS		optional	2.2.10
RHS	synonym for CONSTANTS		2.2.10
RHS'	synonym for CONSTANTS		2.2.10
RANGES		optional	2.2.11
BOUNDS		optional	2.2.12
START POINT		optional	2.2.13
ELEMENT TYPE		optional	2.2.14
ELEMENT USES		optional	2.2.15
GROUP TYPE		optional	2.2.16
GROUP USES		optional	2.2.17
OBJECT BOUND		optional	2.2.18
ENDATA		mandatory	2.2.2

Figure 2.1. Possible indicator cards

Indicator cards must appear in the order shown, except that the GROUPS and VARIABLES sections may be interchanged to allow specification of the linear terms by rows or columns. The cards CONSTANTS, RHS', RHS, RANGES, BOUNDS, START POINT, ELEMENT TYPE, ELEMENT USES, GROUP TYPE, GROUP USES and OBJECT BOUND are optional.

The data cards are divided into six fields. The content of each field varies with each type of data card as described in §2.2. Those in fields 1, 2, 3 and 5 must always be left justified within the field. Field 1, which appears in columns 2 and 3, may contain a *code* (that is, a one of two character string which defines the expected contents of the remaining fields on the card), fields 2, 3 and 5 may hold *names* and fields 4 and 6 might store *numerical values*. The numerical values are defined by up to 12 characters which may include a decimal point and an optional sign (a positive number is assumed unless a – is given). The value may be followed by a decimal exponent, written as an `E` or `D`, followed by a signed or unsigned one or two digit integer; the first blank after the `E` or `D` terminates the field. The names of variables, nonlinear elements or groups may be up to ten characters long. These names may include integer indices, see §2.1.1.

Any card with the character `*` in column 1 is a comment card; the remaining contents of the card is ignored. Such a card may appear anywhere in the data file. In addition, completely blank cards are ignored when scanning the input file and may thus be used to space the data. Finally, the presence of a `$` as the first character in fields 3 or 5 of a data card indicates that the content of the remaining part of the card is a comment and will be ignored.

2.1.1 The names of variables, nonlinear elements and groups

One of the positive features of the MPS standard is the ability to give meaningful names to problem constraints and variables. As our proposal is intended to be MPS compatible, we too have this option. However, one of the less convenient features of the MPS standard is the cumbersome way that repetitious structure is handled. In particular, the name of each variable and constraint must be defined on a separate line, and structure within constraints is effectively ignored when setting up the constraint matrix. We consider it important to overcome this deficiency of the MPS standard when formulating large-scale examples. One way is to allow variable, group and nonlinear element names to have indices and to have syntactic devices which enable the user to define many items at once.

We allow any variable, group or nonlinear element to have a name using up to ten valid characters. A *valid* character is any ASCII character whose decimal code lies in the range 32 to 126 (binary 0100000 to 11111110, hex 20 to 7E) (see, for instance, the Dictionary of Computing, Oxford University Press, 1983). This include lower and upper case roman alphabetic characters, the digits 0 to 9, the blank character and other mathematical and grammatical symbols. A name can be one of the following:

(i) a scalar name of the form `£££££££££` where each `£` is a valid character type excepting that the first `£` cannot be a `$`.

(ii) an array name of the form `name(index)`, where `index` is a list of integer index names, `name` is a list of valid characters (the first character may not be a `$`) and the maximum possible size of the *expanded name* does not exceed ten characters. The list of index names must be of the form `list1,list2,list3`, where `list1`, `list2` and `list3` are predefined index (parameter) names (see §2.2.3, below) and all three indices are optional. The indices are only allowed to take on integer values. Commas are only required as separators; the presence of an open bracket "`(`" announces a list of indices and a close bracket "`)`" terminates the list. A vector *name* is *expanded* as `namenumber1,number2,number3`, where `numberi, i = 1, 2, 3` is the integer value allocated to the index `listi` at the time of use.

As an example, the expanded form of the array name `X(I,J,K)` when `I`, `J` and `K` have the values 3, 4 and 6 respectively would be `X3,4,6`, while it would take the form `X-6,0,3` if `I`, `J` and `K` have the values –6, 0 and 3 respectively. However, `X(I,J,K)` could not be expanded if `I`, `J` and `K` were each allowed to be as large as 100 as, for instance, `X100,100,100` is over ten characters long and thus not a valid expanded name.

An array item may be referred to by either its array name (so long as the index lists have been specified) or by its expanded name. Thus, if `I, J` and `K` have been specified as 2, 7 and 9 respectively, `X(I,J,K)` and `X2,7,9` are identical.

If two separators (opening or closing brackets and commas) are adjacent in an array name, the intervening index is deemed not to exist and is ignored when the name is expanded. Thus, the expanded name of `Y()` is just `Y`, while that of `Z(I,,K)` is `Z3,4` if `I` is 3 and `K` is 4. Furthermore any name which does not include the characters `( , )` or `,` may be used as an array name and is its own expanded name. Thus the name `X` may be a scalar or array name whereas `W(` and `V,` can only be scalar names.

We defer the definition of integer indices until §2.2.3.

Note that blanks are considered to be significant characters. Thus if _ denotes a blank, the names `_X` and `X_` are different. It is recommended that all names are left-shifted within their relevant data fields to avoid possible user-instigated name recognition errors.

2.1.2 Numerical values

The definition of a specific problem normally requires the use of numerical (real) data values. Such values can be specified in two ways. Firstly, the values may simply occur as floating point numbers in data fields 4 and 6. Secondly, values may be allocated to named parameters, known as real parameters, and a value subsequently used by reference to particular real parameter name. This second method may only be used to allocate values on certain cards; when this facility is used, the first character in field 1 on the relevant data card will be a `V` or `Z`. This latter approach is particularly useful when a value is to be used repeatedly or if a value is to be changed within a do-loop (see §2.2.4).

We defer the definition of real parameters until §2.2.3.

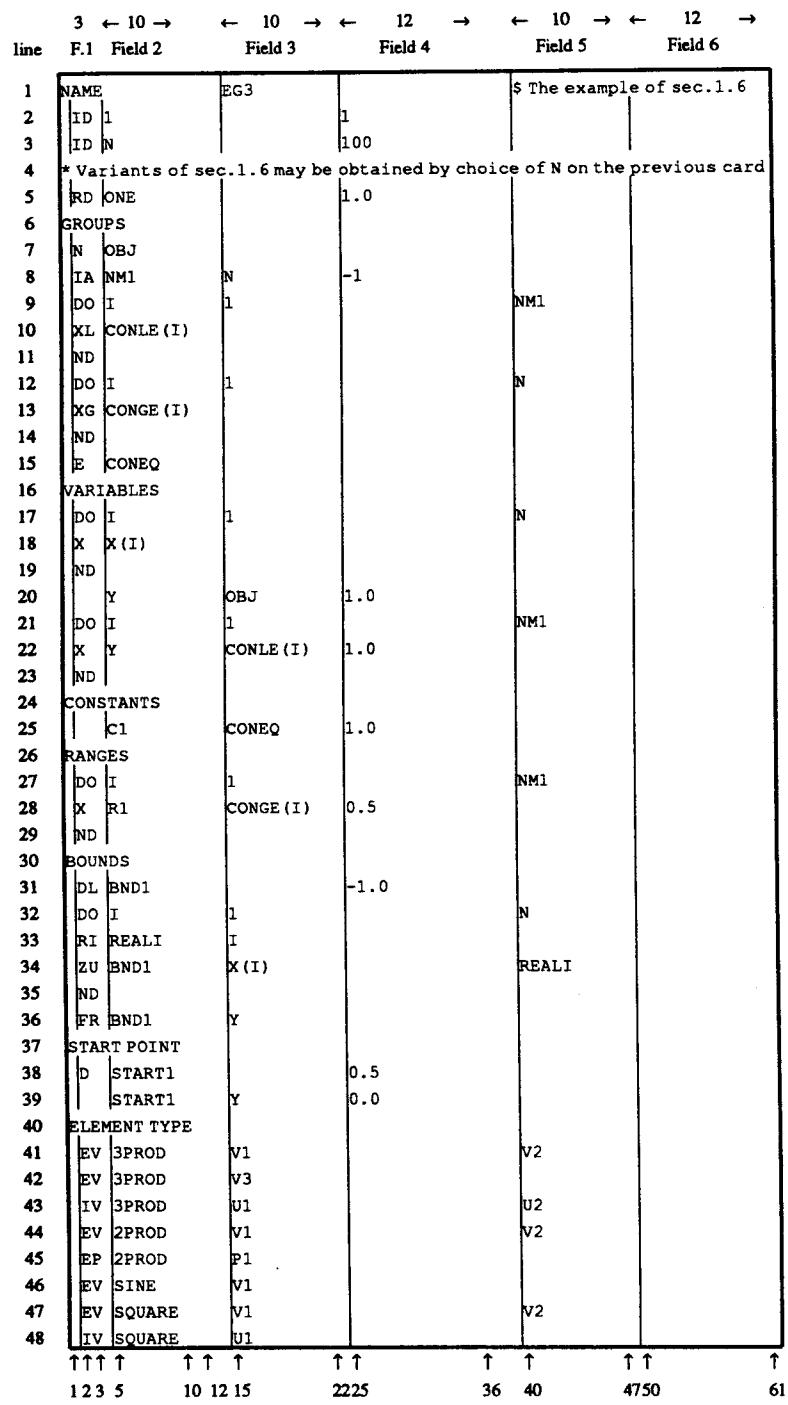
2.1.3 An example.

Before we give the complete syntax for an SDIF file, we give an illustrative example. In order to exhibit as many constructs as possible, we consider how we might encode the example in §1.6. We urge the reader to study this section in detail. As always, there are many possible ways of specifying a particular problem; we give one in Figure 2.1.3. The horizontal and vertical lines are merely included to indicate the extent of data fields. The actual widths of the fields are given at the top of the figure, and the column numbers given at its foot.

The SDIF file naturally divides into two parts. In the first part, lines 2 to 39 of the figure, we specify information regarding linear functions used in the example. In the second part, lines 40 to 93, we specify nonlinear information. The first part is merely an extension of the MPS input format; the second part is new.

The file must always start with a `NAME` card, on which a name (in this case `EG3`) for the example may be given (line 1), and must end with an `ENDATA` card (line 94). A comment is inserted at the end of line 1 as to the source of the example. The character $ identifies the remainder of the line as a comment; the comment is ignored when interpreting the input file.

We next specify names of parameters which will occur frequently in specifying the example (lines 2 to 5). In our case the integer and real parameters `1` and `ONE` are given along with `N`, a problem dimension – here `N` is set to 100, but it would be trivial to change the example in 1.6 to allow variables $x_1, \ldots, x_n$ for any n. We make a comment to this effect on line 4; any card with the character * in column 1 is a comment card and its content ignored when interpreting the input file.



line	F.1	Field 2	Field 3	Field 4	Field 5	Field 6
1	NAME		EG3		$ The example of sec.1.6	
2	ID	1		1		
3	ID	N		100		
4	* Variants of sec.1.6 may be obtained by choice of N on the previous card					
5	RD	ONE		1.0		
6	GROUPS					
7	N	OBJ				
8	IA	NM1	N	-1		
9	DO	I	1		NM1	
10	XL	CONLE(I)				
11	ND					
12	DO	I	1		N	
13	XG	CONGE(I)				
14	ND					
15	E	CONEQ				
16	VARIABLES					
17	DO	I	1		N	
18	X	X(I)				
19	ND					
20		Y	OBJ	1.0		
21	DO	I	1		NM1	
22	X	Y	CONLE(I)	1.0		
23	ND					
24	CONSTANTS					
25		C1	CONEQ	1.0		
26	RANGES					
27	DO	I	1		NM1	
28	X	R1	CONGE(I)	0.5		
29	ND					
30	BOUNDS					
31	DL	BND1		-1.0		
32	DO	I	1		N	
33	RI	REALI	I			
34	ZU	BND1	X(I)		REALI	
35	ND					
36	FR	BND1	Y			
37	START POINT					
38	D	START1		0.5		
39		START1	Y	0.0		
40	ELEMENT TYPE					
41	EV	3PROD	V1		V2	
42	EV	3PROD	V3			
43	IV	3PROD	U1		U2	
44	EV	2PROD	V1		V2	
45	EP	2PROD	P1			
46	EV	SINE	V1			
47	EV	SQUARE	V1		V2	
48	IV	SQUARE	U1			

1 2 3 5 10 12 15 22 25 36 40 47 50 61

Figure 2.1.3 (part 1). SDIF file for the example of §1.6

We now name the problem variables and groups (in our example objective function and constraints) used. The groups may be specified before or after the variables. We chose here to name the groups first.

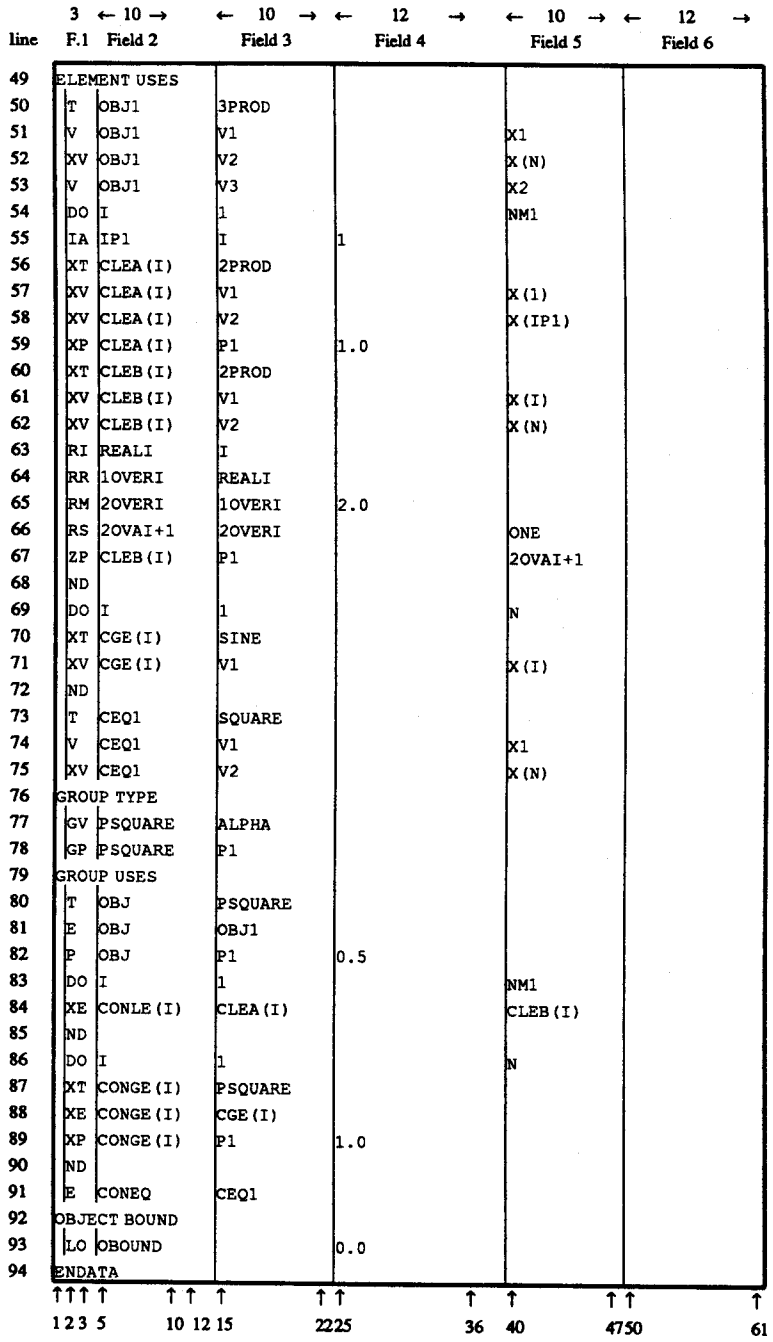


Figure 2.1.3 (part 2). SDIF file for the example of §1.6

The objective function will be known as OBJ (line 7); the character N in field 1 specifies that this is an objective function group. The inequality constraints (1.6.2) and (1.6.3) are named CONLE1, ..., CONLE99 and CONGE1, ..., CONGE100 respectively. Rather than specify them individually, a do loop is used to make an array definition. Thus the constraints CONLE1, ..., CONLE99 are defined *en masse* on lines 9 to 11 with the do loop index I running from the previously defined value 1 to the value NM1. The

integer parameter is defined on line 8 to be the sum of N and the value – 1 and in our case will be 99. The characters XL in field 1 of line 10 indicate that an array definition is being made (the X) and that the groups are less-than-or-equal-to constraints (the L) . The do loop introduced on line 9 with the characters DO in field 1 is terminated on line 11 with the characters ND in its first field. In a similar way, the constraints CONGE1, ..., CONGE99 are defined all together on lines 12 to 14; that these constraints involve bounds on both sides is taken care of by considering them to be greater-than-or-equal-to constraints (XG) on line 13 and later specifying the additional upper bounds in the RANGE section (lines 26 to 29). Finally, the equality constraint (1.6.4) is to be called CONEQ (line 7); the character E in field 1 specifies that this is an equality constraint group.

Having named the groups, we next name the problem variables. At the same time, we include the coefficients of all the linear elements used. The variables are named X1, ..., X100 and Y; an array declaration is made for the former set on lines 17 to 19 and Y is defined on line 20. The character X in field 1 of line 18 indicates that an array definition is used. Only the objective function (1.6.1), inequality constraint (1.6.2) and equality constraint groups (1.6.4) contain linear elements. As well as introducing Y, line 20 also specifies that the linear element associated with group OBJ (field 3) involves variable Y and that Y s coefficient in the linear element is 1.0 (field 4). A do loop is now used in lines 21 to 23 to show that the linear elements for constraints (1.6.2) also use the variable Y. It is assumed that unless a variable is explicitly identified with a linear element, the element is independent of that variable. Thus, although (1.6.4) uses a linear element, the element is constant and need not be specified in the VARIABLES section

The only remaining part of the linear elements which must be specified is the constant term. Again, only nonzero constants need be given. For our example, only the equality constraint group (1.6.4) has a nonzero constant term and this data is specified on lines 24 and 25. The string C1 in field 2 of line 25 is the name given to a specific set of constants. In general more than one set of constants may be specified in the SDIF file and the relevant one selected in a postprocessing stage. Here, of course, we only have one set.

As we have seen, the inequality constraint groups (1.6.3) are bounded from above as well as from below. In the RANGES section (lines 26 to 30) we specify these upper bounds (or range constraints as they are sometimes known). The numerical values ½ are specified for each bound for the relevant groups in an array definition on line 28; the string R1 in field 2 is once again a name given to a specific set of range values as it is possible to define more than one set in the RANGES section.

We now turn to the simple bounds (1.6.5) which are specified in lines 30 to 36 of the example. All problem variables are assumed to have lower bounds of zero and no upper bounds unless otherwise specified. All but one of the variables for our example have lower bounds of – 1. We thus change the default value for the value of the lower bound on line 31 – the set of bounds is named BND1. The character D in field 1 indicates that the default is being changed and the character L specifies that it is the lower bound default that is to be changed. The variable x_i is given an upper bound of i. We encode that in a do loop on lines 32 to 35 of the figure. The do loop index I is an integer. We change its current value to a real on line 33 and assign that value as the upper bound on line 34. The character Z in field 1 of this line indicates that an array definition is being made and that the data is taken from a parameter in field 5 (as opposed to a specified numerical value in field 4) and the character U specifies that the upper bound value is to be assigned. The variable y is unbounded or, as it is often known, free. This is specified on line 36, the string FR in field 1 indicating that Y is free.

The final "linear" piece of information given is an estimate of the solution to the problem (if known) or at least a set of values from which to start a minimization algorithm. This information is given on lines 37

to 39. For our problem, we choose the values $x_i = \frac{1}{2}$, $1 \le i \le 100$ and $y = 0$. Unless otherwise specified, all starting values take a default of zero. We change that default on line 38 to $\frac{1}{2}$ – the set of starting values are named START1 – and then specify the individual value for the variable Y on line 39.

