The SIF Reference Report (revised version)

Andrew R. Conn, Nicholas I. M. Gould,
Dominique Orban and Philippe L. Toint

March 16, 2003

Abstract

In this report, we provide a definitive description of the standard input format. It is intended to be used primarily as a reference document.

Contents

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 application 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.

The format described in this report was motivated directly by the difficulties the authors were experiencing entering test examples to the LANCELOT large-scale nonlinear optimization package. 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 here: a standard input format (SIF) 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 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 provision of a SIF 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, such 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 a SIF 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 [9, 6] is a good recommendation for such flexibility. The formality required by the SIF approach may admittedly appear formidable for very simple problems, but is soon repaid when dealing with more complex ones.

- Of course, the SIF 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 SIF 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 SIF 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 SIF for nonlinear programming problems should conform to the MPS format. We were thus led to choose 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 SIF must be based on fixed format for the SIF file. Indeed, blanks are significant characters in MPS, when they appear in the right data fields, and cannot be used as 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, a format which is not always ideally suited to nonlinear problems. Our present proposal accepts these limitations.

- The SIF 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 SIF interpreter using whatever facilities are locally available.

- In principle, the SIF should not be dependent on a particular high level programming language. However, as the intention is that SIF files may be converted into executable programs, restrictions on the symbolic names allowed by different programming languages may influence the choice of names within the problem description itself. For instance, in Fortran, symbolic variables may only contain up to six characters from a restricted set. We have chosen to base the present proposal, where necessary, on Fortran as this appears to be the most restrictive of the more popular high level languages. This dependence has been isolated as much as possible.

The authors are very well aware of the shortcomings of the SIF approach when compared to more elaborate modelling languages (see, for example, GAMS [1], AMPL [7], and OMP [4]. 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 SIF now. This (we hope, intermediate) step is indeed crucial for the development and comparison of algorithms for solving large scale nonlinear problems, without which a more elaborate tool would be meaningless anyway. The SIF 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 SIF 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.

In the next section, we explain how we propose to exploit the structure in problems of the form (2.1)–(2.4) . We do this both in general and for a number of examples. Details of the way such structure may be expressed in a standard data input format follow in Section 3. The input of nonlinear information for element and group functions is covered in Section 4 and Section 5 respectively. The formats proposed in Sections 3–5 are quite rigid. A more flexible, free-form, input is considered in Section 6. The relationship to existing work is presented in Section 7 and conclusions drawn in Section 7.

2 An introduction to nonlinear optimization problem structure

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 $f(x)$ is the set of all vectors w for which $f(x+w) = f(x)$ for all possible vectors x. This phenomenon encompasses function sparsity. For instance, the function

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

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. However the function

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

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 $F(x)$ 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 [11]. 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.

LANCELOT considers the problem of minimizing or maximizing an objective function of the form

$$F(x) = \sum_{i \in I_O} g_i \left(\sum_{j \in J_i} w_{i,j} f_j(\bar{x}_j) + a_i^T x - b_i \right) + \tfrac{1}{2} \sum_{j=1}^{n} \sum_{k=1}^{n} h_{j,k} x_j x_k, \tag{2.1}$$

where $x = (x_1, x_2, \cdots, x_n)$, within the "box" region

$$l_i \le x_i \le u_i, \quad l \le i \le n \tag{2.2}$$

(where either bound on each variable may be infinite), and where the variables are required to satisfy the extra conditions

$$g_i \left(\sum_{j \in J_i} w_{i,j} f_j(\bar{x}_j) + a_i^T x - b_i \right) = 0 \ (i \in I_E) \tag{2.3}$$

and

$$0 \left\{ \begin{matrix} \le \\ \ge \end{matrix} \right\} g_i \left(\sum_{j \in J_i} w_{i,j} f_j(\bar{x}_j) + a_i^T x - b_i \right) \left\{ \begin{matrix} \le \\ \ge \end{matrix} \right\} r_i, \ (i \in I_I) \tag{2.4}$$

for some index sets I_0, I_E and I_I and (possibly infinite) values r_i. The univariate functions g_i are known as *group functions*. The argument

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

is known as the i-th *group*. The functions $f_j, j = 1, \cdots, n_e$, are called *nonlinear* element functions. They are functions of the problem variables $\bar{x}_j$, where the $\bar{x}_j$ are either small subsets of x or such that f_j has a large invariant subspace for some other reason. The constants $w_{i,j}$ are known as *weights*, while the function $a_i^T x - b_i$ is known as the *linear* element for the i-th group. **New** The additional term $\frac{1}{2} \sum_{j=1}^{n} \sum_{k=1}^{n} h_{j,k} x_j x_k$ in the objective function is the *quadratic objective* group; the leading $\frac{1}{2}$ is there by convention.

It is more common to call the group functions in (2.3) equality constraint functions, those in (2.4) inequality constraint functions and the sum of those in (2.1) the objective function.

When stating a structured nonlinear optimization problem of the form (2.1)–(2.4), we need to specify the group functions, linear and nonlinear elements and the way that they all fit together.

2.1 Problem, Elemental and Internal Variables

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

$$(x_{29} + x_3 - 2x_{17})e^{x_{29}-x_{17}} \tag{2.5}$$

which has the structural form

$$f_1(v_1, v_2, v_3) = (v_1 + v_2 - 2v_3)e^{v_1-v_3}, \tag{2.6}$$

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

$$f_2(v_1, v_2) = v_1 \sin v_2, \tag{2.7}$$

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

$$f_3(v_1, v_2) = (v_1 + v_2)\log(v_1 - v_2), \tag{2.8}$$

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 in (2.6), we have

$$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & -2 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \tag{2.9}$$

while in (2.7), we have

$$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \tag{2.10}$$

In general the transformation will be of the form

$$u = Wv \tag{2.11}$$

and this transformation is *useful* if the matrix W has fewer rows than columns.

2.2 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

$$\sum_{i=1}^{n-1} (x_i x_{i+1})^i \tag{2.12}$$

naturally decomposes into the sum of $n-1$ group functions, $\alpha, \alpha^2, \cdots, \alpha^{n-1}$. Each group is a nonlinear element function $v_1 v_2$ 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

$$\sum_{i=1}^{n-1} (x_i x_{i+1} + x_1/x_i)^i \tag{2.13}$$

naturally decomposes into the sum of the same $n-1$ group functions. Each group is the sum of two nonlinear elements $v_1 v_2$ (where $v_1 = x_i$ and $v_2 = x_{i+1}$) and v_1/v_2 (where $v_1 = x_1$ and $v_2 = x_i$). 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

$$\sum_{i=1}^{n-1} (i x_i x_{i+1})^i \tag{2.14}$$

naturally decomposes into the sum of the same $n-1$ group functions. Each group is a nonlinear element $p_1 v_1 v_2$ 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 (2.12) and (2.14) use a single element type, where as (2.13) uses two types. When defining the data for problems of the form (2.1)–(2.4), 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

$$\sum_{i=1}^{n-1} (x_i x_{i+1})^2 \tag{2.15}$$

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

$$\sum_{i=1}^{n-1} i(x_i x_{i+1})^2 \tag{2.16}$$

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.

2.3 An Example

We now consider the small example problem,

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

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 (2.1). Here, we consider partitioning F into groups as

$$\underset{\substack{\uparrow \\ \text{group 1}}}{(x_1)^2} \quad + \quad \underset{\substack{\uparrow \\ \text{group 2}}}{(x_2 x_3)^4} \quad + \quad \underset{\substack{\uparrow \\ \text{group 3}}}{(x_2 \sin(x_1 + x_3) + x_1 x_3 + x_2)}$$

Notice the following:

1. 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.
2. group 2 uses the non-trivial group function $g_2(\alpha) = \alpha^4$. The group contains a single *nonlinear* element; this element function is $x_2 x_3$. 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.
3. 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 $x_1 x_3$, which has two elemental variables 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.

2.4 A Second Example

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

$$\text{minimize} \quad F(x_1, \cdots, x_{1000}) \equiv \sum_{i=1}^{999} \sin(x_i^2 + x_{1000}^2 + x_1 - 1) + \tfrac{1}{2} \sin(x_{1000}^2). \qquad (2.17)$$

Once again, there are a number of ways of casting this problem in the form (2.1), 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 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.

2.5 A Final Example

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

$$\text{minimize} \quad \tfrac{1}{2}((x_1 - x_{100})x_2 + y)^2 + 2x_1^2 + 2x_1x_{100} \tag{2.18}$$

subject to the constraints

$$x_1x_{i+1} + (1 + \tfrac{2}{i})x_ix_{100} + y \le 0 \quad (1 \le i \le 99), \tag{2.19}$$

$$0 \le (\sin x_i)^2 \le \tfrac{1}{2} \quad (1 \le i \le 100), \tag{2.20}$$

$$(x_1 + x_{100})^2 = 1 \tag{2.21}$$

and the simple bounds

$$-1 \le x_i \le i \quad (1 \le i \le 100). \tag{2.22}$$

As before, there are a number of ways of casting this problem in the form (2.1)–(2.4). We chose to decompose the problem as follows:

1. the objective function comprises two groups, the first of which uses the non-trivial group function $g(\alpha) = \frac{1}{2}\alpha^2$. This 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. **New** The second group may be considered as a quadratic objective group, and written as $\frac{1}{2}(h_{1,1}x_1x_1 + h_{1,100}x_1x_{100} + h_{100,1}x_{100}x_1)$, where $h_{1,1} = 4$ and $h_{1,100} = h_{100,1} = 2$.

2. The next set of groups, inequality constraints, $x_1x_{i+1} + (1 + 2/i)x_ix_{100} + y \le 0$ for $1 \le i \le 99$ are of the form (2.4) 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.

3. The following set of groups, again inequality constraints, $0 \leq (\sin x_i)^2 \leq \frac{1}{2}$ for $1 \leq i \leq 100$, are of the form (2.4) 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 $sinv_1$ 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.

4. The last group, an equality constraint, $(x_1 + x_{100})^2 - 1 = 0$, is of the form (2.3). 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 as well as an quadratic objective group 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 that it would be inefficient to pass the data group-by-group and element-by-element. Again, we will introduce ways to specify this repetitious structure using a convenient shorthand.

3 The Standard Data Input Format

We now consider how to pass the data for optimization problems to an optimization procedure. In our description, we will concentrate on our third example, as presented in Section 2.5; 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.

