NOP – a compact input format for nonlinear optimization problems

A. Neumaier
Institut für Mathematik, Universität Wien
Strudlhofgasse 4, A-1090 Wien, Austria
WWW: http://solon.cma.univie.ac.at/~neum
email: neum@cma.univie.ac.at

February 1996, revised May 1995

Abstract. This paper defines a compact format for specifying general constrained nonlinear optimization problems. The proposed format is a nonlinear analogue of an explicit representation of sparse matrices by means of index lists and values of the corresponding matrix entries. Thus the format abstracts from the meaning of the problem and hence does not allow names for variables or parameters, but it explicitly displays the internal structure of the problem. This is a very useful feature for global or large scale local optimization.

Key words: large-scale optimization, global optimization, nonlinear programming, test problems, input format

1991 MSC codes: 90C30

1. Introduction

This paper defines a compact format NOP for specifying general constrained nonlinear optimization problems. The proposed format is a nonlinear analogue of an explicit representation of sparse matrices by means of index lists and values of the corresponding matrix entries. Thus the format abstracts from the meaning of the problem and hence does not allow names for variables or parameters, but it explicitly displays the internal structure of the problem. This is a very useful feature for global or large scale local optimization. The structure we capture in the NOP format is precisely that what we found relevant for a global optimization program [5] that is under development in our research group.

In contrast to the SIF input format (cf Section 2.5 below) proposed by Conn, Gould, and Toint [2] for their LANCELOT package, the amount of overhead in the formulation of smaller problems is very slight: For example, Rosenbrock's function (that takes nearly a page to describe in SIF) can be represented in a few lines in such a way that the least squares structure is visible in the representation.

Together with planned interfaces to the optimization package MINOS (Murtagh & Saunders [8]) and a global optimization code based on it, and to the modeling language AMPL (Fourer, Gay & Kernighan [4]) to allow the automatic structuring of input on a higher level, this is a promising tool for the formulation and solution of nonlinear optimization problems.

Each *NOP file* consists of a sequence of *records* describing a constrained optimization problem of the form

$$\begin{gathered}\min \omega x_s \\ \text{s.t. } E_\nu(x), \quad \nu = 1, \ldots, N; \quad x' \le x \le x'',\end{gathered}$$

possibly with additional integer constraints (see **variable declarations**). The bound constraints $x' \le x \le x''$ may have infinite bounds to allow for one-sided bounds and free variables.

The so-called *elements* $E_\nu(x)$ are constraints of the form

$$\sum_k f(a, b_k, x_{J_k}) \in [q] \tag{1}$$

or

$$\sum_k f(a, b_k, x_{J_k}) + b = x_j, \tag{2}$$

and a few irregular variants allowing simple coding of boolean expressions, polynomials, trigonometric polynomials, and a limited form of branching.

Here f is a so-called *element function*, a, b_k, b are parameters or parameter vectors, x_{J_k} is a subvector indexed by the index list J_k, and $[q]$ is a possibly unbounded interval, possibly restricted to integers or with zero adjoined. The contributions $f(a, b_k, x_{J_k})$ are referred to as the *pieces* of the element. (Elements containing a single piece only are, of course, permitted.)

The element functions that we found most useful in coding a large number of problems are listed in Table I and Table II. (The *fill mode*, etc., will be explained later.) For all these functions it is possible to get a complete analytic overview over ranges and inverse ranges, which makes these elements suitable for applications in a branch and bound framework such as that proposed in [5].

Other element functions may be defined by the user, but this may degrade performance of algorithms that require good range information. (It is fully satisfactory, however, for local optimization applications.)

Table I. Predefined elements with regular fill mode. The last column contains the elements that result from an index list of the form $1..m$ (for functions of a single variable) or $1..2m$ (for functions of two variables)

