stichting
mathematisch
centrum

MC

AFDELING MATHEMATISCHE BESLISKUNDE
(DEPARTMENT OF OPERATIONS RESEARCH)

BW 155/82 MAART

F.J. BURGER & J.C.P. BUS

AN ALGOL-68 PACKAGE FOR THE SOLUTION OF SYSTEMS OF NONLINEAR EQUATIONS; USER MANUAL

Preprint

kruislaan 413 1098 SJ amsterdam

Printed at the Mathematical Centre, 413 Kruislaan, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

1980 Mathematics subject classification: 65H10, 68B04.

An ALGOL-68 package for the solution of systems of nonlinear equations; user manual *)

by

F.J. Burger & J.C.P. Bus

ABSTRACT

This report gives a user manual for an ALGOL 68 package for the solution of systems of nonlinear equations. It presents software for systems with and without known analytical derivatives as well as for reduction of systems with linear functions or linear occurring variables.

KEY WORDS & PHRASES: *systems of nonlinear equations, software*

---

*) This report will be submitted for publication elsewhere.

1. INTRODUCTION

This report gives a user manual for an ALGOL-68 package for the solution of systems of nonlinear equations. A theoretical description of this package as well as extensive testing of the algorithms used, is reported in BUS [1980a]. An ALGOL 60 version is described in BUS [1980b]. The package makes use of the programming system for operations on vectors and matrices: TORRIX, given by van der MEULEN & VELDHORST [1978]. Moreover, numerical operators are defined similarly to HEMKER & WINTER [1979]. Due to these choices we are able to present the package in such a way that it can be used with ALGOL 68 as well as FORTRAN routines for the bulk of the numerical algebraic computations. The ALGOL 68 - FORTRAN interface is given in BOS & WINTER [1978] and the package is linked to the NAG software library.

In order to keep this manual clear and handy we have avoided to describe complicated modes as long as they are internally to the package. Furthermore, we give no arguments for the chosen set up of the package, other than the wish to keep close to the various publications referred to above.

Section 2 gives a description of the mathematical problem and a short review of the methods used. Section 3 gives descriptions of the key modes and operators. Here we describe how the various operators can be used to solve a system of nonlinear equations. In Section 4 we give some remarks about the numerical algebra prelude and we specify control commands for the use of the package, including the FORTRAN interface. In Section 5 we list the references. Finally, Section 6 gives an example program and Section 7 the source texts of preludes and routines, as far as these are not published elsewhere.

2. PROBLEM DEFINITION AND METHODS

The general problem of solving a system of nonlinear equations is called *problem* P throughout this manual. If the derivatives are available we call it *problem* Pj. Furthermore, if linear equations occur or some variables appear only linearly we may use reduction techniques to obtain smaller nonlinear problems. In such cases we talk about *problem* P(j)f and P(j)v, respectively. We give precise definitions below.

PROBLEM P is defined by the following data:

1. A nonlinear function F: $D \to \mathbb{R}^n$, where D is some open set in $\mathbb{R}^n$. F as well as D are defined by a routine of mode '*func*', which should be provided by the user.
2. An initial guess $x_0 \in D$ to a solution of the equation $F(x) = 0$. This vector has to be given as the left-hand operand of '*nlssolve*'.
3. Real numbers $\varepsilon_{af} \geq 0$, $\varepsilon_{rf} \geq \varepsilon$ (where ε denotes the *machine precision*) such that

$$\|F(x) - fl_\varepsilon(F(x))\| \leq \varepsilon_{rf}\|F(x)\| + \varepsilon_{af}.$$

   Here $fl_\varepsilon(F(x))$ denotes the value of F(x) computed on the machine with precision ε. ε_{rf} and ε_{af} can be set with the operators '*epsrf*' and '*epsaf*', respectively.
4. Real numbers $\delta_f \geq \varepsilon_{af}$, $\delta_{rx} \geq \varepsilon$ and $\delta_{ax} \geq 0$, which specify the accuracy that is required for a computed approximation $\hat{x}$ to a solution x^* of $F(x) = 0$, in the following sense

   (a) $\quad \|x^* - \hat{x}\| \leq \|\hat{x}\|\delta_{rx} + \delta_{ax}$,

   (b) $\quad \|F(\hat{x})\| \leq \delta_f$.

   Contrary to (b), (a) cannot be guaranteed (see safe stopping at the end of this section). δ_f, δ_{rx} and δ_{ax} can be set by the operators '*dlf*', '*dlrx*' and '*dlax*', respectively.

PROBLEM Pj is defined by the data of problem P together with:

5. A mapping J: $D \to GL(n)$ (the set of $(n \times n)$-matrices), which defines the *jacobian* matrix:

$$J(x) = F'(x) = \left(\frac{\partial F_i(x)}{\partial x_j}\right)_{ij}$$

   of the system of equations $F(x) = 0$. Here $F(x) = (F_1(x), \ldots, F_n(x))^T$. J is defined by a routine of mode '*jacob*', which has to be provided by the user.
6. Real numbers $\varepsilon_{aj} \geq 0$, $\varepsilon_{rj} \geq \varepsilon$ such that

$$\|J(x)-fl_\varepsilon(J(x))\| \le \varepsilon_{rj}\|J(x)\| + \varepsilon_{aj}.$$

ε_{rj} and ε_{aj} can be set with the operators '*epsrj*' and '*epsaj*', respectively.

Sometimes, some equations of the system are linear in the variables. One can take advantage of this property by reducing the problem to a smaller system of nonlinear equations only. Suppose, after reordering, the function F can be given by

$$F(x) = \begin{pmatrix} G(x) \\ Ax - b \end{pmatrix}, \tag{2.1}$$

where G: $D \to \mathbb{R}^{n-m}$ is nonlinear, A is an $(m\times n)$-matrix, b is an m-vector and $m < n$ is the number of linear equations: $Ax - b = 0$.

PROBLEM P(j)f is defined by the data of problem P (or Pj), where F: $D \to \mathbb{R}^n$ is replaced by G: $D \to \mathbb{R}^{n-m}$ (note that J(x) is an $((n-m)\times n)$-matrix now), together with:

7. An $(m\times n)$-matrix representing A in (2.1). This matrix can be specified by the operator '*linmat*'.
8. An m-vector b representing b in (2.1). This vector can be specified by the operator '*linvec*'.

Another possibility for reduction of the problem to a smaller nonlinear problem occurs if some variables occur only linearly. In that case, after reordering of the variables, we can write the function F as

$$F(x) = Ay + G(z), \tag{2.2}$$

where G: $\bar{D} \to \mathbb{R}^n$ with $\bar{D}$ the projection of $D \subset \mathbb{R}^n$ to $\mathbb{R}^{n-m}$, m the number of linear variables and $x = \binom{y}{z}$, $y \in \mathbb{R}^m$ represents the linear variables and $z \in \mathbb{R}^{n-m}$ represents the nonlinear variables.

PROBLEM P(j)v is defined by the data of problem P (or Pj), where F is replaced by G: $\bar{D} \to \mathbb{R}^n$ (note that J(x) is a $(n\times(n-m))$-matrix), together with:

9. An (n×m)-matrix representing A in (2.2). This matrix can be specified by the operator '*linmat*'.

REMARKS.

1. It is recommended to provide the jacobian if it is not too difficult to program the analytic expressions for the jacobian elements. It increases efficiency of the methods if one evaluation of the jacobian is cheaper than n evaluations of the function. And, usually, it makes higher accuracy possible.
   One should never provide a numerically approximated jacobian. In such a case one should use a method which has an approximation method built in.
2. We demand the matrix A in problems P(j)f and P(j)v to be full rank.
3. It is not possible to take advantage from both linear equations and variables together.
4. Although the definition of the problem requires specification of all data, the programming package provides default specifications for most quantities. It is recommended, but not necessary, that the user defines his own specifications.

The methods used in this package are all Newton-like methods. We now give a listing of methods and features which are available.

A. If the jacobian subroutine is provided.
   1. *Newton*: a restrained Newton-like method using bisection for step length control and gaussian elimination for the solution of the linear system in each step.
      Specification with '*method*' *newton*.
   2. *Generalized Newton*: a strict Newton-like method (no step length control) using singular value decomposition to solve (possibly singular) linear systems in each step.
      Specification with '*method*' *general newton*.
   3. *Poly Newton*: Newton and, if it fails, subsequently generalized Newton.
      Specification with '*method*' *poly newton*.

B. No jacobian subroutine provided.

1. *Difference Newton*: as Newton, but with the jacobian approximated with forward differences.
   Specification with '*method*' *diff newton*.
2. *Generalized difference Newton*: as Generalized Newton, but with the jacobian approximated with forward differences.
   Specification with '*method*' *general diff newton*.
3. *Poly difference Newton*: Difference Newton and, if it fails, subsequently generalized difference Newton.
   Specification with '*method*' *poly diff newton*.

The methods given above are described in detail in BUS [1980a] with names: AB, GAS, SNOLEQJ, DB, GDS and SNOLEQ, successively. Together with the algorithms the following features (also described in the given reference) are available in the package.

F.1. Automatic scaling

Functions and variables are scaled before the iterative process is started, in such a way that row and column norms of the (approximated) jacobian at the initial point are in the intervals $[\frac{1}{4}n, \sqrt{n}]$ and $[\frac{1}{2}, 1]$, respectively. Automatic scaling can be performed using operator '*scale*'. It is only effectuated in (difference) Newton and if the user explicitly specifies it. It may have unpleasant side effects, e.g. the required precisions are attained for the scaled problem, and strong nonlinearity may cause scaling at the initial point to be useless or undesirable near a solution but it is not adapted.

F.2. Conditional updating of the (approximated) jacobian

If, based on certain conditions, it appears to be safe to approximate the jacobian in a given step by simply updating the (approximate) jacobian of the last step, one can allow the methods (difference) Newton to do so. If one does not want this feature, one has to specify this with operator '*updok*'. This feature saves evaluation of the jacobian (in Newton) and of the function (in difference Newton) and is worthwhile if such evaluations are expensive. It is undesirable if very high accuracy is required.

F.3. Safe stopping

In (difference) Newton we use stopping criteria which reasonably guarantee the achievement of the required accuracy defined in P.4. It may save iteration steps when solving easy problems. For difficult problems, particularly if the jacobian has bad condition at the solution, the approximated solution will be more precise than using the standard criterion. Sometimes an error message is given due to the fact that the safe criterion can not be satisfied. The standard criterion is

$$\|x_{k+1}-x_k\| \leq \|x_{k+1}\|\delta_{rx} + \delta_{ax}$$

with k the iteration index. This criterion is used in generalized (difference) Newton. If the safe stopping criterion is not wanted then one can specify this using operator *'safe'*.

3. DESCRIPTION OF MODES AND OPERATORS

We first describe the important modes in the package. In this section (P..) means reference to part of the problem definitions in Section 2.

'mode' 'func' = 'proc' ('vec','vec','ref' 'bool') 'void' ;

This is the mode of the problem function subroutine (P.1).
For a given *'func' funrout*, the call *funrout(x,f,in)* should result in:
- if x ∉ D then *in* = *'false'*
- if x ∈ D then *in* = *'true'* and f = F(x).

'mode' 'jacob' = 'proc' ('vec','mat','ref' 'bool') 'void' ;

This is the mode of the jacobian subroutine (Pj.5).
For a given *'jacob' jac* the call *jac(x,b,in)* should result in
- if x ∉ D then *in* = *'false'*
- if x ∈ D then *in* = *'true'* and b = J(x).

```
'mode' 'info' = 'struct' ('real' epsrf, epsaf, epsrj, epsaj, dlf, dlrx,
                               dlax, nrmf, nrmdx,
                          'int' ctit, ctfu, ctjc, ctlu, ctsv, er,
                          'bool' init, 'vec' f, 'mat' b),

        'reduinfo' = 'struct' ('mat' la, 'vec' lb),

        'scalinfo' = 'struct' ('bool' sing, fscal, xscal, 'vec' ffacs,
                     xfacs, 'proc'('vec','info','ref''scalinfo') 'void'
                     scale, bckscale);
```

These modes provide process information structures for the general problem, for the case that reduction is possible due to linear equations, and for the case that scaling is performed, respectively.
The fields have the following meaning:

epsrf : ε_{rf} (P.3), set by *'epsrf'*;
epsaf : ε_{af} (P.3), set by *'epsaf'*;
epsrj : ε_{rj} (Pj.6), set by *'epsrj'*;
epsaj : ε_{aj} (Pj.6), set by *'epsaj'*;
dlf : δ_{f} (P.4), set by *'dlf'*;
dlrx : δ_{rx} (P.4), set by *'dlrx'*;
dlax : δ_{ax} (P.4), set by *'dlax'*;
nrmf : $\|F(x)\|$, with x the current iterate during the iterative process;
nrmdx : $\|x-x_-\|$, with x the current iterate and x_- its predecessor;
ctit : iteration counter;
ctfu : function evaluation counter;
ctjc : jacobian evaluation counter;
ctlu : lu-decomposition (gaussian elimination) counter;
ctsv : singular value decomposition counter;
er : error report;
init : indicates whether F(x) (and J(x)) are given in f (and b);
f : F(x);
b : J(x) or its approximation;
la : the matrix A defining either linear equations or linear variables (Pf.7, Pv.9);

lb : the vector b in case of linear equations (Pf.8);
sing : indicates singularity of the jacobian (approximation) at the initial point if scaling is performed;
fscal : indicates whether the equations are scaled;
xscal : indicates whether the variables are scaled;
ffacs : scaling factors for equations, if scaled;
xfacs : scaling factors for variables, if scaled;
scale : the routine performing scaling, which is dummy if scaling is not allowed;
bckscale : the routine performing rescaling (dummy if scaling not allowed).

```
'mode' 'nlsprb'    = 'ref' 'nlsprob',
       'nlsprob'   = 'struct' ('vec' x, 'func' fun, 'ref' 'info' info,
                     'ref' 'reduinfo' rinfo, 'ref' 'scalinfo' sinfo),
       'nlsprbjc'  = 'ref' 'nlsprobjc',
       'nlsprobjc' = 'struct' ('vec' x, 'func' fun, 'jacob' jac,
                     'ref' 'info' info,
                     'ref' 'reduinfo' rinfo, 'ref' 'scalinfo' sinfo);
```

