NUMERICAL METHOD FOR SOLUTION OF SYSTEMS OF NON-STATIONARY SPATIALLY ONE-DIMENSIONAL NONLINEAR DIFFERENTIAL EQUATIONS

S. K. