stichting
mathematisch
centrum

MC

AFDELING NUMERIEKE WISKUNDE
(DEPARTMENT OF NUMERICAL MATHEMATICS)

NW 88/80 AUGUSTUS

J.C.P. BUS

AN ALGOL 60 PACKAGE FOR THE SOLUTION OF NONLINEAR EQUATIONS

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, 65H04.

An ALGOL 60 package for the solution of nonlinear equations

by

J.C.P. Bus

ABSTRACT

This paper is meant to be a user manual for an ALGOL 60 software package. This package contains procedures for solving systems of nonlinear equations using Newton-like methods. The functions involved in these equations are assumed to be differentiable. The software package is based on the NUMAL software library.

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

CONTENTS

1. INTRODUCTION

This paper contains a description of an ALGOL 60 package for the solution of systems of nonlinear equations. It gives implementations of the algorithms SNOLEQ and SNOLEQJ, which are defined and evaluated in [1].
This software package has the following structure.

A. Procedures to be used if one has available a procedure which defines the analytic jacobian (matrix of partial derivatives).
These are:

reducej — to be used if one or more equations are linear in all variables;

oneqj — for finding a solution of one nonlinear equation in one variable; a bracketing interval is not required.

snoleqj — for finding a solution of n (n > 1) nonlinear equations in n variables.

reducej creates a smaller problem with nonlinear equations only, which is to be solved by oneqj if it appears to be a one-dimensional problem, and by snoleqj otherwise.
oneqj uses the procedure zeroinder (see [2]) as soon as an interval is found which brackets a solution.
snoleqj is described in [1] and uses a restrained Newton algorithm with a possibility of explicit scaling and conditional updating called SCABU in [1] and a strict generalized Newton algorithm called GAS in [1].

B. Procedures to be used if no procedure is available which defines the analytic jacobian.
These are:

reduce — to be used if one or more equations are linear in all variables;

oneq — for finding a solution of one nonlinear equation in one variable; a bracketing interval is not required;

snoleq — for finding a solution of n (n > 1) nonlinear equations in n variables.

reduce creates a smaller problem with nonlinear equations only, which is to be solved by oneq if it appears to be a one-dimensional problem, and by snoleq otherwise.
oneq uses the procedure zeroin (see [2]) as soon as an interval is found which brackets a solution.

snoleq is described in [1] and uses a restrained difference Newton algorithm with a possibility of explicit scaling and conditional updating called SCDBU in [1] and a strict generalized difference Newton algorithm called GDS in [1].

C. Auxiliary procedures:

matrix and vector operations:
sum, nrm, cnddiag, scale, backscale, tridec, trisol, svdec, svsol.
some modules for the newton-like algorithms which are used by at least two of the newton-like procedures:
resbis, stopful, stopspl.
some procedures for output:
outvec, outmat, monitor1, monitor2.

2. PROBLEM DESCRIPTION

Let be given a function F: D --> S, with D and S open sets in the n-dimensional real space. Suppose that F has continuous first derivatives on D, although these do not have to be known explicitly. Let J(x) be tne jacobian matrix of partial first derivatives of F. Let eps denote the machine precision and fl(A) the value of the expression A evaluated with precision eps. Suppose there exist nonnegative constants eprf, epaf, eprj and epaj such that

$$nrm(F(x) - fl(F(x))) < eprf * nrm(F(x)) + epaf,$$

$$nrm(J(X) - fl(J(x))) < eprj * nrm(J(x)) + epaj,$$

$$eprf >= eps,\ eprj >= eps,\ epaf >= 0,\ epaj >= 0,$$

where nrm denotes the euclidean (for vectors) or spectral (for matrices) norm.
Then the problem is to compute an approximation xk to a solution xs of the equation: F(x) = 0, such that

$$nrm(xs - xk) <= dlrx * nrm(xk) + dlax,$$

$$nrm(F(xk)) <= dlf,$$

for prescribed precisions dlrx, dlax and dlf. It is assumed that dlrx, dlax and dlf are chosen properly with respect to eprf, epaf, eprj and epaj.
An initial guess to the solution is assumed to be given.

If n-p (for p < n) of the function components are linear in all variables, then tne problem can be denoted by

$$F(x) = 0,$$
$$Ax = b,$$

where F(x) is a p-vector, A is a (n-p) by n matrix and b a (n-p)-vector. In such a case the problem can be reduced to a problem of p-th order (see [1]).

3. DESCRIPTION OF PROCEDURES

3.1. Key parameters of procedures

n: <arithmetic expression>
the number of variables and equations of the problem.

p: <arithmetic expression>
the number of nonlinear components of the function.

x: <array identifier>
"array" x[1:n];
entry: the initial guess to the solution;
exit: if the run is successful (see description of res[1]) then the approximation of the solution.

fun: <procedure identifier>
"boolean" "procedure" fun(n, x, f); "value" n;
"integer" n; "array" x, f;
This user supplied procedure has to deliver in "array" f[1:p] the function vector evaluated at x (given in "array" x[1:n]). Note that p = n if no linear functions are specified. If x lies outside the domain of the function then fun should deliver the value "false", which causes regular error termination; otherwise "true" has to be delivered. Note that this provides a possibility to avoid arithmetic overflow, for instance if the process diverges to infinity.

jacob: <procedure identifier>
"procedure" jacob(n, x, jac); "value" n;
"integer" n; "array" x, jac;
This user supplied procedure should deliver:
jac[i,j] = dF[i]/dx[j] (i=1,..., p; j=1,..., n),
where F[i] denotes the i-th component of F(x) and x[j] the j-th variable. It is assumed that this procedure supplies the analytic derivatives. If these are not known, then procedures have to be used which do not require this procedure.

la: <array identifier>
"array" la[1:n-p, 1:n];
The matrix A of the linear part has to be given in la (see section 2).

lb: <array identifier>
"array" lb[1:n-p];
The right hand side b of the linear part has to be given in lb (see section 2).

prec: <array identifier>
"array" prec[1:7];
One should specify the required precisions in prec as follows:
prec[1]: dlf;
prec[2]: dlrx;
prec[3]: dlax;
prec[4]: eprf;
prec[5]: epaf;
prec[6]: eprj;
prec[7]: epaj.

res: <array identifier>
"array" res[1:10];
In this array some auxiliary results are delivered.
res[1] : sum(r[i] * 10 ** (2(p-i)), i = 1, ..., p), where p (= 1, 2, or 3) gives the number of calls of a Newton-like program and r[i] gives the report number of the i-th program. If r[p] = 0 (res[1] = 0 (mod 100)) then the total run is successful. The report numbers r[i] have the following meaning:
0 : successful;
1 : no progress, maybe due to too high required precision;
2 : no progress relative to error in function;
3 : stationary point of norm of the function, no solution;
4 : too many function evaluations or iterations required;
5 : numerical singularity in triangular decomposition;
6 : failure of singular value decomposition;
7 : rank of jacobian approximately equal to zero;
8 : error in jacobian approximation yields a possibly singular jacobian (rank unequal zero);
9 : nearby singularity of jacobian expected;
10: difference approximation impossible, point on boundary of domain;
11: divergence out of domain of function, call of fun delivered "false";
12: starting point not in domain of the function;
13: the matrix of the linear components is not of full rank;
14: oneq or oneqj obtains no solution.
res[2] : nrm(F(xk)), with xk the approximated solution (see section 2);
res[3] : total number of iteration steps performed;
res[4] : total number of triangular decompositions performed;
res[5] : total number of singular value decompositions performed;
res[6] : total number of calls of fun;
res[7] : total number of calls of jacob;
res[8] : condition of row scaling matrix if row scaling is performed, otherwise 1;
res[9] : condition of column scaling matrix if column scaling is performed, otherwise 1;
res[10]: an estimation of the condition number of the jacobian approximation.

methods: <procedure identifier>
"procedure" methods(clasi, geni, scale, update);
"boolean" clasi, geni, scale, update;
With methods one may choose the values of the booleans clasi, geni, scale and update unequal to their default values. These identifiers have the following meaning:
clasi : use of a Newton-like method with triangular decomposition is allowed;
default: clasi = "true";
geni : use of a Newton-like method with singular value decomposition is allowed;
default: geni = "true";
scale : explicit scaling is allowed in the Newton-like algorithms that use triangular decomposition.
default: scale = "false";
update: conditional updating is allowed in restrained (difference) Newton methods;
default: update = "true".
Note that scaling may be favourable sometimes but not always, moreover the stopping criterion on the vector of variables is based on the scaled function and variables. Updating is usually favourable except when very high precision (almost machine precision) is required. If clasi and geni are both given the value "false", then the procedure terminates without applying any method for calculating a solution and without error message. For further information about the usefulness of these options see [1].

monitor: <procedure identifier>
This procedure provides a possibility of printing intermediate results, so that progress can be shown. This procedure can also be used to keep control on the iteration process. However, one should be careful in doing so, as the process can easily be disturbed. The heading of monitor reads:

```
"procedure" monitor(ph, n, ic, imax, dc, fc, jc, x, f,
    nrmf, b, upd, scl, rows, cols, lab, om, bt, kp,
    nbi, e, er, hs);
"value" ph, n, dc, fc, jc, scl, lab, hs;
"integer" ph, n, ic, imax, dc, fc, jc, scl, er;
"boolean" upd;
"real" nrmf, lab, om, bt, kp, nbi, e, hs;
"array" x, f, b, rows, cols;
```

The meaning of the parameters is:
ph : phase in which the procedure is called;
ph = 0: called in initial phase;
ph = 1: at the end of an iteration step;
ph = 2: after completion.
n : number of variables and functions.
ic : the iteration index.
imax: the maximum number of iteration steps allowed.
dc : the number of matrix decomposition performed so far.

fc : the number of function evaluations used so far.
jc : the number of jacobian evaluations used so far.
x : x[1:n]; the vector of variables.
f : f[1:n]; the function vector F(x) at x.
nrmf: nrm(F(x)).
b : b[1:n,1:n]; the approximation to the jacobian in snoleq(j); in reduce(j) the n*p matrix which defines the projection for the projected function (see [1]).
upd : boolean which indicates whether conditional updating is allowed.
scl : indicator for scaling;
scl = 0: no scaling performed;
scl = 1: only row (function) scaling is performed;
scl = 2: only column (variable) scaling is performed;
scl = 3: row and column scaling is performed.
rows: rows[1:n]; row scaling factors if scl > 0.
cols: cols[1:n]; column scaling factors if scl > 0.
lab : step length factor labda.
om : approximation to Lipschitz constant omega.
bt : approximation to beta.
kp : approximation to condition number kappa.
nbi : approximation to norm of inverse of b.
e : approximation to error in b relative to J(x).
er : integer indicating whether an error is detected; if er /= 0 then the procedure will terminate at the end of the current step.
hs : step length used in the difference formulas.

For further explication of these values see [1].
Two example monitor procedures are described in subsection 3.2. Note that monitor can also be used to change the values of some parameters in order to control the process in some specific way. However, care should be taken with such usage.

f: <array identifier>
"array" f[1:n];
entry and exit: the function value F(x).

b: <array identifier>
"array" b[1:n,1:n];
entry and exit: an approximation to J(x).

errex: <procedure identifier>
"procedure" errex(i); "value" i; "integer" i;
errex is called in situations that an error (with number i) is detected. It is sometimes used to perform a jump to an appropriate label. This occurs only in auxiliary procedures of the package.

3.2. Auxiliary procedures

3.2.1. OUTVEC

```
"procedure" outvec(ch, x, k, u, m, e, s);
"value" ch, k, u, m, e;
"integer" ch, k, u, m, e; "array" x; "string" s;
"code" 90001;
```

outvec prints the vector x[k:u] on channel ch with mantissa length m and e digits in the exponent, preceded by the string s.

3.2.2. OUTMAT

```
"procedure" outmat(ch, a, lr, ur, lc, uc, m, e, s);
"value" ch, lr, ur, lc, uc, m, e;
"integer" ch, lr, ur, lc, uc, m, e;
"array" a; "string" s;
"code" 90002;
```

outmat prints the matrix a[lr:ur,lc:uc] on channel ch with mantissa length m and e digits in the exponent, preceded by the string s.

3.2.3. SUM

```
"real" "procedure" sum(k, u, i, xi); "value" k, u;
"integer" k, u, i; "real" xi;
"code" 99800;
```

sum gives the sum of xi for i:= k (1) u.

3.2.4. NRM

```
"real" procedure" nrm(n, x); "value" n;
"integer" n; "array" x;
"code" 99801;
```

nrm calculates the norm of the vector x[1:n].

3.2.5. CNDDIAG

```
"real" "procedure" cnddiag(n, d); "value" n;
"integer" n; "array" d;
"code" 99802;
```

cnddiag calculates the spectral norm of the diagonal matrix given in d[1:n]. If this computation would cause overflow, then giant (see [2]) is delivered.

3.2.6. SCALE

```
"integer" "procedure" scale(n, b, x, f, rows, cols);
"value" n; "integer" n; "array" b, x, f, rows, cols;
"code" 99803;
```

scale performs scaling of the jacobian approximation b, given in b[1:n,1:n], the variables given in x[1:n] and the function vector f, given in f[1:n].
If on exit:
scale = -1, then a zero row or column is detected in b,
scale = 0, then no scaling is performed because the row and column norms are sufficiently near 1,
scale = 1, then only row scaling is performed,
scale = 2, then only column scaling is performed,
scale = 3, then row and column scaling is performed.
Note that row scaling implies scaling of the equations and column scaling implies scaling of the variables. The row scaing factors are given in rows[1:n]; the column scaling factors in cols[1:n]. These factors are equal to 1 if no scaling is performed. The scaling method is described in detail in [1].

3.2.7. BACKSCALE

```
"procedure" backscale(n, b, x, f, scl, rows, cols);
"value" n, scl; "integer" n, scl; "array" b, x, f, rows, cols;
"code" 99804;
```

backscale performs backscaling of the jacobian approximation, given in b[1:n,1:n], the variables given in x[1:n] and the function vector given in f[1:n]. It is assumed that scaling has been performed by the procedure scale (see above),that the value of the integer expression scl equals the value delivered by that call of scale and that rows[1:n] and cols[1:n] contain the row scaling and column scaling factors delivered by that call of scale.

3.2.8. TRIDEC

```
"procedure" tridec(n, a, alr, pl, p2, aux, errex); "value" n;
"integer" n; "array" a, alr, aux;
"integer" "array" pl, p2;
"procedure" errex;
"code" 99805;
```

tridec performs a triangular decomposition of the matrix a, given in a[1:n,1:n]. The decomposition is delivered in the matrix alr[1:n,1:n] and the row and column pivoting indices in pl, p2[1:n], respectively. Tridec calls the procedure gsselm, described in [2]. The "array" aux[0:7] is the same as for gsselm. If the triangular decomposition cannot be performed because of a too small pivot, then errex(5) is called.

3.2.9. TRISOL

```
"procedure" trisol(n, lr, pl, p2, rhs, sol); "value" n;
"integer" n; "array" lr, rns, sol;
"integer" "array" pl, p2;
"code" 99806;
```

trisol performs forward and backward substitution to obtain the solution of the linear system whose triangularly decomposed form is given in lr[1:n,1:n]. The pivoting arrays as delivered by tridec (see above) have to be given in the arrays pl, p2[1:n], the right hand side in rns[1:n] and the solution is delivered in sol[1:n].

3.2.10. SVDEC

```
"procedure" svdec(n, a, val, v, em, errex); "value" n;
"integer" n; "array" a, val, v, em; "procedure" errex;
"code" 99812;
```

svdec performs a singular value decomposition of the matrix a, given in a[1:n,1:n]. The u-matrix is delivered in a, the v-matrix in v[1:n,1:n] and the singular values in val[1:n] in non-increasing order (see [1]). The auxiliary array em[0:7] is the same as for the procedure qrisngvaldec in [2]. If no singular value decomposition can be obtained within the prescribed precision or number of iteration steps then errex(6) is called.

3.2.11. SVSOL

```
"procedure" svsol(n, u, val, v, r, rhs, sol); "value" n, r;
"integer" n, r; "array" u, val, v, rhs, sol;
"code" 99813;
```

svsol computes the solution of the linear system whose singular value decomposition is given in u[1:n,1:r], val[1:r] and v[1:n,1:r] (see [1]), where r is the rank of the matrix. The right hand side of the linear system has to be given in rhs[1:n] and the solution is delivered in sol[1:n].

3.2.12. RESBIS

```
"procedure" resbis(n, x, dx, nrmdx, labda, f, level, lfun, er,
epf, cnt); "value" n; "integer" n, er, cnt;
"real" nrmx, nrmdx, labda, level, epf;
"array" x, dx, f; "real" "procedure" lfun;
"code" 99809;
```

resbis performs the restraining with bisection in each step of the Newton-like process (see [1]). It searches for a step length factor labda such that the level function, defined by lfun, in the point defined by x - labda * dx, is less than its value at x (given by the value of level). On exit, x and f contain the vector x - labda * dx and the function vector at this point. The values of nrmx and nrmdx are equal to the norms of x and dx on entry as well as on exit. The value of er indicates error performance and cnt gives the number of evaluations of lfun. In epf an absolute tolerance on the numerical value of the function vector has to be given.

3.2.13. STOPFUL

```
"procedure" stopful(nrmdx, delx, nrmf, delf, epf, epaf, fix,
    kappa, e, omega, beta, labda, er, it, itmax, errex);
"value" delx, delf, epaf, fix;
"integer" it, itmax, er; "boolean" fix;
"real" nrmdx, delx, nrmf, delf, epf, epaf, e, kappa, omega,
    beta, labda;
"procedure" errex;
"code" 99810;
```

stopful evaluates the stopping criteria. It uses the special convergence criteria developed in [1]. nrmdx and nrmf denote the norms of the step length and of the function. delx and delf should specify the required precision in the variables and the function. epf gives the numerical precision of the function; epaf gives the absolute precision of the function. The meaning of fix, kappa, e, omega, beta, labda, it and itmax is explained in [1]. If er is not equal to zero on entry, then stopful calls errex, unless the convergence criteria are satisfied. stopful deliveres "true" if the convergence criteria are satisfied; it deliveres "false" if these criteria are not satisfied and no reasons for error exits are discovered. If some error is detected, then stopful calls errex.

5.2.14. STOPSPL

```
"boolean" "procedure" stopspl(nrmdx, dlx, nrmf, dlf, epaf, er,
    it, itmax, errex);
"value" dlx, nrmf, itmax;
"integer" er, it, itmax; "real" nrmdx, dlx, nrmf, dlf, epaf;
"procedure" exit;
"code" 99811;
```

stopspl evaluates the simple stopping criteria (see [1]). The meaning of the parameters is as in stopful. stopspl deliveres "true" if the simple stopping criteria are satisfied, otherwise it deliveres "false". It calls errex if er is not equal to zero and the stopping criteria are not satisfied.

3.2.15. MONITOR1

```
"procedure" monitor1(ph, n, ic, imax, dc, fc, jc, x, f, nrmf,
    b, upd, scl, rows, cols, lab, om, bt, kp, nbi, e, er, hs);
"value" ph, n, dc, fc, jc, scl, lab, hs;
"integer" ph, n, ic, imax, dc, fc, jc, scl, er;
"boolean" upd; "real" nrmf, lab, om, bt, kp, nbi, e, hs;
"array" x, f, b, rows, cols;
"code" 99901;
```

monitor1 performs printing of auxiliary results during the initialization phase, each iteration step and at the end of the run. In the initialization phase information is given about the scaling and a heading for the printing per step. In each iteration step one line of 108 symbols is printed. It gives the values of it, dc, fc, jc, nrmf, lb, om, bt, kp, nbi. At the end of the run the success or failure is reported and the approximate solution and function vector is printed. All results are printed on channel 71.

