report no. WD 85-03

BOUNDPAK
User's Manual (revised version)
R.M.M. Mattheij, G.W.M. Staarink

october 1985

project 830901
BOUNDPAK
A Package for Solving Boundary Value Problems
User's Manual
Chapter I
Linear non-stiff Boundary Value Problems

R.M.M. Mattheij
G.W.M. Staarink

Mathematisch Instituut / Wiskundige Dienstverlening
Katholieke Universiteit
Toernooiveld
6525 ED Nijmegen
The Netherlands
BOUNDPAK is a package for computing solutions of boundary value problems (BVP). At this moment BOUNDPAK contains routines to compute solutions of two-point linear BVP for non-stiff problems, which are described in the first chapter. The routines are designed for either general boundary conditions (BC), (the routines DMUTSG and SMUTSG for respectively double precision and single precision computation), separated BC, (the routines DMUTSS and SMUTSS) or partially separated BC, (the routines DMUTSP and SMUTSP).

The manual is designed in such a way that both the user who only likes to know how to call the routines and the user who likes to know more about the method should find his way through the following description. The first user may restrict himself to § 1 and § 5 and also might consult § 4 if more information is needed. The present routines not only try to compute the desired solution accurately, but also aim at a reliable diagnosis when the computation encounters difficulties. In any case (also if no problems are reported) the routines deliver useful stability constants of the problem.

It is our intention to extend BOUNDPAK in the near future with routines for other specific cases. Comments and reports about the present routines are very much welcomed.

Last update: July 1985

AMS (MOS) Classification: 65L10, 68C05
§ 1. Introduction to the method used

In this section we give a brief description of the method we use to solve two point boundary value problems (BVP). For an extensive description see §§ 2, 3 and 4.

Consider the ordinary differential equation (ODE):

\[ \frac{dx}{dt} = L(t)x(t) + r(t) \quad \alpha \leq t \leq \beta \]

and the boundary condition (BC):

\[ M_\alpha x(\alpha) + M_\beta x(\beta) = b \]

where \( L(t) \) is an \( n \times n \)-matrix function, \( x(t) \) and \( r(t) \) an \( n \)-vector function, \( M_\alpha, M_\beta \) \( n \times n \)-matrices and \( b \) an \( n \)-vector.

Each solution of (1.1) can be written as:

\[ x(t) = F(t)c + w(t) \]

where \( F(t) \) is a fundamental solution of the homogeneous part of (1.1a) and \( w(t) \) a particular solution of (1.1a).

Basically our method is a multiple shooting method, although in a special implementation. So the interval \([\alpha, \beta]\) is divided into \( N \) subintervals, say, \([t_i, t_{i+1}]\), \((t_0 = \alpha; t_N = \beta)\). On each subinterval \([t_i, t_{i+1}]\) a particular solution \( w_i(t) \) and a fundamental solution \( F_i(t) \) is computed. The computed fundamental solutions are such that

\[ F_{i+1}(t_{i+1}) = F_{i+1}(t_{i+1})U_{i+1} = Q_{i+1}U_{i+1} \]

where \( Q_{i+1} \) is orthogonal and \( U_{i+1} \) upper triangular.

On each subinterval we have

\[ x(t) = F_i(t)a_i + w_i(t) \]

Matching at the endpoints of the subintervals leads to

\[ F_i(t_{i+1})a_i + w_i(t_{i+1}) = F_{i+1}(t_{i+1})a_{i+1} + w_{i+1}(t_{i+1}) \]

which results into the recursion

\[ a_{i+1} = U_{i+1}a_i + Q_{i+1}^{-1}[w_i(t_{i+1}) - w_{i+1}(t_{i+1})] \]

Denoting \( d_{i+1} = Q_{i+1}^{-1}[w_i(t_{i+1}) - w_{i+1}(t_{i+1})] \) we have:

\[ a_{i+1} = U_{i+1}a_i + d_{i+1} \]

It is easy to see that
(1.8) \quad x(t_i) = Q_i a_i + w_i(t_i).

Now, any solution \{a_i\} of recursion (1.7) can be written as:

(1.9) \quad a_i = \Phi_i c + z_i,

where \{\Phi_i\}^N_{i=0} is a fundamental solution of (1.7), i.e.

(1.10) \quad \Phi_{i+1} = U_{i+1} \Phi_i

and \{z_i\}^N_{i=0} a particular solution of (1.7).

After computing a fundamental solution \{\Phi_i\}^N_{i=0} and a particular solution \{z_i\}^N_{i=0}, we can compute \(c \) from (cf. (1.1b))

(1.11) \quad [M_a Q_0 \Phi_0 + M_b Q N \Phi_N]c = b - M_a w_0(t_0) - M_b w_N(t_N) - M_a Q_0 z_0 - M_b Q N z_N.

Then \(x(t_i)\) follows from (1.9) and (1.8)

Remark 1.12

If the matrix \([M_a Q_0 \Phi_0 + M_b Q N \Phi_N]\) is ill-conditioned, computing \(c \) from (1.11) may result in inaccurate computation of the \(x(t_i)\). The routines compute a condition number \(CN\) which indicates whether this matrix is ill-conditioned or not (cf. (3.12)). An other problem is that errors might be propagated in an unstable way when the recursion (1.7) is used (although this should not be any problem in a well-conditioned case). The routines compute an estimate of the amplification of errors, which we call the amplification factor (in fact an other condition number).
chosen such that

\[ F_{\text{i+1}}(t_{\text{i+1}}) = F_{\text{i+1}}(t_{\text{i+1}})U_{\text{i+1}} = Q_{\text{i+1}}U_{\text{i+1}}, \quad i = 0,1,\ldots,N-1, \]

where \( Q_{\text{i+1}} \) is an orthogonal matrix and \( U_{\text{i+1}} \) an uppertriangular matrix. By letting \( U_0 = I \), we may include case \( i = -1 \), if we choose \( F_0(t_0) \) orthogonal.

By matching the relations (2.7) at the points \( t_{\text{i+1}} \) we then obtain:

\[ x(t_{\text{i+1}}) = F_{\text{i+1}}(t_{\text{i+1}})a_{\text{i+1}} + w_{\text{i+1}}(t_{\text{i+1}}) \]
\[ = Q_{\text{i+1}}U_{\text{i+1}}a_{\text{i}} + w_{\text{i}}(t_{\text{i+1}}), \quad i=0,1,\ldots,N-1. \]

If we denote

\[ d_{\text{i+1}} = Q_{\text{i+1}}^{-1}[w_{\text{i}}(t_{\text{i+1}}) - w_{\text{i+1}}(t_{\text{i+1}})] , \]

we thus obtain the following uppertriangular recursion:

\[ a_{\text{i+1}} = U_{\text{i+1}}a_{\text{i}} + d_{\text{i+1}} , i=0,1,\ldots,N-1. \]

(This explains the name of the code "Multiple shooting Using Triangular Systems").

By our choice of the \( F_{\text{i}} \) we immediately see that:

\[ a_{\text{i}} = Q_{\text{i}}^{-1} \left( x(t_{\text{i}}) - w_{\text{i}}(t_{\text{i}}) \right). \]

Now let \( \{\Phi_{\text{i}}\}_{\text{i}=0}^{\text{N}} \) be a fundamental solution of (2.11) (cf. (3.4)), i.e.

\[ \Phi_{\text{i+1}} = U_{\text{i+1}}\Phi_{\text{i}}, \quad i = 0,1,\ldots,N-1 \]

and let \( \{z_{\text{i}}\}_{\text{i}=0}^{\text{N}} \) be some particular solution of (2.11) (cf. (3.3)). Then there should exist some vector \( c \) such that

\[ a_{\text{i}} = \Phi_{\text{i}} c + z_{\text{i}}, \quad i = 0,1,\ldots,N-1. \]

From (2.12) and (2.14) we therefore obtain the relation:

\[ x(t_{\text{i}}) = w_{\text{i}}(t_{\text{i}}) + Q_{\text{i}}(z_{\text{i}} + \Phi_{\text{i}} c), \quad i = 0,1,\ldots,N-1. \]

After substituting \( x(t_0) = x(\alpha) \) and \( x(t_N) = x(\beta) \) in the BC (2.2) we thus find:

\[ \left[ M_{\alpha} Q_{\alpha}^{\alpha} + M_{\beta} Q_{N}^{\beta} N \right] c = b - M_{\alpha} w_{\alpha}(\alpha) - M_{\beta} w_{\beta}(\beta) - M_{\alpha} Q_{\alpha}^{\alpha} x_{\alpha} - M_{\beta} Q_{N}^{\beta} x_{N}. \]

The vector \( c \) which follows from (2.16) gives us the desired solution values \( x(t) \) via (2.15).

**Remark 2.17**

In the case that the ODE (2.1) is homogeneous, i.e. \( r(t) = 0, \quad t \in [\alpha,\beta] \), there is no particular solution to be computed. Then (2.7), (2.9), (2.11), (2.12), (2.15) and (2.16) are to be replaced by:
\[ (2.7) \quad x(t) = F_1(t)a_i, \]
\[ (2.9) \quad x(t_{i+1}) = F_{i+1}(t_{i+1})a_{i+1} = Q_{i+1}U_{i+1}a_i, \]
\[ (2.11) \quad a_{i+1} = U_{i+1}a_i, \]
\[ (2.12) \quad a_i = Q_i^{-1}x(t_i), \]
\[ (2.15) \quad x(t_i) = Q_i\Phi_i c, \]
\[ (2.16) \quad [M_{\alpha}Q_0\Phi_0 + M_{\beta}Q_N\Phi_N]c = b, \]
respectively (for relevant indices i).

§ 2.2 BVP with Partially Separated BC

Sometimes the BC \( (2.2) \) is known to have a few zero rows in the matrix \( M_\alpha \) and/or the matrix \( M_\beta \). These zeros can be utilized to reduce the computational labour, in that a smaller number of basis solutions has to be computed. For our discussion the following typical BC is to be considered:

\[ (2.18a) \quad ^1M_\alpha x(\alpha) + ^1M_\beta x(\beta) = b^1, \]
\[ (2.18b) \quad ^2M_\alpha x(\alpha) = b^2. \]

Here \( ^1M_\alpha \) and \( ^1M_\beta \) are \( k_s \times n \)-matrices, \( ^2M_\alpha \) an \( (n-k_s) \times n \)-matrix and \( b^1 \) and \( b^2 \) vectors of dimension \( k_s \) and \( n-k_s \) respectively. (i.e. \( M_\beta \) has zeros in its last \( n-k_s \) rows). The reduction in computing \( F_i \) consists of the fact that we only compute its first \( k_s \) columns, viz. \( \{F_i^1\} \), by requiring that:

\[ (2.19a) \quad ^2M_\alpha F_0^1(\alpha) = 0, \]
\[ (2.19b) \quad ^2M_\alpha w_0(\alpha) = b^2. \]

The particular solution \( w_0(\alpha) \) is then chosen such that it satisfies the decoupled initial value part, i.e.

\[ (2.19b) \quad ^2M_\alpha w_0(\alpha) = b^2. \]

Formally we thus see that the desired solution \( x \) should lie in a linear variety \( w_0 + \text{span}(F_0^2) \), where \( F_0^2 \) is just some complementary part of the fundamental solution \( F_0^1 \). From \( (2.18) \) and \( (2.19) \) we see that \( \text{span}(w_0) \perp \text{span}(F_0^1) \).

Now we can proceed as in the general case, i.e. we can divide \( [a,b] \) into subintervals \( [t_i,t_{i+1}] \), \( i=0,1,\ldots,N-1 \). On each subinterval \( [t_i,t_{i+1}] \) a partial fundamental solution \( F_i^1 \) and a particular solution \( w_i \) is computed such that at the initial point of the interval:
\[ \text{span}(F_i(t_i)) = \text{span}(F_{i-1}(t_i)), \]

(2.20) \[ w_i(t_i) \perp \text{span}(F_i(t_i)), \]

where \[ w_i(t_i) \in w_{i-1}(t_i) + \text{span}(F_{i-1}(t_i)). \]

This then means that there exist \( k \)-vectors \( a_i \), such that for any \( i \)

(2.21) \[ x(t) = F_i(t)a_i + w_i(t). \]

In our algorithm we choose \( F_i(t_i) \) such that its columns are orthogonal. The analogue of (2.8) reads therefore:

(2.22) \[ F_i(t_{i+1}) = F_{i+1}(t_{i+1})V_{i+1} = Q_{i+1}^1V_{i+1}, \]

where the \( n \times k \)-matrix \( Q_{i+1}^1 \) has orthogonal columns and \( V_{i+1} \) is a \( k \times k \)-uppertriangular matrix. Now if we denote (cf (2.10))

(2.23) \[ d_{i+1}^1 = (Q_{i+1})^T[w_i(t_{i+1}) - w_{i+1}(t_{i+1})], \]

then we obtain the following reduced uppertriangular recursion:

(2.24) \[ a_{i+1}^1 = V_{i+1}a_i^1 + d_{i+1}^1, \quad i = 0, \ldots, N-1. \]

**Remark 2.25**

Since we choose \( w_{i+1}(t_{i+1}) \) orthogonal to \( \text{span}(F_{i+1}(t_{i+1})) = \text{span}(Q_{i+1}^1) \), we see that we actually can simplify (2.23) to

(2.26) \[ d_{i+1}^1 = (Q_{i+1}^1)^T w_i(t_{i+1}). \]

**Remark 2.27**

\( w_{i+1}(t_{i+1}) \) is uniquely determined by the requirements (2.20). We apparently should project \( w_i(t_{i+1}) \) onto \( \text{span}(Q_{i+1}^1) \) and subtract this from \( w_i(t_{i+1}) \). Hence we find:

(2.28) \[ w_{i+1}(t_{i+1}) = w_i(t_{i+1}) - Q_{i+1}^1 (Q_{i+1}^1)^T w_i(t_{i+1}). \]

The computation of the \( a_i^1 \) from the BC is done in a similar way as in the preceding subsection; we compute a fundamental solution \( \{ \varphi_i^1 \}_{i=0}^N \) and a particular solution \( \{ z_{i, i=0}^1 \} \) of (2.24). Since for some \( k \)-vector \( c_i^1 \) there must hold:

(2.29) \[ a_i^1 = \varphi_i^1 c_i^1 + z_{i}^1, \]
we obtain the desired solution from

\[(2.30) \quad x(t_i) = w_i(t_i) + Q^i_1(z^i_1 + Q^i_1c^i).\]

After substituting \(x(t_0) = x(\alpha)\) and \(x(t_N) = x(\beta)\) in the BC (2.17a) we thus find \(c^i\) from

\[(2.31) \quad \left[ M_0^i Q^i_0 + M_1^i Q^i_1 \right] c^i = b^i - M_0^i w_0(\alpha) - M_1^i w_{N-1}(\beta) - M_0^i Q^i_0 z^i_0 - M_1^i Q^i_1 z^i_1.
\]

**Remarks 2.32**

(i) If the ODE is homogeneous we still have to compute solutions \(w_i\) (but now of the homogeneous ODE) such that (2.19b) is satisfied.

(ii) If the ODE is homogeneous and moreover \(b^2 = 0\), then we can skip the computation of \(w_i\) and put \(d_i = 0\) for all \(i\). In such a case we have to replace (2.21), (2.24), (2.29), (2.30) and (2.31) by

\[(2.21)^1 \quad x(t) = F^1_i(t)a^1_i,\]

\[(2.24)^1 \quad a^1_{i+1} = V_{i+1}a^1_i,\]

\[(2.29)^1 \quad a^1_i = \Phi^1_i c^1,\]

\[(2.30)^1 \quad x(t_i) = w_i(t_i) + Q^i_1c^i,\]

\[(2.31)^1 \quad \left[ M_0^i Q^i_0 + M_1^i \right] c^i = b^1,\]

respectively.