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 [2], we preserve the MPS terminology and call these lines *cards.*

A SIF comprises one or more files. The first of these files is known as the Standard *Data* Input Format (SDIF). As its name suggests, data which describes how the parts of the optimization problem are related, together with all fixed constants, are given in this file. Indeed, a SIF for linear programming problems can be completely specified by an MPS file; the SIF comprises a single section, the SDIF file, and that section is merely the MPS file.

3.1 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:

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

- Data cards, which contain the actual data.
- 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 Table 3.1.

Keyword	Comments	Presence	Described in §
`NAME`		mandatory	3.2.1
	either		
`GROUPS`		mandatory	3.2.6
`ROWS`	synonym for `GROUPS`		3.2.6
`CONSTRAINTS`	synonym for `GROUPS`		3.2.6
`VARIABLES`		mandatory	3.2.7
`COLUMNS`	synonym for `VARIABLES`		3.2.7
	or		
`VARIABLES`		mandatory	3.2.8
`COLUMNS`	synonym for `VARIABLES`		3.2.8
`GROUPS`		mandatory	3.2.9
`ROWS`	synonym for `GROUPS`		3.2.9
`CONSTRAINTS`	synonym for `GROUPS`		3.2.9
`CONSTANTS`		optional	3.2.10
`RHS`	synonym for `CONSTANTS`		3.2.10
`RHS'`	synonym for `CONSTANTS`		3.2.10
`RANGES`		optional	3.2.11
`BOUNDS`		optional	3.2.12
`START POINT`		optional	3.2.13
`QUADRATIC`		**optional**	3.2.14
`HESSIAN`	**synonym for** `QUADRATIC`		3.2.14
`QUADS`	**synonym for** `QUADRATIC`		3.2.14
`QUADOBJ`	**synonym for** `QUADRATIC`		3.2.14
`QSECTION`	**synonym for** `QUADRATIC`		3.2.14
`ELEMENT TYPE`		optional	3.2.15
`ELEMENT USES`		optional	3.2.16
`GROUP TYPE`		optional	3.2.17
`GROUP USES`		optional	3.2.18
`OBJECT BOUND`		optional	3.2.19
`ENDATA`		mandatory	3.2.2

Table 3.1: Possible indicator card

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`, `QUADRATIC`, `HESSIAN`, `QUADS`, `QUADOBJ`, `QSECTION`, `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 Section 3.2. 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 or 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 Section 3.1.1).

Any card with the character * in column 1 is a comment card; the remaining contents of the card are 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.

3.1.1 Names

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.

Unless otherwise indicated, we allow any name which uses 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 [17]). This includes 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:

1. a scalar name of the form *££££££££££* where each *£* is a valid character type excepting that the first *£* cannot be a `$`. A completely blank string is also not allowed. Futhermore, the strings `'SCALE'`, `'MARKER'`, `'DEFAULT'`, `'INTEGER'` and `'ZERO-ONE'` are reserved for special operations.

2. 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 Section 3.2.3, 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. An array *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 Section 3.2.3.

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.

3.1.2 Fortran Names

A notable exception to the above are Fortran names. A *Fortran* name takes the form of a sequence of one to six upper case letters or digits, the first of which must not be a digit. These names are used in Sections 3.2.15–3.2.18, 4.4.1–4.4.3 and 5.2.1.

3.1.3 Numerical Values

The definition of a specific problem normally requires the use of numerical (integer or real) data values. Such values can be specified in two ways. Firstly, the values may simply occur as integer or floating-point numbers in data fields 4 and 6. Secondly, values may be allocated to named parameters, known as integer or real parameters, and a value subsequently used by reference to a particular integer or 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 an X or a 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 Section 3.2.4).

We defer the definition of integer and real parameters until Section 3.2.3.

3.1.4 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 Section 2.5. 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 Figures 3.1 and 3.2, pages 18 and 20. 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 93). 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 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 is ignored when interpreting the input file.

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 choose here to name the groups first. 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 (2.19) and (2.20) 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, `NM1`, 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 `RANGES` section (lines 26 to 29). Finally, the equality constraint (2.21) is to be called `CONEQ` (line 15); 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 (2.18), inequality constraint (2.19) and equality constraint groups (2.21) 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 (2.19) 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 (2.21) 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 (2.21) 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 (2.20) 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 $\frac{1}{2}$

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 (2.22) 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 `L` specifies that it is the lower bound default that is to be changed. The string `'DEFAULT'` in field 3 indicates that the default is being 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 $x_i = \frac{1}{2}, 1 \leq i \leq 100$ 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. **New** Firstly, we recall that there is a quadratic objective group, $x_1^2 + 2x_1x_{100} \equiv \frac{1}{2}(4x_1x_1 + 2x_1x_{100} + 2x_{100}x_1)$. We need to specify the nonzero coefficients of the terms x_jx_k, and in our cases these are $h_{1,1} = 4$ and $h_{1,100} = h_{100,1} = 2$. The rule that we adopt is that there is no need to supply both nonzeros $h_{j,k}$ and $h_{k,j}$ since they are the same, and that one (whichever is unimportant) suffices. Thus $h_{1,1} = 4$ and we (arbitrarily) choose to give $h_{1,100} = 2$. In the `QUADRATIC` section on lines 39a to 39b, we indicate that the quadratic objective has two terms involving x_1; the coefficient 4 is given for the $x_1^2 \equiv x_1x_1$ term, while that for the x_1x_{100} term is assigned the value 2.

Next, we saw in Section 2.5 that there are four element types for the problem, being of the form (i) $(v_1 - v_2)v_3$, (ii) $p_1v_1v_2$, (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 `X2` 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 (`ZV` 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 (2.19), each being of the same type `2PROD`. We name these elements `CLEA1`, ..., `CLEA99` and `CLEB1`, ..., `CLEB99`. The assignments are made on lines 54 to 67 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 `ZV`. 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 65 and assigned on line 66. Line 63 assigns `REALI` to have the floating point value of the index `I`. This new value is then divided into the value 2 on line 64 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 66. 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 (2.20) each use a single element, named `CGE1`, ..., `CGE100`, of type `SINE` and the appropriate array assignments are made on lines 68 to 70. Finally, the equality constraint (2.21) is named `CEQ1` and typed `SQUARE` with appropriate elemental variable assignments on lines 72 to 74.

We next need to specify the nontrivial group types. This is done in the `GROUP TYPE` section on lines 75 to 77. We saw in Section 2.5 that a single nontrivial group, $p_1\alpha^2$, is required. On line 76, the name `PSQUARE` is given for the type and the group type variable α is named `ALPHA`. The string `GV` pin 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 78 to 90. The objective function group is nontrivial and its type is announced on line 79. The group uses the single nonlinear element `OBJ1` specified on line 80 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 announce their purposes. The inequality groups (2.19) 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 82 to 84; line 83 is flagged as assigning elements to a group with the string `XE` in field 1. The second set of inequality constraints (2.20) 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 85 to 89. Lastly the trivial equality constraint group (2.21) is assigned the nonlinear element `CEQ1` on line 90.

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 (2.18) cannot be smaller than zero. This data is specified on lines 91 and 92. The string `LO` in field

1 of line 92 indicates that a lower bound is known for the value of (2.18). 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 initially assumed to lie between plus and minus infinity.

line	F.1	Field 2	Field 3	Field 4	Field 5	Field 6
	<>	<—10—>	<—10—>	<—-12—->	<—10—>	<—-12—->
1	NAME		EG3		$ The example of §1.6	
2	IE	1		1		
3	IE	N		100		
4	*Variants of §1.6 obtained by choice of N on previous card					
5	RE	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	LO	BND1	'DEFAULT'	-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		START1	'DEFAULT'	0.5		
39		START1	Y	0.0		
39a	QUADRATIC					
39b		X1	X1	4.0	X100	2.0
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 ↑14	↑15 ↑24	↑25 ↑36	↑40 ↑49	↑50 ↑61

Figure 3.1: SDIF file (part 1) for the example of Section 2.5

3.2 Indicator and Data Cards

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

3.2.1 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 8 characters long. The syntax for the `NAME` card is given in Figure 3.3.

3.2.2 The `ENDATA` Indicator Card

The `ENDATA` indicator card simply announces the end 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 3.4.

3.2.3 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, multiplication and division). 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 Section 3.2.4 below).

The syntax for associating a parameter with a specific value is given in Figure 3.5.

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 a `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 `IE`, 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 `IR`, the integer parameter value named in field 2 is to be assigned the value of the nearest integer (closer to zero) to the value of the real parameter `rl--p-name` specified in field 3. The parameter appearing in field 3 must have already been assigned a value.

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 `IS`, the integer parameter named in field 2 is to be formed by subtracting the value of the parameter `int-p-name` referred to in field 3 from 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.

F.1	Field 2	Field 3	Field 4	Field 5	Field 6
<>	<—10—>	<—10—>	<—-12—->	<—10—>	<—-12—->
ELEMENT USES					
T	OBJ1	3PROD			
V	OBJ1	V1		X1	
ZV	OBJ1	V2		X(N)	
V	OBJ1	V3		X2	
DO	I	1		NM1	
IA	IP1	I	1		
XT	CLEA(I)	2PROD			
ZV	CLEA(I)	V1		X(1)	
ZV	CLEA(I)	V2		X(IP1)	
XP	CLEA(I)	P1	1.0		
XT	CLEB(I)	2PROD			
ZV	CLEB(I)	V1		X(I)	
ZV	CLEB(I)	V2		X(N)	
RI	REALI	I			
RD	2OVERI	REALI	2.0		
R+	2OVAI+1	2OVERI		ONE	
ZP	CLEB(I)	P1		2OVAI+1	
ND					
DO	I	1		N	
XT	CGE(I)	SINE			
ZV	CGE(I)	V1		X(I)	
ND					
T	CEQ1	SQUARE			
V	CEQ1	V1		X1	
ZV	CEQ1	V2		X(N)	
GROUP TYPE					
GV	PSQUARE				
GP	PSQUARE	P1			
GROUP USES					
T	OBJ	SQUARE			
E	OBJ	OBJ1			
P	OBJ	P1	0.5		
DO	I	1		NM1	
XE	CONLE(I)	CLEA(I)		CLEB(I)	
ND					
DO	I	1		N	
XT	CONGE(I)	PSQUARE			
XE	CONGE(I)	CGE(I)			
XP	CONGE(I)	P1	1.0		
ND					
E	CONEQ	CEQ1			
OBJECT BOUND					
LO	OBOUND		0.0		
ENDATA					
↑1 ↑2 ↑3	↑5 ↑10 ↑14	↑15 ↑24	↑25 ↑36	↑40 ↑49	↑50 ↑61