3.2.16. MONITOR2

```
"procedure" monitor2(ph, n, ic, imax, dc, fc, jc, x, f, nrmf,
    b, upd, scl, rows, cols, lab, om, bt, kp, nbi, e, er, hs);
"value" ph, n, dc, fc, jc, scl, lab, hs;
"integer" ph, n, ic, imax, dc, fc, jc, scl, er;
"boolean" upd; "real" nrmf, lab, om, bt, kp, nbi, e, hs;
"array" x, f, b, rows, cols;
"code" 99902;
```

monitor2 performs printing of auxiliary results during the initialization phase, each iteration step and at the end of the run. Besides some values of controlling parameters, this procedure also prints the value of the variables, the function components and the elements of the jacobian approximation. This extended printing requires a number of lines per step. The results are printed on channel 71.

3.3. Basic Newton-like programs

3.3.1. ABU (restrained Newton method with conditional updating)

```
"procedure" abu(n, x, f, b, fun, jacob, prec, res, monitor);
"value" n; "integer" n;
"array" x, f, b, prec, res;
"boolean" "procedure" fun; "procedure" jacob, monitor;
"code" 99850;
```

abu calculates an approximation to the solution of the nonlinear system defined by fun. Its analytic jacobian is given by jacob. An initial guess x0, the function value at x0 and the jacobian matrix at x0 should be given in x, f and b, respectively. The method used is a restrained Newton method with conditional updating, where the step length factor is calculated using bisection. This method is described in detail in [1]. For a detailed description of the parameters see section 3.1.

3.3.2. GAS (generalized strict Newton method)

```
"procedure" gas(n, x, f, b, fun, jacob, prec, res, monitor);
"value" n; "integer" n;
"array" x, f, b, prec, res;
"boolean" "procedure" fun; "procedure" jacob, monitor;
"code" 99851;
```

gas calculates an approximation to the solution of the nonlinear system defined by fun. Its analytic jacobian is given by jacob. An initial guess x0, the function value at x0 and the jacobian matrix at x0 should be given in x, f and b, respectively. The method used is a strict generalized Newton method, which is described in detail in [1]. For a detailed description of the parameters see section 3.1.

3.3.3. DBU (restrained difference Newton method with conditional updating)

```
"procedure" dbu(n, x, f, b, fun, prec, res, monitor);
"value" n; "integer" n;
"array" x, f, b, prec, res;
"boolean" "procedure" fun; "procedure" monitor;
"code" 99852;
```

dbu calculates an approximation to the solution of the nonlinear system defined by fun. An initial guess x0, the function value at x0 and an approximation to the jacobian matrix at x0 should be given in x, f and b, respectively. The method used is a restrained difference Newton method with conditional updating, which uses bisection to calculate the step length factor. This method is described in detail in [1]. A full description of the parameters is given in section 3.1.

3.3.4. GDS (strict generalized difference Newton method)

```
"procedure" gds(n, x, f, b, fun, prec, res, monitor);
"value" n; "integer" n;
"array" x, f, b, prec, res;
"boolean" "procedure" fun; "procedure" monitor;
"code" 99853;
```

gds calculates an approximation to the solution of the nonlinear system defined by fun. An initial guess x0, the function value at x0 and an approximation to the jacobian matrix at x0 should be given in x, f and b, respectively. The method used is a strict generalized difference Newton method, which is described in detail in [1]. For a full description of the parameters see section 3.1.

3.4. Programs for one equation in one variable.

3.4.1. ONEQJ (solving one equation with Newton's method)

```
"boolean" "procedure" oneqj(x, fu, dfu, dlf, dlrx, dlax);
"value" dlf, dlrx, dlax;
"real" x, dlf, dlrx, dlax; "real" "procedure" fu, dfu;
"code" 99858;
```

oneqj calculates an approximation to the solution of the equation defined by fu, with derivative defined by dfu.
The headings of these user-supplied procedures are:

```
"real" "procedure" fu(x); "value" x; "real" x;
"real" "procedure" dfu(x); "value" x; "real" x;
```

fu and dfu deliver the values of the function and its derivative at x.
If oneqj deliveres "false", then no solution is found within the required precision. This precision is defined by: abs(x) * dlrx + dlax or, if no points with different signs of function values can be found, then we use abs(fu(x)) < dlf.
oneqj tries to find an interval which brackets a solution with use of Newton's method. Once such an interval is found, the procedure zeroinder ([2]) is called.

3.4.2. ONEQ (solving one equation with secant method)

```
"boolean" "procedure" oneq(x, fu, dlf, dlrx, dlax);
"value" dlf, dlrx, dlax;
"real" x, dlf, dlrx, dlax; "real" "procedure" fu;
"code" 99859;
```

oneq calculates an approximation to the solution of the equation defined by fu. The heading of this procedure is:

```
"real" "procedure" fu(x); "value" x; "real" x;
```

fu delivers the value of the function at x.
If oneq deliveres "false", then no solution is found within the required precision. This precision is defined by: abs(x) * dlrx + dlax or, if no points with different signs of function values can be found, then we use abs(fu(x)) < dlf.
oneq tries to find an interval which brackets a solution with use of the secant method. Once such an interval is found, the procedure zeroin ([2]) is called.

3.5. User programs

3.5.1. SNOLEQJ

Solving n equations in n variables if the analytic jacobian is available.

```
"procedure" snoleqj(n, x, fun, jacob, prec, res, methods,
                    monitor);
"value" n; "integer" n;
"array" x, prec, res;
"boolean" "procedure" fun;
"procedure" jacob, methods, monitor;
"code" 99854;
```

snoleqj calculates a solution of a system of n equations in n variables using a poly-algorithm which is fully described in [1]. This algorithm uses ABU, a restrained Newton method with conditional updating (which performs triangular decomposition of the jacobian approximation), and GAS, a strict generalized Newton method (which performs singular value decompositon). If scaling is not allowed (default) then ABU is restricted to 40 iteration steps and when it fails to solve the problem then GAS is called (also at most 40 steps). If scaling is allowed then two calls of ABU are permitted (both restricted to 20 iteration steps) with scaled function and variables. Hence, rescaling may be performed once. If no solution is obtained then GAS is called with the unscaled function (40 steps). The options of scaling and conditional updating can be controlled by methods. This procedure can also be used to avoid calling of either ABU or GAS. A detailed description of the parameters of snoleqj is given in section 3.1.

3.5.2. SNOLEQ

Solving n equations in n variables if no analytic jacobian is available.

```
"procedure" snoleq(n, x, fun, prec, res, methods, monitor);
"value" n; "integer" n;
"array" x, prec, res;
"boolean" "procedure" fun;
"procedure" methods, monitor;
"code" 99855;
```

snoleq calculates a solution of a system of n equations in n variables using a poly-algorithm which is fully described in [1]. This algorithm uses DBU, a restrained difference Newton method with conditional updating (which performs triangular decomposition of the jacobian approximation), and GDS, a strict generalized Newton method (which performs singular value decomposition). If scaling is not allowed (default) then DBU is restricted to 40 iteration steps and when it fails to solve the problem then GAS is called (also at most 40 steps).

If scaling is allowed then two calls of DBU are permitted (both restricted to 20 steps) with scaled function and variables. Hence, rescaling may be performed once. If no solution is obtained, then GDS is called with the unscaled function (at most 40 steps). The options of scaling and conditional updating can be controlled by methods. This procedure can also be used to avoid calling of either DBU or GDS. A detailed description of the parameters of snoleg is given in section 3.1.

3.5.3. REDUCEJ

Solving n linear and nonlinear equations in n variables if the analytic jacobian is available.

```
"procedure" reducej(n, p, x, la, lb, fun, jacob, prec, res,
                    methods, monitor);
"value" n, p; "integer" n, p;
"array" x, la, lb, prec, res;
"boolean" "procedure" fun;
"procedure" jacob, methods, monitor;
"code" 99856;
```

reducej solves n linear and nonlinear equations in n variables. The problem is reduced to a p * p nonlinear problem and subsequently solved by calling snoleqj. The reduction method is described in detail in [1]. A full description of the parameters of reducej is given in section 3.1.

3.5.4. REDUCE

Solving n linear and nonlinear equations in n variables if no analytic jacobian is available.

```
"procedure" reduce(n, p, x, la, lb, fun, prec, res, methods,
                   monitor);
"value" n, p; "integer" n, p;
"array" x, la, lb, prec, res;
"boolean" "procedure" fun;
"procedure" methods, monitor;
"code" 99857;
```

reduce solves n linear and nonlinear equations in n variables. The problem is reduced to a p * p nonlinear problem and subsequently solved calling snoleq. The reduction method is described in detail in [1]. A full description of the parameters of reduce is given in section 3.1.

3.6. Procedures and code number reference tables

3.6.1. MACHINE CONSTANTS FROM NUMAL3 (see [2])

```
arreb                30002;
dwarf                30003;
giant                30005;
overflow             30008;
```

3.6.2. MATRIX-VECTOR PROCEDURES FROM NUMAL3 (see [2])

```
colcst               31131;
dupmat               31035;
dupvec               31030;
elmcol               34023;
elmcolvec            34022;
elmrow               34024;
elmvec               34020;
fulmatvec            31500;
fultamvec            31501;
ichcol               34031;
ichrow               34032;
infnrmcol            31063;
infnrmvec            31061;
inimat               31011;
inivec               31010;
matmat               34013;
mattam               34015;
matvec               34011;
maxelmrow            34025;
mulvec               31020;
rotcol               34040;
rowcst               31132;
tammat               34014;
vecvec               34010;
```

3.6.3. MATRIX DECOMPOSITIONS AND LINEAR SYSTEMS FROM NUMAL3 (see [2])

```
gsselm               34231;
homsolsvd            34284;
hshreabid            34260;
pretfmmat            34262;
psttfmmat            34261;
qrisngvaldec         34273;
qrisngvaldecbid      34271;
sol                  34051;
solelm               34061;
```

3.6.4. OTHER PROCEDURES FROM NUMAL3 (see [2])

```
jacobnnf            34437;
random              11015;
setrandom           11014;
zeroin              34150;
zeroinder           34453;
```

3.6.5. PROCEDURES DESCRIBED IN THIS MANUAL

```
abu                 99850;
backscale           99804;
cnddiag             99802;
dbu                 99852;
gas                 99851;
gds                 99853;
monitor1            99901;
monitor2            99902;
nrm                 99801;
oneq                99859;
oneqj               99858;
outmat              90002;
outvec              90001;
reduce              99857;
reducej             99856;
resbis              99807;
scale               99803;
snoleq              99855;
snoleqj             99854;
stopful             99810;
stopspl             99811;
sum                 99800;
svdec               99812;
svsol               99813;
tridec              99805;
trisol              99806.
```

4. EXAMPLE PROGRAMS

```
"BEGIN" "COMMENT" EXAMPLE OF USE OF SNOLEQJ WITH MONITOR1 AND
  TEST FUNCTION I.2 FROM [1], WITH C = 10;

  "BOOLEAN" "PROCEDURE" FUN(N, X, F); "VALUE" N;
  "INTEGER" N; "ARRAY" X, F;
  "BEGIN" "INTEGER" I; "BOOLEAN" INDOMAIN;
    INDOMAIN:= "TRUE";
    "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
    "BEGIN" "IF" ABS(X[I]) > 100 "THEN" INDOMAIN:= "FALSE"
    "END"; "IF" INDOMAIN "THEN"
    "BEGIN" "REAL" AID; AID:= 10;
        "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO" AID:= AID * X[I];
        F[1]:= AID - 1;
        "FOR" I:= 2 "STEP" 1 "UNTIL" N "DO"
        F[I]:= EXP(-X[I-1]) + EXP(-X[I]) - 1.1
    "END";
    FUN:= INDOMAIN
  "END" DEFINITION OF FUNCTION;

  "PROCEDURE" JACOBIAN(N, X, JAC); "VALUE" N;
  "INTEGER" N; "ARRAY" X, JAC;
  "BEGIN" "INTEGER" I, J; "REAL" AID;
    "FOR" J:= 1 "STEP" 1 "UNTIL" N "DO"
    "BEGIN" AID:= 10;
      "FOR" I:= 1 "STEP" 1 "UNTIL" J-1, J+1 "STEP" 1 "UNTIL" N
      "DO" AID:= AID * X[I]; JAC[1,J]:= AID
    "END";
    "FOR" I:= 2 "STEP" 1 "UNTIL" N "DO"
    "BEGIN" "FOR" J:= 1 "STEP" 1 "UNTIL" I-2, I+1 "STEP" 1
      "UNTIL" N "DO" JAC[I,J]:= 0;
      JAC[I,I-1]:= -EXP(-X[I-1]); JAC[I,I]:= -EXP(-X[I])
    "END"
  "END" DEFINITION OF JACOBIAN OF FUNCTION;

  "PROCEDURE" SNOLEQJ(N, X, FUN, JACOB, PREC, RES, METHOD,
  MONITOR);
  "CODE" 99854;
  "PROCEDURE" MONITOR(PH, N, IC, IMAX, DC, FC, JC, X, F, NRMF,
  B, UPD, SCL, ROWS, COLS, LAB, OM, BT, KP, NBI, E, ER, HS);
  "CODE" 99901;

  "INTEGER" I, N; "REAL" AID;
  "ARRAY" X[1:3], PREC[1:7], RES[1:10];
  "REAL" "PROCEDURE" ARREB; "CODE" 30002;

  "PROCEDURE" METHODS(B1, B2, B3, B4);
  "BOOLEAN" B1, B2, B3, B4;
  "COMMENT" WE USE THE DEFAULT VERSION;;
                                                    "COMMENT"
```

```
  #;
  N:= 3;
  "FOR" I:= 1, 2, 3 "DO" PREC[I]:= "-7;
  AID:= ARREB * N; "FOR" I:= 4, 5, 6, 7 "DO" PREC[I]:= AID;
  "FOR" I:= 1 "STEP" 2 "UNTIL" N "DO" X[I]:= 10 ** (-2/3);
  "FOR" I:= 2 "STEP" 2 "UNTIL" N "DO" X[I]:= 1;

  OUTPUT(71,"("*,"("RUN OF SNOLEQJ WITH FUNCTION I.2")",/")");
  SNOLEQJ(N, X, FUN, JACOBIAN, PREC, RES, METHODS, MONITOR);

  OUTPUT(71, "("/,"("AUXILIARY RESULTS")",/,
  "("ERROR NUMBER                  ")", 5ZD, /,
  "("NORM FUNCTION VECTOR          ")", +.5D"+3D, /,
  "("NUMBER OF ITERATIONS          ")", 5ZD, /,
  "("NUMBER OF TRIANG. DECOMP.     ")", 5ZD, /,
  "("NUMBER OF SING. VAL. DECOMP.")", 5ZD, /,
  "("NUMBER OF CALLS FUN           ")", 5ZD, /,
  "("NUMBER OF CALLS JACOB         ")", 5ZD, /,
  "("COND. NUMB. ROW SCALING       ")", +.5D"+3D, /,
  "("COND. NUMB. COL SCALING       ")", +.5D"+3D, /,
  "("COND. NUMB. APPROX. JACOB.    ")", +.5D"+3D, /")",
  RES[1], RES[2], RES[3], RES[4], RES[5], RES[6], RES[7],
  RES[8], RES[9],RES[10])
"END"
```

```
********************
* END OF ITERATION *
********************

ITERATION SUCCESSFUL

APPROX. SOLUTION

    1   +.31825610790992"+000       2   +.98729401781228"+000       3   +.31825610790992"+000

FUNCTION

    1   +.36600"-009       2   +.79439"-011       3   +.79439"-011

AUXILIARY RESULTS
ERROR NUMBER                      0
NORM FUNCTION VECTOR        +.36617"-009
NUMBER OF ITERATIONS              5
NUMBER OF TRIANG. DECOMP.         5
NUMBER OF SING. VAL. DECOMP.      0
NUMBER OF CALLS FUN              15
NUMBER OF CALLS JACOB             0
COND. NUMB. ROW SCALING     +.10000"+001
COND. NUMB. COL SCALING     +.10000"+001
COND. NUMB. APPROX. JACOB.  +.86194"+001
```

```
"BEGIN" "COMMENT" EXAMPLE OF USE OF SNOLEQ WITH MONITOR2 AND
  TEST FUNCTION I.2 FROM [1], WITH C = 10;

  "BOOLEAN" "PROCEDURE" FUN(N, X, F); "VALUE" N;
  "INTEGER" N; "ARRAY" X, F;
  "BEGIN" "INTEGER" I; "BOOLEAN" INDOMAIN;
    INDOMAIN:= "TRUE";
    "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
    "BEGIN" "IF" ABS(X[I]) > 100 "THEN" INDOMAIN:= "FALSE"
    "END"; "IF" INDOMAIN "THEN"
    "BEGIN" "REAL" AID; AID:= 10;
        "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO" AID:= AID * X[I];
        F[1]:= AID - 1;
        "FOR" I:= 2 "STEP" 1 "UNTIL" N "DO"
        F[I]:= EXP(-X[I-1]) + EXP(-X[I]) - 1.1
    "END";
    FUN:= INDOMAIN
  "END" DEFINITION OF FUNCTION;

  "PROCEDURE" SNOLEQ(N, X, FUN, PREC, RES, METHOD, MONITOR);
  "CODE" 99855;
  "PROCEDURE" MONITOR(PH, N, IC, IMAX, DC, FC, JC, X, F, NRMF,
  B, UPD, SCL, ROWS, COLS, LAB, OM, BT, KP, NBI, E, ER, HS);
  "CODE" 99902;

  "INTEGER" I, N; "REAL" AID;
  "ARRAY" X[1:3], PREC[1:7], RES[1:10];
  "REAL" "PROCEDURE" ARREB; "CODE" 30002;

  "PROCEDURE" METHODS(B1, B2, B3, B4);
  "BOOLEAN" B1, B2, B3, B4;
  "COMMENT" WE USE THE DEFAULT VERSION;;

  N:= 3;
  "FOR" I:= 1, 2, 3 "DO" PREC[I]:= "-7;
  PREC[4]:= PREC[5]:= ARREB* N;
  "FOR" I:= 1 "STEP" 2 "UNTIL" N "DO" X[I]:= 10 ** (-2/3);
  "FOR" I:= 2 "STEP" 2 "UNTIL" N "DO" X[I]:= 1;

  OUTPUT(71,"("*,"("RUN OF SNOLEQ WITH FUNCTION I.2")",/")");
  SNOLEQ(N, X, FUN, PREC, RES, METHODS, MONITOR);

  OUTPUT(71, "("/,"("AUXILIARY RESULTS")",/,
  "("ERROR NUMBER                 ")", 5ZD, /,
  "("NORM FUNCTION VECTOR         ")", +.5D"+3D, /,
  "("NUMBER OF ITERATIONS         ")", 5ZD, /,
  "("NUMBER OF TRIANG. DECOMP.    ")", 5ZD, /,
  "("NUMBER OF SING. VAL. DECOMP.")", 5ZD, /,
  "("NUMBER OF CALLS FUN          ")", 5ZD, /,
  "("NUMBER OF CALLS JACOB        ")", 5ZD, /,
  "("COND. NUMB. ROW SCALING      ")", +.5D"+3D, /,
  "("COND. NUMB. COL SCALING      ")", +.5D"+3D, /,
  "("COND. NUMB. APPROX. JACOB.   ")", +.5D"+3D, /")",
  RES[1], RES[2], RES[3], RES[4], RES[5], RES[6], RES[7],
  RES[8], RES[9],RES[10])
"END"
```

```
RUN OF SNOLEQ WITH FUNCTION I.2


START OF ITERATION, NO SCALING PERFORMED


*******************************************************
NEW ITERATION: ITS, DCS, FUS, JCS:   0    0    0    0
*******************************************************

VARIABLES

    1   +.21544"+000        2   +.10000"+001        3   +.21544"+000

FUNCTION

    1   -.53584"+000        2   +.74063"-001        3   +.74063"-001


JACOBIAN

             1              2              3

   1  +.21544"+001  +.46416"+000  +.21544"+001
   2  -.80618"+000  -.36788"+000  +.00000"+000
   3  +.00000"+000  -.36788"+000  -.80618"+000


NORM(F) = +5.4598208991480"-001
LABDA = +1.0000000000000"+000
OMEGA = +1.0000000000000"+000
BETA = +1.0000000000000"+000
KAPPA = +1.0000000000000"+000
NORM JAC INV = +1.0000000000000"+000
ERROR JAC APPR = +0.0000000000000"+000
DIFF STEP JAC APPR = +2.6599601045673"-007


*******************************************************
NEW ITERATION: ITS, DCS, FUS, JCS:   1    1    1    0
*******************************************************

VARIABLES

    1   +.34984"+000        2   +.90680"+000        3   +.34984"+000

FUNCTION

    1   +.10982"+000        2   +.86138"-002        3   +.86138"-002


JACOBIAN

             1              2              3

   1  +.21544"+001  +.46416"+000  +.21544"+001
   2  -.80618"+000  -.36788"+000  +.00000"+000
   3  +.00000"+000  -.36788"+000  -.80618"+000
```

```
NORM(F) = +1.1049479777195"-001
LABDA = +1.0000000000000"+000
OMEGA = +1.0000000000000"+000
BETA = +2.1168601533572"-001
KAPPA = +8.3530889355061"-001
NORM JAC INV = +2.5915643125036"+000
ERROR JAC APPR = +1.2596949883859"-006
DIFF STEP JAC APPR = +2.6599601045673"-007


*********************************************************
NEW ITERATION: ITS, DCS, FUS, JCS:    2    2    5    0
*********************************************************

VARIABLES

    1   +.31754"+000        2   +.98450"+000        3   +.31754"+000

FUNCTION

    1   -.72895"-002        2   +.15597"-002        3   +.15597"-002


JACOBIAN

              1              2              3

   1  +.31724"+001  +.12239"+001  +.31724"+001
   2  -.70480"+000  -.40381"+000  +.00000"+000
   3  +.00000"+000  -.40381"+000  -.70480"+000


NORM(F) = +7.6158881308662"-003
LABDA = +1.0000000000000"+000
OMEGA = +3.8035141239447"+000
BETA = +9.0132947951112"-002
KAPPA = +2.5877634826471"+000
NORM JAC INV = +2.6553756558501"+000
ERROR JAC APPR = +2.1611591985854"-006
DIFF STEP JAC APPR = +3.3179485321782"-007


*********************************************************
NEW ITERATION: ITS, DCS, FUS, JCS:    3    3    9    0
*********************************************************

VARIABLES

    1   +.31826"+000        2   +.98728"+000        3   +.31826"+000

FUNCTION

    1   +.17754"-004        2   +.16246"-005        3   +.16246"-005
```

```
JACOBIAN

            1              2              3

   1  +.31262"+001  +.10083"+001  +.31262"+001
   2  -.72794"+000  -.37362"+000  +.00000"+000
   3  +.00000"+000  -.37362"+000  -.72794"+000


NORM(F) = +1.7902300933794"-005
LABDA = +1.0000000000000"+000
OMEGA = +7.4886896741979"-001
BETA = +2.9549488550016"-003
KAPPA = +1.2129688211492"+000
NORM JAC INV = +2.7511282823138"+000
ERROR JAC APPR = +1.0330143349028"-006
DIFF STEP JAC APPR = +1.6184290990953"-007


*****************************************************
NEW ITERATION: ITS, DCS, FUS, JCS:   4    4   10    0
*****************************************************

VARIABLES

    1   +.31826"+000        2   +.98729"+000        3   +.31826"+000

FUNCTION

    1   -.16516"-006        2   +.62564"-008        3   +.62564"-008

JACOBIAN

            1              2              3

   1  +.31277"+001  +.10140"+001  +.31277"+001
   2  -.72780"+000  -.37311"+000  +.13512"-003
   3  +.13512"-003  -.37311"+000  -.72780"+000


NORM(F) = +1.6539917125726"-007
LABDA = +1.0000000000000"+000
OMEGA = +1.8567248387700"+000
BETA = +1.6212406488581"-005
KAPPA = +2.8324610407311"+000
NORM JAC INV = +2.7631561167311"+000
ERROR JAC APPR = +5.5456096732026"-003
DIFF STEP JAC APPR = +1.6184290990953"-007


*****************************************************
NEW ITERATION: ITS, DCS, FUS, JCS:   5    5   11    0
*****************************************************

VARIABLES

    1   +.31826"+000        2   +.98729"+000        3   +.31826"+000

FUNCTION

    1   +.36600"-009        2   +.79439"-011        3   +.79439"-011
```

