1967

# Structure of a Language for a Numerical Analysis Problem Solving Systems

Lawrence R. Symes

Roger V. Roman

Report Number:
67-012

# STRUCTURE OF A LANGUAGE FOR A NUMERICAL ANALYSIS PROBLEM SOLVING SYSTEM

Lawrence R. Symes (Senior Author)
Roger V. Roman
Computer Science Center
Purdue University
Lafayette, Indiana

CSD TR 12

## ABSTRACT

A general description is given of the NAPSS language presently being implemented at Purdue University. The NAPSS language is an interactive problem-oriented language, designed primarily for stating numerical problems in a mathematical like notation. It permits the direct manipulation of arrays and function and also includes several built-in higher level numerical operations which may appear anywhere in arithmetic statements. The method of solution of common numerical problems is provided automatically by the system. NAPSS may also be used as a procedural language, allowing all the flexibility of procedural languages, but eliminating unnecessary clerical operations.

## INTRODUCTION:

A project has been undertaken at Purdue University to design and construct an interactive system for solving numerical problems [1]. The system has been designed to accept input in a language which is very close to natural mathematical notation and also to provide for the solution of problems without requiring specially trained programmers and numerical analysts.

The use of polyalgorithms [2] reduces the amount of problem analysis needed before the problem is presented to the system, the language reduces the amount of time required to transform the problem into a form acceptable by the system, and the interaction between the user and the system along with incremental execution increases the users control over the solution of his problem [3].

This paper is primarily concerned with the structure of the Numerical Analysis Problem Solving System (NAPSS) language.

## AIMS OF THE LANGUAGE

The NAPSS language attempts to eliminate many of the unnecessary rules and restrictions that are in other languages, while at the same time not imposing on the languages' flexibility.

The NAPSS language permits the manipulation of vectors, arrays and functions with the same ease that procedural languages, FORTRAN, are able to manipulate scalars.

Strictly clerical statements, such as those used for declaring array dimensions, and variable type or mode are not required in NAPSS, but can be included if the user desires.

The arithmetic expression in NAPSS is more general than that of procedural languages, permitting the appearance of any number of the built-in numerical operators ($\int$ integration, ' transpose, and ' differentiation) in addition to the five basic arithmetic operations (+,-,*,/,↑).

NAPSS, by permitting the manipulation of entities larger than scalars, not requiring declarations, and allowing the direct use of numerical operations in arithmetic expressions, allows a user to present his problem to the computer in a form which is very close to its natural mathematical form.

NAPSS also permits the automatic solution of numerical problems. The user need only supply to the system a description of the problem to be solved and the variables to be solved. The system by means of polyalgorithms selects the method to be used. For example to have the system solve the second degree initial value problem

$$y''(x) + y'(x) + e^{x}y(x) = \sin 2x,\ 0<x<2,\ y'(0) = 1,\ y(0) = 0,$$

a user would simply say:

```
SOLVE  y''(x) + P(x)y'(x) + r(x)y(x) = g(x),
 FOR   y(x),WITH y'(x←0) ← 1, y(x←0) ← 0,
       P(x) ← 1, r(x) ← EXP(x), g(x) ← SIN(2x),
  ON   0 < x < 2;
```

LANGUAGE

Rather than present a detailed description of the NAPSS language [4], we will describe the unique features by examining several of the more important statements in the language and by an example program.

ARITHMETIC EXPRESSIONS

NAPSS contains the five basic arithmetic operators (+,-,*,/,↑) in addition to several built-in numerical operators.

The * may be omitted in cases where no ambiguity results.

2ZA is equivalent to 2*ZA

C1 2D is equivalent to C1*2*D

X↑2Y is equivalent to (X↑2)*Y

In the second example above the blank between the C1 and the 2 is significant as a variable name may be composed of more than one letter or digit, the first being a letter.

The built-in numerical operators are: | | absolute values, ' derivative of function of one variable, ' transpose of vectors and 2-dimension matrices, DER partial differentiation, $\int$ Riemann integration.