Figure 3.2: SDIF file (part 2) for the example of Section 2.5

	<—8–>	
	Field 3	
NAME	prob-nam	
	↑ 15 ↑ 24	

Figure 3.3: The indicator card `NAME`

ENDATA

Figure 3.4: The indicator card `ENDATA`

<>	<—10—>	<—10—>	<—-12—->	<—10—>
F.1	Field 2	Field 3	Field 4	Field 5
IE	int-p-name		numerical-vl	
IR	int-p-name	rl–p-name		
IA	int-p-name	int-p-name	numerical-vl	
IS	int-p-name	int-p-name	numerical-vl	
IM	int-p-name	int-p-name	numerical-vl	
ID	int-p-name	int-p-name	numerical-vl	
I=	int-p-name	int-p-name		
I+	int-p-name	int-p-name		int-p-name
I-	int-p-name	int-p-name		int-p-name
I*	int-p-name	int-p-name		int-p-name
I/	int-p-name	int-p-name		int-p-name
RE	rl–p-name		numerical-vl	
RI	rl–p-name	int-p-name		
RA	rl–p-name	rl–p-name	numerical-vl	
RS	rl–p-name	rl–p-name	numerical-vl	
RM	rl–p-name	rl–p-name	numerical-vl	
RD	rl–p-name	rl–p-name	numerical-vl	
RF	rl–p-name	funct-name	numerical-vl	
R=	rl–p-name	rl–p-name		
R+	rl–p-name	rl–p-name		rl–p-name
R-	rl–p-name	rl–p-name		rl–p-name
R*	rl–p-name	rl–p-name		rl–p-name
R/	rl–p-name	rl–p-name		rl–p-name
R(	rl–p-name	funct-name		rl–p-name
AE	r-p-a-name		numerical-vl	
AI	r-p-a-name	int-p-name		
AA	r-p-a-name	r-p-a-name	numerical-vl	
AS	r-p-a-name	r-p-a-name	numerical-vl	
AM	r-p-a-name	r-p-a-name	numerical-vl	
AD	r-p-a-name	r-p-a-name	numerical-vl	
AF	r-p-a-name	funct-name	numerical-vl	
A=	r-p-a-name	r-p-a-name		
A+	r-p-a-name	r-p-a-name		r-p-a-name
A-	r-p-a-name	r-p-a-name		r-p-a-name
A*	r-p-a-name	r-p-a-name		r-p-a-name
A/	r-p-a-name	r-p-a-name		r-p-a-name
A(	r-p-a-name	funct-name		r-p-a-name
↑ 2 ↑ 3	↑ 5 ↑ 14	↑ 15 ↑ 24	↑ 25 ↑ 36	↑ 40 ↑ 49

Figure 3.5: Possible cards for specifying parameter values

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

If the string is `I=`, the value of the integer parameter named in field 2 is to be set to the integer value specified for the parameter in field 3. The parameter appearing in field 3 must have already been assigned a value.

If the string is `I+`, the value of the integer parameter named in field 2 is to be calculated by adding 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.

If the string is `I-`, the value of the integer parameter named in field 2 is to be calculated by subtracting the value of the integer parameters `int-p-name` referred to in field 5 from that in field 3. The parameters appearing in fields 3 and 5 must have already been assigned values.

If the string is `I*`, 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. The parameters appearing in fields 3 and 5 must have already been assigned values.

Finally, if the string is `I/`, the value of the integer parameter named in field 2 is to be formed by dividing the value specified for the integer parameters in field 3 by that specified for the integer parameters in field 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 can only be at most seven characters in length. Furthermore, such a parameter name may not include the characters "`(`", "`)`" or "`,`".

If the string is `RE`, 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 ten 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 `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 `RS`, the value of the real parameter named in field 2 is to be formed by subtracting the value of the real parameter `rl--p-name` referred to in field 3 from 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 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 `RD`, the value of the parameter named in field 2 is to be formed by dividing the real value specified in field 4 by the value specified for the real parameter in field 3. The parameter appearing in field 3 must have already been assigned a value.

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 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.

If the string is `R=`, the parameter value named in field 2 is to be assigned the value of the real parameter `rl--p-name` referred to in field 3. The parameter appearing in field 3 must have already been assigned a value.

If the string is `R+`, 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 `R-`, the parameter value named in field 2 is to be formed by subtracting the value of the real parameter `rl--p-name` referred to in field 5 from the value of that referred to in field 3. The parameters appearing in fields 3 and 5 must have already been assigned values.

If the string is `R*`, 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 `R/`, the parameter value named in field 2 is to be formed by dividing the value of the real parameter `rl--p-name` referred to in field 3 by the value of that referred to in field 5. The parameters appearing in fields 3 and 5 must have already been assigned values.

Finally, if the string is `R(`, 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 `RF` 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 is 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 `AF` and `A(` cards, must be a real parameter array name with a valid index.

Parameter assignments may be made at any point within the SDIF file between the `NAME` 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.

3.2.4 Do-loops

A do-loop may occur at any point in the `GROUPS`, `VARIABLES`, `CONSTANTS`, `RANGES`, `BOUNDS`, `START POINT`, **NewQUADRATIC**, `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 3.6.

The two-character string in data field 1 specifies 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

<> F.1	<—10—> Field 2	<—10—> Field 3		<—10—> Field 5	
DO	int-p-name	int-p-name		int-p-name	
DI	int-p-name	int-p-name			
one or more array definitions					
OD	int-p-name				
ND					

↑↑ 2 3 ↑ 5 ↑ 14 ↑ 15 ↑ 24 ↑ 40 ↑ 49

Figure 3.6: Syntax for do-loops

have been defined on previous data cards. The parameter name defined in field 2 can occupy up to ten 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 3.5 — that is, cards whose first field are `IE`, `IR`, `IA`, `IS`, `IM`, `ID`, `I=`, `I+`, `I-`, `I*`, `I/`, `RE`, `RI`, `RA`, `RS`, `RM`, `RD`, `RF`, `R=`, `R+`, `R-`, `R*`, `R/`, `R(`, `AE`, `AI`, `AA`, `AS`, `AM`, `AD`, `AF`, `A=`, `A+`, `A-`, `A*`, `A/` or `R(` — 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 Section 3.4.

3.2.5 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 [3], 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 [5, pp30–31], 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 Section 3.2.6 and Section 3.2.7. 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 Section 3.2.8 and Section 3.2.9.

3.2.6 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 3.7.

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

<>	<—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
↑↑ 2 3	↑ 5 ↑ 14	↑ 15 ↑ 24	↑ 25 ↑ 36	↑ 40 ↑ 49	↑ 50 ↑ 61

Figure 3.7: Possible data cards for `GROUPS, ROWS` or `CONSTRAINTS`(column-wise)

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 x, and the k-th group has a linear element $a_k^T x - b_k$, the linear element that will be passed to the optimization procedure could be $a_k^T x - y_k - b_k$, 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 $a_k^T x + y_k - b_k$, 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.

`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 a parameter assignment (see Section 3.2.3).

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

$$\text{group in field } 2 = \text{group in field } 3 * \text{field } 4 + \text{group in field } 5 * \text{field } 6.$$

  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.

3.2.7 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 3.8.

<> F.1	<—10—> Field 2	<—10—> Field 3	<—-12—-> Field 4	<—10—> Field 5	<—-12—-> Field 6
VARIABLES or COLUMNS					
	varbl-name	$$$$$$$$$$	numerical-vl	$$$$$$$$$$	numerical-vl
X	varbl-name	$$$$$$$$$$	numerical-vl	$$$$$$$$$$	numerical-vl
Z	varbl-name	$$$$$$$$$$		r-p-a-name	
↑↑ 2 3	↑ 5 ↑ 14	↑ 15 ↑ 24	↑ 25 ↑ 36	↑ 40 ↑ 49	↑ 50 ↑ 61

Figure 3.8: 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 is 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 Section 3.2.3).

The string `$$$$$$$$$$` in data field 3 is used for five 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`'.

- It may be used to specify that the variable(s) mentioned in field 2 is(are) only allowed to take integer values. In this case field 3 will contain the string `'INTEGER'`.

- It may be used to specify that the variable(s) mentioned in field 2 is(are) only allowed to take the values 0 or 1. In this case field 3 will contain the string `'ZERO-ONE'`.

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.

3.2.8 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 3.9.

<> F.1	<—10—> Field 2	<—10—> Field 3	<—-12—-> Field 4	<—10—> Field 5	<—-12—-> Field 6
VARIABLES or COLUMNS					
	varbl-name	$$$$$$$$$$	numerical-vl		
X	varbl-name	$$$$$$$$$$	numerical-vl		
Z	varbl-name	$$$$$$$$$$		r-p-a-name	
↑↑ 2 3	↑ 5 … ↑ 14	↑ 15 … ↑ 24	↑ 25 … ↑ 36	↑ 40 … ↑ 49	↑ 50 … ↑ 61

Figure 3.9: 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 Section 3.2.3).

The string `$$$$$$$$$$` in data field 3 is used for four 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.
- It may be used to specify that the variable(s) mentioned in field 2 is(are) only allowed to take integer values. In this case field 3 will contain the string `'INTEGER'`.
- It may be used to specify that the variable(s) mentioned in field 2 is(are) only allowed to take the values 0 or 1. In this case field 3 will contain the string `'ZERO-ONE'`.

3.2.9 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 3.10.

<>	<—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
↑↑ 2 3	↑ 5 ↑ 14	↑ 15 ↑ 24	↑ 25 ↑ 36	↑ 40 ↑ 49	↑ 50 ↑ 61

Figure 3.10: Possible data cards for `GROUPS`, `ROWS`, or `CONSTRAINTS`(row-wise)

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 Section 3.2.6.

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 Section 3.2.3). 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

  $$\text{group in field 2} = \text{group in field 3} * \text{field 4} + \text{group in field 5} * \text{field 6}.$$

  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.

3.2.10 The `CONSTANTS`, `RHS` or `RHS'` Data Cards

The `CONSTANTS`, `RHS` or `RHS'` indicator cards are used interchangeably to announce the definition of a vector of the constant terms b_i (in the constrained case, the right-hand-sides) for each linear element. The syntax for data following this indicator card is given in Figure 3.11.

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 ten characters long. More than one vector of group constants may be defined.

The strings `$$$$$$$$$$` is used for two purposes.

- It may be used to assign a default value to all the constants in a particular vector. In this case field 3 will contain the string 'DEFAULT'.