JACOBIAN

	1	2	3
1	+.31310"+001	+.10049"+001	+.31310"+001
2	-.72792"+000	-.37276"+000	+.11460"-004
3	+.11460"-004	-.37276"+000	-.72792"+000

NORM(F) = +3.6617294046149"-010
LABDA = +1.0000000000000"+000
OMEGA = +2.6730152337970"+002
BETA = +7.0258100194025"-008
KAPPA = +1.3299688073252"+000
NORM JAC INV = +2.7529466816224"+000
ERROR JAC APPR = +5.6829444176851"-003
DIFF STEP JAC APPR = +1.6184290990953"-007

RUN OF SNOLEQJ WITH FUNCTION I.2

START OF ITERATION, NO SCALING PERFORMED

ITS	DCS	FUS	JCS	NORM(F)	LABDA	OMEGA	BETA	KAPPA	NRMJACINV	ERRJAC	DIFFSTEP
0	0	0	0	.546"+000	.100"+001	.100"+001	.100"+001	.100"+001	.100"+001	.000"+000	.000"+000
1	1	1	0	.110"+000	.100"+001	.100"+001	.212"+000	.835"+000	.259"+001	.208"-011	.000"+000
2	2	2	1	.762"-002	.100"+001	.380"+001	.901"-001	.259"+001	.266"+001	.311"-011	.000"+000
3	3	3	2	.179"-004	.100"+001	.749"+000	.295"-002	.121"+001	.275"+001	.318"-011	.000"+000
4	4	4	2	.165"-006	.100"+001	.186"+001	.162"-004	.283"+001	.276"+001	.554"-002	.000"+000
5	5	5	2	.366"-009	.100"+001	.267"+003	.703"-007	.133"+001	.275"+001	.568"-002	.000"+000

END OF ITERATION
ITERATION SUCCESSFUL

APPROX. SOLUTION

1 +.31825610790993"+000 2 +.98729401781225"+000 3 +.31825610790993"+000

FUNCTION

1 +.36604"-009 2 +.79510"-011 3 +.79510"-011

AUXILIARY RESULTS

ERROR NUMBER	0
NORM FUNCTION VECTOR	+.36621"-009
NUMBER OF ITERATIONS	5
NUMBER OF TRIANG. DECOMP.	5
NUMBER OF SING. VAL. DECOMP.	0
NUMBER OF CALLS FUN	6
NUMBER OF CALLS JACOB	3
COND. NUMB. ROW SCALING	+.10000"+001
COND. NUMB. COL SCALING	+.10000"+001
COND. NUMB. APPROX. JACOB.	+.86194"+001

```
"BEGIN" "COMMENT" EXAMPLE OF USE OF REDUCEJ WITH MONITOR1;

  "BOOLEAN" "PROCEDURE" FUN(N, X, F); "VALUE" N;
  "INTEGER" N; "ARRAY" X, F;
  "BEGIN" "INTEGER" I; "BOOLEAN" INDOMAIN;
    INDOMAIN:= "TRUE";
    "FOR" I:= 1, 2, 3, 4 "DO"
    "BEGIN" "IF" ABS(X[I]) > 100 "THEN" INDOMAIN:= "FALSE"
    "END"; "IF" INDOMAIN "THEN"
    "BEGIN" F[1]:= -(X[2] - X[1] ** 2) * X[1] * 40
        - (1 - X[1]) * 2;
        F[2]:= (X[2] - X[1] ** 2) * 20 + X[3] + X[4] - 2
    "END";
    FUN:= INDOMAIN
  "END" DEFINITION OF FUNCTION;

  "PROCEDURE" JACOBIAN(N, X, JAC); "VALUE" N;
  "INTEGER" N; "ARRAY" X, JAC;
  "BEGIN" JAC[1,1]:= X[1] ** 2 * 120 - X[2] * 40 + 2;
    JAC[1,2]:= -X[1] * 40; JAC[1,3]:= JAC[1,4]:= 0;
    JAC[2,1]:= -X[1] * 40; JAC[2,2]:= 20;
    JAC[2,3]:= JAC[2,4]:= 1
  "END" DEFINITION OF JACOBIAN;

  "PROCEDURE" REDUCEJ(N, P, X, LA, LB, FUN, JACOB, PREC, RES,
  METHODS, MONITOR);
  "CODE" 99856;
  "PROCEDURE" MONITOR(PH, N, IC, IMAX, DC, FC, JC, X, F, NRMF,
  B, UPD, SCL, ROWS, COLS, LAB, OM, BT, KP, NBI, E, ER, HS);
  "CODE" 99901;

  "INTEGER" I, N, P; "REAL" AID;
  "ARRAY" X[1:4], LB[1:2], LA[1:2,1:4], PREC[1:7], RES[1:10];
  "REAL" "PROCEDURE" ARREB; "CODE" 30002;

  "PROCEDURE" METHODS(B1, B2, B3, B4);
  "BOOLEAN" B1, B2, B3, B4;
  "COMMENT" WE USE THE DEFAULT VERSION;;

  N:= 4; P:= 2;
  "FOR" I:= 1, 2, 3 "DO" PREC[I]:= "-7;
  AID:= ARREB * N; "FOR" I:= 4, 5, 6, 7 "DO" PREC[I]:= AID;
  LA[1,1]:= 1; LA[1,2]:= -1; LA[1,3]:= 2; LA[1,4]:= 0;
  LA[2,1]:= 2; LA[2,2]:= 2; LA[2,3]:= 1; LA[2,4]:= -2;
  LB[1]:= 2; LB[2]:= 3;
  X[1]:= X[3]:= -1.2; X[2]:= X[4]:= 1;

  OUTPUT(71,"("*,"("RUN OF REDUCEJ ")",/")");
  REDUCEJ(N, P, X, LA, LB, FUN, JACOBIAN, PREC, RES, METHODS,
  MONITOR);

                                                  "COMMENT"
```

```
  #;
  OUTPUT(71, "("/,"("AUXILIARY RESULTS")",/,
  "("ERROR NUMBER                  ")", 5ZD, /,
  "("NORM FUNCTION VECTOR          ")", +.5D"+3D, /,
  "("NUMBER OF ITERATIONS          ")", 5ZD, /,
  "("NUMBER OF TRIANG. DECOMP.     ")", 5ZD, /,
  "("NUMBER OF SING. VAL. DECOMP.")", 5ZD, /,
  "("NUMBER OF CALLS FUN           ")", 5ZD, /,
  "("NUMBER OF CALLS JACOB         ")", 5ZD, /,
  "("COND. NUMB. ROW SCALING       ")", +.5D"+3D, /,
  "("COND. NUMB. COL SCALING       ")", +.5D"+3D, /,
  "("COND. NUMB. APPROX. JACOB.    ")", +.5D"+3D, /")",
  RES[1], RES[2], RES[3], RES[4], RES[5], RES[6], RES[7],
  RES[8], RES[9],RES[10])
"END"
```

```
RUN OF REDUCEJ


START OF ITERATION, NO SCALING PERFORMED

 ITS   DCS   FUS   JCS    NORM(F)     LABDA       OMEGA       BETA        KAPPA      NRMJACINV    ERRJAC     DIFFSTEP
   0     0     0     0  .141"+002  .100"+001  .100"+001  .100"+001  .100"+001  .100"+001  .000"+000  .000"+000
   1     1     1     0  .216"+001  .100"+001  .100"+001  .101"+001  .139"+001  .635"-001  .317"-012  .000"+000
   2     2     6     1  .167"+001  .625"-001  .155"+002  .769"+001  .526"+002  .813"+000  .309"-011  .000"+000
   3     3     9     2  .143"+001  .250"+000  .631"+001  .117"+001  .199"+002  .769"-001  .564"-012  .000"+000
   4     4    11     3  .132"+001  .500"+000  .297"+000  .944"+000  .248"+002  .134"+000  .130"-011  .000"+000
   5     5    12     4  .736"+000  .100"+001  .341"+000  .546"+000  .220"+002  .202"+000  .276"-011  .000"+000
   6     6    13     5  .375"-002  .100"+001  .986"+000  .766"-001  .732"+001  .270"+000  .486"-011  .000"+000
   7     7    14     6  .349"-004  .100"+001  .119"+001  .350"-002  .667"+002  .291"+000  .533"-011  .000"+000
   8     8    15     6  .495"-007  .100"+001  .340"-001  .416"-006  .852"+000  .291"+000  .104"-001  .000"+000
END OF ITERATION
ITERATION SUCCESSFUL

APPROX. SOLUTION

    1   -.45055854144165"+000       2   +.16397831730829"+001

FUNCTION

    1   -.48342"-007       2   +.10642"-007
END OF ITERATION
ITERATION SUCCESSFUL

APPROX. SOLUTION

    1   +.99999999676613"+000       2   +.99999999457912"+000       3   +.99999999890632"+000       4   +.99999999079834"+000

FUNCTION

    1   -.48342"-007       2   +.10642"-007       3   -.32685"-012       4   +.11369"-012

AUXILIARY RESULTS
ERROR NUMBER                    0
NORM FUNCTION VECTOR        +.49499"-007
NUMBER OF ITERATIONS            8
NUMBER OF TRIANG. DECOMP.       8
NUMBER OF SING. VAL. DECOMP.    0
NUMBER OF CALLS FUN            16
NUMBER OF CALLS JACOB           7
COND. NUMB. ROW SCALING     +.10000"+001
COND. NUMB. COL SCALING     +.10000"+001
COND. NUMB. APPROX. JACOB.  +.20798"+002
```

```
"BEGIN" "COMMENT" EXAMPLE OF USE OF REDUCE WITH MONITOR1;

  "BOOLEAN" "PROCEDURE" FUN(N, X, F); "VALUE" N;
  "INTEGER" N; "ARRAY" X, F;
  "BEGIN" "INTEGER" I; "BOOLEAN" INDOMAIN;
    INDOMAIN:= "TRUE";
    "FOR" I:= 1, 2, 3, 4 "DO"
    "BEGIN" "IF" ABS(X[I]) > 100 "THEN" INDOMAIN:= "FALSE"
    "END"; "IF" INDOMAIN "THEN"
    "BEGIN" F[1]:= -(X[2] - X[1] ** 2) * X[1] * 40
        - (1 - X[1]) * 2;
        F[2]:= (X[2] - X[1] ** 2) * 20 + X[3] + X[4] - 2
    "END";
    FUN:= INDOMAIN
  "END" DEFINITION OF FUNCTION;

  "PROCEDURE" REDUCE(N, P, X, LA, LB, FUN, PREC, RES, METHODS,
  MONITOR);
  "CODE" 99857;
  "PROCEDURE" MONITOR(PH, N, IC, IMAX, DC, FC, JC, X, F, NRMF,
  B, UPD, SCL, ROWS, COLS, LAB, OM, BT, KP, NBI, E, ER, HS);
  "CODE" 99901;

  "INTEGER" I, N, P; "REAL" AID;
  "ARRAY" X[1:4], LB[1:2], LA[1:2,1:4], PREC[1:7], RES[1:10];
  "REAL" "PROCEDURE" ARREB; "CODE" 30002;

  "PROCEDURE" METHODS(B1, B2, B3, B4);
  "BOOLEAN" B1, B2, B3, B4;
  "COMMENT" WE USE THE DEFAULT VERSION;;

  N:= 4; P:= 2;
  "FOR" I:= 1, 2, 3 "DO" PREC[I]:= "-7;
  AID:= ARREB * N; "FOR" I:= 4, 5 "DO" PREC[I]:= AID;
  LA[1,1]:= 1; LA[1,2]:= -1; LA[1,3]:= 2; LA[1,4]:= 0;
  LA[2,1]:= 2; LA[2,2]:= 2; LA[2,3]:= 1; LA[2,4]:= -2;
  LB[1]:= 2; LB[2]:= 3;
  X[1]:= X[3]:= -1.2; X[2]:= X[4]:= 1;

  OUTPUT(71,"("*,"("RUN OF REDUCE ")",/")");
  REDUCE(N, P, X, LA, LB, FUN, PREC, RES, METHODS, MONITOR);

  OUTPUT(71, "("/,"("AUXILIARY RESULTS")",/,
  "("ERROR NUMBER                 ")", 5ZD, /,
  "("NORM FUNCTION VECTOR         ")", +.5D"+3D, /,
  "("NUMBER OF ITERATIONS         ")", 5ZD, /,
  "("NUMBER OF TRIANG. DECOMP.    ")", 5ZD, /,
  "("NUMBER OF SING. VAL. DECOMP.")", 5ZD, /,
  "("NUMBER OF CALLS FUN          ")", 5ZD, /,
  "("NUMBER OF CALLS JACOB        ")", 5ZD, /,
  "("COND. NUMB. ROW SCALING      ")", +.5D"+3D, /,
  "("COND. NUMB. COL SCALING      ")", +.5D"+3D, /,
  "("COND. NUMB. APPROX. JACOB.   ")", +.5D"+3D, /")",
  RES[1], RES[2], RES[3], RES[4], RES[5], RES[6], RES[7],
  RES[8], RES[9],RES[10])
"END"
```

```
RUN OF REDUCE


START OF ITERATION, NO SCALING PERFORMED

 ITS  DCS  FUS  JCS    NORM(F)     LABDA       OMEGA       BETA        KAPPA      NRMJACINV    ERRJAC     DIFFSTEP
   0    0    0    0  .141"+002  .100"+001  .100"+001  .100"+001  .100"+001  .100"+001  .000"+000  .943"-006
   1    1    1    0  .216"+001  .100"+001  .100"+001  .101"+001  .139"+001  .635"-001  .116"-005  .943"-006
   2    2    8    0  .167"+001  .625"-001  .155"+002  .769"+001  .526"+002  .813"+000  .336"-005  .138"-006
   3    3   13    0  .143"+001  .250"+000  .631"+001  .117"+001  .199"+002  .769"-001  .757"-006  .113"-006
   4    4   17    0  .132"+001  .500"+000  .297"+000  .944"+000  .248"+002  .134"+000  .519"-006  .475"-007
   5    5   20    0  .736"+000  .100"+001  .341"+000  .546"+000  .220"+002  .202"+000  .219"-006  .225"-006
   6    6   23    0  .375"-002  .100"+001  .986"+000  .766"-001  .732"+001  .270"+000  .397"-006  .188"-006
   7    7   26    0  .349"-004  .100"+001  .119"+001  .350"-002  .667"+002  .291"+000  .306"-006  .905"-007
   8    8   27    0  .495"-007  .100"+001  .340"-001  .416"-006  .852"+000  .291"+000  .104"-001  .905"-007
END OF ITERATION
ITERATION SUCCESSFUL

APPROX. SOLUTION

    1   -.45055854144015"+000        2   +.16397831730801"+001

FUNCTION

    1   -.48384"-007        2   +.10659"-007
END OF ITERATION
ITERATION SUCCESSFUL

APPROX. SOLUTION

    1   +.99999999676496"+000        2   +.99999999457778"+000        3   +.99999999890624"+000        4   +.99999999079578"+000

FUNCTION

    1   -.48384"-007        2   +.10659"-007        3   -.32685"-012        4   +.14211"-012

AUXILIARY RESULTS
ERROR NUMBER                      0
NORM FUNCTION VECTOR         +.49544"-007
NUMBER OF ITERATIONS              8
NUMBER OF TRIANG. DECOMP.         8
NUMBER OF SING. VAL. DECOMP.      0
NUMBER OF CALLS FUN              30
NUMBER OF CALLS JACOB             0
COND. NUMB. ROW SCALING      +.10000"+001
COND. NUMB. COL SCALING      +.10000"+001
COND. NUMB. APPROX. JACOB.   +.20798"+002
```

REFERENCES

[1] BUS J.C.P., Numerical solution of systems of nonlinear equations, Mathematical Centre Tracts 122 (1980), Mathematisch Centrum, Amsterdam.
[2] HEMKER, P.W. et al.,NUMAL, A library of numerical procedures in ALGOL60, Third revision (1979), Mathematisch Centrum, Amsterdam.

