MATHEMATICS OF COMPUTATION, VOLUME 28, NUMBER 128
OCTOBER 1974, PAGES 981–1003

A Variable Order Finite Difference Method for Nonlinear Multipoint Boundary Value Problems

By M. Lentini and V. Pereyra

Abstract. An adaptive finite difference method for first order nonlinear systems of ordinary differential equations subject to multipoint nonlinear boundary conditions is presented. The method is based on a discretization studied earlier by H. B. Keller. Variable order is provided through deferred corrections, while a built-in natural asymptotic estimator is used to automatically refine the mesh in order to achieve a required tolerance. Extensive numerical experimentation and a FORTRAN program are included.

1. Introduction. In this paper, we intend to show how a finite difference technique can be developed to produce high order approximations to the solution of multipoint, nonlinear boundary value problems for first order systems of equations.

We shall present extensive numerical evidence and comparisons with results published in the current literature showing that the method is extremely accurate and that it performs very efficiently.

Moderate accuracy can also be obtained economically in terms of time and storage by working on very coarse meshes. All our results have been obtained with a general purpose program, whose structure can be (and has been) employed in other applications (Pereyra [20]).

Following Keller [11] we consider the nonlinear first order system

$$y'(t) - f(t, y(t)) = 0, \quad a \leqslant t \leqslant b, \tag{1.1a}$$

subject to the multipoint boundary conditions:

$$g(y(\tau_1), \cdots, y(\tau_N)) = 0, \quad a \leqslant \tau_1 < \tau_2 < \cdots < \tau_N \leqslant b. \tag{1.1b}$$

The vector functions $y(t), f(t, y)$, and g will take values in $\mathbf{R}^n$. Considering the nonuniform net $\{t_j\}$:

Received September 17, 1973.

AMS (MOS) subject classifications (1970). Primary 65L10, 65B05, 34–04.

Key words and phrases. Adaptive finite difference method, variable order method, nonlinear multipoint boundary value problem, first order systems boundary problem.

$$t_0 = a, \quad t_j = t_{j-1} + h_j, \quad 1 \leqslant j \leqslant J, \quad t_J = b, \tag{1.2}$$
$$h \equiv \max h_j, \quad \{\tau_i\} \subset \{t_j\},$$

the simple finite difference scheme

$$N_h u_j \equiv \frac{1}{h_j}(u_j - u_{j-1}) - ½[f(t_{j-1}, u_{j-1}) + f(t_j, u_j)] = 0, \quad j = 1, \cdots, J, \tag{1.3a}$$

$$g(u_{j_1}, \cdots, u_{j_N}) = 0, \tag{1.3b}$$

will produce $O(h^2)$ accurate discrete approximations under mild conditions which will be spelled out in Section 2. An asymptotic expansion in even powers of h for the global discretization error $u_j - y(t_j)$ can be shown to exist, and this knowledge justifies the use of deferred corrections which will increment the order of the method in two units per correction, working always on the *same* basic mesh.

The adaptive scheme of Section 4 is designed so that the highest order method, compatible with the current mesh and with increasing returns in accuracy, is always used. The main tool employed to decide which path to follow in the program logical tree is the very natural and effective asymptotic error estimator described in Section 3.

By reduction to first order systems in the usual way, systems of higher order equations can be treated. In this respect, we remark that, in sharp contrast to other high order methods, not only the unknown function but *all* its derivatives up to one unit less than the order of the equation are approximated with the *same* asymptotic order.

From the current literature, we have chosen a set of representative problems used to test variational spline methods, shooting and parallel shooting, and a finite difference technique similar to (1.3), but where high order is achieved via Richardson extrapolation.

Numerical results obtained with our technique are presented in Section 5. In each case, we give pointers to the papers in which the test has been used before, and in a few relevant cases we compare different numerical results. Due to the fact that most numerical tests in this area are published with little detail concerning implementation, computer times, and so on, it is hard to make any final judgement about the relative merits of the different techniques. The ultimate comparison will be that given by the user which will require: ease of use, applicability or adaptability to its particular problem, and overall: economy in computer cost and reliability.

Our program (which is appended) has been developed with these requirements in mind, and we have tried to achieve the quality and high standards of the general purpose software currently available for initial value problems.

In this first stage, we present a version which is not as general as the one described theoretically. We consider only linear two-point boundary conditions of the form $Ay(a) + By(b) = \alpha$ and uniform meshes.

However, we have used (Section 6) a variation of the program (not presented here) which handles a jump discontinuity. It is fairly clear that a general program can be written with a moderate amount of additional work. We are really waiting to develop an effective automatic procedure for choosing nonuniform meshes before undertaking a more general program.

2. Keller's Results for the Basic Method [8], [9], [11].

The main theoretical support for our method is provided by the thorough analysis that H. B. Keller has made of the second order scheme (1.3), and by the general theory of deferred corrections developed by the second author of this paper [16]. For completeness, we shall now describe the minimum material necessary to present our results.

A solution $y^*(t)$ of (1.1) is said to be *isolated* if the linearized problem (around y^*) has a unique solution. We assume that (1.1) has an isolated solution $y^*(t)$. Then, for sufficiently small h_0 and all $h \leqslant h_0$, we have:

(i) The difference equation (1.3) has a unique solution in a neighborhood of $\{y^*(t_i)\}$, which can be computed by Newton's method. The convergence is quadratic for appropriate initial values.

(ii) $\max_j \|u_j - y^*(t_j)\| = O(h^2)$.

(iii) $u_j - y^*(t_j) = \Sigma_{\nu=1}^{m} h^{2\nu} e_\nu(t_j) + O(h^{2m+2}), j = 0, \cdots, J$.

(iv) Writing (1.3) in vector form

$$\Phi_h(U) = 0, \quad \text{with} \quad U = (u_0, \cdots, u_J)^T, \quad \text{and}$$

$$\Phi_h(U) = \begin{bmatrix} g(u_{j_1}, \cdots, u_{j_N}) \\ N_h u_1 \\ \vdots \\ N_h u_J \end{bmatrix}$$

we have the stability condition

$$\|U - V\| \leqslant C\|\Phi_h(U) - \Phi_h(V)\|, \quad C \text{ independent of } h, \tag{2.1}$$

(which can be readily obtained from (3.4a) of [11]).

Most of these results can be extended to the important case in which the data functions f and g are only piecewise smooth, with jump discontinuities allowed at the boundary points $\{\tau_j\}$.

We shall also need the Fréchet derivative (Jacobian matrix) of the operator $\Phi_h(U)$. This matrix has the following block structure:

$$\Phi'_h(U) = \left[\begin{array}{c|cccc} G_0 & G_1 & \cdots & & G_J \\ \hline S_1 & R_1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & & & S_J & R_J \end{array}\right] \tag{2.2}$$

where all the submatrices are of size $n \times n$, and

$$G_j = (\partial/\partial u_j)g, \qquad j = 0, \cdots, J,$$
$$S_j = -\left[\frac{1}{h_j}I + \frac{1}{2}f'_{j-1}\right], \qquad R_j = \frac{1}{h_j}I - \frac{1}{2}f'_j, \qquad j = 1, \cdots, J,$$
$$[f'_j]_{st} = (\partial f_s/\partial u_t)(t_j, u_j).$$

3. Deferred Corrections and Asymptotic Error Estimates.

By using Taylor series we can easily obtain an asymptotic expansion for the local discretization error $\tau_h(y^*) \equiv \Phi_h(y^*)$. In fact,

$$\tau_h(y^*)(t_{j-1} + \tfrac{1}{2}h_j) = -\sum_{\nu=1}^{L} \frac{\nu}{2^{2\nu-1}(2\nu+1)} \cdot f^{(2\nu)}_{j-1/2} \frac{h_j^{2\nu}}{(2\nu)!} + O(h^{2L+2}), \tag{3.1}$$

where

$$f^{(2\nu)}_{j-1/2} \equiv f^{(2\nu)}(t_{j-1} + \tfrac{1}{2}h_j, y^*(t_{j-1} + \tfrac{1}{2}h_j)).$$

Let $F_k(y^*)$ be the segment of the expansion (3.1) containing its first k terms. For each $1 \leqslant j \leqslant J$, let i_j be the only index for which

$$\tau_{i_j} < t_{j-1} + \tfrac{1}{2}h_j < \tau_{i_j+1}. \tag{3.2}$$

Then we define S_k as

$$S_k(y^*)(t_{j-1} + \tfrac{1}{2}h_j) \equiv \sum_{i=0}^{2k} w_{j,i} f_{j-p_j+i}, \tag{3.3}$$

where the p_j are chosen so that

$$\tau_{i_j} \leqslant t_{j-p_j+i} \leqslant \tau_{i_j+1}. \tag{3.4}$$

is satisfied. The weights $w_{j,i}$ are chosen so that

$$S_k(y^*) = \tau_k(y^*) + O(h^{2k+2}) \tag{3.5}$$

at each $t_{j-1} + \frac{1}{2}h_j, j = 1, \cdots, J$.

Clearly, (3.4) imposes a condition on the mesh:

CM: There must be at least $(2k - 1)$ mesh points between boundary points.

Though it is not strictly necessary to require (3.4) in the case we are considering at present, we prefer to assume (3.4) to hold since this will be essential in the case of piecewise smooth data, where we must avoid straddling a singularity in order to obtain the desired accuracy (cf. Section 6).

Once CM is assured, (3.3) can always be constructed since it is simply a numerical differentiation formula applied to each component of the (perhaps piecewise smooth) vector function f. With a small modification, the correction generator of [19] can be used for this purpose (cf. also [16], [17]).

From [16], [18], it follows that, if $Y^{(k-1)}$ is an $O(h^{2k})$ accurate discrete solution, then (3.5) is satisfied if y^* is replaced by $Y^{(k-1)}$ and that, for $k \geqslant 1$, the solution $\Delta^{(k-1)}$ of the linear problem

$$\Phi_h'(Y^{(k-1)})\Delta = S_{k-1}(Y^{(k-2)}) - S_k(Y^{k-1}) \tag{3.6}$$

is an asymptotic error estimator for $e_{k-1} \equiv Y_{k-1} - \phi_h y^*$, where $S_0 \equiv 0$, and ϕ_h projects $y^*(x)$ on the mesh functions. In fact, we shall have

$$\Delta_{k-1} = e_{k-1} + O(h^{2k+2}). \tag{3.7}$$

The successively more accurate mesh solutions $Y^{(k)}$ are obtained by deferred corrections, i.e., by solving for $Y^{(k)}$ the nonlinear problems:

$$\Phi_h(Y) = S_h(Y^{(k-1)}), \qquad k = 1, \cdots. \tag{3.8}$$

These nonlinear problems can be solved by Newton's method, i.e., by the iteration:

$$Y_{l+1} = Y_l - [\Phi_h'(Y_l)]^{-1}\Phi_h(Y_l), \tag{3.9}$$

starting from an appropriate Y_0. Naturally, each step is performed by solving a linear system entirely similar to (3.6). Because of the special structure of the sparse matrix Φ_h', there are various direct methods which can be employed. In the case of separated two-point boundary conditions, it is advisable to use the band or block tridiagonal methods of Varah [25] and Keller [11].

Clearly, the difficulties in the asymptotic theory observed before in the simple two-point boundary value problem [19], because of the use of different differentiation formulas at different points, are also present in this case at each subinterval. However, as in the simpler problem, we hope to show with our numerical experimentation that, notwithstanding these theoretical difficulties, this is a very effective technique.