For example:

i) | | absolute value

| X-Y | denotes the absolute value of the arithmetic expression X-Y. If X and Y are not scalars, the absolute value of each element of the vector or matrix resulting from X-Y is taken.

ii) ' - differentiation

f'(X) denotes $\frac{df(X)}{dX}$ while

f'"(3.5) denotes $\left.\frac{d^3f(X)}{d^3X}\right|_{X = 3.5}$

iii) ' - transpose

A' denotes the transpose of the vector or matrix A, while B[i,*]' denotes the transpose of the ith row of the matrix B, i.e. it creates a column vector.

A "*" in place of a subscript specifies that the *'ed (starred) subscript varies over its entire range and thus can be used to obtain subarrays.

iv) DER - partial differentiation

DER (g(x,y) / (x↑2,y)) is the linear notation for

$$\frac{\partial^3 g(x,y)}{\partial^2 x \partial y}.$$

DER ((x↑3+ay+g(x)) / (x,y) | x←2, y←4)

denotes

$$\left.\frac{\partial^2 (x^3+a*y+g(x))}{\partial x \partial y}\right|_{\substack{x=2\\y=4}}$$

v) ∫ - integration

∫ t / (1+t↑2), (t←0 TO 2) denotes $\int_0^2 \frac{t}{1+t^2}\,dt$

∫∫ f(x,y), (x←0 TO y), (y←0 TO 1) denotes

$$\int_0^1 \int_0^y f(x,y)\,dx\,dy.$$

In addition to operations on components, arithmetic can be performed on arrays.

A+B is the sum of the arrays A and B, (they must be of the same size).

A*B or A B is the matrix product, (A and B must be conformable).

A↑-1 is the inverse of A (A must be square and non-singular).

5A causes each element of the matrix A to be multiplied by 5.

(1,2,...,6)' (I+4 for I 7 TO 12) is an example of the multiplication of two explicitly declared vectors. All explicitly

declared vectors are considered to be column vectors initially. The first vector contains the integers 1 through 6 while the second contains the integers 11 through 16.

As was seen when looking at the differentiation operator and integration operator, functions in addition to arrays can be manipulated in arithmetic expressions.

ASSIGNMENT STATEMENT

There are two types of assignment statements in NAPSS.

The first type is the usual procedural language, Fortran or Algol, type of assignment statement where the name on the left assumes the value of the expression on the right. The name on the left is separated from the arithmetic expression on the right in NAPSS by ←.

Declaration statements can be omitted in NAPSS because when a variable appears on the left of an assignment statement it obtains its attributes from the expression on the right and its own mode of use.

examples

1 a ← 5

a ← a B1

If B1 is a 5 x 4 complex matrix, a in line 2 will be redefined to be a 5 by 4 complex matrix whose indices range over the same values as the indices of B1.

```
2  f(x) ← x↑3+cos(x), (-2 < x < 10)
   h(v,w) ← v↑2w,(-10 < v < 4 AND w > 0)
   k ← 6
   g(y) ← f(y) + 5 h(y,k)
```

Line 4 defines g(y) to be equal to the function $y^3+\cos y+30y^2$ on the interval (-2,4). Since no explicit domain is defined for g, it is the intersection of the domains of f and h.

As can be seen from the above example the arguments of a function are simply place-markers, also since k is not an argument of the function g, the current value of k,6, is used in line 4.

```
3  f(x,y) ← x↑2y, (x↑2+y↑2 < 4 OR 4< = x< = 5 AND 0 < y < = 1)
       ← x↑2+y↑3, (y>=2+x↑2) ← x↑3+5y↑3
```

The above is equivalent to

$$f(x,y) = \begin{cases} x^2y, & x^2+y^2 < 4 \\ x^2y, & 4 \leq x \leq 5 \text{ AND } 0 < y \leq 1 \\ x^2+y^3, & y \geq 2 + x^2 \\ x^3 + 5y^3, & \text{elsewhere} \end{cases}$$

```
4  A[1,*] ← (1,2, ..., n)
   A[2,*] ← A[1,*] + (n FOR n TIMES)
   A[3,*] ← (3n↓I FOR I ← 1 TO n)
```

The above defines A to be the n by 3 matrix

$$\begin{bmatrix} 1 & n+1 & 2n+1 \\ 2 & n+2 & 2n+2 \\ & \cdots & \\ n & 2n & 3n \end{bmatrix}$$

and it also exhibits some additional methods permitted in NAPSS for generating vectors.