APPENDIX: SOURCE TEXTS

```
"CODE" 90001;
"PROCEDURE" OUTVEC(CH, X, L, U, M, E, S); "VALUE" CH, L, U, M, E;
"INTEGER" CH, L, U, M, E; "ARRAY" X; "STRING" S;
"BEGIN" "INTEGER" LV, LP, PP, BP, LU, NUL, NL, LLC, K, LL, I, J;
  "PROCEDURE" LAYOUT;
  "IF" I = 0 "THEN" FORMAT("("XS,//")",CHLENGTH(S)) "ELSE"
  FORMAT("("X(3ZD3B,+.XD"+XD3B),/")",NUL,M, E);
  "PROCEDURE" LIST(ITEM); "PROCEDURE" ITEM;
  "IF" I = 0 "THEN" ITEM(S) "ELSE"
  "FOR" J:= 1 "STEP" 1 "UNTIL" NUL "DO"
  "BEGIN" K:= (J - 1) * NL + I + L - 1; ITEM(K); ITEM(X[K]) "END";
  SYSPARAM(CH, 7, LP); SYSPARAM(CH, 3, PP); SYSPARAM(CH, 5, BP);
  LV:= U - L + 1; LU:= M + E + 14; NUL:= (BP - 1) // LU;
  NL:= LV // NUL; K:= LV - NL * NUL; "IF" K = 0 "THEN" LLC:= NL
  "ELSE"
  "BEGIN" NL:= NL + 1; LLC:= NL + K - NUL;
  LABEL: "IF" LLC < 0 "THEN"
    "BEGIN" LLC:= LLC + NL; NUL:= NUL - 1; "GOTO" LABEL "END"
  "END";
  SYSPARAM(CH, 1, K);
  "IF" LP - PP < NL + 3 "AND" (PP ^= 0 "OR" K ^= 0) "THEN"
  OUTPUT(CH, "("*")") "ELSE" OUTPUT(CH, "("/")");
  "FOR" I:= 0 "STEP" 1 "UNTIL" NL "DO"
  "BEGIN" OUTLIST(CH, LAYOUT, LIST);
    "IF" I = LLC "THEN" NUL:= NUL - 1
  "END"
"END" OUTVEC;
        "EOP"


"CODE" 90002;
"PROCEDURE" OUTMAT(CH,A,LR,UR,LC,UC,M,E,S);
"VALUE" CH,LR,UR,LC,UC,M,E; "INTEGER" CH,LR,UR,LC,UC,M,E;
"ARRAY" A; "STRING" S;
"BEGIN" "INTEGER" L,NUMR,P,V,W,UP;
  "PROCEDURE" LAYOUTSTRING; FORMAT("("XS,//")",CHLENGTH(S));
  "PROCEDURE" STRING(ITEM); "PROCEDURE" ITEM; ITEM(S);
  "PROCEDURE" LAYOUTBLOCK;
  FORMAT("("5B,X(XBZZDXB),//,X(BZZDB,X(B+.XD"+XDB),/),//")",
  L,V,W,NUMR,L,M,E);

  "PROCEDURE" MATRIXBLOCK(ITEM); "PROCEDURE" ITEM;
  "BEGIN" "INTEGER" I,J;
    "FOR" J:=P "STEP" 1 "UNTIL" UP "DO" ITEM(J);
    "FOR" I:=LR "STEP" 1 "UNTIL" UR "DO"
    "BEGIN" ITEM(I); "FOR" J:=P "STEP" 1 "UNTIL" UP "DO"
      ITEM(A(/I,J/))
    "END"
  "END" MATRIXBLOCK;

  SYSPARAM(CH,3,P); SYSPARAM(CH,7,W);
  NUMR:=UR-LR+1; "IF" P "NOT EQUAL" 0 "THEN"
  "BEGIN" "IF" P+NUMR + 8 > W "THEN" OUTPUT(CH,"("*")")
    "ELSE" OUTPUT(CH,"("//")")
  "END";
  OUTLIST(CH,LAYOUTSTRING,STRING);
  W:=M+E+6; SYSPARAM(CH,5,L); L:=(L-6) "/" W;                    "COMMENT"
```

```
  #;
  W:=W-3; V:=W "/" 2; W:=W-V;
  "FOR" P:=LC "STEP" L "UNTIL" UC "DO"
  "BEGIN" UP:=P + L - 1; "IF" UP > UC "THEN"
    "BEGIN" UP:=UC; L:=UC - P + 1 "END";
    OUTLIST(CH,LAYOUTBLOCK,MATRIXBLOCK)
  "END"
"END" OUTMAT;
        "EOP"

"CODE" 99800;
"REAL" "PROCEDURE" SUM(L, U, I, XI);
"VALUE" L, U; "INTEGER" L, U, I; "REAL" XI;
"BEGIN" "REAL" S; S:= 0;
  "FOR" I:= L "STEP" 1 "UNTIL" U "DO" S:= S + XI; SUM:= S
"END" SUM;
        "EOP"


"CODE" 99801;
"REAL" "PROCEDURE" NRM(N, X); "VALUE" N, X; "INTEGER" N;
"ARRAY" X;
"BEGIN" "REAL" "PROCEDURE" DWARF; "CODE" 30003;
  "REAL" "PROCEDURE" VECVEC(L, U, S, A, B); "CODE" 34010;
  "REAL" "PROCEDURE" INFNRMVEC(L, U, K, A); "CODE" 31061;
  "PROCEDURE" MULVEC(L, U, S, A, B, X); "CODE" 31020;
  "INTEGER" I; "REAL" S1;
  S1:= INFNRMVEC(1, N, I, X); "IF" S1 <= DWARF "THEN"
  NRM:= 0 "ELSE"
  "BEGIN" S1:= 2 ** ENTIER(LN(S1) / LN(2));
    MULVEC(1, N, 0, X, X, 1 / S1);
    NRM:= SQRT(VECVEC(1, N, 0, X, X)) * S1
  "END"
"END" NRM;
        "EOP"


"CODE" 99802;
"REAL" "PROCEDURE" CNDDIAG(N, D);
"VALUE" N; "INTEGER" N; "ARRAY" D;
"BEGIN" "INTEGER" I; "REAL" AID, MIN, MAX;
  "REAL" "PROCEDURE" GIANT; "CODE" 30004;
  "BOOLEAN" "PROCEDURE" OVERFLOW(X); "CODE" 30008;
  MIN:= MAX:= ABS(D[1]); "FOR" I:= 2 "STEP" 1 "UNTIL" N "DO"
  "BEGIN" AID:= ABS(D[I]);
    "IF" AID > MAX "THEN" MAX:= AID "ELSE"
    "IF" AID < MIN "THEN" MIN:= AID
  "END"; AID:= MAX / MIN;
  CNDDIAG:= "IF" OVERFLOW(AID) "THEN" GIANT "ELSE" AID
"END" CNDDIAG;
        "EOP"
```

```
"CODE" 99803;
"INTEGER" "PROCEDURE" SCALE(N, B, X, F, ROWS, COLS);
"VALUE" N; "INTEGER" N; "ARRAY" B, X, F, ROWS, COLS;
"BEGIN" "INTEGER" I, S; "BOOLEAN" SR, SC;
  "REAL" LN2, MAX, MIN, AID, DWRF;
  "REAL" "PROCEDURE" DWARF; "CODE" 30003;
  "PROCEDURE" INIVEC(L, U, A, X); "CODE" 31010;
  "REAL" "PROCEDURE" INFNRMROW(L, U, I, J, A); "CODE" 31062;
  "REAL" "PROCEDURE" INFNRMCOL(L, U, I, J, A); "CODE" 31063;
  "PROCEDURE" COLCST(L, U, I, A, X); "CODE" 31131;
  "PROCEDURE" ROWCST(L, U, I, A, X); "CODE" 31132;
  DWRF:= DWARF; SR:= SC:= "FALSE";
  LN2:= LN(2); MAX:= N * 4; MIN:= 1 / MAX;
  INIVEC(1, N, ROWS, 1); INIVEC(1, N, COLS, 1);
  "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
  "BEGIN" "INTEGER"K; "REAL" RNORM;
    RNORM:= INFNRMROW(1, N, I, K, B);
    "IF" RNORM < DWRF "THEN"
    "BEGIN" S:= -1; "GOTO" END "END" "ELSE"
    "IF" RNORM > MAX "OR" RNORM < MIN "THEN"
    "BEGIN" SR:= "TRUE"; ROWS[I]:= 2 ** ENTIER(-LN(RNORM)/LN2)
    "END"
  "END";
  S:= 0; "IF" SR "THEN"
  "BEGIN" S:= 1; "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
    "BEGIN" AID:= ROWS[I]; "IF" AID ^= 1 "THEN"
      "BEGIN" ROWCST(1, N, I, B, AID); F[I]:= F[I] * AID "END"
    "END"
  "END" FUNCTION AND ROW SCALING;
  "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
  "BEGIN" "INTEGER" K; "REAL" CNORM;
    CNORM:= INFNRMCOL(1, N, I, K, B);
    "IF" CNORM < DWRF "THEN"
    "BEGIN" S:= -1; "GOTO" END "END" "ELSE"
    "IF" CNORM > MAX "OR" CNORM < MIN "THEN"
    "BEGIN" SC:= "TRUE"; COLS[I]:= 2 ** ENTIER(-LN(CNORM)/LN2)
    "END"
  "END";
  "IF" SC "THEN"
  "BEGIN" S:= "IF" SR "THEN" 3 "ELSE" 2;
    "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
    "BEGIN" AID:= COLS[I]; "IF" AID ^= 1 "THEN"
      "BEGIN" COLCST(1, N, I, B, AID); X[I]:= X[I] / AID "END"
    "END"
  "END" VARIABLES AND COL SCALING;
  END: SCALE:= S;
"END" SCALE;
        "EOP"


"CODE" 99804;
"PROCEDURE" BACKSCALE(N, B, X, F, SCL, ROWS, COLS);
"VALUE" N, SCL; "INTEGER" N, SCL; "ARRAY" B, X, F, ROWS, COLS;
"BEGIN" "INTEGER" I, J; "REAL" AID;
  "PROCEDURE" COLCST(L, U, I, A, X); "CODE" 31131;
  "PROCEDURE" ROWCST(L, U, I, A, X); "CODE" 31132;
  "IF" SCL = 1 "OR" SCL = 3 "THEN"
  "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
  "BEGIN" AID:= ROWS[I]; "IF" AID ^= 1 "THEN"
    "BEGIN"  ROWCST(1, N, I, B, 1 / AID); F[I]:= F[I] / AID "END"
```

```
  "END"; "IF" SCL > 1 "THEN"
  "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
  "BEGIN" AID:= COLS[I]; "IF" AID ^= 1 "THEN"
    "BEGIN"  COLCST(1, N, I, B, 1 / AID); X[I]:= X[I] * AID "END"
  "END"
"END" BACKSCALE;
        "EOP"


"CODE" 99805;
"PROCEDURE" TRIDEC(N, A, ALR, P1, P2, AUX, ERREX);
"VALUE" N; "INTEGER" N;
"ARRAY" A, ALR, AUX; "INTEGER" "ARRAY" P1, P2; "PROCEDURE" ERREX;
"BEGIN" "PROCEDURE" GSSELM(A, N, AUX, RI, CI); "CODE" 34231;
  "PROCEDURE" DUPMAT(LR, UR, LC, UC, A, B); "CODE" 31035;
  DUPMAT(1, N, 1, N, ALR, A);
  GSSELM(ALR, N, AUX, P1, P2);
  "IF" AUX[3] < N "THEN" ERREX(5)
"END" TRIDEC;
        "EOP"


"CODE" 99806;
"PROCEDURE" TRISOL(N, LR, P1, P2, RHS, SOL);
"VALUE" N; "INTEGER" N;
"ARRAY" LR, RHS, SOL;
"INTEGER" "ARRAY" P1, P2;
"BEGIN" "INTEGER" I;
  "PROCEDURE" SOLELM(A, N, RI, CI, B); "CODE" 34061;
  "PROCEDURE" DUPVEC(L, U, S, A, B); "CODE" 31030;
  DUPVEC(1, N, 0, SOL, RHS);
  SOLELM(LR, N, P1, P2, SOL)
"END" TRISOL;
        "EOP"


"CODE" 99812;
"PROCEDURE" SVDEC(N, A, VAL, V, EM, ERREX); "VALUE" N; "INTEGER" N;
"ARRAY" A, VAL, V, EM; "PROCEDURE" ERREX;
 "BEGIN" "INTEGER" "PROCEDURE" QRISNGVALDEC(A, N, M, VAL, V, EM);
   "CODE" 34273;
   "PROCEDURE" HOMSOLSVD(A, VAL, V, N, M); "CODE" 34284;
   "IF" QRISNGVALDEC(A, N, N, VAL, V, EM) ^= 0 "THEN" ERREX(6);
   HOMSOLSVD(A, VAL, V, N, N)
"END" SVDEC;
        "EOP"


"CODE" 99813;
"PROCEDURE" SVSOL(N, U, VAL, V, R, RHS, SOL); "VALUE" N, R;
"INTEGER" N, R; "ARRAY" U, VAL, V, RHS, SOL;
"BEGIN" "INTEGER" I;
  "PROCEDURE" FULMATVEC(LR, UR, LC, UC, A, B, C); "CODE" 31500;
  "PROCEDURE" FULTAMVEC(LR, UR, LC, UC, A, B, C); "CODE" 31501;
  FULTAMVEC(1, N, 1, R, U, RHS, SOL);
  "FOR" I:= 1 "STEP" 1 "UNTIL" R "DO"
  SOL[I]:= SOL[I] / VAL[I];
  FULMATVEC(1, N, 1, R, V, SOL, SOL)
"END" SVSOL;
        "EOP"
```

```
"CODE" 99809;
"PROCEDURE" RESBIS(N, X, NRMX, DX, NRMDX, LABDA, F, LEVEL,
LFUN, ER, EPF, CNT); "VALUE" N; "INTEGER" N, ER, CNT;
"REAL" NRMX, NRMDX, LABDA, LEVEL, EPF;
"ARRAY" X, DX, F; "REAL" "PROCEDURE" LFUN;
"BEGIN" "BOOLEAN" BIS;
  "REAL" LEVEL1, RGIANT, MIN; "ARRAY" X1, F1[1:N];
  "PROCEDURE" MULVEC(L, U, S, A, B, X); "CODE" 31020;
  "PROCEDURE" DUPVEC(L, U, S, A, B); "CODE" 31030;
  "REAL" "PROCEDURE" ARREB; "CODE" 30002;
  "REAL" "PROCEDURE" GIANT; "CODE" 30004;
  "PROCEDURE" ELMVEC(L, U, S, A, B, X); "CODE" 34020;
  CNT:= 0; LEVEL1:= LEVEL * 2; LABDA:= 2; ER:= 0;
  RGIANT:= GIANT; MIN:= ARREB * NRMX * 2 / NRMDX;
L: "IF" ABS(LEVEL - LEVEL1) < EPF "AND" LEVEL ^=  RGIANT
  "THEN"
  "BEGIN" "IF" ER = 2 "THEN" BIS:= "FALSE" "ELSE"
    "BEGIN" ER:= 2; BIS:= "TRUE" "END"
  "END" "ELSE"
  "BEGIN" ER:= 0; BIS:= LEVEL1 >= LEVEL "END";
  "IF" LABDA <= MIN "THEN" ER:= 1 "ELSE" "IF" BIS "THEN"
  "BEGIN" LABDA:= LABDA * 0.5;
    DUPVEC(1, N, 0, X1, X); ELMVEC(1, N, 0, X1, DX, -LABDA);
    LEVEL1:= LFUN(X1, F1);
    CNT:= CNT + 1; "GOTO" L
  "END";
  DUPVEC(1, N, 0, X, X1); DUPVEC(1, N, 0, F, F1);
  MULVEC(1, N, 0, DX, DX, -LABDA); NRMDX:= NRMDX * LABDA;
  LEVEL:= LEVEL1
"END" RESBIS;
        "EOP"


"CODE" 99810;
"BOOLEAN" "PROCEDURE" STOPFUL(NRMDX, DELX, NRMF, DELF, EPF,
EPAF, FIX, KAPPA, E, OM, BETA, LABDA, ER, IT, ITMAX, ERREX);
"VALUE" DELX, DELF, EPAF, FIX;
"INTEGER" IT, ITMAX, ER; "BOOLEAN" FIX;
"REAL" NRMDX, DELX, NRMF, DELF, EPF, EPAF, E, KAPPA, OM,
BETA, LABDA;
"PROCEDURE" ERREX;
"BEGIN" "INTEGER" I; "BOOLEAN" BL;
  "REAL" "PROCEDURE" ARREB; "CODE" 30002;
  "IF" NRMF < EPAF "THEN" BL:= "TRUE" "ELSE"
  "IF" E >= 1 - ARREB "THEN" ERREX(8) "ELSE"
  "BEGIN" "REAL" KSI1, KSI2, ALFA2, A1, K2, TAU2I;
    KSI1:= (1 + E) / (1 - E);
    KSI2:= (1 - (E + 2) * E) / (1 - E);
    ALFA2:= OM * BETA * 2; A1:= ALFA2 * KSI1; K2:= KSI2 * KSI2;
    "IF" E < 0.4142 "AND" LABDA = 1 "AND"
       ("IF" A1 < K2 "THEN"
      NRMDX <= DELX / (2 / (KSI2 + SQRT(K2 - A1)) - 1)
      "ELSE" "FALSE")
       "AND" NRMF < DELF
    "THEN" BL:= "TRUE" "ELSE" "IF" IT = 1 "THEN"
    BL:= "FALSE" "ELSE"
    "BEGIN" "IF" E * KAPPA >= 0.5 "AND" ^FIX "THEN" ERREX(8);
      TAU2I:= ((1 + KAPPA * 2) * ALFA2) ** 2;
      "IF" TAU2I <= 1 "THEN" TAU2I:= 1;
      "IF" NRMF <= EPF * TAU2I "THEN" ERREX(9); BL:= "FALSE"
    "END"
```

```
  "END";
  "IF" IT  >= ITMAX "AND" ^ BL "THEN" ERREX(4);
  "IF" BL "THEN" ER:= 0;
  "IF" ER ^= 0 "THEN" ERREX(ER);
  STOPFUL:= BL
"END" STOPFUL;
        "EOP"


"CODE" 99811;
"BOOLEAN" "PROCEDURE" STOPSPL(NRMDX, DLX, NRMF, DLF, EPAF, ER, IT,
ITMAX, ERREX); "VALUE" DLX, NRMF, ITMAX;
"INTEGER" ER, IT, ITMAX; "REAL" NRMDX, DLX, NRMF, DLF, EPAF;
"PROCEDURE" ERREX;
  "BEGIN" "BOOLEAN" BL;
  "IF" NRMF = EPAF "THEN" ERREX(0);
  BL:= NRMDX < DLX "AND" NRMF < DLF;
  "IF" IT >= ITMAX "AND" ^BL "THEN" ERREX(4);
  "IF" BL "THEN" ER:= 0;
  "IF" ER ^= 0 "THEN" ERREX(ER);