We now specify the nonlinear information. We saw in §1.6 that there are four element types for the problem, being of the form (i) $(v_1 - v_2)v_3$, (ii) $p_1 v_1 v_2$, (iii) $\sin v_1$ and (iv) $(v_1 + v_2)^2$. In the ELEMENT TYPE section on lines 40 to 48, we record details of these types. We name the four types (i)–(iv) 3PROD, 2PROD, SINE and SQUARE respectively. For 3PROD, we define the elemental variables (lines 41 and 42) to be V1, V2 and V3 and the internal variables (line 43) to be U1 and U2. Elemental variables may be defined, two to a line, on lines for which field 1 is EV. Internal variables, on the other hand, are defined on lines with IV in field 1. Similar definitions are made for 2PROD (line 44), SINE (line 46) and SQUARE (line 47). The type 2PROD also makes use of a parameter p_1. This is named P1 on line 45 for which field 1 reads EP.

Having specified the element types, we next specify individual nonlinear elements in the ELEMENT USES section. As we have seen, the objective function group uses a single nonlinear element of type 3PROD. We name this particular element OBJ1. On line 50, the character T in field 1 indicates that the OBJ1 is of type 3PROD. The assignment of problem to elemental variables is made on lines 51 to 53. Problem variables X1 and X3 are assigned to elemental variables V1 and V3; the assignment is indicated by the character V in field 1. In order to assign x_{100} (or in general x_n) to v_2, we assign the array entry X(N) to V2. Notice that as an array element is being used, this must be specially flagged (XV in field 1) as otherwise the wrong variable (called X(N) rather than X100 which is the expanded form of X(N)) would be assigned. There are two nonlinear elements for each inequality constraint group (1.6.2), each being of the same type 2PROD. We name these elements CLEA1, ..., CLEA99 and CLEB1, ..., CLEB99. The assignments are made on lines 54 to 68 within a do loop. On lines 56 and 60 the elements are named and their types assigned. As array assignments are being used, field 1 for both lines contains the string XT. The elemental variables are then associated with problem variables on lines 57-58 and 61-62 respectively. Again array assignments are used and field 1 contains the string XV. Notice that on line 58 v_2 is assigned the problem variable x_{i+1}, where the index IP1 is defined as the sum of the index I and the integer value 1 on line 55. It remains to assign values for the parameter p_1 for each element. This is straightforward for the elements CLEA1, ..., CLEA99 as the required value is always 1 and the assignment is made on line 59 on a card with first field XP. The remaining elements have varying parameter values $1 + 2/i$. This value is calculated on lines 63 to 66 and assigned on line 67. Line 63 assigns REALI to have the floating point value of the index I and the reciprocal of this value, 1OVERI, is obtained on the next line. This new value is then multiplied by the value 2 on line 65 and the value assigned to ONE is added to the resulting value on the final line. Thus the parameter 2OVAI+1 holds the required value of p_1 and the array assignment is made on line 67. On this line the string ZP indicates that an array assignment is being made taking its value from the parameter 2OVAI+1 in field 4 (the Z) and that the elemental parameter P1 in field 3 is to be assigned (the P). The definition of the nonlinear elements for the remaining constraint groups is straightforward. The inequality constraints (1.6.3) each use a single element, named CGE1, ..., CGE100, of type SINE and the appropriate array assignments are made on lines 69 to 72. Finally, the equality constraint (1.6.4) is named CEQ1 and typed SQUARE with appropriate elemental variable assignments on lines 73 to 75.

We next need to specify the nontrivial group types. This is done in the GROUP TYPE section on lines 76 to 78. We saw in §1.6 that a single nontrivial group, $p_1 \alpha^2$, is required. On line 77, the name PSQUARE is given for the type and the group type variable α is named ALPHA. The string GV in field 1 indicates that a type and its variable are to be defined. On the following line field 1 is GP and this is used to announce that the group type parameter p_1 is named P1.

Finally, we need to allocate nonlinear elements to groups and specify what type the resulting groups are to be. This takes place within the GROUP USES section which runs from line 79 to 91. The objective function group is nontrivial and its type is announced on line 80. The group uses the single nonlinear element OBJ1 specified on line 81 and the group-type parameter p_1 is set to the value $\frac{1}{2}$ on the next line. The characters T, E and P in the first fields of these three cards announcing their purposes. The inequality groups (1.6.2) each use two nonlinear elements, but the groups themselves are trivial (and thus their types do not have to be made explicit). The assignment of the elements to each group is made in an array definition on lines 83 to 85; line 84 is flagged as assigning elements to a group with the string XE in field 1. The second set of inequality constraints (1.6.3) use the nontrivial group type PSQUARE with parameter value 1. Each group uses a single nonlinear element and the appropriate array assignments are contained on lines 86 to 90. Lastly the trivial equality constraint group (1.6.4) is assigned the nonlinear element CEQ1 on line 91.

The definition of the problem is now complete. However, it often helps the intended minimization program if known lower and upper bounds on the possible values of the objective function can be given. For our example, the objective function (1.6.1) cannot be smaller than zero. This data is specified on lines 92 and 93. The string LO in field 1 of line 93 indicates that a lower bound is known for the value of (1.6.1). The string OBOUND in field 2 of this line is a name given to this known bound. The value of the lower bound now follows in field 4. No upper bound need be specified as the function is assumed to lie between plus and minus infinity.

2.2 Indicator and data cards

We now give details of the indicator cards and the data cards which follow them.

2.2.1 The NAME indicator card

The NAME indicator card is used to announce the start of the input data for a particular problem. The user may specify a name for the problem; this name is entered on the indicator card in field 3 and may be at most 10 characters long. The syntax for the NAME card is given in Figure 2.2.1.

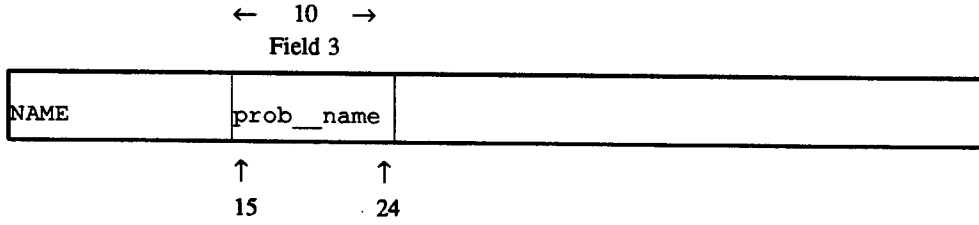


Figure 2.2.1. The indicator card NAME

2.2.2 The ENDATA indicator card

The ENDATA indicator card simply announces the end of of the input data. The data for a particular problem, in the form of indicator and data cards, must lie between a NAME and an ENDATA card. The syntax for the ENDATA card is given in Figure 2.2.2.

```
ENDATA
```

Figure 2.2.2. The indicator card ENDATA

2.2.3 Integer and real parameters

We shall use the word *parameter* to mean the name given to any quantity which is associated with a specified numerical value. The numerical value will be known as the *parameter value*. Integer and real values may be associated with parameters in two ways. The easiest way is simply to set a parameter to a specified parameter value, or to obtain a parameter from a previously defined parameter by simple arithmetic operations (addition, subtraction and multiplication for integer values, addition, subtraction, multiplication and reciprocation for real values). The second way is to have a parameter value specified in a do loop, or to obtain a parameter from one specified in a do loop (see §2.2.4 below).

The syntax for associating a parameter with a specific value is given in Figure 2.2.3.

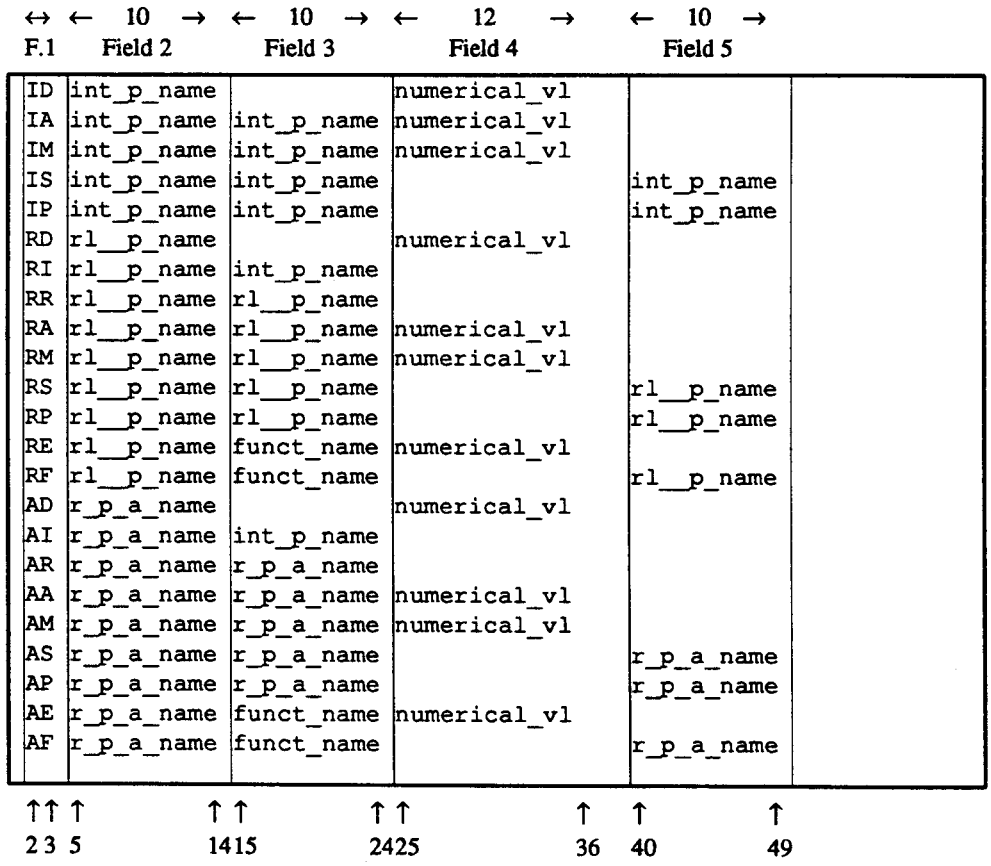


Figure 2.2.3. Possible cards for specifying parameter values

The two character string in data field 1 (F.1) specifies the way in which the parameter value is to be assigned. If the first of these characters is an `I`, the assigned value is an integer; the parameter will be referred to as an *integer parameter* or *integer index*. Alternatively, if the first of these characters is an `R` or an `A`, the assigned value is a real and the parameter will be called a *real parameter*.

If the string is ID, the integer parameter int_p_name named in field 2 is to be given the integer value specified in field 4. The parameter may be up to ten characters long, and the integer value can occupy up to twelve positions.

If the string is IA, the integer parameter named in field 2 is to be formed by adding the value of the parameter int_p_name referred to in field 3 to the integer value specified in field 4. The parameter appearing in field 3 must have already been assigned a value.

If the string is IM, the value of the integer parameter named in field 2 is to be obtained by multiplying the value already specified for the parameter in field 3 by the integer value specified in field 4. Once again, the parameter appearing in field 3 must have already been assigned a value.

If the string is IS, the value of the integer parameter named in field 2 is to be calculated by summing the values of the integer parameters int_p_name referred to in fields 3 and 5. The parameters appearing in fields 3 and 5 must have already been assigned values.

Finally, if the string is IP, the value of the integer parameter named in field 2 is to be formed as the product of the values already specified for the integer parameters in fields 3 and 5. Once again, the parameters appearing in fields 3 and 5 must have already been assigned values.

Note that, as an array name can only be a maximum of 10 characters long, any integer parameter which is to be the index of an array may only be at most seven characters in length. Furthermore, such a parameter name may not include the characters " (", ") " or " , ".

If the string is RD the real parameter rl__p_name named in field 2 is to be given the real value specified in field 4. The parameter may be up to seven characters long, and the real value can occupy up to twelve positions.

If the string is RI, the real parameter value named in field 2 is to be assigned the equivalent floating point value of the integer parameter int_p_name specified in field 3. The parameter appearing in field 3 must have already been assigned a value.

If the string is RR, the real parameter value named in field 2 is to be assigned the reciprocal of the value of the real parameter rl__p_name specified in field 3. Again, the parameter appearing in field 3 must have already been assigned a value.

If the string is RA, the value of the real parameter named in field 2 is to be formed by adding the value of the real parameter rl__p_name referred to in field 3 to the real value specified in field 4. The parameter appearing in field 3 must have already been assigned a value.

If the string is RM, the value of the parameter named in field 2 is to be formed by multiplying the value already specified for the real parameter in field 3 by the real value specified in field 4. Once again, the parameter appearing in field 3 must have already been assigned a value.

If the string is RS, the parameter value named in field 2 is to be formed as the sum of the values of the real parameters rl__p_name referred to in fields 3 and 5. The parameters appearing in fields 3 and 5 must have already been assigned values.

If the string is RP, the value of the real parameter named in field 2 is to be formed as the product of the values already specified for the real parameters in fields 3 and 5. Once again, the parameters appearing in fields 3 and 5 must have already been assigned values.

If the string is RE, the value of the parameter named in field 2 is to be formed by evaluating the function named in field 3 at the real value specified in field 4. The function funct_name – and its mathematical equivalent $f(x)$ – may be one of: ABS ($f(x)=|x|$), SQRT ($f(x)=\sqrt{x}$), EXP ($f(x)=e^x$), LOG

($f(x)=\log_e x$), LOG10 ($f(x)=\log_{10} x$), SIN ($f(x)=\sin x$), COS ($f(x)=\cos x$), TAN ($f(x)=\tan x$), ARCSIN ($f(x)=\sin^{-1} x$), ARCCOS ($f(x)=\cos^{-1} x$), ARCTAN ($f(x)=\tan^{-1} x$), HYPSIN ($f(x)=\sinh x$), HYPCOS ($f(x)=\cosh x$) or HYPTAN ($f(x)=\tanh x$). Certain of the functions may only be evaluated for arguments lying within restricted ranges. The argument for SQRT must be non-negative, those for LOG and LOG10 must be strictly positive, and those for ARCSIN and ARCCOS must be no larger than one in absolute value.

Finally, if the string is RF, the value of the parameter named in field 2 is to be formed by evaluating the function named in field 3 at the value of the real parameter rl__p_name specified in field 5. The function (and its mathematical equivalent) may be any of those named in the previous paragraph and the restrictions on the allowed argument ranges given above still apply.

If the first character in field 1 is an A, an array of real parameters are to be defined. The particular type of definition is as for the R cards, excepting that any name, r_p_a_name, referred to in fields 2, 3 or 5, with the exception of integer parameters named in field 3 of AI cards and functions named in the same field of AE and AF cards, must be an real parameter array name with a valid index.

Parameter assignments may be made at any point within the SDIF file between the START and ENDATA indicator cards. It is anticipated that parameters will be used to store values such as the total number of variables and groups, which are used later in array definitions, and to allow a user to enter regular and repetitious data in a straightforward and compact way.

2.2.4 Do loops

A do loop may occur at any point in the GROUPS, VARIABLES, CONSTANTS, RANGES, BOUNDS, START POINT, ELEMENT USES or GROUP USES sections. Do loops are used to make array definitions, that is, to make compact definitions of several quantities at once. The syntax required for do loops is given in Figure 2.2.4.

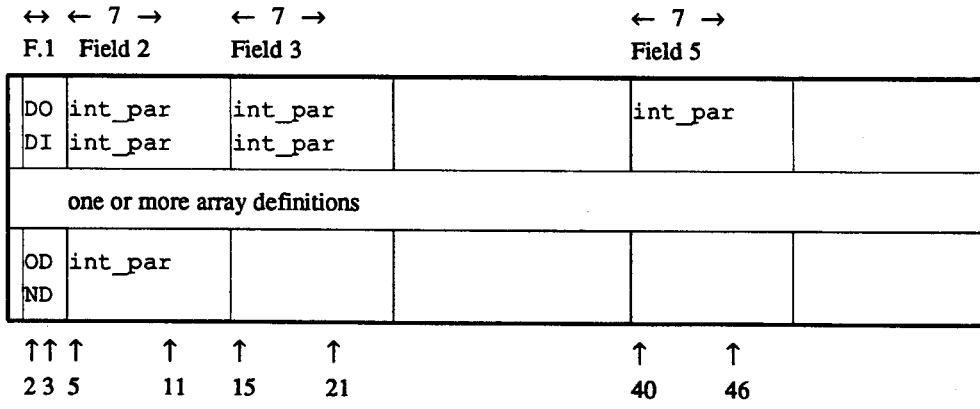


Figure 2.2.4. Syntax for do loops

The two character string in data field 1 specify either the start or the end of a do loop. The start of a loop is indicated by the string DO. In this case an integer parameter named in field 2 is defined to take values starting from the integer parameter value given in field 3 and ending with the last value before the integer parameter value given in field 5 has been surpassed. The parameters named in fields 3 and 5 must have been defined on previous data cards. The parameter name defined in field 2 can occupy up to seven

locations. If the next data card does not have the characters DI as its first field, the parameter defined on the DO card, iloop say, will take all integer values starting from that given in field 3, say istart, and ending on that in field 5, iend say. If istart is larger than iend, the loop will be skipped.

If the data card following a DO card has the string DI in field 1, the do loop parameter named in field 2 is to be incremented by the amount, incr say, specified for the integer parameter given in field 3. Once again, the parameter in field 3 must have been previously defined. The index iloop will now take values

$$\texttt{iloop} = \texttt{istart} + j \cdot \texttt{incr}$$

for all positive j for which iloop lies between (and including) istart and iend. If incr is negative and istart is larger than iend, the parameter specifies a decreasing sequence of values. If incr is positive and istart is larger than iend, or if incr is negative and istart is smaller than iend, the loop will be skipped.

Once a do loop has been started, any array definitions which use its do-loop index specify that the definition is to be made for all values of the integer parameter specified in the loop. Loops can be nested up to three deep; this corresponds to the maximum number of allowed indices in an array index list.

A do loop must be terminated. A particular loop can be terminated on a data card in which field 1 contains the characters OD; the name of the loop parameter must appear in field 2. Alternatively, all loops may be terminated at once using a data card in which field 1 contains the characters ND.

In addition, parameter assignments with the syntax given in Figure 2.2.3 – that is cards whose first field are ID, IA, IM, IS, IP, RD, RI, RR, RA, RM, RS or RP – may be inserted at any point in a do loop; it is only necessary that a parameter is defined prior to its use.

Note that array definitions may occur both within and outside do-loops; all that is required for a successful array definition is that the integer indices used have defined values when they are needed. The use of do loops is illustrated in §2.4.

2.2.5 The definition of variables and groups

In the MPS standard, the constraint matrix, the matrix of linear elements, is input by columns; firstly the names of the constraints are specified in the ROWS section and then variable names and the corresponding matrix coefficients are set one at a time in the COLUMNS section. While there is some justification for this form of matrix entry for linear programming problems – the principal solution algorithm for such problems, the simplex method (Dantzig, 1963), is usually column oriented – there seems no good reason why the coefficients of linear elements might not also be input by rows. After all, it is more natural to think of specifying the constraints for a problem one at a time. Furthermore, requiring that a complete row or column has been specified before the next may be processed is unnecessarily restrictive.

We thus allow the data to be input in a either a group-wise (row-wise) or variable-wise (column-wise) fashion. In a group/row-wise scheme, one or two coefficients and their variable/column names are specified for a given group/row; for a variable/column-wise scheme, one or two coefficients and their group/row names are specified for a given variable/column. We do however still require that in a group/row-wise storage scheme, the names of all the variables/rows *which appear in linear elements* are completely specified before the coefficients are input. Similarly, in a variable/column-wise storage scheme, the names of all the groups/rows *which have a linear element* must be completely specified before the coefficients are input. This allows for some checking of the input data.