name	element function	fill mode	fixed par.	element shape
sum	x	i	-	$\sum x_i$
lin	px	i	-	$\sum p_i x_i$
abs	$\|x\|$	i	-	$\sum \|x_i\|$
abs2	$p\|x\|$	i	-	$\sum p_i \|x_i\|$
sqr	x^2	i	-	$\sum x_i^2$
qu1	px^2	i	-	$\sum p_i x_i^2$
qu2	$(x-p)^2$	i	-	$\sum (x_i - p_i)^2$
qu3	$p_2(x-p_1)^2$	i	-	$\sum p_{i+m}(x_i - p_i)^2$
qu4	$p_1 x + p_2 x^2$	i	-	$\sum (p_i x_i + p_{i+m} x_i^2)$
pow	x^p	i	p	$\sum x_i^p$
powz	x^p (p integral)	i	p	$\sum x_i^p$
log	$\log \|x-p\|$	i	-	$\sum \log \|x_i - p_i\|$
xlog	$x \log x$	i	-	$\sum x_i \log x_i$
exp	$p_1 e^{-p_2 x}$	i	-	$\sum p_i e^{-p_{i+m} x_i}$
gss	$p_3 e^{-\frac{p_2}{2}(x-p_1)^2}$	i	-	$p_{i+2m} e^{-\frac{p_{i+m}}{2}(x-p_i)^2}$
atan	$\arctan(x)$	i	-	$\sum \arctan(x_i)$
pr0	$x_1 x_2$	p	-	$\sum x_{2i-1} x_{2i}$
pr1	$x_1 x_2$	i	-	$\sum x_i x_{i+m}$
pr2	$(x_1 - p_1)(x_2 - p_2)$	i	-	$\sum (x_i - p_i)(x_{i+m} - p_{i+m})$
div	x_1/x_2	p	-	$\sum x_{2i-1}/x_{2i}$
bil	$x_1 x_2 + p_1 x_1 + p_2 x_2$	i	-	$\sum (x_i x_{i+m} + p_i x_i + p_{i+m} x_{i+m})$
qf1	$p x_1 x_2$	p	-	$\sum p_i x_{2i-1} x_{2i}$
qf2	$p_1 x_1^2 + p_2 x_1 x_2 + p_3 x_2^2$	i	-	$\sum (p_i x_i^2 + p_{i+m} x_i x_{i+m} + p_{i+2m} x_{i+m}^2)$
dsq	$(x_1 - x_2)^2$	i	-	$\sum (x_i - x_{i+m})^2$
atan2	$\arctan(x_1/x_2)$	p	-	$\sum \arctan(x_{2i-1}/x_{2i})$

2. Some examples

Before describing the format in detail, we give some examples for later reference. The first three examples are taken from the global optimization test sets of [6, 10, 3].

We have coded many of the test problems in these collections in order to ensure that our input format is easy to use, and does not require too much repetitive or error-prone adaptation, given the original mathematical description. The corresponding NOP files are publicly

Table II. Interpretation of *index list* and *number list* for predefined elements with irregular fill mode. Constant terms in a **pol**, **sin**, or **cos** series must be specified in *element value*. J is a list of indices; i, j, k, l are single indices.

name	element shape	fill mode
prod	$\prod_{i=1}^{n} x_{j_i}$	$j_1 \ \dots \ j_n$
min	$\min_{i=1}^{n} x_{j_i}$	$j_1 \ \dots \ j_n$
max	$\max_{i=1}^{n} x_{j_i}$	$j_1 \ \dots \ j_n$
max1	$\max_{i=1}^{n} \|x_{j_i}\|$	$j_1 \ \dots \ j_n$
max2	$\max_{i=1}^{n} p_i \|x_{j_i}\|$	$j_1 \ \dots \ j_n; p_1 \ \dots \ p_n$
pol	$\sum_{i=1}^{n} p_i {x_j}^{n+1-i}$	$n \ j; \ p_1 \ \dots \ p_n$
sin	$\sum_{i=1}^{n} a_i \sin(i\omega x_j - p_i)$	$n \ j; \ \omega \ p_1 \ \dots \ p_n \ a_1 \ \dots \ a_n$
cos	$\sum_{i=1}^{n} a_i \cos(i\omega x_j - p_i)$	$n \ j; \ \omega \ p_1 \ \dots \ p_n \ a_1 \ \dots \ a_n$
if1	if $x_i \le p$ then x_j else x_k	$i \ j \ k; \ p$
if2	if $x_i \ge p$ then x_j else x_k	$i \ j \ k; \ p$
if3	$\begin{cases} x_j & \text{if } x_i \le p, \\ x_k & \text{if } p < x_i \le q, \\ x_l & \text{if } x_i > q \end{cases}$	$i \ j \ k \ l; \ p \ q$

available via my global optimization pages on the World Wide Web. The URL of the relevant section is

http://solon.cma.univie.ac.at/~/neum/glopt/nop.html

Since my research group is working on extending the definition of NOP to make it more useful, we invite suggestions and criticism on the present version (which is NOP1.0); please respond by email (my address is neum@cma.univie.ac.at).