  STOPSPL:= BL
"END" STOPSPL;
        "EOP"


"CODE" 99901;
"PROCEDURE" MONITOR1(PH, N, IC, IMAX, DC, FC, JC, X, F, NRMF, B,
UPD, SCL, ROWS, COLS, LB, OM, BT, KP, NBI, E, ER, HS);
"VALUE" PH, N, DC, FC, JC, SCL, LB, HS;
"INTEGER" PH, N, IC, IMAX, DC, FC, JC, SCL, ER;
"BOOLEAN" UPD; "REAL" NRMF, LB, OM, BT, KP, NBI, E, HS;
"ARRAY" X, F, B, ROWS, COLS;
"BEGIN" "PROCEDURE" OUTVEC(CH, X, L, U, M, E, S); "CODE" 90001;
  "IF" PH = 0 "THEN"
  "BEGIN" "IF" SCL > 0 "THEN"
    "BEGIN" OUTPUT(71, "("////,
      "("START OF ITERATION, SCALING PERFORMED ")"")");
      OUTVEC(71, ROWS, 1, N, 5, 3, "("ROW SCALING FACS")");
      OUTVEC(71, COLS, 1, N, 5, 3, "("COL SCALING FACS")")
    "END" "ELSE" OUTPUT(71, "("////,
    "("START OF ITERATION, NO SCALING PERFORMED ")",/")");
    OUTPUT(71, "("/,"(" ITS  DCS  FUS  JCS   NORM(F)    LAB")"
    "("DA      OMEGA      BETA       KAPPA     NRMJACINV   ")",
    "("ERRJAC    DIFFSTEP")",/")")
  "END" PRINTING IN INITIALIZING PHASE;
  "IF" PH <= 1 "THEN"
  OUTPUT(71, "("4(3ZDB),8(B.3D"+3DB),/")", IC, DC, FC, JC, NRMF,
  LB, OM, BT, KP, NBI, E, HS) "ELSE" "IF" PH = 2 "THEN"
  "BEGIN" OUTPUT(71, "(""("END OF ITERATION")",/")");
    "IF" ER = 0 "THEN"
    OUTPUT(71, "(""("ITERATION SUCCESSFUL ")",/")") "ELSE"
    OUTPUT(71, "(""("FAILURE OF ITERATION; ERROR IS ")",
    2ZD,///")", ER);
    OUTVEC(71, X, 1, N, 14, 3, "("APPROX. SOLUTION")");
    OUTVEC(71, F, 1, N, 5, 3, "("FUNCTION")")
  "END" "ELSE"
  "BEGIN" "IF" IC = 1 "THEN"
    OUTPUT(71, "("////,"(" REDUCED FUNCTION HAS ORDER 1 ")",//,
    "(" IC    VARIABLE      FUNCTION")",//")");
    OUTPUT(71, "("2ZDB, 2(B+.5D"+3DB),/")", IC, X[1], F[1])
  "END" PRINTING OF RESULTS IN ONEDIMENSIONAL CASE
"END" MONITOR1;
        "EOP"
```

```
"CODE" 99902;
"PROCEDURE" MONITOR2(PH, N, IC, IMAX, DC, FC, JC, X, F, NRMF, B,
UPD, SCL, ROWS, COLS, LB, OM, BT, KP, NBI, E, ER, HS);
"VALUE" PH, N, DC, FC, JC, SCL, LB, HS;
"INTEGER" PH, N, IC, IMAX, DC, FC, JC, SCL, ER;
"BOOLEAN" UPD; "REAL" NRMF, LB, OM, BT, KP, NBI, E, HS;
"ARRAY" X, F, B, ROWS, COLS;
"BEGIN" "PROCEDURE" OUTVEC(CH, X, L, U, M, E, S); "CODE" 90001;
  "PROCEDURE" OUTMAT(CH, A, L, U, I, J, M, E, S); "CODE" 90002;
  "IF" PH = 0 "THEN"
  "BEGIN" "IF" SCL > 0 "THEN"
    "BEGIN" OUTPUT(71, "("////,
      "("START OF ITERATION, SCALING PERFORMED ")"")");
      OUTVEC(71, ROWS, 1, N, 5, 3, "("ROW SCALING FACS")");
      OUTVEC(71, COLS, 1, N, 5, 3, "("COL SCALING FACS")")
    "END" "ELSE" OUTPUT(71, "("////,
    "("START OF ITERATION, NO SCALING PERFORMED ")",/")");
  "END" PRINTING IN INITIALIZING PHASE;
  "IF" PH <= 1 "THEN"
  "BEGIN" OUTPUT(71, "("///,
    "("*****************************************************")",
    /, "("NEW ITERATION: ITS, DCS, FUS, JCS:")", 4(3ZDB),/,
    "("*****************************************************")",
    /")", IC, DC, FC, JC);
    OUTVEC(71, X, 1, N, 5, 3, "("VARIABLES")");
    OUTVEC(71, F, 1, N, 5, 3, "("FUNCTION")");
    OUTMAT(71, B, 1, N, 1, N, 5, 3, "("JACOBIAN")");
    OUTPUT(71, "(""("NORM(F) = ")",N,/,"("LABDA = ")",N,/,
    "("OMEGA = ")",N,/,"("BETA = ")",N,/,"("KAPPA = ")",N,/,
    "("NORM JAC INV = ")",N,/,"("ERROR JAC APPR = ")",N,/,
    "("DIFF STEP JAC APPR = ")",N,/")",
    NRMF, LB, OM, BT, KP, NBI, E, HS)
  "END" "ELSE" "IF" PH = 2 "THEN"
  "BEGIN" OUTPUT(71, "("*, "("********************")",/,
    "("* END OF ITERATION *")",/,
    "("********************")",//")");
    "IF" ER = 0 "THEN"
    OUTPUT(71, "(""("ITERATION SUCCESSFUL ")",/")") "ELSE"
    OUTPUT(71, "(""("FAILURE OF ITERATION; ERROR IS ")",
    2ZD,///")", ER);
    OUTVEC(71, X, 1, N, 14, 3, "("APPROX. SOLUTION")");
    OUTVEC(71, F, 1, N, 5, 3, "("FUNCTION")")
  "END" "ELSE"
  "BEGIN" "IF" IC = 1 "THEN"
    OUTPUT(71, "("////,"(" REDUCED FUNCTION HAS ORDER 1 ")",//,
    "(" IC    VARIABLE      FUNCTION")",//")");
    OUTPUT(71, "("2ZDB, 2(B+.5D"+3DB),/")", IC, X[1], F[1])
  "END" PRINTING OF RESULTS IN ONEDIMENSIONAL CASE
"END" MONITOR2;
        "EOP"
```

```
"CODE" 99850;
"PROCEDURE" ABU(N, X, F, B, FUN, JACOB, PREC, RES, MONITOR);
"VALUE" N; "INTEGER" N; "ARRAY" X, F, B, PREC, RES;
"BOOLEAN" "PROCEDURE" FUN; "PROCEDURE" JACOB, MONITOR;
"BEGIN"
  "INTEGER" I, IT, ITMAX, CNT, CNTF, CNTJ, ER;
  "BOOLEAN" UPDATE, ANL;
  "REAL" SLEVEL, LABDA, ETA, KAPPA, OM, BETA, E, NRMDX, NRMX,
     MACHEPS, EPF, EPRF, EPAF, EPRJ, EPAJ, DLF, DLRX, DLAX,
     RGIANT;
  "ARRAY" DX, XJ, F0, V[1:N], BLR[1:N,1:N], AUX[1:7];
  "INTEGER" "ARRAY" P1, P2[1:N];

  "PROCEDURE" DUPVEC(L, U, S, A, B); "CODE" 31030;
  "PROCEDURE" SETRANDOM(X); "CODE" 11014;
  "REAL" "PROCEDURE" RANDOM; "CODE" 11015;
  "REAL" "PROCEDURE" ARREB; "CODE" 30002;
  "REAL" "PROCEDURE" GIANT; "CODE" 30004;
  "REAL" "PROCEDURE" SUM(L, U, I, XI); "CODE" 99800;
  "REAL" "PROCEDURE" NRM(N, X); "CODE" 99801;
  "REAL" "PROCEDURE" CNDDIAG(N, X); "CODE" 99802;
  "REAL" "PROCEDURE" VECVEC(L, U, S, A, B); "CODE" 34010;
  "PROCEDURE" ELMVEC(L, U, S, A, B, X); "CODE" 34020;
  "PROCEDURE" ELMCOLVEC(L, U, I, A, B, X); "CODE" 34022;
  "PROCEDURE" MULVEC(L, U, S, A, B, X); "CODE" 31020;
  "PROCEDURE" TRIDEC(N, A, LR, P1, P2, AX, EXIT); "CODE" 99805;
  "PROCEDURE" TRISOL(N, LR, P1, P2, RHS, SOL); "CODE" 99806;
  "PROCEDURE" RESBIS(N, X, DX, NRMDX, LABDA, F, LEVEL, LFUN, ER,
  EPF, CNT); "CODE" 99809;
  "BOOLEAN" "PROCEDURE" STOPFUL(NRMDX, DELX, NRMF, DELF, EPF,
  EPAF, UPD, KAPPA, E, OM, BETA, LABDA, ER, IT, ITMAX,
  EXIT); "CODE" 99810;

  "BOOLEAN" "PROCEDURE" CONUPDJAC;
  "IF" IT < 3 "OR" ^UPDATE "OR" E >= 0.1
  "THEN" CONUPDJAC:= "FALSE" "ELSE"
  "BEGIN" "INTEGER" J; "REAL" PU, NRMU;
    "ARRAY" U, DF[1:N];
    DUPVEC(1, N, 0, DF, F); ELMVEC(1, N, 0, DF, F0, -1);
    TRISOL(N, BLR, P1, P2, DF, U);
    PU:= VECVEC(1, N, 0, DX, U); NRMU:= NRM(N, U);
    "IF" E < 1 "THEN"
    E:= (E / (1 - E) + (1 + NRMDX * 1.5 / NRMU) * NRMDX * OM)
    * (1 + E);
    "IF" KAPPA * E < 1 "AND" ABS(PU) > NRMDX * NRMU * MACHEPS
    "AND" E < 0.1 "THEN"
    "BEGIN" ELMVEC(1, N, 0, DF, F0, LABDA);
    "FOR" J:= 1 "STEP" 1 "UNTIL" N "DO"
    ELMCOLVEC(1, N, J, B, DF, U[J] / PU);
    CONUPDJAC:= "TRUE"
    "END" "ELSE" CONUPDJAC:= "FALSE"
  "END" CONUPDJAC;

  "PROCEDURE" CDATALR;
  "BEGIN" "REAL" OM1; "ARRAY" D01, D10, W[1:N];
    "IF" IT > 1 "THEN"
    "BEGIN" TRISOL(N, BLR, P1, P2, F, D01);
     "IF" CONUPDJAC "THEN" ANL:= "FALSE" "ELSE"
     "BEGIN" CNTJ:= CNTJ + 1; ANL:= "TRUE"; JACOB(N, X, B) "END"
    "END"
```

```
    ; TRIDEC(N, B, BLR, P1, P2, AUX, EXIT);
    "IF" IT = 1 "THEN"
    "BEGIN" TRISOL(N, BLR, P1, P2, F, DX); BETA:= NRM(N, DX); OM:= 1
    "END" "ELSE"
    "BEGIN" TRISOL(N, BLR, P1, P2, F0, D10);
    ELMVEC(1, N, 0, D10, DX, 1/LABDA);
    TRISOL(N, BLR, P1, P2, F, DX); ELMVEC(1, N, 0, D01, DX, -1);
    OM1:= NRM(N, D10) * LABDA / NRMDX ** 2; BETA:= NRM(N, DX);
    OM:= NRM(N, D01) / (NRMDX * BETA);
    "IF" OM1 > OM "THEN" OM:= OM1
    "END"; NRMDX:= BETA;
    DUPVEC(1, N, 0, F0, F);
    KAPPA:= AUX[5] * NRMDX / SLEVEL;
    TRISOL(N, BLR, P1, P2, V, D01); ETA:= NRM(N, D01);
    "IF" ANL "THEN"
    "BEGIN" E:= (AUX[5] * (EPRJ + MACHEPS * N * 16) + EPAJ) * ETA;
    "IF" E > 1 - MACHEPS "THEN" E:= 1 - MACHEPS
    "END"
  "END" CDATALR;

  "REAL" "PROCEDURE" LEVELFU(X, F); "ARRAY" X, F;
  LEVELFU:= "IF" ^FUN(N, X, F) "THEN" RGIANT "ELSE" NRM(N, F);

  "PROCEDURE" EXIT(ERR); "VALUE" ERR; "INTEGER" ERR;
  "BEGIN" ER:= ERR; "GOTO" END "END" EXIT;

  "COMMENT" INITIALIZATION;
  SETRANDOM(1 / N);
  "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO" V[I]:= (RANDOM - 0.5) * 2;
  MULVEC(1, N, 0, V, V, 1 / NRM(N, V));
  MACHEPS:= ARREB; DLF:= PREC[1]; DLRX:= PREC[2]; DLAX:= PREC[3];
  EPRF:= PREC[4]; EPAF:= PREC[5]; EPRJ:= PREC[6]; EPAJ:= PREC[7];
  AUX[2]:= MACHEPS; AUX[4]:= 8; ER:= 0;
  RGIANT:= GIANT; ANL:= "TRUE"; UPDATE:= "TRUE";
  BETA:= OM:= ETA:= KAPPA:= 1; E:= 0; IT:= CNTF:= CNTJ:= 0;
  ITMAX:= 40; LABDA:= 1; NRMX:= NRM(N, X); SLEVEL:= NRM(N, F);
  EPF:= (EPRF + MACHEPS) * SLEVEL + EPAF;
  MONITOR(0, N, IT, ITMAX, IT, CNTF, CNTJ, X, F, SLEVEL, B,
  UPDATE, 0, V, V, LABDA, OM, BETA, KAPPA, ETA, E, ER, 0);

LOOP: IT:= IT + 1; CDATALR;
  RESBIS(N, X, NRMX, DX, NRMDX, LABDA, F, SLEVEL, LEVELFU, ER,
  EPF, CNT); NRMX:= NRM(N, X); CNTF:= CNTF + CNT;
  "IF" ^ANL "AND" ER ^= 0 "THEN"
  "BEGIN" ER:= 0; E:= 1 - MACHEPS "END";
  EPF:= (MACHEPS + EPRF) * SLEVEL + EPAF;
  MONITOR(1, N, IT, ITMAX, IT, CNTF, CNTJ, X, F, SLEVEL, B,
  UPDATE, 0, V, V, LABDA, OM, BETA, KAPPA, ETA, E, ER, 0);
  "IF" ^STOPFUL(NRMDX, NRMX * DLRX + DLAX, SLEVEL, DLF, EPF,
  EPAF, ^ANL, KAPPA, E, OM, BETA, LABDA, ER, IT, ITMAX,
  EXIT) "THEN" "GOTO" LOOP;
END: RES[1]:= ER; RES[2]:= SLEVEL; RES[3]:= IT; RES[4]:= IT;
  RES[6]:= CNTF; RES[7]:= CNTJ; RES[10]:= AUX[5] * ETA;
  MONITOR(2, N, IT, ITMAX, IT, CNTF, CNTJ, X, F, SLEVEL, B,
  UPDATE, 0, V, V, LABDA, OM, BETA, KAPPA, ETA, E, ER, 0)
"END" ABU;
        "EOP"
```

```
"CODE" 99851;
"PROCEDURE" GAS(N, X, F, B, FUN, JACOB, PREC, RES, MONITOR);
"VALUE" N; "INTEGER" N; "ARRAY" X, F, B, PREC, RES;
"BOOLEAN" "PROCEDURE" FUN; "PROCEDURE" JACOB, MONITOR;
"BEGIN" "BOOLEAN" SLOW;
  "INTEGER" I, IT, ITMAX, ER, RK;
  "REAL" SLEVEL, ETA, GAMMA, E, NRMDX, NRMX, MACHEPS, EPF, EPRF,
    EPAF, EPRJ, EPAJ, DLF, DLRX, DLAX, KAPPA, SL0;
  "ARRAY" DX, XJ, VAL, W, V[1:N], BV[1:N,1:N], EM[0:7];

  "REAL" "PROCEDURE" VECVEC(L, U, S, A, B); "CODE" 34010;
  "PROCEDURE" DUPVEC(L, U, S, A, B); "CODE" 31030;
  "PROCEDURE" SETRANDOM(X); "CODE" 11014;
  "REAL" "PROCEDURE" RANDOM; "CODE" 11015;
  "PROCEDURE" ELMVEC(L, U, S, A, B, X); "CODE" 34020;
  "PROCEDURE" MULVEC(L, U, S, A, B, X); "CODE" 31020;
  "PROCEDURE" FULMATVEC(LR,UR,LC,UC,A,B,C); "CODE" 31500;
  "PROCEDURE" FULTAMVEC(LR,UR,LC,UC,A,B,C); "CODE" 31501;
  "REAL" "PROCEDURE" ARREB; "CODE" 30002;
  "REAL" "PROCEDURE" NRM(N, X); "CODE" 99801;
  "PROCEDURE" SVDEC(N, A, VAL, V, EM, EXIT); "CODE" 99812;
  "PROCEDURE" SVSOL(N, U, VAL, V, R, RHS, SOL); "CODE" 99813;
  "BOOLEAN" "PROCEDURE" STOPSPL(NDX, DLX, NMF, DLF, EPAF, ER,
  IT, ITMAX, EXIT); "CODE" 99811;

  "PROCEDURE" CDATASV;
  "BEGIN" "REAL" CK, ND; "ARRAY" W0[1:N];
    "IF" IT = 1 "THEN"
    "BEGIN" GAMMA:= 1; FULMATVEC(1, N, 1, N, B, V, W)
    "END" "ELSE"
    "BEGIN" JACOB(N, X, B);
      ELMVEC(1, N, 0, XJ, X, -1); ND:= NRM(N, XJ);
      DUPVEC(1, N, 0, W0, W);
      FULMATVEC(1, N, 1, N, B, V, W);
      ELMVEC(1, N, 0, W0, W, -1);
      GAMMA:= ("IF" ND < MACHEPS * NRMX "THEN" 0
        "ELSE" NRM(N, W0) / ND)
    "END"; SVDEC(N, B, VAL, BV, EM, EXIT);
    DUPVEC(1, N, 0, XJ, X);
    CK:= VAL[1] * EPRJ + EPAJ;
    RK:= 0; "FOR" I:= N "STEP" -1 "UNTIL" 1 "DO"
    "BEGIN" "IF" VAL[I] > CK "THEN"
      "BEGIN" RK:= I; "GOTO" OUT "END"
    "END"; EXIT(7);
  OUT: ETA:= 1 / VAL[RK]; E:= CK * ETA;
    "IF" E > 1 - MACHEPS "THEN" E:= 1 - MACHEPS;