If the groups/rows are specified first, there is no requirement that variables/columns are input one at a time (but of course they may be). When processing the data file, variable/column names should be inspected to see if they are new or where they have appeared before. Likewise, if the variables/columns are specified first, there is no requirement that groups/rows are ordered on input. The coordinates of new data values can then be stored as a linked triple (group/row, variable/column, value). Conversion from such a component-wise input scheme to a row or column based storage scheme may be performed very efficiently if desired (see Duff, Erisman and Reid, 1986, pp. 30-31 and subroutine MC39 in the Harwell Subroutine Library).

If a variable/column-wise input scheme is to be adopted, the data file will contain a `GROUPS/ROWS/CONSTRAINTS` indicator card and section followed by a `VARIABLES/COLUMNS` card and section. The allowed data cards are discussed in §2.2.6 and §2.2.7. If a group/row-wise input scheme is to be adopted, the data file will contain a `VARIABLES/COLUMNS` indicator card and section followed by a `GROUPS/ROWS/CONSTRAINTS` card and section. The data cards for this scheme are discussed in §2.2.8 and §2.2.9.

2.2.6 The `GROUPS, ROWS` or `CONSTRAINTS` data cards (variable/column-wise)

The `GROUPS, ROWS` and `CONSTRAINTS` indicator cards are used interchangeably to announce the names of the groups which make up the objective function or, for constrained problems, the names of the constraints (or rows as they are often known in linear programming applications). The user may give a scaling factor for the groups or constraints In addition, groups which are linear combinations of previous groups may be specified. The syntax for the data cards which follow these indicator cards is given in Figure 2.2.6.

The one or two character string in data field 1 specifies the type of group, row or constraint to be input. Possible values for the first character are:

`N` the group is to be specially marked (for constrained problems, the group/row is an objective function group/row).

`G` the group is to use an extra "artificial" variable; this variable will only occur in this particular group, will be non-negative and its value will be subtracted from the group function. For constrained problems, this is equivalent to requiring the constraint/row be non-negative; the extra variable is then a surplus variable and whether it is used explicitly (considered as a problem variable) or implicitly will depend upon the optimization technique to be used. Thus, if the problem variables are $\mathbf{x}$, and the k–th group has a linear element $\mathbf{a}_k{}^T\mathbf{x}-b_k$, the linear element that will be passed to the optimization procedure could be $\mathbf{a}_k{}^T\mathbf{x}-y_k-b_k$, for some non-negative variable y_k.

`L` the group is to use an extra "artificial" variable; this variable will only occur in this particular group, will be non-negative and its value will be added to the group function. For constrained problems, this is equivalent to requiring the constraint/row be non-positive; the extra variable is then a slack variable and may be used explicitly or implicitly by the optimization procedure. Thus, if the linear element is as specified above, the linear element that will be passed to the optimization procedure could be $\mathbf{a}_k{}^T\mathbf{x}+y_k-b_k$, for some non-negative variable y_k.

`E` the group is a normal one (for constrained problems, the row/constraint is an equality),

`X` and `Z` an array of groups are to be defined at once. When the first character is an `X` or `Z`, the second character may be one of `N, G, L` or `E`. The resulting array of groups then each has the characteristics of an `N, G, L` or `E` group as just described.

↔	← 10 →	← 10 →	← 12 →	← 10 →	← 12 →
F.1	Field 2	Field 3	Field 4	Field 5	Field 6
GROUPS or ROWS or CONSTRAINTS					
N	group_name	$$$$$$$$$$	numerical_vl		
G	group_name	$$$$$$$$$$	numerical_vl		
L	group_name	$$$$$$$$$$	numerical_vl		
E	group_name	$$$$$$$$$$	numerical_vl		
XN	group_name	$$$$$$$$$$	numerical_vl		
XG	group_name	$$$$$$$$$$	numerical_vl		
XL	group_name	$$$$$$$$$$	numerical_vl		
XE	group_name	$$$$$$$$$$	numerical_vl		
ZN	group_name	$$$$$$$$$$		r_p_a_name	
ZG	group_name	$$$$$$$$$$		r_p_a_name	
ZL	group_name	$$$$$$$$$$		r_p_a_name	
ZE	group_name	$$$$$$$$$$		r_p_a_name	
DN	group_name	$$$$$$$$$$	numerical_vl	$$$$$$$$$$	numerical_vl
DG	group_name	$$$$$$$$$$	numerical_vl	$$$$$$$$$$	numerical_vl
DL	group_name	$$$$$$$$$$	numerical_vl	$$$$$$$$$$	numerical_vl
DE	group_name	$$$$$$$$$$	numerical_vl	$$$$$$$$$$	numerical_vl
↑↑	↑ ↑	↑ ↑	↑ ↑ ↑	↑ ↑	↑ ↑ ↑
23	5 14	15 24	25 36	40 49	50 61

Figure 2.2.6. Possible data cards for GROUPS, ROWS or CONSTRAINTS (column-wise)

D the group is to be formed as a linear combination of two previous groups. When the first character is a D, the second character may be one of N, G, L or E. The resulting group then has the characteristics of an N, G, L or E group as just described.

The string group_name in data field 2 gives the name of the group (or row or constraint) under consideration. This name may be up to ten characters long, excepting that the name 'SCALE' is not allowed. For X data cards, the expanded array name must be valid and the integer indices must have been defined in an parameter assignment (see §2.2.3).

The string $$$$$$$$$$ in data field 3 may be blank; this happens when field 2 is merely announcing the name of a group. If it is not blank, it is used for two purposes.

· It may be used to announce that all the entries (if any) in the linear element for the group under consideration are to be scaled, that is *divided* by a constant scale factor; in this case field 3 will contain the string 'SCALE'. If the first character in field 1 is a Z, the string in data field 5 gives the name of a previously defined real parameter and the numerical value associated with this parameter gives the scale factor. Otherwise, the string numerical_vl, occupying up to 12 locations in data field 4, contains the scale factor. Fields 5 and 6 are not then used.

· If the first character in field 1 is a D, the current group is to be formed as a linear combination of the groups mentioned in fields 3 and 5; the multiplication factors are then recorded in fields 4 and 6 respectively. Thus we will have

group in field 2 = group in field 3 * field 4 + group in field 5 * field 6.

In this case, the names of the groups in fields 3 and 5 must have already been defined. The multiplication factors may occupy up to 12 locations in fields 4 and 6.

2.2.7 The VARIABLES or COLUMNS data cards (variable/column-wise)

The VARIABLES or COLUMNS indicator cards are used interchangeably to announce the (problem) variables for the minimization. In addition, the entries for the linear elements are input here. The user may also give a scaling factor for the entries in any column. The syntax for data following this indicator card is given in Figure 2.2.7.

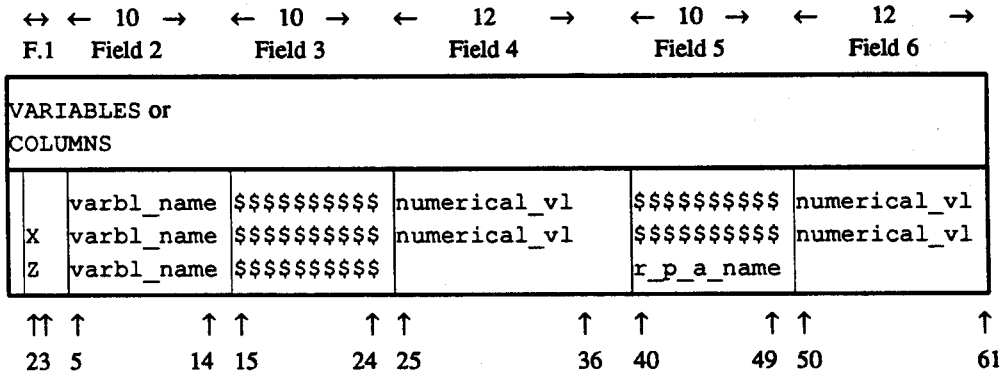


Figure 2.2.7. Possible data cards for VARIABLES or COLUMNS (column-wise)

The string varbl_name in data field 2 gives the name of the variable (or column) under consideration. This name may be up to ten characters long excepting that the name 'SCALE' is not allowed. If data field 1 holds the character X or Z, an array of variables are to be defined. In this case, the expanded array name of the variables (or columns) must be valid and the integer indices must have been defined in a parameter assignment (see §2.2.3).

The string $$$$$$$$ in data field 3 is used for three purposes.

- If the string is empty, the card is just defining the name of a problem variable.
- It may be used to specify that the variable mentioned in field 2 occurs in the linear element for the group given in field 3. In this case, the string in field 3 must have been defined in the GROUPS section. If an array definition is being made, the string in field 3 must be an array name.
- It may be used to announce that all the entries in the linear elements for the variable under consideration are to be scaled; in this case field 3 will contain the string 'SCALE'.

A numerical value, whose purpose depends on the string in the previous field, is now specified. On Z cards, the value is that previously associated with the real parameter r_p_a_name in field 5. On other cards, the actual numerical value numerical_vl may occupy up to 12 characters in data field 4.

If field 3 indicates that an entry for the linear element for a group is to be defined, the specified numerical value gives the coefficient of that entry. If, on the other hand, field 3 indicates that all entries for the variable in field 2 are to be scaled, the specified value gives the scale factor, that is the factor by which each entry is to be divided.

On non Z cards, the strings in fields 5 and 6 are optional and are used exactly as for strings 3 and 4 to define further entries or a scale factor.

2.2.8 The VARIABLES or COLUMNS data cards (group/row-wise)

The VARIABLES or COLUMNS indicator cards are used interchangeably to announce the (problem) variables for the minimization. The user may also give a scaling factor for the entries in the column. The syntax for data following this indicator card is given in Figure 2.2.8.

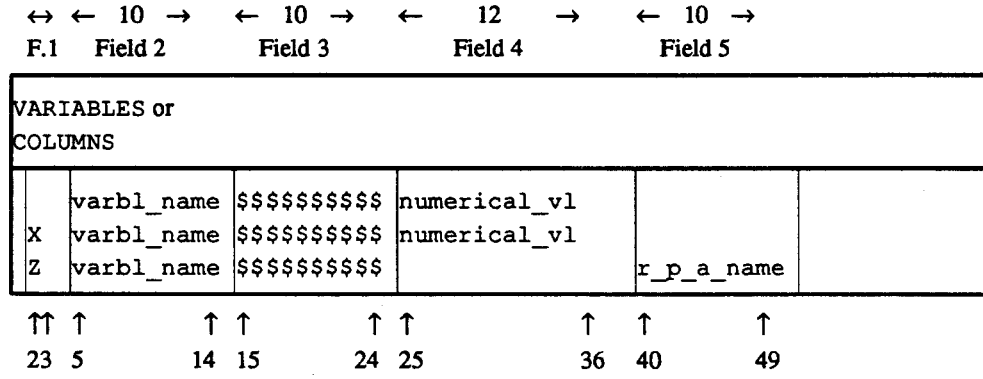


Figure 2.2.8. Possible data cards for VARIABLES or COLUMNS (row-wise)

The string varbl_name in data field 2 gives the name of the variable (or column) under consideration. This name may be up to ten characters long excepting that the name 'SCALE' is not allowed. If data field 1 holds the character X or Z, an array definition is to be made. In this case, the expanded array name of the variables (or columns) must be valid and the integer indices must have been defined in a parameter assignment (see §2.2.3).

The string $$$$$$$$$$ in data field 3 is used for two purposes.

- If the string is empty, the card is just defining the name of a problem variable. Such a card must be inserted for all variables that only appear in nonlinear elements.
- It may be used to announce that all the entries in the linear elements for the variable under consideration are to be scaled. On Z cards, the numerical value of this scale factor, the amount by which each entry is to be divided, is that previously associated with the real parameter r_p_a_name given in field 5. On other cards, the actual scale factor numerical_vl occupies up to 12 characters in data field 4.

2.2.9 The GROUPS, ROWS or CONSTRAINTS data cards (group/row-wise)

The GROUPS, ROWS and CONSTRAINTS indicator cards are used interchangeably to announce the names of the groups which make up the objective function and, for constrained problems, the names of the constraints (or rows as they are often known in linear programming applications). In addition, the entries for the linear elements are input here. The user may give a scaling factor for the groups or constraints Furthermore, groups which are linear combinations of previous groups may be specified. The syntax for the data cards which follow these indicator cards is given in Figure 2.2.9.

The one or two character string in data field 1 specifies the type of group, row or constraint to be input. Possible values for the first character and their interpretations are exactly as in §2.2.6.

↔	← 10 →	← 10 →	← 12 →	← 10 →	← 12 →
F.1	Field 2	Field 3	Field 4	Field 5	Field 6
GROUPS or ROWS or CONSTRAINTS					
N	group_name	$$$$$$$$$$	numerical_vl	$$$$$$$$$$	numerical_vl
G	group_name	$$$$$$$$$$	numerical_vl	$$$$$$$$$$	numerical_vl
L	group_name	$$$$$$$$$$	numerical_vl	$$$$$$$$$$	numerical_vl
E	group_name	$$$$$$$$$$	numerical_vl	$$$$$$$$$$	numerical_vl
XN	group_name	$$$$$$$$$$	numerical_vl	$$$$$$$$$$	numerical_vl
XG	group_name	$$$$$$$$$$	numerical_vl	$$$$$$$$$$	numerical_vl
XL	group_name	$$$$$$$$$$	numerical_vl	$$$$$$$$$$	numerical_vl
XE	group_name	$$$$$$$$$$	numerical_vl	$$$$$$$$$$	numerical_vl
ZN	group_name	$$$$$$$$$$		r_p_a_name	
ZG	group_name	$$$$$$$$$$		r_p_a_name	
ZL	group_name	$$$$$$$$$$		r_p_a_name	
ZE	group_name	$$$$$$$$$$		r_p_a_name	
DN	group_name	$$$$$$$$$$	numerical_vl	$$$$$$$$$$	numerical_vl
DG	group_name	$$$$$$$$$$	numerical_vl	$$$$$$$$$$	numerical_vl
DL	group_name	$$$$$$$$$$	numerical_vl	$$$$$$$$$$	numerical_vl
DE	group_name	$$$$$$$$$$	numerical_vl	$$$$$$$$$$	numerical_vl
↑↑ ↑	↑	↑ ↑	↑	↑ ↑	↑ ↑ ↑
23 5	14	15 24	25 36	40 49	50 61

Figure 2.2.9. Possible data cards for GROUPS, ROWS or CONSTRAINTS (row-wise)

The string group_name in data field 2 gives the name of the group (or row or constraint) under consideration. This name may be up to ten characters long excepting that the name 'SCALE' is not allowed. For X and Z data cards, the expanded array name must be valid and the integer indices must have been defined in a parameter assignment (see §2.2.3). The kind of group (N, L, G, or E) will be taken to be that which is defined on the *first* occurrence of a data card for that group. Subsequent contradictory information will be ignored.

The string $$$$$$$$$$ in data field 3 is used for three purposes.

- It may be used to specify that the group mentioned in field 2 has a linear element involving the variable given in field 3. In this case, the string in field 3 must have been defined in the VARIABLES section. If an array definition is being made, the string in field 3 must be an array name. The numerical value of the coefficient of the linear term corresponding to the variable. must now be specified. On Z cards, the value is that previously associated with the real parameter r_p_a_name given in field 5. On other cards, the actual numerical value numerical_vl may occupy up to 12 characters in data field 4.

- It may be used to announce that all the entries (if any) in the linear element for the group under consideration are to be scaled; in this case field 3 will contain the string 'SCALE'. The numerical value of the scale factor, that is the factor by which the group is to be divided, is now specified exactly as above. In these first two cases, fields 5 and 6 may be used to define further coefficients or a scale factor for non Z cards.

· If the first character in field 1 is a `D`, the current group is to be formed as a linear combination of the groups mentioned in fields 3 and 5; the multiplication factors are then recorded in fields 4 and 6 respectively. Thus we will have

group in field 2 = group in field 3 * field 4 + group in field 5 * field 6.

In this case, the names of the groups in fields 3 and 5 must have already been defined. The multiplication factors may occupy up to 12 locations in fields 4 and 6.

2.2.10 The `CONSTANTS`, `RHS` or `RHS'` data cards

The `CONSTANTS`, `RHS` or `RHS'` indicator cards are used interchangeably to announce the definition of the constant term b_i (in the constrained case, the right-hand-side) for each linear element. The syntax for data following this indicator card is given in Figure 2.2.10.

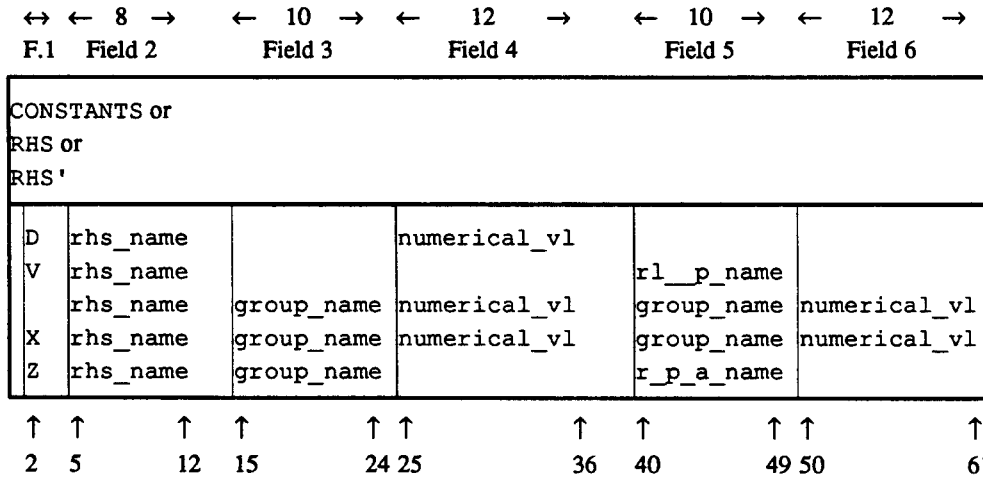


Figure 2.2.10. Possible data cards for `CONSTANTS`, `RHS` or `RHS'`

The string `rhs_name` in data field 2 gives the name of the vector of group constants/ right-hand-sides. This name may be up to eight characters long. More than one vector of group constants may be defined.

The strings `group_name` in data field 3 and (optionally) 5 specify groups/rows/constraints for which the constant term/right-hand-side is to be specified. Such strings must have been defined in the `GROUPS` section. The string `numerical_vl` in data field 4 and (optionally) 6 now contains the numerical value of the constant/right-hand-side and may occupy up to 12 locations.

Constants for an array of groups may also be defined on cards in which field 1 contains the character `X` or `Z`. On such cards, the expanded array name in field 3 and (as an option on `X` cards) 5 must be valid and the integer indices must have been defined in a parameter assignment (see §2.2.3). On `Z` cards, the numerical value of the constant/right-hand-side is that previously associated with the real parameter array, `r_p_a_name`, given in field 5. On `X` cards, the actual numerical value `numerical_vl` may occupy up to 12 characters in data fields 4 and (optionally) 6.

Any constants not specified take a default value. The default value for the components of each vector is initially zero. This default may be changed using a card whose first field is the character `D` or `V` The default value on a `D` card is specified in data field 4 for the vector named in field 2. For an `V` card, the

default value for the vector in field 2 is that associated with the real parameter, `rl__p_name`, named in field 5. The default value applies to each constant not explicitly specified; if the default is to be changed, the change must be made on the first card naming a particular vector of constants.

2.2.11 The RANGES data cards

The RANGES indicator card is used to announce the definition of additional bounds on the artificial variables introduced in the GROUPS section (in the constrained case, this corresponds to saying that specified inequality constraints/rows have both lower and upper bounds). The syntax for data following this indicator card is given in Figure 2.2.11.