§ 2.3 BVP with (completely) Separated BC

Quite often the BC are even simpler than in (2.18), viz if \(M_0^i = 0\) as well. We call this (completely) separated BC. So we have

\[(2.33a) \quad M_0^\beta x(\beta) = b^1,\]

\[(2.33b) \quad M_\alpha x(\alpha) = b^2,\]

where \(M_\beta\) is an \(k \times n\)-matrix and \(M_\alpha\) is an \((n-k) \times n\)-matrix.

We can use a similar approach as in § 2.2. However (2.29) until (2.31) are not needed. Indeed, as can be expected we have an explicit terminal value for the recursion (2.24) to compute the sequence \(\{a^1_0, \ldots, a^1_N\}\). From (2.21) we derive
After substitution in (2.33) we obtain
\[
(2.35) \quad 1M_P Q_1^I a_1^I = b_1^I - 1M_P w_{N-1}(t_N).
\]

Remark 2.36

The same remark as 2.32 applies to the separated case, i.e. if the problem is homogeneous and \( b_2^2 = 0 \), we skip the computation of the \( \{ w_i \} \) and \( \{ z_i^I \} \).
Instead of (2.34) and (2.35) we then have respectively
\[
(2.34)_1 \quad x(t_1) = Q_1^I a_1^I,
\]
\[
(2.35)_1 \quad 1M_P Q_1^I a_1^I = b_1^I.
\]
§ 3. Special features of the methods

There are several aspects which makes MULS different from other Multiple Shooting strategies. In the following subsections we shall describe some of them. This may help to understand the power and also the limitations of the method.

§ 3.1 Numerical realization of the integration

Since the numerical integration accounts for the bulk of the computational labour, it is of fairly great importance to have this done efficiently. A first gain can be achieved quite simply. Realizing that the unstable solutions will inevitably dictate the stepsize if an absolute tolerance is given (and won't do for less if a relative tolerance is required), we need only to use the adaptive integration control for one solution on each subinterval. The other solutions are found on the thus determined grid. In MULS the grid is determined by the particular solution \( w_i \), or, if the problem is homogeneous, by the first column of \( F \) (or \( F_i \)). The latter choice is induced by the desire to have points such that the most unstable solution is still integrated correctly (i.e. up to the required tolerance). See also [6].

§ 3.2 Computation of a fundamental solution and a particular solution of the Multiple Shooting recursion

For solving the BVP with general BC or partially separated BC we have to compute a fundamental solution and a particular solution of recursions (2.11) and (2.24) respectively. As both recursions are of the same nature, we only discuss recursion (2.11).

The important idea behind the decoupling method of § 2 is that in well posed linear BVP, the homogeneous solution space of (2.1) is dichotomic, i.e. is such that for some \( k \) (k-partitioning) there exist a \( k \)-dimensional subspace of increasing solutions and an \((n-k)\)-dimensional subspace of non-increasing solutions. Using this property and starting with a proper \( Q_0 (= F_0 (t_0)) \), we can compute a set of \( U_i \) for which the first \( k \) columns represent the subspace of increasing solutions and the last \((n-k)\) columns the subspace of the non-increasing solutions. In this way we have decoupled the increasing solutions and the non-increasing solutions. This decoupling enables us to compute a fundamental solution of the uppertriangular recursion (2.11) in a stable way as follows:

Let \( k \) be the dimension of the subspace of increasing solutions. Then we partition matrices and vectors as

\[
(3.1) \quad U_i = \begin{bmatrix} B_i & C_i \\ 0 & E_i \end{bmatrix}, \quad a_i = \begin{bmatrix} a_i^1 \\ a_i^2 \end{bmatrix},
\]
where \( B_i \) is a \( k \times k \)-uppertriangular matrix, \( E_i \) an \( (n-k_p) \times (n-k_p) \)-uppertriangular matrix, \( C_i \) a \( (k_p \times (n-k_p)) \)-matrix, \( a_1^i \) a \( k \)-vector and \( a_2^i \) an \( (n-k_p) \)-vector.

The recursion (2.11) can be rewritten as:

\[
\begin{align*}
(3.2a) & \quad a_{i+1}^{2} = E_{i+1}a_{i}^{2} + d_{i+1}^{2} \\
(3.2b) & \quad a_{i+1}^{1} = B_{i+1}a_{i}^{1} + C_{i+1}a_{i}^{2} + d_{i+1}^{1} .
\end{align*}
\]

As the \( B_i \) represent the increasing solutions, the absolute value of the diagonal of \( B_i \) can be expected to be greater than 1, making forward computation of (3.2b) unstable. The \( E_i \) represent the non-increasing solutions, so the absolute value of the diagonal elements of \( E_i \) can be expected to be less than or equal to 1, making forward computation of (3.2a) stable. Hence the obvious strategy for computing a fundamental solution \( \{ \varphi_i \}_{i=0}^{N} \) and a particular solution \( \{ z_i \}_{i=0}^{N} \) of recursion (2.11) is to use (3.2a) in forward direction and (3.2b) in backward direction. So for the particular solution \( \{ z_i \}_{i=0}^{N} \) we have the BC

\[
(3.3) \quad z_0^2 = 0 \quad , \quad z_N^1 = 0 .
\]

Then \( z_i^2 \), \( i=1,2,\ldots,N \), using (3.2a) in forward direction, and \( z_i^1 \), \( i=N-1,N-2,\ldots,0 \) using (3.2b) in backward direction, is computed.

For the fundamental solution we have the recursion

\[
\begin{align*}
(3.4a) & \quad \varphi_{i+1}^{2} = E_{i+1}\varphi_{i}^{2} \\
(3.4b) & \quad \varphi_{i+1}^{1} = B_{i+1}\varphi_{i}^{1} + C_{i+1}\varphi_{i}^{2} + d_{i+1}^{1} ,
\end{align*}
\]

and the BC

\[
(3.5) \quad \varphi_0^{2} = ( O \mid I ) \quad ; \quad \varphi_N^{1} = ( I \mid 0 ) .
\]

Now \( \{ \varphi_i^2 \}_{i=0}^{N} \) is computed via (3.4a) and \( \{ \varphi_i^1 \}_{i=0}^{N} \) is then computed via (3.4b).

§ 3.3 Choosing \( Q_0 \) and \( w_1(t_1) \).

As in fact the matrix \( Q_0 \) generates the sequences of \( \{ Q_i \} \) and \( \{ U_i \} \) it is important to have a proper choice for \( Q_0 \). Indeed as was shown in [3] the desired splitting of the solution space into increasing and non-increasing solutions may not be achieved for general initial matrices \( Q_0 \), though in practice it is most likely that an arbitrary choice will do eventually. Nevertheless for a good stability of the recursion some effort to obtain a good guess is worth paying for. For general BC no information about \( k_p \) nor the direction of the increasing solutions is available, so we just take \( Q_0 = I \). If, after a few

-- 10 --
normalizations, a disorder of eigenvalues of the matrices $U_1$ becomes visible. If we perform a permutation of the columns of $Q_0$ to hopefully restore an ordering in decreasing absolute magnitude. If needed this process is repeated a finite number of times. In §4.3 we return to this.

If the BC are partially separated, one has to realize that $k_S$ and $k_P$ may be different ($k_S > k_P$). Hence, in general one should try to obtain an ordering of the diagonal elements of the $V_1$, at least to such an extent that the $k_P x k_P$ upper part contains the eigenvalues which are in absolute value greater than 1; of course this can only be found by guessing and correcting as in the general case.

Finally, if the BC are completely separated we necessarily have that $k_S = k_P$ (or at least a reasonable choice of $k_S$ if there is no exponential but only an ordinary dichotomy). For this, however, we presuppose the problem to be well conditioned, which will be explained in the next subsection. As far as the $w_i(t_1)$ are concerned, we already remarked that they were in fact determined by our desire to keep $w_i(t_1)$ in the same linear variety as $w_{i-1}(t_1)$. Of course this only makes sense in case the BC are (partially or completely) separated. If we use the strategy for general BC we have a complete freedom again. We have chosen for the option $w_i(t_1) = 0$ because, in general, this gives $O(1)$ components of all solutions involved, notably the desired particular one and the most unstable one. It was discussed in [6] that this was a sensible choice.

§ 3.4 Conditioning and stability.

The accuracy of the solution $x(t)$ of a BVP, using the method as described in §2, depends on:

(i) The accuracy by which the fundamental solution $F_i(t_1)$ and the particular solution $w_i(t_1)$ are computed. (This accuracy is determined by the user.)

(ii) The accuracy by which the vector $c$ in equation (2.16) is computed.

(iii) The accuracy by which the fundamental solution $\{\phi_i\}_{i=0}^N$ or $\{\phi'_i\}_{i=0}^N$ and the particular solution $\{z_i\}_{i=0}^N$ or $\{z'_i\}_{i=0}^N$ of the recursion (2.11) and (2.24) respectively is computed.

First we will discuss point (ii).

Since (2.16) resulted from the boundary conditions we have to investigate the effect of perturbations in the BC on the computed solution. Suppose we have a BC with a perturbed right-hand-side, i.e. instead of (2.2) we have

\[
(3.7) \quad M_a \hat{x}(\alpha) + M_b \hat{x}(\beta) = b + \delta b
\]

As $x$ and $\hat{x}$ are both solutions of the ODE of the BVP there exists a vector $v$ such that

\[
(3.8) \quad \hat{x}(t) - x(t) = F(t)v
\]
where \( F(t) \) is a fundamental solution.

Subtracting (3.2) from (3.7) and using (3.8) we obtain:

\[
(3.9) \quad \left[ M_\alpha F(\alpha) + M_\beta F(\beta) \right] y = \delta b 
\]

So we have

\[
(3.10) \quad \tilde{x}(t) - x(t) = F(t) \left[ M_\alpha F(\alpha) + M_\beta F(\beta) \right]^{-1} \delta b 
\]

and

\[
(3.11) \quad \max_{t \in (\alpha, \beta)} \| \tilde{x}(t) - x(t) \| < \max_{t \in (\alpha, \beta)} \| F(t) \| \left\| M_\alpha F(\alpha) + M_\beta F(\beta) \right\|^{-1} \\| \delta b \|. 
\]

Therefore we define a condition number \( CN \) of a BVP as:

\[
(3.12) \quad CN = \max_{t \in (\alpha, \beta)} \| F(t) \left\| M_\alpha F(\alpha) + M_\beta F(\beta) \right\|^{-1} \| . 
\]

( Notice that \( CN \) is independent of the fundamental solution \( F \), as for any other fundamental solution \( G(t) \), say, there is a constant matrix \( P \) such that \( G(t) = F(t)P \).)

As is shown in [7] if \( \{ \Phi \} \) is defined as in (3.4), then an estimate of \( CN \) is given by:

\[
(3.13) \quad \kappa = \left\| \left[ M_\alpha Q_0 Q_0^T + M_\beta Q_N Q_N^T \right]^{-1} \right\| < 2CN. 
\]

Basically the information to compute \( \kappa \) is available (cf (2.16)). However when the BVP has (partially) separated BC, only \( k_s \) (< \( n \)) columns of \( Q_0, Q_N, \Phi_0, \Phi_N \) are computed. The separated BC can be written as:

\[
(3.14) \quad x(\alpha) + \begin{bmatrix} 1 M_\alpha \\ \vdots \end{bmatrix} x(\beta) = \begin{bmatrix} b^1 \\ b^2 \end{bmatrix}. 
\]

For the condition number \( CN \) we have:

\[
(3.15) \quad CN = \max_{t \in (\alpha, \beta)} \| F(t) \left[ \begin{bmatrix} 1 M_\alpha \\ \vdots \end{bmatrix} F'(a) F'(\beta) + \begin{bmatrix} 1 M_\beta \\ \vdots \end{bmatrix} F'(\beta) F'(\beta) \right]^{-1} \| 
\]

or

\[
(3.15) \quad CN = \max_{t \in (\alpha, \beta)} \| F(t) \left[ \begin{bmatrix} 1 M_\alpha F'(a) + 1 M_\beta F'(\beta) & 1 M_\alpha F^2(a) + 1 M_\beta F^2(\beta) \\ \vdots \end{bmatrix} \right]^{-1} \|. 
\]
As \(CN\) is independent of \(F\) and we have taken \(F\) such that \(2M_F^\alpha(a) = 0\), it is easy to see that if either \([1M_F^\alpha(a) + 1M_F^\beta(\beta)]\) or \(2M_F^2(a)\) is ill-conditioned also the BVP will be ill-conditioned. Hence we compute

\[
\kappa_1 = \| [1M_F^\alpha(a) + 1M_F^\beta(\beta)]^{-1} \|_1 ,
\]

\[
\kappa_2 = \| 2M_F^2(a) \|_1
\]

Although a large \(\kappa_1\) or a large \(\kappa_2\) indicates that the BVP is ill-conditioned it is possible to have an ill-conditioned BVP for which both \(\kappa_1\) and \(\kappa_2\) are of order one. For well-conditioned BVP with separated BC it is necessary that \(F^2\) contains only non-growing modes (in case of completely separated BC, all non-growing modes). To find out whether \(F^2(a)\) would result in computing a growing solution, we recall that for the solution \(x\) we had (cf. (2.29))

\[
x(t) = F^1(t)c^1 + w(t) ,
\]

and completing \(F^1\) to a fundamental solution \(F = (F^1|F^2)\) we thus see that

\[
(3.16a) \quad x(t) = F^2(t)c^2 + z(t) ,
\]

where \(z\) is a particular solution of the ODE of the BVP and \(c^2\) an \((n-k_\beta)\)-vector. Supposing that \(z\) is a smooth solution, a dominant mode in \(F^2\) will influence the growth of \(w(t)\), unless \(c^2 = 0\). However, by computing another particular solution \(v(t)\) say, where

\[
(3.16b) \quad v(t) = F^2(t)e^2 + w(t) , \quad e^2 \neq 0 ,
\]

and thus

\[
(3.16c) \quad w(t) - v(t) = F^2(t)e^2 ,
\]

we have a way to find out whether \(F^2(t)\) contains dominant modes or not (see § 4.4).

For BVP with a dichotomic solution space we have the recursion (cf. (3.2)).

\[
(3.19a) \quad a_{i+1} = E_{i+1}a_i + d_{i+1} ,
\]

\[
(3.19b) \quad a_i = B_{i+1}(a_i - c_{i+1}a_i^2 - d_{i+1}) \quad i=0,\ldots,N-1.
\]

To investigate the stability of (3.19) we examine the effects of additive perturbations \(\{p_i^2\}\) and \(\{p_i^1\}\) of respectively (3.19a) and (3.19b), i.e. suppose \(\{a_i^1\}\) and \(\{a_i^2\}\) satisfy

\[
(3.20a) \quad a_{i+1} = E_{i+1}a_i^1 + d_{i+1}^2 + p_{i+1}^2 ,
\]
(3.20b) \[ \hat{s}_i^1 = B_{i+1}^{-1}(\hat{s}_{i+1}^1 - C_{i+1}\hat{s}_i^2 - \hat{s}_i^1) + p_i^1. \]

Then for \[ \hat{g}_i^{1+1} = \hat{a}_i^{1+1} - a_i^{1+1}, \quad \hat{g}_i^{2+1} = \hat{a}_i^{2+1} - a_i^{2+1} \] we have:

(3.21a) \[ \hat{g}_i^{2+1} = E_{i+1}g_i^{1+1} + p_{i+1}^2; \quad \hat{g}_0^{2+1} = p_0^2, \]

(3.21b) \[ \hat{g}_i^1 = B_{i+1}(g_i^{1+1} - C_{i+1}\hat{g}_i^2) + p_i^1; \quad \hat{g}_N^1 = p_N^1, \]

which results in:

(3.22a) \[ \hat{g}_i^2 = \sum_{l=0}^{i} \left[ (\prod_{j=1}^{i} E_j) p_l^2 \right], \]

(3.22b) \[ \hat{g}_i^1 = \sum_{l=0}^{i} \left[ Q_{i,N}(\prod_{j=1}^{i} E_j) p_l^2 \right] + \sum_{l=1}^{N-1} \left[ \left( \prod_{j=1}^{i} B_j \right)^{-1} Q_{l,N} p_l^2 \right] + \sum_{l=i+1}^{N} \left[ \left( \prod_{j=1}^{i} B_j \right)^{-1} p_l^1 \right], \]

where \( Q_{m,q} \) is a shorter notation for

(3.23a) \[ Q_{m,q} = \sum_{l=m+1}^{q} \left[ \left( \prod_{j=m+1}^{q} B_j \right)^{-1} C_l (\prod_{j=m+1}^{q} E_j) \right]. \]

where

(3.23b) \[ \sum_{l=m+1}^{q} \left[ \left( \prod_{j=m+1}^{q} B_j \right)^{-1} C_l (\prod_{j=m+1}^{q} E_j) \right] \]

If the permutations \( p_1^1, p_2^1 \) are of the same order, i.e. \( \| p_1^1 \| < \delta \), \( \| p_2^1 \| < \delta \) for some \( \delta \) we have:

(3.24a) \[ \| \hat{g}_i^2 \| < \left[ \sum_{l=0}^{i} \left( \prod_{j=l+1}^{i} E_j \right) \| \right] \delta. \]

(3.24b) \[ \| \hat{g}_i^1 \| < \left[ \sum_{l=0}^{i} \left( \prod_{j=l+1}^{i} E_j \right) \| \right] + \left( \sum_{l=1}^{N-1} \left( \prod_{j=1}^{l} B_j \right)^{-1} \| Q_{l,N} \| \right) \]

One easily checks that a proper dichotomy implies reasonably bounded \( \| Q_{m,p} \| \) as well as such bounds for \( \| E_j \| \) and \( \| B_j^{-1} \| \). This then establishes the stability of the computation of \( \Phi_{i=0}^{N} \) and \( a_{i=0}^{N} \).
§ 4. Computational aspects of the method

There are a number of aspects which have not been filled in yet. In this chapter we shall therefore treat some particular implementations as they are realized in MUTS.