    SVSOL(N, B, VAL, BV, RK, F, DX); NRMDX:= NRM(N, DX);
    KAPPA:= VAL[1] * NRMDX / SLEVEL;
    FULTAMVEC(1, N, 1, RK, B, F, W0);
    "IF" NRM(N, W0) < EPF "THEN" ER:= 3
  "END" CDATASV;

  "REAL" "PROCEDURE" LEVELFU(X, F); "ARRAY" X, F;
  "IF" FUN(N, X, F) "THEN" LEVELFU:= NRM(N, F) "ELSE" EXIT(11);

  "PROCEDURE" EXIT(ERR); "VALUE" ERR; "INTEGER" ERR;
  "BEGIN" ER:= ERR; "GOTO" END "END" EXIT
```

```
  COMMENT INITIALIZATION;
  SETRANDOM(1 / N); SLOW:= "FALSE";
  "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO" V[I]:= (RANDOM - 0.5)'* 2;
  MULVEC(1, N, 0, V, V, 1 / NRM(N, V));
  MACHEPS:= ARREB; DLF:= PREC[1]; DLRX:= PREC[2]; DLAX:= PREC[3];
  EPRF:= PREC[4]; EPAF:= PREC[5]; EPRJ:= PREC[6]; EPAJ:= PREC[7];
  EM[0]:= MACHEPS; EM[2]:= EM[6]:= MACHEPS * 10; EM[4]:= N * 10;
  IT:= 0; ITMAX:= 40; ER:= 0; RK:= N; E:= 0; GAMMA:= ETA:= 1;
  SLEVEL:= NRM(N, F); NRMX:= NRM(N, X);
  EPF:= (EPRF + MACHEPS) * SLEVEL + EPAF;
  MONITOR(0, N, IT, ITMAX, IT, 0, 0, X, F, SLEVEL, B, SLOW, 0,
  V, V, 1, GAMMA, NRMDX, KAPPA, ETA, E, ER, 0);

LOOP: IT:= IT + 1; CDATASV;
  ELMVEC(1, N, 0, X, DX, -1); NRMX:= NRM(N, X);
  SL0:= SLEVEL; SLEVEL:= LEVELFU(X, F);
  EPF:= (MACHEPS + EPRF) * SLEVEL + EPAF;
  "IF" ABS(SL0 - SLEVEL) > EPF "THEN" SLOW:= "FALSE" "ELSE"
  "BEGIN" "IF" SLOW "THEN" ER:= 2 "ELSE" SLOW:= "TRUE" "END";
  MONITOR(1, N, IT, ITMAX, IT, IT, IT, X, F, SLEVEL, B, SLOW, 0,
  V, V, 1, GAMMA, NRMDX, KAPPA, ETA, E, ER, 0);
  "IF" ^ STOPSPL(NRMDX, NRMX * DLRX + DLAX, SLEVEL, DLF, EPAF,
  ER, IT, ITMAX, EXIT) "THEN" "GOTO" LOOP;
END: RES[1]:= ER; RES[2]:= SLEVEL; RES[3]:= IT; RES[4]:= 0;
  RES[5]:= IT; RES[6]:= IT; RES[7]:= IT;
  RES[10]:= "IF" RK = 0 "THEN" 1 / MACHEPS "ELSE" VAL[1] / VAL[RK];
  MONITOR(2, N, IT, ITMAX, IT, IT, IT, X, F, SLEVEL, B, SLOW, 0,
  V, V, 1, GAMMA, NRMDX, KAPPA, ETA, E, ER, 0)
"END" GAS;
        "EOP"


"CODE" 99852;
"PROCEDURE" DBU(N, X, F, B, FUN, PREC, RES, MONITOR);
"VALUE" N; "INTEGER" N; "ARRAY" X, F, B, PREC, RES;
"BOOLEAN" "PROCEDURE" FUN; "PROCEDURE" MONITOR;
"BEGIN"
  "INTEGER" I, IT, ITMAX, CNT, CNTF, ER;
  "BOOLEAN" DIF, UPDATE;
  "REAL" SLEVEL, LABDA, NRMBI, KAPPA, OM, BETA, E, HS, NRMDX, NRMX,
    MACHEPS, EPF, EPRF, EPAF, EPRJ, EPAJ, DLF, DLRX, DLAX,
    NRMU1, NRMU2, RGIANT;
  "ARRAY" DX, XJ, F0, V[1:N], BLR[1:N,1:N], AUX[1:7];
  "INTEGER" "ARRAY" P1, P2[1:N];

  "PROCEDURE" DUPVEC(L, U, S, A, B); "CODE" 31030;
  "PROCEDURE" SETRANDOM(X); "CODE" 11014;
  "REAL" "PROCEDURE" RANDOM; "CODE" 11015;
  "REAL" "PROCEDURE" ARREB; "CODE" 30002;
  "REAL" "PROCEDURE" GIANT; "CODE" 30004;
  "REAL" "PROCEDURE" SUM(L, U, I, XI); "CODE" 99800;
  "REAL" "PROCEDURE" NRM(N, X); "CODE" 99801;
  "REAL" "PROCEDURE" CNDDIAG(N, X); "CWDE" 99802;
  "REAL" "PROCEDURE" VECVEC(L, U, S, A, B); "CODE" 34010;
  "PROCEDURE" ELMVEC(L, U, S, A, B, X); "CODE" 34020;
  "PROCEDURE" ELMCOLVEC(L, U, I, A, B, X); "CODE" 34022;
  "PROCEDURE" MULVEC(L, U, S, A, B, X); "CODE" 31020;
  "PROCEDURE" JACOBNNF(N, X, F, JAC, I, DI, FUN); "CODE" 34437;
  "PROCEDURE" TRIDEC(N, A, LR, P1, P2, AX, EXIT); "CODE" 99805;
  "PROCEDURE" TRISOL(N, LR, P1, P2, RHS, SOL); "CODE" 99806;
  "PROCEDURE" RESBIS(N, X, DX, NRMDX, LABDA, F, LEVEL, LFUN, ER,
  EPF, CNT); "CODE" 99809;                                    "COMMENT"
```

```
#;
"BOOLEAN" "PROCEDURE" STOPFUL(NRMDX, DELX, NRMF, DELF, EPF,
EPAF, UPD, KAPPA, E, OM, BETA, LABDA, ER, IT, ITMAX,
EXIT); "CODE" 99810;

"PROCEDURE" CALHS;
"BEGIN" "INTEGER" I; "REAL" C1, C2, S, AID;
  NRMU1:= NRMU2:= 0;
  "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
  "BEGIN" AID:= (ABS(X[I]) + 1) ** 2;
  NRMU1:= NRMU1 + AID; NRMU2:= NRMU2 + 1/AID
  "END";
  NRMU1:= SQRT(NRMU1); NRMU2:= SQRT(NRMU2);
  C1:= NRMU1 * OM * 0.5; C2:= NRMU2 * EPF * NRMBI * 2;
  S:= C1 * C2; "IF" S <= MACHEPS "THEN" S:= MACHEPS;
  HS:= (-1 + SQRT(1 + 1/S)) * C2; "IF" HS > 1 "THEN" HS:= 1;
  "IF" HS < MACHEPS * 100 "THEN" HS:= MACHEPS * 100;
"END" CALHS;

"BOOLEAN" "PROCEDURE" CONUPDJAC;
"IF" IT < 3 "OR" ^UPDATE "OR" E >= 0.1
"THEN" CONUPDJAC:= "FALSE" "ELSE"
"BEGIN" "INTEGER" J; "REAL" PU, NRMU;
  "ARRAY" U, DF[1:N];
  DUPVEC(1, N, 0, DF, F); ELMVEC(1, N, 0, DF, F0, -1);
  TRISOL(N, BLR, P1, P2, DF, U);
  PU:= VECVEC(1, N, 0, DX, U); NRMU:= NRM(N, U);
  "IF" E < 1 "THEN"
  E:= (E / (1 - E) + (1 + NRMDX * 1.5 / NRMU) * NRMDX * OM)
  * (1 + E);
  "IF" KAPPA * E < 1 "AND" ABS(PU) > NRMDX * NRMU * MACHEPS
  "AND" E < 0.1 "THEN"
  "BEGIN" ELMVEC(1, N, 0, DF, F0, LABDA);
  "FOR" J:= 1 "STEP" 1 "UNTIL" N "DO"
  ELMCOLVEC(1, N, J, B, DF, U[J] / PU);
  CONUPDJAC:= "TRUE"
  "END" "ELSE" CONUPDJAC:= "FALSE"
"END" CONUPDJAC;

"PROCEDURE" CDATALR;
"BEGIN" "REAL" OM1; "ARRAY" D01, D10, W[1:N];
  "IF" IT > 1 "THEN"
  "BEGIN" TRISOL(N, BLR, P1, P2, F, D01);
    "IF" CONUPDJAC "THEN" DIF:= "FALSE" "ELSE"
    "BEGIN" CALHS; CNTF:= CNTF + N; DIF:= "TRUE";
      JACOBNNF(N, X, F, B, I, (ABS(X[I]) + 1) * HS, FUN)
  "END" "END";
  TRIDEC(N, B, BLR, P1, P2, AUX, EXIT);
  "IF" IT = 1 "THEN"
  "BEGIN" TRISOL(N, BLR, P1, P2, F, DX); BETA:= NRM(N, DX); OM:= 1
  "END" "ELSE"
  "BEGIN" TRISOL(N, BLR, P1, P2, F0, D10);
    ELMVEC(1, N, 0, D10, DX, 1/LABDA);
    TRISOL(N, BLR, P1, P2, F, DX); ELMVEC(1, N, 0, D01, DX, -1);
    OM1:= NRM(N, D10) * LABDA / NRMDX ** 2; BETA:= NRM(N, DX);
    OM:= NRM(N, D01) / (NRMDX * BETA);
    "IF" OM1 > OM "THEN" OM:= OM1
  "END"; NRMDX:= BETA;
  DUPVEC(1, N, 0, F0, F);
  KAPPA:= AUX[5] * NRMDX / SLEVEL;
  TRISOL(N, BLR, P1, P2, V, D01); NRMBI:= NRM(N, D01);      "COMMENT"
```

```
  #;
    "IF" DIF "THEN"
    "BEGIN" "REAL" C1, C2, AID; AID:= 1 - MACHEPS;
      C1:= NRMU1 * OM * 0.5; C2:= NRMU2 * NRMBI * EPF * 2;
      C2:= C2 / HS + C1 * HS; C1:= 1 - C1 * HS;
      E:= ("IF" C1 <= C2 * AID "THEN" AID "ELSE" C2 / C1)
    "END"
  "END" CDATALR;

  "REAL" "PROCEDURE" LEVELFU(X, F); "ARRAY" X, F;
  LEVELFU:= "IF" ^FUN(N, X, F) "THEN" RGIANT "ELSE" NRM(N, F);

  "PROCEDURE" EXIT(ERR); "VALUE" ERR; "INTEGER" ERR;
  "BEGIN" ER:= ERR; "GOTO" END "END" EXIT;

  "COMMENT" INITIALIZATION;
  SETRANDOM(1 / N);
  "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO" V[I]:= (RANDOM - 0.5) * 2;
  MULVEC(1, N, 0, V, V, 1 / NRM(N, V));
  MACHEPS:= ARREB; DLF:= PREC[1]; DLRX:= PREC[2]; DLAX:= PREC[3];
  EPRF:= PREC[4]; EPAF:= PREC[5];
  AUX[2]:= MACHEPS; AUX[4]:= 8; ER:= 0;
  RGIANT:= GIANT; DIF:= "TRUE"; UPDATE:= "TRUE";
  BETA:= OM:= NRMBI:= KAPPA:= 1; E:= 0; IT:= CNTF:= 0;
  ITMAX:= 40; LABDA:= 1; NRMX:= NRM(N, X); SLEVEL:= NRM(N, F);
  EPF:= (EPRF + MACHEPS) * SLEVEL + EPAF; CALHS;
  MONITOR(0, N, IT, ITMAX, IT, CNTF, 0, X, F, SLEVEL, B,
  UPDATE, 0, V, V, LABDA, OM, BETA, KAPPA, NRMBI, E, ER, HS);

LOOP: IT:= IT + 1; CDATALR;
  RESBIS(N, X, NRMX, DX, NRMDX, LABDA, F, SLEVEL, LEVELFU, ER,
  EPF, CNT); NRMX:= NRM(N, X); CNTF:= CNTF + CNT;
  "IF" ^DIF "AND" ER ^= 0 "THEN"
  "BEGIN" ER:= 0; E:= 1 - MACHEPS "END";
  EPF:= (MACHEPS + EPRF) * SLEVEL + EPAF;
  MONITOR(1, N, IT, ITMAX, IT, CNTF, 0, X, F, SLEVEL, B,
  UPDATE, 0, V, V, LABDA, OM, BETA, KAPPA, NRMBI, E, ER, HS);
  "IF" ^STOPFUL(NRMDX, NRMX * DLRX + DLAX, SLEVEL, DLF, EPF,
  EPAF, ^DIF, KAPPA, E, OM, BETA, LABDA, ER, IT, ITMAX, EXIT)
  "THEN" "GOTO" LOOP;
END: RES[1]:= ER; RES[2]:= SLEVEL; RES[3]:= IT; RES[4]:= IT;
  RES[6]:= CNTF; RES[10]:= AUX[5] * NRMBI;
  MONITOR(2, N, IT, ITMAX, IT, CNTF, 0, X, F, SLEVEL, B,
  UPDATE, 0, V, V, LABDA, OM, BETA, KAPPA, NRMBI, E, ER, HS)
"END" DBU;
        "EOP"
```

```
"CODE" 99853;
"PROCEDURE" GDS(N, X, F, B, FUN, PREC, RES, MONITOR); "VALUE" N;
"INTEGER" N; "ARRAY" X, F, B, PREC, RES;
"BOOLEAN" "PROCEDURE" FUN; "PROCEDURE" MONITOR;
"BEGIN" "BOOLEAN" SLOW;
  "INTEGER" I, IT, ITMAX, CNTF, ER, RK;
  "REAL" SLEVEL, NRMBI, GAMMA, E, EJ, HS, NRMDX, NRMX,
    MACHEPS, EPF, EPRF, EPAF, DLF, DLRX, DLAX, NRMU1, NRMU2,
    KAPPA, SL0;
  "ARRAY" DX, XJ, VAL, W, V[1:N], BV[1:N,1:N], EM[0:7];

  "REAL" "PROCEDURE" VECVEC(L, U, S, A, B); "CODE" 34010;
  "PROCEDURE" DUPVEC(L, U, S, A, B); "CODE" 31030;
  "PROCEDURE" SETRANDOM(X); "CODE" 11014;
  "REAL" "PROCEDURE" RANDOM; "CODE" 11015;
  "PROCEDURE" ELMVEC(L, U, S, A, B, X); "CODE" 34020;
  "PROCEDURE" MULVEC(L, U, S, A, B, X); "CODE" 31020;
  "PROCEDURE" FULMATVEC(LR,UR,LC,UC,A,B,C); "CODE" 31500;
  "PROCEDURE" FULTAMVEC(LR,UR,LC,UC,A,B,C); "CODE" 31501;
  "REAL" "PROCEDURE" ARREB; "CODE" 30002;
  "REAL" "PROCEDURE" NRM(N, X); "CODE" 99801;
  "PROCEDURE" JACOBNNF(N, X, F, B, I, HI, FUN); "CODE" 34437;
  "PROCEDURE" SVDEC(N, A, VAL, V, EM, EXIT); "CODE" 99812;
  "PROCEDURE" SVSOL(N, U, VAL, V, R, RHS, SOL); "CODE" 99813;
  "BOOLEAN" "PROCEDURE" STOPSPL(NDX, DLX, NMF, DLF, EPAF, ER,
  IT, ITMAX, EXIT); "CODE" 99811;

  "PROCEDURE" CALHSG;
  "BEGIN" "REAL" AID, C1, C2;
    NRMU1:= NRMU2:= 0;
    "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
    "BEGIN" AID:= (ABS(X[I]) + 1) ** 2;
      NRMU1:= NRMU1 + AID; NRMU2:= NRMU2 + 1 / AID;
    "END" ; NRMU1:= SQRT(NRMU1); NRMU2:= SQRT(NRMU2);
    C1:= NRMU1 * GAMMA * 0.5; C2:= NRMU2 * EPF * 2;
    HS:= "I|| C1 <= C2 "THEN" 1 "ELSE" SQRT(C2/C1);
    "IF" HS < MACHEPS * 100 "THEN" HS:= MACHEPS * 100
  "END" CALHSG;

  "PROCEDURE" CDATASV;
  "BEGIN" "REAL" CK, ND; "ARRAY" W0[1:N];
    "IF" IT = 1 "THEN"
    "BEGIN" GAMMA:= 1; FULMATVEC(1, N, 1, N, B, V, W)
    "END" "ELSE"
    "BEGIN" CALHSG; CNTF:= CNTF + N;
      JACOBNNF(N, X, F, B, I, (ABS(X[I]) + 1) * HS, FUN);
      ELMVEC(1, N, 0, XJ, X, -1); ND:= NRM(N, XJ);
      DUPVEC(1, N, 0, W0, W);
      FULMATVEC(1, N, 1, N, B, V, W);
      ELMVEC(1, N, 0, W0, W, -1);
      GAMMA:= ("IF" ND < MACHEPS * NRMX "THEN" 0
        "ELSE" NRM(N, W0) / ND)
    "END"; SVDEC(N, B, VAL, BV, EM, EXIT);
    DUPVEC(1, N, 0, XJ, X);
    CK:= NRMU1 * GAMMA * HS * 0.5 + NRMU2 * EPF * 2 / HS;
    RK:= 0; "FOR" I:= N "STEP" -1 "UNTIL" 1 "DO"
    "BEGIN" "IF" VAL[I] > CK "THEN"
      "BEGIN" RK:= I; "GOTO" OUT "END"
    "END"; EXIT(7);
  OUT: NRMBI:= 1 / VAL[RK]; E:= CK * NRMBI;
    "IF" E > 1 - MACHEPS "THEN" E:= 1 - MACHEPS;              "COMMENT"
```

```
  #;
    SVSOL(N, B, VAL, BV, RK, F, DX); NRMDX:= NRM(N, DX);
    KAPPA:= VAL[1] * NRMDX / SLEVEL;
    FULTAMVEC(1, N, 1, RK, B, F, W0);
    "IF" NRM(N, W0) < EPF "THEN" ER:= 3
  "END" CDATASV;

  "REAL" "PROCEDURE" LEVELFU(X, F); "ARRAY" X, F;
  "IF" FUN(N, X, F) "THEN" LEVELFU:= NRM(N, F) "ELSE" EXIT(11);

  "PROCEDURE" EXIT(ERR); "VALUE" ERR; "INTEGER" ERR;
  "BEGIN" ER:= ERR; "GOTO" END "END" EXIT;

  "COMMENT" INITIALIZATION;
  SETRANDOM(1 / N); SLOW:= "FALSE";
  "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO" V[I]:= (RANDOM - 0.5) * 2;
  MULVEC(1, N, 0, V, V, 1 / NRM(N, V));
  MACHEPS:= ARREB; DLF:= PREC[1]; DLRX:= PREC[2]; DLAX:= PREC[3];
  EPRF:= PREC[4]; EPAF:= PREC[5];
  EM[0]:= MACHEPS; EM[2]:= EM[6]:= MACHEPS * 10; EM[4]:= N * 10;
