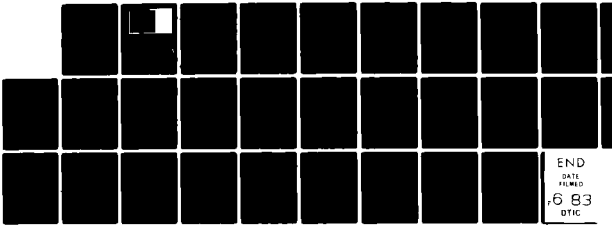
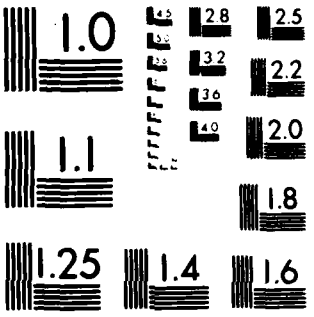
AD-A127 782 COMPUTATIONAL IMPLEMENTATION OF THE MULTIVARIATE HALLEY 1/1
METHOD FOR SOLVIN..(U) WISCONSIN UNIV-MADISON
MATHEMATICS RESEARCH CENTER A A CUYT ET AL. FEB 83
UNCLASSIFIED MRC-TSR-2481 DAAG29-80-C-0041 F/G 12/1 NL





MRC Technical Summary Report #2481

# COMPUTATIONAL IMPLEMENTATION OF THE MULTIVARIATE HALLEY METHOD FOR SOLVING NONLINEAR SYSTEMS OF EQUATIONS

Annie A. M. Cuyt and L. B. Rall

**Mathematics Research Center**
**University of Wisconsin—Madison**
**610 Walnut Street**
**Madison, Wisconsin 53706**

February 1983

(Received February 11, 1983)



**Approved for public release**
**Distribution unlimited**

Sponsored by

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park
North Carolina 27709

and

National Fund for Scientific Research (NWFO) of Belgium

UNIVERSITY OF WISCONSIN-MADISON
MATHEMATICS RESEARCH CENTER

# COMPUTATIONAL IMPLEMENTATION OF THE MULTIVARIATE HALLEY METHOD FOR SOLVING NONLINEAR SYSTEMS OF EQUATIONS

Annie A. M. Cuyt* and L. B. Rall†

Technical Summary Report #2481

February 1983

ABSTRACT

Halley's method for the solution of systems of equations is an iterative procedure which converges cubically under favorable conditions. The multivariate version requires the solution of two linear systems of equations with the same coefficient matrix, following which the correction vector is computed using componentwise multiplication and division of vectors. This report describes a general-purpose computer program which implements this method. The necessary first and second derivatives are obtained by automatic differentiation, so the user need only supply code defining the functions appearing in the system of equations. The program is written in Pascal-SC, using the new data type HESSIAN to represent dependent and independent variables. Numerical examples are given for two simple systems of equations to illustrate the use of the program and the effectiveness of the method.

AMS (MOS) Subject Classifications: 65-04, 65H10, 65V05

Key Words: Nonlinear systems of equations, Halley's method, automatic differentiation, type HESSIAN, Pascal-SC

Work Unit Number 3 - Numerical Analysis

---

*Department of Mathematics, University of Antwerp UIA. Research sponsored by the Belgian National Fund for Scientific Research (NFWO).

†Mathematics Research Center, University of Wisconsin-Madison. Research sponsored in part by the United States Army under Contract No. DAAG29-80-C-0041.

## SIGNIFICANCE AND EXPLANATION

One of the fundamental problems of scientific computation is the efficient numerical solution of systems of nonlinear equations in several variables. Methods are known which converge rapidly in theory, but require first and second partial derivatives of the functions appearing in the system with respect to the variables involved. The necessary derivatives can be evaluated automatically, without resort to numerical approximations, in programs produced by modern compilers which permit user-defined data types and operators. Examples of such compilers are Pascal-SC, Algol 68, and ADA (a trademark of the U. S. Department of Defense). In this study, Pascal-SC (Pascal for Scientific Computation) is used, since it supports accurate floating-point arithmetic for vectors and matrices, as well as scalars. The method taken to illustrate these capabilities is Halley's method, which requires second partial derivatives. By use of type HESSIAN, which consists of the value of a function of n variables, its gradient vector of first derivatives, and its Hessian matrix of second derivatives considered as a triple of basic real, vector, and matrix types, the user need only provide expressions or subroutines for the functions involved, and the compiler then produces code for the derivatives needed, without resort to inaccurate numerical or expensive symbolic differentiation. Since Halley's method converges cubically, its speed can offset the overhead of calculation of the necessary derivatives. A general purpose programming system is provided which consists of two parts: A program which is used to verify that the coding of the functions is correct, and another program which solves the actual system. The first program requires as input the number of equations, the necessary operators, and expressions or subroutines for the functions involved. The second program needs only the number of equations in the system and the already translated and verified code from the first program. A numerical example is given to show that this division of labor leads to efficient and effective numerical solution of a system of nonlinear equations by the method considered.

---

# COMPUTATIONAL IMPLEMENTATION OF THE MULTIVARIATE HALLEY METHOD FOR SOLVING NONLINEAR SYSTEMS OF EQUATIONS

Annie A. M. Cuyt* and L. B. Rall†

1. Nonlinear systems of equations. One of the central problems of scientific computation is the efficient numerical solution of systems of n equations

$$f_i(x_1,x_2,\ldots,x_n) = 0, \quad i = 1,2,\ldots,n, \tag{1.1}$$

in n unknowns $x_1, x_2,\ldots, x_n$. This is a special case of the *operator equation*

$$f(x) = 0, \tag{1.2}$$

in which $f:D \subset R^n \to R^n$, $0 \in R^n$ denotes the zero vector $0 = (0,0,\ldots,0)$, and $x \in R^n$ is sought. If f is an *affine* operator,

$$f(x) = Ax + b, \tag{1.3}$$

with the matrix $A = (a_{ij})$ and the vector $b = (b_1,b_2,\ldots,b_n)$ given, then the system (1.1) is said to be *linear*. This important special case is now fairly well understood from a computational as well as a theoretical standpoint. Otherwise, (1.1) is a nonlinear system, and the situation is quite different with respect to theoretical and practical methods for solution than in the linear case. Most of the methods investigated to date [9], [10] involve some form of iteration, and many also involve approximation of the nonlinear system by a linear system during the various steps of the solution process. It has been observed that some solution procedures work better than others on a given problem, so that in the absence of a clear-cut

---

*Department of Mathematics, University of Antwerp UIA. Research sponsored by the Belgian National Fund for Scientific Research (NFWO).

†Mathematics Research Center, University of Wisconsin-Madison. Research sponsored in part by the United States Army under Contract No. DAAG29-80-C-0041.

criterion for choosing the optimal method, it is advisable to have several choices available in the form of computer programs which are easy to use.

It will be assumed that the operator f corresponding to the system (1.1) has first and second Fréchet derivatives f', f" on its domain $D \subset R^n$ [10]. In this case, the first Fréchet derivative of f at x is represented by the *Jacobian matrix*

$$f'(x) = (\partial f_i(x)/\partial x_j), \tag{1.4}$$

and the second by the *Hessian operator*

$$f''(x) = (\partial^2 f_i(x)/\partial x_j \partial x_k) \tag{1.5}$$

[10]. Necessary values of the derivatives appearing in (1.4) and (1.5) will be obtained by automatic differentiation [12] so that the user need only supply expressions and subroutines for the n functions $f_i(x_1,x_2,\ldots,x_n)$ appearing in (1.1). This avoids both the labor of providing code for derivatives and the inaccuracy of numerical differentiation, as will be explained briefly in §3.

