ORTNRM - A Fortran Subroutine Package for the Solution of Linear Two-Point Boundary Value Problems

S. Silverston

Report Number: 68-018
ORTNRM - A Fortran Subroutine Package for the
Solution of Linear Two-Point Boundary Value Problems

S. Silverston
April, 1968

CSD TR 18
ORTNRM - A Fortran Subroutine Package for the
Solution of Linear Two-Point Boundary Value Problems

S. Silverston

ORTNRM is used to solve linear two-point boundary value problems
by the method of superposition with orthonormalization. See Reference.

Let \( x \) = the independent variable;
\( u(x) \) = the vector of \( n \) dependent variables;
\( f(x) \) = a given \( n \)-vector of functions of \( x \);
\( A(x) \) = an \( n \times n \) matrix;
\( k \) = an integer \( 1 < k < n \);
\( B \) = a constant \((n-k)\times n\) matrix;
\( D \) = a constant \( k \times n \) matrix;
\( C_1 \) = a constant \((n-k)\)-vector;
\( C_2 \) = a constant \( k \)-vector.

We want to solve the problem:

\[
\frac{d}{dx} u(x) = A(x)u(x) + f(x)
\]

\( Bu(a) = C_1 \), \( Du(b) = C_2 \).

The subroutine is entered by the statement:

CALL ORTNRM (N,M,K,Y,DER,CO,
A,NN,H,NP,NT,
TEST, C,NX,
NP01,NP02,ALT,
NERR).
In the following discussion, we break the parameters into 5 groups.

I. System of equations: 
\[ N, M, E, Y, D, R, O \]

In the method of superposition, we actually obtain \( k+1 \) solutions \( y^0, y^1, \ldots, y^k \) as follows:

Choose \( y^0(a) \neq \delta y^0(a) = C_1 \)

Choose \( y^1, \ldots, y^k(a) \neq \)

\[ (y^i, y^j) = \delta_{ij}, \quad 0 \leq i, j \leq k. \]

Solve:
\[
\frac{d}{dx} y^0(x) = A(x)y^0(x) + f(x) \\
\frac{d}{dx} y^i(x) = A(x)y^i(x), \quad 1 \leq i \leq k.
\]

We then solve the system

\[
D[ \sum_{i=1}^{k} \beta_i y^i(b) + y^0(b)] = C_2
\]

for the coefficients \( \beta_1, \ldots, \beta_k \).

The solution to the original problem is then given by

\[
u(x) = y^0(x) + \sum_{i=1}^{k} \beta_i y^i(x).\]

The solution \( y^0(x) \) is called the particular solution. The solutions \( y^i(x) \) are called base solutions.

If \( C_1 = 0 \) and \( f = 0 \), the system is homogeneous. In this case, we let

\[
By^i(a) = 0, \quad 1 \leq i \leq k
\]

and omit the particular solution. We then will solve the system
for the coefficients $\beta_1, ...,\beta_k$.

If we also have $C_2 = 0$, then one of the $\beta$'s must be chosen arbitrarily and the others computed in terms of it. In this case we only can determine $u(x)$ to within a constant multiplier. See discussion of parameter NP02 under Output Options for the normalization convention used here.

The system of equations parameters should be set as follows:

$N$: Integer. No. of dependent variables $n$.

$M$: Integer. No. of solution vectors to be used for superposition.

For inhomogeneous systems, $M = k+1$. For homogeneous systems, $M = k$.

$K$: Integer. No. of base solution vectors $k$ to be used.

$Y$: Real array dimensioned $(N,M)$. Values of the vectors $y^0(a), y^1(a), ..., y^k(a)$, chosen as discussed above.

\[
Y = \begin{bmatrix}
y^0(a) & y^1(a) & \cdots & y^k(a) \\
y^1(a) & y^2(a) & \cdots & y^{k+1}(a) \\
\vdots & \vdots & \ddots & \vdots \\
y^n(a) & y^{n+1}(a) & \cdots & y^{n+k}(a)
\end{bmatrix}
\]

For homogeneous systems,

\[
Y = \begin{bmatrix}
y^1(a) & y^2(a) & \cdots & y^k(a) \\
y^2(a) & y^3(a) & \cdots & y^{k+1}(a) \\
\vdots & \vdots & \ddots & \vdots \\
y^n(a) & y^{n+1}(a) & \cdots & y^{n+k}(a)
\end{bmatrix}
\]
**DER:** Name of subroutine for evaluation of the expressions

\[ A(x)y^0(x) + f(x) \text{ and } A(x)y^i(x) \]

To be called by a statement of the form

`CALL DER (X,Y,DY)` with:

- \( X = \) Real. Value of independent variable \( x \).
- \( Y = \) Real array dimensioned \((N,M)\). Values of solution vectors \( y^0(x), y^1(x), \ldots, y^k(x) \)
- \( DY = \) Real array dimensioned \((N,M)\). Values of

\[ \frac{d}{dx} y^0(x), \frac{d}{dx} y^1(x), \ldots, \frac{d}{dx} y^k(x) \]

The subroutine must (for inhomogeneous systems) compute

\[ A(x)y^0(x) + f(x) \]

and store it in \( DY(1,1), DY(2,1), \ldots, DY(N,1) \). It must similarly compute

\[ A(x)y^i(x), \text{ } i = 1, \ldots, k \]

and store it in \( DY(1,1), DY(2,1), \ldots, DY(N,1), \text{ } I = 2, \ldots, M \).