2.1. The Rosenbrock function

Our first problem [6] is the minimization of the well-known Rosenbrock function in a box,

$$\begin{array}{ll} \min & 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 \\ \text{s.t.} & x_1, x_2 \in [-2, 8]. \end{array}$$

The least squares structure becomes apparent by introducing the variable

$$x_3 = 10x_1^2 - 10x_2, \tag{3}$$

reducing the objective function to

$$x_4 = x_3^2 + (x_1 - 1)^2. \tag{4}$$

A glance at Table I shows that (3) corresponds to the predefined element function qu4 (with two pieces) and (4) corresponds to the predefined element function qu2 (again with two pieces). If we now count the total number of variables and remember the bounds, we end up with the following NOP file.

```
! Rosenbrock function
min dim4
bnd 1..2 in -2,8
! element list
qu4 1 2; 0 -10 10 0 x3
qu2 3 1; 0 1 x4
```

2.2. Another least squares problem

Our second problem [10] is a least squares problem of Kowalik, with additional bound constraints. The data

i	a_i	b_i^{-1}
1	0.1957	0.25
2	0.1947	0.50
3	0.1735	1.00
4	0.1600	2.00
5	0.0844	4.00
6	0.0627	6.00
7	0.0456	8.00
8	0.0342	10.00
9	0.0323	12.00
10	0.0235	14.00
11	0.0246	16.00

define the following problem

$$\begin{array}{ll} \min & \displaystyle\sum_{i=1}^{11} \left(a_i - x_1 \frac{b_i^2 + b_i x_2}{b_i^2 + b_i x_3 + x_4} \right)^2 \\ \text{s.t.} & x_i \in [0, 0.42] \quad (i = 1,2,3,4), \end{array}$$

To model the least squares terms we introduce the nonstandard elementary function

$$\mathrm{quot}(p, x_1, \ldots, x_4) = \frac{x_1(1 + p x_2)}{1 + p x_3 + p^2 x_4} \tag{5}$$

reducing the objective function to

$$\sum_{i=1}^{11} (a_i - \mathrm{quot}(b_i^{-1}, x_1, \dots, x_4))^2. \tag{6}$$

After introducing new variables for the results of quot (which define elements with a single piece only), (6) reduces further to an element with predefined element function qu2 and 11 pieces. The result is the following NOP file.

```
! Kowalik problem
min dim16
bnd 1..4 in 0,.42
new quot n4 p1
x5=x1*(1+p*x2)/(1+p*(x3+p*x4));end
! element list
quot 1..4; .25 x5
quot 1..4; .50 x6
quot 1..4; 1.0 x7
quot 1..4; 2.0 x8
quot 1..4; 4.0 x9
quot 1..4; 6.0 x10
quot 1..4; 8.0 x11
quot 1..4; 10.0 x12
quot 1..4; 12.0 x13
quot 1..4; 14.0 x14
quot 1..4; 16.0 x15
qu2 5..15; &
.1957 .1947 .1735 .1600 .0844 .0627 &
.0456 .0342 .0323 .0235 .0246 x16
```

With a little more work, we can eliminate the user-defined element function by introducing the constants

$$c_i = b_i^{-1}$$

and writing

$$x_{i+4} = x_1 \frac{b_i^2 + b_i x_2}{b_i^2 + b_i x_3 + x_4} = \frac{x_1(1 + c_i x_2)}{1 + c_i x_3 + c_i^2 x_4}.$$