2. The multivariate Halley method. This method is based on the theory of abstract Padé approximants [2], [4], and conditions for its numerical stability have been given by Cuyt [3]. The abstract setting for this method is a Banach algebra [10]: $R^n$ with multiplication of vectors defined componentwise forms such a structure for the norm $\|x\|_\infty = \max_{(i)} |x_i|$, for example. Halley's method starts from an initial approximation $x^0$ to a solution $x = x^*$ of (1.2), and then defines a sequence $\{x^\nu\}$ of successive approximations by the following algorithm:

$$x^{\nu+1} = x^\nu + \frac{(a^\nu)^2}{a^\nu + \frac{1}{2}b^\nu}, \text{ where}$$

$$a^\nu = -f'(x^\nu)^{-1} f(x^\nu) \quad \text{(the } \textit{Newton correction}\text{), and} \tag{2.1}$$

$$b^\nu = f'(x^\nu)^{-1} f''(x^\nu) a^\nu a^\nu, \quad \nu = 0,1,2,\ldots .$$

In order to simplify the computation, the Jacobian matrix $f'(x^\nu)$ is not inverted in

the actual computation; rather, the linear system

$$f'(x^\nu)a^\nu = -f(x^\nu) \tag{2.2}$$

is solved for $a^\nu$, following which the linear system

$$f'(x^\nu)b^\nu = f''(x^\nu)a^\nu a^\nu \tag{2.3}$$

is then solved for $b^\nu$. Since the systems (2.2) and (2.3) have the same coefficient matrices, the decomposition of the Jacobian matrix $f'(x^\nu)$ used to solve (2.2) can also be used to solve (2.3), resulting in a saving of effort.

The total computation effort required for one step of Halley's method is thus:

1°. Evaluation of $f(x^\nu)$, $f'(x^\nu)$, $f''(x^\nu)$;

2°. Solution of (2.2) for $a^\nu$;

3°. Evaluation of $f''(x^\nu)a^\nu a^\nu$;

4°. Calculation of the *Halley correction* $(a^\nu)^2/(a^\nu + \frac{1}{2}b^\nu)$;

5°. Addition of the Halley correction to $x^\nu$.

This sequence of operations is more elaborate than required for Newton's method, which requires only the evaluation of $f(x^\nu)$, $f'(x^\nu)$, the solution of (2.2) for $a^\nu$, and finally the addition of $a^\nu$ to $x^\nu$ to obtain $x^{\nu+1}$. However, in favorable cases, the rate of convergence of Halley's method will be cubic, while Newton's method converges quadratically. Thus, the greater effort required for each step of Halley's method could be offset if fewer steps are required to obtain the accuracy desired.

In connection with the calculation of the Halley correction, an indeterminate form arises if $(a^\nu)_i = (a^\nu + \frac{1}{2}b^\nu)_i = 0$. In this case, the value of the Halley correction is defined by continuity, and its $i$th component is taken to be 0.

Other cubically convergent iteration processes which use the same information as Halley's method are *Chebyshev's method* [9], [10],

$$x^{\nu+1} = x^\nu + a^\nu - \tfrac{1}{2}b^\nu, \quad \nu = 0,1,2,\ldots, \tag{2.4}$$

which is slightly less complicated, and the *method of tangent hyperbolas* [9],

in which the correction vector $c^\nu$ is obtained by solving the linear system

$$[f'(x^\nu) + \tfrac{1}{2}f''(x^\nu)a^\nu]c^\nu = -f(x^\nu), \tag{2.5}$$

from which

$$x^{\nu+1} = x^\nu + c^\nu, \quad \nu = 0,1,2,\ldots . \tag{2.6}$$

This method is more complicated than Halley's method in that in (2.5), alteration of the coefficient matrix in (2.2) is required. In the scalar case (n = 1), Halley's method and the method of tangent hyperbolas coincide after division of the numerator and denominator of the Halley correction by $a^\nu$. The abstract version of Halley's method is defined in a Banach algebra; the Chebyshev and tangent hyperbola method, like Newton's method, do not require multiplication and division of elements, and hence can be defined in the more general setting of a Banach space [2], [9], [10].

3. Use of automatic differentiation. Newton's method and the methods of §2 are sometimes shunned because it is assumed that code has to be supplied for the derivatives, or because the functions $f_i$ are defined by subroutines, rather than expressions. Since the rules for differentiation are well understood, however, the computer can itself produce the required code by automatic differentiation of the given expressions or subroutines [12]. In the case of expressions, programs capable of obtaining first and second derivatives have been in use for some time [5], [7], [10]. More recently, differentiation methods for subroutines have also been developed [6], [12], [14]. Since the latter case is the most general, it will be examined here.

In primitive computing languages such as FORTRAN, automatic differentiation requires interpretation of expressions [5], [7], or precompilation [6], [12]. One of the significant recent advances in computer science has been the development of modern languages such as Pascal-SC, in which the performance of differentiation is based on user-defined data types and operators [13], [14]. To illustrate the basic idea, consider the simple scalar case of a real function f of a single real variable

x. The pair of values (f(x),f'(x)) is the basic example of a datum of type GRADIENT for a given value of x [13]. Writing F = (f(x),f'(x)) to represent an element of this new type of data, the next step is to define the corresponding arithmetic operations and functions in a computable form. For example, for G = (g(x),g'(x)), addition and multiplication are defined by

$$\begin{aligned} F + G &= (f(x)+g(x),f'(x)+g'(x)), \\ F*G &= (f(x)*g(x),f(x)*g'(x)+g(x)*f'(x)), \end{aligned} \tag{3.1}$$

respectively. Similarly, functions such as

$$SIN(F) = (\sin(f(x)),f'(x)*\cos(f(x))) \tag{3.2}$$

are readily definable in a form suitable for computational implementation. The independent variable x is represented by the GRADIENT variable X = (x,1), and an expression of the form

```
F := X*SIN(X + 4.0) - X**3;
```
(3.3)

is then used to obtain both the value of the function $f(x) = x \sin(x + 4.0) - x^3$ and its derivative $f'(x) = x \cos(x + 4.0) + \sin(x + 4.0) - 3x^2$ automatically. Thus, the user need only supply the code (3.3) for the function to be differentiated, once the standard set of GRADIENT operations and functions are available [13].

For the present purpose, second derivatives are needed, so the type GRADIENT is extended to type HESSIAN, a datum of which is the triple F = (f(x),f'(x),f"(x)). Once again, there is no problem in the implementation of arithmetic operations and standard functions, for example,

$$\begin{aligned} F + G &= (f(x)+g(x),f'(x)+g'(x),f"(x)+g"(x)) \\ F*G &= (f(x)*g(x),f(x)*g'(x)+g(x)*f'(x),f(x)*g"(x)+2*f'(x)*g'(x)+g(x)*f"(x)), \end{aligned} \tag{3.4}$$

and

$$SIN(F) = (\sin(f(x)),f'(x)*\cos(f(x)),f"(x)*\cos(f(x))-f'(x)*f'(x)*\sin(f(x))). \tag{3.5}$$

Thus, given the independent variable x as the HESSIAN variable X = (x,1,0), the

evaluation of the expression (3.3) yields the value of the second derivative $f''(x) = -x \sin(x + 4.0) + 2\cos(x + 4.0) - 6x$ as well as the values of the function $f(x)$ and its first derivative $f'(x)$.

Although the formulations of HESSIAN operators and functions are somewhat complicated, programming them is no real challenge, and this needs to be done only once and for all. When available, these subroutines shift the burden of differentiation from the user to the computing machine, which is as it should be. A complete package for arithmetic operators and standard functions has been prepared in Pascal-SC for the multivariate case in which f is a function of n variables, so that $x = (x_1, x_2, \ldots, x_n) \in R^n$, and the HESSIAN variable F is defined by

$$F = (f(x), \nabla f(x), Hf(x)), \tag{3.6}$$

where $\nabla f(x) = f'(x)$ denotes the *gradient vector*

$$\nabla f(x) = (\partial f(x)/\partial x_1, \partial f(x)/\partial x_2, \ldots, \partial f(x)/\partial x_n), \tag{3.7}$$

and $Hf(x) = f''(x)$ is the *Hessian matrix*

$$Hf(x) = (\partial^2 f(x)/\partial x_j \partial x_k) \tag{3.8}$$

of the real-valued function f at x. The HESSIAN variables X[j] corresponding to the independent variables $x_j$ are $X[j] = (x_j, e_j, 0)$, $j = 1,2,\ldots,n$, where $e_j$ is the jth unit vector, and 0 denotes the $n \times n$ zero matrix.