**CO:** Name of subroutine for computation of the values \( \beta_1, \beta_2, \ldots, \beta_k \) in the equation

\[ D[\sum_{i=1}^{k} \beta_i y^i(b) + y^0(b)] = C_2 \]

To be called by a statement of the form

`CALL CO (Y0,Y,BETA)` for an inhomogeneous system

or by a statement of the form

`CALL CO (Y,BETA)` for a homogeneous system.
with:

YO: Real array dimensioned (N). Values of \( y_0(b) \). Omitted for homogeneous system.

Y: Real array dimensioned (N,K). Values of \( y^1(b), y^2(b), \ldots, y^K(b) \).

BETA: Real array dimensioned (K). Subroutine must compute the values of \( \beta_1, \beta_2, \ldots, \beta_K \) and store them in the array BETA.

None of the parameters specifying system of equations, N,M,K,Y, are changed by ORTNRM.

II. Interval and spacing:

\( A, NN, H, NP, NT \)

We solve the problem on the interval

\[ S = [\min(a,b), \max(a,b)] \]

We may break \( S \) up into \( j \) sub-intervals \( S_1, S_2, \ldots, S_j \)

\[ \{x_0=a,x_1\}, \{x_1,x_2\}, \ldots, \{x_{j-2},x_{j-1}\}, \{x_{j-1},x_j\}, \{x_j,b\} \]

such that \( a < x_1 < x_2 < \ldots < x_{j-1} < b \) for \( a < b \),

\( a > x_1 > x_2 > \ldots > x_{j-1} > b \) for \( a > b \).

On each sub-interval \( S_i, 1 \leq i \leq j \), the solution \( u(x) \) will be computed and stored at \( n_i \) equally spaced points, i.e., letting

\[ d_i = \frac{x_i - x_{i-1}}{n_i} \]

at the points \( x_{i-1} + d_i, x_{i-1} + 2d_i, \ldots, x_{i-1} + n_id_i \).
The solution values stored at these solution points only will be available to the user for print-out as well as for use in further computation.

The intervals $d_i$ between solution points are themselves divided into increments of integration. Specifically,

$$d_i = p_i h_i$$

where $h_i$ is the length of the increment of integration, or step-size, for sub-interval $S_i$, and $p_i$ is the number of integration steps between solution points for the sub-interval $S_i$.

We thus have the following relationships among $S_i$, $n_i$, $p_i$, $h_i$, $d_i$, $x_i$, $x_{i-1}$:

$$|S_i| = |x_i - x_{i-1}| = n_i |d_i| = n_i p_i |h_i|$$

The total number of solution points on $S$ is given by

$$t = \sum_{i=1}^J n_i + 1$$

The "+1" is because the initial point $a$ is also taken as a solution point.

The interval and spacing parameters should be set as follows:

- **A**: Real. Initial value ($a$) of the independent variable.
- **NN**: Integer array dimensioned ($J$). The set of numbers $n_1, n_2, \ldots, n_J$ giving the number of solution points in each sub-interval $S_i$, respectively.
- **H**: Real array dimensioned ($J$). The set of step-sizes $h_1, h_2, \ldots, h_J$ to be used on the sub-intervals $S_i$, respectively. The $h$'s should be positive ($>0$) if $a<b$, and negative ($<0$) if $a>b$. 
NP: Integer array dimensioned \((J)\). The set of numbers \(p_1, p_2, \ldots, p_j\) which specify the number of integration increments between solution points for the sub-intervals \(S_i\), respectively.

NT: Integer. The total number \(t\) of solution points on \(S\).

Note that the number of sub-intervals \(j\) does not appear in the parameter list at all. Of course if \(j=1\), then \(NN, H,\) and \(NP\) need not be dimensioned in the calling program.

None of the interval and spacing parameters \(A, NN, H, NP, NT\), are changed by ORTNRM.

This method of specifying interval and spacing is admittedly rather complicated. However, the flexibility it affords the user in varying step-size over the region as well as in specifying output, or solution, points, is quite useful for research purposes. The latter is especially valuable in the extension of this method to non-linear problems.

III. Orthonormalization

\[ Y(x) = [y_1, \ldots, y_k] \]

\[ P \text{ is a } k \times k \text{ matrix} \]

\[ Z(x) = [z_1, \ldots, z_k] \]

\[ (z_i, z_j) = \delta_{ij}, \quad 0 \leq i \leq k, \quad 1 \leq j \leq k, \]
is performed whenever the solution vectors $y^0, y^1, \ldots, y^k$ meet some criterion to be specified. The two types of test considered here are

1) the magnitude test
   Reorthonormalization is performed whenever $|y^i(x)| > C$ for some $i=0,1,2,\ldots,k$ where $C$ is a specified constant $> 0$.