These modes define the basic information structure for a problem P, Pj, P(j)f or P(j)v, together with some process information (the first two for P(f)(v), the last two for Pj(f)(v)).
For the fields *fun*, *jac*, *info*, *rinfo* and *sinfo* we refer to the description of their modes above.

x : gives the initial guess to the solution before entry of the iterative process; the current iterate during the process; the approximate solution after the process.

```
'mode' 'metinfo'   = 'struct'('proc'('metnls') 'vec' nlssolver,
                     'bool' print, safe, updok, 'int' maxits),
       'metnls'    = 'struct' ('nlsprb' nlp, 'ref' 'metinfo' mi),
       'metinfojc' = 'struct'('proc'('metnlsjc') 'vec'
                     nlssolver, 'bool' print, safe, updok,'int' maxits),
       'metnlsjc'  = 'struct' ('nlsprbjc' nlp, 'ref' 'metinfojc' mi);
```

'metinfo' (*'metinfojc'*) specify the method to be used for solving a problem (with specified jacobian).
'metnls' (*'metnlsjc'*) combine the problem information in *'nlsprb'* (*'nlsprbjc'*) with the information about the method in *'metinfo'* (*'metinfojc'*). The description of the field *nlp* is given above with the description of its mode. The other fields have the following meaning:

nlssolver: the routine to solve the problem;
print : indicates whether printing of intermediate results is required. Initial and final information, together with one line per step is produced. *print* is set by *'print'*;
safe : indicates whether a safe stopping criterion has to be used (see Section 2, feature 3); *safe* is set by *'safe'*;
updok : indicates whether conditional updating is allowed (see Section 2, feature 2); *updok* is set by *'updok'*;
maxits : the maximum number of iteration steps allowed during the solution process; *maxits* is set by *'maxits'*.

The dyadic operator that activates the solution process is: *'nlssolve'*. Given an initial guess in *'vec'* *x* and a problem and method specification in *'metnls'* (*'metnlsjc'*) *ml* we activate the solution of the given problem by the given method with the call

x 'nlssolve' ml.

We have given a table of the operators to be used to specify values for the various fields of the data structures with their default values. Most fields are identified with the same name (without quotes) as the operator, except for *'linmat'* (field *la*), *'linvec'* (field *lb*) and *'scale'* (no field in a structure, yields setting of scaling routine).

Table 1

operator	mode right hand side operand	default value	meaning	remarks
'epsrf'	*'real'*	ε^*	ε_{rf}	see P.3
'epsaf'	*'real'*	ε	ε_{af}	see P.3
'epsrj'	*'real'*	ε	ε_{rj}	see Pj.6
'epsaj'	*'real'*	ε	ε_{aj}	see Pj.6
'dlf'	*'real'*	$\sqrt{\varepsilon}$	δ_f	see P.4
'dlrx'	*'real'*	$\sqrt{\varepsilon}$	δ_{rx}	see P.4
'dlax'	*'real'*	$\sqrt{\varepsilon}$	δ_{ax}	see P.4
'scale'	*'bool'*	*'false'*	automatic scaling permitted?	ignored for generalized methods
'linmat'	*'mat'*	*'nil'*	matrix A for linear equations or variables	see Pf, Pv; no combination of Pf and Pv
'linvec'	*'vec'*	*'nil'*	vector b for linear equations	see Pf.
'print'	*'bool'*	*'true'*	printing required?	see program example, Section 5
'safe'	*'bool'*	*'true'/ 'false'*	safe stopping criterion required?	default *'true'* for (difference) Newton, *'false'* otherwise; see Sect.2, feat. 3.
'updok'	*'bool'*	*'true'/ 'false'*	conditional updating required?	default *'true'* for (difference) Newton, *'false'* otherwise; see Sect.2, feat. 2
'maxits'	*'int'*	40/80	max. nb. of iteration steps allowed	default 80 for polymethods, otherwise 40

* ε = small real

All operators have as possible left hand operands variables of mode: *'func'*, *'nlsprb'*, *'nlsprbjc'*. Moreover, for *'print'* up to *'maxits'* we can also have *'metnls'* or *'metnlsjc'*

The operators *'epsrf'* up to *'linvec'* produce a *'nlsprb'* (*'nlsprbjc'*); the other operators produce a *'metnls'* (*'metnlsjc'*).

Finally, we give short descriptions of possible constructions of objects of the various modes.

A. Jacobian routine is not specified

Given a function routine *'func' funrout*.

'nlsprb' nlp := 'setnls' funrout

creates an object of mode *'nlsprb'* yielding default values in all fields of *nlp* except for *fun 'of' nlp* which is set to *funrout*.
Another method, which enables specification of other fields at the same statement is:

'nlsprb' nlp := funrout[*'epsrf' erf*][*'epsaf' eaf*]
[*'epsrj' erj*][*'epsaj' eaj*][*'dlf' df*][*'dlrx' drx*]
[*'dlax' dax*][*'scale' b1*][*'linmat' la*][*'linvec' b*];

where [ ] means that the operator and operand within brackets is optional, although at least one has to be specified; no such specification means default value.

After such a creation of *nlp* the value of a field can be changed by using the appropriate operator (see Table 1).

An object of mode *'metnls'* can subsequently be created by:

$$\textit{'metnls' ml := nlp}\left[\textit{'method'}\ \begin{Bmatrix}\textit{diff newton}\\ \textit{general diff newton}\\ \textit{poly diff newton}\end{Bmatrix}\right]$$
[*'print' pr*][*'safe' b1*][*'updok' b2*][*'maxits' mx*];

Again [ ] means optional (no specification yields default value), although at least one has to be specified. Furthermore, $\begin{Bmatrix}a\\b\\c\end{Bmatrix}$ means that a choice from a, b and c has to be made. If *'method'* is not specified, then the default method *poly diff newton* is chosen (diff is shortation for difference, see Section 2).

Finally, if the initial vector of variables *'vec' x* is given, the solution process can be activated by one of the following calls:

x 'nlssolve' funrout (all fields default, except *fun*, default method: *poly diff newton*);

x 'nlssolve' nlp (default method: *poly diff newton*, fields: *print*, *safe*, *updok* and *maxits* default);

x 'nlssolve' ml.

B. Jacobian routine specified

Given a function routine *'func' funrout* and a jacobian routine *'jacob' jacrout*.

'nlsprbjc' nlpj := funrout 'jacobian' jacrout;

creates an object of mode *'nlsprbjc'* yielding default values in all fields except for *fun (:= funrout)* and *jac (:= jacrout)*. Specification of certain fields can be performed by applying an appropriate operator on *nlpj* (see Table 1).

An object of mode *'metnlsjc'* can subsequently be created by

$$\textit{'metnlsjc' mlj} := \textit{nlpj}\left[\textit{'method'}\left\{\begin{matrix}\textit{newton}\\ \textit{general newton}\\ \textit{poly newton}\end{matrix}\right\}\right]$$
$$\textit{['print' pr]['safe' b1]['updok' b2]['maxits' mx];}$$

where again [ ] means optional, with default value if deleted, the default method is *poly newton* (see Section 2). At least one of the operand specifications has to be given.

Given the initial vector of variables *'vec' x*, the solution process can be activated by one of the following calls:

x 'nlssolve' nlpj (default method and fields: *print*, *safe*, *updok and maxits*);

x 'nlssolve' mlj.

4. NUMERICAL ALGEBRA PRELUDE AND FORTRAN LINK

The numerical algebra prelude and library routines are based on the matrix-vector programming system TORRIX (see van der MEULEN & VELDHORST [1978]) (For references see Section 5). We used a small subset of the modes and operators described in HEMKER & WINTER [1979]. These are equivalent to those used in AFLINK (BOS & WINTER [1978]) which enables us to use the NAG-FORTRAN software library for the bulk of the numerical algebra computations. (There is one difference concerning the operator *'trims'*; we use our own version and define it in the nonlinear system prelude, so that use of the AFLINK operator *'trims'* is avoided). The use of FORTRAN software for matrix decomposition may yield attractive CPU-time savings, particularly for nasty and large problems. We shall not give the source texts of the ALGOL 68 numerical algebra routines in Section 6. Except for some removed errors these can be found in BUS [1980a].
Hence, the system can be used with either ALGOL 68 matrix decomposition routines or FORTRAN (NAG) matrix decomposition routines. The NOS/BE control commands which has to be used are:

```
BEGIN,INIT,,NLS(,FTN).
A68,I = PROGRAM,P = NLSLIB/NLSPRL.
LGO.
```

If FTN is specified then the FORTRAN-NAG link is used.

5. REFERENCES

BOS, H.J. & D.T. WINTER [1978], *AFLINK: A new ALGOL 68-FORTRAN interface*, NN 17/78, Mathematisch Centrum, Amsterdam.

BUS, J.C.P. [1980a], *Numerical solution of systems of nonlinear equations*, Tract 122, Mathematisch Centrum, Amsterdam.

BUS, J.C.P. [1980b], *An ALGOL 60 package for the solution of nonlinear equations*, NW 88/80, Mathematisch Centrum, Amsterdam.

HEMKER, P.W. & D.T. WINTER [1979], *A preliminary report on numerical operators in ALGOL 68*, NW 66/79, Mathematisch Centrum, Amsterdam.

MEULEN, S.G. van der & M. VELDHORST [1978], *TORRIX, A programming system for operations on vectors and matrices over arbitrary fields and of variable size*, Vol. I, Tract 86, Mathematisch Centrum, Amsterdam.

NAG, Numerical Algorithms Group, Software library reference manual.

WIJNGAARDEN, A. van, et al. (Eds) [1976], *Revised report on the algorithmic language ALGOL 68*, Tract 50, Mathematisch Centrum, Amsterdam.

6. PROGRAM EXAMPLE

```
'begin'

    'op' 'pi' = ('vec' x) 'real' :
    'begin' 'real' res:= 1;
       'for' i 'to' 'upb' x 'do' res *:= x[i] 'od'; res
    'end';

    'func' problem one red = ('vec' x,f, 'ref' 'bool' ok) 'void' :
    'begin' ok:= 'true';  f[1]:= - 1 + 'pi' x 'end';

    'func' problem one = ('vec' x,f, 'ref' 'bool' ok) 'void' :
    'begin' ok:= 'true';  f[1]:= - 1 + 'pi' x;
       'real' sum:= 0; 'int' n = 'upb' x;
       'for' i 'to' n 'do' sum+:= x[i] 'od';
       'for' i 'from' 2 'to' n 'do' f[i]:= -(n+1) + x[i] + sum 'od'
    'end';

    'jacob' jac one red = ('vec' x, 'mat' b, 'ref''bool' ok) 'void' :
    'begin' 'real' pi; ok:= (pi:= 'pi' x) /= 0;
       'if' ok
       'then' 'for' i 'to' n 'do' b[1,i]:= pi/x[i] 'od' 'fi'
    'end';

    'jacob' jac one = ('vec' x, 'mat' b, 'ref''bool' ok) 'void' :
    'begin' 'real' pi; ok:= (pi:= 'pi' x) /= 0;
       'if' ok
       'then' 'for' i 'to' n 'do' b[1,i]:= pi/x[i] 'od';
          b[2:n,1:n]:= 1 'into' genmat(n-1, n);
          'for' i 'from' 2 'to' n 'do' b[i,i]:= 2 'od'
       'fi'
    'end';

    'int' n = 10; 'real' eps = 1e-7, epsrf = n * small real;
    'real' epsaf = epsrf;
    'vec' x:= 0.5 'into' genvec(n);
    print((newpage,"problem characteristics",newline,
           "======================",newline,newline,
           "dimension ",whole('upb' x,-2),newline,
           "rel. prec. ",float(epsrf,-9,2,4),newline,
           "abs. prec. ",float(epsaf,-9,2,4),newline));
    outvec(stand out,x,6,3,"initial guess");

    print((newline,newline,newline,newline, "method : poly newton"));
    x 'nlssolve' problem one 'jacobian' jac one 'epsrf' epsrf
    'epsaf' epsaf 'dlf' eps 'dlrx' eps 'dlax' eps 'maxits' 30;

    x:= 0.5 'into' genvec(n);
    print((newpage, "problem as above",newline,newline,newline,
           "method : poly newton with reduction"));
    'mat' la:= 1 'into' genmat(n - 1,n);
```

```
        'for' i 'to' n - 1 'do' la[i,i + 1]:= 2 'od';
        'vec' lb:= (n + 1) 'into' genvec(n - 1);
        x 'nlssolve' problem one red 'jacobian' jac one red 'epsrf' epsrf
        'epsaf' epsaf 'dlf' eps 'dlrx' eps 'dlax' eps 'linmat' la
        'linvec' lb 'maxits' 20;

        x:= 0.5 'into' genvec(n);
        print((newpage,"problem as above with default precisions",newline,
              newline,newline, "method : general diff newton"));
        outvec(stand out,x 'nlssolve' problem one
              'method' general diff newton 'maxits' 20,6,3,"solution")
'end'
```

problem characteristics

dimension 10
rel. prec. 7.11e -14
abs. prec. 7.11e -14

initial guess

1	+5.000000e -1	5	+5.000000e -1	9	+5.000000e -1
2	+5.000000e -1	6	+5.000000e -1	10	+5.000000e -1
3	+5.000000e -1	7	+5.000000e -1		
4	+5.000000e -1	8	+5.000000e -1		