The case of a vector-valued operator f is handled by means of expressions or subroutines for the n functions $f_i$ appearing in (1.1), defined to be the corresponding HESSIAN variables F[i]. In this case, the ith row of the Jacobian matrix (1.4) is simply the gradient vector $\nabla f_i(x)$, while the ith "panel" of the Hessian operator (1.5) is given by the matrix $Hf_i(x)$.

4. Computation with bilinear operators. In the multivariate Halley method (2.1), the right-hand side of the linear system of equations (2.3) for $b^\nu$ is obtained by operating twice on the vector $a^\nu$ with the bilinear operator $f''(x^\nu)$, where the result of the first operation is a matrix, and the second yields a vector [10].

The way in which HESSIAN variables are defined makes it easy to implement these operations. In general, the bilinear operator

$$B = (b_{ijk}) \tag{4.1}$$

will be considered to be composed of n matrices

$$B_1 = (b_{1jk}),\ B_2 = (b_{2jk}),\ \ldots,\ B_n = (b_{njk}), \tag{4.2}$$

which will be called *i-panels*, or simply *panels* of B. For a vector $x \in R^n$, the matrix

$$A = (a_{ij}) = Bx = \left( \sum_{k=1}^{n} b_{ijk} x_k \right) \tag{4.3}$$

will have rows $A_i^T$ given by the matrix-vector product

$$A_i = B_i x, \quad i = 1,2,\ldots,n \tag{4.4}$$

Once the matrix A is formed by computing the vectors (4.4), then the vector

$$y = Ax = Bxx = \left( \sum_{j=1}^{n} \sum_{k=1}^{n} b_{ijk} x_k x_j \right) \tag{4.5}$$

is obtained by a single additional matrix-vector multiplication. Here, $B_i = Hf_i(x^\nu)$,

$$A_i = Hf_i(x^\nu)a^\nu, \quad i = 1,2,\ldots,n, \tag{4.6}$$

and thus

$$f''(x^\nu)a^\nu a^\nu = Aa^\nu, \tag{4.7}$$

so the required vector is obtained by a total of n + 1 matrix-vector multiplications.

In Pascal-SC, vectors and matrices are stored row-wise, so no transposition is required when forming the matrix A from the vectors $A_i$ given by (4.6) [15].

5. Programming Halley's method in Pascal-SC. Central to Pascal-SC, as well as to Pascal [1], is the concept of a *data type*. In Pascal-SC, vectors and matrices over type REAL (the set of floating-point numbers) are considered to be the standard types RVECTOR and RMATRIX, respectively [15]. Following the conventions of Pascal-SC,

n-dimensional vectors and matrices are declared in the program heading by:

```
CONST DIM = n;
 TYPE DIMTYPE = 1..DIM;
      RVECTOR = ARRAY[DIMTYPE]OF REAL;
      RMATRIX = ARRAY[DIMTYPE]OF RVECTOR;
```
(5.1)

and a number of ordinary operations of matrix and vector algebra are implemented [8]. Following (5.1), type HESSIAN is declared by:

```
TYPE HESSIAN = RECORD F: REAL;DF: RVECTOR;HF: RMATRIX END;
```
(5.2)

so that if an expression or the result of a subroutine for computing f(x) is assigned to the HESSIAN variable F, one has

(5.3) F.F = f(x), F.DF = ∇f(x), F.HF = Hf(x).

Step 1° of Halley's method as outlined in §2 is thus taken care of simply by expressing the independent variables $x_1, x_2, \ldots, x_n$ and the values of the functions $f_1, f_2, \ldots, f_n$ in the system (1.1) as HESSIAN variables. For example, consider the simple system of equations

$$e^{-x+y} - 0.1 = 0,$$
$$e^{-x-y} - 0.1 = 0, \tag{5.4}$$

investigated by Cuyt and Van der Cruyssen [2], [4]. The variables involved would be declared to be HESSIAN in the heading of the program (see Appendix A) by

```
VAR X,Y,F,G: HESSIAN;
```
(5.5)

and the functions corresponding to the left-hand sides of (5.4) by

```
F := HEXP(-X+Y) - 0.1;
G := HEXP(-X-Y) - 0.1;
```
(5.6)

in the body of the program. (The user is free to name and order both the independent and dependent variables in any convenient manner. A more systematic approach will be

discussed in the next section.) In (5.6), the convention that the names of standard functions for type HESSIAN begin with "H" has been followed. Evaluations of the expressions in (5.6) requires the following HESSIAN operators and functions:

```
                OPERATOR - (H: HESSIAN) RES: HESSIAN;
                OPERATOR - (HA,HB: HESSIAN) RES: HESSIAN;
(5.7)           OPERATOR - (H: HESSIAN;R: REAL) RES: HESSIAN;
                OPERATOR + (HA,HB: HESSIAN) RES: HESSIAN;
                FUNCTION HEXP(H: HESSIAN): HESSIAN;
```

source code for these is given in the heading of the program listed in Appendix A. The first operator calculates -H, the second HA-HB, and so on.

The independent variables X,Y are initialized as follows: Their *function value* (or simply *value*) parts X.F and Y.F are given initially by input from the user, and are subsequently calculated by the Halley iteration. Their *gradient* parts X.DF and Y.DF are assigned the constant values

```
(5.8)           X.DF[1]:=1; X.DF[2]:=0; Y.DF[1]:=0; Y.DF[2]:=1;
```

as are their *Hessian* parts

```
(5.9)           X.HF:=MRNULL; Y.HF:=MRNULL;
```

where MRNULL is a standard Pascal-SC function which returns the zero matrix.

Execution of the statements (5.6) thus completes step 1° of Halley's method. For step 2°, the Jacobian matrix of the system (5.4) is needed. It is assumed that the declaration

```
(5.10)          VAR JAC,L,M: RMATRIX; A,B,V: RVECTOR; NRS: BOOLEAN;
```

is in the heading of the program, where JAC denotes the desired Jacobian. It is obtained by the assignments

```
(5.11)          JAC[1]:=F.DF; JAC[2]:=G.DF;
```

since its rows are the gradient vectors $\nabla f(x)$ and $\nabla g(x)$, respectively.

Similarly, letting V denote the right-hand side of the linear system (2.2), one has

(5.12)
```
V[1]:= -F.F; V[2]:= -G.F;
```

and all that remains is to solve (2.2) for $A = a^{\nu}$ by means of a standard procedure for solving linear systems of equations, such as

(5.13)
```
SOLVLN(DIM,JAC,M,V,A,NRS);
```

in which the decomposition of JAC is stored as the matrix M, and additional right-hand sides will be expected as long as NRS = FALSE. This completes step 2°.

In step 3°, the matrix $f''(x^{\nu})a^{\nu}$ is computed as the matrix L. This is accomplished by means of the assignments

(5.14)
```
L[1]:= F.HF*A; L[2]:= G.HF*A;
```

using the standard Pascal-SC operator * for matrix by vector multiplication [15]. With this result, the vector $V = f''(x^{\nu})a^{\nu}a^{\nu}$ is obtained from

(5.15)
```
V:= L*A;
```

thus completing step 3°.