After multiplication with the denominator, we may write this as

$$x_{i+4} + c_i x_{i+15} = x_1,$$

where

$$x_{i+15} = x_{i+4} x_3 + c_i x_{i+4} x_4 - x_1 x_2.$$

These constraints can be handled with linear and bilinear elements, resulting in the following NOP file.

```
! Kowalik problem
min dim27
bnd 1..4 in 0,.42
con 1..11; .25 .50 1 2 4 6 8 10 12 14 16
! element list
lin  5 16; 1 c1 x1
qf1  5 3  5 4 1 2; 1 c1 -1 x16
lin  6 17; 1 c2 x1
qf1  6 3  6 4 1 2; 1 c2 -1 x17
lin  7 18; 1 c3 x1
qf1  7 3  7 4 1 2; 1 c3 -1 x18
lin  8 19; 1 c4 x1
qf1  8 3  8 4 1 2; 1 c4 -1 x19
lin  9 20; 1 c5 x1
qf1  9 3  9 4 1 2; 1 c5 -1 x20
lin 10 21; 1 c6 x1
qf1 10 3 10 4 1 2; 1 c6 -1 x21
lin 11 22; 1 c7 x1
qf1 11 3 11 4 1 2; 1 c7 -1 x22
lin 12 23; 1 c8 x1
qf1 12 3 12 4 1 2; 1 c8 -1 x23
lin 13 24; 1 c9 x1
qf1 13 3 13 4 1 2; 1 c9 -1 x24
lin 14 25; 1 c10 x1
qf1 14 3 14 4 1 2; 1 c10 -1 x25
lin 15 26; 1 c11 x1
qf1 15 3 15 4 1 2; 1 c11 -1 x26
qu2 5..15; &
.1957 .1947 .1735 .1600 .0844 .0627 &
.0456 .0342 .0323 .0235 .0246 x27
```

2.3. A quadratically constrained nonlinear program

Our third problem [3] is a nonlinear program with bilinear constraints,

$$\begin{array}{ll}
\min & x_1+x_2+x_3 \\
\text{s.t.} & -1+0.0025(x_4+x_6)\le 0, \\
& -1+0.0025(-x_4+x_5+x_7)\le 0, \\
& -1+0.01(-x_5+x_8)\le 0, \\
& 100x_1-x_1x_6+833.33252x_4-8333.333\le 0, \\
& x_2x_4-x_2x_7-1250x_4+1250x_5\le 0, \\
& x_3x_5-x_3x_8-2500x_5+1250000\le 0, \\
& 100\le x_1\le 10000, \\
& 1000\le x_2\le 10000, \\
& 1000\le x_3\le 10000, \\
& 10\le x_4\le 1000, \\
& 10\le x_5\le 1000, \\
& 10\le x_6\le 1000, \\
& 10\le x_7\le 1000, \\
& 10\le x_8\le 1000.
\end{array}$$

We introduce a variable for the objective function,

$$x_9=x_1+x_2+x_3,$$

using an element of the form sum with 3 pieces. The first three constraints (after adding 1 and dividing by the factor, though these cosmetic operations could be dispensed with) become elements of the form lin with 2, 3, and 2 pieces, respectively. The three bilinear constraints could be handled directly using the predefined element bil, but seeing that there are common factors, and the factor x_5-x_8 common to the third and the 6th constraint, we chose to introduce extra variables

$$x_{10}=x_1(x_6-100),\quad x_{11}=x_4-x_7, \tag{7}$$

$$x_{12}=x_2x_{11},\quad x_{13}=x_8-x_5,\quad x_{14}=x_3x_{13} \tag{8}$$

using the elements pr2, lin, and pr0, which leaves after the corresponding substitution only linear constraints coded by the element function lin. The result is the following NOP file. (The third constraint reduces to a bound constraint, and since these must appear before the corresponding variable is used otherwise, the definition of x_{13} follows *after* the bound declaration.)

```
! Hock - Schittkowski Problem 106
! = Floudas - Pardalos Chapter 3, Test Problem 1
min dim14 opt9
bnd 1 in 100,10000
bnd 2 3 in 1000,10000
bnd 4..8 in 10,1000
! element list
sum 1..3 x9 ! objective function
sum 4 6 <= 40
lin 4 5 7; -1 1 1 <= 40
bnd 13 <= 100
lin 5 8; -1 1 x13
pr2 1 6; 0 -100 x10
lin 10 4; -1 833.33252 <= 83333.333
lin 4 7; 1 -1 x11
pr0 2 11 x12
lin 12 4 5; 1 -1250 1250 <= 0
pr0 3 13 x14
lin 14 5; -1 -2500 <= -1250000
```

2.4. A multiphase chemical mixture problem

For maximal efficiency, it is strongly advised to formulate problems in such a way that the element functions depend on few variables only. All significant summations should appear either explicitly or in matrix-vector expressions; the latter should be used only when the matrices involved are essentially dense. The formal increase of dimension resulting from introducing extra variables and equality constraints to achieve this is usually much less relevant than the gain in structural information, even for problems which are originally unconstrained.