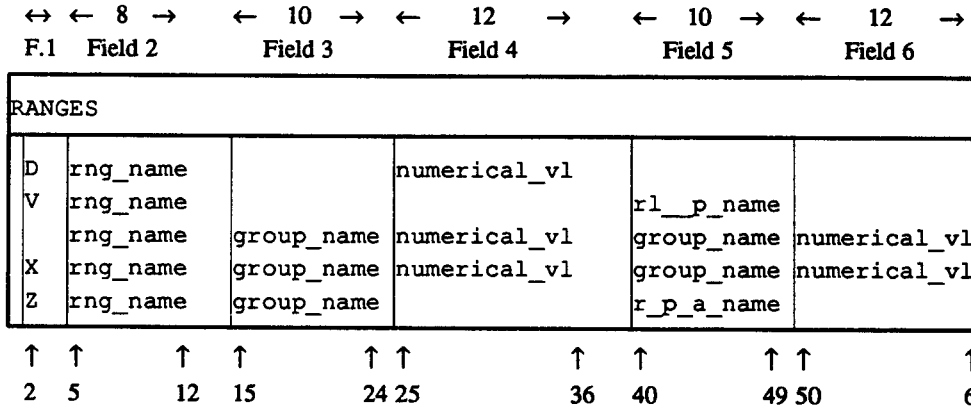


Figure 2.2.11. Possible data cards for RANGES

The string `rng_name` in data field 2 gives the name of the vector of range values. This name may be up to eight characters long. More than one vector of range values may be defined.

The strings `group_name` in data fields 3 and (optionally) 5 specify groups/rows/constraints whose additional variable is to be given a second bound. Only groups initially specified with a `G` or `L` in columns 1 or 2 of field 1 in the GROUPS section have such a variable and therefore only these groups may be specified.

Range values for an array of groups may also be defined on cards on which field 1 is the character `X` or `Z`. On such cards, the expanded array name in field 3 and (as an option on `X` cards) 5 must be valid and the integer indices must have been defined in a parameter assignment (see §2.2.3). On `Z` cards, the range value is that previously associated with the real parameter, `r_p_a_name`, given in field 5. On `X` cards, the actual numerical value `numerical_vl` may occupy up to 12 characters in data fields 4 and (optionally) 6. Using the terminology of §2.2.3, the extra bound is taken to imply the inequality $0 \leq y_k \leq |$ field 4 or 6 $|$ on the artificial variable y_k.

Any component in a range vector not specified takes a default value. The default value for the components of each vector is initially infinite. This default may be changed using a card whose first field is the character `D` or `V` The default value on a `D` card is specified in data field 4 for the vector named in field 2. For an `V` card, the default value for the vector in field 2 is that associated with the real parameter, `rl__p_name`, named in field 5. If the default is to be changed, the change must be made on the first card naming a particular range vector.

2.2.12 The BOUNDS data cards

The BOUNDS indicator card is used to announce data relating to lower and upper bounds on the unknown variables. The syntax for data following this indicator card is given in Figure 2.2.12.

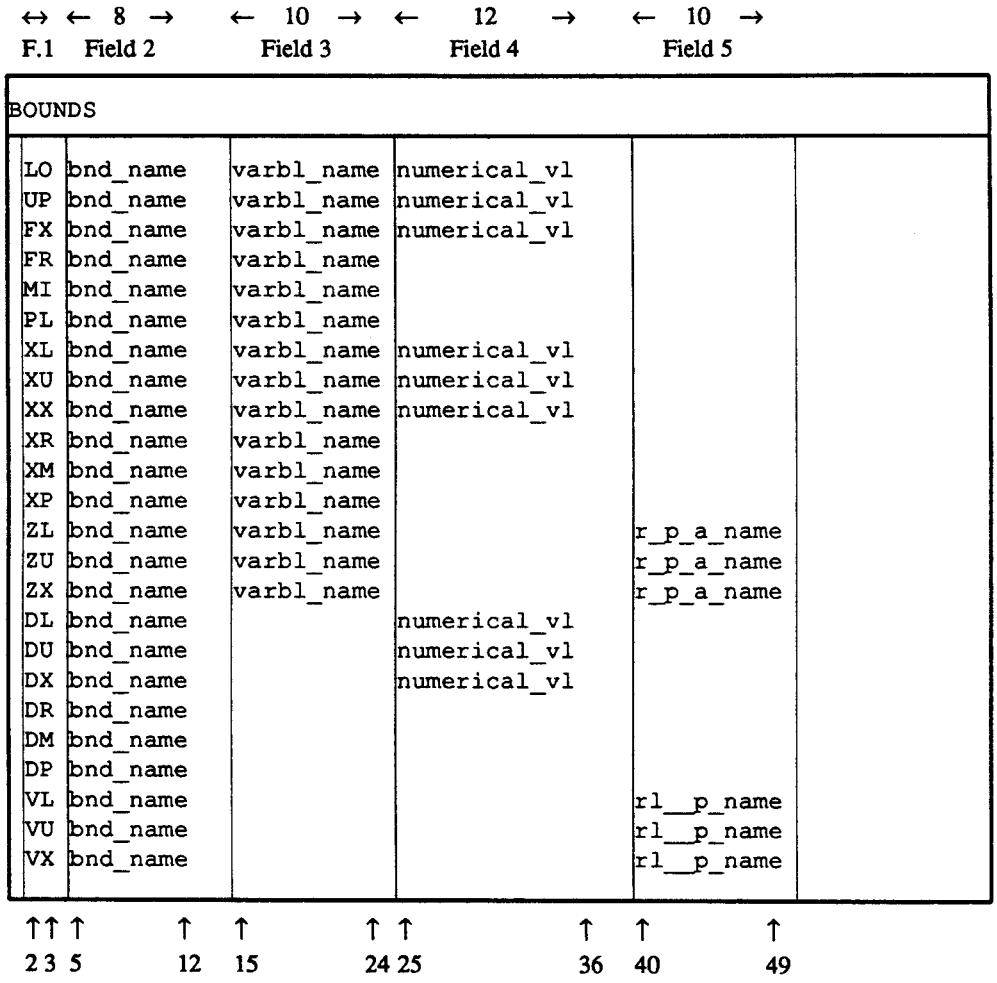


Figure 2.2.12. Possible data cards for BOUNDS

The two character string in data field 1 specifies the type of bound to be input. Possible values are: LO, XL, ZL, DL or VL a lower bound, UP, XU, ZU, DU or VU an upper bound, FX, XX, ZX, DX or VX a fixed variable, i.e., the lower and upper bounds are equal, FR, XR or DR a free variable, i.e., the lower and upper bounds are infinite, MI, XM or DM no lower bound and PL, XP or DP no upper bound. The string bnd_name in data field 2 gives the name of the bound vector under consideration. This name may be up to eight characters long. Several different bound vectors may be defined in the BOUNDS section.

If the card is of type LO, UP, FX, FR, MI or PL, the string varbl_name in data field 3 specifies to which variable the bound is applied. This name may be up to ten characters long and must refer to a variable defined in the VARIABLES data. If the card is of type XL, ZL, XU, ZU, XX, ZX, XR, XM or

XP, the string varbl_name in data field 3 specifies an array of variables which are to be bounded. On such cards, the expanded array name of this string must be valid and the integer indices must have been defined in a parameter assignment (see §2.2.3). This name may be up to ten characters long and must refer to a variable defined in the VARIABLE data. For bounds of type LO, UP, FX, XL, XU or XX, the numerical value of the bound or array of bounds is given as the string numerical_vl using at most 12 characters in data field 4. For bounds of type ZL, ZU or ZX, the numerical value of the array of bounds is that previously associated with the real parameter array r_p_a_name specified in field 5. When both lower and upper bounds on a variable are required, they must be specified on separate cards. Possible combinations are LO-UP, LO-PL, MI-UP XL-XU, XL-XP, XM-XU, ZL-XU, XL-RU, ZL-RU, ZL-XP and XM-RU.

Each bound vector is given default lower and upper bounds on every variable. The value of the default lower bound is initially zero and the upper bound is initially infinite. The default value for the bound vector specified on field 2 on any card whose first string starts with the character D or V may be changed, the particular bound under consideration is as defined above. The default value specified on DL, DU and DX cards has the numerical value given in data field 4. Similarly, the default value specified on VL, VU and VX cards is that associated with the real parameter, rl__p_name, specified in field 5. If the defaults are to be changed, the changes must be made on the first cards naming a particular vector of bounds.

2.2.13 The START POINT data cards

The START POINT indicator card is used to announce initial estimates of the values of the unknown variables. The syntax for data following this indicator card is given in Figure 2.2.13.

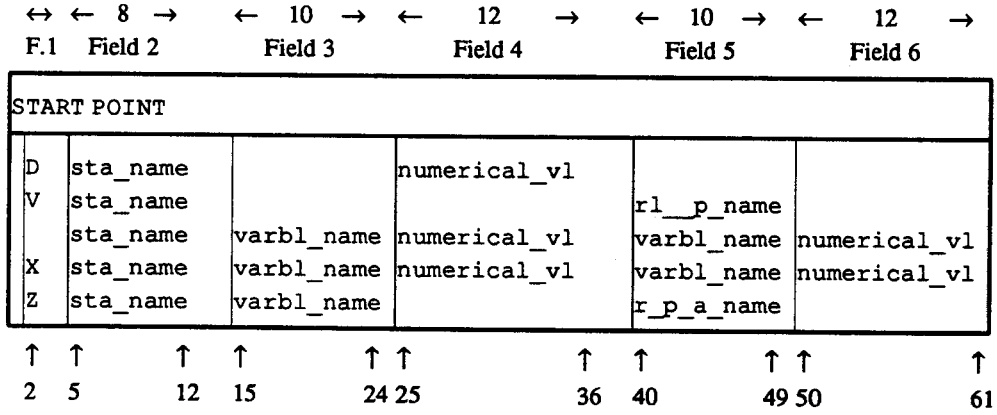


Figure 2.2.13. Possible data cards for START POINT

Any card with an empty field 1 is used to define the starting value for an individual variable. The string sta_name in data field 2 gives the name of a starting vector and may be up to eight characters long. Several different starting vectors may be defined in the START POINT section. The string varbl_name in data field 3 (and optionally field 5) gives the name of the variable whose starting value is to be assigned. This name must refer to a variable defined in the VARIABLE data.

Starting values for an array of variables may also be defined on cards with the character X or Z in field 1; on X cards two arrays may be defined on a single card. On such cards, the expanded array name in field 3 (and field 5 for X cards) must be valid and the integer indices must have been defined in a parameter assignment (see §2.2.3).

Each variable not explicitly specified is assumed to have a default starting value. The default for each vector of starting values is initially zero but may be changed for the starting vector referred to in field 2 of a card containing the character D or V in field 1. However, if the default is to be changed, the change must be made on the first card naming a particular vector of starting values.

It remains to specify the numerical value of the default or individual starting point as appropriate On V or Z cards, the value is that previously associated with the real parameter `rl__p_name` or array of real parameters `rl_p_a_name` (respectively) given in field 5. On other cards, the numerical value is (or values are) specified using up to twelve characters in the string(s) `numerical_vl` in data field 4 (and if required field 6).

2.2.14 The ELEMENT TYPE data cards

The ELEMENT TYPE indicator card is used to announce the data for the different types of nonlinear elements which are to be used. The names of the elemental and, optionally, internal variables and parameters for each element type are specified in this section. The syntax for data cards following the indicator card is given in Figure 2.2.14.

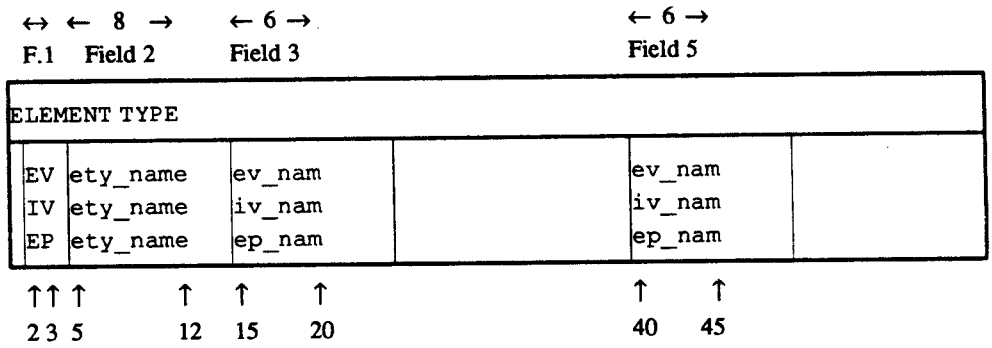


Figure 2.2.14. Possible data cards for ELEMENT TYPE

The string in field 1 may be one of EV, IV or EP. This indicates whether the names of elemental variables (EV), internal variables (IV) or elemental parameters (EP) are to be specified on the given data card. If no cards with the string IV in field 1 are found for a particular element type, the element is assumed to have no useful internal variables; the internal variables are then allocated the same names as the elemental ones. Likewise, if no cards with the string EP in field 1 are found for a particular element type, the element is assumed not to depend on parameter values.

The string `ety_name` in data field 2 gives the name of the element type under consideration. This name may be up to eight characters long. The data for a particular element must be specified on consecutive data cards.

The strings in data fields 3 and (optionally) 5 give the names of elemental variables (field 1 =EV), internal variables (field 1 =IV) or parameters (field 1 =EP) for the element type specified in field 2. These strings may be up to 6 characters long. The names of the variables for different element types may be the same; the names of the elemental variables, internal variables and parameters (if the latter two are given) for a specific element type must all be different.

2.2.15 The ELEMENT USES data cards

The ELEMENT USES indicator card is used to specify the names and types of the nonlinear element functions. The element types may be selected from among those defined in the ELEMENT TYPE section. Associations are made between the problem variables and the elemental variables for the elements used and parameter values are assigned. The syntax for data following this indicator card is given in Figure 2.2.15.

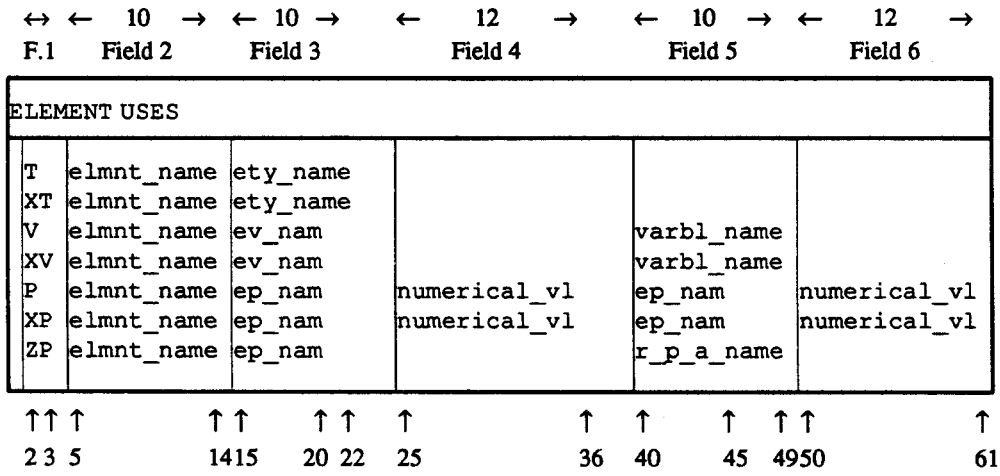


Figure 2.2.15. Possible data cards for ELEMENT USES

For each card, the string elmnt_name in data field 2 gives the name, or an array of names, of a nonlinear element function This name may be up to ten characters long and each nonlinear element name must be unique. On array cards (those prefixed by X or Z), the expanded element array name in field 2 must be valid and the integer indices must have been defined in a parameter assignment (see §2.2.3).

There are three sorts of data cards in the ELEMENT USES section. The first, identified by the characters T or XT in field 1, give the name, or an array of names, of an element and its type. The string ety_name in data field 3 gives the name of the element type to be used. This name may be up to eight characters long and must have appeared in the ELEMENT TYPE section.

The second kind of data card, identified by the characters V or XV in field 1, is used to assign problem variables to the elemental variables appropriate for the element type. On this data card, the string ev_nam in data field 3 gives the name of one of the elemental variables for the given element type. This name must have been set in the ELEMENT TYPE section and can be at most six characters long. The string varbl_name in data field 5 then gives the name of one of the problem variable that is to be assigned to the specified elemental variable. The name of this variable may have been set in the VARIABLES/COLUMNS section or may be a new variable (often known as a *nonlinear variable*) introduced here and can be up to ten characters long. On an XV card, the name of the variable must be an element of an array of variables, with a valid name and index.

The last kind of data card, identified by the characters P, XP or ZP in field 1, is used to assign numerical values to the parameters for the element functions (P) or array of element functions (XP and ZP). On this data card, the strings ep_nam in data fields 3 (and, for P and XP cards, optionally 5) give the names of parameters. These name must have been set in the ELEMENT TYPE section and can be at most six characters long. On P and XP cards, the strings numerical_vl in data fields 4 and

(optionally) 6 contain the numerical value of the parameter. These values may each occupy up to 12 locations within their field. On ZP cards, the string r_p_a_name in data field 5 gives a real parameter array name. This name must have been previously defined and its associated value then gives the numerical value of the parameter.

2.2.16 The GROUP TYPE data cards

The GROUP TYPE indicator card is used to announce the data for the different types of nontrivial groups which are to be used. The names of the group-type variable and, optionally, of group parameters for each group type are specified in this section. The syntax for data cards following the indicator card is given in Figure 2.2.16.

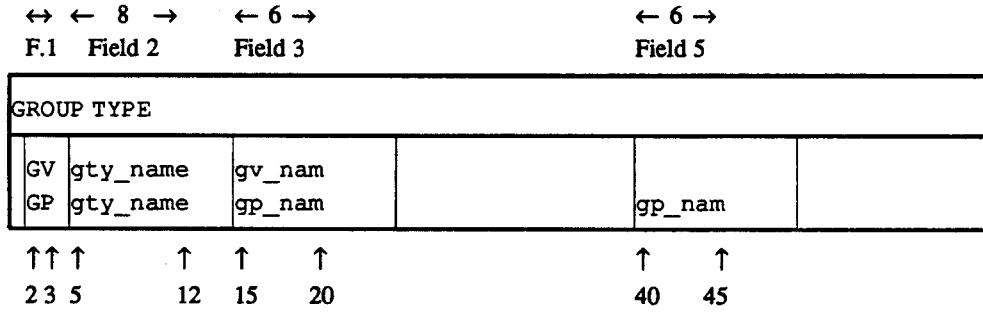


Figure 2.2.16. Possible data cards for GROUP TYPE

The string in field 1 may be either GV or GP. This indicates whether the name of a group-type variable (GV) or one or more group parameters (GP) are to be specified on the given data card. The data for a particular group type must be specified on consecutive data cards. The string gty_name in data field 2 gives the name of a nontrivial group type and may be up to eight characters long. If data field 1 holds GV, the string gv_nam in data field 3 then gives the name of the group-type variable for this group type. This string may be up to 6 characters long. The names of the variables for different group types may be the same. Alternatively, if data field 1 holds GP, the strings gp_nam in data fields 3 and (optionally) 5 give the names of parameters for the group type. These strings may again be up to 6 characters long. The names of parameters for different group types may be the same; the names of the group-type variable and parameters (if the latter appear) for a specific group type must all be different.

2.2.17 The GROUP USES data cards

The GROUP USES indicator card is used to announce which of the nonlinear elements appear in each group and the type of group function involved. The group types may be selected from among those defined in the GROUP TYPE section of the data while the elements may be selected from among the types defined in the ELEMENT USES section. In addition, group parameter values are assigned. The syntax for data following this indicator card is given in Figure 2.2.17.