Solution of the Linear Systems of Equations. We shall describe briefly the direct solution of block systems of the form (2.2). We have included in (2.2) an extra subdivision in order to treat the 2×2 "super-block" matrix:

$$\Phi' = \begin{array}{cc} \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] & \begin{array}{l} \}\, n \\ \}\, n*J \end{array} \\ \begin{array}{cc} \underbrace{\quad}_{n} & \underbrace{\quad}_{n*J} \end{array} & \end{array} \tag{3.10}$$

Considering the partitioned vectors

$$\mathrm{x} = \left[\begin{array}{c} \mathrm{x}_0 \\ \hline \hat{\mathrm{x}} \end{array}\right] = \left[\begin{array}{c} \mathrm{x}_0 \\ \hline \mathrm{x}_1 \\ \vdots \\ \mathrm{x}_J \end{array}\right], \quad \mathrm{b} = \left[\begin{array}{c} \mathrm{b}_0 \\ \hline \hat{\mathrm{b}} \end{array}\right] = \left[\begin{array}{c} \mathrm{b}_0 \\ \hline \mathrm{b}_1 \\ \vdots \\ \mathrm{b}_J \end{array}\right],$$

the super-block system

$$\left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] \left[\begin{array}{c} \mathrm{x}_0 \\ \hline \hat{\mathrm{x}} \end{array}\right] = \left[\begin{array}{c} \mathrm{b}_0 \\ \hline \hat{\mathrm{b}} \end{array}\right], \tag{3.11}$$

is solved by elimination. More explicitly:

$$\mathrm{x}_0 = (A - BD^{-1}C)^{-1}(\mathrm{b}_0 - BD^{-1}\hat{\mathrm{b}}), \tag{3.11a}$$

$$D\hat{\mathrm{x}} = (\hat{\mathrm{b}} - C\mathrm{x}_0). \tag{3.11b}$$

The main part of the computation is the solution of the block-bidiagonal systems with the matrix D, corresponding to the computation of $D^{-1}C, D^{-1}\hat{\mathbf{b}}$. By putting $\tilde{C} = [C|\hat{\mathbf{b}}]$, $\tilde{V} = [V|\mathbf{w}]$, where $V = D^{-1}C$, $\mathbf{w} = D^{-1}\hat{\mathbf{b}}$, we have that the system $D\tilde{V} = \tilde{C}$ is solved by the recursion:

$$\tilde{V}_j = R_j^{-1}(\tilde{C}_j - S_j\tilde{V}_{j-1}), \quad j = 1, \cdots, J, \tag{3.12}$$

where $\tilde{V}_0 \equiv 0$, and the $\tilde{V}_j$, $\tilde{C}_j$ correspond to the appropriate partitionings of $\tilde{V}, \tilde{C}$. Naturally, the matrices R_j are not inverted, but, rather, a good Gaussian elimination code with pivoting is used to solve the corresponding matrix systems. This provides what Keller [11] calls partial pivoting. With $\tilde{V}$, (3.11) reduces to the solution of the linear system

$$(A - BV)\mathbf{x}_0 = (\mathbf{b}_0 - B\mathbf{w}), \tag{3.13a}$$

$$\hat{\bar{\mathbf{x}}} = \mathbf{w} - V\mathbf{x}_0. \tag{3.13b}$$

Observe that in most cases the block-vector B will be quite sparse since there will usually be many more grid points than boundary points. This should be taken into account in the computation.

4. The Adaptive Method. In [19], a variable order algorithm based on results similar to those of Section 3 was developed for the two-point boundary value problem for second order equations. It was indicated there that the scheme had more general applications, as we shall proceed to show now (cf. [20] also).

The problem we set ourselves to solve is the following: "Given a boundary value problem (1.1), a basic mesh Ω_h (containing the boundary points τ_i, $i = 1, \cdots, N$), and a tolerance TOL, find an approximate solution Y^* defined (at least) on Ω_h and satisfying

$$\|Y^* - \phi_h y^*\| \leqslant \text{TOL.''} \tag{4.1}$$

The basic mesh Ω_h is the region in which the user wants to know the solution (minimal description):

$$\Omega_h = \{t_j\}_{j=0,\ldots,J}. \tag{4.2}$$

We define the indices j_i, by

$$\begin{aligned} t_{j_i} &= \tau_i, & i &= 1, \cdots, N, \\ \nu_i &= j_{i+1} - j_i + 1, & i &= 1, \cdots, N-1, \end{aligned} \tag{4.3}$$

and

$$\eta = \min_i \nu_i.$$

By $\Omega_{h/2}$, we shall denote the refinement of Ω_h obtained by including all the midpoints $t_{j-1} + \frac{1}{2}h_j$. Thus, J becomes $2J$.

Algorithm. Let $k = 0$.

If $\eta \geqslant 4$, then we can compute S_1 (cf. (3.1) and (3.5)). In that case, on Ω_h, we solve $\Phi_h(Y) = 0$ for $Y^{(0)}$, and also solve $\Phi'_h(Y^{(0)})\Delta = -S_1(Y^{(0)})$ for $\Delta^{(0)}$. Since $\|\Delta^{(0)}\|$ is an error estimate for $\|Y^{(0)} - \phi_h y^*\|$, we check if $\|\Delta^{(0)}\| \leqslant$ TOL and, if this condition is satisfied, we exit successfully.

If $\eta < 4$, then the first step just described cannot be performed and we refine the mesh and try again. If OLDERROR $\equiv \|\Delta^{(0)}\| >$ TOL, then we enter in the general correction loop:

```
Correction loop: set $k$ equal to $k + 1$;
if $\eta < 2k + 2$ then refine the mesh;
otherwise solve for $Y^{(k)}$ the nonlinear equation:

    $\Phi_h(Y) = S_k(Y^{(k-1)})$.

Compute and save $S_{k+1}(Y^{(k)})$.
Solve for $\Delta^{(k)}$ the linear equation:

    $\Phi'_h(Y^{(k)})\Delta = S_k(Y^{(k-1)}) - S_{k+1}(Y^{(k)})$;

if NEWERROR $\equiv \|\Delta^{(k)}\| \leqslant$ TOL, then exit successfully;
otherwise if NEWERROR $\leqslant C *$ OLDERROR (where $0 < C \leqslant 1$) then
set OLDERROR to NEWERROR and go to Correction loop;
otherwise refine the mesh end.
```

The strategy behind this algorithm is that the highest order method compatible with the current mesh is always used, unless the level of diminishing returns is reached and no further improvement is obtained by increasing the order on the present mesh. This last decision corresponds to the condition NEWERROR $\leqslant C *$ OLDERROR, where the constant C measures the minimum rate of improvement required of a correction in order to continue on the given mesh. This strategy is dictated by the accumulated experience on multiple applications that indicates that greater efficiency is achieved in this way than by refining the mesh prematurely. (Recall that the dimensionality of the problem increases when the mesh is refined.)

Another important feature, especially for nonlinear problems, is the following. After the very first step on the basic mesh, where, usually, we will not have good initial values, we can count on accurate initial values for starting all the successive iterations. In fact, to solve the equations $\Phi_h(Y) = S_k(Y^{(k-1)})$, we can use as starting values $Y^{(k-1)}$ itself, while, upon refinement of the mesh, we can use the latest value of Y on the coarse mesh, plus values interpolated from them for the missing points in the new mesh.

The error estimate NEWERROR is profitably used in two ways, aside from the one already mentioned above. After the first step of the process, we use it to set the level, at which the residual in the solution of the nonlinear equations, must be reduced, in the next step. When the grid is refined, the degree of interpolation used to produce initial values for the new grid points, and the level of correction at which the process will start, are also decided on the basis of information related to earlier estimates. When convergence of Newton's method cannot be achieved due to lack of information to start the process, one might be forced to resort to more elaborate techniques, as we exemplify in Section 7.

5. Numerical Results.

In this section, we shall report on a fairly extensive set of tests, mostly collected in the open literature. In all cases, we write the equations as first order systems, although in the references they might have been treated as high order equations. All results have been obtained on an IBM/360 model 50 computer working with long words ($\approx$ 16 decimal digits), using the FORTRAN program SYSSOL listed in the Appendix. There are two user parameters that must be given to SYSSOL: TOL = user's desired accuracy (see 4.1) and N = number of points in the initial mesh.

Problem 1.

$$\begin{aligned} y_1' &= y_2, \\ y_2' &= y_1^3 - \sin t\,(1 + \sin^2 t), \\ y_1(0) &= y_1(\pi) = 0. \end{aligned}$$

Exact solution. $y_1(t) = \sin t; y_2(t) = \cos t$.

In [19], an adaptive method for second order equations was developed using as a basic discretization the $O(h^4)$ Milne-Numerov formula. Results obtained with SYSSOL are listed in Table 1. The user parameters were: TOL = 5×10^{-15} and $N = 9$.

TABLE 1

	Final estimated error	Final true error	Number of corrections	Final mesh size
SYSSOL	3.2, – 15	2.2, – 15	6	33
[19]	7.0, – 17	2.8, – 15	3	33

We observe in Table 1 that very similar results are obtained with both methods, but, reflecting the fact that 4 orders are gained per correction, the method of [19] requires only half the number of corrections. Computer times are unfortunately non-comparable, since the results for [19] were obtained on an IBM/360 Model 91. However, we believe that the technique of [19] should be preferred whenever it applies. As a matter of reference, the computer time on an IBM/360/50 for SYSSOL was 13.12 sec.

Problem 2.

$$y_1' = y_2,$$
$$y_2' = 400(y_1 + \cos^2 \pi t) + 2\pi^2 \cos 2\pi t,$$
$$y_1(0) = y_1(1) = 0.$$

Exact solution.

$$y_1(t) = \frac{e^{-20}}{1+e^{-20}} e^{20t} + \frac{1}{1+e^{-20}} e^{-20t} - \cos^2 \pi t,$$

$$y_2(t) = \frac{20e^{-20}\, e^{20t}}{1+e^{-20}} - \frac{20}{1+e^{-20}} e^{-20t} + \pi \sin 2\pi t.$$

In Stöer and Bulirsch [24, Chapter 2, §6], this example is used to compare the following methods:

(Ma) Simple shooting method (obviously, the example is designed to fail for this method, and so it does);

(Mb) Multiple shooting of Bulirsch;

(Mc) $O(h^2)$ finite difference method for second order equations;

(Md) Variational method using cubic splines.

We thank Professor Stöer for making his results available to us before [24] was ready.

In Table 2, we compare the maximum absolute errors of the various methods with those obtained by SYSSOL, with TOL $= 5 \times 10^{-11}$, $N = 65$.

TABLE 2

	Max. abs. error	Comments
Ma	1.3, − 3	–
Mb	5.0, − 12	20 intermediary points.
Mc	5.6, − 6	2^{10} mesh points.
Md	1.8, − 6	100 subintervals.
SYSSOL	9.9, − 12	65 mesh points; 7 corrections. 21.74 sec. of computing time.

Problem 3.

$$y_1' = y_2,$$
$$y_2' = e^{y_1},$$
$$y(0) = y(1) = 0.$$

Exact solution.

$$y_1(t) = -\ln 2 + 2\ln\left[c * \sec\left(\frac{c}{2}\left(t - \frac{1}{2}\right)\right)\right],$$
$$y_2(t) = c * \tan\left(\frac{c}{2}\left(t - \frac{1}{2}\right)\right),$$

where c satisfies $c * \sec c/4 = \sqrt{2}$. To 16 significant figures, $c = 1.336055694906108\ldots$.

This problem has been solved by a variety of techniques in [2], [5], [7], [11], [19 [21]. In Table 3, we present some comparative figures. A description of the various methods follows (in some cases, we have chosen only the most accurate results):

M1: Ritz-Galerkin with polynomial subspaces $P_0^{(N)}$.
Basis: indefinite integrals of Legendre polynomials.
Iterative method: Gauss-Seidel-Newton [2].

M2: Ritz-Galerkin with cubic Hermite subspaces $H_0^{(2)}$, coupled with four-point Gaussian quadrature scheme [5].

M3: Ritz-Galerkin with smooth cubic splines $S_p(D^2, \Delta(h), \bar{z})$ [7].

M4: Keller's method with Richardson's extrapolations, [11].

M5: $O(h^8)$, Milne-Numerov, linear deferred corrections [19].

M6: Milne-Numerov with successive extrapolations [19].

M7: Adaptive deferred corrections (SYSSOL).

We report max. abs. error for each method.

TABLE 3

Method	Error	Comments
M1	5.03, – 8	Dimension of $P_0^{(N)} = 6$.
M2	6.28, – 8	Dimension of $H_0^{(2)} = 24$.
M3	7.15, – 7	Dimension of $S_p^{(2)} = 16$.
M4	1.09, – 11	Three extrapolations. Basic mesh, $h = 1/3$.
M5	7.36, – 10	$N = 8$.
M6	4.01, – 12	Two extrapolations. Basic mesh, $h = 1/4$.
M7	5.35, – 12	$k = 2$. $N_0 = 9$. N final = 17. Time on IBM 360/50: 4.1 seconds.
M7	3.98, – 15	$k = 4$. $N_0 = 17$. N final = 33. Computer time: 8.76 seconds.