The second type of assignment statement separates the variable name on the left from the arithmetic expression on the right with an =. This causes the variable on the right to be set symbolically equivalent to the following expression instead of being assigned the value of the expression. Thus the variable names in the expression on the right do not have their values substituted for them when the assignment statement is executed. Values are substituted for the variable names in the expression only when the numeric value of the variable on the left is needed, i.e., when it appears in an expression to the right of an ← or in an output statement for example.

example

```
x ← 4
y = 2x
z ← 2x
x ← 5
w ← z
v ← y
```

The result is v = 10 and w = 8.

When = is used the arithmetic expression on the right must not contain the variable name appearing to the left of the =, nor may any of the variable names appearing in the arithmetic expression be symbolically equivalent to an arithmetic expression containing variable name:

example

i) N = N+1

ii) A = B+C

B = X+A

Both of the above are illegal, and would result in an error message.

More than one variable name, with accompanying ← or =, appearing to the left of an arithmetic expression is equivalent to writing a series of assignment statements.

example

X ← B = A ← 2X↑3 - 4

is equivalent to

X ← 2X↑3 - 4

B = 2X↑3 - 4

A ← 2X↑3 - 4

NAPSS permits symbolic definitions of arrays in terms of their indices. This type of assignment is similar to an assignment statement defining a function except that the name of the array can appear in an arithmetic expression and an array with dimensions corresponding to that with which it is being combined, is generated.

examples

```
A[I,J] = 1/(I+J)
B ← A+C
```

A denotes the Hilbert matrix.
In line 2 if C were a 3 by 3
matrix, the 3 by 3 Hilbert
Matrix C would be added to C and
the result stored in the matrix B.

EQUATIONS

A NAPSS equation consists of two arithmetic expressions separated by an =.

Equations can be labeled for future reference. An equation label consists of a variable name, a period, followed optionally by an integer. When an equation has been labeled, the use of the label is equivalent to writing the equation.

The association of a label with an equation is similar to an assignment statement. The assignment of an equation to a label is done at execution time. The same label may be assigned different equations at various times and denotes the last equation assigned to it.

examples

```
EQ 1.1: 2SIN(X) = A X-2X↑2

EQ 1.2: aX↑2 + b*X + C = 0

        . . .

EQ 1.1: 2CØS(X) = A X - 2X↑2
```

The colon is used to separate the label from the equation.

SOLVE STATEMENT

The SOLVE statement is one of the most important features of NAPSS since it allows the automatic solution of numerical problems. The user need only supply to the system a description of the problem to be solved and the variables to be solved. The statement has the form:

```
SOLVE EQS., FOR VARS. (OPTIONS);
```

where EQS. represents one or more equations or equation labels, VARS. indicates the variables or variables to be solved, and (OPTIONS) represents a list of optional information which may or may not be present. If more than one solution is possible they come back in an array.

example

SOLVE X↑2 - 4 = 0, FOR X ;

will set X[1] to 2 and X[2] to -2.

While details may be left to the system, the user can exercise some control over the solution by providing additional information of the following types (OPTIONS):

1 WITH indicates values to be assigned to variables in the equations. If absent, current values of the variables will be used.

2 ON indicates that only solutions falling within the specified range are desired. If absent, any solution is accepted.

3 NUMBER indicates the maximum number of solutions desired. If absent, the system looks for all possible solutions in the desired range.

4 USING indicates a particular method is to be used. If absent, the system selects a method or methods for solving the given problem by means of the polyalgorithms which use intermediate results to decide on which method to use in the current situation.

5 TYPE indicates the type of equations to be solved (e.g. linear system, polynomial, differential equation). If absent, the system determines the type (so this serves merely to speed computation).

6 ACCURACY indicates the absolute or relative accuracy desired in the solution. If absent, then either the accuracy specified by an accuracy statement (if present), or the standard system accuracy is used.

7 STEP indicates the initial step size to be used (when meaningful). If absent, the initial step size is determined by the accuracy desired.

examples

1 SOLVE TAN(X) = 2X-a, FOR X, WITH a ← PI, ON 0 < X < PI;

This finds the unique solution of

tan X - 2 X+π = 0 on the interval (0,π).