  IT:= 0; ER:= 0; RK:= N; CNTF:= 0; E:= 0; GAMMA:= NRMBI:= 1;
  ITMAX:= 40; SLEVEL:= NRM(N, F); NRMX:= NRM(N, X);
  EPF:= (EPRF + MACHEPS) * SLEVEL + EPAF; CALHSG;
  MONITOR(0, N, IT, ITMAX, IT, CNTF, 0, X, F, SLEVEL, B, SLOW,
  0, V, V, 1, GAMMA, NRMDX, KAPPA, NRMBI, E, ER, HS);

LOOP: IT:= IT + 1; CDATASV;
  ELMVEC(1, N, 0, X, DX, -1); NRMX:= NRM(N, X);
  CNTF:= CNTF + 1; SL0:= SLEVEL; SLEVEL:= LEVELFU(X, F);
  EPF:= (MACHEPS + EPRF) * SLEVEL + EPAF;
  "IF" ABS(SL0 - SLEVEL) > EPF "THEN" SLOW:= "FALSE" "ELSE"
  "BEGIN" "IF" SLOW "THEN" ER:= 2 "ELSE" SLOW:= "TRUE" "END";
  MONITOR(1, N, IT, ITMAX, IT, CNTF, 0, X, F, SLEVEL, B, SLOW,
  0, V, V, 1, GAMMA, NRMDX, KAPPA, NRMBI, E, ER, HS);
  "IF" ^ STOPSPL(NRMDX, NRMX * DLRX + DLAX, SLEVEL, DLF, EPAF,
  ER, IT, ITMAX, EXIT) "THEN" "GOTO" LOOP;
END: RES[1]:= ER; RES[2]:= SLEVEL; RES[3]:= IT; RES[4]:= 0;
  RES[5]:= IT; RES[6]:= CNTF;
  RES[10]:= "IF" RK = 0 "THEN" 1 / MACHEPS "ELSE" VAL[1] / VAL[RK];
  MONITOR(2, N, IT, ITMAX, IT, CNTF, 0, X, F, SLEVEL, B, SLOW,
  0, V, V, 1, GAMMA, NRMDX, KAPPA, NRMBI, E, ER, HS)
"END" GDS;
        "EOP"


"CODE" 99858;
"BOOLEAN" "PROCEDURE" ONEQJ(X, FU, DFU, DLF, DLRX, DLAX);
"VALUE" DLF, DLRX, DLAX; "REAL" X, DLF, DLRX, DLAX;
"REAL" "PROCEDURE" FU, DFU;
"BEGIN" "INTEGER" FAC, S, CT;
  "REAL" F0, F1, DF0, DF1, MAX, DX, TOL, X0, X1, Y, INT;
  "BOOLEAN" "PROCEDURE" ZEROINDER(X, A, F, DF, TOL);
  "CODE" 34453;

  X0:= X; F0:= F1:= FU(X); DF0:= DF1:= DFU(X); CT:= 0; FAC:= 1;
ITERATION: TOL:= DLRX * ABS(X) * DLAX; INT:= X - 0;
  "IF" ABS(F0) < ABS(F1) "THEN" FAC:= 1;
  "IF" ("IF" F1 >= 0 "THEN" F0 <= 0 "ELSE" F0 >= 0) "THEN"
  "BEGIN" Y:= X0; X1:= X;
    ONEQJ:= ZEROINDER(X, Y,
    ("IF" X=X0 "THEN" F0 "ELSE" "IF" X=X1 "THEN" F1 "ELSE" FU(X)),
    ("IF" X=X0 "THEN" DF0 "ELSE" "IF" X=X1 "THEN" DF1
    "ELSE" DFU(X)), DLRX * ABS(X) + DLAX)
  "END"
```

```
  "ELSE" "IF" ABS(INT) >= TOL * 2 "THEN"
  "BEGIN" MAX:= (ABS(X) + 1) * 10; S:= -SIGN(F1 * DF1);
    "IF" S=0 "THEN" S:= SIGN(INT);
    DX:= ("IF" ABS(F1) <= ABS(DF1 * TOL) "THEN" S * TOL * FAC
      "ELSE" "IF" FAC * ABS(F1) >= ABS(DF1 * MAX) "THEN"
      MAX * S "ELSE" -F1 * FAC / DF1);
    X0:= X; X:= X + DX;
    F0:= F1; F1:= FU(X); DF0:= DF1; DF1:= DFU(X);
    "IF" F0 = F1 "THEN" CT:= CT + 1 "ELSE" CT:= 0;
    FAC:= FAC * 2; "IF" CT = 3 "THEN"
    ONEQJ:= "FALSE" "ELSE" "GOTO" ITERATION
  "END" SEARCH FOR DIFFERENT SIGNS
  "ELSE" ONEQJ:= ABS(F1) < DLF
"END" SOLVING ONE EQUATION IN ONE VARIABLE WITH DERIVATIVE;
        "EOP"


"CODE" 99859;
"BOOLEAN" "PROCEDURE" ONEQ(X, FU, DLF, DLRX, DLAX);
"VALUE" DLF, DLRX, DLAX;
"REAL" X, DLF, DLRX, DLAX; "REAL" "PROCEDURE" FU;
"BEGIN" "INTEGER" FAC, S, CT;
  "REAL" Y, A, B, FA, FB, MAX, TOL, P, Q, DX, INT;
  "BOOLEAN" "PROCEDURE" ZEROIN(X, A, F, TOL); "CODE" 34150;

  B:= X; FB:= FU(X); TOL:= DLRX * ABS(X) + DLAX;
  A:= SQRT(TOL) + X; FA:= FU(A); CT:= 0; FAC:= 1;
ITERATION: "IF" ABS(FA) < ABS(FB) "THEN"
  "BEGIN" X:= FA; FA:= FB; FB:= X; X:= A; A:= B; B:= X; FAC:= 1
  "END"; TOL:= DLRX * ABS(X) + DLAX; INT:= B - A;
  "IF" ("IF" FA >= 0 "THEN" FB <= 0 "ELSE" FB >= 0) "THEN"
  "BEGIN" X:= A; Y:= B;
    ONEQ:= ZEROIN(X, Y,
    ("IF" X=A "THEN" FA "ELSE" "IF" X=B "THEN" FB "ELSE" FU(X)),
    DLRX * ABS(X) + DLAX)
  "END" "ELSE" "IF" ABS(INT) >= TOL * 2 "THEN"
  "BEGIN" MAX:= (ABS(X) + 1) * 10;
    P:= INT * FB; "IF" P >= 0 "THEN" Q:= FA - FB "ELSE"
    "BEGIN" P:= -P; Q:= FB - FA "END";
    S:= SIGN(P * Q); "IF" SIGN = 0 "THEN" S:= SIGN(INT); A:= B;
    DX:= ("IF" P <= ABS(Q * TOL) "THEN" S * TOL * FAC "ELSE"
      "IF" FAC * P > ABS(Q * MAX) "THEN" MAX * S "ELSE"
      P * FAC / Q);
    X:= B:= B + DX; FA:= FB; FB:= FU(B); FAC:= FAC * 2;
    "IF" FA = FB "THEN" CT:= CT + 1 "ELSE" CT:= 0;
    "IF" CT = 3 "THEN" ONEQ:= "FALSE" "ELSE" "GOTO" ITERATION
  "END" SEARCH FOR DIFFERENT SIGNS
  "ELSE" ONEQ:= ABS(FB) < DLF
"END" SOLVING ONE EQUATION IN ONE VARIABLE WITHOUT DERIVATIVE;
        "EOP"
```

```
"CODE" 99854;
"PROCEDURE" SNOLEQJ(N, X, FUN, JACOB, PREC, RES, METHODS, MONITOR);
"VALUE" N; "INTEGER" N; "ARRAY" X, PREC, RES;
"BOOLEAN" "PROCEDURE" FUN; "PROCEDURE" JACOB, METHODS, MONITOR;
"BEGIN" "BOOLEAN" CLASI, GENI, SCAL, UPDATE;
  "INTEGER" I, IT, CNTF, CNTJ, CNTDECLR, CNTDECSV, SC, ERROR;
  "REAL" NRMF, MACHEPS;
  "ARRAY" F, ROWS, COLS[1:N], B[1:N,1:N];

  "REAL" "PROCEDURE" VECVEC(L, U, S, A, B); "CODE" 34010;
  "PROCEDURE" COLCST(L, U, I, A, X); "CODE" 31131;
  "PROCEDURE" ROWCST(L, U, I, A, X); "CODE" 31132;
  "REAL" "PROCEDURE" ARREB; "CODE" 30002;
  "REAL" "PROCEDURE" NRM(N, X); "CODE" 99801;
  "PROCEDURE" ABU(N, X, F, B, FUN, JACOB, PREC, RES, MONITOR);
  "CODE" 99850;
  "PROCEDURE" GAS(N, X, F, B, FUN, JACOB, PREC, RES, MONITOR);
  "CODE" 99851;

  "PROCEDURE" MONIT(PH, N, IC, IMAX, DC, FC, JC, X, F, NRMF, B,
  UPD, S, R, C, LAB, OM, BT, KP, NBI, E, ER, HS);
  "VALUE" PH, N, DC, FC, JC, S, LAB, HS;
  "INTEGER" PH, N, IC, IMAX, DC, FC, JC, S, ER; "BOOLEAN" UPD;
  "REAL" NRMF, LAB, OM, BT, KP, NBI, E, HS;
  "ARRAY" X, F, B, R, C;
  "BEGIN" "IF" PH = 0 "THEN"
    "BEGIN" IMAX:= "IF" SCAL "AND" CLASI "THEN" 20 "ELSE" 40;
      UPD:= UPDATE
    "END";
    MONITOR(PH, N, IC, IMAX, DC, FC, JC, X, F, NRMF, B, UPD,
    SC, ROWS, COLS, LAB, OM, BT, KP, NBI, E, ER, HS)
  "END" MONIT;

  CLASI:= GENI:= UPDATE:= "TRUE"; SCAL:= "FALSE";
  METHODS(CLASI, GENI, SCAL, UPDATE); ERROR:= 0;
  IT:= 0; CNTF:= 1; CNTDECLR:= CNTDECSV:= 0; SC:= 0;
  RES[1]:= RES[8]:= RES[9]:= 1;
  "IF" ^FUN(N, X, F) "THEN" "BEGIN" RES[1]:= 12; "GOTO" EXIT "END";
  NRMF:= NRM(N, F); MACHEPS:= ARREB;
  "IF" NRMF < MACHEPS "THEN"
  "BEGIN" ERROR:= 0; "GOTO" FINISH "END";
  JACOB(N, X, B); CNTJ:= 1;
  "IF" SCAL "AND" CLASI "THEN"
  "BEGIN" "BOOLEAN" FIRSTSCALE;
    "REAL" "PROCEDURE" CNDDIAG(N, D); "CODE" 99802;
    "INTEGER" "PROCEDURE" SCALE(N, A, X, F, ROWS, COLS); "CODE" 99803;
    "PROCEDURE" BACKSCALE(N, JAC, X, F, SC, ROWS, COLS); "CODE" 99804;

    "BOOLEAN" "PROCEDURE" SFUN(N, X, F); "VALUE" N;
    "INTEGER" N; "ARRAY" X, F;
    "BEGIN" "INTEGER" I; "BOOLEAN" OK;
      "IF" SC > 1 "THEN"
      "BEGIN" "ARRAY" X1[1:N];
        "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO" X1[I]:= X[I] * COLS[I];
        SFUN:= OK:= FUN(N, X1, F)
      "END" "ELSE" SFUN:= OK:= FUN(N, X, F);
      "IF" OK "AND" (SC = 1 "OR" SC = 3) "THEN"
      "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO" F[I]:= F[I] * ROWS[I]
    "END" SFUN
```

```
#;
  "PROCEDURE" SJACOB(N, X, B); "VALUE" N;
  "INTEGER" N; "ARRAY" X, B;
  "BEGIN" "INTEGER" I;
    "IF" SC > 1 "THEN"
    "BEGIN" "ARRAY" X1[1:N];
      "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO" X1[I]:= X[I] * COLS[I];
      JACOB(N, X1, B)
    "END" "ELSE" JACOB(N, X, B);
    "IF" SC = 1 "OR" SC = 3 "THEN"
    "BEGIN" "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
      ROWCST(1, N, I, B, ROWS[I])
    "END"; "IF" SC > 1 "THEN"
    "BEGIN" "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
      COLCST(1, N, I, B, COLS[I])
    "END"
  "END" SCALED JACOBIAN;

  SC:= SCALE(N, B, X, F, ROWS, COLS); FIRSTSCALE:= "TRUE";
AGAIN: "IF" SC > 0 "THEN"
  ABU(N, X, F, B, SFUN, SJACOB, PREC, RES, MONIT)
  "ELSE" "IF" SC = 0 "THEN"
  ABU(N, X, F, B, FUN, JACOB, PREC, RES, MONIT)
  "ELSE" "BEGIN" ERROR:= 5; "GOTO" OUT "END";
  ERROR:= "IF" FIRSTSCALE "THEN" RES[1]
  "ELSE" ERROR * 100 + RES[1];
  IT:= IT + RES[3]; CNTDECLR:= CNTDECLR + RES[4];
  CNTF:= CNTF + RES[6]; CNTJ:= CNTJ + RES[7];
  "IF" FIRSTSCALE "AND" RES[1] > 1 "AND" RES[1] < 10 "THEN"
  "BEGIN" "INTEGER" SC1; "ARRAY" ROWS1, COLS1[1:N];
    SC1:= SCALE(N, B, X, F, ROWS1, COLS1);
    FIRSTSCALE:= "FALSE";
    "IF" SC1 < 0 "THEN" ERROR := ERROR * 100 + 5;
    "IF" ("IF" SC1 <= 0 "THEN" "FALSE" "ELSE"
    (CNDDIAG(N, ROWS1) > 100 "OR" CNDDIAG(N, COLS1) > 100)) "THEN"
    "BEGIN" "INTEGER" I;
      "IF" SC1 = 1 "OR" SC1 = 3 "THEN"
      "BEGIN" "IF" SC <= 1 "THEN" SC := SC1 "ELSE"
        "IF" SC = 2 "THEN" SC:= 3;
        "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
        ROWS[I]:= ROWS[I] * ROWS1[I]
      "END"; "IF" SC1 = 2 "OR" SC1 = 3 "THEN"
      "BEGIN" "IF" SC <= 1 "THEN" SC:= SC1 "ELSE"
        "IF" SC = 1 "THEN" SC:= 3;
        "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
        COLS[I]:= COLS[I] * COLS1[I]
      "END"; "GOTO" AGAIN
    "END" "ELSE" "IF" SC1 ^= 0 "THEN"
    BACKSCALE(N, B, X, F, SC1, ROWS1, COLS1)
  "END" CHECK ON BAD SCALING AND POSSIBLE RERUN SCABU;
OUT: BACKSCALE(N, B, X, F, SC, ROWS, COLS);
  "IF" SC = 1 "OR" SC = 3 "THEN" RES[8]:= CNDDIAG(N, ROWS);
  "IF" SC > 1 "THEN" RES[9]:= CNDDIAG(N, COLS)
"END" SCALING ALLOWED "ELSE" "IF" CLASI "THEN"
"BEGIN" ABU(N, X, F, B, FUN, JACOB, PREC, RES, MONIT);
  ERROR:= RES[1]; IT:= IT + RES[3]; CNTDECLR:= CNTDECLR + RES[4];
  CNTF:= CNTF + RES[6]; CNTJ:= CNTJ + RES[7]
"END" ONE CALL OF ABU WITH UNSCALED FUNCTION
```

```
  #;
  "IF" GENI "AND" RES[1] > 0 "AND" RES[1] < 10 "THEN"
  "BEGIN" CLASI:= "FALSE";
    GAS(N, X, F, B, FUN, JACOB, PREC, RES, MONIT);
    ERROR:= ERROR * 100 + RES[1];
    IT:= IT + RES[3]; CNTDECSV:= CNTDECSV + RES[5];
    CNTF:= CNTF + RES[6]; CNTJ:= CNTJ + RES[7]
  "END" GENERALIZED METHOD;
FINISH: RES[1]:= ERROR; RES[2]:= NRM(N, F); RES[3]:= IT;
  RES[4]:= CNTDECLR; RES[5]:= CNTDECSV; RES[6]:= CNTF; RES[7]:= CNTJ;
EXIT:
"END" SNOLEQJ;
        "EOP"


"CODE" 99855;
"PROCEDURE" SNOLEQ(N, X, FUN, PREC, RES, METHODS, MONITOR);
"VALUE" N; "INTEGER" N; "ARRAY" X, PREC, RES;
"BOOLEAN" "PROCEDURE" FUN; "PROCEDURE" METHODS, MONITOR;
"BEGIN" "BOOLEAN" CLASI, GENI, SCAL, UPDATE;
  "INTEGER" I, IT, CNTF, CNTDECLR, CNTDECSV, SC, ERROR;
  "REAL" EPF, HS, NRMU1, NRMU2, NRMF, MACHEPS;
  "ARRAY" F, ROWS, COLS[1:N], B[1:N,1:N];

  "REAL" "PROCEDURE" VECVEC(L, U, S, A, B); "CODE" 34010;
  "PROCEDURE" JACOBNNF(N, X, F, JAC, I, DI, FUN); "CODE" 34437;
  "REAL" "PROCEDURE" ARREB; "CODE" 30002;
  "REAL" "PROCEDURE" NRM(N, X); "CODE" 99801;
  "PROCEDURE" DBU(N, X, F, B, FUN, PREC, RES, MONITOR);
  "CODE" 99852;
  "PROCEDURE" GDS(N, X, F, B, FUN, PREC, RES, MONITOR);
  "CODE" 99853;

  "PROCEDURE" MONIT(PH, N, IC, IMAX, DC, FC, JC, X, F, NRMF, B,
  UPD, S, R, C, LAB, OM, BT, KP, NBI, E, ER, HS);
  "INTEGER" PH, N, IC, IMAX, DC, FC, JC, S, ER; "BOOLEAN" UPD;
  "REAL" NRMF, LAB, OM, BT, KP, NBI, E, HS;
  "ARRAY" X, F, B, R, C;
  "BEGIN" "IF" PH = 0 "THEN"
    "BEGIN" IMAX:= "IF" SCAL "AND" CLASI "THEN" 20 "ELSE" 40;
      UPD:= UPDATE
    "END";
    MONITOR(PH, N, IC, IMAX, DC, FC, JC, X, F, NRMF, B, UPD,
    SC, ROWS, COLS, LAB, OM, BT, KP, NBI, E, ER, HS)
  "END" MONIT;

  CLASI:= GENI:= UPDATE:= "TRUE"; SCAL:= "FALSE";
  METHODS(CLASI, GENI, SCAL, UPDATE); ERROR:= 0;
  IT:= 0; CNTF:= 1; CNTDECLR:= CNTDECSV:= 0; SC:= 0;
  RES[1]:= RES[8]:= RES[9]:= 1;