We observe that the only results with an accuracy comparable to SYSSOL are those obtained with successive extrapolations (a very near cousin!) but that, as usual, the accurate results correspond only to the coarsest mesh used (with 2 and 3 interior points respectively in M4 and M6).

Problem 4. (Bending of a thin beam clamped at both ends.)

$$\begin{aligned} y_1' &= y_2, \\ y_2' &= y_3, \\ y_3' &= y_4, \\ y_4' &= (t^4 + 14t^3 + 49t^2 + 32t - 12)e^t, \end{aligned}$$

$$y_1(0) = y_2(0) = y_1(1) = y_2(1) = 0.$$

Exact solution. $y(t) = t^2(1 - t)^2 e^t$.

In [2], [5], this problem is solved by a variational method using smooth Hermite subspaces $H_0^{(2)}(\pi)$ of piecewise cubic polynomials.

In Table 4, we compare max. abs. errors for the solution and its first derivative. The value of k indicates the final number of correction terms.

TABLE 4

Method	Max. abs. error function	Max. abs. error derivative
Ritz-Galerkin $H_0^{(2)}$ of dim. 46	1.70, − 6	1.27, − 4
SYSSOL 17 points $k = 2$	4.70, − 7	9.03, − 7
SYSSOL 33 points $k = 6$	1.82, − 14	9.65, − 15

The computer time on an IBM 360/50 for the most accurate results was of 13.82 sec., using 138 K.-bytes of main storage.

Problem 5.

$$\begin{aligned}
y_1' &= y_2,\\
y_2' &= \beta(y_1 - y_3),\\
y_3' &= y_4,\\
y_4' &= \alpha(y_3 - y_1),\\
y_1(0) &= y_4(0) = y_2(s) = 0; \qquad y_4(s) = c,
\end{aligned}$$

with $s = 10, c = 10^{-3}, \alpha = \beta = 2.5$.

Exact solution.

$$\begin{aligned}
y_1 &= \frac{\beta c}{r^2}\left[\gamma/r + t - \gamma\cosh(rt)/r + \frac{\beta}{\alpha}\sinh(rt)/r\right],\\
y_2 &= \frac{\beta c}{r^2}\left[1 - \gamma\sinh(rt) + \frac{\beta}{\alpha}\cosh(rt)\right],\\
y_3 &= \frac{c}{r^2}\left[\beta\gamma/r + \beta t + \alpha\gamma\cosh(rt)/r - \beta\sinh(rt)/r\right],\\
y_4 &= \frac{c}{r^2}\left[\beta + \alpha\gamma\sinh(rt) - \beta\cosh(rt)\right],
\end{aligned}$$

where

$$r = \sqrt{\alpha + \beta}, \qquad \gamma = \left(\frac{\beta}{\alpha}\cosh(rs) + 1\right) / \sinh(rs).$$

In [3], Falkenberg solves this problem by a method he calls "step wise inversion" which is related to the Godunov-Conte method. This is also a problem which is unstable for simple shooting. In Table 5 we show again max. abs. errors for the various components as obtained with Falkenberg's algorithm and with SYSSOL.

TABLE 5

Method	max. abs. error in y_1	max. abs. error in y_2	max. abs. error in y_3	max. abs. error in y_4
Falkenberg 10 steps	10^{-8}	10^{-9}	10^{-8}	10^{-9}
SYSSOL 33 points $k = 7$	6×10^{-11}	1.5×10^{-10}	3.3×10^{-11}	6.4×10^{-11}

Computer time on IBM 360/50 was 36.10 sec. for $N_0 = 9$.

Finally, in Table 6, we present the results of a fairly extensive set of tests, which shows the behavior of SYSSOL on the five problems of this section for different tolerances and initial step sizes.

TABLE 6

Problem	1					2				
TOL \ N_0	5	9	17	33	65	5	9	17	33	65
10^{-3}	.96 (9)	$\boxed{.83\ (9)}$	1.61 (17)	2.01 (33)	4.57 (65)	9.36 (33)	7.42 (33)	6.27 (33)	$\boxed{4.84\ (33)}$	8.51 (65)
10^{-6}	3.27 (17)	3.21 (17)	$\boxed{2.89\ (17)}$	3.19 (33)	6.19 (65)	15.09 (33)	13.87 (33)	10.73 (33)	$\boxed{10.69\ (33)}$	13.39 (65)
10^{-9}	5.07 (17)	5.2 (17)	$\boxed{4.75\ (17)}$	6.73 (33)	11.48 (65)	26.89 (65)	26.77 (65)	25.39 (65)	26.09 (65)	$\boxed{17.18\ (65)}$

Problem	3					4				
TOL \ N_0	5	9	17	33	65	5	9	17	33	65
10^{-3}	.56 (9)	$\boxed{.38\ (9)}$	.73 (17)	1.41 (33)	4.90 (65)	3.16 (9)	$\boxed{2.51\ (9)}$	3.22 (17)	6.40 (33)	13.11 (65)
10^{-6}	1.38 (9)	$\boxed{1.03\ (9)}$	1.26 (17)	2.81 (33)	6.45 (65)	6.65 (17)	$\boxed{6.24\ (17)}$	6.57 (17)	9.86 (33)	18.98 (65)
10^{-9}	2.61 (17)	2.57 (17)	$\boxed{2.51\ (17)}$	4.01 (33)	7.85 (65)	10.71 (17)	10.23 (17)	$\boxed{8.83\ (17)}$	14.56 (33)	18.68 (65)

Problem	5				
TOL \ N_0	5	9	17	33	65
10^{-3}	2.22 (9)	1.73 (9)	$\boxed{1.61\ (17)}$	3.37 (33)	7.23 (65)
10^{-6}	18.02 (33)	17.50 (33)	14.04 (33)	$\boxed{11.56\ (33)}$	13.09 (65)
10^{-9}	29.81 (33)	29.12 (33)	25.07 (33)	$\boxed{23.32\ (33)}$	26.35 (65)

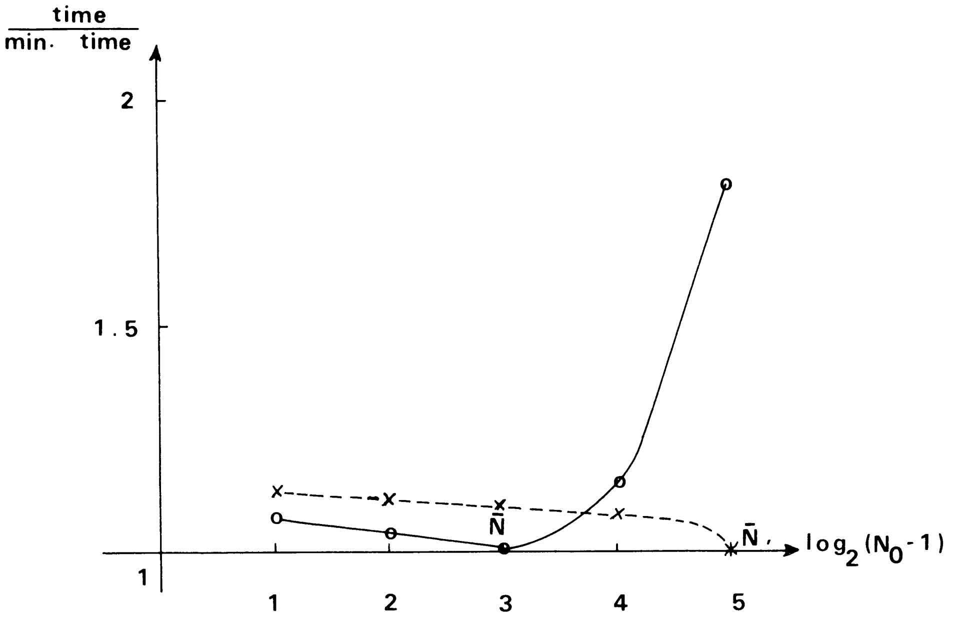
Time in seconds needed to reach the indicated TOLerances in Problems 1 to 5. The number in parentheses is the final number of points in the mesh. The boxes indicate the minimum time for fixed problem and tolerance.

The tolerances chosen (values of TOL) could be described as low, medium and medium-high, while, in the individual results already given, we exemplified the results for high precision (on this computer). These results are important since they show that the algorithm is not geared exclusively towards high precision, which might be inadequate in many present day applications, but also performs economically at "engineering precisions".

We observe in Table 6 that for each problem and a given tolerance (horizontal lines), the final number of mesh points N_f is independent of the initial one N_0, until $N_0 > N_f$, when they start coinciding. What is more important, the minimum time, for given problem and tolerance (marked by a box), is attained when N_0 reaches N_f. We shall call the mesh with this number of points $\overline{N}(\mathrm{TOL})$: the optimal mesh for the problem (and tolerance). There is only one exception in the 15 cases shown: Problem 4, TOL =

10^{-6}. However, we see that the difference in times falls in the area of uncertainty due to the multiprogramming environment (about 10%). Therefore, it would be quite important, from the point of view of this algorithm, to be able to predict $\bar{N}(\mathrm{TOL})$ early in the game.

Another point shown in this table is that underestimating $\bar{N}(\mathrm{TOL})$ is less costly than overestimating it. Very schematically, we can subsume the results of Table 6 in the following diagram, where we present the curves time/(minimum time) versus $\log_2(N_0 - 1)$ for two hypothetical, though typical, cases.



6. **Piecewise Smooth Data.** In [9], Keller develops, in all detail, the theory mentioned in Section 2, restricted to the linear case but allowing jump discontinuities in the function $f(t, y(t)) \equiv A(t)y(t) + g(t)$. The only restriction is that those discontinuities must be limited to occur on the set of boundary points. In Section 3, we introduced a limitation stronger than necessary in the way by which the correction operators S_k were calculated. This was done foreseeing the extension to the piecewise smooth case. In fact, the only care we must exert in order that the whole theory (and practice) of deferred correction holds true in this case, is not to straddle discontinuities in the calculation of the S_k. By working systematically on the smooth subintervals, all the necessary expansions are valid (cf. [9]).

The only small modifications that must be introduced in the general routine are due to the fact that, at discontinuity points, we must use the proper information.

For instance, let $t_{j_i} = \tau_i$ be a discontinuity point of $f(t, y(t))$, and let $f_{j_i}^-, f_{j_i}^+$ be the respective one-sided limits. Then

$$N_h u_{j_i} \equiv \frac{1}{h_{j_i}}(u_{j_i} - u_{j_i-1}) - \frac{1}{2}[f_{j_i-1} + f_{j_i}^-],$$

and

$$N_h u_{j_i+1} \equiv \frac{1}{h_{j_i+1}}(u_{j_i+1} - u_{j_i}) - \frac{1}{2}[f_{j_i}^+ + f_{j_i+1}].$$

Similar care must be exerted in the implementation of formula (3.3) and in the computation of the Jacobian. However, the same code as for the smooth case can be used for computing the S_k at each subinterval $[\tau_i, \tau_{i+1}]$, i.e., for $(t_{j-1} + ½h_j)$, $j = j_i + 1, \cdots, j_{i+1}$.

Problem 6.

$$y_1' = y_2,$$
$$y_2' = y_3,$$
$$y_3' = y_4,$$
$$y_4' = \begin{cases} 24, & 0 \leqslant x \leqslant 1/2, \\ 48, & 1/2 \leqslant x \leqslant 1, \end{cases}$$

$$y_1(0) = y_2(0) = y_1(1) = y_2(1) = 0.$$

Exact solution.

$$y_1(t) = \begin{cases} t^4 - \frac{19}{8}t^3 + \frac{21}{16}t^2, & 0 \leqslant t \leqslant 1/2, \\ 2(t-1)^4 + \frac{29}{8}(t-1)^3 + \frac{27}{16}(t-1)^2, & 1/2 \leqslant t \leqslant 1. \end{cases}$$

$$y_2(t) = \begin{cases} 4t^3 - \frac{57}{8}t^2 + \frac{21}{8}t, & 0 \leqslant t \leqslant 1/2, \\ 8(t-1)^3 + \frac{57}{8}(t-1)^2 + \frac{27}{8}(t-1), & 1/2 \leqslant t \leqslant 1. \end{cases}$$

$$y_3(t) = \begin{cases} 12t^2 - \frac{57}{4}t + \frac{21}{8}, & 0 \leqslant t \leqslant 1/2, \\ 24(t-1)^2 + \frac{57}{4}(t-1) + \frac{27}{8}, & 1/2 \leqslant t \leqslant 1. \end{cases}$$

$$y_4(t) = \begin{cases} 24t - 57/4, & 0 \leqslant t \leqslant 1/2, \\ 48(t-1) + 57/4, & 1/2 \leqslant t \leqslant 1. \end{cases}$$

In [21], this discontinuous problem is solved by a variational method using cubic Hermite spaces $H_0^{(2)}$. In that paper, a comparison is made with a finite difference method, showing the dangers of a naive approach to the problem. In Table 7 we compare the variational method with our finite difference method.