2 EQ.1 : X↑2 + Y↑2 = 4

EQ.2 : X = (Y-1.5)↑2

SOLVE EQ.1, EQ.2, FOR X,Y, ON

0 < X AND 0 < Y, TYPE

POLYNOMIAL SYSTEM;

This finds all solutions of the system:

$$X^2 + Y^2 = 4$$

$$X = (Y-1.5)^2$$

which fall in the first quadrant. If NUMBER 1 were used, only one solution would be obtained.

```
SOLVE A X = LAMDA X, FOR LAMDA, X,
  WITH  A[1,*] ← (-1,0,0),
        A[2,*] ← (3,2,0),
        A[3,*] ← (-1,-1,-1),
ACCURACY  5  DIGITS,
NUMBER 3;
```

This will obtain all 3 eigenvalues and eigenvectors of

$$\begin{bmatrix} -1 & 0 & 0 \\ 3 & 2 & 0 \\ -1 & -1 & 1 \end{bmatrix}$$

LAMDA will be set equal to the vector (-1,2,1) and X will be the 3 x 3 array with eigenvectors as columns:

$$X = \begin{bmatrix} k_1 & 0 & 0 \\ -k_1 & k_2 & 0 \\ 0 & k_2 & k_3 \end{bmatrix}$$

where $k_i \neq 0$, $i = 1,2,3$.

## CONDITIONAL STATEMENT

The NAPSS conditional statement is similar to that in ALGOL, and has the form:

IF BE. THEN $S_1$, $S_2$, ..., $S_n$

ELSE $T_1$, $T_2$, ..., $T_m$;

where BE. is a boolean expression and $S_1$, $S_2$, ..., $S_n$, $T_1$, $T_2$, ..., $T_m$ are NAPSS statements.

The conditional statement as well as other compound statements in NAPSS are terminated with a semi-colon, while simple statements such as assignment statements are terminated with nothing if they are the last thing on a line or card and with a comma if they are followed by anything on the same line or card.

The use of the semi-colon with compound statements solves several problems. In the conditional statement it solves the "dangling ELSE" problem and in compound statements, in general, it permits them to be continued from one line or card to the next without any continuation mark.

If there is no ELSE clause in a conditional statement the semi-colon is placed after the statement Sn.

examples

1 IF X = 2 THEN IF Y=3 THEN Z=4; ELSE Z=5;

2 IF X = 2 THEN IF Y=3 THEN Z=4 ELSE Z=5;;

In example 1 the ELSE is associated with the first IF because the second IF is ended with a semi-colon after its THEN clause. In example 2 the ELSE is associated with the second IF.

ITERATION STATEMENT

The iteration statement has the form

$$IS.\ \ DO\ \ S_1,\ S_2,\ \ldots\ S_n;$$

where IS. represents an iteration specification and $S_1, S_2, \ldots, S_n$ are NAPSS statements. The extent of the iteration is indicated by the semi-colon.

The various iteration specifications are a generalization of those appearing in ALGOL:

i) FOR T ← 0, 1, 16, -3, 5 (T assumes values 0,1,16,-3,5)

ii) FOR Q ← .1 TO .9 BY .3 (Q assumes values .1,.4,.7)

FOR Q ← -2 TO 2 (Q assumes values -2,-1,0,1,2)

FOR Q ← 2 TO -2 (Q assumes values 2,1,0,-1,-2)

FOR Q ← -3, -1,...,6 (Q assumes values -3,-1,1,3,5)

The last example is equivalent to FOR Q ← -3 TO 6 BY (-1-(-3))

iii) Any combination of expressions from above which follow the ← :

FOR C ← 0,1,16,-3,5,.1 TO .9 BY .3,-2 TO 2,-3,-1,...,6,2 TO -2

iv) FOR 72.4 TIMES (loop is executed 72 times)

v) WHILE X > 0 OR Y < 1 loop is executed while the boolean expression is true.

vi) UNTIL |Z-Y| = 1 loop is executed until the boolean expression is true.

vii) Any combination of FOR with WHILE or UNTIL: In this case the loop is executed until one of the conditions is satisfied.

FOR Y ← 0 TO 6X+3 OR While W < .001

FOR Z3 ← 1 TO 10, 15 TO 100 BY 5 OR UNTIL X ↑ 2 < .5 ¢ -6

The ¢ in NAPSS is equivalent to the E in FORTRAN.