The calculation of the Halley correction (step 4°) requires first the computation of the vector $B = b^{\nu}$ by

(5.16)
```
SOLVLN(DIM,JAC,M,V,B,NRS);
```

where V is now obtained from (5.15), and NRS = TRUE. Addition of vectors and multiplication of vectors by real numbers are standard in Pascal-SC; however, componentwise multiplication and divison of vectors are not. Thus, the corresponding operators *,/ must be defined in the heading of the program by

(5.17)
```
OPERATOR * (VA,VB: RVECTOR) RES: RVECTOR;
   VAR U: RVECTOR;I: DIMTYPE;
   BEGIN FOR I:=1 TO DIM DO U[I]:=VA[I]*VB[I];
         RES:= U
   END;
```

and

```
OPERATOR / (VA,VB: RVECTOR) RES: RVECTOR;
   VAR U: RVECTOR; I: DIMTYPE;
   BEGIN FOR I:=1 TO DIM DO
(5.18)          IF (VA[I] = Ø) AND (VB[I] = Ø) THEN U[I]:=Ø
                ELSE U[I]:= VA[I]/VB[I];
                RES:= U
   END;
```

respectively. The division operator is tailored to yield 0 as the limit of the Halley correction as $(a^{\nu})_i \to 0$, which is valid as long as there is a neighborhood of $a^{\nu}$ which does not contain points for which the denominator of the Halley correction is zero while the numerator is nonzero. By the use of the operators (5.17) and (5.18), the Halley correction is the vector V given by

```
(5.19)          V:= (A*A)/(A + Ø.5*B);
```

which completes step 4°.

The final step of one Halley iteration is then

```
(5.20)          X.F:= X.F + V[1]; Y.F:= Y.F + V[2];
```

after which another iteration can be performed, if desired. A set of typical numerical results for this problem, using the program in Appendix A, is given in Appendix B.

6. A more general approach: Type SYSTEM. In the simple example discussed in the previous section, it was convenient to use the ordinary notation X,Y for the independent variables involved, and F,G for the dependent variables corresponding to the system (5.4). For larger systems, it is helpful to adopt a more formal notational convention, in terms of which a general-purpose program can be developed. This is done by the introduction of the data type SYSTEM, which is declared by:

```
(6.1)           TYPE SYSTEM = ARRAY[DIMTYPE]OF HESSIAN;
```

by the use of this data type, a declaration of the form

(6.2)
```
VAR X,F: SYSTEM;
```

can be used, for example, to introduce independent variables X[1],X[2],...,X[DIM], and dependent variables F[1],F[2],...,F[DIM]. Thus, the system of equations

$$\begin{aligned} 16x_1^4 + 16x_2^4 + x_3^4 - 16 &= 0, \\ x_1^2 + x_2^2 + x_3^2 - 3 &= 0, \\ x_1^3 - x_2 &= 0, \end{aligned} \tag{6.3}$$

taken from [13] would be coded as

```
F[1]:= 16*(X[1]**4) + 16*(X[2]**4) + X[3]**4 - 16;
F[2]:= X[1]**2 + X[2]**2 + X[3]**2 - 3;
F[3]:= X[1]**3 - X[2];
```
(6.4)

(parentheses are necessary in (6.4), since ** and * have the same priority in Pascal-SC [1]). Once the statements (6.4) are executed, the value F[i].F of each function, the ith row of the Jacobian matrix of the system F[i].DF, and the ith panel F[i].HF of the Hessian operator are all at the disposal of the programmer.

7. The programs SYSTEST and HALSYS. A brief description of two Pascal-SC programs which can be used to investigate the application of Halley's method to systems of equations will now be presented. First of all, good programming practice requires that the expressions and subroutines provided by the user produce the correct values of the functions $f_i(x)$ appearing in (1.1). The program SYSTEST is provided for this purpose. To use this program, the file SYSTEST.S containing its source code (see Appendix C) is created from SYSTEST.PROG and edited to include the correct value of DIM and expressions or subroutines for the functions F[I]. Source code for the required HESSIAN operators and functions is obtained from the file HESS_PAKET, which contains the 22 arithmetic operators, 5 power operators, and the functions HABS, HSQRT, HEXP, HLN, HARCTAN, HSIN, and HCOS [14]. As explained in

[14], the user may add any other needed HESSIAN functions, procedures, or operators to the program heading if needed to supplement the standard ones. For example, the system (6.4) requires that source code for the following operators appears in the program heading:

```
                OPERATOR * (K: INTEGER;H: HESSIAN) RES: HESSIAN;
                OPERATOR ** (R: REAL;K: INTEGER) RES: REAL;
                OPERATOR ** (H: HESSIAN;K: INTEGER) RES: HESSIAN;
(7.1)
                OPERATOR + (HA,HB: HESSIAN) RES: HESSIAN;
                OPERATOR - (H: HESSIAN;K: INTEGER) RES: HESSIAN;
                OPERATOR - (HA,HB: HESSIAN) RES: HESSIAN;
```

and one sets DIM = 3. After this program is compiled, upon execution it will request initial values for X[1].F,...,X[DIM].F, and then print out the values F[I].F of the functions in the system, the values of the first derivatives DF[I]/DX[J] = F[I].DF[J], and the second derivatives D2F[I]/DX[J]DX[K] = F[I].HF[J,K]. A typical set of output for the system (6.4) is given in Appendix D.

Once the correctness of coding for the system has been verified, the translated code for the procedure FCOMP(VAR X,F: SYSTEM;DIM: INTEGER) is entered in the external library HALLEY_LIB as subroutine number 777 [15] from the intermediate code (ZC) for the program SYSTEST. HALLEY_LIB also contains pretranslated code for the linear equation solver SOLVLN (subroutine number 776) and the standard functions of Pascal-SC [15]. The source code file HALSYS.S is created from HALSYS.PROG with DIM set to its correct value, and the program HALSYS compiled with reference to HALLEY_LIB to bring in the code for the system being solved. The text of HALSYS.PROG is given in Appendix E.

The program HALSYS carries out the actual Halley iteration. First, it asks for initial values of X[1].F,...,X[DIM].F, and then prints these and the corresponding function values F[1].F,...,F[DIM].F. The user then receives the query:

(7.2) RESTART (R) OR ITERATE (Y/N)?

The response "R" will result in the request for another set of initial values, "Y" will give the results of one Halley iteration, while "N" will terminate the program and return control to the operating system. After each step of Halley's method, the query (7.2) will be sent to the user. Of course, the user can introduce a more automatic method for controlling the iteration by editing the file HALSYS.S before compilation. Typical results for the system (6.3) are given in Appendix F. It is interesting to compare these with the corresponding results obtained by Newton's method, and given in [13]. In the latter case, eight iterations were required to reduce the function values to zero, as compared to 6 by Halley's method. However, a few iterations are spent in each calculation chasing a small roundoff error in the function values; the function values are actually negligible after 5 iterations of Newton's method and 3 iterations of Halley's method.

8. Implementation details. The programs described in this report were created and tested using the Pascal-SC compiler developed at the University of Karlsruhe for the Zilog MCZ-1 microcomputer using the RIO 2.06 operating system of Zilog, Inc. No other claims of correctness or usability are made.

## References


1. G. Bohlender, K. Gruner, E. Kaucher, R. Klatte, W. Kramer, U. W. Kulisch, S. M. Rump, Ch. Ullrich, J. Wolff von Gudenberg & W. L. Miranker. PASCAL-SC: A PASCAL for Contemporary Scientific Computation Research Report RC 9009, IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y., 1981.

2. Annie A. M. Cuyt. Abstract Pade Approximants for Operators: Theory and Applications. Ph.D. Dissertation, Department of Mathematics University of Antwerp UIA, 1982.

3. Annie A. M. Cuyt. Numerical stability of the Halley-iteration for the solution of a system of nonlinear equations. Math. Comp. 38 (1982), 171-179.

4. Annie Cuyt & Paul van der Cruyssen. Abstract Pade Approximants for the Solution of a System of Nonlinear Equations. Report 80-17, University of Antwerp UIA, 1980.

5. Julia H. Gray & L. B. Rall. NEWTON: A general purpose program for solving nonlinear systems. Proceedings of the 1967 Army Numerical Analysis Conference, pp. 11-59. U. S. Army Research Office, Durham, N.C., 1967.

6. G. Kedem. Automatic differentiation of computer programs. ACM Trans. Math. Software 6, no. 2 (1980), 150-165.

7. Dennis Kuba & L. B. Rall. A UNIVAC 1108 program for obtaining rigorous error estimates for approximate solutions of systems of equations, Technical Summary Report No. 1168, Mathematics Research Center, University of Wisconsin-Madison, 1972.