TABLE 7

Method	max. abs. error	Comments
Variational	1.25, – 5	Dimension of $H_0^{(2)} = 18$
Five-point finite differences	8.53, – 3	79 mesh points
SYSSOL	4.43, –6	9 points 1 correction
SYSSOL	1.39, – 17	33 points 5 corrections

One of the main points made in [21], when comparing the variational method with the finite difference method, was that while the former method showed a perfect asymptotic behavior, the latter failed to show even second order convergence, and computations on various meshes had a very erratic behavior. The reason for these results is apparent to us now: the 5-point finite difference method straddled the singularity at 1/2 for points near it, while the cubic Hermite method, being essentially a two-point method did not. That is the reason why our finite difference method is also impervious to the jump discontinuity. Our last piece of evidence is to show then that our method has the proper asymptotic behvaior, and we do that in Table 8.

N is the number of mesh points, k is the correction number, and the number in parentheses after a correction column is the computed order of the method for that column. The theoretical order for correction k is $O(h^{2k+2})$.

TABLE 8

$N\backslash k$	0		1		2		3	
9	6.05, – 3	–	4.43, – 6	–	–	–	–	–
17	1.53, – 3	(2.0)	2.75, – 7	(4.0)	1.08, – 9	(6.0)	4.22, – 12	–
33	3.82, – 4	(2.0)	1.72, – 8	(4.0)	1.68, – 11	(6.0)	1.65, – 14	(8.0)
65	9.56, – 5	(2.0)	1.07, – 9	(4.0)	2.62, – 13	(6.0)	6.94, – 17	(7.9)

Problem 6 is linear and has a piecewise polynomial solution: a fairly favorable case.

The following is a nonlinear problem with a nonpolynomial solution with discontinuous second derivative.

Problem 7.

$$y_1' = y_2,$$

$$y_2' = \begin{cases} -e^{y_1}/x^3, & 1 \leqslant x < 1.5, \\ 0, & 1.5 \leqslant x \leqslant 2, \end{cases}$$

$$y_1(1) = 0, \quad y_2(2) = 2/3.$$

Exact solution.

$$y_1(x) = \begin{cases} \ln x, & 1 \leqslant x < 1.5, \\ \frac{2}{3}x + \ln 1.5 - 1, & 1.5 \leqslant x \leqslant 2, \end{cases} \qquad y_2(x) = \begin{cases} 1/x, & 1 \leqslant x < 1.5, \\ 2/3, & 1.5 \leqslant x \leqslant 2. \end{cases}$$

User parameters for this problem were $N_0 = 65$, $\mathrm{TOL} = 5 \times 10^{-15}$. With four corrections on this mesh and a total of eight Newton iterations the tolerance was met, using 18.07 sec. of computing time.*

The results of this section were obtained with a more primitive (and modified) version of SYSSOL, and are reported only as a matter of reference.

7. Use of a Continuation Method for Stubborn Problems.

In [15], [22], [23], some more challenging problems appear. These are "horror" problems generally appearing in practical applications which have resisted the action of most methods. Some of them are impervious to the use of shooting methods, while others present difficulties in the convergence of the iterations used for the solution of the nonlinear equations that occur in the various methods. Difficulties with the simple shooting method have been avoided in many cases by resorting to the more sophisticated technique of parallel shooting [8], [15], which is essentially a hybrid, combining shooting with finite differences. As we have already shown in Problem 2, our algorithm can also overcome the difficulties originated by unstable or stiff systems. In the following problem, however, we found for the first time divergence in Newton's method, when starting from our usual crude values $y_i(t_j) \equiv 0$. Thus, we have been forced to employ a more sophisticated technique for solving the nonlinear equations.

Problem 8. (A boundary layer problem.)

$$y_1' = y_2,$$

$$y_2' = y_3,$$

$$y_3' = -1.55y_1y_3 + .1y_2^2 + 1 - y_4^2 + .2y_2,$$

$$y_4' = y_5,$$

$$y_5' = -1.55y_1y_5 + 1.1y_2y_4 + .2(y_4 - 1),$$

$$y_1(0) = y_2(0) = y_4(0) = y_2(3.5) = 0, \quad y_4(3.5) = 1.$$

* We acknowledge here the tele-debugging abilities of Professor H. B. Keller who discovered an error in the Jacobian matrix in an earlier version, without ever seeing the program.

Exact solution. Unknown.

In [6], Holt presents a numerical solution in graphic form obtained via an ad hoc finite difference method and reports difficulties obtaining initial values. In [15], Osborne uses parallel shooting with success depending again upon the initial values, but, unfortunately, no information is given about the computed solution. In [23], Roberts, Shipman and Ellis replace the system by

$$y' = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0.2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0.2 & 0 \end{bmatrix} y + \epsilon_\nu g(t, y) \tag{7.1}$$

$$\equiv Cy + \epsilon_\nu g(t, y),$$

where $Cy + g \equiv f$. They use a continuation method [14] consisting of setting $\epsilon_\nu = \epsilon_{\nu-1} + \Delta\epsilon$, $\nu = 1, \cdots$, $\epsilon_0 = 0$, and solving the intermediate problems by simple shooting until ϵ_ν reaches the value 1, where the original problem is recovered. A table of the computed missing boundary values is offered, though no mention of their accuracy is made. In the intermediate problems, (it seems), ten iterations are performed, presumably each one of them costing the integration of an initial value problem. No indication of computer times are given.

We have chosen to use a variation of this procedure in which only *one* Newton step is performed for each ϵ_ν, since our aim is simply to provide initial values to start a successful iteration for $\epsilon_\nu = 1$. This goal has been achieved and highly accurate results have been obtained as we show below. Again, the changes in the main program have been minimal; this is offered as an option in the final library subroutine.

In Table 9, we list the calculated missing boundary conditions of Roberts et al., and those obtained with SYSSOL modified as indicated above. We used $\Delta\epsilon = .1$. As usual, k is the correction number. N was 65.

The computer time on an IBM 360/50 was 135.28 sec.

The program SYSSOL given in the Appendix will perform continuation automatically as an option. The user has to embed his problem in a one parameter family of problems

$$y' = f(t, y; \epsilon),$$

such that, for $\epsilon = 0$, the problem is "simple", and, for $\epsilon = 1$, the original problem is recovered. This option is considered automatically whenever the parameter DELEPS $\epsilon(0, 1)$. It is the responsibility of the user to have the appropriate subroutine for calculating $f(t, y; \epsilon)$ and its Jacobian. In this case, initial values for Y must also be given.

TABLE 9

Method	$y_3(0)$	$y_5(0)$	$y_1(3.5)$	$y_3(3.5)$	$y_5(3.5)$
[23]	-0.97819707	0.64678660	-1.5308940	1.1744953	-0.31437074
SYSSOL					
k=0	-0.97793385	0.64706375	-1.5300011	1.1731673	-0.31483749
k=1	-0.97819829	0.64678677	-1.5308960	1.1745015	-0.31437128
k=2	-0.97819758	0.64678682	-1.5308941	1.1744980	-0.31437042
k=3	-0.97819757999	0.64678682479	-1.5308940685	1.1744981003	-0.31437042497
k=4	-0.97819757997	0.64678682478	-1.5308940684	1.1744981012	-0.31437042438

We point out that our asymptotic error estimate indicates that the max. abs. error on all the components (for $k = 4$) is equal to 6.67×10^{-11}, tending to confirm that all the figures shown in the last line of Table 9 are exact.

8. Generalization of the Milne-Numerov Method to Even-Order Systems of Special Type. In this section we shall consider systems of the form:

$$y^{(2r)} = f(t, y, y^{(2)}, \cdots, y^{(2r-2)}) \tag{8.1}$$

with separable two-point boundary conditions. Such systems can always be reduced to larger second order systems with no first derivatives present. Thus, without loss of generality, we consider instead:

$$y'' - f(t, y) = 0 \tag{8.2}$$

with the boundary conditions

$$y(a) = \alpha, \quad y(b) = \beta, \quad \text{and} \quad y(t) = (y_1(t), \cdots, y_n(t)).$$

Taking a uniform mesh with step h, we discretize these systems with the three-point Milne-Numerov formula (cf. [4], [12], [17] for $n = 1$).

$$N_h u_j = \frac{1}{h^2}(u_{j-1} - 2u_j + u_{j+1}) - \frac{1}{12}(f_{j-1} + 10 f_j + f_{j+1}) = 0, \qquad j = 1, \cdots, J-1. \tag{8.3}$$

For smooth f, the whole one-dimensional theory can be generalized to this n-dimensional case. The linear systems appearing in the application of Newton's method to (8.3) are block-tridiagonal and various techniques can be used in their solution. Observe that the resulting method is $O(h^4)$ and, therefore, deferred corrections will provide methods of order $O(h^{4k})$. Also, the fact that we deal with second order systems means that only half the number of equations are used that would be necessary upon reduction to first order systems. This type of problem appears for instance in the two-body equations of motion (cf. [22]), and in systems arising from the Schrödinger equation [1].

Acknowledgement. The accurate typing of this paper was produced in minimal time by Miss Brunilda Cerceau to whom we are very thankful.

Departamento de Computacion
Universidad Central de Venezuela
Apartado 59002
Caracas, Venezuela

Appendix. In the microfiche section of this journal, we present a FORTRAN (level G) implementation of Subroutine SYSSOL, for solving nonlinear two-point boundary value problems for first order systems of the form:

$$y' - f(t, y) = 0, \qquad a < t < b,$$

$$Ay(a) + By(b) = \alpha,$$

where $y(t) = (y_1(t), \cdots, y_m(t))^T, \alpha \in \mathbf{R}^m$ given, and A, B are given $m \times m$ matrices.

The program contains its own documentation and, we hope, is fairly readable. We have added the driver program and all the necessary subroutines to produce the results presented in Table 6.

The subroutine itself contains no input-output instructions and, therefore, it should be fairly transportable. There is only one instruction (the definition of EPSMAC), clearly marked, which is machine dependent. In [19, p. 74] can be found a flow-chart which is sufficiently close to give additional information on the functioning of SYSSOL.

The authors assume no responsibility for any damages that this subroutine may cause, but they will be happy to answer any comments or complaints.

1. A. C. ALLISON, "The numerical solution of coupled differential equations arising from the Schrödinger equation," *J. Computational Phys.*, v. 6, 1970, pp. 378–391. MR 43 #1438.

2. P. G. CIARLET, M. H. SCHULTZ & R. S. VARGA, "Numerical methods of high-order accuracy for nonlinear boundary value problems. I. One dimensional problem," *Numer. Math.*, v. 9, 1967, pp. 394–430. MR 36 #4813.

3. J. C. FALKENBERG, "A method for integration of unstable systems of ordinary differential equations subject to two-point boundary conditions," *Nordisk Tidskr. Informations-behandling* (*BIT*), v. 8, 1968, pp. 86–103. MR 39 #1123.

4. P. HENRICI, *Discrete Variable Methods in Ordinary Differential Equations*, Wiley, New York, 1962. MR 24 #B1772.

5. R. J. HERBOLD, M. H. SCHULTZ & R. S. VARGA, "The effect of quadrature errors in the numerical solution of boundary value problems by variational techniques," *Aequationes Math.*, v. 3, 1969, pp. 247–270. MR **41** #6410.