§ 4.1 The use of RKF45

A very reliable and fairly inexpensive integrator is RKF45, written by L.F. Shampine and H.A. Watts, a Runge Kutta Fehlberg routine which uses fifth order estimates combined with fourth order approximation (cf. [1]). This routine is the working horse in our codes and as long as the system is not stiff (in the sense that there is high activity of some modes) we have found it to work very well indeed (cf [6]). We have changed the original routines to make that it only uses the combined fourth-fifth order integrator for the grid determining solution, see § 3.1. A special routine computes solutions on a given grid by the fifth order only. Another special feature is that it terminates the calculations if five consecutive new points are found. Then an orthogonalization of the solution is performed and a new cycle is started. This QU-decomposition is carried out with elementary hermitians (Householder's method). Rather than in the form \((AQ_i = ) Q_{i+1} U_{i+1}\) we obtain \(Q_i^T\) in factored form. It is obvious that we only need to evaluate the first \(k\) columns of \(Q_{i+1}\) if we have (partially) separated BC. In the next subsection we consider how this will work out in the global computations.

In the original routine RKF45 both a relative and an absolute tolerance has to be supplied. Because of the fact that for general BVP on finite intervals one is mainly interested in absolute accuracy and our strategy makes significant growth per shooting interval unlikely anyway, we recommend to set the relative tolerance sufficiently smaller than the absolute tolerance.

§ 4.2 The choice of shooting points

The idea to have shooting intervals consisting of 5 steps only was induced by considerations of optimal efficiency, cf [6]. It is obvious that this strategy may give many more points for output than is needed by the user. Therefore a special device takes care of assembling these so called minor shooting intervals to major shooting intervals; the latter are such that the initial and terminal points coincide with user requested output points. Here another powerful feature of the decoupling method is revealed. Because of the fact that the \(k\)-partitioning \((k_p)\) coincides with the decoupling into increasing and decreasing modes, forward assembling of increments on minor intervals is relatively stable. Such an assembly may be described as follows: Let \(t_i\) be the initial point of a major shooting interval. Define:

\[
W_0 := U_{i,j} ; \quad G_j := s_j
\]

Now compute for \(s = 1, 2, \ldots\)

\[
W_s := U_{i,j+s} W_{s-1} ; \quad G_s := U_{i,j+s} G_{s-1} + s_{i,j+s}
\]
If \( s \) is large enough, then \( W_s \) describes the increment on the major interval \([t_{i,j}, t_{i,j+1}]\) and \( F_s \) the forcing on that interval, so that

\[
(4.3) \quad a_{i,j}^{j+1} = W_s a_{i,j}^j + F_s
\]

(of course \( s \) is a local index only).

Now we have three possible options for the output points:

(i) choose \( s \) such that \( \|W_s\| < \rho \), \( \rho \) prescribed.

(ii) choose \( s \) such that \( |t_{i,j}^{j+1} - t_{i,j}^j| = \frac{\beta - \alpha}{N} \).

\( (N \) the number of intervals). 

(iii) choose \( s \) such that \( t_{i,j}^{j+1} \) equals the first next specified output point.

Remark 4.4

Of course, it may be that these criteria above need shorter minor shooting intervals at the end of the major shooting interval. This is taken care of by MUTS.

Remark 4.5

Criterion (i) is of interest if one suspects the maximal incremental growth to be changing on \([\alpha, \beta]\) and likes to monitor this so that the solution is equidistributed with respect to this. However, one should realize that it may lead to (undesirably) large intervals if there are mildly growing solutions only.

§ 4.3 The computation of \( Q_0 \) and \( Q_1 \) and the proper splitting.

Suppose we find that the diagonal of the matrix \( U_1 \) is not ordered. Then we use a permutation matrix \( P \), which permutes the columns of \( U_1 \) according to the ordering of the absolute value of these diagonal elements. Of course \( U_1 P \) is no longer upper triangular, so we perform another QU-decomposition, i.e.

\[
(4.6) \quad U_1^{(old)} P = RU_1^{(new)} .
\]

The matrix \( U_1^{(new)} \) replaces \( U_1^{(old)} \), whilst \( Q_0^{(old)} \) is replaced by

\[
(4.7) \quad Q_0^{(new)} := Q_0^{(old)} P
\]

and \( Q_1 \) by
If \( U_1 \) is still not found in order we repeat this procedure. In fact we do the same with the assembled product \( U_s U_{s-1} \ldots U_1 \) on the first major shooting interval. On subsequent major intervals this reordering is no longer feasible. One should realize that neat problems have to be dichotomic (cf. [2]), i.e. after reaching the endpoint of the first major interval, we should have a good idea of \( k_p \). Indeed MUTS chooses \( k_p \) equal to the position of that diagonal element of \( U_2 \) which is the smallest one (in absolute value) being larger than 1. Of course this only makes sense for an ordered diagonal. Although it is expected to be ordered in general, there might be situations where this is not the case. Therefore a global check on the increment on the whole interval \([\alpha, \beta]\) is made. If the ordering is not found satisfactory, a global reordering is performed using permutation matrices according to this. In fact this is rather cheap as it only requires matrix-matrix multiplications plus one LU-decomposition at each outputpoint. This process is moreover stable if the norm of the assembled matrices does not outgrow \( TOL/EPS \), where \( TOL \) is the absolute tolerance and \( EPS \) the machine constant.

If for some reason reordering cannot be performed after \( n \) permutations \( IERROR = 230 \) is given.

If the BC are (partially) separated we have to determine a \( Q^1_0 \) such that

\[
2M^\alpha_0 Q^1_0 = 0 \quad (\text{cf} \ (2.19a))
\]

This can conveniently be done as follows:

Compute elementary hermiteans \( P_1, \ldots, P_n \) such that

\[
(4.9) \quad R := P_n \ldots P_2 2M^\alpha_0
\]

is uppertriangular. Now take \( Q^1_0 \) as the last \( k_s \) columns of

\[
(4.10) \quad Q^1_0 = P_4 \ldots P_n
\]

(It is easily seen that this results in the desired matrix as

\[
\text{span}(2M^\alpha_0) = \text{span}(P_4 \ldots P_{n-k_s} \begin{bmatrix} I_{n-k_s} \\ 0 \end{bmatrix}).
\]

Sometimes it is not clear beforehand whether \( \text{rank}(M_\alpha) < n \) or \( \text{rank}(M_\beta) < n \).

(Note that when \( M_\beta \) has some zero rows, say \( n-k_s \), \( \text{rank}(M_\beta) \) may be smaller than \( k_s \).) In such a case we may invoke the singular decomposition (SVD) of these matrices to determine the numerical rank. So consider

\[
(4.11) \quad M_\alpha = U_\alpha \Sigma_\alpha V_\alpha^T, \quad M_\beta = U_\beta \Sigma_\beta V_\beta^T
\]

where \( U_\alpha, V_\alpha, U_\beta, V_\beta \) are orthogonal matrices and \( \Sigma_\alpha, \Sigma_\beta \) diagonal matrices.

Suppose \( \Sigma_\alpha \) has \( k_{s1} \) non zero diagonal elements and \( \Sigma_\beta \) has \( k_{s2} \) non zero diagonal elements. If both \( k_{s1} = k_{s2} = n \) we do not have separated BC. If \( k_{s2} < n \) we have
So multiplying (2.2) by $U_{\beta}^T$ we obtain

$$U_{\beta}^T M_{\beta} x(\alpha) + U_{\beta}^T M_{\beta} x(\beta) = U_{\beta}^T b,$$

which, denoting $U_{\beta}^T M_{\alpha} = \bar{M}_{\alpha}$, $U_{\beta}^T M_{\beta} = \bar{M}_{\beta}$, $U_{\beta}^T b = \bar{b}$, can be written as

$$1\bar{M}_{\alpha} x(\alpha) + 1\bar{M}_{\beta} x(\beta) = \bar{b}^1$$
$$2\bar{M}_{\alpha} x(\alpha) = \bar{b}^2.$$

This is the form of (2.18). Of course it may be that $k_{s1} < k_{s2}$, in which case it would be more profitable to regard the BVP as a problem on $[\beta,\alpha]$, instead of on $[\alpha,\beta]$. Therefore we compute both the SVD of $M_{\alpha}$ and of $M_{\beta}$ and take the smallest of $k_{s1}$ and $k_{s2}$ with the corresponding initial and terminal points (i.e. either $[\alpha,\beta]$ or $[\beta,\alpha]$).

§ 4.4 The computation of the stability constants.

For general BC the quantity $\kappa$ in (3.13) can be computed without much additional effort, as we need a LU-decomposition anyway. As we remarked $\kappa$ is at most a factor 2 amiss in comparison with the actual condition number (cf. (3.13)). If the BC are (partially) separated we do not have all necessary information about the $E_i$ available. It may be even so that $\kappa_1$ and $\kappa_2$ (see (3.16) and (3.17)) are moderate since the ill-conditioning is concealed by the particular solution $w_i$. In order to detect this we also compute another sequence of particular solutions $\{v_i\}$ such that

$$v_0(t_0) = w_0(t_0) + f_0(t_0)e^2,$$
$$e^2 = \frac{1}{\sqrt{n-k_s}}(1,1,\ldots,1)^T.$$

Then a $\kappa_3$ is computed as:

$$\kappa_3 = \kappa_{\max_i} (\|w_i(t_1) - v_i(t_1)\|_2) .$$

As an estimate for the condition number $CN$ we now better take

$$\kappa = \max (\kappa_1, \kappa_2, \kappa_3) .$$

The user may find the $\kappa$ as an output parameter ER(4).

Of course it is possible that the matrices $[M_{\alpha} Q_{\alpha} M_{\alpha} + M_{\beta} Q_{\beta} M_{\beta}]$, $[1M_{\alpha} Q_{\alpha} + 1M_{\beta} Q_{\beta}]$ or $2M_{\alpha} Q_{\alpha}$ happen to be numerically singular. In that case
Apart from this condition number another quantity is of importance. In fact we need to compute the maximal value in norm of suitable Green's functions (cf. [3]). This is an almost impossible task and therefore we are satisfied with a somewhat heuristical estimate of them. Note that in (3.24) the magnitude of the quantities $||(\Pi E_j)||$ and $||(\Pi B_j^{-1})||$ may be blamed if the local errors are blown up significantly. Hence it makes sense to monitor the diagonal elements of the product matrices $E_p \ldots E_q$ and $B_p^{-1} \ldots B_q^{-1}$ for arbitrary $p$ and $q$ that they essentially reflect the growth of the basis solutions. Thinking of (3.24) we therefore also compute

$$(4.18) \quad A_f^2 = \max_k \left( \max_i \left( 1 + \sum_{l=1}^{i} \| E_j^k \| \right) \right),$$

where $E_j^k$ denotes the $k$-th diagonal element of $E_j$.

$$(4.19) \quad a_{f1}(k) = \max_i \left( 1 + \sum_{l=i+1}^{N} \left( \prod_{j=1}^{l-1} |B_j^k|^{-1} \right) \right),$$

$$(4.20) \quad a_{f2}(k) = \max_i \left( \prod_{l=1}^{N} |B_1^k|^{-1}, \ldots, \prod_{l=N-1}^{N} |B_{N-1}^k|^{-1} \right),$$

$$(4.21) \quad a_{f3}(k) = \max_i \left( \prod_{l=1}^{i} |E_1^k|, \ldots, \prod_{l=i-1}^{i} |E_{i-1}^k|, |E_i^k| \right),$$

$$(4.22) \quad A_f^1 = \max_k \left( a_{f1}(k) + a_{f2}(k) \times a_{f3}(k) \right),$$

where $B_j^k$ denotes the $k$-th diagonal element of $B_j$.

As an estimate of the amplification factor $A_f$ (being a bound for the Green's function in turn) we take:

$$(4.23) \quad A_f = \max \left( A_f^1, A_f^2 \right).$$

The user may find $A_f$ as an output parameter ER(5).

If $A_f$ is such that the global rounding error is larger than the discretization error, a warning error, IERROR = 240, is given.
References


§ 5 Documentation

Of all routines contained in BOUNDPAK only the following four are of direct importance for the user, concerning this chapter.

The double precision versions are:

DMUTSG      for two-point BVP with general BC,
DMUTSP      for two-point BVP with partially separated BC,
DMUTSS      for two-point BVP with completely separated BC,
DMUTS       for two-point BVP with arbitrary BC.

The single precision versions are SMUTSG, SMUTSP, SMUTSS and SMUTS respectively.

DMUTS (SMUTS) can be used for two-point BVP with general BC and for two-points BVP with separated BC. Moreover DMUTS (SMUTS) has the option to find out whether the BC are separated or not (cf. § 4.3). However, DMUTS (SMUTS) uses the IMSL routine LSVDF, so if the IMSL library is not available on your system DMUTS (SMUTS) cannot be used.

A complete list of the BOUNDPAK routines is given in the next section.

On the next pages one may find the documentation for the routines DMUTSG, DMUTSP, DMUTSS and DMUTS.
**SPECIFICATION**

SUBROUTINE DMUTSG(FLIN, FDIF, N, IHOM, A, B, MA, MB, BCV, AMP, ER, NRTI, TI,  
1 NTI, X, U, NU, Q, D, KPART, PHIREC, W, LW, IW, LIW, IERROR)

C INTEGER N, IHOM, NRTI, NTI, NU, LW, IW(1), LIW, IERROR
C DOUBLE PRECISION A, B, MA(N, N), MB(N, N), BCV(N), AMP, ER(5), TI(NTI),
C 1 X(N, NTI), U(NU, NTI), Q(N, N, NTI), D(N, NTI),
C 2 PHIREC(NU, NTI), W(LW)
C EXTERNAL FLIN, FDIF

**Purpose**

DMUTSG solves the two-point BVP:

\[ \frac{dx(t)}{dt} = L(t)x(t) + r(t) \quad , \quad A < t < B \text{ or } B < t < A , \]

with BC:

\[ MBAx(A) + MBx(B) = BCV \]

where \( MA \) and \( MB \) are the BC matrices and \( BCV \) the BC vector.