If a loop is being controlled by more than one index which assumes the same values the iteration can be written as follows:

FOR J,K ← 1,2,..., m DO X[K,J] ← 1/(K+J);

which is equivalent to:

FOR J ← 1,2,..., m DO

FOR K ← 1,2,..., m DO

X[K,J] ← 1/(K+J);;

Iteration variables were included in NAPSS to aid in loop control. For instance, while X represents the present value of X, X↓[-1] represents the previous value of X and X↓[-2] the one before that and so on. The number of previous iterates can be determined at compile time because the value in the square brackets is restricted to be a negative integer. No variable

names can appear in the square brackets. Previous iterates cannot be assigned values directly. They obtain their values as X is assigned different values.

example

To find a root of f(X) = 0, using Newton's method with XO as a starting value we have:

```
X ← XO
FOR 100 TIMES OR UNTIL |X-X↓[-1]| < .00005
  DO  X ← X - f(x) / f'(X) ;
```

The iteration terminates when two successive iterates agree to four decimals, or after 100 iterations. Since the iteration variable X↓[-1] is used in the until clause, the following boolean expression is not tested until the loop has been evaluated once. In general if an iteration variable, let's say X [-5], appears in a WHILE or UNTIL test of a loop, the loop would be executed 5 times before the test is made.

DECLARE STATEMENT

The declare statement is optional in NAPSS, for variables can be contextually declared when they appear in an assignment statement. But the user can explicitly assign some or all of the attributes of a variable in a declare statement. The main advantage in doing this is that when a declared variable appears

on the left of an assignment statement the attributes of an expression on the right which corresponds to the explicitly declared attributes of the variable on the left must agree, or an error message results. The remaining attributes which have not been explicitly associated with a variable name are defined contextually.

example

If A is explicitly declared to be a REAL ARRAY then

$$A \leftarrow (1,2,3)$$

is a valid assignment A because (1,2,3) is a one-dimensional matrix with all real elements. But

$$A \leftarrow (1,2,3+4i)$$

is an invalid assignment to A because all the elements of the array on the left are not real.

The declare statement in NAPSS is an executable statement. Thus, attributes of a variable name can be explicitly changed in a program by having the variable appear in different declare statements. When a variable name appears in more than one declare statement the attributes explicitly associated with it are all of attributes assigned by the last declare statement executed and all of the attributes explicitly associated with the variable name by previously executed declare statements, which have not been explicitly changed.

example

```
DECLARE  A  ARRAY, SINGLE;

   .  .  .

DECLARE  A  SCALAR, COMPLEX;
```

A, after the second declare statement has the attributes SCALAR, COMPLEX, SINGLE explicitly associated with it.

The attributes which can be associated with a variable name are REAL, COMPLEX, SINGLE, DOUBLE, SCALAR, ARRAY, FUNCTION, LOCAL, GLOBAL, INITIAL.

Three attributes cannot be assigned contextually: GLOBAL, LOCAL, and DOUBLE.

examples

i) 
```
DECLARE  A  REAL,  B  COMPLEX, D[-5 TO 10],
                   C[*, -6 TO *], E[5,6];
```

After the above declare statement has been executed A has the attribute REAL; and B the attribute COMPLEX; D is a one dimensional array with index range from -5 to 10; C is a two dimensional array, with the bounds of the first index to be defined contextually and its lower bound of the second index to start at -6 and the upper bound is to be defined contextually; E is a two dimensional array whose first index ranges from 1 to 5 and whose second index ranges from 1 to 6.

ii) DECLARE (A REAL, (B COMPLEX, C REAL) DOUBLE) GLOBAL;

Since attributes can be factored, this is equivalent to:

DECLARE A REAL, GLOBAL, B COMPLEX, DOUBLE, GLOBAL,
C REAL, DOUBLE, GLOBAL;

GLOBAL and LOCAL are unique attributes--they are assigned at compile time.

When a name has the attribute LOCAL associated with it anywhere in a procedure, say procedure A, it denotes a new variable distinct from variables with the same name in procedures containing procedure A. All occurrences of the variable in procedure A refer to the same variable until the variable is assigned either the attribute LOCAL or GLOBAL in a procedure internal to procedure A. A variable name may not be assigned both the attributes LOCAL and GLOBAL in procedure A excluding contained procedures.