2) the angle test
   Reorthonormalization is performed whenever
   
   \[ 57.3 \cos^{-1} \left( \frac{(y^i, y^j)}{\sqrt{(y^i, y^i)(y^j, y^j)}} \right)^{1/2} - |C|, \quad 0 < i < k, \quad 0 < j < k, \quad i \neq j \]

   where $C$ is an angle specified in degrees.

In this program, the solution vectors $y^0, y^1, \ldots, y^k$ are tested at all points where they are computed, whether at "solution points" or points between the solution points.

The user also has the options of

1) reorthonormalizing at every point
2) not reorthonormalizing at all
3) always reorthonormalizing at the last point ($x=b$)

TEST: Integer-Flag for orthonormalization test as follows:

- $TEST = 0$, no test (see below under Iteration)
- $TEST = +1$, magnitude test, always orthonormalize at last point ($b$)
- $TEST = -1$, magnitude test
- $TEST = +2$, angle test, always orthonormalize at last point ($b$)
- $TEST = -2$, angle test
C: Real array dimensioned (J). For J, see preceding section, Interval and Spacing. The orthonormalization criterion, either magnitude (for TEST = ±1) or angle (for TEST = ±2), C may also be set so as to either force orthonormalization at every point, or suppress orthonormalization. See chart below. The orthonormalization criteria can be varied for each sub-interval \( S_i \). (The type of test made, as specified by TEST, is fixed for the whole interval \( S \), however.) Thus \( C \) is actually the set of criteria \( C_1, C_2, \ldots, C_j \) to be used on the sub-intervals \( S_i \), respectively. Of course, as for NN, \( H \), and \( NP \), if \( j=1 \) \( C \) need not be dimensioned in the calling program.

<table>
<thead>
<tr>
<th>Test for re-orth.</th>
<th>Mag. test TEST = ±1</th>
<th>Angle test TEST = ±2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Re-orth. at every point</td>
<td>( 0 &lt; C )</td>
<td>( 0 \leq C &lt; 90 )</td>
</tr>
<tr>
<td></td>
<td>( C = 0 )</td>
<td>( 90 \leq C )</td>
</tr>
<tr>
<td>Do not re-orth. at all</td>
<td>( C &lt; 0 )</td>
<td>( C &lt; 0 )</td>
</tr>
</tbody>
</table>

Note that the do-not-re-orthonormalize option allows ORTNRM to be used as a straight method-of-superposition package without reference to orthonormalization.

NX: Integer. The maximum number of re-orthonormalizations for which space has been allocated. (See below under Storage Space.) It is possible to have as many as
orthonormalizations.

None of the parameters TEST, C, NX, are changed by ORTNRM.

IV. Output options
NP01, NP02, ALT

The user may be interested in the following types of output from ORTNRM:

1) print-out of intermediate vectors. That is, the particular vector and base vectors \( y^0(x), y^1(x), \ldots, y^k(x) \), and, if orthonormalization occurred at \( x \), the particular and base vectors \( z^0(x), z^1(x), \ldots, z^k(x) \) resulting after orthonormalization.

2) availability of the solution vector \( u(x) \) at the specified solution points to the calling program.

This may be stored, without print-out, for use in later computations. (See below under Storage Space.)

3) print-out of the solution vector at specified solution points.

In research work, it is often of interest to compare results obtained using a method being investigated with known "exact values", or with values obtained using another method. For this reason, the option of printing alternate values of the solution \( u(x) \), obtained independently of ORTNRM, along with the values computed by ORTNRM, is provided.
11.

NP01: Integer. Flag for the print-out of intermediate vectors.

NP01 = 0, omit intermediate vector print-out

= 1, print intermediate vectors at the initial point (a), last point (b), and at all points where orthonormalization has occurred.

= 2, print intermediate vectors at all points where orthonormalization has occurred and at all solution points.

When intermediate vectors are printed, the y-vectors are always given, the z-vectors are given whenever orthonormalization has occurred at the point in question.

NP02: Integer. Flag for the output of solution vector \( u(x) \).

NP02 = 0, solution vector \( u(x) \) is not generated. This option is useful mainly in iterative processes, when perhaps only \( u(b) \) is of interest for intermediate iterations. When this option is exercised, the subroutine may be re-entered subsequently to obtain \( u(x) \). See below under Alternate Entry.

= 1, solution vector \( u(x) \) generated and stored (see below under Storage Space) but not printed. When this option is exercised, the subroutine may be entered subsequently to obtain printout. See below under Alternate Entry.

= 2, solution vector \( u(x) \) generated, stored, and printed.