<>	<—10—>	<—10—>	<—-12—->	<—10—>	<—-12—->
F.1	Field 2	Field 3	Field 4	Field 5	Field 6
CONSTANTS or RHS or RHS'					
	rhs–name	$$$$$$$$$$	numerical-vl	group-name	numerical-vl
X	rhs–name	$$$$$$$$$$	numerical-vl	group-name	numerical-vl
Z	rhs–name	$$$$$$$$$$		r-p-a-name	
2 3	5 — 14	15 — 24	25 — 36	40 — 49	50 — 61

Figure 3.11: Possible data cards for `CONSTANTS, RHS` or `RHS'`

- It may contain the name of a group/row/constraint for which the constant term/right-hand-side is to be specified. Such a string 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 Section 3.2.3). 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 third field contains the string `'DEFAULT'` as mentioned above. On such a card, the default value for the vector in field 2 is given in field 4 whenever field 1 is blank or contains the character `X`. If field 1 contains the character `Z`, the default value 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.

3.2.11 The `RANGES` Data Cards

The `RANGES` indicator card is used to announce the definition of a vector 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 3.12.

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

The string `$$$$$$$$$$` is used for two purposes.

- It may be used to assign a default value to all the range values in a particular vector. In this case field 3 will contain the string `'DEFAULT'`.

- It may contain the name of a group/row/constraint for which the range value is to be specified. Such a string must have been defined in the `GROUPS` section.

```
 <> <--10--> <--10--> <---12---> <--10--> <---12--->
 F.1 Field 2   Field 3   Field 4      Field 5   Field 6
RANGES
    range-name$$$$$$$$$$numerical-vl group-namenumerical-vl
 X  range-name$$$$$$$$$$numerical-vl group-namenumerical-vl
 Z  range-name$$$$$$$$$$             r-p-a-name
 ↑↑  ↑        ↑ ↑        ↑ ↑          ↑ ↑        ↑ ↑          ↑
 2 3 5       14 15      24 25        36 40      49 50        61
```

Figure 3.12: Possible data cards for `RANGES`

In addition, the (optional) string `group-name` in data field 5 may also define the name of a group/row/constraint for which the range value is to be specified.

The string `numerical-vl` in data field 4 and (optionally) 6 now contains the numerical value of the relevant range value and may occupy up to 12 locations. Only groups initially specified with a `G` or `L` in columns 1 or 2 of field 1 in the `GROUPS` section use range values 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 Section 3.2.3). 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 Section 3.2.6, the extra bound is taken to imply the inequality $0 \leq y_k \leq |\text{field 4 or 6}|$ 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 third field contains the string `'DEFAULT'` as mentioned above. On such a card, the default value for the vector in field 2 is given in field 4 whenever field 1 is blank or contains the character `X`. If field 1 contains the character `Z`, the default value is that associated with the real parameter, `rl--p-name`, named in field 5. The default value applies to each range value not explicitly specified. If the default is to be changed, the change must be made on the first card naming a particular vector of range values.

3.2.12 The `BOUNDS` Data Cards

The `BOUNDS` indicator card is used to announce a vector of data giving lower and upper bounds on the unknown variables. The syntax for data following this indicator card is given in Figure 3.13.

The two-character string in data field 1 specifies the type of bound to be input. Possible values are: `LO`, `XL` or `ZL`, in the case of a lower bound, `UP`, `XU`, `ZU`,in the case of an upper bound, `FX`, `XX`, `ZX`,in the case of a fixed variable, i.e., the lower and upper bounds are equal, `FR` or `XR` if the variable is free, i.e., the lower and upper bounds are infinite, `MI` or `XM`, if there is no lower bound, and `PL` or `XP`, if there is no upper bound. The string `bound-name` in data field 2 gives the name of the bound vector under consideration. This name may be up to ten characters long. Several different bound vectors may be defined in the `BOUNDS` section.

The string `$$$$$$$$$$` is used for two purposes.

- It may contain the name of a variable/column for which a bound is to be specified.

<>	<—10—>	<—10—>	<—-12—->	<—10—>	<—-12—->
F.1	Field 2	Field 3	Field 4	Field 5	
BOUNDS					
LO	bound-name	$$$$$$$$$$	numerical-vl		
UP	bound-name	$$$$$$$$$$	numerical-vl		
FX	bound-name	$$$$$$$$$$	numerical-vl		
FR	bound-name	$$$$$$$$$$			
MI	bound-name	$$$$$$$$$$			
PL	bound-name	$$$$$$$$$$			
XL	bound-name	$$$$$$$$$$	numerical-vl		
XU	bound-name	$$$$$$$$$$	numerical-vl		
XX	bound-name	$$$$$$$$$$	numerical-vl		
XR	bound-name	$$$$$$$$$$			
XM	bound-name	$$$$$$$$$$			
XP	bound-name	$$$$$$$$$$			
ZL	bound-name	$$$$$$$$$$		r-p-a-name	
ZU	bound-name	$$$$$$$$$$		r-p-a-name	
ZX	bound-name	$$$$$$$$$$		r-p-a-name	
↑↑ 2 3	↑ 5 ↑ 14	↑ 15 ↑ 24	↑ 25 ↑ 36	↑ 40 ↑ 49	

Figure 3.13: Possible data cards for `BOUNDS`

This name may be up to ten characters long and must refer to a variable defined in the `VARIABLE` data.

If the card is of type `LO`, `UP`, `FX`, `FR`, `MI`, or `PL`, the string in data field 3 specifies to which variable the bound is applied. If the card is of type `XL`, `ZL`, `XU`, `ZU`, `XX`, `ZX`, `XR`, `XM` or `XP`, this string 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 Section 3.2.3).

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`–`ZU`, `ZL`–`ZU`, `ZL`–`XP` and `XM`–`ZU`.

- It may be used to assign a default value to all the lower and/or upper bounds in a particular vector. In this case field 3 will contain the string `'DEFAULT'`. 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. These default values may be changed. A new default value for the vector in field 2 is then given in field 4 whenever field 1 is `LO`, `UP`, `FX`, `FR`, `MI`, `PL` or starts with the character `X`. If field 1 starts with the character `Z`, the default value is that associated with the real parameter, `rl--p-name`, named in field 5. The appropriate default value applies to each bound not explicitly specified. If the default is to be changed, the change must be made on the first card naming a particular vector of bounds.

  If default lower and upper bounds of zero and infinity, respectively, are in effect and a card with `MI` or `XM` in field 1 is encountered, the relevant lower bound is changed to minus infinity *and* the upper bound becomes zero. If the same defaults are in effect and an upper bound of zero is specified on a `UP`, `XU` or `ZU` card, the relevant lower bound becomes minus infinity. These two features are necessary for MPS compatibility.

3.2.13 The `START POINT` Data Cards

The `START POINT` indicator card is used to announce a vector of initial estimates of the values of the unknown variables and, in the case of problems with general constraints, Lagrange multipliers. The Lagrangian function associated with (2.1)–(2.4) is the function

$$
\begin{aligned}
l(x,\lambda) \quad = \quad & \sum_{i \in I_0} g_i \left(\sum_{j \in J_i} w_{i,j} f_j(x_j) + a_i^T x - b_i \right) \\
& + \sum_{i \in I_E \cup I_I} \lambda_i g_i \left(\sum_{j \in J_i} w_{i,j} f_j(x_j) + a_i^T x - b_i \right)
\end{aligned}
$$

where the scalars λ_i are known as Lagrange multipliers. Good estimates of these parameters can sometimes be useful for optimization procedures (see, for example, [10]). The syntax for data following this indicator card is given in Figure 3.14.

<> F.1	<—10—> Field 2	<—10—> Field 3	<—-12—-> Field 4	<—10—> Field 5	<—-12—-> Field 6
`START POINT`					
V	start-name	$$$$$$$$$$	numerical-vl	varbl-name	numerical-vl
XV	start-name	$$$$$$$$$$	numerical-vl	varbl-name	numerical-vl
ZV	start-name	$$$$$$$$$$		rl–p-name	
M	start-name	$$$$$$$$$$	numerical-vl	multp-name	numerical-vl
XM	start-name	$$$$$$$$$$	numerical-vl	multp-name	numerical-vl
ZM	start-name	$$$$$$$$$$		rl–p-name	
	start-name	$$$$$$$$$$	numerical-vl	varbl-name	numerical-vl
	start-name	$$$$$$$$$$	numerical-vl	multp-name	numerical-vl
X	start-name	$$$$$$$$$$	numerical-vl	varbl-name	numerical-vl
X	start-name	$$$$$$$$$$	numerical-vl	multp-name	numerical-vl
Z	start-name	$$$$$$$$$$		rl–p-name	
↑↑ 2 3	↑ 5 ↑ 14	↑ 15 ↑ 24	↑ 25 ↑ 36	↑ 40 ↑ 49	↑ 50 ↑ 61

Figure 3.14: Possible data cards for `START POINT`

The `V`, `XV` and `ZV` cards are used to define the starting value for variables. In any of these cards, the string `start-name` in data field 2 gives the name of a starting vector and may be up to ten characters long. Several different starting vectors may be defined in the `START POINT` section. The string `$$$$$$$$$$` in field 3 is used for two purposes.

- It must contain the name of a variable defined in the `VARIABLES` section, when a starting value is to be assigned to that variable. If field 1 does not contain `ZV`, the optional string `varbl-name` in data field 5 may also contain the name of such a variable whose starting value is to be assigned.
- It may be used to assign a default value to all the starting values for variables in a particular vector. In this case field 3 must contain the string `'DEFAULT'`.

Each starting point vector is given a default value for every variable. This value is initially zero, but it may be changed. The appropriate default value applies to each variable not explicitly specified.

The `M`, `XM` and `ZM` cards are used to define the starting value for Lagrange multipliers. In any of those cards, the string `start-name` in data field 2 gives the name of a starting vector and may be up to ten characters long. Several different starting vectors may be defined in the `START POINT` section. The string `$$$$$$$$$$` in field 3 is used for two purposes.

- It must contain the name of a group defined in the `GROUPS` section which is not an objective function group, when a starting value is to be assigned to the corresponding Lagrange multiplier. If field 1 does not contain `ZV`, the optional string `multp-name` in data field 5 may also contain the name of such a multiplier whose starting value is to be assigned.
- It may be used to assign a default value to all the starting values for Lagrange multipliers in a particular vector. In this case field 3 must contain the string `'DEFAULT'`.

  Each starting point vector is given a default value for every Lagrange multiplier. This value is initially zero, but it may be changed. The appropriate default value applies to each multiplier not explicitly specified.