The declaration of a variable to be GLOBAL has the same effect as declaring it to be LOCAL except that all occurrences of the variable in procedures where it has been declared to be GLOBAL refer to the same variable.

The scope of variable names which are not declared to be LOCAL or GLOBAL and are not parameters is the outer most containing procedure.

example

```
EXTERNAL1:  PROCEDURE

                . . .

            DECLARE (A,D) REAL, E GLOBAL;

                . . .

       L2:  A ← Z ← 4 * K

     INTERNAL1:  PROCEDURE (B)

                    . . .

              INTERNAL2:  PROCEDURE

                             . . .

                          DECLARE (A,E,D,K) GLOBAL, Z LOCAL;

                             . . .

                          END

              L3:  Z ← G + K

                       . . .

                   DECLARE (A,E,N) LOCAL;

                   END

                  . . .

            END

EXTERNAL2:  PROCEDURE

            DECLARE (A,E,D) GLOBAL;

                 . . .

            END
```

The variable names Z and K in statement L2 of procedure EXTERNAL1 and statement L3 of procedure INTERNAL1 refer to the same variable, but the variable names Z and K declared in procedure INTERNAL2 refer to different variables.

The variable names A and D in INTERNAL1 refer to different variables than the variables named A and D in EXTERNAL1, INTERNAL2, and EXTERNAL2. But since the variables names A and D in INTERNAL2 and EXTERNAL2 have the attribute GLOBAL they refer to the same variables.

The attribute INITIAL permits the assigning of initial values to arrays or scalars only. The initial values are assigned every time the declare statement is executed, unless a variable name is declared to be GLOBAL. When this is the case initial values are only assigned when:

i) the variable name has no values presently associated with it.

ii) the other attributes, explicitly declared in addition to GLOBAL, cause the previous values associated with the variable name to be destroyed.

example

i) DECLARE A REAL INITIAL (5),
B[3] SINGLE INITIAL (1,2,3) ;

A is set equal to 5 and B is set equal to the vector (1,2,3).

ii) DECLARE A[3,3] INITIAL (1,[2]*,[2](2,)),0,1);

[integer] is used as a repetitive factor and * signifies that no value is to be assigned to the corresponding element. Thus A is set equal to the matrix:

$$\begin{bmatrix} 1 & 2 & 3 \\ - & 3 & 0 \\ - & 2 & 1 \end{bmatrix} .$$

ACCURACY STATEMENT

The accuracy statement permits the user to specify the number of digits he wants retained for all his variables, except those whose accuracy is specified in a declare statement or solve statement.

The accuracy statement is an executable statement, so different accuracies can be used in various segments of a program.

example

ACCURACY 6 DIGITS

This specifies that six-significant figures will be retained for all variables.

SAMPLE PROGRAMS

The following procedures display how a user might program the Gauss Seidel Method for solving the system of linear equations AX=Y using the NAPSS language on these different levels - first on the level of ALGOL, second using array arithmetic and iteration variables, and third using the solve statement.

i) GAUSSEIDEL:

```
PROCEDURE (A,X,Y,N)
DECLARE X[N], XINT[N], A[N,N], Y[N];
FOR  I ← 1 TO  N  DO
        X[I] ← XINT[I] ← 0;
GO TO L1
L2:   FOR I ← 1 TO  N  DO  XINT[I] ← X[I];
L1:   FOR I ← 1,2,...,N DO
        T ← 0
        FOR J ← 1 TO N DO
            T ← T + A[I,J]*X[J];
        X[I] ← (Y[I] - T + A[I,I] * X[I])/A[I,I];
      W ← 0
      FOR I ← 1 TO  N  DO
        T ← ABS(X[I] - XINT[I])
        IF T > W  THEN  W ← T ; ;
      IF  W  >.5¢ -8  THEN GO TO L2
                      ELSE RETURN;
      END
```

```
ii)  GAUSSEIDEL:   PROCEDURE  (A,X,Y,N)
                   DECLARE  X[N]  INITIAL  (0 FOR N TIMES);
                   UNTIL MAX( |X-X+[-1]| ) < 5.¢-8 DO
                       FOR  I ← 1  TO  N  DO
                          X[I] ← (Y[I] - A[I,*]X+A[I,I]X[I])/A[I,I];;
                   END

iii) GAUSSEIDEL:   PROCEDURE (A,X,Y,N)
                   SOLVE A X = Y, FOR X, NUMBER N,
                         TYPE LINEAR SYSTEM,
                         USING GAUSS SEIDEL;
                   END
```

As can be seen from the above examples the amount of programming decreases greatly from example i) to example ii) and in example iii) the user actually needs to know no more about the method than its name.