= 3, solution vector \( u(x) \) generated, stored, and printed along with an alternate solution vector \( u_A(x) \) printed at the same solution points. This alternate solution vector is generated by a user-coded subroutine. See ALT below. The differences between the values computed by ORTNRM and the alternate values are also printed.
The sign of NPO2 serves as a solution-normalization flag for the case of a homogeneous system. Recall that when

\[ f = 0, \quad C_1 = 0, \quad C_2 = 0 \]

the system is homogeneous and the solution \( u \) is determined only to within a constant multiplier. If NPO2 is set negative, the solution \( u \) is normalized according to the convention that the first non-zero component of \( u(a) \) is made equal to 1. (If \( u(a) = 0 \), of course, then \( u(x) = 0 \) for all \( x \).) If the conditions

\[ f = 0 \text{ and } C_1 = 0 \]

do not hold, a minus sign on NPO2 will be ignored. However, ORTNRM does not check for the third necessary condition for a homogeneous system, namely

\[ C_2 = 0. \]

**ALT:** Name of subroutine for computing an alternate solution \( u_A(x) \).

To be called by a statement of the form

\[ \text{CALL ALT}(X,UA) \text{ with:} \]

\[ X: \text{Real. Value of dependent variable } x. \]

\[ UA: \text{Real array dimensioned (N). Values of alternate solution } u_A(x). \]

The subroutine must compute \( u_A(x) \), given \( x \), and store values in \( UA \). This subroutine is called only if NPO2 is set to \( \pm 3 \). If NPO2 is not set to \( \pm 3 \), a dummy name may be used for ALT in the CALL ORTNRM statement.
Neither of the parameters specifying output, NPO1 or NPO2, are changed by ORTNRM.

V. Error flag

NERR

There is only one error condition to be flagged. This is the case when not enough space has been allocated for storage of re-orthonormalization parameters. In other words, the case in which NX is too small.

NERR: Integer variable. If there has been no error, NERR will have been set to 0 on return from ORTNRM. If the above error condition exists, NERR is set to 1 on return from ORTNRM. In case of error, a note is also printed giving details.

The user must include the names for DER, CO, and, if the alternate solution option is used, ALT in an EXTERNAL statement in the calling program.

Storage Space

The user must allocate working storage space for use by ORTNRM. This is done via the labeled COMMON block /SCRATCH/. The ORTNRM package will use the first L locations of /SCRATCH/, where

\[ L = (NT+6)^*N^*H + (3*K*K*(K-1)/2+1)^*NX + K + NT + 1 \]

and the other variables are as in the CALL ORTNRM statement. (For homogeneous systems, L is actually less than the above, by K*NX locations.)
Upon return from ORTNRM, the first \((N, NT)\) locations of /SCRATCH/ will contain the solution \(u(x)\), provided solution generation has been requested. Thus, the full solution \(u(x)\), as evaluated at the NT solution points, becomes available to the calling program. For example, suppose
\[
N = 4, M = 3, K = 2, NT = 11, NX = 25.
\]
The user might include the following statement in the calling program:

```
COMMON /SCRATCH/ U(4,11), S(374)
```

**Auxiliary COMMON Blocks**

ORTNRM uses, in addition to /SCRATCH/, COMMON blocks named

```
/KKKK/ and /MMMM/.
```

ORTNRM does not use blank COMMON.

**Alternate Entry**

ORTNRM may be re-entered to effect solution generation or solution print-out where this has been temporarily suppressed via the flag NP02 as described above.

1) To resume processing after solution \textit{generation} has been suppressed (by setting NP02=0), use

```
CALL SOLN (...)
```

2) To resume processing after solution \textit{print-out} has been suppressed (by setting NP02=+1), use

```
CALL PRM (...)
```
The argument lists for SOLN and PRM are exactly the same as that for ORTNRM, except that the value of NPO2 must be changed.

For CALL SOLN, NPO2 must be \(+1, +2, \text{ or } +3\).

For CALL PRM, NPO2 must be \(+2 \text{ or } +3\).

**Iteration**

In certain iterative processes, it may be necessary to establish a set of orthonormalizing transformations on a first pass, and then use the same transformations at the same points on subsequent passes. (On these subsequent passes, the transformations may not actually affect strict orthonormalization; however, this may be desirable for purposes of keeping all iterations uniform.) This can be accomplished via the parameter TEST. If TEST is set to 0, transformations as established on a previous pass and stored in COMMON block /SCRATCH/ will be used. In this case, testing against orthonormalization criteria, as well as computation of new orthonormalizing transformation coefficients, will be omitted.

**Backward Integration**

Generated solutions will be stored and printed in the direction of increasing \(x\), regardless of whether \(a < b\) or \(b < a\). This is done for the following reasons:

ORTNRM is primarily useful for problems in which there is some numerical instability. In problems of this type, the instability may often exist for one direction of integration but not for the
other. Generating the solution always in the same direction facilitates comparison when the user wants to try solving a problem in both directions.

Another application of ORTNRM is in the area of unstable initial-value problems. Such problems can be worked backwards as boundary-value problems. In this case too it is convenient to have the solution stored and printed in the "forward" direction.

Deck Set-up

The ORTNRM package consists of the following subroutines:

ORTNRM
ORTSUB
RKMKUT
NUGO
ARRAY
RND
FLIP
BLOCK DATA

The largest of these subroutines, ORTSUB, needs 50600 locations to compile on the CDC 6500.