For each card, the string group_name in data field 2 gives the name, or an array of names, of the group(s) (or row(s) or constraint(s)) under consideration. The name may be up to ten characters long and may have been defined in the GROUPS/ROWS/CONSTRAINTS section or may be a new group introduced for the first time here. On array cards (those prefixed by X or Z), the expanded group array

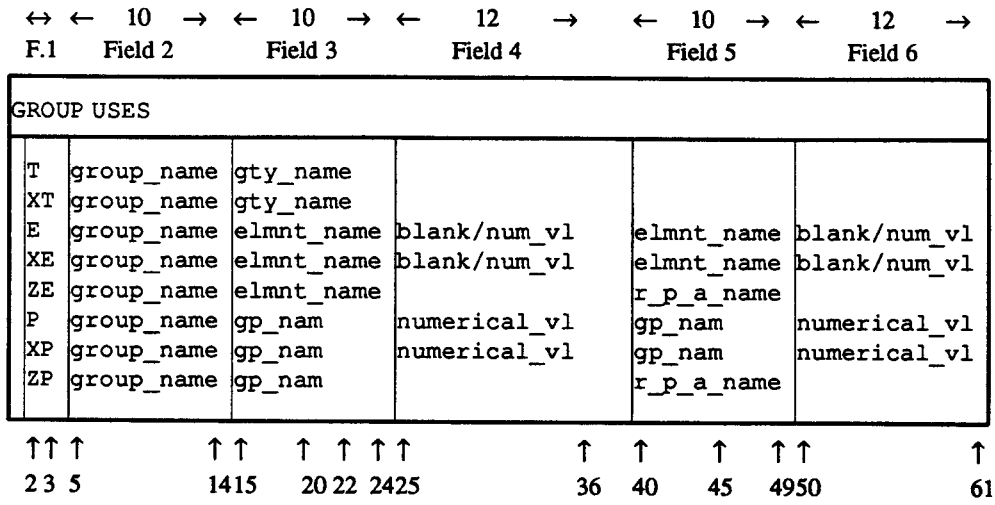


Figure 2.2.17. Possible data cards for GROUP USES

name in field 2 must be valid and the integer indices must have been defined in a parameter assignment (see §2.2.3).

There are three sorts of data cards in the GROUP USES section. The first, identified by the characters T or XT in field 1, give the name, or an array of names, of a group function and its type. T and XT cards are used to allocate a type to a group or an array of groups explicitly. Any group not explicitly typed is assumed to be trivial. The string gty_name in data field 3 gives the name of the group type to be used. This name may be up to eight characters long and must have appeared in the GROUP TYPE section.

The second kind of data card, identified by the characters E, XE or ZE in field 1, is an indication that particular nonlinear elements are to be included in a given group. Optionally the given elements are to be multiplied by specified non-unit weights. On these data cards, the string elmnt_name in data fields 3 (and optionally 5 on E and XE cards) hold the names of nonlinear elements which are to be used. The names in both fields may be up to ten characters long and must have been defined in the ELEMENT USES section. On XE and ZE cards, the names of the nonlinear elements must be components of an array of nonlinear elements, with a valid name and index. The elements are multiplied by given weights. By default, each weight takes the value 1.0. Only non-unit weights need to be specified explicitly. On E and XE cards, non-unit weights have the numerical values specified in data fields 4 (and optionally 6). These values may occupy up to 12 locations of their specified field. The default value of 1.0 is taken whenever these fields are empty. On ZE cards, the string r_p_a_name in data field 5 gives a real parameter array name. This name must have been previously defined and its associated value then gives the numerical value of the weight. Any group that is not named on an E or XE card is taken to have no nonlinear elements.

The last kind of data card, identified by the characters P, XP or ZP in field 1, is used to assign numerical values to the parameters for the group functions (P) or array of group functions (XP and ZP). On this data card, the strings gp_nam in data fields 3 (and, for P and XP cards, optionally 5) give the names of parameters. These name must have been set in the GROUP TYPE section and can be at most six characters long. On P and XP cards, the strings numerical_vl in data fields 4 and (optionally) 6 contain the numerical value of the parameter. These values may each occupy up to 12 locations of their field. On ZP cards, the string r_p_a_name in data field 5 gives a real parameter array name. This name must have been previously defined and its associated value then gives the numerical value of the parameter.

The T or XT card for a particular group must appear before its V, XV, P, XP or ZP cards.

2.2.18 The OBJECT BOUND data cards

The OBJECT BOUND indicator card is used to announce known lower and upper bounds on the value of the objective function for the problem. The syntax for data following this indicator card is given in Figure 2.2.18.

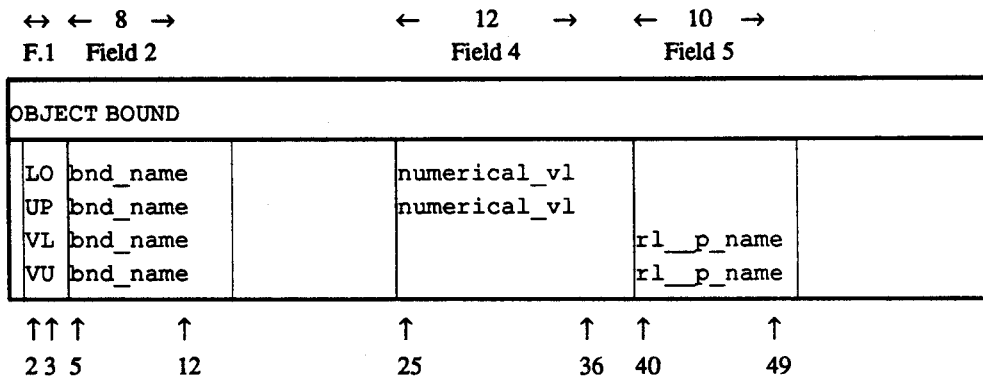


Figure 2.2.18. Possible data cards for OBJECT BOUND

The two character string in data field 1 specifies the type of bound to be input. Possible values are: LO or VL a lower bound, and UP or VU an upper bound. The string bnd_name in data field 2 gives a name to the bounds under consideration. This name may be up to eight characters long. Several different known bounds on the objective function may be defined in the OBJECT BOUND section.

For bounds of type LO or UP, the numerical value of the bound is given as the string numerical_vl using at most 12 characters in data field 4. For bounds of type IL or IU, the numerical value of the bound is that previously associated with the real parameter array r_p_a_name specified in field 5. When both lower and upper bounds on the objective are known, they must be specified on separate cards.

The objective function is assumed by default to be unbounded both below and above. The values for each named bound set may only be changed on a LO, UP, VL or VU card.

2.3 Another example

In §1.4, we gave an example. An SDIF file for this example is given in Figure 2.3.1. The problem is given the name DOC. The groups are referred to as GROUP1/2/3 and the variables are X1/2/3. The vector of bounds is called BN1 and the two types of nonlinear element are ELEMENT1/2. The elemental variables are assigned names beginning with U and the internal variables for the second nonlinear element start with V. The two group types are GTYPE1/2. Finally the nonlinear element in GROUP2 is given the name G2E1, while those in GROUP3 are G3E1/2.

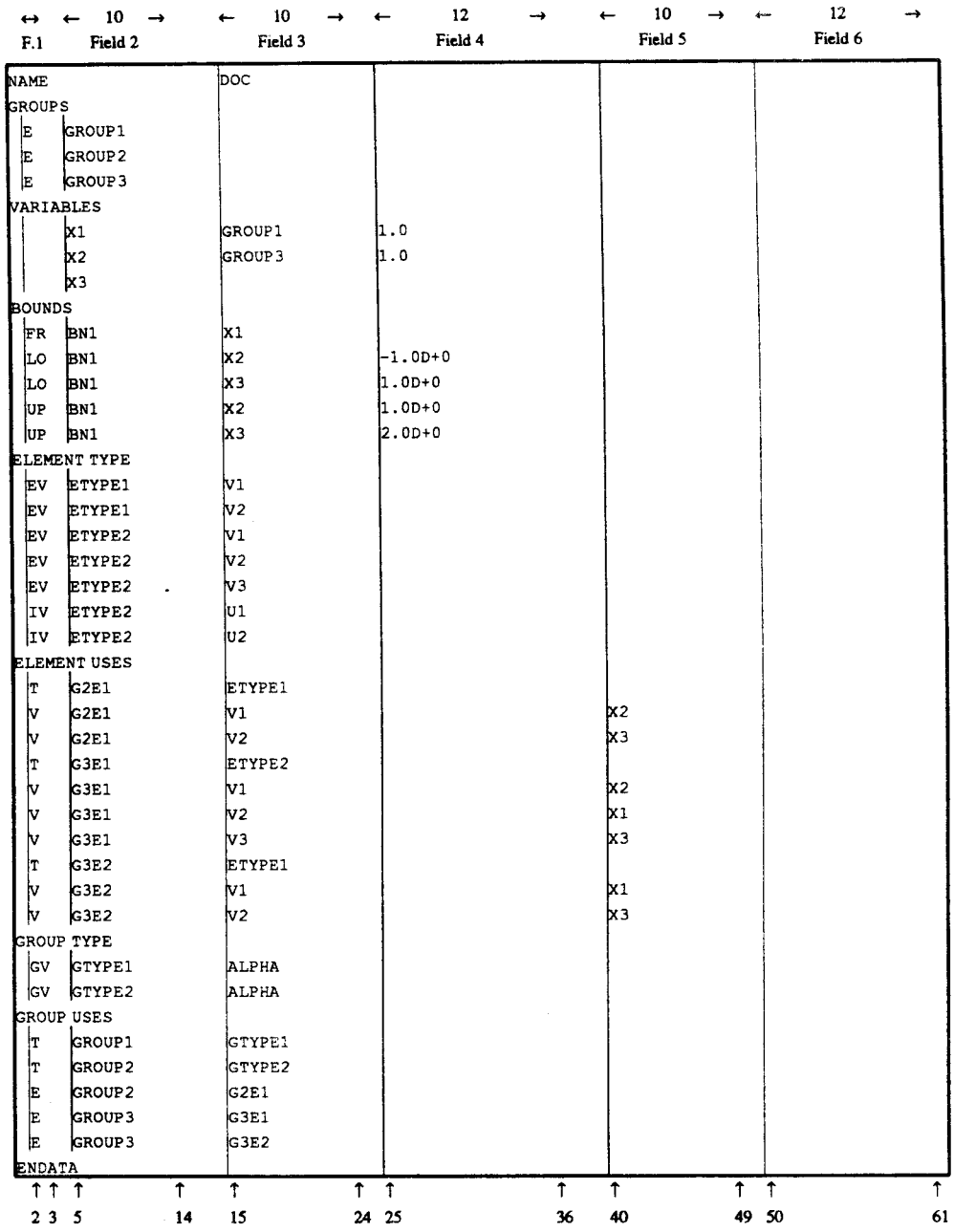


Figure 2.3.1. SDIF file for the example of §1.4

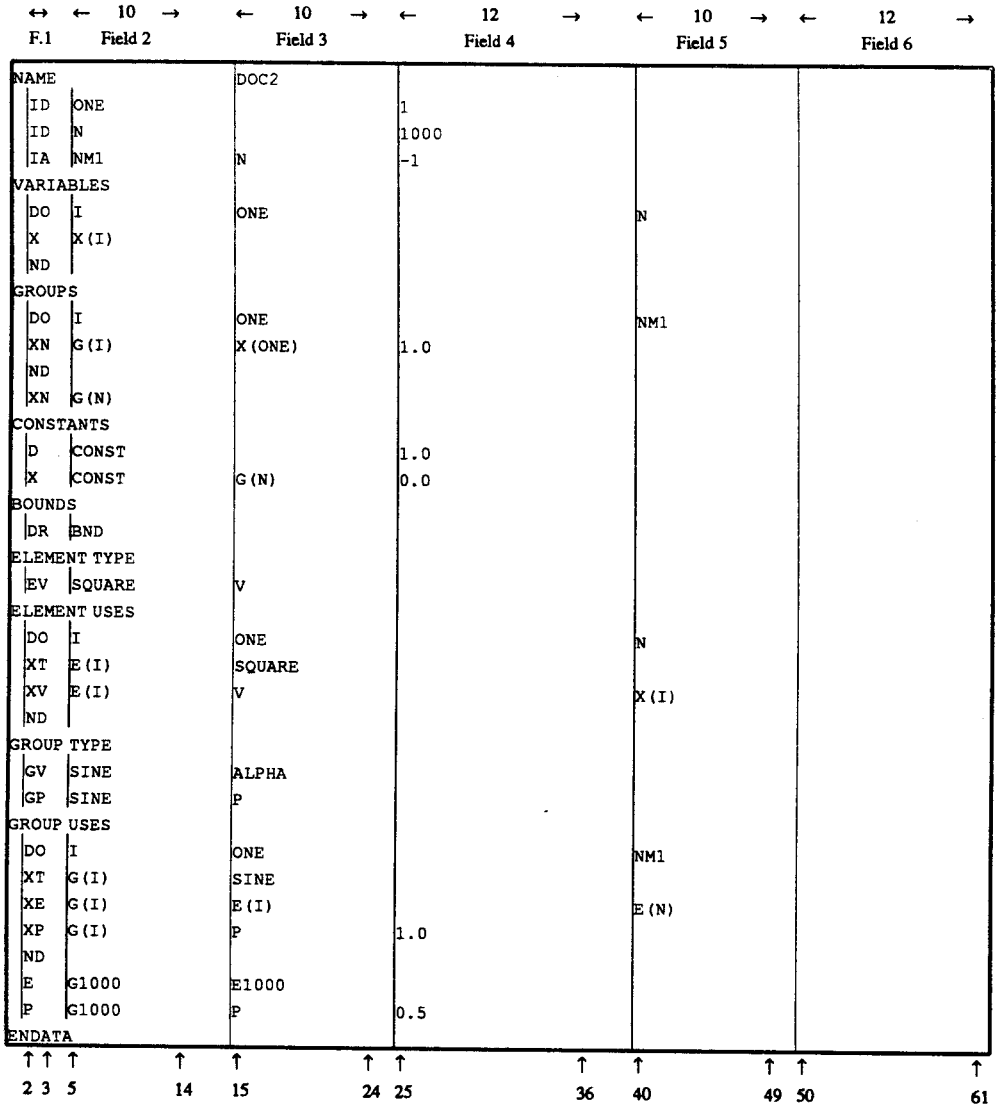


Figure 2.4.1. SDIF file for the example of §1.5

2.4 A further example

In §1.5, we gave a second example. Because of its repetitious structure, this example is well suited to use array names and do loops. An SDIF file for this example is given in Figure 2.4.1. The problem is given the name `DOC2`. The variables are referred to as `X1`, ..., `X1000` and the groups are `G1`, ..., `G1000`. The vector of bounds is called `BND`, the constants are `CONST` and single nonlinear element type is `SQUARE`, with elemental variable `V`. Note that the `BND` section is necessary since the variables are unrestricted and we must override the default lower bounds of zero and upper bounds of infinity. The nonlinear elements are given the names `E1`, ..., `E1000`. Finally, the single group type is `SINE` with group-type variable `ALPHA` and parameter `P`.

3. The standard data input format for describing nonlinear elements

In addition to the problem data described in §2, the user might also wish to specify the nonlinear element functions, and their derivatives, in a systematic way. A particular nonlinear element function is defined in terms of its problem variables and its type; both of these quantities are specified in §2. Thus, the only details which remain to be specified are the function and derivative values of the *element types* and the transformations between elemental and internal variables, if any.

In this section, we present one approach to this issue. As before, data is specified in a file. The file comprises an ordered mixture of indicator and data cards; the latter allow function and derivative definitions in appropriate high-level language statements.

3.1 Introduction to the standard element type input format

3.1.1 The values and derivatives required

It is assumed that a nonlinear element type is specified in terms of internal variables $\mathbf{u}$, whose names are those given on the `ELEMENT TYPE` data cards in an SDIF file (if the element has no useful internal variables, the internal and elemental variables are the same and the internal variables will have been named after the elementals), see §2.2.13. An optimization procedure is likely to require the values of the element functions and their first, and possibly second, derivatives. These derivatives need only be given with respect to the internal variables. For if we denote the gradient and Hessian matrix of an element function f with respect to $\mathbf{u}$ by

$$\nabla_{\mathbf{u}} f \text{ and } \nabla_{\mathbf{uu}} f$$

respectively, the gradient and Hessian matrices with respect to the elemental variables are

$$\mathbf{W}^T \nabla_{\mathbf{u}} f \text{ and } \mathbf{W}^T \nabla_{\mathbf{uu}} f \, \mathbf{W},$$

where $\mathbf{W}$ is defined by (1.2.7).

We thus need only supply derivatives with respect to $\mathbf{u}$. Formally, we must define the function value f, the gradient vector $\nabla_{\mathbf{u}} f$ (i.e., the vector whose i–th component is the first partial derivative with respect to the i–th internal variable) and, possibly, the Hessian matrix $\nabla_{\mathbf{uu}} f$ (i.e., the matrix whose i,j–th entry is the second partial derivative with respect to the i–th and j–th internal variables), all evaluated at $\mathbf{u}$. We now describe how to set up the data for a given problem.

3.1.2 Indicator cards

As before, the user must prepare an input file, the SEIF (Standard Element type Input Format) file, consisting of indicator and data cards. The former contain a simple keyword to specify the type of data that follows. Possible indicator cards are given in Figure 3.1.

keyword	comments	presence	described in §
`ELEMENTS`	same as `NAME`	mandatory	2.2.1
`TEMPORARIES`		optional	3.2.1
`GLOBALS`		optional	3.2.2
`INDIVIDUALS`		optional	3.2.3
`ENDATA`		mandatory	2.2.2

Figure 3.1. Possible indicator cards

Indicator cards must appear in the order shown. The cards `TEMPORARIES` `GLOBALS` and `INDIVIDUALS` are optional.

The data cards are of two kinds. The first are like those described in §2.1. The others use four fields, fields 1, 2 and 3, as before, and field 7 which starts in column 25 and is 41 characters long. This last field is used to hold arithmetic expressions. An *arithmetic expression* is as defined in the Fortran programming language standard (ANSI X3.9-1978) although it is not intended that expressions necessarily be restricted to Fortran but rather to the particular programming language appropriate to the users optimization procedure. We allow the use of any of the chosen language's intrinsic functions in such expression. Continuation of an expression over at most nineteen lines is also permitted.

3.1.3 An example.

Before we give the complete syntax for an SEIF file, we continue the illustrative example that we started in §2.1.3 and show how to specify an input file appropriate for the problem of §1.6. Once again, there are many possible ways of specifying a particular problem; we give one in Figure 3.1.3. The arithmetic expressions given are written in Fortran.

The file must always start with an `ELEMENTS` card, on which a name (in this case `EG3`) for the example may be given (line 1), and must end with an `ENDATA` card (line 40).

We next need to specify the names and attributes of any auxiliary quantities and functions that we intend to use in our high level description of the element functions. These are needed to allow for consistency checks in the proceeding high-level language statements and must always occur in the `TEMPORARIES` section of the input file. Lines 3 to 6 indicate that we shall be using temporary quantities `SINV1`, `ZERO`, `ONE` and `TWOP1` and the character `R` in the first field for these lines states that these quantities will be associated with floating point (real) values. The character `M` in field 1 of Lines 7 and 8 indicate that we may use the intrinsic (machine) functions `SIN` and `COS`. These are of course Fortran intrinsic functions appropriate for the high-level language used here.

We now specify any numerical values which are to be used in one or more element descriptions within the `GLOBALS` section. On lines 10 and 11, we allocate the values 0 and 1 to the previously defined quantities `ZERO` and `ONE`. Note that such cards require the character `G` in field 1 – if an assignment were to take more than 41 characters (the width of field 7), it could be continued on subsequent lines for which the string `G+` is required in field 1.

Finally we need to make the actual definitions of the function and derivative values for the element types and specify the transformations from elemental to internal variables in they are used. Such specifications occur in the `INDIVIDUALS` section from lines 12 to 39 of the example. We recall that there are four element types `3PROD`, `2PROD`, `SINE` and `SQUARE` and that their attributes (names of elemental and internal variables and parameters) have been described in the SDIF file set up in §2.1.3. Two of the element types (`3PROD` and `SQUARE`) use internal variables so we need to describe the relevant transformation for those.