**Parameters**

**FLIN** SUBROUTINE, supplied by the user with specification:

SUBROUTINE FLIN(T, X, F)
DOUBLE PRECISION T, X(N), F(N)

where \( N \) is the order of the system. FLIN must evaluate the homogeneous part of the differential equation, \( L(t)x(t) \), for \( t=T \) and \( x(t)=X \) and place the result in \( F(1), F(2), \ldots, F(N) \).

FLIN must be declared as EXTERNAL in the (sub)program from which DMUTSG is called.

**FDIF** SUBROUTINE, supplied by the user, with specification:

SUBROUTINE FDIF(T, X, F)
DOUBLE PRECISION T, X(N), F(N)

where \( N \) is the order of the system. FDIF must evaluate the righthand-side of the inhomogeneous differential equation, \( L(t)x(t) + r(t) \), for \( t=T \) and \( x(t)=X \) and place the result in \( F(1), F(2), \ldots, F(N) \).

FDIF must be declared as EXTERNAL in the (sub)program from which DMUTSG is called.

In the case that the system is homogeneous FDIF is the same as FLIN.
N INTEGER, the order of the system.
Unchanged on exit.

IHOM INTEGER.
IHOM indicates whether the system is homogeneous or inhomogeneous.
IHOM = 0 : the system is homogeneous,
IHOM = 1 : the system is inhomogeneous.
Unchanged on exit.

A,B DOUBLE PRECISION, the two boundary points.
Unchanged on exit.

MA,MB DOUBLE PRECISION array of dimension (N,N).
On entry : MA and MB must contain the matrices in the BC:
M_A x(A) + M_B x(B) = BCV.
Unchanged on exit.

BCV DOUBLE PRECISION array of dimension (N).
On entry BCV must contain the BC vector.
Unchanged on exit.

AMP DOUBLE PRECISION.
On entry AMP must contain the allowed incremental factor of the homo­
geneous solutions.
AMP should be greater than 1, if not the subroutine will change AMP
into max(ER(1),ER(2)) / ER(3). If NRTI > 0, AMP is a dummy parameter.

ER DOUBLE PRECISION array of dimension (5).
On entry ER(1) must contain a relative tolerance for solving the dif­
fferential equation. If the relative tolerance is smaller then 1.0 e-12
the subroutine will change ER(1) into
1.E-12 + 2 * ER(3).
On entry ER(2) must contain an absolute tolerance for solving the dif­
fferential equation.
On exit ER(2) and ER(3) are unchanged.
On exit ER(4) contains an estimation of the condition number of the
BVP.
On exit ER(5) contains an estimated error amplification factor.

NRTI INTEGER.
On entry:
NRTI = 0, in this case the subroutine determine automatically the
output-points using AMP.
NRTI = 1, in this case the output-points are supplied by the user in
the array TI.
NRTI > 1, in this case the subroutine computes the output-points TI(k)
by:
TI(k) = A + (k-1) * (B - A) / NRTI ;
so TI(1) = A and TI(NRTI+1) = B .
On exit NRTI contains the total number of output-points.

TI DOUBLE PRECISION array of dimension (NRTI).
On entry: if NRTI = 1 , TI must contain the required output-points in
strict monotone order: A=TI(1) < ... < TI(1)=B or A=TI(1) > ... >
TI(l)=B (l denotes the total number of required output-points).
On exit: TI(i), i=1,2,...,NRTI, contains the output-points.
DMUTSG

NTI INTEGER.
NTI is the dimension of TI and one of the dimensions of the arrays X, U, Q, D, PHIREC. NTI must be greater than the total number of output-points + 3.
Unchanged on exit.

X DOUBLE PRECISION array of dimension (N,NTI).
On exit X(i,k), i=1,2,...,N contains the solution of the BVP at the output-point TI(k), k=1,...,NRTI.

U DOUBLE PRECISION array of dimension (NU,NTI).
On exit U(i,k) i=1,2,...,NU contains the relevant elements of the uppertriangular matrix U_k, k=2,...,NRTI. The elements are stored column wise, the jth column of U_k is stored in U(nj+1,k), U(nj+2,k),...,U(nj+j,k), where nj = (j-1) * j / 2.

NU INTEGER.
NU is one of the dimensions of U and PHIREC.
NU must be at least equal to N * (N+1) / 2.
Unchanged on exit.

Q DOUBLE PRECISION array of dimension (N,N,NTI).
On exit Q(i,j,k) i=1,2,...,N, j=1,2,...,N contains the N columns of the orthogonal matrix Q_k, k=1,...,NRTI.

D DOUBLE PRECISION array of dimension (N,NTI).
If IHOM = 0 the array D has no real use and the user is recommended to use the same array for the X and the D.
If IHOM = 1 : on exit D(i,k) i=1,2,...,N contains the inhomogeneous term d_k, k=1,2,...,NRTI, of the multiple shooting recursion.

KPART INTEGER.
On exit KPART contains the global k-partition of the uppertriangular matrices U_k.

PHIREC DOUBLE PRECISION array of dimension (NU,NTI).
On exit PHIREC contains a fundamental solution of the multiple shooting recursion. The fundamental solution is uppertriangular and is stored in the same way as the U_k.

W DOUBLE PRECISION array of dimension (LW).
Used as work space.

LW INTEGER
LW is the dimension of W. LW > 8*N + 2*N*N.
Unchanged on exit.

IW INTEGER array of dimension (LIW)
Used as work space.

LIW INTEGER
LIW is the dimension of IW. LIW > 3*N.
Unchanged on exit.

IERROR INTEGER
Error indicator; IERROR = 0 then there are no errors detected.
Error indicators

Errors detected by the subroutine

IERROR = 0 No errors detected.

IERROR = 100 INPUT ERROR: either N < 2 or IHOM < 0 or NRTI < 0 or NTI < 5 or NU < N * (N+1) / 2 or A=B.
TERMINAL ERROR.

IERROR = 101 INPUT ERROR: either ER(1) or ER(2) or ER(3) is negative.
TERMINAL ERROR.

IERROR = 103 INPUT ERROR: either LW < 8*N + 2*N*N or LIW < 3*N.
TERMINAL ERROR.

IERROR = 120 INPUT ERROR: the routine was called with NRTI = 1, but the given output-points in the array TI are not in strict monotone order.
TERMINAL ERROR.

IERROR = 121 INPUT ERROR: the routine was called with NRTI = 1, but the first given output-point or the last output-point is not equal to A or B.
TERMINAL ERROR.

IERROR = 122 INPUT ERROR: the value of NTI is too small; the number of output-points is greater than NTI-3.
TERMINAL ERROR.

IERROR = 200 This indicates that there is a minor shooting interval on which the incremental growth is greater than the AMP. The cause of this error lies in the used method for computing the fundamental solution.
WARNING ERROR.

IERROR = 213 This indicates that the relative tolerance was too small. The subroutine has changed it into a suitable value.
WARNING ERROR.

IERROR = 215 This indicates that during integration the particular solution or a homogeneous solution has vanished, making a pure relative error test impossible. Must use non-zero absolute tolerance to continue.
TERMINAL ERROR.

IERROR = 216 This indicates that during integration the requested accuracy could not be achieved. User must increase error tolerance.
TERMINAL ERROR.

IERROR = 218 This indicates that the input parameter N <= 0, or that either the relative tolerance or the absolute tolerance is negative.
TERMINAL ERROR.

-- 25 --
DMUTSG

IERROR = 240 This indicates that the global error is probably larger than the error tolerance due to instabilities in the system. Most likely the problem is ill-conditioned. Output value is the estimated error amplification factor.

WARNING ERROR.

IERROR = 250 This indicates that one of the $U_k$ is singular.

TERMINAL ERROR.

IERROR = 260 This indicates that the problem is probably too ill-conditioned with respect to the BC.

TERMINAL ERROR.

***************
Auxiliary Routines
***************

This routine calls the BOUNDPAK library routines DDURI, DDURT, DGTUR, DKPCH, DFUNRC, DFSR, DSVP.

***************
Remarks
***************

DMUTSG is written by G.W.M. Staarink and R.M.M. Mattheij.
Last update: 01-01-1986.

***************
Method
***************

See chapter I of BOUNDPAK User's Manual

***************
Example of the use of DMUTSG
***************

Consider the ordinary differential equation

$$\frac{dx(t)}{dt} = L(t) x(t) + r(t) , \quad 0 \leq t \leq 6$$

and a boundary condition $M_0 x(0) + M_1 x(6) = C$ with

$$L(2t) = \begin{bmatrix}
1 - 2\cos(t) & 0 & 1 + 2\sin(2t)\\
0 & 2 & 0 \\
-1 - 2\sin(2t) & 0 & 1 + 2\cos(2t)
\end{bmatrix}$$

$$r(t) = \begin{bmatrix}
(-1 + 2\cos(2t) - 2\sin(2t))e^t \\
- e^t \\
(1 - 2\cos(2t) - 2\sin(t))e^t
\end{bmatrix} , \quad C = \begin{bmatrix}
1 + e^6 \\
1 + e^6 \\
1 + e^6
\end{bmatrix}$$

and $M_A = M_B = I$. 

-- 26 --
The solution of this problem is: \( x(t) = (e^t, e^t, e^t)^T \).

In the next program the solution is computed and compared to the exact solution.  
This program has been run on a AS9000 VM/CMS computer.

```fortran
DOUBLE PRECISION A,B,MA(3,3),MB(3,3),BCV(3),AMP,ER(5),TI(15),
  X(3,15),U(6,15),Q(3,3,15),D(3,15),PHIREC(6,15),W(42),
2 INTEGER IW(9)
EXTERNAL FLIN,FDIF

C SETTING OF THE INPUT PARAMETERS
C
N = 3
IHOM = 1
ER(1) = 1.D-11
ER(2) = 1.D-6
ER(3) = 1.1D-16
NRTI = 10
NTI = 15
NU = 6
LW = 42
LIW = 9
A = 0.DO
B = 6.DO

C SETTING THE BC MATRICES MA AND MB
C
DO 1100 I = 1 , N
  DO 1000 J = 1 , N
    MA(I,J) = 0.DO
    MB(I,J) = 0.DO
  1000 CONTINUE
MA(I,I) = 1.DO
MB(I,I) = 1.DO
  1100 CONTINUE

C SETTING THE BC VECTOR BCV
C
BCV(1) = 1.DO + DEXP(6.DO)
BCV(2) = BCV(1)
BCV(3) = BCV(1)

C CALL DMUTSG
C
CALL DMUTSG(FLIN,FDIF,N,IHOM,A,B,MA,MB,BCV,AMP,ER,NRTI,NI,NTI,
  X,U,NU,Q,D,KPART,PHIREC,W,LW,LIW,IERRO)
IF ((IERRO.NE.0).AND.(IERRO.NE.200).AND.(IERRO.NE.213).AND.
  (IERRO.NE.240)) GOTO 5000

C COMPUTATION OF THE ABSOLUTE ERROR IN THE SOLUTION AND WRITING
C OF THE SOLUTION AT THE OUTPUTPOINTS
C
WRITE(6,200)
WRITE(6,190) ER(4),ER(5)
```

-- 27 --
WRITE(6,210)
WRITE(6,200)
DO 1500 K = 1 , NRTI
   EXSOL = DEXP(TI(K))
   AE = EXSOL - X(1,K)
   WRITE(6,220) K,TI(K),X(1,K),EXSOL,AE
DO 1300 I = 2 , N
   AE = EXSOL - X(I,K)
   WRITE(6,230) X(I,K),EXSOL,AE
1300 CONTINUE
1500 CONTINUE
STOP
5000 WRITE(6,300) !ERROR
STOP
C
190 FORMAT(' CONDITION NUMBER = ',D10.3,/,1 ' AMPLIFICATION FACTOR = ',D10.3,/
200 FORMAT(' ')
210 FORMAT(' I ',6X,'T',8X,'APPROX. SOL.',9X,'EXACT SOL.',8X,1 'ABS. ERROR')
220 FORMAT(' ',13X,3F7.4,3(3X,D16.9))
230 FORMAT(' ',13X,3(D16.9))
300 FORMAT(' TERMINAL ERROR IN DMUTSG: IERROR = ',I4)
C
END
C
SUBROUTINE FLIN(T,Y,F)
C ----------------------

DOUBLE PRECISION T,Y(3),F(3)
DOUBLE PRECISION TI,SI,CO

TI = 2.DO * T
SI = 2.DO * DSIN(TI)
CO = 2.DO * DCOS(TI)
F(1) = (1.DO - CO)* Y(1) + (1.DO + SI)* Y(3)
F(2) = 2.DO* Y(2)
F(3) = (-1.DO + SI)* Y(1) + (1.DO + CO)* Y(3)
RETURN
C END OF FLIN
END
C
SUBROUTINE FDIF(T,Y,F)
C ----------------------

DOUBLE PRECISION T,Y(3),F(3)
DOUBLE PRECISION TI,SI,CO

CALL FLIN(T,Y,F)
TI = 2.DO * T
SI = 2.DO * DSIN(TI)
CO = 2.DO * DCOS(TI)
TI = DEXP(T)
F(1) = F(1) + (-1.DO + CO - SI)*TI
F(2) = F(2) - TI
F(3) = F(3) + (1.DO - CO - SI)*TI
RETURN
C    END OF FDIF
    END

CONDITION NUMBER  =  0.133D+01
AMPLIFICATION FACTOR  =  0.221D+01

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>APPROX. SOL.</th>
<th>EXACT SOL.</th>
<th>ABS. ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0000</td>
<td>0.1000000001D+01</td>
<td>0.100000000D+01</td>
<td>-0.120756514D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.1000000001D+01</td>
<td>0.100000000D+01</td>
<td>-0.149754604D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.1000000001D+01</td>
<td>0.100000000D+01</td>
<td>-0.130719151D-07</td>
</tr>
<tr>
<td>2</td>
<td>0.6000</td>
<td>0.182211882D+01</td>
<td>0.182211880D+01</td>
<td>-0.230910355D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.182211882D+01</td>
<td>0.182211880D+01</td>
<td>-0.186150286D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.182211880D+01</td>
<td>0.182211880D+01</td>
<td>0.276479217D-08</td>
</tr>
<tr>
<td>3</td>
<td>1.2000</td>
<td>0.332011694D+01</td>
<td>0.332011692D+01</td>
<td>-0.162950000D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.332011695D+01</td>
<td>0.332011692D+01</td>
<td>-0.299702672D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.332011690D+01</td>
<td>0.332011692D+01</td>
<td>0.253190855D-07</td>
</tr>
<tr>
<td>4</td>
<td>1.8000</td>
<td>0.604964745D+01</td>
<td>0.604964746D+01</td>
<td>0.189447806D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.604964752D+01</td>
<td>0.604964746D+01</td>
<td>-0.521154062D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.604964743D+01</td>
<td>0.604964746D+01</td>
<td>0.319208493D-07</td>
</tr>
<tr>
<td>5</td>
<td>2.4000</td>
<td>0.110231764D+02</td>
<td>0.110231764D+02</td>
<td>0.450974791D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.110231764D+02</td>
<td>0.110231764D+02</td>
<td>-0.366462666D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.110231764D+02</td>
<td>0.110231764D+02</td>
<td>0.539664380D-08</td>
</tr>
<tr>
<td>6</td>
<td>3.0000</td>
<td>0.200855369D+02</td>
<td>0.200855369D+02</td>
<td>0.716164905D-08</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.200855369D+02</td>
<td>0.200855369D+02</td>
<td>-0.169556351D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.200855369D+02</td>
<td>0.200855369D+02</td>
<td>-0.136451952D-07</td>
</tr>
<tr>
<td>7</td>
<td>3.6000</td>
<td>0.365982345D+02</td>
<td>0.365982344D+02</td>
<td>-0.159334164D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.365982345D+02</td>
<td>0.365982344D+02</td>
<td>-0.192572500D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.365982344D+02</td>
<td>0.365982344D+02</td>
<td>-0.500945774D-08</td>
</tr>
<tr>
<td>8</td>
<td>4.2000</td>
<td>0.666863311D+02</td>
<td>0.666863310D+02</td>
<td>-0.193062100D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.666863311D+02</td>
<td>0.666863310D+02</td>
<td>-0.313411270D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.666863310D+02</td>
<td>0.666863310D+02</td>
<td>0.170771948D-07</td>
</tr>
<tr>
<td>9</td>
<td>4.8000</td>
<td>0.121510418D+03</td>
<td>0.121510418D+03</td>
<td>0.102888684D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.121510418D+03</td>
<td>0.121510418D+03</td>
<td>-0.503274649D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.121510418D+03</td>
<td>0.121510418D+03</td>
<td>0.372506967D-07</td>
</tr>
<tr>
<td>10</td>
<td>5.4000</td>
<td>0.221406416D+03</td>
<td>0.221406416D+03</td>
<td>0.489649175D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.221406416D+03</td>
<td>0.221406416D+03</td>
<td>-0.360825183D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.221406416D+03</td>
<td>0.221406416D+03</td>
<td>0.207052722D-07</td>
</tr>
<tr>
<td>11</td>
<td>6.0000</td>
<td>0.403428793D+03</td>
<td>0.403428793D+03</td>
<td>0.120757022D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.403428793D+03</td>
<td>0.403428793D+03</td>
<td>0.149755124D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.403428793D+03</td>
<td>0.403428793D+03</td>
<td>0.130721105D-07</td>
</tr>
</tbody>
</table>
SUBROUTINE DMUTSP(FLIN,FDIF,N,IHOM,KSP,A,B,MA,MB,BCV,AMP,ER,NRTI,1
                   TI,NTI,X,U,NQ,NKSP,ZI,D,KPART,PHIREC,W,LW,
                   2
                   IW,LIW,IERROR)
C
C INTEGER N,IHOM,KSP,NRTI,NTI,NU,NKSP,LW,LIW,IERROR
C DOUBLE PRECISION A,B,MA(N,N),MB(N,N),BCV(N),AMP,ER(5),TI(NTI),
C X(N,NTI),U(NU,NTI),Q(N,NKSP,NTI),ZI(NKSP,NTI),
C D(NKSP,NTI),PHIREC(NU,NTI),W(LW)
C EXTERNAL FLIN,FDIF