6. J. F. HOLT, "Numerical solution of nonlinear two-point boundary value problems by finite difference methods," *Comm. ACM,* v. 7, 1964, pp. 366–373. MR **29** #5387.

7. J. W. JEROME & R. S. VARGA, "Generalizations of spline functions and applications to nonlinear boundary value and eigenvalue problems," *Theory and Applications of Spline Functions* (edited by T. N. E. Greville), (Proc. Sem. Math. Research Center, Univ. of Wisconsin, Madison, Wis., 1968), Academic Press, New York, 1969, pp. 103–155. MR 39 #685,

8. H. B. KELLER, *Numerical Methods for Two-Point Boundary-Value Problems,* Blaisdell, Waltham, Mass., 1968. MR **37** #6038.

9. H. B. KELLER, "Accurate difference methods for linear ordinary differential systems subject to linear constraints," *SIAM J. Numer. Anal.,* v. 6, 1969, pp. 8–30. MR **40** #6776.

10. H. B. KELLER, "A new difference scheme for parabolic problems," *Numerical Solution of Partial Differential Equations,* II (edited by B. Hubbard) (SYNSPADE, 1970) (Proc. Sympos., Univ. of Maryland, College Park, Md., 1970), Academic Press, New York, 1971, pp. 327–350. MR **43** #2866.

11. H. B. KELLER, "Accurate difference methods for nonlinear two-point boundary value problems," (manuscript, 1972).

12. M. LEES, "Discrete methods for nonlinear two-point boundary value problems," *Numerical Solution of Partial Differential Equations* (edited by J. H. Bramble) (Proc. Sympos. Univ. Maryland, 1965), Academic Press, New York, 1966, pp. 59–72. MR **34** #2196.

13. M. LENTINI, *Correcciones diferidas para problemas de contorno en sistemas de ecuaciones diferenciales ordinarias de primer orden,* Pub. 73–04, Depto. de Comp., Fac. Ciencias, Univ. Central de Venezuela, Caracas, 1973.

14. J. M. ORTEGA & W. C. RHEINBOLDT, *Iterative Solution of Nonlinear Equations in Several Variables,* Academic Press, New York, 1970. MR **42** #8686.

15. M. R. OSBORNE, "On shooting methods for boundary value problems," *J. Math. Anal. Appl.,* v. 27, 1969, pp. 417–433. MR **39** #6521.

16. V. PEREYRA, "Iterated deferred corrections for nonlinear operator equations," *Numer. Math.,* v. 10, 1967, pp. 316–323. MR **36** #4812.

17. V. PEREYRA, "Iterated deferred corrections for nonlinear boundary value problems," *Numer. Math.,* v. 11, 1968, pp. 111–125. MR 37#1091.

18. V. PEREYRA, "Highly accurate numerical solution of casilinear elliptic boundary value problems in n dimensions," *Math. Comp.,* v. 24, 1970, pp. 771–783. MR **44** #6165.

19. V. PEREYRA, *High Order Finite Difference Solution of Differential Equations,* Stanford Univ. Comp. Sci. Report STAN–CS–73–348, 1973.

20. V. PEREYRA, "Variable order variable step finite difference methods for nonlinear boundary value problems," *Proceedings Conference on the Numerical Solution of Differential Equations,* Dundee, Scotland, Springer-Verlag, Berlin, 1973.

21. F. M. PERRIN, H. S. PRICE & R. S. VARGA, "On higher-order numerical methods for nonlinear two-point boundary value problems," *Numer. Math.,* v. 13, 1969, pp. 180–198. MR **40** #8276.

22. S. M. ROBERTS & J. S. SHIPMAN, "The Kantorovich theorem and two-point boundary value problems," *IBM J. Res. Develop.*, v. 10, 1966, pp. 402–406. MR **34** #2198.

23. S. M. ROBERTS, J. S. SHIPMAN & W. J. ELLIS, "A perturbation technique for nonlinear two-point boundary value problems," *SIAM J. Numer. Anal.*, v. 6, 1969, pp. 347–358. MR 40 #8277.

24. J. STOER & R. BULIRSCH, *Einführung in die Numerische Mathematik* II, Springer-Verlag, Berlin, 1973.

25. J. VARAH, *On the Solution of Block Tridiagonal Systems Arising from Certain Finite-Difference Equations*, Univ. British Columbia, Dept. Comp. Sci. Technical Report 72–02, 1972.

FORTRAN Implementation of Subroutine SYSSOL, For Solving Nonlinear Two-Point Boundary Value Problems For First Order Systems