method : poly newton

start of iteration, no scaling performed

its	fus	jcs	norm(f)	labda	omega	beta	kappa	nrmjacinv	errjac	dfstep
0	1	1	.165e +2	.1e +1	.1e +1	.0e +1	.1e +1	.1e +1	.0e +1	.1e+1
1	11	1	.165e +2	.2e -2	.1e +1	.5e +4	.6e +3	.9e +3	.2e -8	.1e+1
2	13	2	.126e +2	.5e +0	.2e +2	.3e +2	.3e +1	.6e +1	.1e-10	.1e+1
3	14	3	.115e +1	.1e +1	.8e +1	.3e +1	.2e +1	.9e +0	.8e-11	.1e+1
4	15	4	.345e +0	.1e +1	.3e +0	.5e +0	.1e +1	.9e +0	.3e-11	.1e+1
5	16	5	.976e -1	.1e +1	.3e +1	.3e +0	.2e +1	.9e +0	.2e-11	.1e+1
6	17	6	.256e -1	.1e +1	.4e +1	.2e +0	.4e +1	.9e +0	.2e-11	.1e+1
7	18	7	.574e -2	.1e +1	.5e +1	.1e +0	.8e +1	.9e +0	.2e-11	.1e+1
8	19	8	.818e -3	.1e +1	.8e +1	.4e -1	.1e +2	.1e +1	.2e-11	.1e+1
9	20	9	.309e -4	.1e +1	.1e +2	.8e -2	.2e +2	.1e +1	.2e-11	.1e+1
10	21	10	.513e -7	.1e +1	.1e +2	.3e -3	.2e +2	.1e +1	.3e-11	.1e+1
11	22	10	.853e -10	.1e +1	.5e +1	.5e -6	.2e +2	.1e +1	.8e -2	.1e+1

end of iteration
iteration successful

approx. solution

1	+1.0000000e +0	5	+1.0000000e +0	9	+1.0000000e +0
2	+1.0000000e +0	6	+1.0000000e +0	10	+1.0000000e +0
3	+1.0000000e +0	7	+1.0000000e +0		
4	+1.0000000e +0	8	+1.0000000e +0		

function

1	-8.534329e -11	5	-5.684342e -14	9	-5.684342e -14
2	-5.684342e -14	6	-5.684342e -14	10	-5.684342e -14
3	-5.684342e -14	7	-5.684342e -14		
4	-1.136868e -13	8	-5.684342e -14		

problem as above

method : poly newton with reduction

start of iteration, no scaling performed

its	fus	jcs	norm(f)	labda	omega	beta	kappa	nrmjacinv	errjac	dfstep
0	1	1	.575e -2	.1e +1	.1e +1	.0e +1	.1e +1	.1e +1	.0e +1	.1e+1
1	2	1	.820e -3	.1e +1	.1e +1	.4e -1	.1e +1	.7e +1	.2e-12	.1e+1
2	3	2	.311e -4	.1e +1	.1e +2	.8e -2	.1e +1	.1e +2	.2e-12	.1e+1
3	4	3	.519e -7	.1e +1	.1e +2	.3e -3	.1e +1	.1e +2	.2e-12	.1e+1
4	5	3	.868e -10	.1e +1	.5e +1	.5e -6	.1e +1	.1e +2	.8e -2	.1e+1

end of iteration
iteration successful

approx. solution

1	+1.0000000e +0	5	+1.0000000e +0	9	+1.0000000e +0
2	+1.0000000e +0	6	+1.0000000e +0	10	+1.0000000e +0
3	+1.0000000e +0	7	+1.0000000e +0		
4	+1.0000000e +0	8	+1.0000000e +0		

function

1	-8.684964e -11	5	-7.389644e -13	9	-6.821210e -13
2	-7.958079e -13	6	-7.958079e -13	10	-6.252776e -13
3	-9.094947e -13	7	-7.389644e -13		
4	-7.958079e -13	8	-7.389644e -13		

problem as above with default precisions

method : general diff newton

start of iteration, no scaling performed

its	fus	jcs	norm(f)	labda	omega	beta	kappa	nrmjacinv	errjac	dfstep
0	11	0	.165e +2	.1e +1	.1e +1	.0e +1	.1e +1	.1e +1	.0e +1	.7e-6
1	12	0	.109e +29	.1e +1	.1e +1	.5e +4	.3e +4	.5e +4	.2e -1	.7e-6
2	23	0	.378e +28	.1e +1	.1e +15	.2e +3	.1e +1	.2e -25	.6e -8	.1e+1
3	34	0	.131e +28	.1e +1	.1e +15	.1e +3	.1e +1	.4e -25	.4e-10	.1e-2
4	45	0	.454e +27	.1e +1	.1e +15	.1e +3	.1e +1	.1e -24	.6e-10	.9e-3
5	56	0	.157e +27	.1e +1	.1e +15	.1e +3	.1e +1	.3e -24	.1e -9	.5e-3
6	67	0	.545e +26	.1e +1	.1e +15	.1e +3	.1e +1	.7e -24	.2e -9	.3e-3
7	78	0	.189e +26	.1e +1	.1e +15	.9e +2	.1e +1	.2e -23	.3e -9	.2e-3
8	89	0	.654e +25	.1e +1	.1e +15	.8e +2	.1e +1	.4e -23	.4e -9	.1e-3
9	100	0	.227e +25	.1e +1	.1e +15	.7e +2	.1e +1	.1e -22	.7e -9	.8e-4
10	111	0	.785e +24	.1e +1	.1e +15	.7e +2	.1e +1	.3e -22	.1e -8	.5e-4
11	122	0	.272e +24	.1e +1	.1e +15	.6e +2	.1e +1	.7e -22	.2e -8	.3e-4
12	133	0	.943e +23	.1e +1	.1e +15	.5e +2	.1e +1	.2e -21	.3e -8	.2e-4
13	144	0	.327e +23	.1e +1	.1e +15	.5e +2	.1e +1	.5e -21	.4e -8	.1e-4
14	155	0	.113e +23	.1e +1	.1e +15	.4e +2	.1e +1	.1e -20	.7e -8	.8e-5
15	166	0	.392e +22	.1e +1	.1e +15	.4e +2	.1e +1	.3e -20	.1e -7	.5e-5
16	177	0	.136e +22	.1e +1	.1e +15	.3e +2	.1e +1	.8e -20	.2e -7	.3e-5
17	188	0	.470e +21	.1e +1	.1e +15	.3e +2	.1e +1	.2e -19	.3e -7	.2e-5
18	199	0	.163e +21	.1e +1	.1e +15	.3e +2	.1e +1	.5e -19	.5e -7	.1e-5
19	210	0	.565e +20	.1e +1	.1e +15	.2e +2	.1e +1	.1e -18	.7e -7	.8e-6
20	221	0	.196e +20	.1e +1	.1e +15	.2e +2	.1e +1	.4e -18	.1e -6	.5e-6

end of iteration

approx. solution

1	+5.03953864e +3	5	-5.39643839e +1	9	-5.39643840e +1
2	-5.39643840e +1	6	-5.39643840e +1	10	-5.39643839e +1
3	-5.39643840e +1	7	-5.39643839e +1		
4	-5.39643839e +1	8	-5.39643839e +1		

function

1	-1.955941e +19	5	+4.488895e +3	9	+4.488895e +3
2	+4.488895e +3	6	+4.488895e +3	10	+4.488895e +3
3	+4.488895e +3	7	+4.488895e +3		
4	+4.488895e +3	8	+4.488895e +3		

* *
* *failure of iteration : too many function evaluations or iterations required* *
* *

solution

1	+5.039539e +3	5	-5.396438e +1	9	-5.396438e +1
2	-5.396438e +1	6	-5.396438e +1	10	-5.396438e +1
3	-5.396438e +1	7	-5.396438e +1		
4	-5.396438e +1	8	-5.396438e +1		