We exemplify this by looking at a global optimization problem considered by Mfayokurera [7], describing the mixture of n chemical species at fixed temperature, pressure, and mole number y_i^0, $i = 1, \ldots, n$, separating in up to n phases. (The objective function is based on the NRTL model of Renon & Prausnitz [9] and the Gibbs free energy formulation of Castillo & Grossmann [1].)

$$\begin{array}{ll} \min & \displaystyle\sum_{i,k=1}^{n} y_i^k \log \frac{y_i^k}{\sum_{j=1}^n y_j^k} + \sum_{i,k=1}^{n} y_i^k \frac{\sum_{j=1}^n A_{ji} y_j^k}{\sum_{j=1}^n B_{ji} y_j^k} \\ \text{s.t.} & \sum_{k=1}^n y_i^k = y_i^0 \quad (i = 1, \ldots, n), \\ & 0 \le y_i^k \le y_i^0 \quad (i, k = 1, \ldots, n), \\ & y_1^k \le y_1^{k+1} \quad (k = 1, \ldots, n-1). \end{array} \tag{9}$$

The B_{jk} are nonnegative numbers, and $A_{ii} = 0, B_{ii} = 1$, making the problem well-defined. However, extra care is needed to cope with the limit expressions $0 \log 0$ and $0 \frac{v}{0}$ which may occur.

To obtain an element formulation, we introduce extra variables for the inner sums by adding the constraints

$$u^k := \sum_{j=1}^{n} y_j^k, \tag{10}$$

$$v_i^k := \sum_{j\neq i}^{n} A_{ji} y_j^k, \tag{11}$$

$$w_i^k := \sum_{j\neq i}^{n} B_{ji} y_j^k, \tag{12}$$

and expand the expressions

$$\log \frac{y_i^k}{u^k} = \log y_i^k - \log u_k.$$

If we now introduce the element functions

$$\mathrm{xlog}(y) := \begin{cases} y \log y \text{ if } y > 0, \\ 0 \text{ otherwise} \end{cases} \tag{13}$$

(in fact one of the predefined element functions of Table I) and

$$\mathrm{frac}(x, y, z) := \begin{cases} xy/(y+z) \text{ if } y > 0, \\ 0 \text{ otherwise,} \end{cases} \tag{14}$$

we can write the objective function as a sum of three elements

$$f = \sum_{i,k=1}^{n} \mathrm{xlog}(y_i^k) - \sum_{k=1}^{n} \mathrm{xlog}(u^k) + \sum_{i,k=1}^{n} \mathrm{frac}(v_i^k, y_i^k, w_i^k). \tag{15}$$

Since the first and the last sum in (15) have the same structure, one can improve the formulation further by introducing the element function

$$\mathrm{term}(x, y, z) := \begin{cases} y \log y + xy/(y+z) \text{ if } y > 0, \\ 0 \text{ otherwise} \end{cases} \tag{16}$$

and simplify (15) to a sum of two elements

$$f = -\sum_{k=1}^{n} \mathrm{xlog}(u^k) + \sum_{i,k=1}^{n} \mathrm{term}(v_i^k, y_i^k, w_i^k). \tag{17}$$

It is now easy to code this as a NOP file. Note that since the coefficients A_{ji} and B_{ji} appear in several of the new constraints, they must be predefined as constants. Then one can specify (11) and (12) as elements of type lin.

(16) must be handled by user-defined black box routines (key word **sub** in the function declaration) since branches are not allowed in explicit function declarations. The reason is that branches may invalidate automatic differentiation-like routines used to produce global information about a function on a box.

2.5. Conversion from SIF

The SIF format used as input for the LANCELOT package [2] handles nonlinear programs of the form

$$\begin{aligned} \min\ & \sum_{i\in I_0} g_i\left(\sum_{j\in J_i} w_{ij} f_j(x_{K_j}) + a_i^T x - b_i\right) \\ \text{s.t.}\ & l_i \le x_i \le u_i \quad (1 \le i \le n), \\ & g_i\left(\sum_{j\in J_i} w_{ij} f_j(x_{K_j}) + a_i^T x - b_i\right) \in [v_i, w_i] \quad (i \notin I_0). \end{aligned} \tag{18}$$