The package uses COMMON blocks named /SCRATCH/ (discussed above)
/MKKK/
/MNOM/

The ORTNRM package should be placed after the calling program in the deck to allow proper loading of COMMON blocks.

APPENDIX

Fortran listing of ORTHOM package
SUBROUTINE ORTNRM

PARAMETERS
SPECIFYING SYSTEM OF EQUATIONS

1  \( N > 1 \)
2  \( A, M, K, R, N, N, M, P, T \)
3  \( A, M, K, R, N, N, M, P, T \)
4  \( M, P, T \)

PARAMETERS
DEFINING INTERVAL AND SPACING

\( \Delta a, \Delta b, \Delta c, \Delta d \)

PARAMETERS
SPECIFYING ORTHONORMALIZATION

\( A, M, K, R, N, N, M, P, T \)

TEST, C, \( a, b, c, d \)

SPECIFICATION OF USERS OUTPUT OPTIONS

\( M, P, T, A, B, C, D, E, F, G \)

ERROR FLAG

\( FERR \)

EXTERNAL

\( Dมหาคร, A, B, C, D, E, F, G \)

CALL COTSJA

RETURN

END

SUBROUTINE COTSJA

\( \alpha, \beta, \gamma, \delta, \epsilon, \zeta, \eta, \theta, \iota, \kappa, \lambda, \mu, \nu, \xi, \pi, \rho, \sigma, \tau, \upsilon, \phi, \chi, \psi, \omega \)

EXTERNAL

\( Dมหาคร, A, B, C, D, E, F, G \)

DATA (101) \( x, y, z, w, a, b, c, d, e, f, g \)

INITIALIZE

\( T \)

CALL ORTPO

RETURN

END
\[
K = 2
\]
\[
\text{HOM} = K \times \text{EQ}_M
\]
\[
\text{IF} (\text{HOM}) K = 1
\]
\[
K1 = K + 1
\]
\[
K1 = K - 1
\]
\[
\text{NEX} = 0
\]
\[
U = 1
\]
\[
V = 1
\]
\[
\text{IN} = 1
\]
\[
\text{IL} = \text{NT}
\]
\[
\text{MORE} = K \times \text{GT}_1
\]
\[
\text{LAST} = N \times \text{GT}_0
\]
\[
\text{OLDG} = N \times \text{EQ}_0
\]
\[
\text{MAGST} = \text{IABS}(\text{IND}) \times \text{EQ}_1, \text{OR}_e \times \text{NOT} \times \text{MORE}
\]
\[
P = 0
\]
\[
\text{NTC} = 1
\]
\[
XN(1) = X
\]
\[
\text{NCC} = 0
\]
\[
\text{NPLP} = 1
\]
\[
\text{NHLP} = 1
\]
\[
\text{GO TO 17}
\]

\text{START INTEGRATION LOOP}

10 \text{NCC} = \text{NCC} + 1
\text{NPLP} = \text{NPL} \times \text{NCC}
\text{NHLP} = \text{NHLP} \times \text{NCC}
\text{DX} = \text{H} \times \text{FLOAT} \times \text{NHLP}
\text{NTC} = \text{NTC + NPLP}
\text{NF} = U
\text{IF} (A(\text{NCC}) \geq 0) \text{GO TO 13}
\text{NF} = -1
\text{GO TO 17}

13 \text{IF} (\text{MAGST}) \text{GO TO 15}
\text{C} = \text{COS} \times (A(\text{NCC}) / 57.3)
\text{IF} (A(\text{NCC}) \geq 90^\circ) \text{NF} = 1
\text{GO TO 17}

15 A2 = A(\text{NCC})^2
\text{IF} (A(\text{NCC}) \leq 0) \text{NF} = 1

17 \text{DO 301 NPC = 1,NPLP}
\text{LASTBK} = \text{NPC} \times \text{EQ}_0 \times \text{NPLP}, \text{AND} \times \text{NTC} \times \text{EQ}_0 \text{NT}
\text{XN(U+1)} = XN(U) + DX
\text{DO 301 NHC = 1,NHLP}
\text{P = P + 1}
\text{NOPO} = \text{NHLC} \times \text{LT} \times \text{NHLP}
\text{IF} (P \times \text{EQ}_1) \text{GO TO 200}
\text{ENDPT = LASTBK} \times \text{AND} \times \text{NOT} \times \text{NOPO}
\text{CALL RUNKUT} (X,Y,Y(1,1,1),XN+H(\text{NCC}) \times \text{DERIV})
\text{U = U + 1}

\text{BRANCH ON ORTHONORMALIZATION}

\text{IF (OLDG) IF (P-LX(V)) 55,70,55}
\text{IF (NF) 55,19,100}

19 \text{IF} (\text{ENDPT} \times \text{AND} \times \text{LAST}) \text{GO TO 100}