  "IF" ^FUN(N, X, F) "THEN" "BEGIN" RES[1]:= 12; "GOTO" EXIT "END";
  NRMF:= NRM(N, F); MACHEPS:= ARREB;
  "IF" NRMF < MACHEPS "THEN"
  "BEGIN" ERROR:= 0; "GOTO" FINISH "END";
  EPF:= (PREC[4] + ARREB) * NRM(N, F) + PREC[5];
  NRMU1:= NRMU2:= 0; "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
  "BEGIN" "REAL" AID; AID:= (ABS(X[I]) + 1) ** 2;
    NRMU1:= NRMU1 + AID; NRMU2:= NRMU2 + 1 / AID
  "END"; NRMU1:= SQRT(NRMU1); NRMU2:= SQRT(NRMU2);
  HS:= (-1 + SQRT(1 + 1 / (NRMU1 * NRMU2 * EPF))) * EPF * NRMU2 * 2;
  JACOBNNF(N, X, F, B, I, (ABS(X[I]) + 1) * HS, FUN);        "COMMENT"
```

```
#;
CNTF:= CNTF + N; "IF" SCAL "AND" CLASI "THEN"
"BEGIN" "BOOLEAN" FIRSTSCALE;
  "REAL" "PROCEDURE" CNDDIAG(N, D); "CODE" 99802;
  "INTEGER" "PROCEDURE" SCALE(N, A, X, F, ROWS, COLS); "CODE" 99803;
  "PROCEDURE" BACKSCALE(N, JAC, X, F, SC, ROWS, COLS); "CODE" 99804;

  "BOOLEAN" "PROCEDURE" SFUN(N, X, F); "VALUE" N;
  "INTEGER" N; "ARRAY" X, F;
  "BEGIN" "INTEGER" I; "BOOLEAN" OK;
    "IF" SC > 1 "THEN"
    "BEGIN" "ARRAY" X1[1:N];
      "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO" X1[I]:= X[I] * COLS[I];
      SFUN:= OK:= FUN(N, X1, F)
    "END" "ELSE" SFUN:= OK:= FUN(N, X, F);
    "IF" OK "AND" (SC = 1 "OR" SC = 3) "THEN"
    "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO" F[I]:= F[I] * ROWS[I]
  "END" SFUN;

  SC:= SCALE(N, B, X, F, ROWS, COLS); FIRSTSCALE:= "TRUE";
AGAIN: "IF" SC > 0 "THEN"
  DBU(N, X, F, B, SFUN, PREC, RES, MONIT)
  "ELSE" "IF" SC = 0 "THEN"
  DBU(N, X, F, B, FUN, PREC, RES, MONIT)
  "ELSE" "BEGIN" ERROR:= 5; "GOTO" OUT "END";
  ERROR:= "IF" FIRSTSCALE "THEN" RES[1] "ELSE" ERROR * 100 + RES[1];
  IT:= IT + RES[3]; CNTDECLR:= CNTDECLR + RES[4];
  CNTF:= CNTF + RES[6];
  "IF" FIRSTSCALE "AND" RES[1] > 1 "AND" RES[1] < 10 "THEN"
  "BEGIN" "INTEGER" SC1; "ARRAY" ROWS1, COLS1[1:N];
     SC1:= SCALE(N, B, X, F, ROWS1, COLS1);
     FIRSTSCALE:= "FALSE";
    "IF" SC1 < 0 "THEN" ERROR:= ERROR * 100 + 5;
    "IF" ("IF" SC1 <= 0 "THEN" "FALSE" "ELSE"
    (CNDDIAG(N, ROWS1) > 100 "OR" CNDDIAG(N, COLS1) > 100)) "THEN"
    "BEGIN" "INTEGER" I;
      "IF" SC1 = 1 "OR" SC1 = 3 "THEN"
      "BEGIN" "IF" SC <= 1 "THEN" SC:= SC1
        "ELSE" "IF" SC = 2 "THEN" SC:= 3;
        "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
        ROWS[I]:= ROWS[I] * ROWS1[I]
      "END"; "IF" SC1 = 2 "OR" SC1 = 3 "THEN"
      "BEGIN" "IF" SC <= 1 "THEN" SC:= SC1
        "ELSE" "IF" SC = 1 "THEN" SC:= 3;
        "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
        COLS[I]:= COLS[I] * COLS1[I]
      "END"; "GOTO" AGAIN
    "END" "ELSE" "IF" SC1 ^= 0 "THEN"
    BACKSCALE(N, B, X, F, SC1, ROWS1, COLS1)
  "END" CHECK ON BAD SCALING AND POSSIBLE RERUN SCDBU;
OUT: BACKSCALE(N, B, X, F, SC, ROWS, COLS);
  "IF" SC=1 "OR" SC=3 "THEN" RES[8]:= CNDDIAG(N, ROWS);
  "IF" SC>1 "THEN" RES[9]:= CNDDIAG(N, COLS)
"END" SCALING ALLOWED "ELSE" "IF" CLASI "THEN"
"BEGIN" DBU(N, X, F, B, FUN, PREC, RES, MONIT);
  ERROR:= RES[1]; IT:= IT + RES[3]; CNTF:= CNTF + RES[6];
  CNTDECLR:= CNTDECLR + RES[4]
"END" ONE CALL OF DBU WITH UNSCALED FUNCTION
```

```
  #;
  "IF" GENI "AND" RES[1] > 0 "AND" RES[1] < 10 "THEN"
  "BEGIN" CLASI:= "FALSE"; GDS(N, X, F, B, FUN, PREC, RES, MONIT);
    IT:= IT + RES[3]; CNTDECSV:= CNTDECSV + RES[5];
    CNTF:= CNTF + RES[6]; ERROR:= ERROR * 100 + RES[1]
  "END" GENERALIZED METHOD;
FINISH: RES[1]:= ERROR; RES[2]:= NRM(N, F); RES[3]:= IT;
  RES[4]:= CNTDECLR; RES[5]:= CNTDECSV; RES[6]:= CNTF; RES[7]:= 0;
EXIT:
"END" SNOLEQ;
        "EOP"


"CODE" 99856;
"PROCEDURE" REDUCEJ(N, P, X, LA, LB, FUN, JACOB, PREC, RES,
  METHODS, MONITOR);
"VALUE" N, P; "INTEGER" N, P; "ARRAY" X, LA, LB, PREC, RES;
"BOOLEAN" "PROCEDURE" FUN; "PROCEDURE" JACOB, METHODS, MONITOR;
"BEGIN"
  "PROCEDURE" SNOLEQJ(N, X, FUN, JAC, PRC, RS, MTH, MON); "CODE" 99854;
  "BOOLEAN" "PROCEDURE" ONEQJ(X, F, DF, TOLR, TOLA); "CODE" 99858;

  "BOOLEAN" "PROCEDURE" REDFUN(P, Z, F); "VALUE" P;
  "INTEGER" P; "ARRAY" Z, F;
  "BEGIN" "ARRAY" X[1:N];
    FULMATVEC(1, N, 1, P, UUS, Z, X);
    ELMVEC(1, N, 0, X, UY, 1); REDFUN:= FUN(N, X, F)
  "END" REDFUN;

  "PROCEDURE" REDJAC(P, Z, B); "VALUE" P;
  "INTEGER" P; "ARRAY" Z, B;
  "BEGIN" "INTEGER" I, J; "ARRAY" JAC[1:P,1:N];
    FULMATVEC(1, N, 1, P, UUS, Z, X);
    ELMVEC(1, N, 0, X, UY, 1); JACOB(N, X, JAC);
   "FOR" I:= 1 "STEP" 1 "UNTIL" P "DO"
   "FOR" J:= 1 "STEP" 1 "UNTIL" P "DO"
    B[I,J]:= MATMAT(1, N, I, J, JAC, UUS)
  "END" REDJAC;

  "REAL" "PROCEDURE" NRM(N, X); "CODE" 99801;
  "PROCEDURE" DUPMAT(L, U, I, J, A, B); "CODE" 31035;
  "PROCEDURE" INIMAT(LR,UR,LC,UC,A,X); "CODE" 31011;
  "PROCEDURE" ELMVEC(L, U, S, A, B, X); "CODE" 34020;
  "PROCEDURE" FULMATVEC(LR,UR,LC,UC,A,B,C); "CODE" 31500;
  "PROCEDURE" FULTAMVEC(LR,UR,LC,UC,A,B,C); "CODE" 31501;
  "REAL" "PROCEDURE" MATMAT(L, U, I, J, A, B); "CODE" 34013;
  "PROCEDURE" SVDEC(N, A, VAL, V, EM, EXIT); "CODE" 99812;
  "REAL" "PROCEDURE" ARREB; "CODE" 30002;

  "PROCEDURE" EXIT(ER); "VALUE" ER; "INTEGER" ER;
  "BEGIN" RES[1]:= ER; "GOTO" OUT "END";

  "INTEGER" I, J, M, FCNT, MAXF, JCNT; "REAL" MEPS, AID;
  "BOOLEAN" BL; "ARRAY" UUS[1:N,1:P], UY[1:N];
  M:= N - P; "COMMENT" M IS THE NUMBER OF LINEAR COMPONENTS;
  MAXF:= 100; AID:= 0;
  "BEGIN" "ARRAY" EM[0:7], US, V[1:N,1:N], VAL[1:N];
    EM[0]:= MEPS:= ARREB; EM[2]:= EM[6]:= MEPS * 100; EM[4]:= N * 10;
    DUPMAT(1, M, 1, N, V, LA); INIMAT(M+1, N, 1, N, V, 0);
    SVDEC(N, V, VAL, US, EM, EXIT);                          "COMMENT"
```

```
  #;
    BL:= VAL[M] >= VAL[1] * MEPS * 100; "IF" BL "THEN"
    "BEGIN" FULTAMVEC(1, M, 1, M, V, LB, UY);
      "FOR" I:= 1 "STEP" 1 "UNTIL" M "DO" UY[I]:= UY[I]/VAL[I];
      FULMATVEC(1, N, 1, M, US, UY, UY);
      "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
      "FOR" J:= M+1 "STEP" 1 "UNTIL" N "DO"
      UUS[I,J-M]:= US[I,J]
    "END"
  "END";
  "IF" ^BL "THEN" RES[1]:= 13 "ELSE"
  "BEGIN" "ARRAY" Z[1:P];
    ELMVEC(1, N, 0, X, UY, -1); FULTAMVEC(1, N, 1, P, UUS, X, Z);
    "IF" P > 1 "THEN"
    SNOLEQJ(P, Z, REDFUN, REDJAC, PREC, RES, METHODS, MONITOR)
    "ELSE"
    "BEGIN" "REAL" ZZ; "INTEGER" IAID;
      "REAL" "PROCEDURE" F(X); "VALUE" X; "REAL" X;
      "BEGIN" "ARRAY" FU[1:1];
        Z[1]:= X; "IF" ^REDFUN(1, Z, FU) "THEN"
        "BEGIN" RES[1]:= 11; "GOTO" OUT1 "END";
        MONITOR(3, 1, FCNT, IAID, 1, FCNT, FCNT, Z, FU, ABS(FU[1]),
        UUS, BL, 0, UY, UY, 1, AID, AID, AID, AID, AID, IAID, 0);
        F:= FU[1]; FCNT:= FCNT + 1;
        "IF" FCNT >= MAXF "THEN"
        "BEGIN" RES[1]:= 4; "GOTO" OUT1 "END"
      "END" F;

      "REAL" "PROCEDURE" DF(X); "VALUE" X; "REAL" X;
      "BEGIN" "ARRAY" JAC[1:1,1:1]; Z[1]:= X;
        REDJAC(1,Z,JAC); DF:= JAC[1,1]; JCNT:= JCNT + 1
      "END";

      ZZ:= Z[1]; FCNT:= JCNT:= 0; BL:= "FALSE"; IAID:= 0; AID:= 0;
      "IF" ONEQJ(ZZ, F, DF, PREC[1], PREC[2], PREC[3])
      "THEN" RES[1]:= 0 "ELSE" RES[1]:= 14;
    OUT1: RES[3]:= FCNT - 1; RES[4]:= 0;
      RES[6]:= FCNT; RES[7]:= JCNT; Z[1]:= ZZ
    "END";
    FULMATVEC(1, N, 1, P, UUS, Z, X); ELMVEC(1, N, 0, X, UY, 1)
  "END";
OUT: FULMATVEC(1, M, 1, N, LA, X, UY); ELMVEC(1, M, 0, UY, LB, -1);
  "FOR" I:= M "STEP" -1 "UNTIL" 1 "DO" UY[I+P]:= UY[I];
  FUN(N, X, UY);
  MONITOR(2, N, FCNT-1, MAXF-1, 1, FCNT, JCNT, X, UY, NRM(N, UY),
  UUS, "FALSE", 0, UY, UY, 1, AID, AID, AID, AID, AID, RES[1], AID)
"END" POLYALGORITHM FOR SOLVING N LINEAR AND NONLINEAR EQUATIONS
      IN N VARIABLES BY REDUCTION, USING KNOWN DERIVATIVES;
         "EOP"


"CODE" 99857;
"PROCEDURE" REDUCE(N, P, X, LA, LB, FUN, PREC, RES, METHODS, MONITOR);
"VALUE" N, P; "INTEGER" N, P; "ARRAY" X, LA, LB, PREC, RES;
"BOOLEAN" "PROCEDURE" FUN; "PROCEDURE" METHODS, MONITOR;
"BEGIN"
  "PROCEDURE" SNOLEQ(N, X, FUN, PRC, RS, MTH, MON); "CODE" 99855;
  "BOOLEAN" "PROCEDURE" ONEQ(X, F, TOLR, TOLA); "CODE" 99859;
                                                        "COMMENT"
```

```
#;
"BOOLEAN" "PROCEDURE" REDFUN(P, Z, F); "VALUE" P;
"INTEGER" P; "ARRAY" Z, F;
"BEGIN" "ARRAY" X[1:N];
  FULMATVEC(1, N, 1, P, UUS, Z, X);
  ELMVEC(1, N, 0, X, UY, 1); REDFUN:= FUN(N, X, F)
"END" REDFUN;

"REAL" "PROCEDURE" NRM(N, X); "CODE" 99801;
"PROCEDURE" DUPMAT(L, U, I, J, A, B); "CODE" 31035;
"PROCEDURE" INIMAT(LR,UR,LC,UC,A,X); "CODE" 31011;
"PROCEDURE" ELMVEC(L, U, S, A, B, X); "CODE" 34020;
"PROCEDURE" FULMATVEC(LR,UR,LC,UC,A,B,C); "CODE" 31500;
"PROCEDURE" FULTAMVEC(LR,UR,LC,UC,A,B,C); "CODE" 31501;
"PROCEDURE" SVDEC(N, A, VAL, V, EM, EXIT); "CODE" 99812;
"REAL" "PROCEDURE" ARREB; "CODE" 30002;

"PROCEDURE" EXIT(ER); "VALUE" ER; "INTEGER" ER;
"BEGIN" RES[1]:= ER; "GOTO" OUT "END";

"INTEGER" I, J, M, FCNT, JCNT, MAXF; "REAL" MEPS, AID;
"BOOLEAN" BL; "ARRAY" UUS[1:N,1:P], UY[1:N];
M:= N - P; "COMMENT" M IS THE NUMBER OF LINEAR COMPONENTS;
MAXF:= 100; AID:= 0;
"BEGIN" "ARRAY" EM[0:7], US, V[1:N,1:N], VAL[1:N];
  EM[0]:= MEPS:= ARREB; EM[2]:= EM[6]:= MEPS * 100; EM[4]:= N * 10;
  DUPMAT(1, M, 1, N, V, LA); INIMAT(M+1, N, 1, N, V, 0);
  SVDEC(N, V, VAL, US, EM, EXIT);
  BL:= VAL[M] >= VAL[1] * MEPS * 100; "IF" BL "THEN"
  "BEGIN" FULTAMVEC(1, M, 1, M, V, LB, UY);
    "FOR" I:= 1 "STEP" 1 "UNTIL" M "DO" UY[I]:= UY[I]/VAL[I];
    FULMATVEC(1, N, 1, M, US, UY, UY);
    "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
    "FOR" J:= M+1 "STEP" 1 "UNTIL" N "DO"
    UUS[I,J-M]:= US[I,J]
  "END"
"END";
"IF" ^BL "THEN" RES[1]:= 13 "ELSE"
"BEGIN" "ARRAY" Z[1:P];
  ELMVEC(1, N, 0, X, UY, -1); FULTAMVEC(1, N, 1, P, UUS, X, Z);
  "IF" P > 1 "THEN"
  SNOLEQ(P, Z, REDFUN, PREC, RES, METHODS, MONITOR)
  "ELSE"
  "BEGIN" "REAL" ZZ; "INTEGER" IAID;
    "REAL" "PROCEDURE" F(X); "VALUE" X; "REAL" X;
    "BEGIN" "ARRAY" FU[1:1];
      Z[1]:= X; "IF" ^REDFUN(1, Z, FU) "THEN"
      "BEGIN" RES[1]:= 11; "GOTO" OUT1 "END";
      MONITOR(3, 1, FCNT, IAID, 1, FCNT, 0, Z, FU, ABS(FU[1]),
      UUS, BL, 0, UY, UY, 1, AID, AID, AID, AID, AID, IAID, 0);
      F:= FU[1]; FCNT:= FCNT + 1;
      "IF" FCNT >= MAXF "THEN"
      "BEGIN" RES[1]:= 4; "GOTO" OUT1 "END"
    "END" F;

    ZZ:= Z[1]; FCNT:= JCNT:= 0; BL:= "FALSE"; AID:= 0; IAID:=0;
    "IF" ONEQ(ZZ, F, PREC[1], PREC[2], PREC[3])
    "THEN" RES[1]:= 0 "ELSE" RES[1]:= 14;
  OUT1: RES[3]:= FCNT - 1; RES[4]:= 0;
    RES[6]:= FCNT; RES[7]:= JCNT; Z[1]:= ZZ
  "END"
```

```
    ; FULMATVEC(1, N, 1, P, UUS, Z, X); ELMVEC(1, N, 0, X, UY, 1)
  "END";
OUT: FULMATVEC(1, M, 1, N, LA, X, UY); ELMVEC(1, M, 0, UY, LB, -1);
  "FOR" I:= M "STEP" -1 "UNTIL" 1 "DO" UY[I+P]:= UY[I];
  FUN(N, X, UY);
  MONITOR(2, N, FCNT-1, MAXF-1, 1, FCNT, JCNT, X, UY, NRM(N, UY),
  UUS, "FALSE", 0, UY, UY, 1, AID, AID, AID, AID, AID, RES[1], AID)
"END" POLYALGORTIHM FOR SOLVING N LINEAR AND NONLINEAR EQUATIONS
  IN N VARIABLES WITHOUT KNOWN DERIVATIVES BY REDUCTION;
        "EOP"
```