Purpose

DMUTSP solves two-point BVP with partially separated BC of the form:
\[
\frac{dx(t)}{dt} = L(t)x(t) + r(t), \quad A < t < B \text{ or } B < t < A,
\]
with BC:
\[
\begin{align*}
^1M_A x(A) + &^1M_B x(B) = BCV^1 \\
^2M_A x(A) + &BCV^2
\end{align*}
\]

where $^1M_A$ and $^1M_B$ are $(KSP \times N)$ BC matrices, $^2M_B$ an $((N-KSP) \times N)$ BC matrix, $BCV^1$ a KSP BC vector and $BCV^2$ an $(N-KSP)$ BC vector.

Parameters

FLIN SUBROUTINE, supplied by the user with specification:

SUBROUTINE FLIN(T,X,F)
DOUBLE PRECISION T,X(N),F(N)

where N is the order of the system. FLIN must evaluate the homogeneous part of the differential equation, $L(t)x(t)$, for $t=T$ and $x(t)=X$ and place the result in $F(1), F(2), \ldots, F(N)$.

FLIN must be declared as EXTERNAL in the (sub)program from which DMUTSP is called.

FDIF SUBROUTINE, supplied by the user, with specification:

SUBROUTINE FDIF(T,X,F)
DOUBLE PRECISION T,X(N),F(N)

where N is the order of the system. FDIF must evaluate the righthand-side of the inhomogeneous differential equation, $L(t)x(t) + r(t)$, for $t=T$ and $x(t)=X$ and place the result in $F(1), F(2), \ldots, F(N)$.  

-- 30 --
FIDIF must be declared as EXTERNAL in the (sub)program from which DMUTSP is called.
In the case that the system is homogeneous FIDIF is the same as FLIN.

N INTEGER, the order of the system.
Unchanged on exit.

IHOM INTEGER.
IHOM indicates whether the system is homogeneous or inhomogeneous.
IHOM = 0 : the system is homogeneous,
IHOM = 1 : the system is inhomogeneous.
Unchanged on exit.

KSP INTEGER
KSP denotes the k-separation, i.e. the number of rows of \(1M_A\) and \(1M_B\).
On entry: \(0 < KSP < N\)
Unchanged on exit

A,B DOUBLE PRECISION, the two boundary points.
Unchanged on exit.

MA,MB DOUBLE PRECISION array of dimension (N,N).
On entry : MA and MB must contain the matrices in the BC:
\[ MAy(A) + MBy(B) = BCV \]
where \(MA = \begin{bmatrix}
1M_A \\
2M_A
\end{bmatrix} \) and \(MB = \begin{bmatrix}
1M_B \\
0
\end{bmatrix} \)

Note that the last \((N-KSP)\) rows of MB are supposed to be zero. When one has a BVP for which the last \((N-KSP)\) rows of MA are zero instead of the last \((N-KSP)\) rows of MB, interchange A and B, the two boundary points, and MA and MB.
Unchanged on exit.

BCV DOUBLE PRECISION array of dimension (N).
On entry: BCV must contain the BC vector; BCV=(BCV\(^t\),BCV\(^2\)\(^t\)).
Unchanged on exit.

AMP DOUBLE PRECISION.
On entry AMP must contain the allowed incremental factor of the homogeneous solutions.
AMP should be greater than 1, if not the subroutine will change AMP into \(\max(ER(1),ER(2)) / ER(3)\). If NRTI > 0, AMP is a dummy parameter.

ER DOUBLE PRECISION array of dimension (5).
On entry ER(1) must contain a relative tolerance for solving the differential equation. If the relative tolerance is smaller than 1.0 e-12 the subroutine will change ER(1) into 1.E-12 + 2 * ER(3).
On entry ER(2) must contain an absolute tolerance for solving the differential equation.
On entry ER(3) must contain the machine constant.
On exit ER(2) and ER(3) are unchanged.
On exit ER(4) contains an estimation of the condition number of the BVP.
On exit ER(5) contains an estimated error amplification factor.

**NRTI**

INTEGER.

On entry:

NRTI = 0, in this case the subroutine determine automatically the output-points using AMP.

NRTI = 1, in this case the output-points are supplied by the user in the array TI.

NRTI > 1, in this case the subroutine computes the output-points \( \text{TI}(k) \) by:

\[
\text{TI}(k) = A + (k-1) \times (B - A) / \text{NRTI} ;
\]

so \( \text{TI}(1) = A \) and \( \text{TI}(\text{NRTI}+1) = B \).

On exit NRTI contains the total number of output-points.

**TI**

DOUBLE PRECISION array of dimension (NTI).

On entry: if NRTI = 1, TI must contain the required output-points in strict monotone order: \( A = \text{TI}(1) < \ldots < \text{TI}(l) = B \) or \( A = \text{TI}(1) > \ldots > \text{TI}(l) = B \) (1 denotes the total number of required output-points).

On exit: \( \text{TI}(1), i=1,2,\ldots,\text{NRTI} \), contains the output-points.

**NTI**

INTEGER.

NTI is the dimension of TI and one of the dimensions of the arrays X, U, Q, ZI, D, PHIREC. NTI must be greater then the total number of output-points + 3.

Unchanged on exit.

**X**

DOUBLE PRECISION array of dimension (N,NTI).

On exit \( X(i,k), i=1,2,\ldots,N \) contains the solution of the BVP at the output-point \( \text{TI}(k) \), \( k=1,\ldots,\text{NRTI} \).

**U**

DOUBLE PRECISION array of dimension (NU,NTI).

On exit \( U(i,k), i=1,2,\ldots,NU \) contains the relevant elements of the upper-triangular matrix \( U_k \), \( k=2,\ldots,\text{NRTI} \). The elements are stored column wise, the jth column of \( U_k \) is stored in \( U(nj+1,k), U(nj+2,k),\ldots,U(nj+j,k) \), where \( nj = (j-1) \times j / 2 \).

**NU**

INTEGER.

NU is one of the dimensions of U and PHIREC.

NU must be at least equal to \( \text{KSP} \times (\text{KSP}+1) / 2 \).

Unchanged on exit.

**Q**

DOUBLE PRECISION array of dimension (N,NKSP,NTI).

On exit \( Q(i,j,k), i=1,2,\ldots,N, j=1,2,\ldots,\text{KSP} \) contains the KSP columns of the orthogonal matrix \( Q_k \), \( k=1,\ldots,\text{NRTI} \).

**NKSP**

INTEGER.

NKSP is one of the dimension of Q, ZI, D. NKSP must be greater than or equal to KSP.

Unchanged on exit.

**ZI**

DOUBLE PRECISION array of dimension (NKSP,NTI).

The array ZI is used for storing the particular solution \( z_i \), \( i=1,\ldots,\text{NRTI} \) of the multiple shooting recursion.

**D**

DOUBLE PRECISION array of dimension (NKSP,NTI).

On exit \( D(i,k), i=1,2,\ldots,\text{KSP} \) contains the inhomogeneous term \( d_k \), \( k=1,2,\ldots,\text{NRTI} \), of the multiple shooting recursion.

-- 32 --
KPART INTEGER.
On exit KPART contains the global k-partition of the uppertriangular matrices $U_k$.

PHIREC DOUBLE PRECISION array of dimension (NU,NTI).
On exit PHIREC contains a fundamental solution of the multiple shooting recursion. The fundamental solution is uppertriangular and is stored in the same way as the $U_k$.

W DOUBLE PRECISION array of dimension (LW).
Used as work space.

LW INTEGER
LW is the dimension of W. LW > 8*N + 2*N*N.
Unchanged on exit.

IW INTEGER array of dimension (LIW)
Used as work space.

LIW INTEGER
LIW is the dimension of IW. LIW > KSP + 2*N.
Unchanged on exit.

IERROR INTEGER
Error indicator; IERROR = 0 then there are no errors detected.

****************
Error indicators
****************

Errors detected by the subroutine

IERROR = 0 No errors detected.
IERROR = 100 INPUT ERROR: either N < 2 or IHOM < 0 or NRTI < 0 or NTI < 5 or NU < KSP * (KSP+1) / 2 or A = B.
TERMINAL ERROR.
IERROR = 101 INPUT ERROR: either ER(1) or ER(2) or ER(3) is negative.
TERMINAL ERROR.
IERROR = 102 INPUT ERROR: either KSP < 1 or KSP > N or NKSP < KSP.
TERMINAL ERROR.
IERROR = 103 INPUT ERROR: either LW < 8*N + 2*N*N or LIW < KSP + 2*N.
TERMINAL ERROR.
IERROR = 104 INPUT ERROR: KSP = N.
TERMINAL ERROR.
IERROR = 120 INPUT ERROR: the routine was called with NRTI = 1, but the given output-points in the array TI are not in strict monotone order.
TERMINAL ERROR.
IERROR = 121  INPUT ERROR: the routine was called with NRTI = 1, but the first given output-point or the last output-point is not equal to A or B. TERMINAL ERROR.

IERROR = 122  INPUT ERROR: the value of NTI is too small; the number of output-points is greater than NTI - 3. TERMINAL ERROR.

IERROR = 200  This indicates that there is a minor shooting interval on which the incremental growth is greater than the AMP. The cause of this error lies in the used method for computing the fundamental solution. WARNING ERROR.

IERROR = 213  This indicates that the relative tolerance was too small. The subroutine has changed it into a suitable value. WARNING ERROR.

IERROR = 215  This indicates that during integration the particular solution or a homogeneous solution has vanished, making a pure relative error test impossible. Must use non-zero absolute tolerance to continue. TERMINAL ERROR.

IERROR = 216  This indicates that during integration the requested accuracy could not be achieved. User must increase error tolerance. TERMINAL ERROR.

IERROR = 218  This indicates that the input parameter N <= 0, or that either the relative tolerance or the absolute tolerance is negative. TERMINAL ERROR.

IERROR = 240  This indicates that the global error is probably larger than the error tolerance due to instabilities in the system. Most likely the problem is ill-conditioned. Output value is the estimated error amplification factor. WARNING ERROR.

IERROR = 250  This indicates that one of the $U_k$ is singular. TERMINAL ERROR.

IERROR = 260  This indicates that the problem is probably too ill-conditioned with respect to the BC. TERMINAL ERROR.

***************
Auxiliary Routines
***************

This routine calls the BOUNDPAK library routines DCQIZI, DDURI, DDURT, DGTUR, DKPCH, DFUNRC, DPSR, DSBVP.
Remarks

DMUTSP is written by G.W.M. Staarink and R.M.M. Mattheij.
Last update: 01-01-1986.

Method

See chapter I of BOUNDPAK User's Manual

Example of the use of DMUTSP

Consider the ordinary differential equation
\[ \frac{dx}{dt}(t) = L(t)x(t) + r(t), \quad 0 < t < 6 \]
and a BC:

\[
\begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
\end{bmatrix}
\begin{bmatrix}
x(0) \\
x(6) \\
\end{bmatrix}
+ 
\begin{bmatrix}
0 & 0 & 1 \\
0 & 1 & 0 \\
0 & 1 & 0 \\
\end{bmatrix}
\begin{bmatrix}
x(0) \\
x(6) \\
\end{bmatrix}
= 
\begin{bmatrix}
1 + e^5 \\
1 + e^6 \\
1 \\
\end{bmatrix}
\]

where

\[
L(t) = 
\begin{bmatrix}
1 - 2\cos(2t) & 0 & 1 + 2\sin(2t) \\
0 & 2 & 0 \\
-1 + 2\sin(2t) & 0 & 1 + 2\cos(2t) \\
\end{bmatrix}
\]

\[
r(t) = 
\begin{bmatrix}
(-1 + 2\cos(2t) - 2\sin(2t))e^t \\
- e^t \\
(1 - 2\cos(2t) - 2\sin(2t))e^t \\
\end{bmatrix}
\]

The solution of this problem is: \[ x(t) = (e^t, e^t, e^t)^T. \]

In the next program the solution is computed and compared to the exact solution.
This program has been run on an AS9000 VM/CMS computer.
DOUBLE PRECISION A,B,MA(3,3),MB(3,3),BCV(3),AMP,ER(5),TI(15),
X(3,15),U(3,15),Q(3,2,15),WI(2,15),D(2,15),PHIREC(3,15),W(42),
EXSOL,AE
INTEGER IW(8)
EXTERNAL FLIN,FDIF

C SETTING OF THE INPUT PARAMETERS
C
N = 3
IHOM = 1
KSP = 2
ER(1) = 1.D-11
ER(2) = 1.D-6
ER(3) = 1.D-16
NRTI = 10
NTI = 15
NU = 3
NKSP = 2
LW = 42
LIW = 8
A = 0.DO
B = 6.DO

C SETTING THE BC MATRICES MA AND MB
C
DO 1000 I = 1 , N
 DO 1000 J = 1 , N
 MA(I,J) = 0.DO
 MB(I,J) = 0.DO
1000 CONTINUE
MA(1,3) = 1.DO
MA(2,2) = 1.DO
MA(3,1) = 1.DO
MB(1,3) = 1.DO
MB(2,2) = 1.DO

1100 CONTINUE
C
C SETTING THE BC VECTOR BCV
C
BCV(1) = 1.DO + DEXP(6.DO)
BCV(2) = BCV(1)
BCV(3) = 1.DO

C CALL DMUTSP
C
CALL DMUTSP(FLIN,FDIF,N,IHOM,KSP,A,B,MA,MB,BCV,AMP,ER,NRTI, TI,
NTI,X,U,NU,Q,NKSP, WI, D, KPART, PHIREC, W, LW, LIW,
2ERROR)
IF ((ERROR.NE.0).AND.(ERROR.NE.200).AND.(ERROR.NE.213).AND.
1(ERROR.NE.240)) GOTO 5000

C PRINTING OF THE CONDITION NUMBER AND AMPLIFICATION FACTOR.
C
WRITE(6,200)
WRITE(6,205) ER(4),ER(5)
C
C COMPUTATION OF THE ABSOLUTE ERROR IN THE SOLUTION AND WRITING
C OF THE SOLUTION AT THE OUTPUTPOINTS