On line 13, the presence of the character `T` in field 1 announces that the data for the element type `3PROD` is to follow. All the data for this element must be specified before another element type is considered. On lines 14 and 15 we describe the transformation from elemental to internal variables that is used for `3PROD`. Recall that the transformation is $u_1 = v_1 - v_2$ and $u_2 = v_3$. On line 14, the first of these transformations is given, namely that `U1` is to be formed by adding 1.0 times `V1` to -1.0 times `V2`. The second transformation is given on the following line, namely that `U2` is formed by taking 1.0 times `V3`. Both lines are marked as defining transformations by the character `R` in field 1 – continuation lines are possible for transformations which involve more than two elemental variables on lines in which the string `R+` appears in the same field.

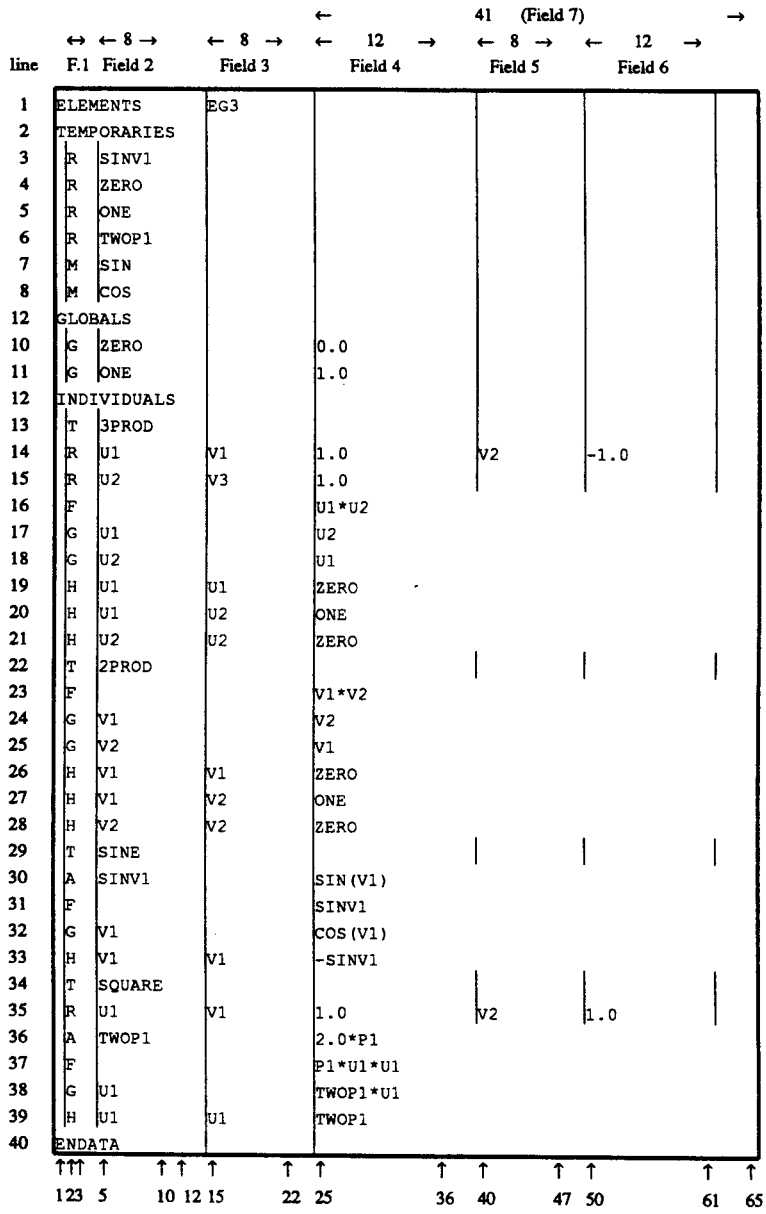


Figure 3.1.3. SEIF file for the nonlinear element types for the example of §1.6

We now specify the function and derivative values of the element type $u_1 u_2$ with respect to its internal variables. On line 16, the code F in field 1 indicates that we are setting the value of the element type to U1*U2, the Fortran expression for multiplying U1 and U2. On lines 17 and 18, we specify the first derivatives of the element type with respect to its two internal variables U1 and U2 – the character G in field 1 indicates that gradient values are to be set. On line 17, the derivative with respect to the variable U1, specified in field 2, is taken and expressed as U2 in field 7. Similarly, on line 18, the derivative with respect to the variable U2 (in field 2), U1, is given in field 7. Finally, on lines 19 to 21, the second partial derivatives with respect to both internal variables are given. These derivatives appear on cards whose first field contains the character H. On line 19, the second derivative with respect to the variables U1 (in field

2) and U1 (in field 3), 0.0, is given in field 7. Similarly the second derivative with respect to the variables U1 (in field 2) and U2 (in field 3), 1.0, occurs in field 7 of line 20 and that with respect to U2 (in field 2) and U2 (in field 3), 0.0, is given in field 7 of the following line.

The same principle is applied to the specification of range transformations, values and derivatives for the remaining element types. The type 2PROD does not use a transformation to internal variables, so derivatives are taken with respect to the elemental variables V1 and V2 (or one might think of the internal variables being V1 and V2, related to the elemental variables through the identity transformation). The values and derivatives for this element type are given on lines 22 to 28. The type SINE again does not use special internal variables and the required value and derivatives are given on lines 29 to 33. Note, however, that the value and its second derivative with respect to v_1 both use the quantity $\sin v_1$; for efficiency, we set the auxiliary quantity SINV1 to the Fortran value SIN(V1) on line 30 and thereafter refer to SINV1 on lines 31 and 33. Notice that this definition of auxiliary quantities occurs on a line whose first field contains the character A. Finally, the type SQUARE, which uses an transformation from elemental to internal variables $u_1 = v_1 + v_2$, is defined on lines 34 to 39. Again notice that the quantity $2p_1$ occurs in both first and second derivatives, so the auxiliary quantity TWOP1 is set on line 36 to hold this value.

3.2 Data cards

The ELEMENTS and ENDATA indicator cards perform the same function as the cards NAME and ENDATA in §2.2.1 and 2.2.2 The problem name specified in field 3 on the ELEMENTS card must be the same as that given in the same field on the NAME card of the SDIF file.

3.2.1 The TEMPORARIES data cards

When specifying the function and derivative values of a nonlinear element, it often happens that an expression occurs more than once. It is then convenient to define an auxiliary parameter to have the value of the common expression and henceforth to refer to the auxiliary parameter. For instance, a nonlinear element of the two internal variables u_1 and u_2 might be $u_1 e^{u_2}$. (The names of the internal variables have already been specified in the ELEMENT TYPE section of the SDIF and are known as *reserved* parameters.) Its gradient vector (vector of first partial derivatives) has components e^{u_2} and $u_1 e^{u_2}$. If we define the auxiliary parameter $w = e^{u_2}$, the derivatives are then w and $u_1 w$.

↔ ← 6 →
F.1 Field 2

TEMPORARIES		
I	p_name	
R	p_name	
M	p_name	
F	p_name	

↑↑ ↑ ↑
23 5 10

Figure 3.2.1. Possible data cards for TEMPORARIES

The TEMPORARIES indicator card is used to announce the names of any auxiliary parameters which are to be used in defining the function and derivative values of the nonlinear elements. This list should also include the name of any intrinsic and external functions used. The syntax for data cards following the indicator card is given in Figure 3.2.1.

The single character string in field 1 specifies the type of auxiliary parameter that is to be defined. Possible types are integer (I), real (R), intrinsic function (M) or external function (F). The string, p_name, in field 2 then gives the name of the auxiliary parameter. The name can be up to six characters long, must be a valid name for the programming language under consideration, but must not be a reserved one, i.e., one of the names assigned to the internal variables for the element in question in the ELEMENT TYPES section of the SDIF (see, §2.2.13). Any auxiliary parameter that is to be used must be defined in the TEMPORARIES section along with all intrinsic and external function names.

3.2.2 The GLOBALS data cards

The GLOBALS indicator card is used to announce the assignment of general parameter values. The syntax for data cards following the indicator card is given in Figure 3.2.2.

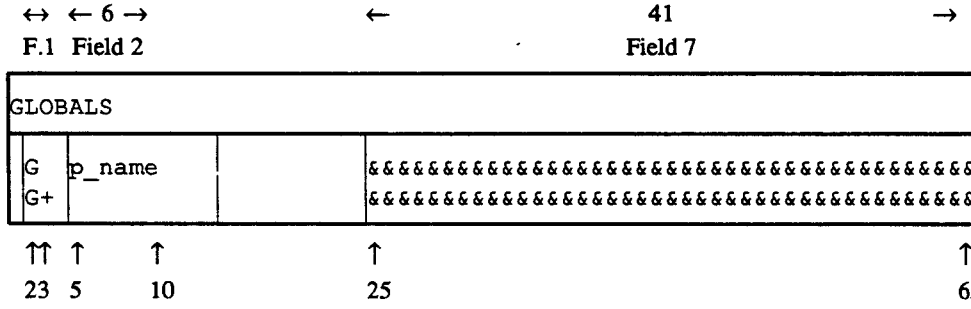


Figure 3.2.2. Possible data cards for GLOBALS

The string in field 1 specifies whether a new assignment is to be made (G) or whether the assignment is the continuation of an existing one (G+). If the assignment is new, the string, p_name, in field 2 gives the name of the auxiliary parameter that is to be defined; this name can be up to six characters long and must have been previously defined in the TEMPORARIES section. If the assignment is a continuation, the continuation refers to the auxiliary variable last named on a P card. At most nineteen continuations of a single assignment are allowed. The string in field 7 is an arithmetic expression. The assignment

auxiliary variable last named in field 2 ← field 7

is made, where ← means "is given the value"; any variable mentioned in the arithmetic expression must either be reserved (see §3.2.1), or have been defined in the TEMPORARIES section and, in the case of an integer or real variable, allocated a value itself on a previous GLOBALS data card.

The GLOBALS section is intended for the definition of auxiliary variables which occur in more than one element type. If an auxiliary variable occurs in a single element type, it may be defined in the INDIVIDUALS section (see §3.2.3).

3.2.3 The INDIVIDUALS data cards

The INDIVIDUALS indicator card is used to announce the definition of function and derivative values and the transformation between elemental and internal variables for the types of nonlinear element functions required. The syntax for data cards following the indicator card is given in Figure 3.2.3.

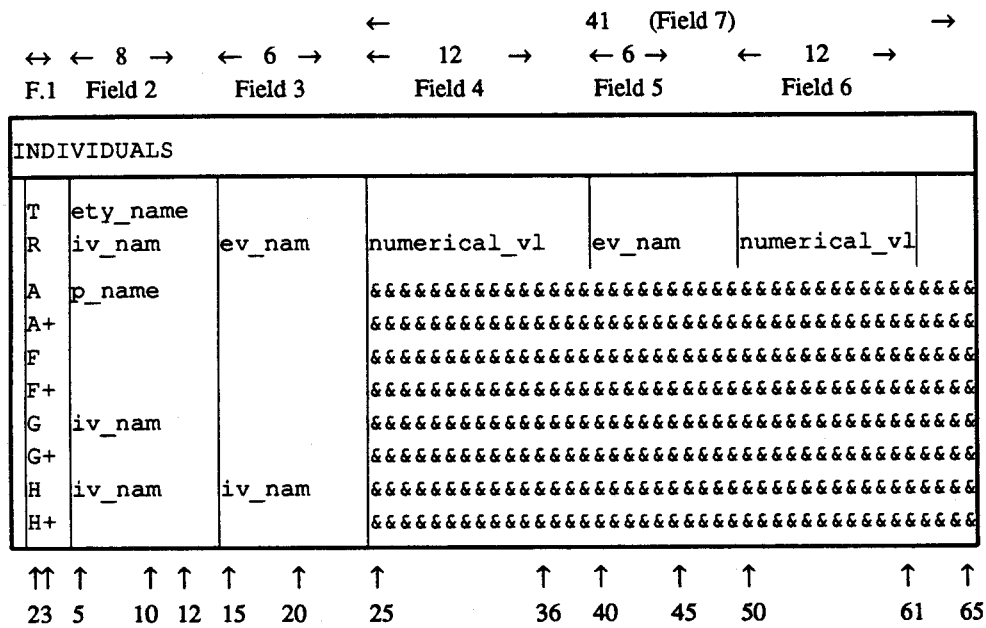


Figure 3.2.3. Possible data cards for INDIVIDUALS

The one or two character string in field 1 specifies the type of data contained on the card. Possible values for the first character of the string are:

T This card announces that a new element type is to be considered. The string, ety_name, in field 2 gives the name of the element type; the name may be up to eight characters long and must have been defined in the ELEMENT TYPE section of the SDIF file (see §2.2.13).

R This card announces that information concerning the transformation between the elemental and internal variables for the element type is to be given. Such information is appropriate only for element types which have been defined with internal variables in the ELEMENT TYPE section of the SDIF file (see §2.2.13). The transformation is specified by the matrix W of §1.3; only nonzero coefficients of W need be specified here.

The string, inv_name, in field 2 contains the name of an internal variable (i.e., row of W). The name may be up to six characters long and must have been defined on an IV data line in the ELEMENT TYPE section of the SDIF file. The strings, iv_nam, in fields 3 and (optionally) 5 then give the names of elemental variables (i.e., columns of W). The names may be up to six characters long and must have been defined on EV data lines in the ELEMENT TYPE section of the SDIF file. The strings in fields 4 and (optionally) 6 contain the numerical values of the coefficients of W corresponding to the row given in field 2 and the columns given in fields 3 and 5 respectively. These numerical values may each be up to 12 characters long. The entries of W may be defined in any order.

As an example, the transformation (1.2.5) could be entered with three R data cards. On the first, field 2 would hold the name given to the internal variable u_1; field 3 would hold the name given to the elemental variable v_1 and field 4 would contain 1.0. Similarly field 5 would hold the name given to the elemental variable v_2 and field 6 would also contain 1.0. On the second, field 2 would also hold the name given to the internal variable u_1; field 3 would now hold the name given to the elemental variable v_3 and field 4 would contain –2.0. On the third card, field 2 would hold the name

given to the internal variable u_2; field 3 would hold the name given to the elemental variable v_1 and field 4 would contain 1.0. Field 5 would now hold the name given to the elemental variable v_3 and field 6 would contain −1.0.

A This card announces that an auxiliary parameter, specific to the current element type, is to be assigned a value. The string, `p_name`, in field 2 gives the name of the auxiliary parameter that is to be defined; this name can be up to six characters long and must have been previously defined in the `TEMPORARIES` section. The string in field 7 is an arithmetic expression. The assignment

auxiliary variable named in field 2 ← field 7

is made, where again ← means "is given the value"; any variable mentioned in the arithmetic expression must either be reserved (see §3.2.1), or have been defined in the `TEMPORARIES` section. If in this latter case, the variable is integer or real, it must have been allocated a value itself either on a previous `GLOBALS` data card or on a previous `P` card for the current element type in the `ELEMENTS` section.

F This card specifies the value of the nonlinear element. The string in field 7 is an arithmetic expression; the assignment

nonlinear element function ← field 7

is made; any variable mentioned in the expression must obey the rules set out in the `A` section above.

G This card specifies the value of a component of the gradient of the nonlinear element. The string, `iv_nam`, in field 2 contains the name of an internal variable. The component of the gradient specified on the card will be taken with respect to this variable. The string can be up to six characters long and must have been defined on an `IV` data line in the `ELEMENT TYPE` section of the SDIF file. The string in field 7 is an arithmetic expression; the assignment

derivative of element w.r.t. variable in field 2 ← field 7

is made; any variable mentioned in the arithmetic expression must obey the rules set out in the `A` section above. Once the user starts to form the gradient for an element type, any component not explicitly specified will be assumed to have the value zero.

H This card specifies the value of a component of the Hessian matrix of the nonlinear element. The strings, `iv_nam`, in field 2 and 3 contain the names of internal variables. The component of the Hessian specified on the card will be taken with respect to these variables. The string can be up to six characters long and must have been defined on an `IV` data line in the `ELEMENT TYPE` section of the SDIF file. The string in field 7 is an arithmetic expression; the assignment

second derivative of element w.r.t. variables in fields 2 and 3 ← field 7

is made; any variable mentioned in the arithmetic expression must obey the rules set out in the `A` section above. `H` cards are optional. However, once the user starts to specify the Hessian matrix for an element type, any component not specified will be assumed to have the value zero. The matrix is assumed to be symmetric and so the user needs only supply values for one of

$$\frac{\partial^2 f}{\partial u_i \partial u_j} \text{ or } \frac{\partial^2 f}{\partial u_j \partial u_i} \quad (i \neq j);$$

it does not matter which. Observe that defaulting Hessian components to zero gives a very simple way of inputing sparse matrices; however, as we stressed in the introduction, we do not recommend this method of specifying invariant subspaces.

The data started on a A, F, G and H card may be continued on a card whose first field contains a A+, F+, G+ or H+ respectively. Such cards contain an arithmetic expression in field 7 and no further data; the arithmetic expression must obey the rules set out in the A section above. At most nineteen continuations of a single assignment are allowed. card

The data for a single element type must occur on consecutive cards and in the order given in Figure 3.2.3. A new element type is deemed to have started whenever a T card is encountered. The F card is compulsory for all element types; elements with useful transformations from elemental to internal variables must also have R cards. The data for a particular card type is considered to have been completed whenever another card type is encountered.

3.3 Two further examples

In §1.4, we gave an example. An SEIF file for this example is given in Figure 3.3.1. The problem is again given the name DOC. The two types of nonlinear element were assigned the names ELEMENT1/2 by the previous SDIF file. The elemental variables were given names beginning with V and the internal variables for the second nonlinear element started with U. The constant 0.0 occurs in the derivatives of both elements, so an auxiliary variable is assigned to hold its value. The function value and derivatives of the second element type use both sines and cosines of u_2 and again auxiliary variables are assigned to hold these values, this time as variables local to ELEMENT2. The second derivatives are sufficiently straightforward to compute that we provide them.

We gave a second example in §1.5. An SEIF file for this example is given in Figure 3.3.2. The problem is again given the name DOC2. The only type of nonlinear element was assigned the name SQUARE in the previous SDIF file, its elemental variable was called V and there was no useful range transformation.

4. The standard data input format for describing nontrivial groups

In addition to the problem data and the nonlinear element types described in §2 and 3, the user might also wish to specify the nontrivial group functions, and their derivatives, in a systematic way. A particular nontrivial group function is defined in terms of its group type and variable; both of these quantities are specified in §2. Thus, the only details which remain to be specified are the function and derivative values of the group types.

Once again, we present an approach to this issue. As before, data is specified in a file. The file comprises an ordered mixture of indicator and data cards; the latter allow function and derivative definitions in appropriate high-level language statements.

4.1 Introduction to the standard group type input format

4.1.1 The values and derivatives required

It is assumed that a nonlinear group type is specified in terms of its group-type variable as described on a `GROUP TYPE` data card in an SDIF file, see §2.2.13. An optimization procedure is likely to require the values of the group functions and their first and second derivatives (taken with respect to the variable). We now describe how to set up the data for a given problem.

4.1.2 Indicator cards

As before, the user must prepare an input file, the SGIF (Standard Group type Input Format) file, consisting of indicator and data cards. The former contain a simple keyword to specify the type of data that follows. Possible indicator cards are given in Figure 4.1.

keyword	comments	presence	described in §
`GROUPS`	same as `NAME`	mandatory	2.2.1
`TEMPORARIES`		optional	3.2.1
`GLOBALS`		optional	3.2.2
`INDIVIDUALS`		optional	4.2.1
`ENDATA`		mandatory	2.2.2

Figure 4.1. Possible indicator cards

Indicator cards must appear in the order shown. The cards `TEMPORARIES`, `GLOBALS` and `INDIVIDUALS` are optional.