by

M. LENTINI & V. PEREYRA

```
      SUBROUTINE SYSSOL(M,N,A,B,ALPHA,A1,B1,TOL,DELEPS,X,Y,
     *                      ABT,FF,JACOB,JERROR)
C                                                                      *
C     PURPOSE :  THIS IS A VARIABLE ORDER, VARIABLE                    *
C  (UNIFORM) STEP SOLVER FOR TWO-POINT BOUNDARY VALUE FIRST*
C  ORDER SMOOTH NONLINEAR SYSTEMS OF THE FORM                          *
C                                                                      *
C   (1)    Y' = F(X,Y)    ,   A1 . Y(A) + B1 . Y(B) = ALPHA *
C                                                                      *
C  IT ATTEMPTS TO PRODUCE A DISCRETE SOLUTION WITH MAXIMUM *
C  ABSOLUTE ERROR LESS THAN  TOL  ON A UNIFORM GRID CONTAI-*
C  NING THE ONE DEFINED BY THE USER WITH THE PARAMETER  N. *
C  THE METHOD, MORE DETAILS AND TESTS ARE DESCRIBED IN       *
C  'A VARIABLE ORDER, VARIABLE STEP, FINITE DIFFERENCE      *
C  METHOD FOR NONLINEAR,MULTIPOINT BOUNDARY VALUE PROBLEMS'*
C  BY M. LENTINI AND V. PEREYRA, PUB. 73-06, DEPTO. DE      *
C  COMPUTACION, FAC. DE CIENCIAS, UNIVERSIDAD CENTRAL DE    *
C  VENEZUELA (1973).                                                   *
C  ENQUIRIES AND COMMENTS CAN BE ADDRESSED TO                          *
C                                                                      *
C  M. LENTINI  AND  V. PEREYRA                                         *
C  DEPARTAMENTO DE COMPUTACION                                         *
C  APARTADO 59002 - CARACAS                                            *
C                                                                      *
C  MADE IN VENEZUELA                   AUGUST 1973                     *
C                                                                      *
C***********************************************************************
      CONTINUE
C***********************************************************************
C                                                                      *
C   USE :  PRESENT DIMENSIONING OF ARRAYS ALLOWS PROCESSING*
C          OF SYSTEMS OF UP TO                                         *
C                              MMAX = 10 EQUATIONS                     *
C          ON GRIDS WITH UP TO                                         *
C                              NMAX = 650/M POINTS.                    *
C          THESE SIZES CAN BE VARIED BY CHANGING THE                   *
C          DIMENSIONS APPROPRIATELY.                                   *
C                                                                      *
C        THE USER MUST SPECIFY EACH OF THE FOLLOWING:                  *
C                                                                      *
C    M  -  NUMBER OF EQUATIONS                                         *
C    N  -  NUMBER OF POINTS IN THE BASIC GRID (COUNTING THE*
C          END POINTS). N  MUST BE  > 3.                               *
C          UPON EXIT, N  WILL CONTAIN THE FINAL GRID SIZE  *
C          IN WHICH  Y  HAS BEEN COMPUTED.                             *
C    A  -  LEFT BOUNDARY POINT                                         *
C    B  -  RIGHT BOUNDARY POINT                                        *
C    ALPHA ,A1 ,B1  -  ARRAYS OF DIMENSIONS  MMAX                      *
C                    CONTAINING BOUNDARY CONDITIONS.SEE(1) *
```

```
C     TOL   -  DESIRED FINAL ABSOLUTE ERROR.                      .  *
      CONTINUE
C   DELEPS - (A) IF DELEPS <= 0 THEN THE PROGRAM SETS Y=0 AS*
C                INITIAL VALUES.                                     *
C                IF DELEPS IS SET TO ANY VALUE >= 1 THEN THE*
C                PROGRAM ASSUMES THAT THE USER WILL GIVE             *
C                INITIAL VALUES FOR Y. THESE INITIAL VALUES *
C                (ON THE INITIAL MESH) MUST BE STORED AS             *
C                INDICATED BELOW (SEE OUTPUT PARAMETERS).
C            (B) A CONTINUATION METHOD CAN BE EMPLOYED AS AN*
C                OPTION IN SEVERELY NON LINEAR CASES IN              *
C                ORDER TO GENERATE GOOD INITIAL VALUES. THIS*
C                PRESUPOSES THAT THE USER HAS EMBEDDED HIS           *
C                PROBLEM IN A ONE PARAMETER FAMILY:                  *
C                Y' = F(X, Y, EPSNU),                                *
C                SUCH THAT FOR EPSNU =0 THE RESULTING                *
C                PROBLEM IS ''SIMPLER'', AND FOR EPSNU = 1           *
C                THE ORIGINAL PROBLEM IS RECOVERED. THE              *
C                PROGRAM WILL AUTOMATICALLY ATTEMPT TO GO            *
C                FROM EPSNU = 0 TO EPSNU = 1 IN STEPS OF             *
C                DELEPS. THUS IN THIS CASE
C                0 < DELEPS < 1                                      *
C                ONCE EPSNU > = 1 THE REGULAR PROCEDURE              *
C                CONTINUES. THE USER MUST GIVE INITIAL               *
C                VALUES FOR Y AS IN (A).                             *
      CONTINUE
C                WHEN USING THIS OPTION THE USER MUST                *
C                INCLUDE IN HIS SUBROUTINES FF, JACOB                *
C                THE COMMON CARDS                                    *
C                COMMON /C1/ EPSNU
C                IN ORDER TO DESCRIBE THE PARAMETRIZED               *
C                PROBLEM                                             *
C                                                                    *
C*********************************************************************
      CONTINUE
C*********************************************************************
C                                                                    *
C        *** USER PROVIDED SUBROUTINES ***                           *
C                                                                    *
C                                                                    *
C     FF   -  NAME OF SUBROUTINE FOR EVALUATING  F(X,Y)  ON          *
C             THE WHOLE MESH.  THE SUBROUTINE ITSELF MUST BE *
C             PROVIDED BY THE USER AND IT SHOULD HAVE THE            *
C             FOLLOWING HEADING                                      *
C                                                                    *
C                    SUBROUTINE FF(X,Y,N,F)                          *
C                                                                    *
C             WHERE  F  IS THE VECTOR CONTAINING THE VALUES  *
C             OF  F(X,Y). STORAGE OF  F  IS THE SAME AS THAT *
C             OF  Y  (SEE BELOW)                                     *
      CONTINUE
C     JACOB  -  NAME OF A SUBROUTINE FOR EVALUATING THE              *
C               JACOBIAN OF  F(X,Y)  AT A GIVEN GRID POINT. *
C               THE SUBROUTINE ITSELF MUST BE PROVIDED BY            *
C               THE USER, AND IT SHOULD HAVE THE FOLLOWING           *
```

```
C              HEADING:                                         *
C                                                               *
C                                                               *
C                  SUBROUTINE JACOB(XX,YY,XJAC)                 *
C                                                               *
C              WHERE  XX  IS THE GIVEN GRID POINT, YY  IS       *
C              THE CURRENT VALUE OF  Y  AT  XX, AND THE         *
C              MMAX*MMAX  ARRAY  XJAC  SHOULD BE FILLED BY      *
C              THE USER WITH THE CORRESPONDING PARTIAL          *
C              DERIVATIVES ACCORDING TO                         *
C                   XJAC(I,J) = DFI/DYJ                         *
C                                                               *
C****************************************************************
      CONTINUE
C****************************************************************
C                                                               *
C              *** OUTPUT  PARAMETERS ***                       *
C                                                               *
C    SEE  N  ABOVE                                              *
C                                                               *
C    Y  -  NMAX*MMAX-VECTOR CONTAINING THE COMPUTED             *
C          SOLUTION ON THE FINAL GRID                           *
C          THE GRID VECTOR FUNCTION  Y(X)  IS STORED            *
C          SEQUENTIALLY IN THE FORM:                            *
C             Y1(X1),Y2(X1),....,YM(X1),Y1(X2),....,YM(XN),     *
C          THE PROGRAM SETS  Y  TO ZERO INITIALLY WHEN          *
C          DELEPS <= 0.                                         *
C    X  -  NMAX-VECTOR CONTAINING THE FINAL GRID POINTS.        *
C    ABT - M-VECTOR CONTAINING MAX ABSOLUTE ERROR ON THE        *
C          GRID FOR EACH COMPONENT OF THE APPROXIMATE           *
C          SOLUTION                                             *
      CONTINUE
C    JERROR  -  ERROR CODED INTEGER VARIABLE.                   *
C      JERROR = 0   REQUIRED TOLERANCE WAS REACHED              *
C      JERROR = 1   M  OR INITIAL  N  ARE OUT OF RANGE.         *
C             CORRECTING ACTION:  CHECK THAT INPUT DATA         *
C             SATISFIES  0 < M <= MMAX  AND  3 < N <= NMAX      *
C             AND RERUN.                                        *
C      JERROR = 2   THE PROGRAM HAS ATTEMPTED TO USE A GRID*
C           WITH MORE THAN  NMAX  POINTS                        *
C           CORRECTING ACTION:  IF PARTIAL RESULTS SEEM         *
C           REASONABLE, INCREASE  MMAX*NMAX = 650 AND           *
C           CHANGE DIMENSIONS IN ALL SUBROUTINES ACCORDING.*
C           IN ORDER TO ALLOW FINER GRIDS                       *
C****************************************************************
      CONTINUE
C****************************************************************
C                                                               *
C   *** OTHER SUBROUTINES NEEDED BY SYSSOL ***                  *
C                                                               *
C  SYSLIN - SOLVES A BLOCK LINEAR SYSTEM OF SPECIAL TYPE.       *
C  DGELG  - SOLVES A LINEAR SYSTEM OF EQUATIONS                 *
C  DARRAY - TRANSFORMS BETWEEN TWO AND ONE DIMENSIONAL          *
C           STORAGE.                                            *
C  U2DCGS - DEFERRED CORRECTION GENERATOR AND INTERPOLATION*
```

```
C  COEGEN - WEIGHT GENERATOR                                 *
C                                                            *
C************************************************************
C
      IMPLICIT REAL*8(A-H,O-Z)
      COMMON /C1/ EPSNU
      LOGICAL DIVNEW
C************************************************************
C                                                            *
C  IN CALLING PROGRAM THE PARAMETER ARRAY X MUST BE          *
C  DIMENSIONED AS X( 650/M )                                 *
C                                                            *
      DIMENSION X(1)
C                                                            *
C  DIMENSIONS INVOLVED IN THE FOLLOWING PARAMETER ARRAYS     *
C  ARE MMAX = 10 AND NMAX*MMAX = 650, AND THEY MUST BE       *
C   DIMENSIONED ACCORDINGLY IN THE CALLING PROGRAM           *
C                                                            *
      DIMENSION ALPHA(10),A1(10,10),B1(10,10),Y(650),ABT(10)
C                                                            *
C  FOLLOWING ARRAYS ARE WORKING AREAS.                       *
      DIMENSION F(650),UU(650),RES(650),SK(650),
C                                                            *
     *ABSEX(10),ABR(10),TEMP(10)
C                                                            *
C  WORKING AREAS WITH SIZES RELATED TO MAX. NUMBER OF        *
C  DEFERRED CORRECTIONS = 20 , WHICH SHOULD BE ADEQUATED     *
C     FOR ALL PURPOSES.                                      *
C                                                            *
      DIMENSION AA(50),BB(50),CC(20)
C************************************************************
C                                                            *
C
      MMAX=10
      NMAX=650/M