-- 36 --
WRITE(6,200)
WRITE(6,210)
WRITE(6,200)
DO 1500 K = 1 , NRTI
  EXSOL = DEXP(TI(K))
  AE = EXSOL - X(1,K)
WRITE(6,220) K,TI(K),X(1,K),EXSOL,AE
DO 1300 I = 2 , N
  AE = EXSOL - X(I,K)
  WRITE(6,230) X(I,K),EXSOL,AE
1300 CONTINUE
1500 CONTINUE
STOP
5000 WRITE(6,300) !ERROR
STOP
C
200 FORMAT( ' ' )
205 FORMAT( ' CONDITION NUMBER = ',D10.3,/,1 ' AMPLIFICATION FACTOR = ',D10.3,/) 
210 FORMAT( ' I ',5X,'T',8X,'APPROX. SOL.',9X,'EXACT SOL.',8X, 1 'ABS. ERROR' )
220 FORMAT( ' ',I3,3X,F7.4,3(3X,D16.9))
230 FORMAT( ' ',I3,3(3X,D16.9))
300 FORMAT( ' TERMINAL ERROR IN DMUTSP: IERROR = ',I4)
C
END
C
SUBROUTINE FLIN(T,Y,F)
C ----------------------
C
DOUBLE PRECISION T,Y(3),F(3)
DOUBLE PRECISION TI,SI,CO
C
TI = 2.DO * T
SI = 2.DO * DSIN(TI)
CO = 2.DO * DCOS(TI)
F(1) = (1.DO - CO) * Y(1) + (1.DO + SI) * Y(3)
F(2) = 2.DO * Y(2)
F(3) = (-1.DO + SI) * Y(1) + (1.DO + CO) * Y(3)
C
RETURN
C
END OF FLIN
END
C
SUBROUTINE FDIF(T,Y,F)
C ----------------------
C
DOUBLE PRECISION T,Y(3),F(3)
DOUBLE PRECISION TI,SI,CO
C
CALL FLIN(T,Y,F)
TI = 2.DO * T
SI = 2.DO * DSIN(TI)
CO = 2.DO * DCOS(TI)
TI = DEXP(T)
F(1) = F(1) + (-1.DO + CO - SI)*TI
F(2) = F(2) - TI
F(3) = F(3) + (1.DO - CO - SI)*TI
\begin{verbatim}
RETURN
END OF FDIF
END

CONDITION NUMBER = 0.104D+01
AMPLIFICATION FACTOR = 0.143D+01

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>APPROX. SOL.</th>
<th>EXACT SOL.</th>
<th>ABS. ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0000</td>
<td>0.100000000D+01</td>
<td>0.100000000D+01</td>
<td>0.000000000D+00</td>
</tr>
<tr>
<td></td>
<td>0.100000000D+01</td>
<td>0.100000000D+01</td>
<td>-0.16235211D-07</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.6000</td>
<td>0.182211882D+01</td>
<td>0.182211882D+01</td>
<td>-0.19406179D-07</td>
</tr>
<tr>
<td></td>
<td>0.182211882D+01</td>
<td>0.182211882D+01</td>
<td>-0.17042006D-07</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.2000</td>
<td>0.332011694D+01</td>
<td>0.332011694D+01</td>
<td>-0.14228071D-07</td>
</tr>
<tr>
<td></td>
<td>0.332011694D+01</td>
<td>0.332011694D+01</td>
<td>-0.24211887D-07</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.8000</td>
<td>0.604964745D+01</td>
<td>0.604964745D+01</td>
<td>-0.18070638D-07</td>
</tr>
<tr>
<td></td>
<td>0.604964745D+01</td>
<td>0.604964745D+01</td>
<td>-0.40753121D-07</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2.4000</td>
<td>0.110231763D+02</td>
<td>0.110231763D+02</td>
<td>-0.50256520D-07</td>
</tr>
<tr>
<td></td>
<td>0.110231763D+02</td>
<td>0.110231763D+02</td>
<td>-0.60417003D-07</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3.0000</td>
<td>0.200855369D+02</td>
<td>0.200855369D+02</td>
<td>-0.76301383D-08</td>
</tr>
<tr>
<td></td>
<td>0.200855369D+02</td>
<td>0.200855369D+02</td>
<td>-0.19834715D-07</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3.6000</td>
<td>0.365982345D+02</td>
<td>0.365982345D+02</td>
<td>-0.27245139D-07</td>
</tr>
<tr>
<td></td>
<td>0.365982345D+02</td>
<td>0.365982345D+02</td>
<td>-0.18231453D-07</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4.2000</td>
<td>0.666863511D+02</td>
<td>0.666863511D+02</td>
<td>-0.22372667D-07</td>
</tr>
<tr>
<td></td>
<td>0.666863511D+02</td>
<td>0.666863511D+02</td>
<td>-0.25348153D-07</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4.8000</td>
<td>0.121510417D+03</td>
<td>0.121510417D+03</td>
<td>-0.39352439D-07</td>
</tr>
<tr>
<td></td>
<td>0.121510417D+03</td>
<td>0.121510417D+03</td>
<td>-0.36732040D-07</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>5.4000</td>
<td>0.221406416D+03</td>
<td>0.221406416D+03</td>
<td>-0.51578780D-07</td>
</tr>
<tr>
<td></td>
<td>0.221406416D+03</td>
<td>0.221406416D+03</td>
<td>-0.57477737D-07</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>6.0000</td>
<td>0.403428793D+03</td>
<td>0.403428793D+03</td>
<td>-0.64316054D-08</td>
</tr>
<tr>
<td></td>
<td>0.403428793D+03</td>
<td>0.403428793D+03</td>
<td>-0.16235333D-07</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.403428793D+03</td>
<td>0.403428793D+03</td>
<td>-0.15301282D-07</td>
<td></td>
</tr>
</tbody>
</table>
\end{verbatim}
**SUBROUTINE DMUTSS(FLIN,FDIF,N,IHOM,KSP,A,B,MA,BCV,AMP,ER,NRTI,TI,NTI,1**
**X,U,NU,Q,NKSP,D,ZI,W,LW,IW,LIW,IERROR)**

**C** INTEGER N,IHOM,KSP,NRTI,NTI,NU,NKSP,LW,LIW,IERROR
**C** DOUBLE PRECISION A,B,MA(N,N),BCV(N),AMP,ER(5),TI(NTI),X(N,NTI),
**C** U(NU,NTI),Q(N,NKSP,NTI),D(NKSP,NTI),ZI(NKSP,NTI),
**C** W(LW)
**C** EXTERNAL FLIN,FDIF

**Purpose**

DMUTSS solves two-point BVP with completely separated BC of the form:

\[
\frac{dx(t)}{dt} = L(t)x(t) + r(t), \quad A < t < B \text{ or } B < t < A,
\]

with BC:

\[
1_{\text{MA}}x(B) = BCV^1
\]
\[
2_{\text{MA}}x(A) = BCV^2
\]

where \(1_{\text{MA}}\) is a \((\text{KSP} \times N)\) BC matrix, \(2_{\text{MA}}\) an \((\text{(N-KSP)} \times N)\) BC matrix, \(BCV^1\) a KSP BC vector and \(BCV^2\) an \((N-KSP)\) BC vector.

**Parameters**

**FLIN** SUBROUTINE, supplied by the user with specification:

SUBROUTINE FLIN(T,X,F) DOUBLE PRECISION T,X(N),F(N)

where \(N\) is the order of the system. FLIN must evaluate the homogeneous part of the differential equation, \(L(t)x(t)\), for \(t=T\) and \(x(t)=X\) and place the result in \(F(1), F(2), \ldots, F(N)\).

FLIN must be declared as EXTERNAL in the (sub)program from which DMUTSS is called.

**FDIF** SUBROUTINE, supplied by the user, with specification:

SUBROUTINE FDIF(T,X,F) DOUBLE PRECISION T,X(N),F(N)

where \(N\) is the order of the system. FDIF must evaluate the righthand-side of the inhomogeneous differential equation, \(L(t)x(t) + r(t)\), for \(t=T\) and \(x(t)=X\) and place the result in \(F(1), F(2), \ldots, F(N)\).

FDIF must be declared as EXTERNAL in the (sub)program from which DMUTSS is called.
In the case that the system is homogeneous, FDIF is the same as FLIN.

**N** INTEGER, the order of the system.
Unchanged on exit.

**IHOM** INTEGER.
IHOM indicates whether the system is homogeneous or inhomogeneous.
IHOM = 0: the system is homogeneous,
IHOM = 1: the system is inhomogeneous.
Unchanged on exit.

**KSP** INTEGER.
KSP denotes the k-separation, i.e. the number of rows of $^1M_A$.
On entry: $0 < KSP < N$.
Unchanged on exit.

**A,B** DOUBLE PRECISION, the two boundary points.
Unchanged on exit.

**MA** DOUBLE PRECISION array of dimension (N,N).
The boundary conditions are of the form:
$^1M_A x(B) = BCV^1$ ; $^2M_A x(A) = BCV^2$.
On entry the first KSP rows of MA must contain $^1M_A$ and the last (N-KSP) rows of MA must contain $^2M_A$.
Unchanged on exit.

**BCV** DOUBLE PRECISION array of dimension (N).
On entry: BCV(1), ..., BCV(KSP) must contain BCV$^1$ and
BCV(KSP+1), ..., BCV(N) must contain BCV$^2$.
Unchanged on exit.

**AMP** DOUBLE PRECISION.
On entry AMP must contain the allowed incremental factor of the homogeneous solutions.
AMP should be greater than 1, if not the subroutine will change AMP into $\max(ER(1), ER(2)) / ER(3)$. If NRTI > 0, AMP is a dummy parameter.

**ER** DOUBLE PRECISION array of dimension (5).
On entry ER(1) must contain a relative tolerance for solving the differential equation. If the relative tolerance is smaller than 1.0 e-12 the subroutine will change ER(1) into $1.E-12 + 2 * ER(3)$.
On entry ER(2) must contain an absolute tolerance for solving the differential equation.
On entry ER(3) must contain the machine constant.
On exit ER(2) and ER(3) are unchanged.
On exit ER(4) contains an estimation of the condition number of the BVP.
On exit ER(5) contains an estimated error amplification factor.

**NRTI** INTEGER.
On entry:
NRTI = 0, in this case the subroutine determine automatically the output-points using AMP.
NRTI = 1, in this case the output-points are supplied by the user in the array TI.

NRTI > 1, in this case the subroutine computes the output-points TI(k) by:

\[ TI(k) = A + (k-1) \times \frac{B - A}{NRTI} \]

so TI(1) = A and TI(NRTI+1) = B.

On exit NRTI contains the total number of output-points.

TI DOUBLE PRECISION array of dimension (NTI).

On entry: if NRTI = 1, TI must contain the required output-points in strict monotone order: A=TI(1) < ... < TI(l)=B or A=TI(1) > ... > TI(l)=B (l denotes the total number of required output-points).

On exit: TI(i), i=1,2,...,NRTI, contains the output-points.

NTI INTEGER.

NTI is the dimension of TI and one of the dimensions of the arrays X, U, Q, ZI, D, PHIREC. NTI must be greater then the total number of output-points + 3.

Unchanged on exit.

X DOUBLE PRECISION array of dimension (N,NTI).

On exit X(i,k), i=1,2,...,N contains the solution of the BVP at the output-point TI(k), k=1,...,NRTI.

U DOUBLE PRECISION array of dimension (NU,NTI).

On exit U(i,k), i=1,2,...,NU contains the relevant elements of the upper-triangular matrix \( U_k \), k=2,...,NRTI. The elements are stored column wise, the jth column of \( U_k \) is stored in \( U(nj+1,k), U(nj+2,k),...,U(nj+j,k) \), where \( nj = (j-1) \times j / 2 \).

NU INTEGER.

NU is one of the dimensions of U and PHIREC.

NU must be at least equal to KSP * (KSP+1) / 2.

Unchanged on exit.

Q DOUBLE PRECISION array of dimension (N,NKSP,NTI).

On exit Q(i,j,k), i=1,2,...,N, j=1,2,...,NKSP contains the KSP columns of the orthogonal matrix \( Q_k \), k=1,...,NRTI.

NKSP INTEGER.

NKSP is one of the dimension of Q, ZI, D. NKSP must be greater than or equal to KSP.

Unchanged on exit.

D DOUBLE PRECISION array of dimension (NKSP,NTI).

On exit D(i,k), i=1,2,...,NKSP contains the inhomogeneous term \( d_k \), k=1,2,...,NRTI, of the multiple shooting recursion.

ZI DOUBLE PRECISION array of dimension (NKSP,NTI).

The array ZI is used for storing the particular solution of the multiple shooting recursion.

W DOUBLE PRECISION array of dimension (LW).

Used as work space.

LW INTEGER
LW is the dimension of W. LW > 8*N + 2*N*N.
Unchanged on exit.

IW INTEGER array of dimension (LIW)
Used as work space.

LIW INTEGER
LIW is the dimension of IW. LIW > KSP + 2*N.
Unchanged on exit.

IERROR INTEGER
Error indicator; IERROR = 0 then there are no errors detected.

***************
Error indicators
***************

Errors detected by the subroutine

IERROR = 0 No errors detected .
IERROR = 1 No errors detected, but integration from B to A.
IERROR = 100 INPUT ERROR: either N < 2 or IHOM < 0 or NRTI < 0 or NTI < 5 or NU < KSP * (KSP+1) / 2 or A = B.
TERMINAL ERROR.
IERROR = 101 INPUT ERROR: either ER(1) or ER(2) or ER(3) is negative.
TERMINAL ERROR.
IERROR = 102 INPUT ERROR: either KSP < 1 or KSP > N or NKSP < KSP.
TERMINAL ERROR.
IERROR = 103 INPUT ERROR: either LW < 8*N + 2*N*N or LIW < KSP + 2*N.
TERMINAL ERROR.
IERROR = 105 INPUT ERROR: KSP = N.
TERMINAL ERROR.
IERROR = 107 INPUT ERROR: IHOM = 0 and BCV = 0, so the solution will be zero.
TERMINAL ERROR.
IERROR = 120 INPUT ERROR: the routine was called with NRTI = 1, but the given output-points in the array TI are not in strict monotone order.
TERMINAL ERROR.
IERROR = 121 INPUT ERROR: the routine was called with NRTI = 1, but the first given output-point or the last output-point is not equal to A or B.
TERMINAL ERROR.
IERROR = 122 INPUT ERROR: the value of NTI is too small; the number of output-points is greater than NTI-3.
TERMINAL ERROR.

-- 42 --
IERROR = 200  This indicates that there is a minor shooting interval on which the incremental growth is greater than the AMP. The cause of this error lies in the used method for computing the fundamental solution. WARNING ERROR.

IERROR = 213  This indicates that the relative tolerance was too small. The subroutine has changed it into a suitable value. WARNING ERROR.

IERROR = 215  This indicates that during integration the particular solution or a homogeneous solution has vanished, making a pure relative error test impossible. Must use non-zero absolute tolerance to continue. TERMINAL ERROR.

IERROR = 216  This indicates that during integration the requested accuracy could not be achieved. User must increase error tolerance. TERMINAL ERROR.

IERROR = 218  This indicates that the input parameter N <= 0, or that either the relative tolerance or the absolute tolerance is negative. TERMINAL ERROR.

IERROR = 220  The computed recursion is not dichotomic. Check the condition number and the amplification factor. WARNING ERROR.

IERROR = 240  This indicates that the global error is probably larger than the error tolerance due to instabilities in the system. Most likely the problem is ill-conditioned. Output value is the estimated error amplification factor. WARNING ERROR.

IERROR = 250  This indicates that one of the $U_k$ is singular. TERMINAL ERROR.

IERROR = 260  This indicates that the problem is probably too ill-conditioned with respect to the BC. TERMINAL ERROR.