The effect of a card whose field 1 is blank or contains `X` or `Z` is similar to that of `V`, `XV`, `ZV`, `M`, `XM` and `ZM` cards, except that

- variables and Lagrange multipliers may be mixed up on cards whose field 3 does not contain `'DEFAULT'`,
- default values are assigned to both variables and Lagrange multipliers on card whose field 3 contains `'DEFAULT'`. The default value for both variables and multipliers is initially zero.

If the defaults are to be changed, the change must be made on the first card naming a particular starting point vector.

Starting values for an array of variables or Lagrange multipliers may only be defined on cards whose field 1 begins with the character `X` or `Z`; 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 Section 3.2.3).

It remains to specify the numerical value of the default or individual starting point as appropriate. On cards whose field 1 starts with `Z`, the value is that previously associated with the real parameter `rl--p-name` or array of real parameters `rl--p-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).

New

3.2.14 The QUADRATIC, HESSIAN, QUADS, QUADOBJ or QSECTION Data Cards

The `QUADRACTIC`, `HESSIAN`, `QUADS`[1], `QUADOBJ`[2] and `QSECTION`[3] indicator cards are used interchangeably to announce any nonzero coefficients $h_{j,k}$ in the quadratic objective group $\frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n} h_{j,k}x_jx_k$. Only one of each pair $(h_{j,k}, h_{k,j})$, $(j \neq k)$, of "off-diagonal" terms should be given, but which is unimportant. Any repeated coefficients will be summed. The syntax for data following these indicator cards is given in Figure 3.15.

```
<> <---10---> <---10---> <----12----> <---10---> <----12---->
F.1  Field 2    Field 3     Field 4     Field 5     Field 6
+-----------------------------------------------------------+
|QUADRATIC or                                               |
|HESSIAN or                                                 |
|QUADS or                                                   |
|QUADOBJ or                                                 |
|QSECTION                                                   |
+-----------------------------------------------------------+
||   |varbl-name|varbl-name|numerical-vl|varbl-name|numerical-vl|
||X  |varbl-name|varbl-name|numerical-vl|varbl-name|numerical-vl|
||Z  |varbl-name|varbl-name|            |r-p-a-name|            |
+-----------------------------------------------------------+
 ↑↑  ↑        ↑ ↑        ↑ ↑          ↑   ↑        ↑ ↑         ↑
 2 3 5       14 15      24 25        36  40       49 50       61
```

Figure 3.15: Possible data cards for `HESSIAN`, `QUADS`, `QUADOBJ` or `QSECTION`

The strings `varbl-name` in data fields 2 and 3 (and optionally 2 and 5 for those cards whose field 1 does not contain `Z`) give the names of pairs of problem variables x_j and x_k for which $h_{j,k}$ is nonzero. All problem variables must have been previously set in the `VARIABLES/COLUMNS` section. Additionally, on a `Z` card, the name of the variable must be an element of an array of variables, with a valid name and index, while on a `V` card, the name may be either a scalar or an array name.

On cards whose field 1 is either empty or contains the character `X`, the strings `numerical-vl` in data fields 4 and (optionally) 6 contain the associated numerical values of the coefficients $h_{j,k}$. On cards for which field 1 contains the character `Z`, 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.

3.2.15 The ELEMENT TYPE Data Cards

The `ELEMENT TYPE` indicator card is used to announce the data for the different types of nonlinear elements 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 3.16.

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

[1] This indicator card is included for compatibility with the proposed MPS format extension of Ponceleón [16].

[2] This indicator card was given by Maros and Meszaros [14], for their compatible proposed MPS extension.

[3] This indicator card is included for compatibility with the OSL [12] extensions to the MPS format.

<> F.1	<—10—> Field 2	<–6-> Field 3		<–6-> Field 5	
ELEMENT TYPE					
EV	etype-name	ev-nam		ev-nam	
IV	etype-name	iv-nam		iv-nam	
EP	etype-name	ep-nam		ep-nam	

↑↑ 2 3 ↑ 5 ↑ 14 ↑ 15 ↑ 20 ↑ 40 ↑ 45

Figure 3.16: Possible data cards for `ELEMENT TYPE`

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 `etype-name` in data field 2 gives the name of the element type under consideration. This name may be up to ten 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 must be valid Fortran names (see Section 3.1.2). 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.

3.2.16 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 3.17.

<> F.1	<—10—> Field 2	<–6-> Field 3	<—-12—-> Field 4	<–6-> Field 5	<—-12—-> Field 6
ELEMENT USES					
T	$$$$$$$$$$	etype-nam			
XT	$$$$$$$$$$	etype-nam			
V	elmnt-name	ev-nam		varbl-name	
ZV	elmnt-name	ev-nam		varbl-name	
P	elmnt-name	ep-nam	numerical-vl	ep-nam	numerical-vl
XP	elmnt-name	ep-nam	numerical-vl	ep-nam	numerical-vl
ZP	elmnt-name	ep-nam		r-p-a-nam	

↑↑ 2 3 ↑ 5 ↑ 14 ↑ 15 ↑ 20 ↑ 24 ↑ 25 ↑ 36 ↑ 40 ↑ 45 ↑ 49 ↑ 50 ↑ 61

Figure 3.17: Possible data cards for `ELEMENT USES`

There are three sorts of data cards in the `ELEMENT USES` section. For cards of the second and third kinds, 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 Section 3.2.3).

The first kind of cards, identified by the characters `T` or `XT` in field 1, give the name, or an array of names, of an element and its type. On such cards, the string `$$$$$$$$$$` in field 2 is used for two purposes.

- It may contain the name, or an array of names, of a nonlinear element function whose type is to be defined. This name may be up to ten characters long and each nonlinear element name must be unique. On `XT` cards the expanded element array name in field 2 must be valid and the integer indices must have been defined in a parameter assignment (see Section 3.2.3).
- It may be used to assign a default type to all the nonlinear element functions. In this case field 2 must contain the string '`DEFAULT`'. Any element not explicitly typed is assumed to belong to a default type. If a default is to be used, it must be specified on a card and such a card may only appear before the first `T` or `XT` card in the `ELEMENT USES` section. The string `etype-name` in data field 3 gives the name of the element type to be used. This name may be up to ten characters long and must have appeared in the `ELEMENT TYPE` section.

The second kind of data card, identified by the characters `V` or `ZV` 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 be a valid Fortran name (see Section 3.1.2). The string `varbl-name` in data field 5 then gives the name 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 a `ZV` 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 string `ep-nam` in data field 3 (and, for `P` and `XP` cards, optionally 5) must give the name of a parameter. This name must have been set in the `ELEMENT TYPE` section and be a valid Fortran name, see Section 3.1.2. 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.

3.2.17 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 3.2.17.

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 `gtype-name` in data field 2 gives the name of a nontrivial group type and may be up to ten 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. This string must be a valid

<>	<—10—>	<–6->		<–6->
F.1	Field 2	Field 3		Field 5
GROUP TYPE				
GV	gtype-name	gv-nam		
GP	gtype-name	gp-nam		gp-nam

2 3 5 14 15 20 40 45

Figure 3.18: Possible data cards for `GROUP TYPE`

Fortran name (see Section 3.1.2). 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 must again be valid Fortran names. 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.

3.2.18 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 3.19.

<>	<—10—>	<–6->	<—-12—->	<–6->	<—-12—->
F.1	Field 2	Field 3	Field 4	Field 5	Field 6
GROUP USES					
T	group-name	gtype-nam			
XT	group-name	gtype-nam			
E	group-name	elmnt-nam	blank/num-vl	elmnt-name	blank/num-vl
XE	group-name	elmnt-nam	blank/num-vl	elmnt-name	blank/num-vl
ZE	group-name	elmnt-nam		r-p-a-name	
P	group-name	gp-nam	numerical-vl	gp-nam	numerical-vl
XP	group-name	gp-nam	numerical-vl	gp-nam	numerical-vl
ZP	group-name	gp-nam		r-p-a-name	

2 3 5 14 15 20 24 25 36 40 45 49 50 61

Figure 3.19: Possible data cards for `GROUP USES`

There are three sorts of data cards in the `GROUP USES` section. For cards of the second and third kinds, 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 must have been defined in the `GROUPS/ROWS/CONSTRAINTS` section.

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 Section 3.2.3).

The first kind of cards, 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. On such cards, the string `$$$$$$$$$$` in field 2 is used for two purposes.

- It may contain the name, or an array of names, of a group function whose type is to be defined. The name may be up to ten characters long and must have been defined in the `GROUPS/ROWS/CONSTRAINTS` section. On `XT` cards the expanded element array name in field 2 must be valid and the integer indices must have been defined in a parameter assignment (see Section 3.2.3).
- It may be used to assign a default type to all the group functions. In this case field 2 must contain the string '`DEFAULT`'. Such a card may only appear before the first `T` or `XT` card in the `GROUP USES` section. Any group not explicitly typed is assumed to belong to a default type. The initial default type is trivial but the default may be changed. The string `gtype-name` in data field 3 gives the name of the group type to be used. This name may be up to ten 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 a re to be included in a given group. Optionally the given elements may be multiplied by specified 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 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, the weights are assigned 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 names must have been set in the `GROUP TYPE` section and be valid Fortran name, see Section 3.1.2. 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 `E`, `XE`, `ZE`, `P`, `XP` or `ZP` cards.

3.2.19 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 3.20.

The two-character string in data field 1 specifies the type of bound to be input. Possible values are: `LO`, `XL` or `ZL` for a lower bound, and `UP`, `XU` or `ZU` for an upper bound. The string `obbnd-name` in data field 2 gives a name to the bounds under

<> F.1	<—10—> Field 2	<—-12—-> Field 4	<—10—> Field 5
OBJECT BOUND			
LO	obbnd-name	numerical-vl	
UP	obbnd-name	numerical-vl	
XL	obbnd-name	numerical-vl	
XU	obbnd-name	numerical-vl	
ZL	obbnd-name		rl–p-name
ZU	obbnd-name		rl–p-name
2 3	5 14	25 36	40 49

Figure 3.20: Possible data cards for `OBJECT BOUND`

consideration. This name may be up to ten 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 `ZL` or `ZU`, 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`, `XL`, `XU`, `ZL` or `ZU` card.

3.3 Another Example

In Section 2.3, we gave an example. An SDIF file for this example is given in Figure 3.21. 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`.

3.4 A Further Example

In Section 2.4, 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 3.22. 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 the 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`.