The g_i are called group functions, and the arguments of the g_i are the groups. The w_{ij} are weights, the f_j nonlinear element functions depending on subvectors x_{K_j} of x, and the $a_i^T x - b_i$ are referred to as linear element functions. To convert to the NOP format, we introduce extra variables by adding the constraints

$$y_j = f_j(x_{K_j}), \tag{19}$$

$$z_i = \sum_{j\in J_i} w_{ij} y_j + a_i^T x - b_i \tag{20}$$

if $w_{ij} \ne 1$ or $a_i \ne 0$, and

$$z_i = \sum_{j\in J_i} f_j(x_{K_j}) - b_i \tag{21}$$

otherwise. In this way the objective function simplifies to a simple element

$$f = \sum_{i\in I_0} g_i(z_i) \tag{22}$$

(or a sum of several elements if the g_i are different), and the constraints become elements with a single piece only,

$$g_i(z_i) \in [v_i, w_i]. \tag{23}$$

3. External format

Most records start with a keyword that defines the interpretation of records. A line feed or ! ends the record, unless preceded by a continuation mark &. The length of a record is currently constrained by the requirement that it specifies at most 300 indices and at most 300 other numbers.

Only the first 120 characters of a line are significant; moreover, blank lines and text after ! or & is ignored. Single blanks are usually significant; blanks before or after a comma or semicolon are ignored. Multiple blanks are treated as a single blank.

In the following description of records, blanks are indicated by _ (underscore), letters to be written as they stand are in boldface, and names to be substituted are in italic. Parts of records in square brackets [...] are optional and may occur in any order.

There are the following kinds of records:

Problem declaration. This must be the first record of each NOP file, and it occurs nowhere else. It has the form

case [_ **dim***d*] [_ **opt***s*]

where

case, one of **min**, **max**, **all**, defines the type of the optimization problem (find global minimizers, global maximizers, or all feasible points),

d defines the number of variables, and

s defines the index of the variable to be optimized.

The defaults for unspecified options are $d = 1$, $s = d$.

Constant declarations. These are records of the form

con _ *index list*; *number list*

where

index list is a sequence of positive integers separated by blanks, and

number list is a sequence of numbers (or previously defined constants) separated by blanks. Corresponding entries of these lists (which must have the same length) define abbreviations

c*index* = *number*,

valid in all later records.

Indices may have at most 3 digits (at present), and numbers must have standard FORTRAN77 format (integer, single or double precision) and at most 24 characters. A range of consecutive integers can be abbreviated in an index or number list as

first..last

Consecutively numbered constants can be abbreviated as

c*first..last*

An r-fold repetition of a number (or constant) can be abbreviated as

*number****r*
c*index****r*

Variable declarations. Variables with restrictions of their range, in particular bound constraints, must be declared before their first use in a constraint. Undeclared variables get at their first use an implicit declaration as unrestricted.

A variable declaration is a record of the form

bnd _ *index list* [_ *bound*] [_ *restr*]

where

bound is given in one of the forms

= _ *number*
>= _ *number*
<= _ *number*
in _ *number, number*

and specifies the same restriction for all variables in *index list*.

restr, one of **bin** or **int**, specifies for all variables in *index list* a binary constraint (0 or 1; option **bin**) or an integer constraint (integer within *bounds*; option **int**), respectively. If *restr* is missing, all variables in *index list* are constrained to be within *bounds*.

In case of binary constraints, bounds may not be specified.

Function declarations. A number of names for frequently occuring element functions are predefined (see Table I and Table II). These need not (and may not) be declared. (Note that for binary variables, the `min` and `max` elements code for `and` and `or`.) Other functions can be given via subroutines able to perform the tasks needed for the application in question (keyword **sub**), or via the INTCOM interface (keyword **new**); they can be used after their definition.

Table III. The INTCOM syntax

```
only the first 80 characters of a line are significant
! = rest of line is ignored
; and line feed separate statements
no significant blanks, no continuation lines
no distinction is made between lower and upper case
integers have at most 6 digits,
other numbers at most 25 characters
****************** indices *****************************
<digits>.. up to 6 digits
<int>:: <digits>|(<digits>)
***************** variables ****************************
<var>:: x|p|x<int>|p<int>
***************** operators ****************************
<unary>:: -|<up to 4 letters>
<binary>.. up to 4 of the characters +-*/><
           but > or < not leading
***************** constants ****************************
<const>.. length <= 25
    everywhere digits or . or : except leading :
    brackets () [] {} in pairs, but leading only [{
    comma , only within brackets
    d d+ d- e e+ e- when a digit follows immediately
      and some digit (not nec. adjacent) precedes it
    + - only when it is adjacently preceded by one
      of de([{,
***************** expressions **************************
<expr>:: <const>|-<const>|<var>|-<var>|
         (<expr>)|<unary>(<expr>)|<expr><binary><expr>
***************** statements ***************************
<statement>:: begin|end|<var>=<expr>|i=<iexpr>|
              <statement>;|<statement>;<statement>
********************************************************
```