7. SOURCE TEXTS

naprel:

```
# numerical algebra prelude,
  j.c.p. bus, update 811028,
  to be compiled by: a68,i=lfn,p=numal3/tormin,n.
  where tormin is an optimized version of the torrix basis prelude
  (see meulen en veldhorst [1978])
#

'begin'

  'mode' 'prb' = 'struct'('real' relacc, absacc, reltol, abstol,
                          'int' maxit);
  'mode' 'prob' = 'ref''prb';
  'mode' 'matprob' = 'struct'('mat' mat, 'prob' prob),
         'lud' = 'struct'('mat' mat, lu, 'vec' perm, 'real' d1,
                          'int' id, 'bool' ready),
         'svd' = 'struct'('mat' u, v, 'vec' sngval,
                          'bool' ready, trims called);

  'op' 'defprob' = ('mat' m)'prob':
    ('heap' 'prb' prob:=
     (small real, small real * 10, small real * 10,
      'size' m * 10); prob
    );

  'op' ('matprob')'lud' 'dec' = 'pr' xref dcmp 'pr' 'skip';
  'op' ('matprob')'svd' 'svdec' = 'pr' xref svdmp 'pr' 'skip';

  'op' 'dec' = ('mat') 'lud' :  'dec' 'matprob' (m, 'defprob' m);

  'op' 'svdec' = ('mat') 'svd' : 'svdec' 'matprob' (m, 'defprob' m);

  'op' 'check' = ('lud' lud)'bool': ready 'of' lud;
  'op' 'check' = ('svd' svd)'bool': ready 'of' svd;

  'op' ('lud', 'vec')'vec' 'sol' = 'pr' xref slv 'pr' 'skip';
  'op' ('svd', 'vec')'vec' 'sol' = 'pr' xref ssvdv 'pr' 'skip';

  'op' ('real','svd') 'svd' 'trims' = 'pr' xref trims 'pr' 'skip';


  'op' 'sqr' = ('vec' x)'real' : x * x;

   'op' 'round' = ('vec' v) 'index' :
   ('index' p:= genintarray('lwb' v,'upb' v);
     'for' i 'from' 'lwb' v 'to' 'upb' v 'do' p[i]:= 'round' v[i] 'od';
     p
   );
```

```
    'prio' 'sol' = 2, 'trims' = 3;
    'real' min real = 2.0 ** -975;
    'pr' prog 'pr' 'skip'
'end' # naprel, numerical algebra prelude#
```

```
nlsprl : 'begin'

   # prelude for solving systems of nonlinear equations
     j.c.p. bus, update 811028,
     to be compiled by: a68,i=lfn,p=nlslib/naprel,n.
   #

'mode' 'func' = 'proc' ('vec','vec','ref' 'bool') 'void' ;
'mode' 'jacob' = 'proc' ('vec','mat','ref' 'bool') 'void' ;

'mode' 'info' = 'struct' ('real' epsrf, epsaf, epsrj, epsaj, dlf, dlrx,
                                 dlax, nrmf, nrmdx,
                          'int' ctit, ctfu, ctjc, ctlu, ctsv, er,
                          'bool' init, 'vec' f, 'mat' b);

'mode' 'reduinfo' = 'struct' ('mat' la, 'vec' lb);

'mode' 'scalinfo' = 'struct' ('bool' sing, fscal, xscal, 'vec' ffacs,
                    xfacs, 'proc'('vec','info','ref''scalinfo') 'void'
                    scale, bckscale);

'mode' 'nlsprb' = 'ref' 'nlsprob';
'mode' 'nlsprob' = 'struct' ('vec' x, 'func' fun, 'ref' 'info' info,
                   'ref' 'reduinfo' rinfo, 'ref' 'scalinfo' sinfo);

'mode' 'metnls' = 'struct' ('nlsprb' nlp, 'ref' 'metinfo' mi);

'mode' 'metinfo' = 'struct'('proc'('metnls') 'vec' nlssolver,
                   'bool' print, safe, updok, 'int' maxits);

'mode' 'nlsprbjc' = 'ref' 'nlsprobjc';
'mode' 'nlsprobjc' = 'struct' ('vec' x, 'func' fun, 'jacob' jac,
                     'ref' 'info' info,
                     'ref' 'reduinfo' rinfo, 'ref' 'scalinfo' sinfo);

'mode' 'metnlsjc' = 'struct' ('nlsprbjc' nlp, 'ref' 'metinfojc' mi);

'mode' 'metinfojc' = 'struct'('proc'('metnlsjc') 'vec' nlssolver,
                                 'bool' print, safe, updok,'int' maxits);

'info' definfos = (small real, small real, small real, small real,
                   sqrt(small real), sqrt(small real), sqrt(small real),
                   maxreal, 0, 0, 0, 0, 0, 0, 0, 'false', 'nil', 'nil');

'proc' dummy = ('vec' x, 'info' info, 'ref''scalinfo' sinfo) 'void' :
               'skip';

'proc' ('vec', 'info', 'ref''scalinfo') 'void' scale =
'pr' xref scale 'pr' 'skip';
'proc' ('vec', 'info', 'ref''scalinfo') 'void' bckscale =
```

```
'pr' xref bckscal 'pr' 'skip';

'reduinfo' defreduinfo = ('nil', 'nil');

'scalinfo' defscalinfo = ('false', 'false', 'false', 'nil', 'nil',
                          dummy, dummy);

'op' 'setnls' = ('func' fun) 'nlsprb' :
     'heap' 'nlsprob' :=
      ('nil',fun,'heap' 'info':= definfos,
       'heap' 'reduinfo':= defreduinfo,
       'heap' 'scalinfo':= defscalinfo);

'op' 'jacobian' = ('func' fun, 'jacob' jac) 'nlsprbjc' :
     'heap' 'nlsprobjc' :=
      ('nil', fun, jac, 'heap' 'info':= definfos,
       'heap' 'reduinfo':= defreduinfo,
       'heap' 'scalinfo':= defscalinfo);

'op' 'method' = ('func' fun, 'ref' 'metinfo' mi) 'metnls' :
                ('setnls' fun, mi);

'op' 'method' = ('nlsprb' nlp, 'ref' 'metinfo' mi) 'metnls' : (nlp,mi);

'op' 'method' = ('nlsprbjc' nlp, 'ref' 'metinfojc' mi) 'metnlsjc' :
                (nlp,mi);

'proc' diff newton = 'ref' 'metinfo' :
       'heap' 'metinfo':= (dbu, 'true', 'true', 'true', 40);

'proc' ('metnls') 'vec' dbu = 'pr' xref dbu 'pr' 'skip';

'proc' general diff newton = 'ref' 'metinfo' :
       'heap' 'metinfo':= (gds, 'true', 'false', 'false', 40);

'proc' ('metnls') 'vec' gds = 'pr' xref gds 'pr' 'skip';

'proc' poly diff newton = 'ref' 'metinfo' :
       'heap' 'metinfo':= (snoleq, 'true', 'true', 'true', 80);

'proc' ('metnls') 'vec' snoleq  = 'pr' xref snoleq 'pr' 'skip';

'proc' newton = 'ref' 'metinfojc' :
       'heap' 'metinfojc':= (abu, 'true', 'true', 'true', 40);
'proc' ('metnlsjc') 'vec' abu = 'pr' xref abu 'pr' 'skip';

'proc' general newton = 'ref' 'metinfojc' :
       'heap' 'metinfojc':= (gas, 'true', 'true', 'true', 40);

'proc' ('metnlsjc') 'vec' gas = 'pr' xref gas 'pr' 'skip';
```

```
'proc' poly newton = 'ref' 'metinfojc' :
       'heap' 'metinfojc':= (snoleqj, 'true', 'true', 'true', 40);

'proc' ('metnlsjc') 'vec' snoleqj = 'pr' xref snoleqj 'pr' 'skip';

'op' 'nlssolve' = ('vec' x, 'func' fun) 'vec' :
     x 'nlssolve' 'setnls' fun;

'op' 'nlssolve' = ('vec' x, 'nlsprb' nlp) 'vec' :
      x 'nlssolve' nlp 'method' poly diff newton;

'op' ('vec', 'metnls') 'vec' 'nlssolve' = 'pr' xref nsvmns 'pr' 'skip';

'op' 'nlssolve' = ('vec' x, 'nlsprbjc' nlp) 'vec' :
      x 'nlssolve' nlp 'method' poly newton;

'op' ('vec', 'metnlsjc') 'vec' 'nlssolve' = 'pr' xref nsvmnsj 'pr'
                                            'skip';

'op' 'linmat' = ('func' fun, 'mat' la) 'nlsprb' :
     'setnls' fun 'linmat' la;

'op' 'linmat' = ('nlsprb' nlp, 'mat' la) 'nlsprb' :
     (la 'of' rinfo 'of' nlp:= la; nlp);

'op' 'linmat' = ('nlsprbjc' nlp, 'mat' la) 'nlsprbjc' :
     (la 'of' rinfo 'of' nlp:= la; nlp);

'op' 'linvec' = ('func' fun, 'vec' lb) 'nlsprb' :
     'setnls' fun 'linvec' lb;

'op' 'linvec' = ('nlsprbjc' nlp, 'vec' lb) 'nlsprbjc' :
     (lb 'of' rinfo 'of' nlp:= lb; nlp);

'op' 'linvec' = ('nlsprb' nlp, 'vec' lb) 'nlsprb' :
     (lb 'of' rinfo 'of' nlp:= lb; nlp);

'op' 'scale' = ('func' fun, 'bool' scl) 'nlsprb' :
     'setnls' fun 'scale' scl;

'op' 'scale' = ('nlsprb' nlp, 'bool' scl) 'nlsprb' :
     ( scale 'of' sinfo 'of' nlp:= ( scl ! scale ! dummy );
       bckscale 'of' sinfo 'of' nlp:= ( scl ! bckscale ! dummy );
       nlp);

'op' 'scale' = ('nlsprbjc' nlp, 'bool' scl) 'nlsprbjc' :
     ( scale 'of' sinfo 'of' nlp:= ( scl ! scale ! dummy );
       bckscale 'of' sinfo 'of' nlp:= ( scl ! bckscale ! dummy );
       nlp);
```

```
'op' 'epsrf' = ('func' fun, 'real' epsrf) 'nlsprb' :
     'setnls' fun 'epsrf' epsrf;

'op' 'epsrf' = ('nlsprb' nlp, 'real' epsrf) 'nlsprb' :
     (epsrf 'of' info 'of' nlp:= epsrf; nlp);

'op' 'epsrf' = ('nlsprbjc' nlp, 'real' epsrf) 'nlsprbjc' :
     (epsrf 'of' info 'of' nlp:= epsrf; nlp);

'op' 'epsaf' = ('func' fun, 'real' epsaf) 'nlsprb' :
     'setnls' fun 'epsaf' epsaf;

'op' 'epsaf' = ('nlsprb' nlp, 'real' epsaf) 'nlsprb' :
     (epsaf 'of' info 'of' nlp:= epsaf; nlp);

'op' 'epsaf' = ('nlsprbjc' nlp, 'real' epsaf) 'nlsprbjc' :
     (epsaf 'of' info 'of' nlp:= epsaf; nlp);

'op' 'epsrj' = ('nlsprbjc' nlp, 'real' epsrj) 'nlsprbjc' :
     (epsrj 'of' info 'of' nlp:= epsrj; nlp);

'op' 'epsaj' = ('nlsprbjc' nlp, 'real' epsaj) 'nlsprbjc' :
     (epsaj 'of' info 'of' nlp:= epsaj; nlp);

'op' 'dlf' = ('func' fun, 'real' dlf) 'nlsprb' :
     'setnls' fun 'dlf' dlf;

'op' 'dlf' = ('nlsprb' nlp, 'real' dlf) 'nlsprb' :
     (dlf 'of' info 'of' nlp:= dlf; nlp);

'op' 'dlf' = ('nlsprbjc' nlp, 'real' dlf) 'nlsprbjc' :
     (dlf 'of' info 'of' nlp:= dlf; nlp);

'op' 'dlrx' = ('func' fun, 'real' dlrx) 'nlsprb' :
     'setnls' fun 'dlrx' dlrx;

'op' 'dlrx' = ('nlsprb' nlp, 'real' dlrx) 'nlsprb' :
     (dlrx 'of' info 'of' nlp:= dlrx; nlp);

'op' 'dlrx' = ('nlsprbjc' nlp, 'real' dlrx) 'nlsprbjc' :
     (dlrx 'of' info 'of' nlp:= dlrx; nlp);

'op' 'dlax' = ('func' fun, 'real' dlax) 'nlsprb' :
     'setnls' fun 'dlax' dlax;

'op' 'dlax' = ('nlsprb' nlp, 'real' dlax) 'nlsprb' :
     (dlax 'of' info 'of' nlp:= dlax; nlp);

'op' 'dlax' = ('nlsprbjc' nlp, 'real' dlax) 'nlsprbjc' :
```

```
    (dlax 'of' info 'of' nlp:= dlax; nlp);

'op' 'print' = ('func' fun, 'bool' print) 'metnls' :
    'setnls' fun 'print' print;

'op' 'print' = ('nlsprb' nlp, 'bool' print) 'metnls' :
    (nlp, 'heap' 'metinfo':= (snoleq, print, 'true', 'true', 40));

'op' 'print' = ('metnls' mn, 'bool' print) 'metnls' :
    (print 'of' mi 'of' mn:= print; mn);

'op' 'print' = ('metnlsjc' mn, 'bool' print) 'metnlsjc' :
    (print 'of' mi 'of' mn:= print; mn);

'op' 'print' = ('nlsprbjc' nlp, 'bool' print) 'metnlsjc' :
     (nlp, 'heap' 'metinfojc':= (snoleqj, print, 'true', 'true', 40));

'op' 'safe' = ('func' fun, 'bool' safe) 'metnls' :
    'setnls' fun 'safe' safe;

'op' 'safe' = ('nlsprb' nlp, 'bool' safe) 'metnls' :
    (nlp, 'heap' 'metinfo':= (snoleq, 'true', safe, 'true', 40));

'op' 'safe' = ('metnls' mn, 'bool' safe) 'metnls' :
    (safe 'of' mi 'of' mn:= safe; mn);

'op' 'safe' = ('metnlsjc' mn, 'bool' safe) 'metnlsjc' :
    (safe 'of' mi 'of' mn:= safe; mn);

'op' 'safe' = ('nlsprbjc' nlp, 'bool' safe) 'metnlsjc' :
     (nlp, 'heap' 'metinfojc':= (snoleqj, 'true', safe, 'true', 40));

'op' 'updok' = ('func' fun, 'bool' updok) 'metnls' :
    'setnls' fun 'updok' updok;

'op' 'updok' = ('nlsprb' nlp, 'bool' updok) 'metnls' :
    (nlp, 'heap' 'metinfo':= (snoleq, 'true', 'true', updok, 40));

'op' 'updok' = ('metnls' mn, 'bool' updok) 'metnls' :
    (updok 'of' mi 'of' mn:= updok; mn);

'op' 'updok' = ('metnlsjc' mn, 'bool' updok) 'metnlsjc' :
    (updok 'of' mi 'of' mn:= updok; mn);

'op' 'updok' = ('nlsprbjc' nlp, 'bool' updok) 'metnlsjc' :
     (nlp, 'heap' 'metinfojc':= (snoleqj, 'true', 'true', updok, 40));

'op' 'maxits' = ('func' fun, 'int' maxits) 'metnls' :
    'setnls' fun 'maxits' maxits;
```

```
'op' 'maxits' = ('nlsprb' nlp, 'int' maxits) 'metnls' :
     (nlp, 'heap' 'metinfo':= (snoleq, 'true', 'true', 'true',maxits));

'op' 'maxits' = ('metnls' mn, 'int' maxits) 'metnls' :
     (maxits 'of' mi 'of' mn:= maxits; mn);

'op' 'maxits' = ('metnlsjc' mn, 'int' maxits) 'metnlsjc' :
     (maxits 'of' mi 'of' mn:= maxits; mn);

'op' 'maxits' = ('nlsprbjc' nlp, 'int' maxits) 'metnlsjc' :
      (nlp,
       'heap' 'metinfojc':= (snoleqj, 'true', 'true', 'true', maxits));