8. U. Kulisch & W. L. Miranker. Computer Arithemetic in Theory and Practice. Academic Press, New York, 1981.

9. J. M. Ortega & W. C. Rheinboldt. Iterative Solution of Nonlinear Equations in Several Variables. Academic Press, New York, 1970.

10. L. B. Rall. Computational Solution of Nonlinear Operator Equations. Reprinted by Krieger, Huntington, N.Y., 1979.

11. L. B. Rall. Applications of software for automatic differentiation in numerical computation. Computing, Suppl. 2 (1980), 141-156.

12. L. B. Rall. Automatic Differentiation: Techniques and Applications. Lecture Notes in Computer Science No. 120, Springer-Verlag, Berlin-Heidelberg, New York, 1981.

13. L. B. Rall. Differentiation in PASCAL-SC: Type GRADIENT. Technical Summary Report No. 2400, Mathematics Research Center, University of Wisconsin-Madison, 1982.

14. L. B. Rall. Differentiation and Generation of Taylor Coefficients in PASCAL-SC. Technical Summary Report No. 2452, Mathematics Research Center, University of Wisconsin-Madison, 1982.

15. J. Wolff von Gudenberg. Gesamte Arithmetik des PASCAL-SC Rechners: Benutzerhandbuch. Institute for Applied Mathematics, University of Karlsruhe, 1981.

## APPENDIX A

A Pascal-SC program for the solution of system (5.4) by Halley's method.

```
PROGRAM HALLEY(INPUT,OUTPUT);

CONST DIM = 2;

TYPE  DIMTYPE = 1..DIM;
      RVECTOR = ARRAY[DIMTYPE]OF REAL;
      RMATRIX = ARRAY[DIMTYPE]OF RVECTOR;
      HESSIAN = RECORD F:REAL;DF: RVECTOR;HF: RMATRIX END;

   VAR X,Y,F,G: HESSIAN;C: CHAR;V,A,B: RVECTOR;JAC,L,M: RMATRIX;
       NRS: BOOLEAN;

(* The following are standard Pascal-SC matrix and vector functions and
 operators from MR_PAKET. *)

FUNCTION MRNULL: RMATRIX;  (* This returns the zero matrix. *)
 VAR I,J: DIMTYPE;
   C: RMATRIX;
 BEGIN
  FOR I:=1 TO DIM DO
   FOR J:=1 TO DIM DO
    C[I,J] := 0;
  MRNULL := C
 END;

OPERATOR + (A,B: RVECTOR) RES: RVECTOR;
 VAR I: DIMTYPE;
 BEGIN FOR I:=1 TO DIM DO A[I] := A[I]+B[I];
   RES := A
 END;

OPERATOR * (A: REAL; B: RVECTOR) RES: RVECTOR;
 VAR I: DIMTYPE;
 BEGIN FOR I:=1 TO DIM DO B[I] := A*B[I];
   RES := B
 END;

OPERATOR * (A: RMATRIX; B: RVECTOR) RES: RVECTOR;
 VAR I: DIMTYPE;
   BVAR: RVECTOR;
 BEGIN
  BVAR := B;
  FOR I:=1 TO DIM DO
  B[I] := SCALP (A[I],BVAR,0);
  RES := B
 END;
```

```
(* Special operators for componentwise multiplication and division of vectors,
 see (5.17) and (5.18). *)

OPERATOR * (VA,VB: RVECTOR) RES: RVECTOR;
   VAR U: RVECTOR;I: DIMTYPE;
   BEGIN FOR I:=1 TO DIM DO U[I]:=VA[I]*VB[I];
         RES:=U
   END;

OPERATOR / (VA,VB: RVECTOR) RES: RVECTOR;
   VAR U: RVECTOR;I: DIMTYPE;
   BEGIN FOR I:=1 TO DIM DO
         IF (VA[I]=0) AND (VB[I]=0) THEN U[I]:=0
         ELSE U[I]:=VA[I]/VB[I];
         RES:=U
   END;

(* The required HESSIAN operators and function (5.7) for the evaluation of
 the system (5.6) follow. *)

OPERATOR + (HA,HB: HESSIAN) RES: HESSIAN;
   VAR I,J: DIMTYPE;U: HESSIAN;
   BEGIN U.F:=HA.F+HB.F;FOR I:=1 TO DIM DO
     BEGIN U.DF[I]:=HA.DF[I]+HB.DF[I];
           FOR J:=1 TO DIM DO
           U.HF[I][J]:=HA.HF[I][J]+HB.HF[I][J]
     END;
         RES:=U
   END;

OPERATOR - (H: HESSIAN) RES: HESSIAN;
   VAR I,J: DIMTYPE;U: HESSIAN;
   BEGIN U.F:=-H.F;FOR I:=1 TO DIM DO
        BEGIN U.DF[I]:=-H.DF[I];
              FOR J:=1 TO DIM DO
              U.HF[I][J]:=-H.HF[I][J]
        END;
     RES:=U
   END;

OPERATOR - (H: HESSIAN;R: REAL) RES: HESSIAN;
   VAR U: HESSIAN;
   BEGIN U.F:=H.F-R;U.DF:=H.DF;U.HF:=H.HF;
         RES:=U
   END;
```

```
OPERATOR - (HA,HB: HESSIAN) RES: HESSIAN;
   VAR I,J: DIMTYPE;U: HESSIAN;
   BEGIN U.F:=HA.F-HB.F;FOR I:=1 TO DIM DO
        BEGIN U.DF[I]:=HA.DF[I]-HB.DF[I];
              FOR J:=1 TO DIM DO
              U.HF[I][J]:=HA.HF[I][J]-HB.HF[I][J]
        END;
     RES:=U
   END;



FUNCTION HEXP(H: HESSIAN): HESSIAN;
   VAR I,J: DIMTYPE;U: HESSIAN;
   BEGIN U.F:=EXP(H.F);
     FOR I:=1 TO DIM DO
        BEGIN U.DF[I]:=U.F*H.DF[I];
          FOR J:=1 TO I DO
             BEGIN U.HF[I][J]:=U.F*H.HF[I][J]+U.DF[I]*H.DF[J];
                   IF I<>J THEN U.HF[J][I]:=U.HF[I][J]
             END;
        END;
     HEXP:=U
   END;

(* The next procedure solves a linear system of equations with coefficient
 matrix JAC and right-hand side V for the solution vector S.  The decomposi-
 tion of JAC is stored as the matrix M.  If NRS = FALSE, then additional
 right-hand sides are expected.  Pre-translated code for this procedure is
 stored in the library HALLEY_LIB as subroutine number 776. *)

PROCEDURE SOLVLN(DIM:INTEGER;VAR JAC,M:RMATRIX;VAR V,S: RVECTOR;NRS:BOOLEAN);
   EXTERNAL 776;

BEGIN (* Initialization of gradients and Hessians of independent variables. *)

      X.DF[1]:=1;X.DF[2]:=0;Y.DF[1]:=0;Y.DF[2]:=1;
      X.HF:=MRNULL;Y.HF:=MRNULL;
      C:='R';WHILE C = 'R' DO

  BEGIN  (* VALUE INITIALIZATION *)

      WRITELN('ENTER X,Y');READ(X.F,Y.F);
      C:='Q'; WHILE C = 'Q' DO

   BEGIN  (* MAIN PROGRAM *)

        (* Calculate function values and derivatives.  Print values of
         independent and dependent variables. *)

     F:=HEXP(-X+Y)-0.1;
     G:=HEXP(-X-Y)-0.1;
```

```
      WRITELN('(X,Y) = (',X.F,' , ',Y.F,')');
      WRITELN('(F,G) = (',F.F,' , ',G.F,')');

      WRITELN('RESTART (R) OR ITERATE (Y/N)?');READ(C,C);

      (* In response to "R", the program will ask for new initial values of
       the independent variables; an input of "Y" will result in one Halley
       iteration being performed, while "N" will terminate the program and
       return control to the operating system.  Two characters are read,
       since the first read from the console will always be a blank, the
       second being the character entered by the user in response to the
       prompt "*". *)


    WHILE C = 'Y' DO

      BEGIN (* HALLEY ITERATION *)

      (* Construct Jacobian matrix JAC and right-hand side V of (2.2). *)

        JAC[1]:=F.DF;JAC[2]:=G.DF;V[1]:=-F.F;V[2]:=-G.F;

        (* Solve (2.2) for A. *)

        NRS:=FALSE;SOLVLN(DIM,JAC,M,V,A,NRS);

        (* Construct the right-hand side V of (2.3). *)

        L[1]:=F.HF*A;L[2]:=G.HF*A;V:=L*A;

        (* Solve (2.3) for B. *)

        NRS:=TRUE;SOLVLN(DIM,JAC,M,V,B,NRS);

     (* Compute the Halley correction V and update independent variables. *)

        V:=A*A/(A+0.5*B);
        X.F:=X.F+V[1];Y.F:=Y.F+V[2];
        C:='Q'

      END;  (* HALLEY ITERATION *)

  END;  (* MAIN PROGRAM *)

 END  (* VALUE INITIALIZATION *)

END.
```