A function declaration is a record of the form

key ␣ *name* [␣ **mod***f*] [␣ **m***m*] [␣ **n***n*] [␣ **p***p*] [␣ **q***q*]
text
...
text

where

key is one of **sub** or **new**,

name is the reference name for the function (at most 24 alphanumerical characters, not one of the standard key words),

f, one of the letters **p** or **i**, defines the fill mode of elements using this function (see **element declarations**),

m is the number of output variables,
n is the number of input variables,
p is the number of piece-dependent parameters, and
q is the number of piece-independent parameters.

The defaults for unspecified options are
$f = \mathbf{p}$, $m = 1, n = 1, p = 0, q = 0$.

When *key* = **new**, a file *name*.syn is created (or overwritten) when reading the NOP file. For *key* = **sub**, it is assumed instead that all information about the function is accessible by a black box routine. (This feature is provided to allow for widest flexibility. The kind of information to be made available in such a subroutine and the calling sequence depends on the application program, and should be documented in the manual of any software using NOP input. For branch and bound applications, the information to be provided by the user in such a subroutine includes, apart from function values and gradients, reasonable outer enclosures of the range, Lipschitz constants, and similar global information. For predefined element functions, this kind of information can be produces automatically.)

text (allowed only when *key* = **new**), defines the lines of *name*.syn, conforming to the INTCOM syntax, described in Table III. In particular, only the first 80 characters of a line are significant, and the last line must terminate with **end**. (INTCOM also allows for nonstandard operators and constants, in case the application knows how to interpret these.) The variables are $x = x1 = x(1),\ x2 = x(2),\ \ldots$, with input variables coming first, output variables next; intermediate (internal) variables must have indices $> m + n$. The parameters are $p = p1 = p(1),\ p2 = p(2),\ \ldots$, with the piece-independent parameters coming first.

Element declarations. These are records of the form
name ␣ *index list* ␣ *element value*
or
name ␣ *index list*; *number list* ␣ *element value*
where
name is the reference name of a function and
element value is either
[␣ *bound*] [␣ *restr*]
or
x*index* [␣ *number*]
specifying, respectively, restrictions for the sum of the pieces (as in **variable declarations** or its assignment to a variable, after incre-

Table IV. Summary of external format

Problem declaration

case [␣ **dim***d*] [␣ **opt***s*]
case is one of **min**, **max**, **all**,
d defines the number of variables,
s defines the index of the variable to be optimized.

Default values: $b = 1$, $d = 1$, $s = d$.

Constant declaration

con ␣ *index list*; *number list*

Abbreviations:

first..last, **c***first..last*
*number****r*, **c***index****r*

Variable declaration

bnd ␣ *index list* [␣ *bound*] [␣ *restr*]
bound is one of
= ␣ *number*
>= ␣ *number*
<= ␣ *number*
in ␣ *number, number*
restr is one of **bin**, **int**

Function declaration

key ␣ *name* [␣ **mod***f*] [␣ **m***m*] [␣ **n***n*] [␣ **p***p*] [␣ **q***q*]

followed by text when *key* is **new**.

key is one of **sub** or **new**,
name is the reference name for the function
f is one of **p** or **i**,
m is the number of output variables,
n is the number of input variables,
p is the number of piece-dependent parameters, and
q is the number of piece-independent parameters.

Default values: $f = \mathbf{p}$, $m = 1, n = 1, p = 0, q = 0$.

Element declaration. One of

name ␣ *index list* ␣ *element value*
name ␣ *index list*; *number list* ␣ *element value*
name is the reference name of a function
element value is of one of the forms
[␣ *bound*] [␣ *restr*]
x*index* [␣ *number list*]

menting it by the number given.