\text{TEST FOR ORTHOGONALITY OR MAGNITUDE}

\text{G = 0.}
\text{MM = M}
DO 30 I = 1,M
IF (MAGTST) MM = I
DO 30 J = 1,MM
E = 0.
DO 20 L = 1,N
20 E = E + Y(L,I,1)*Y(L,J,1)
Y(I,J,5) = E
30 IF (E.GT.G) G = E
IF (MAGTST) GO TO 52
DO 40 I = 1,M
DO 40 J = I,M
Y(I,J,5) = Y(I,J,5)/G
T = 1
M1 = M - 1
DO 45 I = 1,M1
L = I + 1
DO 45 J = L,N
45 IF (Y(I,J,5)**2.GT.0.0001) Y(I,J,5) = Y(I,J,5)/G
50 IF (T) 100,55,100
52 DO 54 J = 1,M
54 IF (Y(I,J,5)**2.GT.0.0001) GO TO 100
C C

NO RE-ORTHONORMALIZATION

55 IF (NPO) GO TO 65
57 DO 60 J = 1,N
DO 60 1 = 1,M
60 Z(J,I,U) = Y(J,I,1)
C IS PRINT-OUT INDICATED
65 REG = .TRUE.
IF (NPO.GT.0 .AND. ( ENDPT .OR. P.EQ.1)) GO TO 222
IF (NPO = 2) 300,220,300
C C

RE-ORTHONORMALIZATION USING OLD COEFFICIENTS

70 IF (V.LT.NX) GO TO 74
WRITE (6,71)
71 FORMAT (95HINSUFFICIENT STORAGE FOR ORTHONORMALIZATION PARAMETERS / 78H WITH PREVIOUSLY DETEERMINED PARAMETERS, ERROR RETURN TO CALLING PROGRAM GIVEN. / 32H SOLUTION GENERATION SUPPRESSED, )
NERR = 1
RETURN
C C

ORTHOGONALIZATION

74 DO 80 Q = 1,N
L = 0
DO 80 J = K,M
Z(Q,I,U) = Y(Q,I,1)
IF (I.EQ.K) GO TO 80
I = I - 1
DO 75 J = K,I
L = L + 1
75 Z(Q,I,U) = Z(Q,I,U) - OMEGA(L,V)*Y(Q,J,1)
CONTINUE
NORMALIZATION

80 DO 85 I = K,M
IR = I - KM
DO 85 J = 1,N
85 \( Z(J,1,U) = R(1,R) \times Z(J,1,U) \)

C BRANCH ON HOMOGENEITY
90 IF (HOM) GO TO 190
GO TO 183

C RE-ORTHONORMALIZATION WITH NEW COEFFICIENTS
100 IF (V-NLOX) GO TO 105
NORXS = NORXS + 1
IN = U
105 LX(V) = P
C FIRST VECTOR AND MOD**2
E = 0
DO 110 I = 1,N
E = E + Y(I,K,X)**2
110 Z(I,K,U) = Y(I,K,X)
R(1,R) = lo/E
C BEGIN MAJOR ORTHONORMALIZATION LOOP
IF (NOT MORE) GO TO 165
L = 0
DO 160 1 = K+M
11 = I - 1
LO = L
C BEGIN LOOP TO DETERMINE OMEGAS
DO 140 J = K, I-1
L = L + 1
C OBTAIN FIRST TERM OF EXPRESSION FOR OMEGA (IN D, P.)
D = 0
DO 120 Q = 1,N
D = D + Y(Q,I,J) \times Z(Q,J,U)
IR = I - KM1
DG = D \times R(I,R)
S = J + 1
IF (S GT 11) GO TO 140
DO 130 Q = S,11
D = 0
DO 125 T = 1,N
125 D = D + Y(T,T,J) \times Z(T,O,U)
IR = Q - KM1
IM = (IR-2) \times (IR-1)/2 + J - KM1
130 DG = DG - D \times R(IR,R) \times OMEGA(IM,V)
140 OMEGA(L,V) = DG
C END OF OMEGA LOOP
C ORTHOGONALIZATION
DO 150 Q = 1,N
L = L0
Z(Q,1,U) = Y(Q,1,1)
DO 150 J = K,11
L = L + 1
150 Z(Q,1,U) = Z(Q,1,U) - OMEGA(L,V) \times Y(Q,J,1)
1R = I - KM1
E = 0
DO 155 Q = 1,N
155 E = E + Z(Q,1,U)**2
160 R(1,R) = lo/E
C END MAJOR ORTHONORMALIZATION LOOP
C NORMALIZATION
165 DO 170 I = K,M
IR = I - KM1
R(IR,V) = SQRT(R(IR,V))
DO 170 J = 1,N
170 Z(J,1,U) = R(IR,V)*Z(J,1,U)
C CALCULATE ALPHAS (IN D*, P*)
IF (HOM) GO TO 190
DO 180 I = 2,M
D = 0.
DO 175 J = 1,N
175 D = D + Y(J,1,1)*Z(J,1,U)
180 ALPHA(I-1,V) = D
C ORTHOGONALIZE PARTICULAR SOLUTION
183 DO 185 J = 1,N
Z(J,1,U) = Z(J,1,U) - ALPHA(I-1,V)*Z(J,1,U)
C IS PRINT-OUT INDICATED
190 REG = .FALSE.
V = V + 1
IF (NP01) 222,290,222
C PRINT-OUT OF VECTORS
C FIRST POINT ~ SET UP LIMITS ~ PRINT HEADING
200 IF (NP01.EQ.0) GO TO 57
NK = 2 - K
NBK = (M-1)/6 + 1
NXS = M - 6*(NBK-1)
HED(4) = H6
IF (M.EQ.6) HED(4) = SP6
WRITE (6,205)
205 FORMAT (7H10RTNRM 42X 20HINTERMEDIATE VECTORS)
DO 210 I = 1,NBK
210 WRITE (6,FMT) X*(Y(S,T,1),T=J,L,S=1,N)
GO TO 57
C PRINT Y-VECTORS
220 IF (NP01) GO TO 300
222 J = -5
FMT(2) = BEG(1)
FMT(3) = EE(6)
FMT(5) = EE(6)
DO 240 I = 1,NBK
240 J = J + 6
L = J + 5
IF (1.LE.NBK) GO TO 225
FMT(3) = EE(NXS)
FMT(5) = EE(NXS)
L = M
225 IF (1.LE.1) GO TO 230
WRITE (6,FMT) X*(Y(S,T,1),T=J,L,S=1,N)
FMT(2) = BEG(2)
GO TO 240
230 WRITE(6,FMT) (Y(S,T,1),T=J,L,S=1,N)
240 CONTINUE
IF (REG) GO TO 300
C PRINT Z-VECTORS
J = -5
FMT(3) = EE(6)
FMT(5) = EE(6)
DO 250 I = 1,NBK
J = J + 6
L = J + 5
IF (I.NE.NBK) GO TO 250
FMT(3) = EE(NXS)
FMT(5) = EE(NXS)
L = M