## APPENDIX B

Output of the program in Appendix A for the initial values X.F = 4.3, Y.F = 2.0.

```
INITIAL VALUES

(X,Y) = ( 4.30000000000E+00 ,  2.00000000000E+00)
(F,G) = ( 2.58843723000E-04 , -9.81636952230E-02)

RESULTS OF ITERATION NUMBER 1

(X,Y) = ( 3.33615528246E+00 ,  1.03597241993E+00)
(F,G) = ( 2.40511813000E-04 , -8.73756488903E-02)

RESULTS OF ITERATION NUMBER 2

(X,Y) = ( 2.56081800937E+00 ,  2.59679794972E-01)
(F,G) = ( 1.44792583000E-04 , -4.04237220038E-02)

RESULTS OF ITERATION NUMBER 3

(X,Y) = ( 2.30817563469E+00 ,  5.68378530700E-03)
(F,G) = ( 9.32479600000E-06 , -1.12110099570E-03)

RESULTS OF ITERATION NUMBER 4

(X,Y) = ( 2.30258515119E+00 ,  6.12025800000E-08)
(F,G) = ( 3.00000000000E-10 , -1.19396000000E-08)

RESULTS OF ITERATION NUMBER 5

(X,Y) = ( 2.30258509299E+00 ,  4.57643840C00E-12)
(F,G) = ( 0.00000000000E+00 ,  0.00000000000E+00)
```

## APPENDIX C

### The program SYSTEST for testing correctness of HESSIAN systems of equations.

The source code for this program is in the file SYSTEST.PROG.

```
PROGRAM SYSTEST(INPUT,OUTPUT);

CONST DIM = #;   (* Replace "#" by the dimension of the system tested. *)

TYPE  DIMTYPE = 1..DIM;
      RVECTOR = ARRAY[DIMTYPE]OF REAL;
      RMATRIX = ARRAY[DIMTYPE]OF RVECTOR;
      HESSIAN = RECORD F:REAL;DF: RVECTOR;HF: RMATRIX END;
       SYSTEM = ARRAY[DIMTYPE]OF HESSIAN;

 VAR  X,F: SYSTEM;JAC: RMATRIX;I,J,K: DIMTYPE;C: CHAR;

PROCEDURE FCOMP(VAR X,F: SYSTEM;DIM: DIMTYPE);

(* Insert source code for the operators, functions, and procedures required
 for computation of the system being tested here, for example, the source code
 for the operators (7.1) in the case of the system (6.4).  Source code for
 HESSIAN operators and functions is in the file HESS_PAKET. *)

 BEGIN (* SYSTEM DEFINITION *)

(* Insert code defining the system to be tested here, for example, the system
 (6.4):

      F[1]:=16*(X[1]**4)+16*(X[2]**4)+X[3]**4-16;
      F[2]:=X[1]**2+X[2]**2+X[3]**2-3;
      F[3]:=X[1]**3-X[2]; *)

 END;  (* SYSTEM DEFINITION *)

(* The following are standard Pascal-SC functions from MR_PAKET: MRNULL
returns the zero matrix, and MRID returns the identity matrix.  These are
used for initialization of the Hessian and gradient parts of the
independent variables, respectively. *)
```

```
FUNCTION MRNULL: RMATRIX;
 VAR I,J: DIMTYPE;
   C: RMATRIX;
 BEGIN
  FOR I:=1 TO DIM DO
   FOR J:=1 TO DIM DO
    C[I,J] := 0;
  MRNULL := C
 END;

FUNCTION MRID: RMATRIX;
 VAR I,J: DIMTYPE;
   C: RMATRIX;
 BEGIN
  FOR I:=1 TO DIM DO
   FOR J:=1 TO DIM DO
    IF I=J THEN C[I,J] := 1
    ELSE C[I,J] := 0;
  MRID := C
 END;

BEGIN (* Initialization of gradients and Hessians of independent variables. *)

      JAC:=MRID;FOR I:=1 TO DIM DO

      BEGIN X[I].DF:=JAC[I];X[I].HF:=MRNULL

      END;

      C:='Y';WHILE C = 'Y' DO

   BEGIN (* MAIN PROGRAM *)

     (* Input Portion *)

        WRITELN('ENTER INDEPENDENT VARIABLES');
        FOR I:=1 TO DIM DO READ(X[I].F);

      (* Calculation of function values and derivatives. *)

        FCOMP(X,F,DIM);WRITELN(' ');

      (* Output Portion *)

        FOR I:=1 TO DIM DO
        BEGIN WRITELN('FUNCTION VALUE:'); WRITELN('F[',I:2,'] = ',F[I].F);
              WRITELN('FIRST DERIVATIVES:');FOR J:= 1 TO DIM DO
              WRITELN('DF[',I:2,']/DX[',J:2,'] = ',F[I].DF[J]);
              WRITELN('SECOND DERIVATIVES:');FOR J:=1 TO DIM DO
              FOR K:=1 TO J DO
              WRITELN('D2F[',I:2,']/DX[',J:2,']DX[',K:2,'] = ',F[I].HF[J,K])
        END;  (* Output Portion *)
```

```
        WRITELN('DO YOU WANT TO ENTER MORE VALUES (Y/N)?');READ(C,C)

   END; (* MAIN PROGRAM *)

END.
```