      IF((M.LE.0).OR.(M.GT.MMAX).OR.(N.LE.3).OR.(N.GT.NMAX))GO TO 1
      GO TO 10
1     JERROR=1
      RETURN
C
C<<<....             ERROR  EXIT  1 .     .........
C
C     INITIALIZATION
C
10    MPN=M*N
C
C************************************************************
C     THIS CONSTANT IS MACHINE DEPENDENT:                    *
C     EPSMAC  IS APPROXIMATELY 10* RELATIVE PRECISION OF     *
C     FLOATING POINT ARITHMETIC                              *
      EPSMAC=5.D-15
C************************************************************
C
      EPSNU=0.D0
      JERROR=0
      M1=M+1
      M2=M*M
      IF (DELEPS.GT.0.D0)GO TO 60
C
C     FIRST  APPROXIMATION  FOR  Y
C
40    DO 50 I=1,MPN
50    Y(I)=0.D0
C
60    BB(1)=1.D0
      DO 105 I=2,50
105   BB(I)=0.D0
C***********************************************************
C                                                          *
C  IN CORRECTION  NU=0 THE RESIDUAL IN THE NEWTON ITERATION*
C  MUST BE REDUCED IN NORM BELOW  EPS=C*H**4 , WHERE  C  IS*
C  A SMALL CONSTANT. FOR  NU > 0, WE USE THE ERROR ESTIMATE*
C  CORRESPONDING TO CORRECTION (NU-1) : ERROLD, IN ORDER TO*
C  OBTAIN THE NEW  EPS= C*H**2*ERROLD.  IN ALL CASES  EPS  *
C  IS NOT ALLOWED TO BE SMALLER THAN  EPSMAC.              *
C                                                          *
C***********************************************************
      EPS=DMAX1(EPSMAC,.01D0*((B-A)/(N-1))**4)
      NU=0
C
C....>>>       ENTER  UPON  STEP  HALVING        .......
C
 120  ERROLD=1.0D20
      KMAX=(N-2)/2
      MPNM=MPN-M
      C1=0.8D0
      N1=N-1
      H=(B-A)/N1
      HCUA=H**2
      DO 150 I=1,MPNM
150   SK(I)=0.D0
      X(1)=A
      X(N)=B
      DO 200 I=2,N1
          I1=I-1
200       X(I)=A+I1*H
      IF(NU .EQ. 0)GO TO 405
C
C     WHEN STEP-HALVING WE HAVE TO INITIALIZE  SK  IF  NU .GT.
C
      CALL FF(X,Y,N,F)
      CALL U2DCGS(NU,2,2,N1,M,AA,F,RES,IERROR)
      DO 300 I=1,MPNM
300   SK(I)=H*RES(I)
C
C....>>>             NEWTON ITERATION STARTS        .......
C
 405  INWT=0
      REOLD=1.0D20
      DIVNEW=.FALSE.
      IF(NU+1.LE.KMAX)GO TO 410
C
C     MAXIMUM NUMBER OF CORRECTIONS ON THIS MESH HAS BEEN
C     REACHED.  GO TO  'STEP HALVING'
C
      NU=NU+1
      GO TO 2600
C.....>>>      LABEL 410 IS INPUT FOR NEWTON ITERATION      .......
C
C
C     RESIDUAL  COMPUTATION
C
410   RABS=0.D0
      DO 700 I=1,M
         SUM=ALPHA(I)
         DO 600 J=1,M
 600     SUM=SUM-A1(I,J)*Y(J)-B1(I,J)*Y(MPNM+J)
         RES(I)=SUM
         TEM=DABS(RES(I))
         IF(TEM.GT.RABS)RABS=TEM
700   CONTINUE
800   CALL FF (X,Y,N,F)
      DO 900 I=2,N
         K1=(I-1)*M
         DO 900 J=1,M
         K1J=K1+J
         K1JM=K1J-M
         RES(K1J)=-Y(K1J)+Y(K1JM)+H/2*(F(K1J)+F(K1JM))+SK(K1JM)
         TEM=DABS(RES(K1J))
         IF(TEM.GT.RABS)RABS=TEM
900   CONTINUE
C
C     THE FIRST TIME THROUGH WE DON'T CHECK ANYTHING
C
905   IF(INWT .EQ. 0) GO TO 950
910   IF(RABS.LT.REOLD.OR.INWT.EQ.1)GO TO 920
C
C     IF THE RESIDUAL INCREASES AFTER THE FIRST ITERATION
C     WE ASSUME DIVERGENCE AND GO TO HALVE THE STEP SIZE
C
      DIVNEW=.TRUE.
      NU=NU+1
      GO TO 2600
920   IF(RABS.LE.EPS.OR.INWT.GE.5)GO TO 1500
C
C.....         NEWTON EXIT (CONVERGENCE OR TOO MANY ITERATIONS)
C
950   CALL SYSLIN(M,N,X,Y,H,JACOB,RES,A1,B1,UU)
      REOLD=RABS
C
C     APPROXIMATE SOLUTION IS CORRECTED
C
1100  DO 1300 I=1,MPN
1300  Y(I)=Y(I)+UU(I)
C
```

```
C      NEXT TWO INSTRUCTION ARE FOR CONTROL OF PARAMETER
C      IN CONTINUATION METHOD
C
       EPSNU=DMIN1(EPSNU+DELEPS,1.D0)
       IF(EPSNU.GE.1.D0.OR.DELEPS.LE.0.D0)INWT=INWT+1
       GO TO 410
C
C      CORRECTION AND ERROR CONTROL STARTS
C
1500   NU=NU+1
       NU2=2*NU+1
       AA(NU2)=-DFLOAT(NU)/DFLOAT((2**(2*NU-1)*NU2))
       AA(NU2+1)=0.D0
       CALL U2DCGS(NU,2,2,N1,M,AA,F,RES,IERROR)
       IF(IERROR .EQ. 1) GO TO 2600
       DO 1700 I=1,MPNM
          AUXI=RES(I)*H
          RES(I)=SK(I)-AUXI
          SK(I)=AUXI
1700   CONTINUE
       CALL SYSLIN(M,N,X,Y,H,JACOB,RES,A1,B1,UU)
C
C      ESTIMATE FOR MAX. ABSOLUTE ERROR (BY COMPONENTS)
C
       ERRNEW=0.D0
       DO 1900 J=1,M
1900   ABT(J)=0.D0
       DO 2100 I=1,N
       DO 2100 J=1,M
          KI=(I-1)*M+J
          U1=DABS(UU(KI))
          IF(U1 .GT. ABT(J)) ABT(J)=U1
2100   CONTINUE
       DO 2300 J=1,M
       IF(ABT(J) .GT. ERRNEW) ERRNEW=ABT(J)
2300   CONTINUE
       K=NU-1
2500   IF(ERRNEW .LE. TOL)RETURN
C
C<<<....            PRECISION  ACHIEVED       ............
C
C      CC(K+1) CONTAINS ESTIMATED ERROR FOR CORRECTION  K
C      ON MESH SIZE H/2 (UNDER THE HYPOTHESIS : ERROR IN
C           CORRECTION  K(H) = C*H**(2*K+2)).
C
       CC(NU)=ERRNEW*4.D0**(-NU)
       IF(ERRNEW .LE. .1*ERROLD) GO TO 2550
       IF(ERRNEW .GT. C1*ERROLD) GO TO 2600
       C1=.5D0*C1
C
C      EITHER KEEP CORRECTING ...
C
C*************************************************************
C                                                            *
C  THE ERROR REDUCTION THRESHOLD C1 IS SET ORIGINALLY (AND *
C  ARBITRARILY) TO 0.8. IF  C1*ERROLD < ERRNEW  WE HALVE    *
C  THE STEP. EACH TIME THAT  0.1*ERROLD < ERRNEW <C1*ERROLD*
C  WE SET C1 TO 0.5*C1, THUS ACTUALLY ALLOWING THIS TO HA- *
C  PPEN A MAXIMUM OF THREE TIMES, BEFORE THE MORE STRICT    *
C  TEST WITH 0.1*ERROLD TAKES OVER COMPLETELY.              *
C  ERROLD IS THE ERROR ESTIMATE FOR THE LAST CORRECTION BUT *
C  ONE, WHILE ERRNEW IS THE ONE CORRESPONDING TO THE LAST   *
C  CORRECTION.                                              *
C                                                           *
C************************************************************
2550  ERROLD=ERRNEW
      EPS=DMAX1(EPSMAC,.01D0*H**2*ERROLD)
      GO TO 405
C
C....>>>       OR  HALVE  THE  STEP SIZE        ......
C
2600  IF(2*N-1.LE.NMAX)GO TO 2625
      JERROR=2
      RETURN
C
C<<<....              TOO MANY GRID POINTS         ......
C
2625  N=2*N-1
      MPN=M*N
C
C      IF NEWTON DIVERGED WE START AGAIN WITH NU=0
C
      IF(DIVNEW)GO TO 40
C************************************************************
C                                                           *
C  NOW WE DECIDE THE LEVEL OF CORRECTION ON  THE NEW GRID   *
C  WE ASSUME THAT THE LAST ESTIMATED ERROR (PRESENTLY  IN   *
C  ERROLD) WILL BE PRESERVED AFTER INTERPOLATING, AND       *
C  THEREFORE WE LOCATE THE FIRST INDEX I FOR WHICH          *
C  CC(I)<=ERROLD, WHERE  CC(I) IS THE PREDICTED ERROR FOR   *
C  THE (I-1) CORRECTION ON THE NEW GRID.                    *
C                                                           *
C************************************************************
      NUINT=NU
      IF(ERRNEW.GE.ERROLD)GO TO 2650
      ERROLD=ERRNEW
2650  DO 2700 I=1,NU
      IF(ERROLD .LT. CC(I)) GO TO 2700
      GO TO 2750
 2700 CONTINUE
 2750 NU=I-1
      EPS=DMAX1(EPSMAC,.005D0*H**2*ERROLD)
C************************************************************
C                                                           *
C  COMPUTATION OF FIRST APPROXIMATION FOR  Y  ON NEW GRID   *
C  BY MEANS OF U2DCGS WILL GIVE ORDER OF INTERPOLATION      *
C  (2*NU+2), WHERE  NU  IS THE LAST SUCCESFULL CORRECTION   *
C  PERFORMED ON THE COARSER GRID.                           *
C                                                           *
C************************************************************
```

```
2800  NO2=(N+1)/2
      MPO2=M*NO2
      DO 2900 I=1,MPO2
2900  RES(I)=Y(I)
      NO21=NO2-1
      CALL U2DCGS(NUINT,2,0,NO21,M,BB,Y,SK,IERROR)
      DO 3100 I=1,NO21
      KI=(I-1)*M
      KI2=2*KI
      DO 3100 L=1,M
      Y(KI2+L)=RES(KI+L)
3100  Y(KI2+M+L)=SK(KI+L)
      DO 3200 L=1,M
3200  Y(MPN-M+L)=RES(MPO2-M+L)
      GO TO 120
C.....          START  ON  NEW  GRID          ......
      END


C-----------------------------------------------------------------


      SUBROUTINE SYSLIN(M,N,X,Y,H,JACOB,RES,A1,B1,UU)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION X(1),Y(1),UU(1),RES(1)
C*****************************************************************
C                                                                *
C              *** SOLUTION OF LINEAR SYSTEM ***                 *
C                                                                *
C*****************************************************************
C                                                                *
C  FOLLOWING ARRAYS ARE WORKING AREAS. DIMENSIONS INVOLVED       *
C  ARE :  MMAX=10 , MMAX+1 = 11 , MMAX**2=100 ,                  *
C         NMAX*MMAX=650, MMAX*(MMAX+1)=110                       *
C                                                                *
      DIMENSION T(110),U(10),S(10,10),VM(10,11),R(100),
     *          V(650,11),AUX(10,11),A1(10,10),B1(10,10)
      DOUBLE PRECISION JAB(10,10)
C                                                                *
C*****************************************************************
C
C*****************************************************************
C                                                                *
C  IT SOLVES THE 2*2 BLOCK SYSTEM                                *
C                   | A | B |  | XO |    | BO |                  *
C                   |---|---|  |----| =  |----|                  *
C                   | C | D |  | X  |    | B1 |                  *
C  WHERE A IS M*M AND D IS (M*N)*(M*N) AND ALL THE OTHER         *
C  BLOCKS HAVE THE APPROPIATE DIMENSIONS. D IS BLOCK LOWER       *
C  BIDIAGONAL, WITH MAIN DIAGONAL BLOCKS R(I) AND                *
C  SUB-DIAGONAL S(I), ALL OF SIZES M*M.                          *
C  C HAS ONLY THE FIRST BLOCK DIFFERENT FROM ZERO, AND B         *
C  ONLY THE LAST BLOCK DIFFERENT FROM ZERO.                      *
```

```
C                                                          *
C***********************************************************
      CONTINUE
C***********************************************************
C                                                          *
C               *** OUTLINE OF THE METHOD ***              *
C                                                          *
C  FIRST WE FORM C'=(C|B1) AND THEN WE SOLVE THE MATRIX    *
C  SYSTEM  DV'=C'  (V'=(V|W))                              *
C  BY THE RECURSION: V'(0)=0,                              *
C  R(J)V'(J)=(C'(J)-S(J)V'(J-1)), J=1,...,N                *
C  THESE LINEAR SYSTEMS ARE SOLVED BY A STANDARD GAUSSIAN  *
C  ELIMINATION CODE (SUBROUTINE DGELG).                    *
C  FINALLY XO IS THE SOLUTION OF THE LINEAR SYSTEM         *
C  (A - B V) XO = BO - B W  AND                            *
C  X = W - V XO                                            *
C                                                          *
C***********************************************************
      CONTINUE
C***********************************************************
C                                                          *
C                       *** CAUTION ***                    *
C                                                          *
C  THIS SUBROUTINE MANIPULATES SOME MATRICES AS ONE        *
C  DIMENSIONAL ARRAYS.                                     *
C  SUBROUTINES DGELG AND DARRAY ARE FROM THE IBM/360       *
C  SCIENTIFIC SUBROUTINE PACKAGE.                          *
C  THE SUBROUTINE DARRAY TRANSFORMS BETWEEN TYPES OF       *
C  STORAGE.                                                *
C                                                          *
C***********************************************************
C
950   CALL JACOB(X(1),Y,JAB)
      M2=M*M
      MPN=M*N
      M1=M+1
C                                                          *
C               *** SOLUTION OF D.V' = C' ***              *
C                                                          *
      DO 1000 I=1,M
 1000 T(M2+I)=RES(M+I)
      DO 1800 L=2,N
      K1=(L-1)*M
      DO 1100 J=1,M
 1100 U(J)=Y(K1+J)
C
C         *** GENERATION OF JACOBIAN ***
C
      H2=.5D0*H
      DO 50 J=1,M
      K1=(J-1)*M
      DO 50 I=1,M
      S(I,J)=H2*JAB(I,J)
      IF(I .EQ. J) S(I,J)=S(I,J)+1.D0
   50 CONTINUE
```

```
      CALL JACOB(X(L),U,JAB)
      DO 60 I=1,M
      DO 60 J=1,M
         KI=(J-1)*M+I
         R(KI)=-H2*JAB(I,J)