The following is a complete program which solves the matrix equation $AX=Y$ by the Hestenes and Stiefel Conjugate Gradient Method, and determines the eigenvalues of $A$ by Krylov's Method. The program appears as it would if it had been inputted through an on-line terminal. The numbers on the left are statement numbers printed by the system. Underlined messages are also printed by the system.

EXECUTE MODE

```
1,000   CONJ:  PROCEDURE (Y)
2.000          DECLARE  X[N]  LOCAL, INITIAL (0 FOR N TIMES);
3,000          Z ← S ← Y
4.000          FOR  N  TIMES OR UNTIL MAX (|S|) < = .0001 DO
5.000              T ← A * Z
6.000              L ← (Z's)/(Z'T)
7.000              X ← X + L Z, S ← S - L * T
9.000              MU ← -Z'*(A * S)/(Z'T)
10,000             Z ← S + MU * Z;
11,000         RETURN X , END
13.000         READ N; DECLARE A[N,N], Y[N], B[N,0 TO N];
15.000         READ A,Y;
16,000         X ← CONJ(Y)
17,000         PRINT "THE SOLUTION OF AX=Y BY CONJUGATE GRADIENT METHOD IS",X;
18,000         READ B[*,0];
19,000         FOR K ← 1 TO N-1 DO
20,000             B[*,K] ← A'B[*,K-1];
21,000         BN ← A'B[*,N-1]
22,000         CALL GAUSSEIDEL (B,Q,BN,N)
23,000         p(t) ← (-+)↑N + (-1)↑(N+1) SUM(Q[J] t↑(J-1), FOR J ← 1 TO N)
24.000         SOLVE  p(t) = 0  FOR  T ;
25.000         PRINT "EIGENVALUE FOR MATRIX", A, "ARE" t;
26,000         END
```

After the user has logged on at a remote terminal, the system responds with the message EXECUTE MODE. The user at a terminal has the option of operating in either execution mode, where each instruction on the console level is executed as it is received or in a suppressed mode where each instruction is compiled into internal text and stored for later execution. A compiled NAPSS program or procedure is executed interpretively. When operating in execute mode and a procedure statement is typed in, as in statement number 1.000, the system temporarily enters suppressed mode until the matching end statement is received, statement number 12.000, and then re-enters execute mode.

Since X is declared to be local in procedure CONJ, the variable named X in procedure CONJ refers to a different X than the X in the console level routine, for instance the X in statement 16.000. The console level routine is the outer most procedure when operating in execute mode.

The call statement, number 22.000, causes the routine GAUSSEIDEL to be brought in from the users library as an external routine. A user may save any routine or his previous days work in his own library.

When operating from a terminal, statements are numbered, not lines. This is done to aid the user if he should have to perform any editing.

If a user has not initialized a variable before using it the system will issue a message when it attempts to execute the statement in which the undefined variable occurs, and will request him to initialize it. Than can be done indirectly by inserting an assignment statement into the program before the statement with the undefined variable in it and restarting there, or by assigning a value directly to the variable in the edit mode.

## References

1. RICE, J. R.; ROSEN, S.; NAPSS--A numerical analysis problem solving system, Proceedings - A.C.M. National Meeting 1966, p. 51.

2. RICE, J. R.; ON THE CONSTRUCTION OF POLYALGORITHMS FOR AUTOMATIC NUMERICAL ANALYSIS, Purdue University Technical Report CSD TR10 June 1967.

3. ROMAN, R. V.; SYMES, L.R.; IMPLEMENTATION CONSIDERATIONS IN NUMERICAL ANALYSIS PROBLEM SOLVING SYSTEM, Purdue University Technical Report, June 1967.

4. SYMES, L. R.; ROMAN, R. V.; SYNTACTIC AND SEMANTIC DESCRIPTION of the NUMERICAL ANALYSIS PROGRAMMING LANGUAGE (NAPSS), Purdue University Technical Report, CSD TR 11, May 1967.