250 WRITE (6,FMT) ((Z(S+T,U),T=J,L),S=1,N)
C
C
C
C

290 DO 295 I = 1,M
DO 295 J = 1,N
295 Y(J+1) = Z(J,I,U)
CALL NUGO
C
C
C
C

300 IF (NOPO) U = U - 1
301 CONTINUE
IF (NTC.LT.NT) GO TO 10
LX(V) = P + 1
CALL NUGO

END INTEGRATION LOOP

ENTRY SOLN
IF (HOM) CALL COEFS (Z(1,1,NT),BETA(1,V))
IF (.NOT.HOM) CALL COEFS (Z(1,1,NT),Z(1,2,NT),BETA(1,V))
CALCULATE INTERMEDIATE BETAS

308 IF (0.EQ.1) GO TO 340
S = Q - 1
DO 310 I = 1,KR
E = ALPHA(I,Q)
IF (.NOT. HOM) E = E - ALPHA(I,S)
310 Y(I+1) = R(I,S)*E
KD = 1
DO 335 K = KD,KR
BETA(I,S) = Y(I+1)
IF (I.EQ.KR) GO TO 335
KD = KD + 1
DO 335 K = KD,KR
L = (K-1)*(K-2)/2 + 1
330 BETA(I,S) = BETA(I,S) - OMEGA(L,S)*Y(K+1)
335 CONTINUE
G = S
GO TO 308

REVERSE ARRAYS IF INTEGRATION WAS BACKWARDS

340 IF (H(1) .GT. 0.) GO TO 350
CALL FLIP (XN,NT,1)
DO 342 I = 1,M
DO 342 J = 1,N
DO 342 CALL FLIP (Z(J,I,1),NT,NXM)
IF (V .EQ. 1) GO TO 350
DO 346 I = 1,KR
CALL FLIP (BETA(I,1),V,KR)
J = V - 1
P = P + 2
DO 348 I = 1,J
CALL FLIP (LX(I,J,1))
IF (NORM < 0.0) GO TO 350
1L = NT - IN + 1
IN = 1

NORMALIZATION FOR HOMOGENEOUS SYSTEM

350 G = 1.
IF (.NOT. HOM OR NP02 .GT. 0) GO TO 370
DO 360 I = 1,N
G = G + BETA(J,1)*Z(I,J,1)
G = G
G = 1./G

COMP, AND PRINT LOOP

370 REG = 'FALSE'
380 NP2 = IABS(NP02)
NPC = 0
NBK = 56/(N+1)
NC = 0
Q = 0
V = 1
MORE = N*GT*KR
KRI = KR + 1
P = 1
NKC = 1
NPC = 0
IF (IN .EQ. 1) GO TO 388
U = 1
GO TO 384
383 NNC = NNC + 1
384 NLP = NN(NNC)
DO 386 NPC = 1,NLP
P = P + NP(NNC)
U = U + 1
IF (U .EQ. IN) GO TO 388
GO TO 383