The data cards are of a single kind, using four fields, fields 1, 2, 3 and 7, exactly as described in §3.1.2.

4.1.3 An example.

Before we give the complete syntax for an SGIF file, we finish the illustrative example that we started in §2.1.3 and §3.1.3 and show how to specify an input file appropriate for the problem of §1.6. The format is fairly similar to that for the SEIF file of §3. Once again, there are many possible ways of specifying a particular problem; we give one in Figure 4.1.3.

The file must always start with a `GROUPS` card, on which a name (in this case `EG3`) for the example may be given (line 1), and must end with an `ENDATA` card (line 10).

We next need to specify the names and attributes of any auxiliary quantities and functions that we intend to use in our high level description of the group functions. These are needed to allow for

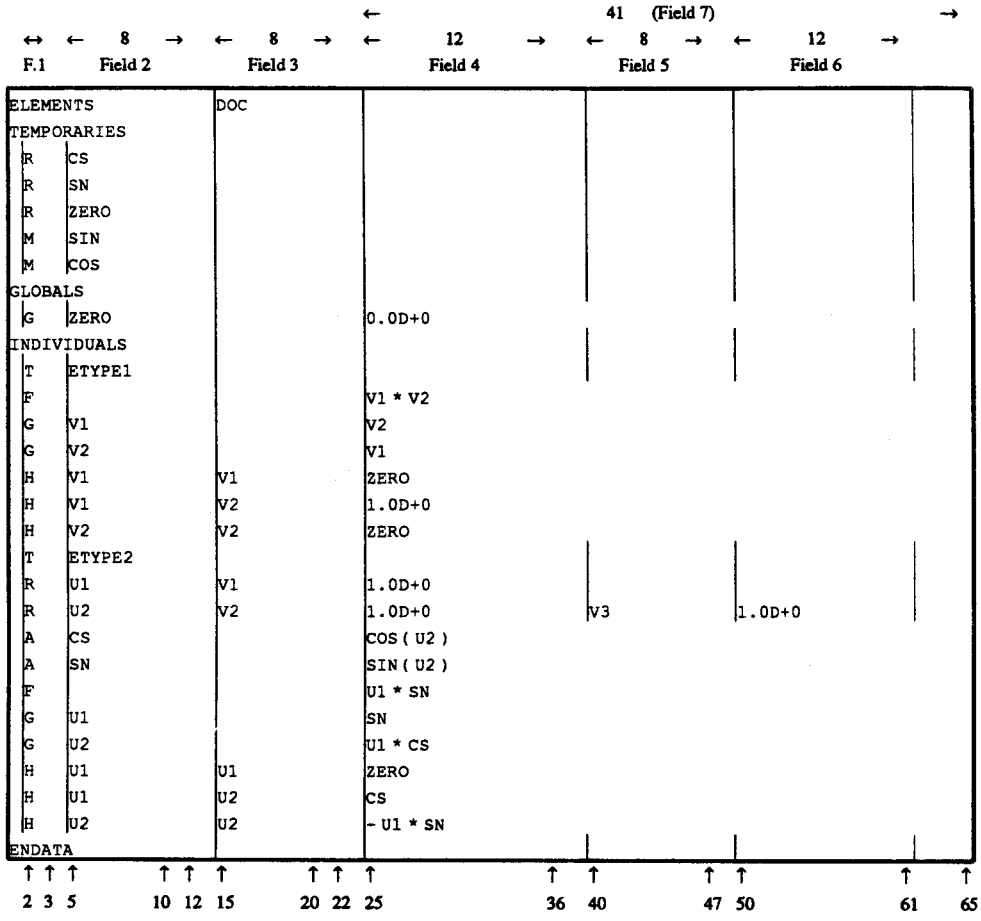


Figure 3.3.1. SEIF file for the element types for the example of §1.4

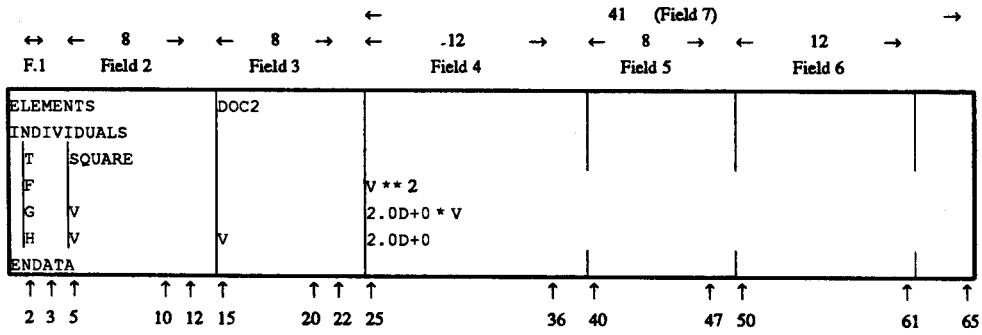


Figure 3.3.2. SEIF file for the element type for the example of §1.5

consistency checks in the proceeding high-level language statements and must always occur in the TEMPORARIES section of the input file. Line 3 indicates that we shall be using temporary quantities TWOP1 and the character R in the first field of this lines states that the quantity will be associated with a floating point (real) value.

We now make the actual definitions of the function and derivative values for the nontrivial group type

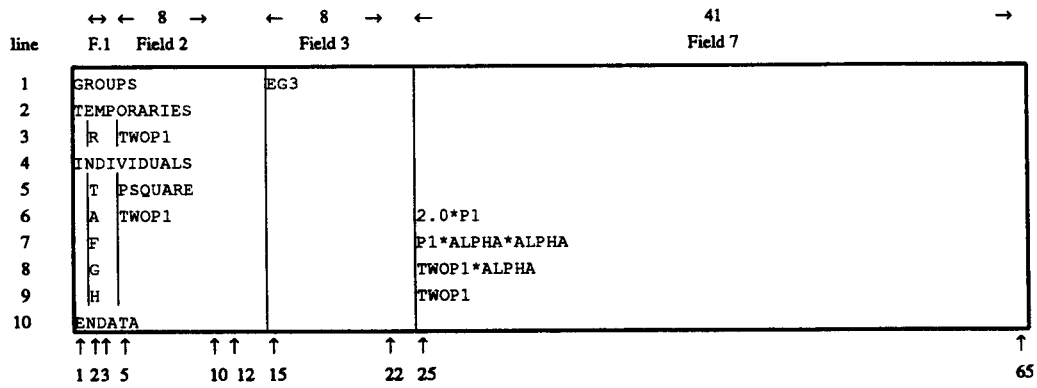


Figure 4.1.3. SGIF file for the nontrivial group type for the example of §1.6

used; we recall that there is a single nontrivial group type PSQUARE and that its attributes (name of group-type variable and parameter) has been described in the SDIF file set up in §2.1.3. This definition takes place within the INDIVIDUALS section. The presence of the character T in field 1 of line 5 announces that the data for the group type PSQUARE is to follow. All the data for this group must be specified before another group type is considered. We note that the quantity $2p_1$ occurs in both first and second derivatives of the group type function and so the auxiliary quantity TWOP1 is set on line 6 to hold this value. The first field of a line on which such an assignment is made contains the character A. The value (line 7), its first derivative (line 8) and second derivative (line 9) with respect to the group-type variable are now given. A Fortran expression for these values occurs in field 7 on each of these line; the lines contain the characters F, G and H respectively in field 1 for such assignments.

If there had been more than a single group type with one or more expressions in common, these expressions could have been assigned to previously attributed quantities in a GLOBALS section. This section would then have appeared between the TEMPORARIES and INDIVIDUALS sections.

4.2 Data cards

The GROUPS and ENDATA indicator cards perform the same function as the cards NAME and ENDATA in §2.2.1 and 2.2.2 Likewise, the TEMPORARIES and GLOBALS data cards have exactly the same syntax as those in §3.2.1 and 3.2.2, excepting that the reserved parameters are now the group-type variables specified in the GROUP TYPE section of the SDIF file.

4.2.1 The INDIVIDUALS data cards

The INDIVIDUALS indicator card is used to announce the definition of function and derivative values for the types of nontrivial group functions required. The syntax for data cards following the indicator card is given in Figure 4.2.1.

The one or two character string in field 1 specifies the type of data contained on the card. Possible values for the first character of the string are:

T This card announces that a new group type is to be considered. The string, gty name, in field 2 gives the name of the group type; the name may be up to eight characters long and must have been defined in the GROUP TYPE section of the SDIF file (see §2.2.15).

A This card announces that an auxiliary parameter, specific to the current group type, is to be assigned a value. The string, p_name, in field 2 gives the name of the auxiliary parameter that is to be

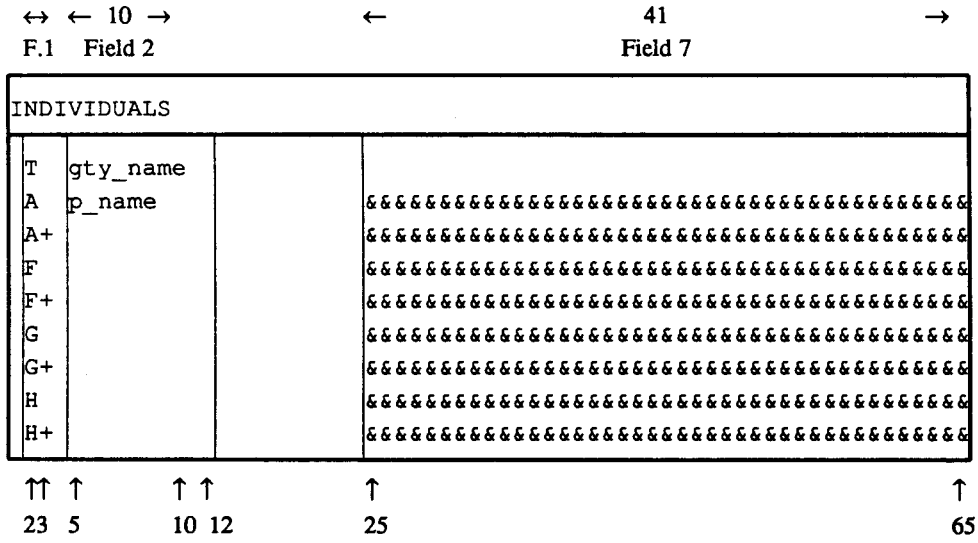


Figure 4.2.1. Possible data cards for INDIVIDUALS

defined; this name can be up to six characters long and must have been previously defined in the TEMPORARIES section. The string in field 7 is an arithmetic expression. The assignment

auxiliary variable named in field 2 ← field 7

is made; any variable mentioned in the arithmetic expression must either be reserved (see §3.2.1), or have been defined in the TEMPORARIES section. If, in this latter case, the variable is integer or real, it must have been allocated a value itself either on a previous GLOBALS data card or on a previous P card for the current element type in the INDIVIDUALS section.

F This card specifies the value of the nontrivial group. The string in field 7 is an arithmetic expression; the assignment

nontrivial group function ← field 7

is made; any variable mentioned in the expression must obey the rules set out in the A section above.

G This card specifies the value of the first derivative of the nonlinear group function with respect to its group-type variable. The string in field 7 is an arithmetic expression; the assignment

first derivative of group function ← field 7

is made; any variable mentioned in the arithmetic expression must obey the rules set out in the A section above.

H This card specifies the value of the second derivative of the the nonlinear group function with respect to its group-type variable. The string in field 7 is an arithmetic expression; the assignment

second derivative of group function ← field 7

is made; any variable mentioned in the arithmetic expression must obey the rules set out in the A section above.

The data started on a A, F, G and H card may be continued on a card whose first field contains a A+, F+, G+ or H+ respectively. Such cards contain an arithmetic expression in field 7 and no further data; the arithmetic expression must obey the rules set out in the A section above. At most nineteen continuations of a single assignment are allowed.

The data for a single group type must occur on consecutive cards and in the order given in Figure 4.2.1. A new group type is deemed to have started whenever a T card is encountered. The F card is compulsory for all group types.

4.3 Two further examples

In §1.4, we gave an example. An SGIF file for this example is given in Figure 4.3.1.

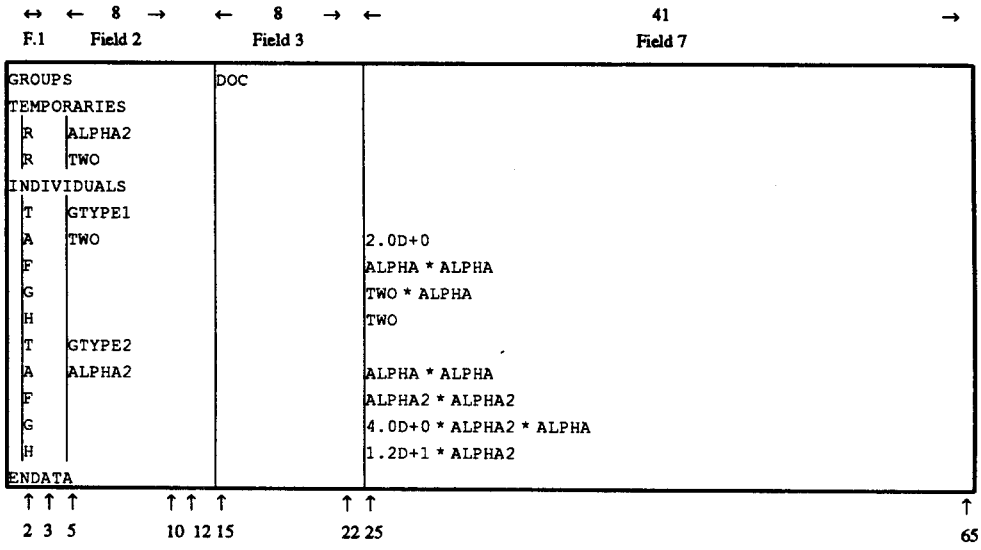


Figure 4.3.1. SGIF file for the nontrivial group types for the example of §1.4

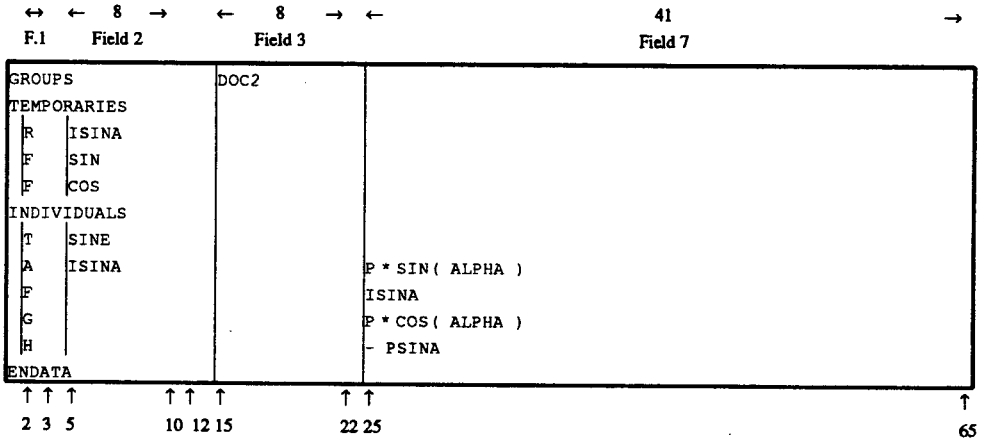


Figure 4.3.2. SGIF file for the nontrivial group type for the example of §1.5

The problem is again given the name DOC. The two types of nontrivial groups were assigned the names GTYPE1/2 by the previous SDIF file, each with group-type variables ALPHA. The function and derivatives values of the second group type, $g(\alpha)=\alpha^4$ all use some product of α^2, so an auxiliary variable is assigned to hold this value, the variable being local to the group type. Likewise, the derivatives of the first group type, $g(\alpha)=\alpha^2$ both use some product of 2.0, so another auxiliary variable is assigned to hold its value.

We gave a second example in §1.4. An SGIF file for this example is given in Figure 4.3.2. The problem is again given the name `DOC2`. The single nontrivial group type was given the name `SINE` by the previous SDIF file, with the group-type variable `ALPHA` and the single parameter `P`. The function and second derivatives both depend on the product of the parameter with the sine of the group type variable, so an auxiliary variable is assigned to hold this value.

5. Other standards and proposals

There have been a number of other proposed standards for input. The most popular approaches use a high level modelling language to specify problems. Typical examples are GAMS (Brooke, Kendrick and Meeraus, 1988), AMPL (Fourer, Gay and Kernighan, 1987) and OMP (Decker, Louveaux, Mortier, Schepens and Looveren, 1987). Such approaches are useful for specifying repetitious structures, but do not really attempt to cope with useful nonlinear structure (like invariant subspaces). Recent work by Fourer, Gay and Kernighan (1989) hopes to overcome this disadvantage.

We have recently become aware of other suggestions for the input of large-scale structured problems. These proposals are based upon representing nonlinear functions in their factorable (Lenard, 1989) or functional forms (McCormick and Rahnavard, 1989). Such forms are the the logical extensions of (1.1.1) in which a function is decomposed completely into basic building blocks. The advantage of such schemes is the potential for the automatic calculation of derivatives, but this must be weighed against the difficulty of describing how the building blocks are assembled. We await further details of these interesting proposals

6. Conclusions

We have made a proposal for a standard input format for the specification of (large-scale) nonlinear programming problems. In its full generality, the user needs to provide three input files. The first describes the structure of the problem and the decomposition of the problem into group and element functions. The second and third then specify the values and derivatives of these functions. It is anticipated that the first file will be used to provide input parameters for a user's optimization procedure, while the remaining two will be used to generate problem evaluation subprograms.

Such an approach has already been implemented at Harwell as one way of presenting problem data to the authors' large-scale nonlinear programming package (SBMIN/LANCELOT, in preparation, see Conn, Gould and Toint, 1988a,b,1989, Conn, Gould, Lescrenier and Toint, 1987). Indeed, as the input parameter list for our optimization procedure is rather long, we regard the approach given here as the most reliable method of setting up a particular problem.

7. References

A. Brooke, D. Kendrick and A. Meeraus *GAMS: A user's guide*, Scientific Press, 1988.

A.G. Buckley, "Test functions for unconstrained minimization", Technical Report CS-3, Computing Science Division, Dalhousie University, 1989.

A.R. Conn, N.I.M. Gould, M. Lescrenier and Ph.L. Toint, "Performance of a multifrontal scheme for partially separable optimization", Technical Report CS-88-04, Department of Computer Science, University of Waterloo, Waterloo, Ontario, CANADA, 1988.

A.R. Conn, N.I.M Gould and Ph.L. Toint, "Global convergence of a class of trust region algorithms for optimization with simple bounds", *SIAM Journal on Numerical Analysis* **25** 433-460, 1988a.

A.R. Conn, N.I.M. Gould and Ph.L. Toint, "Testing a class of methods for solving minimization problems with simple bounds on the variables", *Mathematics of Computation* **50** 399-430, 1988b.

A.R. Conn, N.I.M. Gould and Ph.L. Toint, "A Globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds", *SIAM Journal on Numerical Analaysis* (to appear) 1990.

G.B. Dantzig, *Linear programming and extensions*, Princetown University Press, 1963.

B.D. Decker, F. Louveaux, C. Mortier, G. Schepens and A.V. Looveren, *Linear and mixed nteger programming with OMP*, Beyers and Partners, 1987.

I.S. Duff, A.M. Erisman and J.K. Reid, *Direct methods for Sparse matrices*, Oxford University Press, 1986.

I.S. Duff, R.G. Grimes and J.G. Lewis, "Sparse matrix test problems", *ACM Transactions on mathematical software* **15** 1-14, 1989.

R. Fourer, D.M. Gay and B.W. Kernighan "AMPL: a mathematical programming language", Computer science technical report 133, AT&T Bell Laboratories, 1987.