In fill mode **p** or **i**, the lists begin with the piece-independent indices or parameters. An index 0 refers to a dummy variable with constant value 1. The parameters and the indices defining the subvectors for each piece are assigned depending on the fill mode of the function. In fill mode **p**, the lists J_k and b_k in (1) or (2) defining the pieces are filled sequentially *p*iece by *p*iece, while in fill mode **i**, these lists are filled for all pieces in parallel, *i*ndex by *i*ndex. The number of input variables must match the numbers required by the element for a suitable number of pieces.

The predefined elements from Table I are filled in a regular way, as specified in the table. In case of several variables, different fill modes imply a different arrangement of the coefficients. In order that the coding involves no error-prone permutations, we decided on the fill mode for these element functions based on our experience with ease of coding. However, for product elements both fill modes have been found most suitable, depending on the way the problem was given originally; therefore we assigned two different element names `pr0` and `pr1` to the element function x_1x_2, differing only in the fill mode. Some predefined elements have an irregular fill mode; see Table II.

For example, the four records

pr0 1..6 in $1, 2$
pr1 1..6 $= 5$
lin 1 3..10; 1 .5**6 -1 -1 x12 1.5
qu4 1..3; 3 2 3 .5 1 1.5 $>= 0$

define the elements

$x_1x_2 + x_3x_4 + x_5x_6 \in [1, 2]$,
$x_1x_4 + x_2x_5 + x_3x_6 = 5$,
$x_1 + 0.5(x_3 + ... + x_8) - x_9 - x_{10} + 1.5 = x_{12}$,
$3x_1 + 0.5x_1^2 + 2x_2 + x_2^2 + 3x_3 + 1.5x_3^2 \geq 0$,

respectively. (pr0,pr1,lin and qu4 have fill modes **p,i,i,i**; cf. Table I.)

4. Application

We have written a FORTRAN77 program which reads a *NOP file* and translates it into a format suitable for interpretation within other programs. In the resulting internal representation, all elements are converted into equality constraints by adding additional slack variables when necessary. All index ranges, constants, repetitions and zero increments are expanded. The complete problem information (apart from the .syn files) is stored in a common block.

This program is used in a global optimization package GLOPT [5], written in Fortran 77 that is currently under development. GLOPT uses a branch and bound technique to split to problem recursively into subproblems which are either eliminated or reduced in their size. This is done by an extensive use of the block separable structure of the optimization problem, as displayed in the NOP format. The latest news can always be found on my global optimization web pages mentioned at the beginning of Section 2.

Acknowledgment

Acknowledgment. The major part of this work was done while the author was at AT&T Bell Laboratories, Murray Hill, NJ. Later support by the European External Research Program (Contract No. AU-043) by DIGITAL Equipment Corp., Austria, is acknowledged, too.

References

1. J. Castillo and I.E. Grossmann, Computation of phase and chemical equilibria, Computers Chem. Engin. 5 (1981), 99-108.
2. A.R. Conn, N.I.M. Gould, and Ph. L. Toint, LANCELOT. A Fortran Package for large-scale nonlinear optimization. Springer, Berlin 1992.
3. C. A. Floudas and P.M. Pardalos, A collection of test problems for constrained global optimization algorithms, Lecture Notes Comp. Sci. 455, Springer, Berlin 1990.
4. R. Fourer, D.M. Gay, B.W. Kernighan, AMPL. A modeling language for mathematical programming, Scientific Press, San Francisco 1993.
5. S. Dallwig, A. Neumaier and H. Schichl, GLOPT – A Program for Constrained Global Optimization, these proceedings.
6. C. Jansson and O. Knüppel, A global minimization method: The multi-dimensional case, Technical Report 92.1, Hamburg-Harburg 1992. (electronic copy at http://www.ti3.tu-harburg.de/Software.html)
7. A. Mfayokurera, Nonconvex phase equilibria computations by global minimization, M. Sc. Thesis, Dept. of Chem. Engineering, Univ. of Wisconsin, Madison, Wisconsin, 1989.
8. B. A. Murtagh and M.A. Saunders, MINOS 5.1 user's guide, Tech. Report SOL 83-20R, Stanford Univ., Stanford, Calif. 1983, revised 1987.
9. H. Renon and J.M. Prausnitz, Local compositions in thermodynamic excess functions for liquid mixtures, AIChE 14 (1968), 135-144.
10. G. Walster, E. Hansen and S. Sengupta, Test results for a global optimization algorithm, pp.272-287 in: Numerical Optimization 1984 (P.T. Boggs et al., eds), SIAM, Philadelphia 1985.