<>	<—10—>	<—10—>	<—-12—->	<—10—>	<—-12—->
F.1	Field 2	Field 3	Field 4	Field 5	Field 6
NAME		DOC			
GROUPS					
E	GROUP1				
E	GROUP2				
E	GROUP3				
VARIABLES					
	X1	GROUP1	1.0		
	X2	GROUP3	1.0		
	X3				
BOUNDS					
FR	BN1	X1			
LO	BN1	X2	-1.0D+0		
LO	BN1	X3	1.0D+0		
UP	BN1	X2	1.0D+0		
UP	BN1	X3	2.0D+0		
ELEMENT TYPE					
EV	ETYPE1	V1			
EV	ETYPE1	V2			
EV	ETYPE2	V1			
EV	ETYPE2	V2			
EV	ETYPE2	V3			
IV	ETYPE2	U1			
IV	ETYPE2	U2			
ELEMENT USES					
T	G2E1	ETYPE1			
V	G2E1	V1		X2	
V	G2E1	V2		X3	
T	G3E1	ETYPE2			
V	G3E1	V1		X2	
V	G3E1	V2		X1	
V	G3E1	V3		X3	
T	G3E2	ETYPE1			
V	G3E2	V1		X1	
V	G3E2	V2		X3	
GROUP TYPE					
GV	GTYPE1	ALPHA			
GV	GTYPE2	ALPHA			
GROUP USES					
T	GROUP1	GTYPE1			
T	GROUP2	GTYPE2			
E	GROUP2	G2E1			
E	GROUP3	G3E1			
E	GROUP3	G3E2			
ENDATA					
↑↑ 2 3	↑ 5 ↑ 14	↑ 15 ↑ 24	↑ 25 ↑ 36	↑ 40 ↑ 49	↑ 50 ↑ 61

Figure 3.21: SDIF file for the example of Section 2.3

<> F.1	<—10—> Field 2	<—10—> Field 3	<—-12—-> Field 4	<—10—> Field 5	<—-12—-> Field 6
NAME		DOC2			
IE	ONE		1		
IE	N		1000		
IA	NM1		-1		
VARIABLES					
DO	I	ONE		N	
	X	X(I)			
ND					
GROUPS					
DO	I	ONE		NM1	
XN	G(I)	X(ONE)	1.0		
ND					
XN	G(N)				
CONSTANTS					
	CONST	'DEFAULT'	1.0		
X	CONST	G(N)	0.0		
BOUNDS					
FR	BND	'DEFAULT'			
ELEMENT TYPE					
EV	SQUARE	V			
ELEMENT USES					
DO	I	ONE		N	
XT	E(I)	SQUARE			
ZV	E(I)	V		X(I)	
ND					
GROUP TYPE					
GV	SINE	ALPHA			
GP	SINE	P			
GROUP USES					
DO	I	ONE		NM1	
XT	G(I)	SINE			
XE	G(I)	E(I)		E(N)	
XP	G(I)	P	1.0		
ND					
E	G1000	E1000			
P	G1000	P	0.5		
ENDATA					

↑ 2 ↑ 3 ↑ 5 ↑ 14 ↑ 15 ↑ 24 ↑ 25 ↑ 36 ↑ 40 ↑ 49 ↑ 50 ↑ 61

Figure 3.22: SDIF file for the example of Section 2.4

4 The Standard Input Format for Nonlinear Elements

In addition to the problem data described in Section 3, 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 Section 3. 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.

4.1 Introduction to the Standard Element Type Input Format

4.1.1 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 Section 3.2.15. An optimization procedure is likely to require the values of the element functions and possibly their first and 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 u by

$$\nabla_u f \text{ and } \nabla_{uu} f$$

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

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

where W is defined by (2.11).

We thus need only supply derivatives with respect to u. Formally, we must define the function value f, possibly the gradient vector $\nabla_u f$ (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 $\nabla_{uu} f$ (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 u. We now describe how to set up the data for a given problem.

4.2 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 4.1.

Keyword	Comments	Presence	Described in §
`ELEMENTS`	same as `NAME`	mandatory	3.2.1
`TEMPORARIES`		optional	4.4.1
`GLOBALS`		optional	4.4.2
`INDIVIDUALS`		optional	4.4.3
`ENDATA`		mandatory	3.2.2

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 two kinds. The first are like those described in Section 3.1. 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). We allow the use of any of the language's intrinsic functions in such an expression. Continuation of an expression over at most nineteen lines is also permitted.

4.3 An Example

Before we give the complete syntax for an SEIF file, we continue the illustrative example that we started in Section 3.1.4 and show how to specify an input file appropriate for the problem of Section 2.5. Once again, there are many possible ways of specifying a particular problem; we give one in Figure 4.2. 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 subsequent 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 indicates 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 `A` 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 `A+` 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 if 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 Section 3.1.4. 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 involving more than two elemental variables on lines in which the string `R+` appears in the same field.

We now specify the function and derivative values of the element type u_1u_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

			<——41——Field 7——>		
<>	<—10—>	<—10—>	<—12—>	<—10—>	<—12—>
F.1	Field 2	Field 3	Field 4	Field 5	Field 6
ELEMENTS		EG3			
TEMPORARIES					
R	SINV1				
R	ZERO				
R	ONE				
R	TWOP1				
M	SIN				
M	COS				
GLOBALS					
A	ZERO		0.0		
A	ONE		1.0		
INDIVIDUALS					
T	3PROD				
R	U1	V1	1.0	V2	-1.0
R	U2	V3	1.0		
F			U1*U2		
G	U1		U2		
G	U2		U1		
H	U1	U1	ZERO		
H	U1	U2	ONE		
H	U2	U2	ZERO		
T	2PROD				
F			V1*V2		
G	V1		V2		
G	V2		V1		
H	V1	V1	ZERO		
H	V1	V2	ONE		
H	V2	V2	ZERO		
T	SINE				
A	SINV1		SIN(V1)		
F			SINV1		
G	V1		COS(V1)		
H	V1	V1	-SINV1		
T	SQUARE				
R	U1	V1	1.0	V2	1.0
A	TWO		2.0		
F			U1*U1		
G	U1		TWO*U1		
H	U1	U1	TWO		
ENDATA					

1 2 3 5 10 14 15 20 24 25 36 40 49 50 61 65

Figure 4.2: SEIF file for the example of Section 2.5

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 a transformation from elemental to internal variables $u_1 = v_1 + v_2$, is defined on lines 34 to 39. Again notice that the value 2.0 occurs in both first and second derivatives, so the auxiliary quantity TWO is set on line 36 to hold this value.

4.4 Data Cards

The ELEMENTS and ENDATA indicator cards perform the same function as the cards NAME and ENDATA in Section 3.2.1 and 3.2.2. 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.

4.4.1 The TEMPORARIES Data Card

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 $u_1 w$.

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 4.3.

```
<><–6->
F.1 Field 2
TEMPORARIES
 I   p-name
 R   p-name
 L   p-name
 M   p-name
 F   p-name
 ↑↑  ↑     ↑
 2 3 5     10
```

Figure 4.3: Possible data cards for `TEMPORARIES`

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`), logical (`L`), intrinsic function (`M`) or external function (`F`). The string `p-name` in field 2 then gives the name of the auxiliary parameter. The name must be a valid Fortran name, see Section 3.1.2, but must not be a reserved one, i.e., one of the names assigned to the internal variables or parameters for the element in question in the `ELEMENT TYPE` section of the SDIF (see, Section 3.2.15). Any auxiliary parameter that is to be used must be defined in the `TEMPORARIES` section along with all intrinsic and external function names.

4.4.2 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 4.4.

```
<><–6->  <–6->                  <————————————41————————————->
F.1 Field 2  Field 3                           Field 7
GLOBALS
 A   p-name           &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&& &
 A+                   &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&& &
 I   l-name  p-name   &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&& &
 I+                   &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&& &
 E   l-name  p-name   &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&& &
 E+                   &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&& &
 ↑↑  ↑    ↑   ↑    ↑          ↑                             ↑
 2 3 5    10  15   20         25                            65
```

Figure 4.4: Possible data cards for `GLOBALS`

The one or two character string in field 1 specifies the type of assignment that is to be made from the card. Possible values for the first character of the string are:

`A` This card announces that an auxiliary parameter 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 must be a valid Fortran name, see Section 3.1.2, and must have been previously defined in the `TEMPORARIES` section. The string in field 7 is an arithmetic expression. The assignment

$$\text{auxiliary variable named in field 2} \leftarrow \text{field 7}$$

is made, where again $\leftarrow$ means "is given the value"; any variable mentioned in the arithmetic expression must either be reserved (see Section 4.4.1), 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 on a previous `GLOBALS` data card.

`I` This card announces that an auxiliary parameter is to be assigned a value whenever a second logical auxiliary parameter has the value `.TRUE.` The string `p-name` in field 3 gives the name of the auxiliary parameter that is to be defined; this name must be a valid Fortran name, see Section 3.1.2, and must have been previously defined in the `TEMPORARIES` section. The string in field 7 is an arithmetic expression. The assignment

$$\text{auxiliary variable named in field 3} \leftarrow \text{field 7}$$

will be made if and only if the logical auxiliary parameter `l-name` specified in field 2 has the value `.TRUE.`; the logical parameter must have been previously defined in the `TEMPORARIES` section and allocated a value in the `GLOBALS` section. The arithmetic expression must obey the rules set out in the `A` section above.

`E` This card announces that an auxiliary parameter is to be assigned a value whenever a second logical auxiliary parameter has the value `.FALSE.` The string `p-name` in field 3 gives the name of the auxiliary parameter that is to be defined; this name must be a valid Fortran name, see Section 3.1.2, and must have been previously defined in the `TEMPORARIES` section. The string in field 7 is an arithmetic expression. The assignment

$$\text{auxiliary variable named in field 3} \leftarrow \text{field 7}$$

will be made if and only if the logical auxiliary parameter, `l-name`, specified in field 2 has the value `.FALSE.`; the logical parameter must have been previously defined in the `TEMPORARIES` section and allocated a value in the `GLOBALS` section. The arithmetic expression must obey the rules set out in the `A` section above.

The data started on an `A`, `I` and `E` card may be continued on a card whose first field contains an `A+`, `I+` or `E+` 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 `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 Section 4.4.3).

4.4.3 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 4.5.

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 `etype-name` in field 2 gives the name of the element type; the name may be up to ten characters long and must have been defined in the `ELEMENT TYPE` section of the SDIF file (see Section 3.2.15).