         IF(I.EQ.J)R(KI)=R(KI)+1.DO
60    CONTINUE
      IF(L.NE.2)GO TO 1300
      DO 1200 J=1,M
      K1=(J-1)*M
      DO 1200 I=1,M
 1200 T(K1+I)=-S(I,J)
      GO TO 1700
C                                                                       *
C                 COMPUTATION OF (C' - S.V')                            *
C                                                                       *
1300  DO 1500 K1=1,M1
      DO 1500 I=1,M
      SUM=0.DO
      DO 1400 J=1,M
 1400 SUM=SUM+S(I,J)*VM(J,K1)
 1500 T((K1-1)*M+I)=SUM
      DO 1600 I=1,M
 1600 T(M2+I)=T(M2+I)+RES((L-1)*M+I)
1700  CALL DGELG (T,R,M,M1,1.D-7,IER)
 1750 CALL DARRAY (1,M,M1,10,11,T,VM)
      DO 1800 J=1,M1
      DO 1800 I=1,M
 1800 V((L-1)*M+I,J)=VM(I,J)
C                                                                       *
C                 *** END OF RECURSION ***                              *
C .                                                                     *
      DO 2000 J=1,M1
      DO 2000 I=1,M
      SUM=0.DO
C                                                                       *
C     PRODUCTS B.V AND B.W
C                                                                       *
      DO 1900 K=1,M
 1900 SUM=SUM+B1(I,K)*VM(K,J)
 2000 AUX(I,J)=SUM
C                                                                       *
C     (A - B.V)
C                                                                       *
      DO 2100 J=1,M
      K1=(J-1)*M
      DO 2100 I=1,M
 2100 R(K1+I)=A1(I,J)-AUX(I,J)
      DO 2200 I=1,M
 2200 UU(I)=RES(I)-AUX(I,M1)
C                                                                       *
C     SOLUTION OF LINEAR SYSTEM
C     (A - B.V) XO = (BO - B.W)
C     AND COMPUTATION OF X
C                                                                       *
```

```
      DO 1 I=1,N
 1    C(I)=BB(I)
      DO 11 I=1,N
 11   ALF(I)=I-NP-0.5D0
 2    NN=N-1
      DO 6 I=1,NN
      LL=N-I
      DO 6 J=1,LL
      K=N-J+1
  6   C(K)=C(K)-ALF(I)*C(K-1)
      DO 8 I=1,NN
      K=N-I
      XKIN=1.D0/K
      KM1=K+1
      DO 8 J=KM1,N
      C(J)=C(J)*XKIN
      JM1=J-1
  8   C(JM1)=C(JM1)-C(J)
      RETURN
      END


C---------------------------------------------------------------


C  DRIVER PROGRAM FOR PROBLEMS 1 TO 5
C  SEE REFERENCE IN COMMENTS TO SUBROUTINE SYSSOL
      IMPLICIT REAL*8(A-H,O-Z)
      EXTERNAL FF1,JACOB1
      EXTERNAL FF2,JACOB2
      EXTERNAL FF3,JACOB3
      EXTERNAL FF4,JACOB4
      EXTERNAL FF5,JACOB5
      COMMON /P5/ ALPHA1,BETA1,S1,C1
      DIMENSION A1(10,10),B1(10,10),ALPHA(10),Y(650),X(326),ABT(10)
      NO=3
      DO 20 IN=1,5
      NO=2*NO-1
      TOL=1.D0
      DO 20 KL=1,3
      TOL=TOL*1.D-3
      DO 20 IP=1,5
      N=NO
      WRITE(3,200)IP
      DELEPS=0.D0
      GO TO (1,2,3,4,5),IP
      M=2
      A=0.D0
      B=3.14159265358979300
      ALPHA(1)=0.D0
      ALPHA(2)=0.D0
      DO 1000 I=1,2
      DO 1000 J=1,2
      A1(I,J)=0.D0
1000  B1(I,J)=0.D0
      A1(1,1)=1.D0
      B1(2,1)=1.D0
      CALL SYSSOL(M,N,A,B,ALPHA,A1,B1,TOL,DELEPS,X,Y,ABT,FF1,JACOB1,
     *JERROR)
      GO TO 10
2     M=2
      A=0.D0
      B=1.D0
      ALPHA(1)=0.D0
      ALPHA(2)=20.D0*(1.D0-DEXP(-20.D0))/(1.D0+DEXP(-20.D0))
      DO 2000 I=1,2
      DO 2000 J=1,2
      A1(I,J)=0.D0
2000  B1(I,J)=0.D0
      A1(1,1)=1.D0
      B1(2,2)=1.D0
      CALL SYSSOL(M,N,A,B,ALPHA,A1,B1,TOL,DELEPS,X,Y,ABT,FF2,JACOB2,
     *JERROR)
      GO TO 10
3     M=2
      A=0.D0
      B=1.D0
      DO 3000 I=1,2
      ALPHA(I)=0.D0
      DO 3000 J=1,2
      A1(I,J)=0.D0
3000  B1(I,J)=0.D0
      A1(1,1)=1.D0
      B1(2,1)=1.D0
      CALL SYSSOL(M,N,A,B,ALPHA,A1,B1,TOL,DELEPS,X,Y,ABT,FF3,JACOB3,
     *JERROR)
      GO TO 10
4     M=4
      A=0.D0
      B=1.D0
      DO 4000 I=1,4
      ALPHA(I)=0.D0
      DO 4000 J=1,4
      A1(I,J)=0.D0
4000  B1(I,J)=0.D0
      A1(1,1)=1.D0
      A1(2,2)=1.D0
      B1(3,1)=1.D0
      B1(4,2)=1.D0
      CALL SYSSOL(M,N,A,B,ALPHA,A1,B1,TOL,DELEPS,X,Y,ABT,FF4,JACOB4,
     *JERROR)
      GO TO 10
5     M=4
      ALPHA1=2.5D0
      BETA1=ALPHA1
      S1=10.D0
      C1=0.1D-2
      A=0.D0
      B=10.D0
      DO 5000 I=1,4
      ALPHA(I)=0.D0
      DO 5000 J=1,4
      A1(I,J)=0.D0
5000  B1(I,J)=0.D0
      A1(1,1)=1.D0
      A1(2,4)=1.D0
      B1(3,2)=1.D0
      B1(4,4)=1.D0
      ALPHA(4)=C1
      CALL SYSSOL(M,N,A,B,ALPHA,A1,B1,TOL,DELEPS,X,Y,ABT,FF5,JACOB5,
     *JERROR)
 10   WRITE(3,16)M,N,A,B,(ALPHA(I),I=1,M)
      WRITE(3,18)((A1(I,J),J=1,M),I=1,M)
      WRITE(3,18)((B1(I,J),J=1,M),I=1,M)
      WRITE(3,19)TOL
      WRITE(3,13)(ABT(J),J=1,M)
      WRITE(3,15)
 20   CONTINUE
55555 STOP
  12  FORMAT(' *   ERROR ESTIMADO=',D12.3,' EN CORRECCION',I3,' *')
  13  FORMAT(' ERROR ESTIMADO POR COMPONENTES'/' ',10D12.3)
  15  FORMAT(1H0,'**************************************************'/)
16    FORMAT(1H0,                    ' NUMERO DE ECUACIONES',I2/' NUMERO DE
     * PUNTOS DE LA RED',I3/' ',
     *'EXTREMO IZQUIERDO DEL INTERVALO=',F10.6,2X,'EXTREMO DERECHO DEL I
     *NTERVALO=',F10.6//' CONDICION DE CONTORNO'/' ',5(F10.6,2X))
18    FORMAT(' MATRIZ DE CONDICION DE CONTORNO'/' ',5(F10.6,2X))
19    FORMAT(' TOLERANCIA',D12.2)
100   FORMAT(3I2,2F23.15/3F23.15)
200   FORMAT(1H1,' PROBLEMA',I4//)
300   FORMAT(1H0,'  CONDICIONES DE CONTORNO'/5D23.15)
 500  FORMAT(1H0,D10.2,6D20.12)
      END


C------------------------------------------------------------


      SUBROUTINE FF1(X,Y,N,FF)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION X(1),Y(1),FF(1)
      DO 10 I=1,N
      KI=(I-1)*2+1
      FF(KI)=Y(KI+1)
  10  FF(KI+1)=(1.D0-Y(KI)**2)*Y(KI+1)+4.D0*Y(KI)-5.D0*DSIN(X(I))-
     *(DCOS(X(I)))**3
      RETURN
      END
      SUBROUTINE JACOB1(X,YX,JAB)
      IMPLICIT REAL*8 (A-H,O-Z)
      DOUBLE PRECISION JAB(10,10)
      DIMENSION YX(1)
      JAB(1,1)=0.D0
      JAB(2,1)=-2.D0*YX(1)*YX(2)+4.D0
      JAB(1,2)=1.D0
      JAB(2,2)=1.D0-YX(1)**2
      RETURN
      END
      SUBROUTINE FF2(X,Y,N,FF)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION X(1),Y(1),FF(1)
      PI=3.14159265358979 3D0
      DPI2=2.D0*PI**2
      DPI=2.D0*PI
      DO 10 I=1,N
      KI=(I-1)*2+1
      FF(KI)=Y(KI+1)
 10   FF(KI+1)=400.D0*(Y(KI)+DCOS(PI*X(I))**2)+DPI2*DCOS(DPI*X(I))
      RETURN
      END
      SUBROUTINE JACOB2(X,YX,JAB)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION YX(1)
      DOUBLE PRECISION JAB(10,10)
      JAB(1,1)=0.D0
      JAB(2,1)=400.D0
      JAB(1,2)=1.D0
      JAB(2,2)=0.D0
      RETURN
      END
      SUBROUTINE FF3(X,Y,N,FF)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION X(1),Y(1),FF(1)
      DO 10 I=1,N
      KI=(I-1)*2+1
      FF(KI)=Y(KI+1)
10    FF(KI+1)=DEXP(Y(KI))
      RETURN
      END
      SUBROUTINE JACOB3(X,YX,JAB)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION YX(1)
      DOUBLE PRECISION JAB(10,10)
      JAB(1,1)=0.D0
      JAB(2,1)=DEXP(YX(1))
      JAB(1,2)=1.D0
      JAB(2,2)=0.D0
      RETURN
      END
      SUBROUTINE FF4(X,Y,N,FF)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION X(1),Y(1),FF(1)
      DO 10 I=1,N
      KI=(I-1)*4+1
      FF(KI)=Y(KI+1)
      FF(KI+1)=Y(KI+2)
      FF(KI+2)=Y(KI+3)
   10 FF(KI+3)=DEXP(X(I))*((((X(I)+14.D0)*X(I)+49.D0)*X(I)+32.D0)*X(I)
     2   -12.D0)
      RETURN
      END
      SUBROUTINE JACOB4(X,YX,JAB)
      IMPLICIT REAL*8 (A-H,O-Z)
      DOUBLE PRECISION JAB(10,10)
      DIMENSION YX(1)
      DO 5 I=1,4
      DO 5 J=1,4
    5 JAB(I,J)=0.D0
      JAB(1,2)=1.D0
      JAB(2,3)=1.D0
      JAB(3,4)=1.D0
      RETURN
      END
      SUBROUTINE FF5(X,Y,N,FF)
      IMPLICIT REAL*8 (A-H,O-Z)
      COMMON /P5/ ALPHA1,BETA1,S1,C1
      DIMENSION X(1),Y(1),FF(1)
      DO 10 I=1,N
      KI=(I-1)*4+1
      FF(KI)=Y(KI+1)
      FF(KI+1)=BETA1*(Y(KI)-Y(KI+2))
      FF(KI+2)=Y(KI+3)
   10 FF(KI+3)=ALPHA1*(Y(KI+2)-Y(KI))
      RETURN
      END
      SUBROUTINE JACOB5(X,YX,JAB)
      IMPLICIT REAL*8 (A-H,O-Z)
      COMMON /P5/ ALPHA1,BETA1,S1,C1
      DOUBLE PRECISION JAB(10,10)
      DO 10 I=1,4
      DO 10 J=1,4
   10 JAB(I,J)=0.D0
      JAB(2,1)=BETA1
      JAB(1,2)=1.D0
      JAB(2,3)=-BETA1
      JAB(3,4)=1.D0
      JAB(4,1)=-ALPHA1
      JAB(4,3)=ALPHA1
      RETURN
      END
```

```
2210  CALL DGELG(UU,R,M,1,1.0D-7,IER)
 2250 DO 2400 I=M1,MPN
      SUM=V(I,M1)
      DO 2300 J=1,M
 2300 SUM=SUM-V(I,J)*UU(J)
 2400 UU(I)=SUM
      RETURN
      END


C-----------------------------------------------------------------


      SUBROUTINE U2DCGS(K,P,Q,N,M,A,Y,S,IERROR)
      IMPLICIT REAL*8(A-H,O-Z)
      INTEGER P,Q
      DIMENSION A(50),Y(650),S(650),C(50)
C    *****************************************************************
C    *                                                               *
C    *   THIS IS A TWO POINT BOUNDARY VALUE DEFERRED CORRECTION GENERA- *
C    *   TOR FOR SYSTEMS OF M EQUATIONS. GIVEN THE ASYMPTOTIC EXPANSION *
C    *      T(K) = SUM(A(J)*(D**(J-1))Y/(J-1) * H**(J-1))             *
C    *              J = Q+1,...Q+P*K                                 *
C    *   AND VECTOR FUNCTION VALUES  Y(1),....,Y(N+1), CORRESPONDING TO *
C    *   AN UNIFORMLY H-SPACED MESH : X(I) = X(1) + (I-1)*H , I=1,...,N+1*
C    *   U2DCGS WILL PRODUCE  S(1),....,S(N-1): AN  H**(Q+P*K)  ORDER   *
C    *   APPROXIMATION TO  T(K)  AT MIDWAY BETWEEN EACH PAIR OF CONSECU-*
C    *   TIVE GRID POINTS                                            *
C    *   FOR FIXED INTEGERS N,P,Q, A RESTRICTION ON  K  IS           *
C    ***********          K .LE. (N+1-Q)/P                ************
C    *   ALSO  P .GE. 1  ,  K .GE. 1                                 *
C    *   IERROR = 1  MEANS THAT ONE OF THESE CONDITIONS HAVE BEEN VIOLA-*
C    *   TED AND NO CORRECTION HAS BEEN COMPUTED. A(1),....,A(Q) ARE SET *
C    *   TO ZERO BY  U2DCGS.                                         *
C    *   BOTH  Y  AND  S ARE STORED AS VECTORS: Y(1,X(1)),Y(2,X(1)),.... *
C    *                                                               *
C    *   FOR MORE DETAILS SEE CHAPTER III OF 'HIGH ORDER FINITE DIFFE-  *
C    *   RENCE SOLUTION OF DIFFERENTIAL EQUATIONS' BY V. PEREYRA. TECHN *
C    *   REP. STAN-CS-73-348 , STANFORD UNIVERSITY (1973).           *
C    *                                                               *
C    *   APRIL 1973  ********* M. LENTINI  & V. PEREYRA  ****************
C    *                                                               *
C    *****************************************************************
      IF (K .GT. (N+1-Q)/P .OR. P .LT. 1 .OR.             K .LT. 1)
     1  GO TO 100
      IF (Q.EQ.0) GO TO 10
      DO 20 I=1,Q
  20  A(I)=0.
10    KK1=Q+P*K
      KK=KK1-1
      KMID=KK1/2
      IERROR=0
      KMID1=KMID-1
```

```
C
C      UNSYMMETRIC APPROXIMATION  LEFT BOUNDARY
C
  1    IF(KMID1 .LT. 1) GO TO 25
       DO 5 I=1,KMID1
       CALL COEGEN(KK1,I,C,A)
       DO 7 L=1,M
       ACUM=0.
       DO 4 J=1,KK1
  4    ACUM=ACUM+C(J)*Y((J-1)*M+L)
       IT=(I-1)*M+L
  7    S(IT)=ACUM
  5    CONTINUE
C
C      CENTER RANGE
C
 25    CALL COEGEN(KK1,KMID,C,A)
       NF=N+1-KK1+KMID
       DO 40 I=KMID,NF
       II=I-KMID
       DO 39 L=1,M
       ACUM=0.
       DO 38 J=1,KK1
 38    ACUM=ACUM+C(J)*Y((II+J-1)*M+L)
       IT=(I-1)*M+L
 39    S(IT)=ACUM
 40    CONTINUE
C
C      RIGHT  BOUNDARY
C
       KMIDP1=KMID+1
       DO 50 I=KMIDP1,KK
       CALL COEGEN(KK1,I,C,A)
       II=N-KK
       DO 49 L=1,M
       ACUM=0.
       DO 48 J=1,KK1
 48    ACUM=ACUM+C(J)*Y((II+J-1)*M+L)
       IT=(I+II-1)*M+L
 49    S(IT)=ACUM
 50    CONTINUE
       RETURN
100    IERROR=1
       RETURN
       END
       SUBROUTINE COEGEN(N,NP,C,BB)
       IMPLICIT REAL*8(A-H,O-Z)
       DIMENSION C(50),BB(50),ALF(50)
C     ******************************************************************
C     *   THIS IS A SLIGHTLY MODIFIED VERSION IN FORTRAN IV OF THE ALGOL *
C     *   PROCEDURE  PVAND , P. 901 OF "SOLUTION OF VANDERMONDE SYSTEMS  *
C     *   OF EQUATIONS" BY  A. BJORCK AND V. PEREYRA. MATH. COMP. VOL. 24*
C     *   PP. 893-903 (1970), WHERE A COMPLETE DESCRIPTION OF THE METHOD *
C     *   USED CAN BE FOUND.                                             *
C     ******************************************************************
```