'op' ('vec') 'real' 'nrm' = 'pr' xref nrm 'pr' 'skip';

'op' 'det' = ('lud' lud) 'real' : (d1 'of' lud) * 2.0 ** (id 'of' lud);

'mode' 'auxil' = 'struct'('vec' x, 'ref''vec' dx,'ref' 'info' info,
                  'int' maxits,
                  'ref''real' labda, om, kappa, epsf, e, hs, nrmx,nrmbi,
                  'ref''bool' dif, 'proc'('int') 'void' errex);

'proc' ('vec','vec','mat','real','func','ref''bool') 'void' jacobnnf =
        'pr' xref jacob 'pr' 'skip';

'proc' ('auxil', 'proc'('vec','vec') 'real')
        'void' resbis = 'pr' xref resbis 'pr' 'skip';

'proc' ('auxil') 'bool' stopful = 'pr' xref stopful 'pr' 'skip';
'proc' ('auxil') 'bool' stopspl = 'pr' xref stopspl 'pr' 'skip';

'proc' ('int','auxil','scalinfo','bool') 'void' monitor =
        'pr' xref monitor 'pr' 'skip';

'proc' ('ref''file','vec','int','int','string') 'void' outvec =
        'pr' xref outvec 'pr' 'skip';

'proc' ('ref''file','mat','int','int','string') 'void' outmat =
        'pr' xref outmat 'pr' 'skip';

'proc' ('int') 'void' error = 'pr' xref error 'pr' 'skip' ;

'prio' 'nlssolve' = 2, 'method' = 3, 'setnls' = 3, 'epsrf' = 3,
       'epsaf' = 3, 'dlf' = 3, 'dlrx' = 3, 'dlax' = 3, 'scale' = 3,
       'linmat' = 3, 'linvec' = 3, 'safe' = 3, 'updok' = 3,
       'maxits = 3, 'print' = 3, 'nrm' = 9, 'det' = 9, 'jacobian' = 3;
#
'ref' 'file' nlsbook:= stand out;

'pr' prog 'pr' 'skip'

'end'
```

```
'begin'

'op' 'nrm' = ('vec' x) 'real' :
'pr' xdef nrm 'pr'
'begin' 'real' s1:= 'maxabs' x;
      'if' s1 <= minreal 'then' 0
      'else' s1:= 2.0 ** 'entier' ( ln(s1)/ln(2));
         'vec' v = (1/s1) * x; sqrt(v * v) * s1
      'fi'
'end' 'pr' fedx 'pr';

'skip'

'end'
```

```
'begin'

'proc' outvec = ('ref''file' ch, 'vec' x, 'int' m,e, 'string' s) 'void':
'pr' xdef outvec 'pr'
'begin'
      'int' lp = 60, cl = 80, lu = m + e + 15, u = 'upb' x, l = 'lwb' x,
            w = m + e + 5;
      'int' lv = u - l + 1, 'int' nul:= cl 'over' lu;
      'int' nl:= lv 'over' nul, k:= lv - nl * nul;
      'int' llc = 'if' k = 0 'then' nl + 1
                  'else' nl +:= 1; 'int' llc:= nl + k - nul + 1;
                     'while' llc <= 0
                     'do' llc +:= nl; nul -:= 1 'od';
                     llc
                  'fi';
      'int' pp = line number(ch);
      'if' lp - pp < nl + 3 'and' ( pp /= 0 'or' char number(ch) /= 0)
      'then' newpage(ch) 'else' newline(ch) 'fi';
      put(ch,(s,newline,newline)); nul:= (nul - 1) * nl;
      'for' i 'to' nl
      'do' 'int' mn = l + i - 1; 'if' i = llc 'then' nul -:= nl 'fi';
         'for' j 'from' mn 'by' nl 'to' mn + nul
         'do' put(ch,(whole(j,-4),"   ",float(x[j],w,m,e+1),"   "))'od';
         newline(ch)
      'od'
'end' 'pr' fedx 'pr';

'skip'

'end'
```

```
'begin'

'proc' outmat = ('ref''file' ch, 'mat' a, 'int' m,e, 'string' s) 'void':
'pr' xdef outmat 'pr'
'begin'
      'int' lp =60, cl = 80, lu = m + e + 6, w = m + e + 5,
            lr = 1 'lwb' a, ur = 1 'upb' a, lc = 2 'lwb' a,
            uc = 2 'upb' a, pp = line number(ch);
      'if' pp /= 0
      'then' 'if' pp + ur -lr + 9 > lp 'then' newpage(ch)
             'else' newline(ch); newline(ch) 'fi'
      'fi';
      put(ch,(s,newline,newline));
      'int' l:= (cl - 6) 'over' lu, 'int' v = - ((lu + 13) 'over' 2);
      'for' p 'from' lc 'by' l 'to' uc
      'do' 'int' up = 'if' p + l - 1 > uc 'then' l:= uc - p + 1; uc
                      'else' p + l - 1 'fi';
         put(ch,whole(p,v));
         'for' j 'from' p + 1 'to' up 'do' put(ch,whole(j,-lu)) 'od';
         newline(ch); newline(ch);
         'for' i 'from' lr 'to' ur
         'do' put(ch,whole(i,-4));
            'for' j 'from' p 'to' up
            'do' put(ch,(" ",float(a[i,j],w,m,e + 1))) 'od';
            newline(ch)
         'od';
         newline(ch); newline(ch)
      'od'
'end' 'pr' fedx 'pr';

'skip'

'end'
```

```
'begin'

'proc' error = ('int' er) 'void' :
'pr' xdef error 'pr'
'begin'
      [] 'string' errtext =
      ( "no progress, maybe due to too high required precicion",
      "no progress relative to error function",
      "stationary point of norm of the function, no solution",
      "too many function evaluations or iterations required",
      "numerical singularity in triangular decomposition",
      "failure of singular value decomposition",
      "rank of jacob. approx. equal to zero",
      "error in jacob. approx. yields possible singular jacob.",
      "nearby singularity of jacobian expected",
      "difference approx. impossible, point of boundary domain",
      "divergence out of domain of function",
      "starting point not in domain of function",
      "epsrf set to default (small real)",
      "epsaf set to default (0)",
      "epsrj set to default (small real)",
      "epsaj set to default (0)",
      "dlf set to default (epsaf)",
      "dlrx set to default (small real)",
      "dlax set to default (small real)",
      "matrix of linear part not full rank",
      "vector of linear part not full rank");

      'int' n = 'upb' errtext[er] 'over' 2 + 15;
      put(nlsbook,(newline,newline));
      'for' i 'to' n 'do' put(nlsbook,"* ") 'od';
      'if' er < 13 'or' er > 19
      'then' put(nlsbook,(newline,"*  failure of iteration : "))
      'else' put(nlsbook,(newline,"*  error in problem     : ")) 'fi';
      put(nlsbook,errtext[er]);
      'for' i 'from' 'upb' errtext[er] 'mod' 2 'to' 1
      'do' put(nlsbook," ") 'od';
      put(nlsbook,("*",newline));
      'for' i 'to' n 'do' put(nlsbook,"* ") 'od';
      put(nlsbook,newline)
'end'
'pr' fedx 'pr';

'skip'

'end'
```

```
'begin'

'proc' monitor = ('int' ph, 'auxil' aux, 'scalinfo' sinfo, 'bool' redu)
                 'void' :
'pr' xdef monitor 'pr'
'begin' 'info' info = info 'of' aux;
        'int' er = er   'of' info,
        'bool' fscal = fscal 'of' sinfo,
               xscal = xscal 'of' sinfo;

      'if' ph = 0
      'then' put(nlsbook,(newline,newline,newline,newline));
         'if' fscal 'or' xscal
         'then' put(nlsbook,"start of iteration, scaling performed");
            'if' fscal 'then' outvec(nlsbook,ffacs 'of' sinfo,5,3,
                                     "row scaling facs") 'fi';
            'if' xscal 'then' outvec(nlsbook,xfacs 'of' sinfo,5,3,
                                     "col scaling facs") 'fi'
         'else' put(nlsbook,("start of iteration, no scaling performed",
                             newline))
         'fi';
         put(nlsbook,(newline," its  fus  jcs  norm(f)    labda",
             "    omega    beta    kappa  nrmjacinv errjac dfstep",
                  newline))
      'fi';
      'if' ph <= 1
      'then' 'real' lb = labda 'of' aux;
             put(nlsbook,(whole(ctit 'of' info,-4),
                 whole(ctfu 'of' info,-5),
                 whole(ctjc 'of' info,-5),"  ",
                 float(nrmf 'of' info,-9,3,4),"  ",
                 float(lb,-7,1,4),"  ",
                 float(om 'of' aux,-7,1,4),"  ",
                 float(nrmdx 'of' info/lb,-6,1,3),"  ",
                 float(kappa 'of' aux,-7,1,4),"  ",
                 float(nrmbi 'of' aux,-7,1,4),"  ",
                 float(e 'of' aux,-6,1,3),"  ",
                 float(hs 'of' aux,-5,1,2),newline))
      'else' 'if' ph = 2
         'then' put(nlsbook,("end of iteration",newline));
            'if' er = 0
            'then' put(nlsbook,("iteration successful",newline)) 'fi'
         'fi';
         'if' ph = 3 'or' (ph = 2 'and' 'not' redu)
         'then' outvec(nlsbook,x 'of' aux,
                       'entier'(-ln(dlrx 'of' info)/ln(10)) + 1,
                       3,"approx. solution");
            'if' er < 10
            'then' 'int' nc = 'entier'(-ln(epsrf 'of' info)
                                 /ln(10)) + 1;
```

```
                outvec(nlsbook,f 'of' info,
                      'if' nc < 6 'then' nc 'else' 6 'fi', 3, "function")
            'fi'
          'fi'
       'fi'
'end' 'pr' fedx 'pr';

'skip'

'end'
```

```
'begin'

'proc' resbis = ('auxil' aux, 'proc'('vec', 'vec') 'real' lfun) 'void' :
'pr' xdef resbis 'pr'
'begin' 'vec' x = x 'of' aux,
              dx  =  dx 'of' aux,
        'ref''real' nrmdx = nrmdx 'of' info 'of' aux,
                    labda = labda 'of' aux,
                    level = nrmf 'of' info 'of' aux,
        'ref''int'  er    = er 'of' info 'of' aux;
        'vec' x1:= x, 'real' min = small real * nrmx 'of' aux * 2/nrmdx,
        'real' level1:= level * 2;
        labda:= 2; er:= 0;

      'while' 'bool' bis = 'if' 'abs'(level - level1) < epsf 'of' aux
                                'and' level /= max real
                           'then' 'if' er = 2 'then' 'false'
                                  'else' er:= 2; 'true' 'fi'
                           'else' er:= 0; level1 >= level 'fi';
         'if' labda <= min 'then' er:= 1; 'false' 'else' bis 'fi'
      'do' labda *:= 0.5; x1:= x - labda * dx;
         level1:= lfun(x1,f 'of' info 'of' aux)
      'od';
      x:= x1; dx *< -labda; nrmdx *:= labda; level:= level1; 'skip'
'end'
'pr' fedx 'pr';

'skip'

'end'
```

```
'begin'

'proc' stopspl = ('auxil' aux) 'bool' :
'pr' xdef stopspl 'pr'
'begin' 'real'   nrmf = nrmf 'of' info 'of' aux,
        'ref''int' er = er 'of' info 'of' aux,
        'proc'('int') 'void' errex = errex 'of' aux;
      'if' nrmf<= epsaf 'of' info 'of' aux 'then' errex(0) 'fi';
      'bool' bl = nrmdx 'of' info 'of' aux < nrmx 'of' aux
                  * dlrx 'of' info 'of' aux + dlax 'of' info 'of' aux
                'and' nrmf < dlf 'of' info 'of' aux;
      'if' ctit 'of' info 'of' aux >= maxits 'of' aux 'and' 'not' bl
      'then' errex(4) 'fi';
      'if' bl 'then' er:= 0 'elif' er /= 0 'then' errex(er) 'fi'; bl
'end' 'pr' fedx 'pr';

'skip'

'end'


'begin'

'proc' stopful = ('auxil' aux) 'bool' :
'pr' xdef stopful 'pr'
'begin' 'real' nrmdx = nrmdx 'of' info 'of' aux,
               nrmf  = nrmf 'of' info 'of' aux,
               kappa = kappa 'of' aux,
               e     = e 'of' aux,
               labda = labda 'of' aux,
       'ref''int' er = er 'of' info 'of' aux,
       'int'      it = ctit 'of' info 'of' aux,
       'proc'('int') 'void' errex  = errex 'of' aux;
     'bool' bl = 'if' nrmf < epsaf 'of' info 'of' aux 'then' 'true'
     'elif' e >= 1 - small real 'then' errex(8); 'true'
     'else' 'real' ksi1 = (1 + e)/(1 - e),
                   ksi2 = (1 - (e + 2) * e)/(1 - e),
                   alfa2 = om 'of' aux * nrmdx/labda * 2;
       'real' a1 = alfa2 * ksi1, k2 = ksi2 * ksi2;
       'if' e < 0.4142 'and' labda = 1 'and'
            'if' a1 < k2
            'then' nrmdx <= nrmx 'of' aux * dlrx 'of' info 'of' aux
              + dlax 'of' info 'of' aux/(2/(ksi2 + sqrt(k2 - a1)) - 1)
            'else' 'false' 'fi' 'and' nrmf < dlf 'of' info 'of' aux
       'then' 'true'
       'elif' it = 1 'then' 'false'
       'else' 'if' e * kappa >= 0.5 'and' dif 'of' aux