F.1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7
<>	<—10—>	<—10—>	<—-12—->	<—10—>	<—-12—->	<————-41—-Field 7————->
INDIVIDUALS						
T	etype-name					
R	iv-nam	ev-nam	numerical-vl	ev-nam	numerical-vl	
A	p-name		&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&			
A+			&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&			
I	l-name	p-name	&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&			
I+			&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&			
E	l-name	p-name	&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&			
E+			&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&			
F			&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&			
F+			&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&			
G	iv-nam		&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&			
G+			&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&			
H	iv-nam	iv-nam	&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&			
H+			&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&			

↑↑ ↑ ↑ ↑ ↑ ↑ ↑↑ ↑ ↑ ↑ ↑ ↑ ↑

2 3 5 10 14 15 20 24 25 36 40 49 50 61 65

Figure 4.5: Possible data cards for `INDIVIDUALS`

`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 Section 3.2.15). The transformation is specified by the matrix W of Section 2.2; 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 must be a valid Fortran name, see Section 3.1.2, and 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 must be valid Fortran names and 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 (2.9) 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 must be a valid Fortran name, see Section 3.1.2, and have been previously defined in the `TEMPORARIES` section. The string in field 7 is an arithmetic expression. The assignment

$$\text{auxiliary variable named in field 2} \leftarrow \text{field 7}$$

is made, where again ← means "is given the value"; any variable mentioned in the arithmetic expression must either be reserved (see Section 4.4.1), 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 `A`, `E` or `I` card for the current element type in the `ELEMENTS` section.

`I` This card announces that an auxiliary parameter, specific to the current element type, is to be assigned a value whenever a second logical auxiliary parameter has the value `.TRUE.` The string, `p-name`, in field 3 gives the name of the auxiliary parameter that is to be defined; this name must be a valid Fortran name, see Section 3.1.2, and have been previously defined in the `TEMPORARIES` section. The string in field 7 is an arithmetic expression. The assignment

$$\text{auxiliary variable named in field 3} \leftarrow \text{field 7}$$

will be made if and only if the logical auxiliary parameter, `l-name`, specified in field 2 has the value `.TRUE.`; the logical parameter must have been previously defined in the `TEMPORARIES` section and allocated a value in the `GLOBALS` or `INDIVIDUALS` section. The arithmetic expression must obey the rules set out in the `A` section above.

`E` This card announces that an auxiliary parameter, specific to the current element type, is to be assigned a value whenever a second logical auxiliary parameter has the value `.FALSE.` The string, `p-name`, in field 3 gives the name of the auxiliary parameter that is to be defined; this name must be a valid Fortran name, see Section 3.1.2, and have been previously defined in the `TEMPORARIES` section. The string in field 7 is an arithmetic expression. The assignment

$$\text{auxiliary variable named in field 3} \leftarrow \text{field 7}$$

will be made if and only if the logical auxiliary parameter, `l-name`, specified in field 2 has the value `.FALSE.`; the logical parameter must have been previously defined in the `TEMPORARIES` section and allocated a value in the `GLOBALS` or `INDIVIDUALS` section. The arithmetic expression must obey the rules set out in the `A` section above.

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

$$\text{nonlinear element function} \leftarrow \text{field 7}$$

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 must be a valid Fortran name, see Section 3.1.2, and have been defined on an `IV` data line, for a nonlinear element defined with internal variables, or an `EV` data line, for an element without explicit internal variables, in the `ELEMENT TYPE` section of the SDIF file. The string in field 7 is an arithmetic expression; the assignment

$$\text{derivative of element w.r.t. variable in field 2} \longleftarrow \text{field 7}$$

is made; any variable mentioned in the arithmetic expression must obey the rules set out in the `A` section above. `G` cards are optional. However, 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 fields 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. Either string must be a valid Fortran name, see Section 3.1.2, and have been defined on an `IV` data line, for a nonlinear element defined with internal variables, or an `EV` data line, for an element without explicit internal variables, in the `ELEMENT TYPE` section of the SDIF file. The string in field 7 is an arithmetic expression; the assignment

$$\text{second derivative of element w.r.t. variables in fields 2 and 3} \longleftarrow \text{field 7}$$

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 generally recommend this method of specifying invariant subspaces.

The data started on an `A`, `I`, `E`, `F`, `G` and `H` card may be continued on a card whose first field contains an `A+`, `I+`, `E+`, `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 element type must occur on consecutive cards and in the order given in Figure 4.5, excepting that `A`, `I` and `E` cards may be intermixed. 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.

4.5 Two Further Examples

In Section 2.3, we gave an example. An SEIF file for this example is given in Figure —refF3.3.1. 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 (although, we could have just not specified these particular components, which would then have taken their default zero 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.

```
                                       <-----------41---Field 7----------->
 <> <---10---> <---10---><---12----> <---10---><---12---->
 F.1  Field 2    Field 3    Field 4     Field 5    Field 6
ELEMENTS      DOC
TEMPORARIES
 R  CS
 R  SN
 R  ZERO
 R  SIN
 R  COS
GLOBALS
 G  ZERO                0.0D0
INDIVIDUALS
 T  ETYPE1
 F                      V1*V2
 G  V1                  V2
 G  V2                  V1
 H  V1        V1        ZERO
 H  V1        V2        1.0D0
 H  V2        V2        ZERO
 T  ETYPE2
 R  U1        V1        1.0D0
 R  U2        V2        1.0D0          V3        1.0D0
 A  CS                  COS(U2)
 A  SN                  SIN(U2)
 F                      U1*SN
 G  U1                  SN
 G  U2                  U1*CS
 H  U1        U1        ZERO
 H  U1        U2        CS
 H  U2        U2        -U1*SN
ENDATA
 ↑↑ ↑    ↑   ↑ ↑    ↑   ↑↑         ↑  ↑       ↑ ↑         ↑   ↑
 2 3 5   10  14 15  20  24 25      36 40      49 50       61  65
```

Figure 4.6: SEIF file for the element types for the exampleof Section 2.3

We gave a second example in Section 2.4. An SEIF file for this example is given in Figure 4.7 on page 54. 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.

```
                                <--------41---Field 7-------->
<> <---10---> <---10---><---12---> <---10---><---12--->
F.1  Field 2   Field 3    Field 4     Field 5     Field 6
ELEMENTS      DOC2
INDIVIDUALS
 T  SQUARE
 F                      V**2
 G  V                   2.0D0*V
 H  V         V         2.0D0
ENDATA
↑↑  ↑    ↑    ↑ ↑    ↑    ↑↑        ↑    ↑        ↑  ↑       ↑    ↑
2 3 5   10   14 15  20   24 25     36   40       49 50      61   65
```

Figure 4.7: SEIF file for the element types for the exampleof Section 2.4

5 The Standard Input Format for Nontrivial Groups

In addition to the problem data and the nonlinear element types described in Section 3 and 4, 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 Section 3. 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.

5.1 Introduction to the Standard Group Type Input Format

5.1.1 The Values and Derivatives Required

It is assumed that a nonlinear group type i s specified in terms of its group-type variable as described on a `GROUP TYPE` data card in an SDIF file, see Section 3.2.17. 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.

5.1.2 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 5.1.

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 Section 4.2.

Keyword	Comments	Presence	Described in §
`GROUPS`	same as `NAME`	mandatory	3.2.1
`TEMPORARIES`		optional	4.4.1
`GLOBALS`		optional	4.4.2
`INDIVIDUALS`		optional	5.2.1
`ENDATA`		mandatory	3.2.2

Figure 5.1: Possible indicator cards

5.1.3 An Example

Before we give the complete syntax for an SGIF file, we finish the illustrative example that we started in Section 3.1.4 and Section 4.3 and show how to specify an input file appropriate for the problem of Section 2.5. The format is fairly similar to that for the SEIF file of Section 4. Once again, there are many possible ways of specifying a particular problem; we give one in Figure 5.2.

line	F.1 <>	Field 2 <—10—>	Field 3 <—10—>	Field 7 <————-41————>
1	GROUPS		EG3	
2	TEMPORARIES			
3	R	TWOP1		
4	INDIVIDUALS			
5	T	PSQUARE		
6	A	TWOP1		2.0*P1
7	F			P1*ALPHA*ALPHA
8	G			TWOP1*ALPHA
9	H			TWOP1
10	ENDATA			
	↑↑↑ 1 2 3	↑ ↑ ↑ 5 10 14	↑ ↑ ↑ 15 20 24	↑ ↑ 25 65

Figure 5.2: SGIF file for the element types for the exampleof Section 2.5

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 consistency checks in the subsequent 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 used; we recall that there is a single nontrivial group type `PSQUARE` and that its attributes (name of group-type variable and parameter) have been described in the SDIF file set up in Section 3.1.4. 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 lines; 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.

5.2 Data Cards

The `GROUPS` and `ENDATA` indicator cards perform the same function as the cards `NAME` and `ENDATA` in Section 3.2.1 and 3.2.2 Likewise, the `TEMPORARIES` and `GLOBALS` data cards have exactly the same syntax as those in Section 4.4.1 and 4.4.2, excepting that the reserved parameters are now the group-type variables specified in the `GROUP TYPE` section of the SDIF file.

5.2.1 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 5.2.1.

<>	<—10—>	<—10—>	<——————-41——————>
F.1	Field 2	Field 3	Field 7
T	gtype-name		
A	p-name		&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&& &&
A+			&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&& &&
I	l-name	p-name	&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&& &&
I+			&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&& &&
E	l-name	p-name	&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&& &&
E+			&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&& &&
F			&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&& &&
F+			&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&& &&
G			&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&& &&
G+			&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&& &&
H			&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&& &&
H+			&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&& &&
↑↑ 2 3	↑ 5 ↑ 10 ↑ 14	↑ 15 ↑ 20 ↑ 24	↑ 25 ↑ 65

Figure 5.3: 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 group type is to be considered. The string `gtype-name` in field 2 gives the name of the group type; the name may be up to ten characters long and must have been defined in the `GROUP TYPE` section of the SDIF file (see Section 3.2.16).

`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 defined; this name must be a valid Fortran name, see Section 3.1.2, and have been previously defined in the `TEMPORARIES` section. The string in field 7 is an arithmetic expression. The assignment

$$\text{auxiliary variable named in field 2} \leftarrow \text{field 7}$$

is made; any variable mentioned in the arithmetic expression must either be reserved (see Section 5.2), 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 `A` card for the current element type in the `INDIVIDUALS` section.

`I` This card announces that an auxiliary parameter, specific to the current group type, is to be assigned a value whenever a second logical auxiliary parameter has the value `.TRUE.` The string, `p-name`, in field 3 gives the name of the auxiliary parameter that is to be defined; this name must be a valid Fortran name, see Section 3.1.2, and have been previously defined in the `TEMPORARIES` section. The string in field 7 is an arithmetic expression. The assignment

$$\text{auxiliary variable named in field 3} \leftarrow \text{field 7}$$

will be made if and only if the logical auxiliary parameter, `l-name`, specified in field 2 has the value `.TRUE.`; the logical parameter must have been previously defined in the `TEMPORARIES` section and allocated a value in the `GLOBALS` or `INDIVIDUALS` section. The arithmetic expression must obey the rules set out in the `A` section above.