EXAMPLE: Source code for the HESSIAN operators (7.1) for the system (6.4).

```
OPERATOR + (HA,HB: HESSIAN) RES: HESSIAN;
   VAR I,J: DIMTYPE;U: HESSIAN;
   BEGIN U.F:=HA.F+HB.F;FOR I:=1 TO DIM DO
     BEGIN U.DF[I]:=HA.DF[I]+HB.DF[I];
           FOR J:=1 TO DIM DO
           U.HF[I,J]:=HA.HF[I,J]+HB.HF[I,J]
     END;
         RES:=U
   END;

OPERATOR - (H: HESSIAN;K: INTEGER) RES: HESSIAN;
   VAR U: HESSIAN;
   BEGIN U.F:=H.F-K;U.DF:=H.DF;U.HF:=H.HF;
         RES:=U
   END;

OPERATOR - (HA,HB: HESSIAN) RES: HESSIAN;
   VAR I,J: DIMTYPE;U: HESSIAN;
   BEGIN U.F:=HA.F-HB.F;FOR I:=1 TO DIM DO
        BEGIN U.DF[I]:=HA.DF[I]-HB.DF[I];
              FOR J:=1 TO DIM DO
              U.HF[I,J]:=HA.HF[I,J]-HB.HF[I,J]
        END;
     RES:=U
   END;

OPERATOR * (K: INTEGER;H: HESSIAN) RES: HESSIAN;
   VAR I,J: DIMTYPE;U: HESSIAN;
   BEGIN U.F:=K*H.F;FOR I:=1 TO DIM DO
        BEGIN U.DF[I]:=K*H.DF[I];
              FOR J:=1 TO DIM DO
              U.HF[I,J]:=K*H.HF[I,J]
        END;
     RES:=U
   END;
```

```
OPERATOR ** (R: REAL;K: INTEGER) RES: REAL;
   VAR L: INTEGER;U: REAL;
   BEGIN IF K <= 0 THEN U:=1/R;
         IF (K = 0) OR (R = 1) THEN U:=1
         ELSE IF K = 1 THEN U:=R
              ELSE BEGIN L:=ABS(K);U:=1;REPEAT IF L MOD 2 = 1
                           THEN U:=R*U;L:= L DIV 2; IF L <> 0
                           THEN R:=R*R UNTIL L = 0;
                           IF K < 0 THEN U:=1/U
                   END;
      RES:=U
   END;

OPERATOR ** (H: HESSIAN;K: INTEGER) RES: HESSIAN;
   VAR I,J: DIMTYPE;M,MM:REAL;U: HESSIAN;
   BEGIN
     IF K = 0 THEN
     BEGIN IF H.F = 0 THEN
           BEGIN WRITELN('EXPONENTIATION ERROR 0**0');SVR(0)
           END
           ELSE BEGIN U.F:=1;FOR I:=1 TO DIM DO
                      BEGIN U.DF[I]:=0;FOR J:=1 TO DIM DO U.HF[I,J]:=0
                      END;
                END;
     END
     ELSE IF K = 1 THEN U:=H
     ELSE IF K = 2 THEN
          BEGIN U.F:=H.F*H.F;M:=2*H.F;FOR I:=1 TO DIM DO
                BEGIN U.DF[I]:=M*H.DF[I];FOR J:=1 TO I DO
                      BEGIN U.HF[I,J]:=M*H.HF[I,J]+2*H.DF[I]*H.DF[J];
                            IF I <> J THEN U.HF[J,I]:=U.HF[I,J]
                      END;
                END
          END
     ELSE BEGIN MM:=H.F**(K-2);M:=H.F*MM;U.F:=H.F*M;
                M:=K*M;MM:=K*(K-1)*MM;FOR I:=1 TO DIM DO
                BEGIN U.DF[I]:=M*H.DF[I];FOR J:=1 TO I DO
                      BEGIN U.HF[I,J]:=M*H.HF[I,J]+MM*H.DF[I]*H.DF[J];
                            IF I <> J THEN U.HF[J,I]:=U.HF[I,J]
                      END;
                END;
          END;
      RES:=U
   END;
```

## APPENDIX D

Typical output of the program SYSTEST.

Values computed for the system (6.4) with X[1] = X[2] = X[3] = 1.0.

```
FUNCTION VALUE:
F[ 1] =  1.70000000000E+01
FIRST DERIVATIVES:
DF[ 1]/DX[ 1] =  6.40000000000E+01
DF[ 1]/DX[ 2] =  6.40000000000E+01
DF[ 1]/DX[ 3] =  4.00000000000E+00
SECOND DERIVATIVES:
D2F[ 1]/DX[ 1]DX[ 1] =  1.92000000000E+02
D2F[ 1]/DX[ 2]DX[ 1] =  0.00000000000E+00
D2F[ 1]/DX[ 2]DX[ 2] =  1.92000000000E+02
D2F[ 1]/DX[ 3]DX[ 1] =  0.00000000000E+00
D2F[ 1]/DX[ 3]DX[ 2] =  0.00000000000E+00
D2F[ 1]/DX[ 3]DX[ 3] =  1.20000000000E+01
FUNCTION VALUE:
F[ 2] =  0.00000000000E+00
FIRST DERIVATIVES:
DF[ 2]/DX[ 1] =  2.00000000000E+00
DF[ 2]/DX[ 2] =  2.00000000000E+00
DF[ 2]/DX[ 3] =  2.00000000000E+00
SECOND DERIVATIVES:
D2F[ 2]/DX[ 1]DX[ 1] =  2.00000000000E+00
D2F[ 2]/DX[ 2]DX[ 1] =  0.00000000000E+00
D2F[ 2]/DX[ 2]DX[ 2] =  2.00000000000E+00
D2F[ 2]/DX[ 3]DX[ 1] =  0.00000000000E+00
D2F[ 2]/DX[ 3]DX[ 2] =  0.00000000000E+00
D2F[ 2]/DX[ 3]DX[ 3] =  2.00000000000E+00
FUNCTION VALUE:
F[ 3] =  0.00000000000E+00
FIRST DERIVATIVES:
DF[ 3]/DX[ 1] =  3.00000000000E+00
DF[ 3]/DX[ 2] = -1.00000000000E+00
DF[ 3]/DX[ 3] =  0.00000000000E+00
SECOND DERIVATIVES:
D2F[ 3]/DX[ 1]DX[ 1] =  6.00000000000E+00
D2F[ 3]/DX[ 2]DX[ 1] =  0.00000000000E+00
D2F[ 3]/DX[ 2]DX[ 2] =  0.00000000000E+00
D2F[ 3]/DX[ 3]DX[ 1] =  0.00000000000E+00
D2F[ 3]/DX[ 3]DX[ 2] =  0.00000000000E+00
D2F[ 3]/DX[ 3]DX[ 3] =  0.00000000000E+00
```

## APPENDIX E

The Pascal-SC program HALSYS for the solution of systems of equations by Halley's method.

Source code for this program is contained in the file HALSYS.PROG.

```
PROGRAM HALSYS(INPUT,OUTPUT);

CONST DIM = #;  (* Replace "#" by the dimension of the system to be solved. *)

TYPE  DIMTYPE = 1..DIM;
      RVECTOR = ARRAY[DIMTYPE]OF REAL;
      RMATRIX = ARRAY[DIMTYPE]OF RVECTOR;
      HESSIAN = RECORD F:REAL;DF: RVECTOR;HF: RMATRIX END;
       SYSTEM = ARRAY[DIMTYPE]OF HESSIAN;

 VAR  I: DIMTYPE;X,F: SYSTEM;C: CHAR, A,B,V: RVECTOR;JAC,L,M: RMATRIX;
      NRS:BOOLEAN;


OPERATOR + (A,B: RVECTOR) RES: RVECTOR;
 VAR I: DIMTYPE;
 BEGIN FOR I:=1 TO DIM DO A[I] := A[I]+B[I];
   RES := A
 END;

OPERATOR * (A: REAL; B: RVECTOR) RES: RVECTOR;
 VAR I: DIMTYPE;
 BEGIN FOR I:=1 TO DIM DO B[I] := A*B[I];
   RES := B
 END;

OPERATOR * (A: RMATRIX; B: RVECTOR) RES: RVECTOR;
 VAR I: DIMTYPE;
   BVAR: RVECTOR;
 BEGIN
  BVAR := B;
  FOR I:=1 TO DIM DO
  B[I] := SCALP (A[I],BVAR,0);
  RES := B
 END;

OPERATOR * (A,B: RVECTOR) RES: RVECTOR;
   VAR I: DIMTYPE;C: RVECTOR;
   BEGIN FOR I:=1 TO DIM DO C[I]:=A[I]*B[I];
         RES:=C
   END;
```

```
OPERATOR / (A,B: RVECTOR) RES: RVECTOR;
   VAR I: DIMTYPE;C: RVECTOR;
   BEGIN FOR I:= 1 TO DIM DO
         IF (A[I]=0) AND (B[I]=0) THEN C[I]:=0
         ELSE C[I]:=A[I]/B[I];
         RES:=C
   END;

FUNCTION MRID: RMATRIX;  (* Returns the identity matrix *)
 VAR I,J: DIMTYPE;
   C: RMATRIX;
 BEGIN
  FOR I:=1 TO DIM DO
   FOR J:=1 TO DIM DO
    IF I=J THEN C[I,J] := 1
    ELSE C[I,J] := 0;
  MRID := C
 END;

FUNCTION MRNULL: RMATRIX;  (* Returns the zero matrix *)
 VAR I,J: DIMTYPE;
   C: RMATRIX;
 BEGIN
  FOR I:=1 TO DIM DO
   FOR J:=1 TO DIM DO
    C[I,J] := 0;
  MRNULL := C
 END;

PROCEDURE SOLVLN(DIM:INTEGER;JAC,M:RMATRIX;V,S:RVECTOR;NRS:BOOLEAN);
   EXTERNAL 776;

PROCEDURE FCOMP(VAR X,F: SYSTEM;DIM: DIMTYPE);
   EXTERNAL 777;

BEGIN (* Initialization of gradients and Hessians of independent variables. *)

      L:=MRID;FOR I:=1 TO DIM DO
      BEGIN X[I].DF:=L[I];X[I].HF:=MRNULL
      END;
      C:='R';WHILE C = 'R' DO

 BEGIN (* VALUE INITIALIZATION *)

      WRITELN('ENTER INDEPENDENT VARIABLES');
      FOR I:=1 TO DIM DO READ(X[I].F);
      C:='Q'; WHILE C = 'Q' DO
```

```
BEGIN  (* MAIN PROGRAM *)

   (* Compute function values and derivatives; print values of independent
    and dependent variables. *)

   FCOMP(X,F,DIM);

   FOR I:=1 TO DIM DO
   BEGIN WRITELN('X[',I:2,'] = ',X[I].F,' F[',I:2,'] = ',F[I].F);
   END;

   WRITELN('RESTART (R) OR ITERATE (Y/N)?');
   READ(C,C);WHILE C = 'Y' DO

   BEGIN (* HALLEY ITERATION *)

         FOR I:=1 TO DIM DO
         BEGIN JAC[I]:=F[I].DF; (* JACOBIAN MATRIX *)
               V[I]:=-F[I].F; (* RIGHT HAND SIDE *)
         END;

        (* Solve for A. *)

         NRS:=FALSE;SOLVLN(DIM,JAC,M,V,A,NRS);

        (* Compute B. *)

         FOR I:=1 TO DIM DO L[I]:=F[I].HF*A;V:=L*A;
         NRS:=TRUE;SOLVLN(DIM,JAC,M,V,B,NRS);

     (* Compute the Halley correction and update independent variables. *)

         V:=A*A/(A+0.5*B);FOR I:=1 TO DIM DO X[I].F:=X[I].F+V[I];

         C:='Q'

   END;  (* HALLEY ITERATION *)

  END;  (* MAIN PROGRAM *)

 END  (* VALUE INITIALIZATION *)

END.
```