CALCULATE SOLUTIONS
388 DO 600 I = 1,N+1
390 IF (LX(V).GT.P) GO TO 400
   V = V + 1
   GO TO 390
400 IF (REG) GO TO 610
   IF (HOM) GO TO 420
   DO 410 J = 1,N
      Y(J,1,1) = Z(J,1,1)
   DO 410 K = 2,M
      Y(J,1,K) = Y(J,1,K-1) + BET(A(K-1,V)*Z(J,K,I)
   GO TO 440
410 IF (NPC.LT.NNC) GO TO 445
   NNC = NNC + 1
   NPC = 0
   NPC = NPC + 1
   P = P + NP(NNC)
    PAGE HEADING
   IF (NPA2.EQ.1) GO TO 580
   NC = NC + 1
   IF (MOD(NC,NBK).NE.1) GO TO 480
   NPG = NPG + 1
   WRITE (6,450) NPG
      FORMAT (7H10RTN RM 10X 14H SOLUTION (PAGE 13, 1H) )
   IF (NPA2.EQ.2) WRITE (6,460)
   IF (NPA2.EQ.3) WRITE (6,470)
   450 FORMAT (1H07X1HX11X 4HBETA 16X 1HU 13X 9HU COMPARE 9X 4HDIFF )
   PRINT SOLUTIONS
   480 IF (NPA2.EQ.3) GO TO 520
      WRITE (6,490) XN(J),Y(J,1,1),J=1,KR)
   490 FORMAT (1H0F11 o 4»3E18.B8, 2E18.O8 )
      IF (MORE) WRITE (6,500) Y(J,1,1),J=KRI,N)
   500 FORMAT (10X1E18.B8 )
   IF (REG) GO TO 600
   GO TO 580
   520 CALL EXACT (XN(1),Y(1,1,2))
   DO 530 J = 1,N
      530 Y(J,1,3) = Y(J,1,1) - Y(J,1,2)
   WRITE (6,540) XN(1) (BETA(J,V),Y(J,1,1),Y(J,1,2),J=1,KR)
   540 FORMAT (1H0F11.4»2E18.B8 / (E30.8, E18.8 ) )
      IF (MORE) WRITE (6,550) Y(J,1,1),J=KRI,N)
   550 FORMAT (30X 2E18.B8, 2E18.O8 )
   IF (REG) GO TO 600
   STORE SOLUTIONS
   DO 590 J = 1,N
   Q = Q + 1
590 Z(Q,1,1) = Y(J,1,1)
   600 CONTINUE
ENTRY PRM
REG = TRUE
GO TO 380
ENTRY PRM
REG = TRUE
GO TO 380

610 DO 620 T = 1, N
620 Y(T »1»1) = Z(Q,1»1)
GO TO 440
END
SUBROUTINE RUNKUT (X»Y»D»N»H»DERIV)
DIMENSION Y(N), D(N»5), E(2)
COMMON /MMMM/ MID
DOUBLE PRECISION XDP, H2, EDP
LOGICAL MID
EQUIVALENCE (EDP » E)
H2 = 0.5*D3LE(H)
H6 = H/6.
IF (MID) GO TO 20
DO 10 I = 1, N
EDP = Y(I)
D(I,1) = E(1)
D(I,2) = E(2)
MID = CTRUE,
XDP = X
10 CALL DERIV (X»Y, D(I,1))
DO 20 I = 1, N
D(I,3) = Y(I) + SNGL(H2)*D(I,4)
XDP = XDP + H2
CALL DERIV (SNGL(XDP), D(I,3), D(I,5))
DO 40 I = 1, N
D(I,4) = D(I,4) + 2*H*D(I,5)
40 D(I,3) = Y(I) + SNGL(H2)*D(I,5)
CALL DERIV (SNGL(XDP), D(I,3), D(I,5))
DO 60 I = 1, N
D(I,4) = D(I,4) + 2*H*D(I,5)
60 Y(I) = RND(EDP)
XDP = XDP + H2
X = RND(XDP)
CALL DERIV (X»Y, D(I,5))
DO 60 I = 1, N
D(I,4) = D(I,4) + D(I,5)
E(1) = D(I,1)
E(2) = D(I,2)
EDP = EDP + H6*D(I,4)
DO 60 I = 1, N
D(I,1) = E(1)
D(I,2) = E(2)
60 Y(I) = RND(EDP)
RETURN
END
SUBROUTINE NUGO
COMMON /MMMM/ MID
LOGICAL MID
MID = FALSE
RETURN
END
FUNCTION RND (D)
DIMENSION D(2)
EQUIVALENCE (A,J)
RND = D
IF ( (ABS(D(2)) .GT. 4000000000000000b0) ) RETURN
A = ABS(D)
J = J + 1
A = A .LT. 4000000000000000b0
RND = SIGN(A,D)
RETURN
END

SUBROUTINE ARRAY (Y,S,N,M)
DIMENSION Y(N,M),S(N,M)
DO 10 I = 1,M
  DO 10 J = 1,N
    S(J,I) = Y(J,I)
10 RETURN
END

SUBROUTINE FLIP (Y,N,J)
DIMENSION Y(I)
L = (N/2 - 1)*J + 1
M = N*J + 1
DO 10 I = 1,L,J
  M = M - J
  E = Y(M)
  Y(M) = Y(I)
10  Y(I) = E
RETURN
END

BLOCK DATA
COMMON /KKKK/ K1,NTK,NK /MMMM/ MID
LOGICAL MID
DATA K1 /0/
DATA MID /F/
END