***************
Auxiliary Routines
***************

This routine calls the BOUNDPAK library routines DCQIZI, DDURI, DDURT, DKPCH, DCROUT, DSOLDE, DSOLUP.

***************
Remarks
***************

***************
Method
***************

See chapter I of BOUNPAK User's Manual

***************
Example of the use of DMUTSS
***************

Consider the ordinary differential equation

\[ \frac{dx(t)}{dt} = L(t)x(t) + r(t), \quad 0 < t < 6 \]

and a BC:

\[
\begin{bmatrix}
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
x(6) \\
x(0) \\
1
\end{bmatrix}
\begin{bmatrix}
e^6 \\
e^6 \\
1
\end{bmatrix}
\]

where

\[
L(t) =
\begin{bmatrix}
1 - 2\cos(2t) & 0 & 1 + 2\sin(2t) \\
0 & 2 & 0 \\
-1 + 2\sin(2t) & 0 & 1 + 2\cos(2t)
\end{bmatrix}
\]

\[
r(t) =
\begin{bmatrix}
(-1 + 2\cos(t) - 2\sin(2t))e^t \\
- e^t \\
(1 - 2\cos(2t) - 2\sin(2t))e^t
\end{bmatrix}
\]

The solution to this problem is: \( x(t) = (e^t, e^t, e^t)^T \).

In the next program the solution is computed and compared to the exact solution.
This program has been run on a AS9000 VM/CMS computer.

```
DOUBLE PRECISION A,B,MA(3,3),BCV(3),AMP,ER(5),TI(15),
1 X(3,15),U(3,15),Q(3,2,15),WI(2,15),D(2,15),W(42),
2 EXSOL,AE
INTEGER IW(8)
EXTERNAL FLIN,FDIF
C
C SETTING OF THE INPUT PARAMETERS
C
N = 3
IHOM = 1
KSP = 2
ER(1) = 1.D-11
ER(2) = 1.D-6
ER(3) = 1.1D-16
```
NRTI = 10
NTI = 15
NU = 3
NKSP = 2
LW = 42
LIW = 8
A = 0.DO
B = 6.DO

C SETTING THE BC MATRIX MA

DO 1000 I = 1, N
   DO 1000 J = 1, N
      MA(I,J) = 0.DO
1000 CONTINUE
MA(1,3) = 1.DO
MA(2,2) = 1.DO
MA(3,1) = 1.DO
1100 CONTINUE

C SETTING THE BC VECTOR BCV

BCV(1) = DEXP(6.DO)
BCV(2) = BCV(1)
BCV(3) = 1.DO

CALL DMUTSS

CALL DMUTSS(FLIN,FDIF,N,IHOM,KSP,A,B,MA,BCV,AMP,ER,NRTI,NI,NTI, 
           X,U,NU,Q,NKSP,D,NI,LIW,W,LW,IW,IER)
IF ((IER == 0).AND. (IER == 200).AND. (IER == 213).AND. 
    (IER == 220).AND. (IER == 240)) GOTO 5000

C PRINTING OF THE CONDITION NUMBER AND AMPLIFICATION FACTOR.

WRITE(6,200)
WRITE(6,205) ER(4),ER(5)

C COMputation OF THE ABSOLUTE ERROR IN THE SOLUTION AND WRITING 
C OF THE SOLUTION AT THE OUTPUTPOINTS

WRITE(6,200)
WRITE(6,210)
WRITE(6,200)
DO 1500 K = 1 , NRTI
   EXSOL = DEXP(TI(K))
   AE = EXSOL - X(1,K)
   WRITE(6,220) K, TI(K), X(1,K), EXSOL, AE
DO 1300 I = 2, N
   AE = EXSOL - X(I,K)
   WRITE(6,230) X(I,K), EXSOL, AE
1300 CONTINUE
1500 CONTINUE
STOP
5000 WRITE(6,300) IERROR
STOP

200 FORMAT( ' ')
205 FORMAT( ' CONDITION NUMBER = ', D10.3, '/')
1 ' AMPLIFICATION FACTOR = ',D10.3,/
210 FORMAT( 'I',6X,'T',8X,'APPROX. SOL.',9X,'EXACT SOL.',8X,
1 'ABS. ERROR'
220 FORMAT( ' ',I3,3X,F7.4,3(3X,D16.9))
230 FORMAT( ' ',I3,3X,D16.9))
300 FORMAT( ' TERMINAL ERROR IN DMUTSS: ERROR = ',I4)

END

SUBROUTINE FLIN(T,Y,F)
DOUBLE PRECISION T,Y(3),F(3)
DOUBLE PRECISION TI,SI,CO

TI = 2.DO*T
SI = 2.DO*DSIN(TI)
CO = 2.DO*DCOS(TI)
F(1) = (1.DO - CO)*Y(1) + (1.DO + SI)*Y(3)
F(2) = 2.DO*Y(2)
F(3) = (-1.DO + SI)*Y(1) + (1.DO + CO)*Y(3)

RETURN

END OF FLIN

SUBROUTINE FDIF(T,Y,F)
DOUBLE PRECISION T,Y(3),F(3)
DOUBLE PRECISION TI,SI,CO

CALL FLIN(T,Y,F)
TI = 2.DO*T
SI = 2.DO*DSIN(TI)
CO = 2.DO*DCOS(TI)
TI = DEXP(T)
F(1) = F(1) + (-1.DO + CO - SI)*TI
F(2) = F(2) - TI
F(3) = F(3) + (1.DO - CO - SI)*TI

RETURN

END OF FDIF

CONDITION NUMBER = 0.128D+01
AMPLIFICATION FACTOR = 0.226D+01

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>APPROX. SOL.</th>
<th>EXACT SOL.</th>
<th>ABS. ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.0000</td>
<td>0.403428793D+03</td>
<td>0.403428793D+03</td>
<td>0.194528411D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.403428793D+03</td>
<td>0.403428793D+03</td>
<td>0.0000000000D+00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.403428793D+03</td>
<td>0.403428793D+03</td>
<td>0.0000000000D+00</td>
</tr>
<tr>
<td>2</td>
<td>5.4000</td>
<td>0.221406416D+03</td>
<td>0.221406416D+03</td>
<td>-0.114642528D-06</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.221406416D+03</td>
<td>0.221406416D+03</td>
<td>0.899770427D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>3</td>
<td>4.8000</td>
<td>0.221406416D+03</td>
<td>0.221406416D+03</td>
<td>-0.437768790D-07</td>
</tr>
<tr>
<td>4</td>
<td>4.2000</td>
<td>0.121510418D+03</td>
<td>0.121510418D+03</td>
<td>-0.29583042D-07</td>
</tr>
<tr>
<td>5</td>
<td>3.6000</td>
<td>0.666863309D+02</td>
<td>0.666863310D+02</td>
<td>-0.167401488D-06</td>
</tr>
<tr>
<td>6</td>
<td>3.0000</td>
<td>0.365982342D+02</td>
<td>0.365982344D+02</td>
<td>-0.241659911D-06</td>
</tr>
<tr>
<td>7</td>
<td>2.4000</td>
<td>0.200855369D+02</td>
<td>0.200855369D+02</td>
<td>-0.170953172D-07</td>
</tr>
<tr>
<td>8</td>
<td>1.8000</td>
<td>0.604964751D+01</td>
<td>0.604964746D+01</td>
<td>-0.414294898D-07</td>
</tr>
<tr>
<td>9</td>
<td>1.2000</td>
<td>0.332011687D+01</td>
<td>0.332011692D+01</td>
<td>-0.100124572D-06</td>
</tr>
<tr>
<td>10</td>
<td>0.6000</td>
<td>0.182211886D+01</td>
<td>0.182211886D+01</td>
<td>-0.135680970D-06</td>
</tr>
<tr>
<td>11</td>
<td>0.0000</td>
<td>0.100000000D+01</td>
<td>0.100000000D+01</td>
<td>-0.66062318D-07</td>
</tr>
</tbody>
</table>

--- 47 ---
SUBROUTINE DMUTS(FLIN, FDIF, N, IHOM, LBCTYP, KSP, A, B, MA, MB, BCV, ER, AMP,
            1 NRTI, TI, NTI, X, U, NU, Q, NKSP, ZI, D, KPART, PHIREC,
            2 W, LW, IW, LIW, IERROR)
C
INTEGER N, IHOM, KSP, NRTI, NTI, NU, NKSP, LW, IW(LIW), LIW, IERROR
C
DOUBLE PRECISION A, B, MA(N, N), MB(N, N), BCV(N), ER(5), AMP, TI(NTI),
C 1 X(N, NTI), U(NU, NTI), Q(N, NKSP, NTI), ZI(NKSP, NTI),
C 2 D(NKSP, NTI), PHIREC(NU, NTI), W(LW)
C
EXTERNAL FLIN, FDIF

Purpose
***************
DMUTS solves two-point boundary value problems (BVP) of the form:

\[ \frac{dx(t)}{dt} = L(t)x(t) + r(t) \quad , \quad A < t < B , \]

with BC:

\[ MAx(A) + MBx(B) = BCV \]

where \( MA \) and \( MB \) are (\( N \times N \)) BC matrices and BCV an \( N \) BC vector.

DMUTS can be used for any type of BC and has moreover an option to find out if
rank\( (MA) \) or rank\( (MB) \) are smaller than \( N \).

Parameters
***************

FLIN SUBROUTINE, supplied by the user with specification:

SUBROUTINE FLIN(T, X, F)
DOUBLE PRECISION T, X(N), F(N)

where \( N \) is the order of the system. FLIN must evaluate the homogeneous
part of the differential equation, \( L(t)x(t) \) for \( t=T \) and \( x(t)=X \) and
place the result in \( F(1), F(2), \ldots, F(N) \).

FLIN must be declared as EXTERNAL in the (sub)program from which DMUTS is called.

FDIF SUBROUTINE, supplied by the user, with specification:

SUBROUTINE FDIF(T, X, F)
DOUBLE PRECISION T, X(N), F(N)

where \( N \) is the order of the system. FDIF must evaluate the righthand-
side of the inhomogeneous differential equation, \( L(t)x(t) + r(t) \), for
\( t=T \) and \( x(t)=X \) and place the result in \( F(1), F(2), \ldots, F(N) \).

FDIF must be declared as EXTERNAL in the (sub)program from which DMUTS is called.
In the case that the system is homogeneous FDIF is the same as FLIN.

N INTEGER, the order of the system.
Unchanged on exit.

IHOM INTEGER.
IHOM indicates whether the system is homogeneous or inhomogeneous.
IHOM = 0 : the system is homogeneous,
IHOM = 1 : the system is inhomogeneous.
Unchanged on exit.

IBCTYP INTEGER.
IBCTYP denotes the type of BC (boundary conditions).
IBCTYP = 1 : general BC ;
IBCTYP = 2 : partially separated BC ;
IBCTYP = 3 : completely separated BC ;
IBCTYP = 4 : in this case the routine checks whether the
BC are separated or not. If this check is positive new MA,
MB and BCV are computed and the BVP solved as if IBCTYP = 2,
else the BVP is solved as if IBCTYP = 1.

KSP INTEGER.
KSP denotes the k-separation.
On entry KSP must contain:
If IBCTYP = 1 : KSP = N.
If IBCTYP = 2 : 0 < KSP < N.
If IBCTYP = 3 : 0 < KSP < N.
On exit KSP is unchanged, unless IBCTYP = 4, then KSP contains the
computed k-separation

A, B DOUBLE PRECISION, the two boundary points.
Unchanged on exit.

MA, MB DOUBLE PRECISION array of dimension (N,N).
On entry :
IBCTYP = 1 : MA and MB must contain the matrices in the BC:
MAx(A) + MBx(B) = BCV. Unchanged on exit.
IBCTYP = 2 : MA and MB must contain the matrices in the BC:
MAx(A) + MBx(B) = BCV, where the last (N-KSP) rows of MB are supposed
to be zero. When the last (N-KSP) rows of MA are zero instead of the
last (N-KSP) rows of MB, interchange A and B, the two boundary points,
and MA and MB.
Unchanged on exit.
IBCTYP = 3 : here the BC are of the form :
2MAx(A) = BCV ; 1MAx(B) = BCV. The first KSP rows of MA must con-
tain 1MA, the last (N-KSP) must contain 2MA. In this case MB is a
dummy array.
Unchanged on exit.
IBCTYP = 4 : MA and MB must contain the matrices in the BC:
MAx(A) + MBx(B) = BCV. If the BC are separated MA and MB are changed
accordingly on exit.

BCV DOUBLE PRECISION array of dimension (N).
On entry BCV must contain the BC vector. If IBCTYP = 3 , BCV =
(BCV1, BCV2)T.
If IBCTYP = 4 and the BC are separated BCV is changed accordingly on exit, else BCV is unchanged on exit.

**ER**

DOUBLE PRECISION array of dimension (5).

On entry ER(1) must contain a relative tolerance for solving the differential equation. If the relative tolerance is smaller then 1.0 e-12 the subroutine will change ER(1) into 1.e-12 + 2 * ER(3)

On entry ER(2) must contain an absolute tolerance for solving the differential equation.

On entry ER(3) must contain the machine constant.

On exit ER(2) and ER(3) are unchanged.

On exit ER(4) contains an estimation of the condition number of the BVP.

On exit ER(5) contains an estimated error amplification factor.

**AMP**

DOUBLE PRECISION.

On entry AMP must contain the allowed incremental factor of the homogeneous solutions.

AMP should be greater than 1, if not the subroutine will change AMP into max(ER(1),ER(2)) / ER(3). If NRTI > 0, AMP is a dummy parameter.

**NRTI**

INTEGER.

On entry:

NRTI = 0, in this case the subroutine determines automatically the output points using AMP.

NRTI = 1, in this case the output points are supplied by the user in the array TI.

NRTI > 1, in this case the subroutine computes the output points TI(k) by:

\[ TI(k) = A + (k-1) * (B - A) / NRTI; \]

so TI(1) = A and TI(NRTI+1) = B.

On exit NRTI contains the total number of output-points.

**TI**

DOUBLE PRECISION array of dimension (NTI).

On entry: if NRTI = 1 , TI must contain the required output-points in strict monotone order: A=TI(1) < ... < TI(1)=B or A=TI(1) > ... > TI(1)=B (1 denotes the total number of required output-points).

On exit: TI(i), i=1,2,...,NRTI, contain the output-points.

**NTI**

INTEGER.

NTI is the dimension of TI and one of the dimensions of the arrays X, U, Q, ZI, D, PHIREC. NTI must be greater then the total number of output points + 3.

Unchanged on exit.

**X**

DOUBLE PRECISION array of dimension (N,NTI).

On exit X(i,k), i=1,2,...,N contains the solution of the BVP at the output-point TI(k), k=1,...,NRTI.

**U**

DOUBLE PRECISION array of dimension (NU,NTI).

On exit U(i,k) i=1,2,...,NU contains the relevant elements of the upper-triangular matrix \( U_k \), k=2,...,NRTI. The elements are stored column wise, the jth column of \( U_k \) is stored in U(nj+1,k), U(nj+2,k),..., U(nj+j,k), where nj = (j-1) * j / 2.

**NU**

INTEGER.

NU is one of the dimensions of U and PHIREC.
If IBCTYP = 1 or IBCTYP = 4 NU must be at least equal to N * (N+1) / 2.
If IBCTYP = 2 or IBCTYP = 3 NU must be at least equal to KSP * (KSP+1) / 2.
Unchanged on exit.

Q DOUBLE PRECISION array of dimension (N,NKSP,NTI).
On exit Q(i,j,k) i=1,2,...,N, j=1,2,...,KSP contains the KSP columns of the orthogonal matrix Qk, k=1,...,NRTI.

NKSP INTEGER.
NKSP is one of the dimension of Q, ZI, D. NKSP must be greater than or equal to KSP.
Unchanged on exit.

ZI DOUBLE PRECISION array of dimension (NKSP,NTI).
If IBCTYP = 1 the array ZI has no real use and the user is recommended to use the same array for the ZI and the D.
If IBCTYP > 1 the array ZI is used for storing the particular solution of the multiple shooting recursion.