              'then' errex(8) 'fi';
          'real' tau2i:= ((1 + kappa * 2) * alfa2) ** 2;
          'if' tau2i <= 1 'then' tau2i:= 1 'fi';
          'if' nrmf <= epsf 'of' aux * tau2i 'then' errex(9) 'fi';
          'false'
```

```
        'fi'
    'fi';
    'if' it >= maxits 'of' aux 'and' 'not' bl 'then' errex(4)
    'elif' bl 'then' er:= 0 'elif' er /= 0 'then' errex(er) 'fi';
    bl
'end'
'pr' fedx 'pr';

'skip'

'end'
```

```
'begin'

'proc' jacobnnf = ('vec' x, f, 'mat' b, 'real' hs, 'func' fun,
                   'ref''bool' ok) 'void' :
'pr' xdef jacob 'pr'
'begin' 'int' n = 'upb' x; 'vec' f1:= genvec(n); ok:= 'true';
    'for' i 'to' n 'while' ok
    'do' 'real' aid = x[i]; 'real' step:= ('abs' aid + 1) * hs;
      x[i]+:= step; fun(x,f1,ok);
      'if'
        'if' ok 'then' 'true'
        'else' step:= ('abs' aid + 1) * small real * 100;
          x[i]:= aid + step; fun(x,f1,ok); ok
        'fi'
      'then' b[,i]:= (f1 - f)/step
      'fi';
      x[i]:= aid
    'od'
'end' 'pr' fedx 'pr';

'skip'

'end'
```

```
'begin'

'proc' scale = ('vec' x, 'info' info, 'ref''scalinfo' sinfo) 'void' :
'pr' xdef scale 'pr'
'begin' 'ref''vec' ffacs = ffacs 'of' sinfo,
                   xfacs = xfacs 'of' sinfo,
        'vec' f     = f     'of' info ,
        'mat' b     = b     'of' info ,
        'ref''bool' sing  = sing  'of' sinfo,
                    fscal = fscal 'of' sinfo,
                    xscal = xscal 'of' sinfo;
      'int' n = 'upb' x;
      'real' max = n * 4; 'real' min = 1/max, ln2 = ln(2);

      ffacs:= 1 'into' genvec(n); xfacs:= 1 'into' genvec(n);
      sing:= 'false'; fscal:= 'false'; xscal:= 'false';
      'for' i 'to' n
      'do' 'int' k; 'real' rnorm = k 'maxabs' b[i,];
         'if' rnorm < minreal 'then' sing:= 'true'; stop 'fi';
         'if' rnorm > max 'or' rnorm < min
         'then' fscal:= 'true';
            'real' aid = 2.0 ** 'entier'(-ln(rnorm)/ln2);
            ffacs[i]:= aid; b[i,] *<aid; f[i] *:= aid
         'fi'
      'od';
      'for' i 'to' n
      'do' 'int' k; 'real' cnorm = k 'maxabs' b[,i];
         'if' cnorm < minreal 'then' sing:= 'true'; stop 'fi';
         'if' cnorm > max 'or' cnorm < min
         'then' xscal:= 'true';
            'real' aid = 2.0 ** 'entier'(-ln(cnorm)/ln2);
            xfacs[i]:= aid; b[,i] *<aid; x[i] /:= aid
         'fi'
      'od';
stop : 'skip'
'end' 'pr' fedx 'pr';

'proc' bckscale = ('vec' x, 'info' info, 'ref''scalinfo' sinfo) 'void' :
'pr' xdef bckscal 'pr'
'begin' 'vec' ffacs = ffacs 'of' sinfo,
              xfacs = xfacs 'of' sinfo,
              f     = f     'of' info ,
        'mat' b     = b     'of' info ,
        'ref''bool' fscal = fscal 'of' sinfo,
                    xscal = xscal 'of' sinfo;
      'int' n = 'upb' x;

      'if' fscal
      'then' 'for' i 'to' n
         'do' 'real' aid:= ffacs[i];
```

```
            'if' aid /= 1 'then' b[i,] /< aid; f[i] /:= aid 'fi'
         'od';
         fscal:= 'false'
      'fi';
      'if' xscal
      'then' 'for' i 'to' n
         'do' 'real' aid:= xfacs[i];
            'if' aid /= 1 'then' b[,i] /< aid; x[i] *:= aid 'fi'
         'od';
         xscal:= 'false'
      'fi'
'end' 'pr' fedx 'pr';

'skip'

'end'
```

```
'begin'

'proc' gds = ('metnls' mn) 'vec' :
'pr' xdef gds 'pr'
'begin'

      'proc' genranvec = ('int' n) 'vec' :
      'begin' 'proc' ran = ('int' i) 'real' : next random(setr) - 0.5;
         'vec' v = ran 'into' genvec(n);
         v /< 'nrm' v
      'end';

      'proc' calhsg = 'real' :
      'begin' nrmu1:= nrmu2:= 0;
         'for' i 'to' n
         'do' 'real' aid = (1 + 'abs' x[i]) ** 2; nrmu1+:= aid;
            nrmu2+:= 1/aid
         'od';
         nrmu1:= sqrt(nrmu1); nrmu2:= sqrt(nrmu2);
         'real' c1 = nrmu1 * gamma * 0.5, c2 = nrmu2 * epsf * 2;
         'real' hs = 'if' c1 <= c2 'then' 1 'else' sqrt(c2/c1) 'fi';
         'if' hs <= small real * 100 'then' small real * 100 'else' hs
         'fi'
      'end';

      'proc' cdatasv = 'void' :
      'begin' 'svd' svd;
         gamma:= 'if' it - mnit = 1 'then' w:= b * v; 1
         'else' hs:= calhsg; cntf +:= n; jacobnnf(x,f,b,hs,fun,ok);
            'if' 'not' ok 'then' exit(10) 'fi';
            'real' nrmw = 'nrm' (w - (w:= b * v));
            'if' nrmdx < small real * nrmw 'then' 1/small real
            'else' nrmw/nrmdx
            'fi'
         'fi';
         'if' 'not' 'check' (ctsv +:= 1; svd:= 'svdec' b)
         'then' exit(6) 'fi';
         'real' maxval = 'max' sngval 'of' svd;
         'real' ck = nrmu1 * gamma * hs * 0.5 + nrmu2 * epsf * 2/hs
                   + maxval * n * small real;
         svd:= ck 'trims' svd;
         rk:= 'upb' sngval 'of' svd;
         'if' rk = 0 'then' exit(7) 'fi';
         nrmbi:= 1/(sngval 'of' svd)[rk]; e:= ck * nrmbi;
         'if' e > 1 - small real 'then' e:= 1 - small real 'fi';
         dx:= svd 'sol' f; nrmdx:= 'nrm' dx;
         kappa:= maxval * nrmdx/slevel;
         'if' 'nrm' (f * u 'of' svd)< epsf 'then' er:= 3 'fi';
         'skip'
      'end';
```

```
'proc' levelfu = ('vec' x, f) 'real' :
'begin' fun(x,f,ok);
   'if' 'not' ok 'then' exit(11) 'fi'; 'nrm' f
'end';

'proc' exit = ('int' err) 'void' :
'begin' er:= err; end 'end';

'nlsprb' nlp = nlp 'of' mm;
'ref''info' info = info 'of' nlp;
'vec' x = x 'of' nlp, 'func' fun = fun 'of' nlp,
'real' dlf = dlf 'of' info,
       dlrx  = dlrx 'of' info,
       dlax = dlax 'of' info,
       epsrf = epsrf 'of' info,
       epsaf = epsaf 'of' info,
'ref''real' slevel = nrmf 'of' info,
            nrmdx  = nrmdx 'of' info,
'ref''int' it   = ctit 'of' info,
           cntf = ctfu 'of' info,
           ctsv = ctsv 'of' info,
           er   = er   'of' info,
'int'  maxits = maxits 'of' mi 'of' mm,
      mnit = ctit 'of' info,
'bool' print = print 'of' mi 'of' mm;
'int' n = 'upb' x,
'real' e:= 0, gamma:= 1, nrmbi:= 1, kappa:= 1,
       epsf, hs, nrmu1, nrmu2,
       nrmx:= 'nrm' x,
'bool' slow:= 'false', ok:= 'true',
'vec' dx;
'auxil' aux = (x, dx, info, maxits, 'heap' 'real' := 1,
              gamma, kappa, epsf, e, hs, nrmx, nrmbi, slow, exit);

'if' init 'of' info
'then' slevel:= 'nrm' f 'of' info;
   'if' slevel <= epsaf 'then' exit(0) 'fi';
   epsf:= (epsrf + small real) * slevel + epsaf; hs:= calhsg
'else' 'vec' f:= genvec(n); 'mat' b:= gensquare(n);
   f 'of' info:= f; b 'of' info:= b;
   cntf +:= n + 1; slevel:= levelfu(x,f);
   'if' slevel <= epsaf 'then' exit(0) 'fi';
   epsf:= (epsrf + small real) * slevel + epsaf;
   hs:= calhsg; jacobnnf(x,f,b,hs,fun,ok);
   'if' 'not' ok 'then' exit(10) 'fi';
   init 'of' info:= 'true'
'fi';

'mat' b = b 'of' info, 'vec' f = f 'of' info;
```

```
      'int' setr:= maxint 'over' n;
      'int' rk:= n, 'vec' v:= genranvec(n), xj, w;
      'if' print 'then'
        monitor(0, aux, sinfo 'of' nlp, 'false')
      'fi';

      'while'
         'if' it = mnit 'then' 'true' 'else' 'not' stopspl(aux) 'fi'
      'do' it+:= 1; cdatasv; nrmx:= 'nrm' (x -< dx); cntf+:= 1;
         'real' sl0:= slevel; slevel:= levelfu(x,f);
         epsf:= (epsrf + small real) * slevel + epsaf;
         'if' 'abs'(sl0 - slevel) > epsf 'then' slow:= 'false'
         'elif' slow 'then' er:= 2 'else' slow:= 'true' 'fi';
         'if' print 'then' monitor(1, aux, sinfo 'of' nlp, 'false') 'fi'
      'od';
end : 'if' print 'then' monitor(2, aux, sinfo 'of' nlp,
                'mat'(la 'of' rinfo 'of' nlp) 'isnt' 'mat'('nil')) 'fi';
      'if' er /= 0 'then' error(er) 'fi';
      x
'end'
'pr' fedx 'pr';

'skip'

'end'
```

```
'begin'

'proc' dbu = ('metnls' mn) 'vec' :
'pr' xdef dbu 'pr'
'begin'

      'proc' genranvec = ('int' n) 'vec' :
      'begin' 'proc' ran = ('int' i) 'real' : next random(setr) - 0.5;
         'vec' v = ran 'into' genvec(n);
         v /< 'nrm' v
      'end';

      'proc' calhs = 'real' :
      'begin' nrmu1:= nrmu2:= 0;
         'for' i 'to' n
         'do' 'real' aid = (1 + 'abs' x[i]) ** 2; nrmu1+:= aid;
            nrmu2+:= 1/aid
         'od';
         nrmu1:= sqrt(nrmu1); nrmu2:= sqrt(nrmu2);
         'real' c1 = nrmu1 * om * 0.5, c2 = nrmu2 * epsf * nrmbi * 2;
         'real' s = 'if' c1 * c2 < small real 'then' small real
                    'else' c1 * c2 'fi';
         'real' hs:= (-1 + sqrt(1 + 1/s)) * c2;
         'if' hs > 1 'then' 1
         'elif' hs < small real * 100 'then' small real * 100 'else' hs
         'fi'
      'end';

      'proc' conupdjac = 'bool' :
      'if' it - mnit < 3 'or' 'not' update 'or' e >= 0.1 'then' 'false'
      'else' 'vec' df:= f - f0; 'vec' u = lud 'sol' df;
         'real' pu = dx * u, nrmu = 'nrm' u;
         'if' e < 1
         'then' e:= (e/(1 - e) + ( 1 + nrmdx * 1.5/nrmu) * nrmdx * om)
                    * (1 + e)
         'fi';
         'if' kappa * e < 1 'and' 'abs' pu > nrmdx * nrmu * small real
              'and' e < 0.1
         'then' df +< labda * f0;
            'for' j 'to' 'upb' u 'do' b[,j] +< u[j]/pu * df 'od';
            'true'
         'else' 'false' 'fi'
      'fi';

      'proc' cdatalr = 'void' :
      'begin' 'vec' d01;
         'if' it - mnit > 1
         'then' d01:= lud 'sol' f;
            dif:= 'if' conupdjac 'then' 'false'
                  'else' hs:= calhs; cntf +:= n;
```

```
                jacobnnf(x,f,b,hs,fun,ok);
                'if' 'not' ok 'then' exit(10) 'fi';
                'true'
            'fi'
    'fi';
    'if' 'not' 'check' (ctlu +:= 1; lud:= 'dec' b)
    'then' exit(5) 'fi';
    'if' it - mnit = 1
    'then' beta:= 'nrm' (dx:= lud 'sol' f); om:= 1
    'else' 'real' om1:= 'nrm'((lud 'sol' f0) * labda + dx)
                        /nrmdx ** 2;
       beta:= 'nrm'(dx:= lud 'sol' f);
       om:= 'nrm'(d01 - dx)/(nrmdx * beta);
       'if' om1 > om 'then' om:= om1 'fi'
    'fi';
    nrmdx:= beta; f0:= 'copy' f; kappa:= 'maxabs' b * nrmdx/slevel;
    nrmbi:= 'nrm'(lud 'sol' v);
    'if' dif
    'then' 'real' aid = 1 - small real,
       'real' c1:= nrmu1 * om * 0.5 * hs;
       'real' c2 = (nrmu2 * nrmbi * epsf * 2)/hs + c1;
       c1:= 1 - c1;
       e:= 'if' c1 <= c2 * aid 'then' aid 'else' c2/c1 'fi'
    'fi';
    'skip'
'end';

'proc' levelfu = ('vec' x, f) 'real' :
'begin' fun(x,f,ok); cntf +:= 1;
    'if' ok 'then' 'nrm' f 'else' max real 'fi'
'end';

'proc' exit = ('int' err) 'void' :
'begin' er:= err; end 'end';

'nlsprb' nlp = nlp 'of' mm;
'ref''info' info = info 'of' nlp,
'ref''scalinfo' sinfo = sinfo 'of' nlp;
'vec' x = x 'of' nlp,
'ref''vec' xfacs = xfacs 'of' sinfo,
           ffacs = ffacs 'of' sinfo,
'func' fun:= fun 'of' nlp,
'real' dlf = dlf 'of' info,
       dlrx = dlrx 'of' info,
       dlax = dlax 'of' info,
       epsrf = epsrf 'of' info,
       epsaf = epsaf 'of' info,
'ref''real' slevel = nrmf 'of' info,
            nrmdx  = nrmdx 'of' info,
'ref''int' it   = ctit 'of' info,
```

```
            cntf = ctfu 'of' info,
            ctlu = ctlu 'of' info,
            er   = er   'of' info,
'int'  maxits = maxits 'of' mi 'of' mn,