R. Fourer, D.M. Gay and B.W. Kernighan, "A high-level language would make a good standard form for nonlinear programming problems", talk at the CORS/TIMS/ORSA meeting, Vancouver, 1989.

D.M. Gay, "Electronic mail distribution of linear programming test problems", Mathematical Programming Society COAL Newsletter, December, 1985.

A. Griewank and Ph.L. Toint, "Partitioned Variable Metric Updates for Large Structured Optimization Problems", *Numerische Mathematik* **39** 119-137, 1982.

W. Hock and K. Schittkowski, *Test examples for nonlinear programming codes*, Lecture notes in economics and mathematical systems, volume 187, Springer-Verlag, Berlin, 1981.

M. Lenard "Standardizing the interface with nonlinear optimizers", talk at the CORS/TIMS/ORSA meeting, Vancouver, 1989.

G. McCormick and P. Rahnavard "Representation of unconstrained optimization", talk at the CORS/TIMS/ORSA meeeting, Vancouver, 1989.

J.J. Moré, "A collection of nonlinear model problems", Preprint MCS-P60-0289, Argonne National Laboratory, 1989.

J.J. Moré, B.S. Garbow and K.E. Hillstrom, "Testing unconstrained optimization software", *ACM Transactions on mathematical software* **7** 17-41, 1981.

K. Schittkowski, *More test examples for nonlinear programming codes*, Lecture notes in economics and mathematical systems, volume 282, Springer-Verglag, Berlin, 1987.

8. An appendix of further examples.

Although it has been necessary to give an exact specification of the allowed input formats, we feel that the best way to learn how to input problems is by studying specific examples. Consequently, we present here five further examples which use many of the salient features of the standard. We hope that readers will be encouraged to trying converting their favourite test examples into the standard form.

A collection of SDIF files for many sets of standard test problems is currently being compiled. To date, we have translated the problems given by Moré, Garbow and Hilstrom (1981), Moré (1989) and Buckley (1989). Work is proceeding on the problems given in the books by Hock and Schittkowski (1981) and Schittkowski (1987). It is our intention to put the resulting files in the public domain.

8.1 A simple unconstrained problem

Firstly, we give an SDIF file for the famous Rosenbrock function in 2 variables (Moré, Garbow and Hillstrom, 1981, page 27, problem 28). Here the objective function is

$$100(x_2 - x_1^2)^2 + (1 - x_1)^2.$$

There are no constraints and the given starting value for an unconstrained minimization is $x_1 = -1.2$, $x_2 = 1$. An appropriate SDIF file might be: .

```
NAME          ROSNB2
VARIABLES
    X1
    X2
GROUPS
 N  G1        X2        1.0
 N  G1        'SCALE'   0.01
 N  G2        X1        1.0
CONSTANTS
    ROSNB     G2        1.0
BOUNDS
 DR ROSNB
START POINT
    ROSNB     X1        -1.2          X2        1.0
ELEMENT TYPE
 EV SQ        V1
ELEMENT USES
 T  E1        SQ
 V  E1        V1                      X1
GROUP TYPE
 GV L2        GVAR
GROUP USES
 T  G1        L2
 E  G1        E1
 T  G2        L2
OBJECT BOUND
*   Least square problems are bounded below by zero
 LO ROSNB              0.0
```

```
ENDATA

*   Specify the form of the different element types

ELEMENTS        ROSNB2

INDIVIDUALS

 T  SQ

 F                          - V1 * V1
 G  V1                      - V1 - V1
 H  V1          V1          - 2.0

ENDATA

*  Specify the form of the different group types

GROUPS          ROSNB2

INDIVIDUALS

 T  L2

 F                          GVAR * GVAR
 G                          GVAR + GVAR
 H                          2.0

ENDATA
```

8.2 A simple constrained problem

Next, we give an SDIF file for the 65th constrained example collected by Hock and Schittkowski (1981, page 87). The objective function is

$$(x_1 - x_2)^2 + (x_1 + x_2 - 10)^2 / 9 + (x_3 - 5)^2.$$

There is a single inequality constraint

$$x_1^2 + x_2^2 + x_3^3 \leq 48$$

and additional simple bounds

$$-4.5 \leq x_1 \leq 4.5, \; -4.5 \leq x_2 \leq 4.5, \; -5 \leq x_3 \leq 5$$

on the variables. The values $x_1 = -5$, $x_2 = 5$, $x_3 = 0$ give the starting point for the minimization. Noting the use of range transformations, an appropriate SDIF file might be:

```
NAME            HS65

*  Number of variables

 ID N                       3

*  Useful parameters

 ID 1                       1
 ID 3                       3
 RD NINE                    9.0
 RR 1/9         NINE

VARIABLES

    X1
    X2
    X3

GROUPS

*  Objective function

 N  OBJ

*   Constraints functions

 L  CON
```

```
CONSTANTS

    HS65      CON       48.0

BOUNDS

 LO HS65      X1        -4.5
 UP HS65      X1        4.5

 LO HS65      X2        -4.5
 UP HS65      X2        4.5

 LO HS65      X3        -5.0
 UP HS65      X3        5.0

START POINT

    HS65      X1        -5.0          X2        5.0
    HS65      X3        0.0

ELEMENT TYPE

 EV DIFSQR    V1                      V2
 IV DIFSQR    U

 EV SUMSQR    V1                      V2
 IV SUMSQR    U

 EV SQ-5      V
 EV SQ        V

ELEMENT USES

*   Nonlinear elements for the objective function

 T  O1        DIFSQR
 V  O1        V1                      X1
 V  O1        V2                      X2

 T  O2        SUMSQR
 V  O2        V1                      X1
 V  O2        V2                      X2

 T  O3        SQ-5
 V  O3        V                       X3

*   Nonlinear elements for the constraint function

 DO I         1                       3

 XT C(I)      SQ
 XV C(I)      V                       X(I)

 ND

GROUP USES

 E  OBJ       O1                      O3
 ZE OBJ       O2                      1/9
 E  CON       C1                      C2
 E  CON       C3

ENDATA

*   Specify the form of the different element types

ELEMENTS      HS65

TEMPORARIES

 R  DIF

INDIVIDUALS

*  square of V, where V = U1 - U2

 T  DIFSQR

 R  U         V1        1.0           V2        -1.0
 F                      U * U
 G  U                   U + U
 H  U         U         2.0

*  square of ( U - 10 ), where U = V1 + V2

 T  SUMSQR

 R  U         V1        1.0           V2        1.0
```

```
 A  DIF                  U - 10
 F                       DIF * DIF
 G  U                    DIF + DIF
 H  U          U         2.0

*  square of ( V - 5 )

 T  SQ-5

 A  DIF                  V - 5
 F                       DIF * DIF
 G  V                    DIF + DIF
 H  V          V         2.0

*  square of V

 T  SQ

 F                       V * V
 G  V                    V + V
 H  V          V         2.0

ENDATA

*  Specify the form of the different group types

GROUPS         HS65

*  All groups are trivial

ENDATA
```

8.3 A system of nonlinear equations with single-indexed variables

Here we give an SDIF file for the discrete boundary value problem in n variables given by Moré, Garbow and Hillstrom (1981, page 27, problem 28). We set the problem up as a system of nonlinear equations (constraints)

$$2x_i - x_{i-1} - x_{i+1} + h^2(x_i + ih + 1)^3/2 = 0, \quad (i = 1, \ldots, n),$$

where $h = 1/(n+1)$ and $x_0 = x_{n+1} = 0$. We specify $n = 8$ and use the starting point $x_i = ih(ih - 1)$ for $i = 1, \ldots, n$. An appropriate SDIF file might be:

```
NAME           BDVLE10

*   N is the number of internal discretization points

 ID N                    8

*   Define useful parameters

 ID 0                    0
 ID 1                    1
 IA N+1         N        1

 RI RN+1        N+1
 RR H           RN+1
 RP H2          H                       H
 RM HALFH2      H2       0.5

VARIABLES

 DO I           0                       N+1

 X  X(I)

 ND

GROUPS

 DO I           1                       N
 IA I-1         I        -1
 IA I+1         I         1

 XE G(I)        X(I-1)   -1.0            X(I)       2.0
 XE G(I)        X(I+1)   -1.0

 ND
```

```
BOUNDS

 DR BDVLE
 XX BDVLE      X(0)       0.0
 XX BDVLE      X(N+1)     0.0

START POINT

 X  BDVLE      X(0)       0.0            X(N+1)     0.0

 DO I          1                         N

 RI RI         I
 RP IH         RI                        H
 RA IH-1       IH         -1.0
 RP TI         IH                        IH-1

 Z  BDVLE      X(I)                      TI

 ND

ELEMENT TYPE

 EV WCUBE      V
 EP WCUBE      B

ELEMENT USES

 DO I          1                         N

 RI REALI      I
 RP IH         REALI                     H

 XT E(I)       WCUBE
 XV E(I)       V                         X(I)
 ZP E(I)       B                         IH

 ND

GROUP USES

 DO I          1                         N

 ZE G(I)       E(I)                      HALFH2

 ND

ENDATA

*   Specify the form of the different element types

ELEMENTS       BDVLE10

*   the cube of (V + B)

TEMPORARIES

 R  VPLUSB

INDIVIDUALS

 T  WCUBE
 A  VPLUSB                 V + B
 F                         VPLUSB**3
 G  V                      3.0 * VPLUSB**2
 H  V          V           6.0 * VPLUSB

ENDATA

*  Specify the form of the different group types

GROUPS         BDVLE10

*  all groups are trivial

ENDATA
```

8.4 An unconstrained problem with double indexed variables

Now, we give an SDIF file for the linear minimum surface problem formulated by Toint (see, Buckley, 1989, page 71). This example illustrates the use of two-dimensional arrays. The problem is described in the comments included in the following SDIF file:

```
NAME          LMSRF100

*   The problem comes from the discretization of the minimum surface
*   problem on the unit square: given a set of boundary conditions on
*   the four sides of the square, one must find the surface which
*   meets these boundary conditions and is of minimum area.

*   The unit square is discretized into (p-1)**2 little squares. The
*   heights of the considered surface above the corners of these little
*   squares are the problem variables,  There are p**2 of them.
*   Given these heights, the area above a little square is
*   approximated by the
*     S(i,j) = sqrt( 1 + 0.5(p-1)**2 ( a(i,j) + b(i,j) ) ) / (p-1)**2
*   where
*     a(i,j) = x(i,j) - x(i+1,j+1)
*   and
*     b(i,j) = x(i+1,j) - x(i,j+1)

*   In the Linear Minimum Surface, the boundary conditions are given
*   as the heights of a given plane above the square boundaries.  This
*   plane is specified by its height above the (0,0) point (H00 below),
*   and its slopes along the first and second coordinate
*   directions in the plane (these slopes are denoted SLOPEJ and SLOPEI

*   P is the number of points in one side of the unit square

 ID P                   10

*   Define the plane giving the boundary conditions

 RD H00                 1.0
 RD SLOPEJ              4.0
 RD SLOPEI              8.0

*   Define a few helpful parameters

 IS TWOP        P                       P
 IA P-1         P           -1
 IP PP-1        P                       P-1
 RI RP-1        P-1
 RR INVP-1      RP-1
 RP RP-1SQ      INVP-1                  INVP-1
 RR SCALE       RP-1SQ
 RP SQP-1       RP-1                    RP-1
 RM PARAM       SQP-1       0.5

 ID 1                       1
 ID 2                       2

 RP STON        INVP-1                  SLOPEI
 RP WTOE        INVP-1                  SLOPEJ
 RS H01         H00                     SLOPEJ
 RS H10         H00                     SLOPEI

VARIABLES

*   Define one variable per discretized point in the unit square

 DO J           1                       P
 DO I           1                       P

 X  X(I,J)

 ND

GROUPS

*   Define a group per little square

 DO I           1                       P-1
 DO J           1                       P-1

 ZN S(I,J)      'SCALE'                 SCALE

 ND

CONSTANTS
```

```
 D  LMSRF               -1.0

BOUNDS

 DR LMSRF

*   Fix the variables on the lower and upper edges of the unit square

 DO J          1                          P

 IA J-1        J          -1
 RI RJ-1       J-1
 RP TH         RJ-1                       WTOE
 RS TL         TH                         H00
 RS TU         TH                         H10

 ZX LMSRF      X(1,J)                     TL
 ZX LMSRF      X(P,J)                     TU

 ND

*   Fix the variables on the left and right edges of the unit square

 DO I          2                          P-1

 IA I-1        I          -1
 RI RI-1       I-1
 RP TV         RI-1                       STON
 RS TR         TV                         H00
 RS TL         TV                         H01

 ZX LMSRF      X(I,P)                     TL
 ZX LMSRF      X(I,1)                     TR

 ND

START POINT

*   All variables not on the boundary are set to 0.0

 D  LMSRF               0.0

*   Start from the boundary values on the lower and upper edges

 DO J          1                          P

 IA J-1        J          -1
 RI RJ-1       J-1
 RP TH         RJ-1                       WTOE
 RS TL         TH                         H00
 RS TU         TH                         H10

 Z  LMSRF      X(1,J)                     TL
 Z  LMSRF      X(P,J)                     TU

 ND

*   Start from the boundary values on the left and right edges

 DO I          2                          P-1

 IA I-1        I          -1
 RI RI-1       I-1
 RP TV         RI-1                       STON
 RS TR         TV                         H00
 RS TL         TV                         H01

 Z  LMSRF      X(I,P)                     TL
 Z  LMSRF      X(I,1)                     TR

 ND

ELEMENT TYPE

*   The only element type.

 EV ISQ        V1                         V2
 IV ISQ        U

ELEMENT USES

*   Each little square has two elements using diagonal and
*   antidiagonal corner values

 DO I          1                          P-1
 IA I+1        I          1
 DO J          1                          P-1
 IA J+1        J          1
```

```
 XT A(I,J)     ISQ
 XV A(I,J)     V1                        X(I,J)
 XV A(I,J)     V2                        X(I+1,J+1)

 XT B(I,J)     ISQ
 XV B(I,J)     V1                        X(I+1,J)
 XV B(I,J)     V2                        X(I,J+1)

 ND

GROUP TYPE

*   Groups are of the square root type

 GV SQROOT     ALPHA

GROUP USES

*   All groups are of SQROOT type.

 DO I          1                         P-1
 DO J          1                         P-1

 XT S(I,J)     SQROOT
 ZE S(I,J)     A(I,J)                    PARAM
 ZE S(I,J)     B(I,J)                    PARAM

 ND

OBJECT BOUND

 LO LMSRF                0.0

ENDATA

*  Specify the form of the different element types

ELEMENTS       LMSRF100

INDIVIDUALS

*   Difference squared

 T  ISQ
 R  U          V1        1.0             V2        -1.0
 F                       U * U
 G  U                    U + U
 H  U          U         2.0

ENDATA

*  Specify the form of the different group types

GROUPS         LMSRF100

TEMPORARIES

 M  SQRT
 R  SQRAL

INDIVIDUALS

*   square root groups

 T  SQROOT
 A  SQRAL                SQRT(ALPHA)
 F                       SQRAL
 G                       0.5D0 / SQRAL
 H                       -0.25D0 / ( SQRAL * ALPHA )

ENDATA
```

8.5 A constrained problem with triple indexed variables

Finally, we give an SDIF file for the problem of determining the maximum growth possible when performing Gaussian elimination with complete pivoting. Starting with an n by n real matrix $X^{(1)}$, we let $X^{(k)}$ be the matrix which remains to be eliminated after k–1 steps of Gaussian elimination without pivoting. Let $x_{i,j,k}$ be the (i,j)–th entry of $X^{(k)}$. We thus wish to maximize $x_{n,n,n}$ subject to the restrictions that the matrices $X^{(k)}$ and $X^{(k+1)}$ are related to each other by elimination restrictions, that the largest

element in the bottom $n-k-1$ block of $X^{(k)}$ occurs in position (k,k,k) and that the initial matrix $X^{(1)}$ is scaled so that the largest entry in magnitude is 1. This leads to the problem

$$\text{minimize} -x_{n,n,n}$$

subject to the elimination constraints

$$x_{i,j,k+1} - x_{i,j,k} + x_{i,k,k} x_{k,j,k} / x_{k,k,k} = 0 \quad \text{for } k < i \le n, k < j \le n \text{ and } k = 1, \ldots, n-1,$$

constraints which make the signs of the pivots unique,

$$x_{k,k,k} \ge 0 \quad \text{for } k = 1, \ldots, n,$$

a normalizing constraint, $x_{1,1,1} = 1$, and complete pivoting constraints

$$-1 \le x_{i,j,1} \le 1 \quad \text{for } 1 \le i \le n, 1 \le j \le n$$

and

$$-x_{k,k,k} \le x_{i,j,k} \le x_{k,k,k} \quad \text{for } k \le i \le n, k \le j \le n \text{ and } k = 2, \ldots, n-1.$$

These details, along with a suitable starting point, are given for the case $n=6$ in the following SDIF file:

```
NAME          GAUSS91
*   size of the matrices = n
 ID N                   6
*   other parameter definitions
 ID 1                   1
 ID 2                   2
 IA N-1        N        -1
VARIABLES
 DO K          1                       N
 DO J          K                       N
 DO I          K                       N
 X  X(I,J,K)
 ND
GROUPS
*  objective function
 DO K          N                       N
 XN OBJ        X(K,K,K)  -1.0
 ND
*   elimination constraints
 DO K          1                       N-1
 IA K+         K         1
 DO I          K+                      N
 DO J          K+                      N
 XE E(I,J,K)   X(I,J,K+) 1.0           X(I,J,K)  -1.0
 ND
*   complete pivoting constraints (submatrices 2 to n-1)
 DO K          2                       N-1
 DO I          K                       N
 DO J          K                       N
 XL M(I,J,K)   X(I,J,K)  1.0           X(K,K,K)  -1.0
 XG P(I,J,K)   X(I,J,K)  1.0           X(K,K,K)  1.0
 ND
```

```
BOUNDS

*   default = free variables

 DR GAUSS

*   complete pivoting constraints (submatrix 1)

 DO I          1                          N
 DO J          1                          N

 XL GAUSS      X(I,J,1)   -1.0
 XU GAUSS      X(I,J,1)   1.0

 ND

*   ensure pivotal elements are nonnegative

 DO K          1                          N

 XL GAUSS      X(K,K,K)   0.0

 ND

*   normalize first pivot

 FX GAUSS      X1,1,1     1.0

START POINT

*   default value for starting point component

 D  GAUSS                 0.01

*   Set initial matrices to perturbed identities

 DO K          1                          N
 DO I          K                          N

 X  GAUSS      X(I,I,K)   1.0

 ND

ELEMENT TYPE

 EV ELIM       V1                         V2
 EV ELIM       V3

ELEMENT USES

 DO K          1                          N-1

 IA K+         K          1

 DO I          K+                         N
 DO J          K+                         N

 XT A(I,J,K)   ELIM
 XV A(I,J,K)   V1                         X(I,K,K)
 XV A(I,J,K)   V2                         X(K,J,K)
 XV A(I,J,K)   V3                         X(K,K,K)

 ND

GROUP USES

 DO K          1                          N-1

 IA K+         K          1

 DO I          K+                         N
 DO J          K+                         N

 XE E(I,J,K)   A(I,J,K)

 ND

ENDATA

*  Specify the form of the different element types

ELEMENTS       GAUSS91

TEMPORARIES

 R  VALUE
 R  V3SQ
```

```
INDIVIDUALS

 T  ELIM

 A  VALUE                 V1 * V2 / V3
 A  V3SQ                  V3 * V3

 F                        VALUE

 G  V1                    V2 / V3
 G  V2                    V1 / V3
 G  V3                    - VALUE / V3

 H  V1          V2        1.0 / V3
 H  V1          V3        - V2 / V3SQ
 H  V2          V3        - V1 / V3SQ
 H  V3          V3        2.0 * VALUE / V3SQ * V3

ENDATA

*  Specify the form of the different group types

GROUPS         GAUSS91

* all groups are trivial

ENDATA
```