D DOUBLE PRECISION array of dimension (NKSP,NTI).
If IHOM = 0 the array D has no real use and the user is recommended to use the same array for the X and the D.
If IHOM = 1 : on exit D(i,k) i=1,2,...,KSP contains the inhomogeneous term d_k, k=1,2,...,NRTI, of the multiple shooting recursion.

KPART INTEGER.
On exit KPART contains the global k-partition of the upper-triangular matrices U_k.

PHIREC DOUBLE PRECISION array of dimension (NU,NTI).
On exit PHIREC contains a fundamental solution of the multiple shooting recursion. The fundamental solution is uppertriangular and is stored in the same way as the U_k.

W DOUBLE PRECISION array of dimension (LW).
Used as work space.

LW INTEGER
LW is the dimension of W. LW > 8*N + 2*N*N.
Unchanged on exit.

IW INTEGER array of dimension (LIW)
Used as work space.

LIW INTEGER
LIW is the dimension of IW. LIW > 3*N.
Unchanged on exit.

IERROR INTEGER
Error indicator, the error numbers less than 100 are informative errors.
Error indicators

Errors detected by the subroutine

IERRO = 0  No errors detected and IBCTYP = 1.
IERRO = 10  No errors detected and IBCTYP = 2
IERRO = 20  No errors detected, IBCTYP = 3 and integration from A to B.
IERRO = 21  No errors detected, IBCTYP = 3 and integration from B to A.
IERRO = 30  No errors detected, IBCTYP = 4 and general BC.
IERRO = 31  No errors detected, IBCTYP = 4, partially separated BC and integration from A to B.
IERRO = 32  No errors detected, IBCTYP = 4, partially separated BC and integration from B to A.
IERRO = 100 INPUT ERROR: either N < 2 or IHOM < 0 or NRTI < 0 or NTI < 5 or IBCTYP < 1 or IBCTYP > 4 or NU < KSP * (KSP+1) / 2 or A = B TERMINAL ERROR.
IERRO = 101 INPUT ERROR: either ER(1) or ER(2) or ER(3) is negative. TERMINAL ERROR.
IERRO = 102 INPUT ERROR: IBCTYP = 2 or 3, either KSP < 1 or KSP > N or NKSP < KSP. TERMINAL ERROR.
IERRO = 103 INPUT ERROR: either LW < 8*N + 2*N*N or LIW < 3*N. TERMINAL ERROR.
IERRO = 104 INPUT ERROR: IBCTYP = 2 and KSP = N. TERMINAL ERROR.
IERRO = 105 INPUT ERROR: IBCTYP = 3 and KSP = N. TERMINAL ERROR.
IERRO = 107 INPUT ERROR: IBCTYP = 3, IHOM = 0 and BCV = 0 so the solution will be zero. TERMINAL ERROR.
IERRO = 120 INPUT ERROR: the routine was called with NRTI = 1, but the given output-points in the array TI are not in strict monotone order. TERMINAL ERROR.
IERRO = 121 INPUT ERROR: the routine was called with NRTI = 1, but the first given output-point or the last output-point is not equal to A or B. TERMINAL ERROR.

-- 52 --
IERROR = 122  INPUT ERROR: the value of NTI is too small; the number of output-points is greater than NTI-3. TERMINAL ERROR.

IERROR = 200  This indicates that there is a minor shooting interval on which the incremental growth is greater than the AMP. The cause of this error lies in the used method for computing the fundamental solution. WARNING ERROR.

IERROR = 213  This indicates that the relative tolerance was too small. The subroutine has changed it into a suitable value. WARNING ERROR.

IERROR = 215  This indicates that during integration the particular solution or a homogeneous solution has vanished, making a pure relative error test impossible. Must use non-zero absolute tolerance to continue. TERMINAL ERROR.

IERROR = 216  This indicates that during integration the requested accuracy could not be achieved. User must increase error tolerance. TERMINAL ERROR.

IERROR = 218  This indicates that the input parameter N <= 0, or that either the relative tolerance or the absolute tolerance is negative. TERMINAL ERROR.

IERROR = 220  This indicates that IBCTYP = 3, but the recursion is not dichotomic. Check the condition number and the amplification factor. WARNING ERROR.

IERROR = 240  This indicates that the global error is probably larger than the error tolerance due to instabilities in the system. Most likely the problem is ill-conditioned. Output value is the estimated error amplification factor. WARNING ERROR.

IERROR = 250  This indicates that one of the U_k is singular. TERMINAL ERROR.

IERROR = 260  This indicates that the problem is probably too ill-conditioned with respect to the BC. TERMINAL ERROR.

***************
Auxiliary Routines
***************

This routine calls the BOUNDPAK library routines DMUTSG, DMUTSP, DMUTSS, DMATVC and DSVD.
DMUTS is written by G.W.M. Staarink and R.M.M. Mattheij.
Last update: 01-01-1986.

See chapter I of BOUNDPAK User's Manual

Consider the ordinary differential equation

\[
\frac{dx(t)}{dt} = L(t)x(t) + r(t), \quad 0 < t < 6
\]

and a BC: \( M_A x(0) + M_B x(6) = C \), where

\[
L(t) = \begin{bmatrix}
1 - 2\cos(2t) & 0 & 1 + 2\sin(2t) \\
0 & 2 & 0 \\
-1 + 2\sin(t) & 0 & 1 + 2\cos(2t)
\end{bmatrix}
\]

\[
r(t) = \begin{bmatrix}
(-1 + 2\cos(2t) - 2\sin(2t))e^t \\
- e^t \\
(1 - 2\cos(2t) - 2\sin(2t))e^t
\end{bmatrix}
\]

\[
C = \begin{bmatrix}
1 + e^6 \\
1 + e^6 \\
1 + e^6
\end{bmatrix}
\]

\[
M_A = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0
\end{bmatrix}, \quad M_B = \begin{bmatrix}
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0
\end{bmatrix}
\]

The solution to this problem is: \( x(t) = (e^t, e^t, e^t)^T \).

In the next program the solution is computed and compared to the exact solution, using the option IBCTYP = 4.

This program has been run on a AS9000 VM/CMS computer.

```fortran
DOUBLE PRECISION A,B,MA(3,3),MB(3,3),BCV(3),AMP,ER(5),TI(15),
1 X(3,15),U(6,15),Q(3,3,15),WI(3,15),D(3,15),PHIREC(6,15),W(42),
2 EXSOL,AE
INTEGER IW(9)
EXTERNAL FLIN,Fdif
C
C  SETTING OF THE INPUT PARAMETERS
C
N = 3
IHOM = 1
```
ER(1) = 1.0D-11
ER(2) = 1.0D-6
ER(3) = 1.1D-16
NRTI = 10
NTI = 15
NU = 6
LW = 42
LIW = 9
A = 0.0D0
B = 6.0D0

C

SETTING THE BC MATRICES MA AND MB
C

Do 1000 I = 1 , N
Do 1000 J = 1 , N
MA(I,J) = 0.0D0
MB(I,J) = 0.0D0
1000 Continue
MA(1,1) = 1.0D0
MA(2,2) = 1.0D0
MA(3,1) = 1.0D0
MB(1,3) = 1.0D0
MB(2,2) = 1.0D0
MB(3,1) = 1.0D0

C

SETTING THE BC VECTOR BCV
C

BCV(1) = 1.0D0 + DEXP(6.0D0)
BCV(2) = BCV(1)
BCV(3) = BCV(1)

C

CALL DMUTS
C

IBCTYP = 4
CALL DMUTS(FLIN,FDIF,N,IHOM,IBCTYP,KSP,A,B,MA,MB,BCV,ER,AMP,
1 NRTI,NTI,NU,Q,N,WI,D,KPART,PHIREC,W,LW,LIW,
2 IERROR)
1 (IERROR.NE.220).AND.(IERROR.NE.240)) GOTO 5000

C

PRINTING OF IERROR, KSP AND THE BC-MATRICES MA AND MB.
C

WRITE(6,200)
WRITE(6,210) IERROR,KSP
WRITE(6,200)
WRITE(6,220) (MA(I,J),J=1,N)
WRITE(6,230) (MA(I,J),J=1,N)
WRITE(6,220) (MA(J,I),J=1,N)
WRITE(6,200)
WRITE(6,220) (MB(I,J),J=1,N)
WRITE(6,240) (MB(I,J),J=1,N)
WRITE(6,220) (MB(I,J),J=1,N)

C

PRINTING OF THE CONDITION NUMBER AND THE AMPLIFICATION FACTOR.
C

WRITE(6,200)
WRITE(6,245) ER(4),ER(5)
WRITE(6,200)
COMPUTATION OF THE ABSOLUTE ERROR IN THE SOLUTION AND WRITING OF THE SOLUTION AT THE OUTPUT POINTS

WRITE(6,200)
WRITE(6,250)
WRITE(6,200)
DO 1500 K = 1, NRTI
   EXSOL = DEXP(TI(K))
   AE = EXSOL - X(1,K)
   WRITE(6,260) K, TI(K), X(1,K), EXSOL, AE
   DO 1300 I = 2, N
      AE = EXSOL - X(I,K)
      WRITE(6,270) X(I,K), EXSOL, AE
1300 CONTINUE
1500 CONTINUE
STOP
5000 WRITE(6,300) IERROR
STOP

200 FORMAT( ' ')
210 FORMAT( ' IERROR = ',I4,5X,'KSP = ',I2)
220 FORMAT( ' ',5X,D16.9,3X,D16.9,3X,D16.9)
230 FORMAT( ' MA = ',D16.9,3X,D16.9,3X,D16.9)
240 FORMAT( ' MB = ',D16.9,3X,D16.9,3X,D16.9)
245 FORMAT( ' CONDITION NUMBER = ',D10.3,/,1 ' AMPLIFICATION FACTOR = ',D10.3)
250 FORMAT( ' I ',6X,'T',8X,'APPROX. SOL.',8X,'EXACT SOL.',8X,1 'ABS. ERROR')
260 FORMAT( ' ',I3,3X,F7.4,3(3X,D16.9))
270 FORMAT( ' ',I3,3X,D16.9))
300 FORMAT( ' TERMINAL ERROR IN DMUTS: IERROR = ',I4)

SUBROUTINE FLIN(T,Y,F)
DOUBLE PRECISION T,Y(3),F(3)
DOUBLE PRECISION TI,SI,CO

TI = 2.DO * T
SI = 2.DO * DSIN(TI)
CO = 2.DO * DCOS(TI)
P(1) = (1.DO - CO) * Y(1) + (1.DO + SI) * Y(3)
P(2) = 2.DO * Y(2)
P(3) = (-1.DO + SI) * Y(1) + (1.DO + CO) * Y(3)

RETURN

SUBROUTINE FDIF(T,Y,F)
DOUBLE PRECISION T,Y(3),F(3)
DOUBLE PRECISION TI,SI,CO

CALL FLIN(T,Y,F)
TI = 2.DO * T
\[ \text{SI} = 2.0 \times \text{DSIN}(\text{TI}) \]
\[ \text{CO} = 2.0 \times \text{DCOS}(\text{TI}) \]
\[ \text{TI} = \text{DEXP}(T) \]
\[ F(1) = F(1) + (-1.0 \times \text{CO} - \text{SI}) \times \text{TI} \]
\[ F(2) = F(2) - \text{TI} \]
\[ F(3) = F(3) + (1.0 \times \text{CO} - \text{SI}) \times \text{TI} \]

```
RETURN
C END OF FDIF
END
```

<table>
<thead>
<tr>
<th>IERROR</th>
<th>KSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>2</td>
</tr>
</tbody>
</table>

```
MA = 0.000000000000000+00 -0.100000000000000+00 0.000000000000000+00
-0.693889390D-16 0.000000000000000+00 0.000000000000000+00

MB = 0.000000000000000+00 -0.100000000000000+00 0.000000000000000+00
0.707106781D+00 0.000000000000000+00 -0.707106781D+00
```

**CONDITION NUMBER** = 0.100D+01
**AMPLIFICATION FACTOR** = 0.238D+01

```
<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>APPROX. SOL.</th>
<th>EXACT SOL.</th>
<th>ABS. ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.0000</td>
<td>0.403428794D+03</td>
<td>0.403428793D+03</td>
<td>-0.2767673785D-07</td>
</tr>
<tr>
<td>2</td>
<td>5.4000</td>
<td>0.221406416D+03</td>
<td>0.221406416D+03</td>
<td>-0.607166726D-07</td>
</tr>
<tr>
<td>3</td>
<td>4.8000</td>
<td>0.121510417D+03</td>
<td>0.121510418D+03</td>
<td>-0.166990533D-07</td>
</tr>
<tr>
<td>4</td>
<td>4.2000</td>
<td>0.666863100D+02</td>
<td>0.666863100D+02</td>
<td>0.208843661D-07</td>
</tr>
<tr>
<td>5</td>
<td>3.6000</td>
<td>0.365982344D+02</td>
<td>0.365982344D+02</td>
<td>0.109484816D-06</td>
</tr>
<tr>
<td>6</td>
<td>3.0000</td>
<td>0.200855369D+02</td>
<td>0.200855369D+02</td>
<td>0.159975709D-08</td>
</tr>
<tr>
<td>7</td>
<td>2.4000</td>
<td>0.110231764D+02</td>
<td>0.110231764D+02</td>
<td>0.191302697D-06</td>
</tr>
<tr>
<td>8</td>
<td>1.8000</td>
<td>0.604964750D+01</td>
<td>0.604964750D+01</td>
<td>0.103793344D-06</td>
</tr>
<tr>
<td>9</td>
<td>1.2000</td>
<td>0.332011689D+01</td>
<td>0.332011690D+01</td>
<td>0.73084134D-09</td>
</tr>
<tr>
<td>10</td>
<td>0.6000</td>
<td>0.182211870D+01</td>
<td>0.182211880D+01</td>
<td>0.697112947D-07</td>
</tr>
</tbody>
</table>
```

**CONDITION NUMBER** = 0.100D+01
**AMPLIFICATION FACTOR** = 0.238D+01
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>0.0000</td>
<td>0.182211884D+01</td>
<td>0.182211880D+01</td>
<td>-0.398562618D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.999999972D+00</td>
<td>0.100000000D+01</td>
<td>0.276764626D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.999999924D+00</td>
<td>0.100000000D+01</td>
<td>0.756544148D-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.999999945D+00</td>
<td>0.100000000D+01</td>
<td>0.547105300D-07</td>
</tr>
</tbody>
</table>
§ 6 List of BOUNDPAK routine names

The single precision version of BOUNDPAK contains the routines:

SAMBES SBCMAV SCDI SCDIG SCEXIN SCHRHS SCONDW
SCPHIS SCQIZI SCRHOl SCROUT SCWISB SDURU SDRUT
SFCBVVP SFQUS SFUNRC SGKPMP SGTUR SINPRO SINTCH
SKFCH SLUDEC SMATVC SMTSE SMTSI SMTSMP SMTSS
SMUTS SMUTSE SMUTSG SMUTSI SMUTSM SMUTSP SMUTSS
SPSR SQEVAi KS QSEVAL SQUDEC SRFKGS SRKFSM SRFKFS
SSBVVP SSOLDE SSOLUP SSORTD SSVD SUPUP

The double precision version of BOUNDPAK contains the routines:

DAMTES DBCMAV DCIDI DCDIG DCEXIN DCHRHS DCNDW
DCPHIS DCQIZI DCRHOL DCROUT DCWISB DDUUR DDURT
DFCBVP DFQUS DFUNRC DGKPM P DTUR DINPRO DINTCH
DKFCH DLUDEC DMATVC DMTSE DMTSI DMTSMP DMTSS
DMUTS DMUTSE DMUTSG DMUTSI DMUTSM DMUTSP DMUTSS
DPDR DQEVAK DQEVAL DQUDEC DRKFGS DRKFSM DRKFIS
DSBVVP DSOLDE DSOLUP DSORTD DSVD DUPUP