       mnit = ctit 'of' info,
'bool' print = print 'of' mi 'of' mn,
       update = updok 'of' mi 'of' mn,
'ref''bool' xscal = xscal 'of' sinfo,
            fscal = fscal 'of' sinfo;
'int' n = 'upb' x,
'real' e:= 0, nrmbi:= 1, kappa:= 1, beta:= 1, om:= 1, labda:= 1,
       epsf, hs, nrmu1, nrmu2, nrmx,
'bool' dif:= 'true', ok:= 'true',
'vec' dx;
'auxil' aux = (x, dx, info, maxits, labda, om, kappa,
               epsf, e, hs, nrmx, nrmbi, dif, exit);

'if' init 'of' info
'then'
   (scale 'of' sinfo)(x,info,sinfo);
   'if' sing 'of' sinfo 'then' exit(5) 'fi';
   slevel:= 'nrm' f 'of' info;
   'if' slevel <= epsaf 'then' exit(0) 'fi';
   epsf:= (epsrf + small real) * slevel + epsaf; hs:= calhs
'else' 'vec' f:= genvec(n); 'mat' b:= gensquare(n);
   f 'of' info:= f; b 'of' info:= b;
   slevel:= levelfu(x,f);
   'if' slevel <= epsaf 'then' exit(0)
   'elif' slevel = maxreal 'then' exit(12) 'fi';
   epsf:= (epsrf + small real) * slevel + epsaf;
   hs:= calhs; jacobnnf(x,f,b,hs,fun,ok); cntf +:= n;
   'if' 'not' ok 'then' exit(10) 'fi';
   init 'of' info:= 'true';
   (scale 'of' sinfo)(x,info,sinfo);
   'if' sing 'of' sinfo 'then' exit(5) 'fi';
   slevel:= 'nrm' f 'of' info;
   'if' slevel <= epsaf 'then' exit(0) 'fi';
   epsf:= (epsrf + small real) * slevel + epsaf
'fi';

'if' fscal 'or' xscal
'then' fun:= ('vec' x,f,'ref''bool' ok) 'void' :
   'begin' 'if' xscal
      'then' 'vec' x1 = genvec(n);
         'for' i 'to' n 'do' x1[i]:= x[i] * xfacs[i] 'od';
         (fun 'of' nlp)(x1,f,ok)
      'else' (fun 'of' nlp)(x,f,ok)
      'fi';
      'if' fscal 'then' 'for' i 'to' n 'do' f[i] *:= ffacs[i] 'od'
      'fi';
```

```
            'skip'
         'end'
      'fi';

      'mat' b = b 'of' info, 'vec' f = f 'of' info;
      'int' setr:= maxint 'over' n;
      'int' rk:= n, 'vec' v:= genranvec(n), f0, 'lud' lud;
      nrmx:= 'nrm' x;
      'proc'('auxil') 'bool' stop =
         'if' safe 'of' mi 'of' mn 'then' stopful 'else' stopspl 'fi';
      'if' print 'then' monitor(0, aux, sinfo, 'false') 'fi';

      'while' 'if' it = mnit 'then' 'true' 'else' 'not' stop(aux) 'fi'
      'do' it+:= 1; cdatalr; resbis(aux,levelfu);
         nrmx:= 'nrm' x;
         'if' 'not' dif 'and' er /= 0
         'then' er:= 0; e:= 1 - small real 'fi';
         epsf:= (small real + epsrf) * slevel + epsaf;
         'if' print 'then' monitor(1, aux, sinfo, 'false') 'fi'
      'od';
end : (bckscale 'of' sinfo)(x,info,sinfo);
      'if' print 'then' monitor(2,aux,sinfo,
                'mat'(la 'of' rinfo 'of' nlp) 'isnt' 'mat'('nil')) 'fi';
      'if' er /= 0 'then' error(er) 'fi';
      x
'end'
'pr' fedx 'pr';

'skip'

'end'



'begin'

'proc' snoleq = ('metnls' mn) 'vec' :
'pr' xdef snoleq 'pr'
'begin' 'ref''int' er = er 'of' info 'of' nlp 'of' mn;
      'int' maxtot:= maxits 'of' mi 'of' mn;
      dbu(mn 'maxits' maxtot 'over' 2);
      mn 'maxits' maxtot;
      'if' er > 0 'and' er < 10 'then' er:= 0; gds(mn) 'fi';
      x 'of' nlp 'of' mn
'end'
'pr' fedx 'pr';

'skip'

'end'
```

```
'begin'

'op' 'nlssolve' = ('vec' x, 'metnls' mm) 'vec' :
'pr' xdef nsvmns 'pr'
'begin' 'proc' exit = ('int' er) 'void' :
      'begin' error(er); end 'end';

      'nlsprb' nlp = nlp 'of' mm;
      'mat' la = la 'of' rinfo 'of' nlp,
      'vec' lb = lb 'of' rinfo 'of' nlp;

      'ref''real' epsrf = epsrf 'of' info 'of' nlp,
                  epsaf = epsaf 'of' info 'of' nlp,
                  dlf   = dlf   'of' info 'of' nlp,
                  dlrx  = dlrx  'of' info 'of' nlp,
                  dlax  = dlax  'of' info 'of' nlp,
      'int' er = er 'of' info 'of' nlp;

      'if' epsrf < small real 'then' error(13); epsrf:= small real 'fi';
      'if' epsaf < 0          'then' error(14); epsaf:= 0          'fi';
      'if' dlf   < epsaf      'then' error(17); dlf  := epsaf      'fi';
      'if' dlrx  < small real 'then' error(18); dlrx := small real 'fi';
      'if' dlax  < small real 'then' error(19); dlax := small real 'fi';

      x 'of' nlp:= x;
      'if' (la 'is' 'mat'('nil'))
      'then' (nlssolver 'of' mi 'of' mm)(mm)
      'elif' lb 'is' 'vec'('nil')
      'then' 'int' m = 1 'upb' la, p = 2 'upb' la;
         'mat' a0 = 0 'into' gensquare(m);
         a0[1:m,1:p]:= la; 'svd' asv0, asv;
         'if' 'not' 'check'(asv0:= 'svdec' a0) 'then' exit(6) 'fi';
         asv:= ('max'(sngval 'of' asv0)* small real * 100) 'trims' asv0;
         'if' 'upb' (sngval 'of' asv) /= p 'then' exit(20) 'fi';
         v 'of' asv:= (v 'of' asv)[1:p,];
         'mat' u2 = (u 'of' asv0)[,p + 1:m];
         'vec' g = genvec(m);
         'func' fun:= fun 'of' nlp;

         fun 'of' nlp:= ('vec' x,f, 'ref''bool' ok) 'void' :
         'begin' fun(x,g,ok);
            'if' ok 'then' f:= g * u2 'fi'
         'end';

         x 'of' nlp:= x[1:m - p];
         'if' init 'of' info 'of' nlp
         'then' 'vec' f = f 'of' info 'of' nlp,
                'mat' b = b 'of' info 'of' nlp;
            f:= f * u2;
            'for' i 'to' m - p 'do' b[,i]:= b[,i] * u2 'od'
```

```
        'fi';
        (nlssolver 'of' mi 'of' mn)(mn);
        'if' er < 10
        'then' x 'of' nlp:= x 'of' nlp - (asv 'sol' g)['at' m - p + 1];
          f 'of' info 'of' nlp:= u2 * f 'of' info 'of' nlp
        'fi';
        'if' print 'of' mi 'of' mn
        'then' monitor(3,
          'auxil'(x 'of' nlp, 'nil', info 'of' nlp, 0, 'nil', 'nil',
              'nil', 'nil', 'nil', 'nil', 'nil', 'nil', exit),
          sinfo 'of' nlp, 'false')
        'fi';
        fun 'of' nlp:= fun;
        f 'of' info 'of' nlp:= g; init 'of' info 'of' nlp:= 'false'
      'else' 'int' m = 'upb' x, p = 1 'upb' la;
        'if' 2 'upb' la /= m 'then' exit(20)
        'elif' 'upb' lb /= p 'then' exit(21) 'fi';
        'mat' a0 = 0 'into' gensquare(m); 'svd' asv0, asv;
        a0[1:p, 1:m] := la;
        'if' 'not' 'check'(asv0:= 'svdec' a0) 'then' exit(6) 'fi';
        asv:= ('max'(sngval 'of' asv0) * small real * 10) 'trims' asv0;
        'if' 'upb'(sngval 'of' asv) /= p 'then' exit(20) 'fi';
        'vec' sol = asv 'sol' lb;
        'mat' v2 = (v 'of' asv0)[1:m,p + 1 : m];
        'if' init 'of' info 'of' nlp
        'then' 'mat' b = b 'of' info 'of' nlp;
          'for' i 'to' m - p 'do' b[i,]:= b[i,] * v2 'od'
        'fi';
        'func' fun:= fun 'of' nlp;
        fun 'of' nlp:= ('vec' x, f, 'ref''bool' ok) 'void' :
        fun(sol + v2 * x,f,ok);
        x 'of' nlp:= (x - sol) * v2;
        x 'of' nlp:= sol +
                     v2 * (nlssolver 'of' mi 'of' mn)(mn);
        'if' er < 10
        'then' f 'of' info 'of' nlp:= f 'of' info 'of' nlp +
          (la * x 'of' nlp - lb)['at' m - p + 1]
        'fi';
        'if' print 'of' mi 'of' mn
        'then' monitor(3,
          'auxil'(x 'of' nlp, 'nil', info 'of' nlp, 0, 'nil', 'nil',
              'nil', 'nil', 'nil', 'nil', 'nil', 'nil', exit),
          sinfo 'of' nlp, 'false')
        'fi';
        fun 'of' nlp:= fun; init 'of' info 'of' nlp:= 'false'
      'fi';
end : x 'of' nlp
'end' 'pr' fedx 'pr';

'skip'

'end'
```

```
'begin'

'proc' gas = ('metnlsjc' mn) 'vec' :
'pr' xdef gas 'pr'
'begin'

      'proc' genranvec = ('int' n) 'vec' :
      'begin' 'proc' ran = ('int' i) 'real' : next random(setr) - 0.5;
         'vec' v = ran 'into' genvec(n);
         v /< 'nrm' v
      'end';


      'proc' cdatasv = 'void' :
      'begin' 'svd' svd;
         gamma:= 'if' it - mnit = 1 'then' w:= b * v; 1
         'else' ctjc +:= 1; jacob(x,b,ok);
            'if' 'not' ok 'then' exit(11) 'fi';
            'real' nrmw = 'nrm' (w - (w:= b * v));
            'if' nrmdx < small real * nrmw 'then' 1/small real
            'else' nrmw/nrmdx
            'fi'
         'fi';
         'if' 'not' 'check' (ctsv +:= 1; svd:= 'svdec' b)
         'then' exit(6) 'fi';
         'real' maxval = 'max' sngval 'of' svd;
         'real' ck = maxval * epsrj + epsaj;
         svd:= ck 'trims' svd;
         rk:= 'upb' sngval 'of' svd;
         'if' rk = 0 'then' exit(7) 'fi';
         eta:= 1/(sngval 'of' svd)[rk]; e:= ck * eta;
         'if' e > 1 - small real 'then' e:= 1 - small real 'fi';
         dx:= svd 'sol' f; nrmdx:= 'nrm' dx;
         kappa:= maxval * nrmdx/slevel;
         'if' 'nrm' (f * u 'of' svd)< epsf 'then' er:= 3 'fi';
         'skip'
      'end';

      'proc' levelfu = ('vec' x, f) 'real' :
      'begin' fun(x,f,ok); cntf +:= 1;
        'if' 'not' ok 'then' exit(11) 'fi'; 'nrm' f
      'end';

      'proc' exit = ('int' err) 'void' :
      'begin' er:= err; end 'end';

      'nlsprbjc' nlp = nlp 'of' mn;
      'ref''info' info = info 'of' nlp;
      'vec' x = x 'of' nlp, 'func' fun = fun 'of' nlp,
      'jacob' jacob = jac 'of' nlp,
```

```
'real' dlf = dlf 'of' info,
       dlrx  = dlrx 'of' info,
       dlax = dlax 'of' info,
       epsrf = epsrf 'of' info,
       epsaf = epsaf 'of' info,
       epsrj = epsrj 'of' info,
       epsaj = epsaj 'of' info,
'ref''real' slevel = nrmf 'of' info,
            nrmdx  = nrmdx 'of' info,
'ref''int' it   = ctit 'of' info,
           cntf = ctfu 'of' info,
           ctjc = ctjc 'of' info,
           ctsv = ctsv 'of' info,
           er   = er   'of' info,
'int'  maxits = maxits 'of' mi 'of' mn,
       mnit = ctit 'of' info,
'bool' print = print 'of' mi 'of' mn;
'int' n = 'upb' x,
'real' e:= 0, gamma:= 1, eta:= 1, kappa:= 1,
       epsf,
       nrmx:= 'nrm' x,
'bool' slow:= 'false', ok:= 'true',
'vec' dx;
'auxil' aux = (x, dx, info, maxits, 'heap' 'real' := 1,
gamma, kappa, epsf, e, 'heap' 'real' := 0, nrmx, eta, slow, exit);

'if' init 'of' info
'then' slevel:= 'nrm' f 'of' info;
   'if' slevel <= epsaf 'then' exit(0) 'fi';
   epsf:= (epsrf + small real) * slevel + epsaf
'else' 'vec' f:= genvec(n); 'mat' b:= gensquare(n);
   f 'of' info:= f; b 'of' info:= b;
   slevel:= levelfu(x,f);
   'if' slevel <= epsaf 'then' exit(0) 'fi';
   epsf:= (epsrf + small real) * slevel + epsaf;
   ctjc +:= 1; jacob(x,b,ok);
   'if' 'not' ok 'then' exit(12) 'fi';
   init 'of' info:= 'true'
'fi';

'mat' b = b 'of' info, 'vec' f = f 'of' info;
'int' setr:= maxint 'over' n;
'int' rk:= n, 'vec' v:= genranvec(n), xj, w;
'if' print 'then'
   monitor(0, aux, sinfo 'of' nlp, 'false')
'fi';

'while'
   'if' it = mnit 'then' 'true' 'else' 'not' stopspl(aux) 'fi'
'do' it+:= 1; cdatasv; nrmx:= 'nrm' (x -< dx);
```

```
            'real' sl0:= slevel; slevel:= levelfu(x,f);
            epsf:= (epsrf + small real) * slevel + epsaf;
            'if' 'abs'(sl0 - slevel) > epsf 'then' slow:= 'false'
            'elif' slow 'then' er:= 2 'else' slow:= 'true' 'fi';
            'if' print 'then' monitor(1, aux, sinfo 'of' nlp, 'false') 'fi'
        'od';
end :   'if' print 'then' monitor(2, aux, sinfo 'of' nlp,
                    'mat'(la 'of' rinfo 'of' nlp) 'isnt' 'mat'('nil')) 'fi';
        'if' er /= 0 'then' error(er) 'fi';
        x
'end'
'pr' fedx 'pr';

'skip'

'end'
```

```
'begin'

'proc' abu = ('metnlsjc' mm) 'vec' :
'pr' xdef abu 'pr'
'begin'