## APPENDIX F

### Results of the Solution of the System (6.3) by the Program HALSYS.

INITIAL VALUES

X[ 1] = 1.00000000000E+00 F[ 1] = 1.70000000000E+01
X[ 2] = 1.00000000000E+00 F[ 2] = 0.00000000000E+00
X[ 3] = 1.00000000000E+00 F[ 3] = 0.00000000000E+00

RESULTS OF ITERATION NUMBER 1

X[ 1] = 8.91118701964E-01 F[ 1] = 9.37521623100E-01
X[ 2] = 7.05429341548E-01 F[ 2] = -9.44922445000E-03
X[ 3] = 1.30339083879E+00 F[ 3] = 2.20137281800E-03

RESULTS OF ITERATION NUMBER 2

X[ 1] = 8.77982528233E-01 F[ 1] = 1.03685680000E-03
X[ 2] = 6.76786689302E-01 F[ 2] = -9.09324000000E-06
X[ 3] = 1.33082582033E+00 F[ 3] = 9.05738500000E-06

RESULTS OF ITERATION NUMBER 3

X[ 1] = 8.77965760275E-01 F[ 1] = 1.00000000000E-10
X[ 2] = 6.76756970519E-01 F[ 2] = 0.00000000000E+00
X[ 3] = 1.33085541162E+00 F[ 3] = 0.00000000000E+00

RESULTS OF ITERATION NUMBER 4

X[ 1] = 8.77965760274E-01 F[ 1] = 0.00000000000E+00
X[ 2] = 6.76756970516E-01 F[ 2] = 0.00000000000E+00
X[ 3] = 1.33085541162E+00 F[ 3] = 2.00000000000E-12

RESULTS OF ITERATION NUMBER 5

X[ 1] = 8.77965760274E-01 F[ 1] = 0.00000000000E+00
X[ 2] = 6.76756970517E-01 F[ 2] = 0.00000000000E+00
X[ 3] = 1.33085541162E+00 F[ 3] = 1.00000000000E-12

RESULTS OF ITERATION NUMBER 6

X[ 1] = 8.77965760274E-01 F[ 1] = 0.00000000000E+00
X[ 2] = 6.76756970518E-01 F[ 2] = 0.00000000000E+00
X[ 3] = 1.33085541162E+00 F[ 3] = 0.00000000000E+00

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER<br>#2481 | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle)<br>COMPUTATIONAL IMPLEMENTATION OF THE MULTIVARIATE HALLEY METHOD FOR SOLVING NONLINEAR SYSTEMS OF EQUATIONS | | 5. TYPE OF REPORT & PERIOD COVERED<br>Summary Report - no specific reporting period<br>6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s)<br>Annie A. M. Cuyt and L. B. Rall | | 8. CONTRACT OR GRANT NUMBER(s)<br>DAAG29-80-C-0041 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>Mathematics Research Center, University of Wisconsin<br>610 Walnut Street<br>Madison, Wisconsin 53706 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS<br>Work Unit Number 3 - Numerical Analysis |
| 11. CONTROLLING OFFICE NAME AND ADDRESS<br>U. S. Army Research Office<br>P.O. Box 12211<br>Research Triangle Park, North Carolina 27709 | | 12. REPORT DATE<br>February 1983<br>13. NUMBER OF PAGES<br>30 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) | | 15. SECURITY CLASS. (of this report)<br>UNCLASSIFIED<br>15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Nonlinear systems of equations, Halley's method, automatic differentiation, type HESSIAN, Pascal-SC

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Halley's method for the solution of systems of equations is an iterative procedure which converges cubically under favorable conditions. The multivariate version requires the solution of two linear systems of equations with the same coefficient matrix, following which the correction vector is computed using componentwise multiplication and division of vectors. This report describes a general-purpose computer program which implements this method. The necessary first and second derivatives are obtained by automatic differentiation, so the user need only suppy code defining the functions appearing in the system of equations.

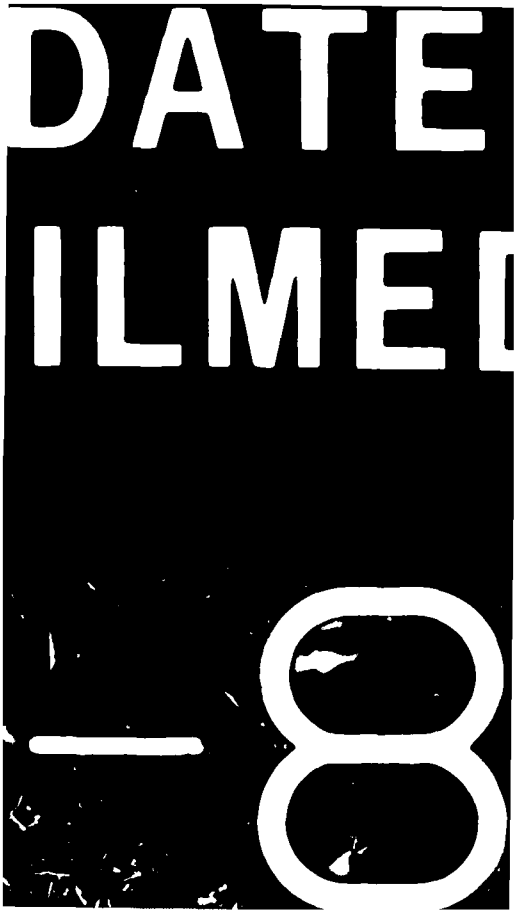
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT (continued)

The program is written in Pascal-SC, using the new data type HESSIAN to represent dependent and independent variables. Numerical examples are given for two simple systems of equations to illustrate the use of the program and the effectiveness of the method.


DATE
ILME
8