`E` This card announces that an auxiliary parameter, specific to the current group type, is to be assigned a value whenever a second logical auxiliary parameter has the value `.FALSE.` The string, `p-name`, in field 3 gives the name of the auxiliary parameter that is to be defined; this name must be a valid Fortran name, see Section 3.1.2, and have been previously defined in the `TEMPORARIES` section. The string in field 7 is an arithmetic expression. The assignment

$$\text{auxiliary variable named in field 3} \leftarrow \text{field 7}$$

will be made if and only if the logical auxiliary parameter, `l-name`, specified in field 2 has the value `.FALSE.`; the logical parameter must have been previously defined in the `TEMPORARIES` section and allocated a value in the `GLOBALS` or `INDIVIDUALS` section. The arithmetic expression must obey the rules set out in the `A` section above.

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

$$\text{nontrivial group function} \leftarrow \text{field 7}$$

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

$$\text{first derivative of group function} \leftarrow \text{field 7}$$

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

$$\text{second derivative of group function} \longleftarrow \text{field 7}$$

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

The data started on an A, I, E, F, G and H card may be continued on a card whose first field contains an A+, I+, E+, 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 5.3. A new group type is deemed to have started whenever a T card is encountered. The F card is compulsory for all group types.

5.3 Two Further Examples

In Section 2.3, we gave an example. An SGIF file for this example is given in Figure 5.4.

```
<>  <--10-->  <--10--><----------------41---------------->
F.1  Field 2   Field 3              Field 7
GROUPS        DOC
TEMPORARIES
 R  ALPHA2
 R  TWO
INDIVIDUALS
 T  GTYPE1
 A  TWO                 2.0D+0
 F                      ALPHA*ALPHA
 G                      TWO*ALPHA
 H                      TWO
 T  GTYPE2
 A  ALPHA2              ALPHA*ALPHA
 F                      ALPHA2*ALPHA2
 G                      4.0*ALPHA2*ALPHA
 H                      12.0*ALPHA2
ENDATA
 ↑↑  ↑    ↑   ↑   ↑    ↑   ↑ ↑                          ↑
 2 3 5   10  14  15   20  24 25                         65
```

Figure 5.4: SGIF file for the nontrivial group types for the exampleof Section 2.3

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 Section 2.4. An SGIF file for this example is given in Figure 5.5 on page 59. 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.

```
<> <--10--> <--10--><-------------41------------->
F.1 Field 2  Field 3            Field 7
GROUPS        DOC2
TEMPORARIES
 R  ISINA
 M  SIN
 M  COS
INDIVIDUALS
 T  SINE
 A  ISINA                P*SIN(ALPHA)
 F                       ISINA
 G                       P*COS(ALPHA)
 H                       -ISINA
ENDATA
↑↑  ↑    ↑   ↑   ↑    ↑   ↑ ↑                     ↑
2 3 5   10  14  15   20  24 25                    65
```

Figure 5.5: SGIF file for the nontrivial group type for the exampleof Section 2.4

6 Free Form Input

So far, we have been quite specific in the format that we allow. In this section, we consider a second format which, though closely connected to the first, allows one to input problems in a less rigid fashion. Although we refer to this second format as *free format*, the freedom really lies in how the data can be laid out in an input file, not in any extra enhancements to the content of a file.

The input style discussed in Sections 3–5 is known as fixed format. Each SDIF/SEIF/SGIF file is assumed to be in fixed format unless otherwise specified. A fixed format file has data arranged in specified fields of given length and normally does not allow for much data on a single card. A free form file, on the other hand, is one where considerable data may be conveyed on a single line. The data does not have to lie in prespecified fields. However, we shall insist that *any free form file can be translated to fixed format and interpreted correctly, in this format, in a single sequential pass through the file.*

We allow a further pair of indicator cards in any SDIF/SEIF/SGIF file. These cards, like those described in Section 3.1, Section 4.2 and Section 5.1.2, contain a single keyword starting in column 1. The new keywords are given in Figure 6.1.

Keyword	Presence
`FREE FORMAT`	optional
`FIXED FORMAT`	optional

Figure 6.1: Additional indicator cards

Any data that lies between a `FREE FORMAT` card and the next `FIXED FORMAT` or `ENDATA` card is considered to be in free format. Likewise, any data that lies between a `FIXED FORMAT` card and the next `FREE FORMAT` or `ENDATA` card is considered to be in fixed format. The file is considered to be in fixed format when the `NAME` (SDIF file), `ELEMENTS` (SEIF file) or `GROUPS` (SGIF file) card is first encountered and thus no initial `FIXED FORMAT` card is required.

Fixed format data is exactly as described in Sections 3–5. The data on a free format data card consists of a number of *strings* separated by *separators*. The characters "`-`", "`;`", "`$`" and " " (blank) are separators and should not therefore be used as significant characters within strings. For example, in free format, `X1;2` will be interpreted as two strings `X1` and `2`. The separators have the following meanings:

(blank) indicates that the previous string has finished and that a new string will follow. One or more blanks is interpreted as a single blank.

`;` indicates that the previous string has finished and that a new string will follow. Moreover, if the file is translated into fixed format, the new string will appear on a new card.

`_` indicates that the previous string has finished and that the next string is empty. Each `-` indicates a separate empty string so that `___` indicates three empty strings.

`$` indicates that the previous string has finished and that the remainder of the card is to be considered as a comment (and thus ignored when the file is interpreted).

A free format card may contain up to 160 characters. On translation into fixed format, a free format card will be divided into one or more fixed format cards depending on how many card separators "`;`" are encountered. Each fixed format card may hold up to six strings; these strings are numbered 1 to 6.

String 1 is examined to see if the first 12 characters identify the card as an indicator. If so, these characters are placed in columns 1 to 12 on the card and the remaining strings discarded. Otherwise, the card is a data card and the first two-characters of string 1, together with the most recently identified indicator card are used to determine the structure of the remainder of the card; two character code must occur as field 1 in the indicated section of Sections 3–5 of this report. The first 2, 10, 10, 12 (41 on some SEIF/SGIF cards), 10, and 12 characters of strings 1–6, respectively, are extracted and placed on a single data card starting in columns 2, 5, 15, 25, 40, and 50, respectively. Left-over parts of strings are discarded. The assembled card is now in fixed format and may be interpreted as such. Thus although a free format card may appear to allow more flexibility, the requirement that the translated card conforms to the fixed input format places considerable responsibility on the user to specify the content of strings correctly.

As an example, a free format variant of the SDIF file given in Figure 3.21 might be:

```
NAME          DOC
FREE FORMAT
GROUPS;E GROUP1;E GROUP2;E GROUP3
VARIABLES;_X1 GROUP1 1.0;_X2 GROUP3 1.0;_X3
BOUNDS;FR BN1 X1;LO BN1 X2 -1.0;LO BN1 X3 1.0
       UP BN1 X2 1.0;UP BN1 X3 2.0
ELEMENT TYPE
EV ETYPE1 V1;EV ETYPE1 V2
EV ETYPE2 V1;EV ETYPE2 V2;EV ETYPE2 V3
IV ETYPE2 U1;IV ETYPE2 U2
ELEMENT USES
T G2E1 ETYPE1;V G2E1 V1_X2;V G2E1 V2_X3
T G3E1 ETYPE2;V G3E1 V1_X2;V G3E1 V2_X1;V G3E1 V3_X3
```

```
T G3E2 ETYPE1;V G2E1 V1_X1;V G2E1 V2_X3
GROUP TYPE;GV GTYPE1 ALPHA;GV GTYPE2 ALPHA
GROUP USES
T GROUP1 GTYPE1; GROUP2 GTYPE2
E GROUP2 G2E1;E GROUP3 G3E1;E GROUP3 G2E2
ENDATA
```

7 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 [1], AMPL [7] and OMP [4]. Such approaches are useful for specifying repetitious structures, but do not really attempt to cope with useful nonlinear structure (like invariant subspaces). Recent work [8] 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 [13] or functional forms [15]. Such forms are the the logical extensions of (2.1) 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.

8 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.

References

[1] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: a User's Guide*. The Scientific Press, Redwood City, USA., 1988.

[2] International Business Machine Corporation. Mathematical programming system/360 version 2, linear and separable programming-user's manual. Technical Report H20-0476-2, IBM Corporation, 1969. MPS Standard.

[3] G. B. Dantzig. *Linear Programming and Extensions*. Princeton Uiversity Press, Princeton, USA, 1963.

[4] D. Decker, F. Louveaux andf G. Mortier, G. Schepens, and A. V. Looveren. *Linear and Mixed Integer Programming with OMP*. Beyers and Partners, Brasschaat, Belgium, 1987.

[5] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, Oxford, England, 1986.

[6] I. S. Duff, Roger G. Grimes, and John G. Lewis. Sparse matrix test problems. *ACM Transactions on Mathematical Software*, 15(1):1–14, 1989.

[7] R. Fourer, D. M. Gay, and B. W. Kernighan. AMPL: A mathematical programming language. Computer science technical report, AT&T Bell Laboratories, Murray Hill, USA, 1987.

[8] 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, Vanvouver, 1989.

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

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

[11] A. Griewank and Ph. L. Toint. On the unconstrained optimization of partially separable functions. In M. J. D. Powell, editor, *Nonlinear Optimization 1981*, pages 301–312, London, 1982. Academic Press.

[12] IBM Optimization Solutions and Library. *QP Solutions User Guide*. IBM Corportation, 1998.

[13] M. Lenard. Standardizing the interface with nonlinear optimizers. talk at the CORS/TIMS/ORSA Meeting, Vancouver, 1989.

[14] I. Maros and C. Meszaros. A repository of convex quadratic programming problems. *Optimization Methods and Software*, 11-12:671–681, 1999.

[15] G. P. McCormick and P. Rahnavard. Representation of unconstrained optimization. talk at the CORS/TIMS/ORSA Meeting, Vancouver, 1989.

[16] D. B. Ponceleón. *Barrier methods for large-scale quadratic programming*. PhD thesis, Department of Computer Science, Stanford University, Stanford, California, USA, 1990.

[17] Oxford University Press. *Dictionary of Computing*. Oxford University Press, Oxford, 1983.