      'proc' genranvec = ('int' n) 'vec' :
      'begin' 'proc' ran = ('int' i) 'real' : next random(setr) - 0.5;
         'vec' v = ran 'into' genvec(n);
         v /< 'nrm' v
      'end';

      'proc' conupdjac = 'bool' :
      'if' it - mnit < 3 'or' 'not' update 'or' e >= 0.1 'then' 'false'
      'else' 'vec' df:= f - f0; 'vec' u = lud 'sol' df;
         'real' pu = dx * u, nrmu = 'nrm' u;
         'if' e < 1
         'then' e:= (e/(1 - e) + ( 1 + nrmdx * 1.5/nrmu) * nrmdx * om)
                    * (1 + e)
         'fi';
         'if' kappa * e < 1 'and' 'abs' pu > nrmdx * nrmu * small real
              'and' e < 0.1
         'then' df +< labda * f0;
            'for' j 'to' 'upb' u 'do' b[,j] +< u[j]/pu * df 'od';
            'true'
         'else' 'false' 'fi'
      'fi';

      'proc' cdatalr = 'void' :
      'begin' 'vec' d01;
         'if' it - mnit > 1
         'then' d01:= lud 'sol' f;
            anl:= 'if' conupdjac 'then' 'false'
                  'else' cntj +:= 1; jacob(x,b,ok);
                     'if' 'not' ok 'then' exit(11) 'fi';
                     'true'
                  'fi'
         'fi';
         'if' 'not' 'check' (ctlu +:= 1; lud:= 'dec' b)
         'then' exit(5) 'fi';
         'if' it - mnit = 1
         'then' beta:= 'nrm' (dx:= lud 'sol' f); om:= 1
         'else' 'real' om1:= 'nrm'((lud 'sol' f0) * labda + dx)
                             /nrmdx ** 2;
            beta:= 'nrm'(dx:= lud 'sol' f);
            om:= 'nrm'(d01 - dx)/(nrmdx * beta);
            'if' om1 > om 'then' om:= om1 'fi'
         'fi';
         nrmdx:= beta; f0:= 'copy' f; kappa:= 'maxabs' b * nrmdx/slevel;
         eta:= 'nrm'(lud 'sol' v);
```

```
   'if' anl
   'then' e:= ('maxabs' b * (epsrj + small real * n * 16) + epsaj)
              * eta;
      'if' e > 1 - small real 'then' e:= 1 - small real 'fi'
   'fi'
'end';

'proc' levelfu = ('vec' x, f) 'real' :
'begin' fun(x,f,ok); cntf +:= 1;
   'if' ok 'then' 'nrm' f 'else' max real 'fi'
'end';

'proc' exit = ('int' err) 'void' :
'begin' er:= err; end 'end';

'nlsprbjc' nlp = nlp 'of' mn;
'ref''info' info = info 'of' nlp,
'ref''scalinfo' sinfo = sinfo 'of' nlp;
'vec' x = x 'of' nlp,
'ref''vec' xfacs = xfacs 'of' sinfo,
           ffacs = ffacs 'of' sinfo,
'func' fun:= fun 'of' nlp,
'jacob' jacob:= jac 'of' nlp,
'real' dlf = dlf 'of' info,
       dlrx = dlrx 'of' info,
       dlax = dlax 'of' info,
       epsrf = epsrf 'of' info,
       epsaf = epsaf 'of' info,
       epsrj = epsrj 'of' info,
       epsaj = epsaj 'of' info,
'ref''real' slevel = nrmf 'of' info,
            nrmdx  = nrmdx 'of' info,
'ref''int' it   = ctit 'of' info,
           cntf = ctfu 'of' info,
           cntj = ctjc 'of' info,
           ctlu = ctlu 'of' info,
           er   = er   'of' info,
'int'  maxits = maxits 'of' mi 'of' mn,
       mnit = ctit 'of' info,
'bool' print = print 'of' mi 'of' mn,
       update = updok 'of' mi 'of' mn,
'ref''bool' xscal = xscal 'of' sinfo,
            fscal = fscal 'of' sinfo;
'int' n = 'upb' x,
'real' e:= 0, eta:= 1, kappa:= 1, beta:= 1, om:= 1, labda:= 1,
       epsf, nrmx,
'bool' anl:= 'true', ok:= 'true',
'vec' dx;
'auxil' aux = (x, dx, info, maxits, labda, om, kappa,
               epsf, e, 'heap' 'real' := 1, nrmx, eta, anl, exit);
```

```
'if' init 'of' info
'then'
   (scale 'of' sinfo)(x,info,sinfo);
   'if' sing 'of' sinfo 'then' exit(5) 'fi';
   slevel:= 'nrm' f 'of' info;
   'if' slevel <= epsaf 'then' exit(0) 'fi';
   epsf:= (epsrf + small real) * slevel + epsaf
'else' 'vec' f:= genvec(n); 'mat' b:= gensquare(n);
   f 'of' info:= f; b 'of' info:= b;
   slevel:= levelfu(x,f);
   'if' slevel <= epsaf 'then' exit(0)
   'elif' slevel = maxreal 'then' exit(12) 'fi';
   epsf:= (epsrf + small real) * slevel + epsaf;
   jacob(x,b,ok); cntj +:= 1;
   'if' 'not' ok 'then' exit(12) 'fi';
   init 'of' info:= 'true';
   (scale 'of' sinfo)(x,info,sinfo);
   'if' sing 'of' sinfo 'then' exit(5) 'fi';
   slevel:= 'nrm' f 'of' info;
   'if' slevel <= epsaf 'then' exit(0) 'fi';
   epsf:= (epsrf + small real) * slevel + epsaf
'fi';

'if' fscal 'or' xscal
'then' fun:= ('vec' x,f,'ref''bool' ok) 'void' :
   'begin' 'if' xscal
      'then' 'vec' x1 = genvec(n);
         'for' i 'to' n 'do' x1[i]:= x[i] * xfacs[i] 'od';
         (fun 'of' nlp)(x1,f,ok)
      'else' (fun 'of' nlp)(x,f,ok)
      'fi';
      'if' fscal 'and' ok
      'then' 'for' i 'to' n 'do' f[i] *:= ffacs[i] 'od'
      'fi'
   'end';

   jacob:= ('vec' x, 'mat' b, 'ref''bool' ok) 'void' :
   'begin' 'if' xscal
      'then' 'vec' x1 = genvec(n);
         'for' i 'to' n 'do' x1[i]:= x[i] * xfacs[i] 'od';
         (jac 'of' nlp)(x1,b,ok);
         'if' ok 'then' 'for' i 'to' n 'do' b[,i] *< xfacs[i] 'od'
         'fi'
      'else' (jac 'of' nlp)(x,b,ok)
      'fi';
      'if' fscal 'and' ok
      'then' 'for' i 'to' n 'do' b[i,] *< ffacs[i] 'od' 'fi'
   'end'
'fi';
```

```
      'mat' b = b 'of' info, 'vec' f = f 'of' info;
      'int' setr:= maxint 'over' n;
      'int' rk:= n, 'vec' v:= genranvec(n), f0, 'lud' lud;
      nrmx:= 'nrm' x;
      'proc'('auxil') 'bool' stop =
         'if' safe 'of' mi 'of' mn 'then' stopful 'else' stopspl 'fi';
      'if' print 'then' monitor(0, aux, sinfo, 'false') 'fi';

      'while' 'if' it = mnit 'then' 'true' 'else' 'not' stop(aux) 'fi'
      'do' it+:= 1; cdatalr; resbis(aux,levelfu);
         nrmx:= 'nrm' x;
         'if' 'not' anl 'and' er /= 0
         'then' er:= 0; e:= 1 - small real 'fi';
         epsf:= (small real + epsrf) * slevel + epsaf;
         'if' print 'then' monitor(1, aux, sinfo, 'false') 'fi'
      'od';
end : (bckscale 'of' sinfo)(x,info,sinfo);
      'if' print 'then' monitor(2,aux,sinfo,
                'mat'(la 'of' rinfo 'of' nlp) 'isnt' 'mat'('nil')) 'fi';
      'if' er /= 0 'then' error(er) 'fi';
      x
'end'
'pr' fedx 'pr';

'skip'

'end'



'begin'

'proc' snoleqj = ('metnlsjc' mn) 'vec' :
'pr' xdef snoleqj 'pr'
'begin' 'ref''int' er = er 'of' info 'of' nlp 'of' mn;
      'int' maxtot:= maxits 'of' mi 'of' mn;
      abu(mn 'maxits' maxtot 'over' 2);
      mn 'maxits' maxtot;
      'if' er > 0 'and' er < 10 'then' er:= 0; gas(mn) 'fi';
      x 'of' nlp 'of' mn
'end'
'pr' fedx 'pr';

'skip'

'end'
```

```
'begin'

'op' 'nlssolve' = ('vec' x, 'metnlsjc' mn) 'vec' :
'pr' xdef nsvmnsj 'pr'
'begin' 'proc' exit = ('int' er) 'void' :
      'begin' error(er); end 'end';

      'nlsprbjc' nlp = nlp 'of' mn;
      'ref''info' info = info 'of' nlp,
      'mat' la = la 'of' rinfo 'of' nlp,
      'vec' lb = lb 'of' rinfo 'of' nlp;

      'ref''real' epsrf = epsrf 'of' info,
                  epsaf = epsaf 'of' info,
                  epsrj = epsrj 'of' info,
                  epsaj = epsaj 'of' info,
                  dlf   = dlf   'of' info,
                  dlrx  = dlrx  'of' info,
                  dlax  = dlax  'of' info,
      'int' er = er 'of' info;

      'if' epsrf < small real 'then' error(13); epsrf:= small real 'fi';
      'if' epsaf < 0          'then' error(14); epsaf:= 0          'fi';
      'if' epsrj < small real 'then' error(15); epsrj:= small real 'fi';
      'if' epsaj < 0          'then' error(16); epsaj:= 0          'fi';
      'if' dlf   < epsaf      'then' error(17); dlf  := epsaf      'fi';
      'if' dlrx  < small real 'then' error(18); dlrx := small real 'fi';
      'if' dlax  < small real 'then' error(19); dlax := small real 'fi';

     x 'of' nlp:= x;
     'if' (la 'is' 'mat'('nil'))
     'then' (nlssolver 'of' mi 'of' mn)(mn)
     'elif' lb 'is' 'vec'('nil')
     'then' 'int' m = 1 'upb' la, p = 2 'upb' la;
        'mat' a0 = 0 'into' gensquare(m);
        a0[1:m,1:p]:= la; 'svd' asv0, asv;
        'if' 'not' 'check'(asv0:= 'svdec' a0) 'then' exit(6) 'fi';
        asv:= ('max'(sngval 'of' asv0)* small real * 100) 'trims' asv0;
        'if' 'upb' (sngval 'of' asv) /= p 'then' exit(20) 'fi';
        v 'of' asv:= (v 'of' asv)[1:p,];
        'mat' u2 = (u 'of' asv0)[,p + 1:m];
        'vec' g = genvec(m), 'mat' b1 = genmat(m,m - p);
        'func' fun:= fun 'of' nlp;

        fun 'of' nlp:= ('vec' x,f, 'ref''bool' ok) 'void' :
        'begin' fun(x,g,ok);
           'if' ok 'then' f:= g * u2 'fi'
        'end';

        'jacob' jacob:= jac 'of' nlp;
```

```
    jac 'of' nlp:= ('vec' x, 'mat' b, 'ref''bool' ok) 'void' :
    'begin' jacob(x,b1,ok);
       'if' ok
       'then' 'for' i 'to' m - p 'do' b[,i]:= b1[,i] * u2 'od' 'fi'
    'end';

    x 'of' nlp:= x[1:m - p];
    'if' init 'of' info
    'then' 'vec' f = f 'of' info,
           'mat' b = b 'of' info;
       f:= f * u2;
       'for' i 'to' m - p 'do' b[,i]:= b[,i] * u2 'od'
    'fi';
    (nlssolver 'of' mi 'of' mn)(mn);
    'if' er < 10
    'then' x 'of' nlp:= x 'of' nlp - (asv 'sol' g)['at' m - p + 1];
       f 'of' info:= u2 * f 'of' info
    'fi';
    'if' print 'of' mi 'of' mn
    'then' monitor(3,
       'auxil'(x 'of' nlp, 'nil', info 'of' nlp, 0, 'nil', 'nil',
           'nil', 'nil', 'nil', 'nil', 'nil', 'nil', exit),
       sinfo 'of' nlp, 'false')
    'fi';
    fun 'of' nlp:= fun; jac 'of' nlp:= jacob;
    f 'of' info:= g; init 'of' info:= 'false'
'else' 'int' m = 'upb' x, p = 1 'upb' la;
    'if' 2 'upb' la /= m 'then' exit(20)
    'elif' 'upb' lb /= p 'then' exit(21) 'fi';
    'mat' a0 = 0 'into' gensquare(m); 'svd' asv0, asv;
    a0[1:p, 1:m] := la;
    'if' 'not' 'check'(asv0:= 'svdec' a0) 'then' exit(6) 'fi';
    asv:= ('max'(sngval 'of' asv0) * small real * 10) 'trims' asv0;
    'if' 'upb'(sngval 'of' asv) /= p 'then' exit(20) 'fi';
    'vec' sol = asv 'sol' lb;
    'mat' v2 = (v 'of' asv0)[1:m,p + 1 : m];
    'mat' b1 = genmat(m - p,m);
    'if' init 'of' info
    'then' 'mat' b = b 'of' info;
       'for' i 'to' m - p 'do' b[i,]:= b[i,] * v2 'od'
    'fi';
    'func' fun:= fun 'of' nlp;
    fun 'of' nlp:= ('vec' x, f, 'ref''bool' ok) 'void' :
    fun(sol + v2 * x,f,ok);

    'jacob' jacob:= jac 'of' nlp;
    jac 'of' nlp:= ('vec' x, 'mat' b, 'ref''bool' ok) 'void' :
    'begin' jacob(sol + v2 * x, b1,ok);
       'if' ok
       'then' 'for' i 'to' m - p 'do' b[,i]:= b1 * v2[,i] 'od' 'fi'
```

```
            'end';

            x 'of' nlp:= (x - sol) * v2;
            x 'of' nlp:= sol +
                         v2 * (nlssolver 'of' mi 'of' mn)(mn);
            'if' er < 10
            'then' f 'of' info:= f 'of' info +
               (la * x 'of' nlp - lb)['at' m - p + 1]
            'fi';
            'if' print 'of' mi 'of' mn
            'then' monitor(3,
               'auxil'(x 'of' nlp, 'nil', info 'of' nlp, 0, 'nil', 'nil',
                   'nil', 'nil', 'nil', 'nil', 'nil', 'nil', exit),
               sinfo 'of' nlp, 'false')
            'fi';
            fun 'of' nlp:= fun; jac 'of' nlp:= jacob;
            init 'of' info:= 'false'
         'fi';
end : x 'of' nlp
'end' 'pr' fedx 'pr';

'skip'

'end'
```

INDEX

of mode, operator, field and routine names. [ ] refers to definitions and descriptions in Section 2. ( ) refers to page numbers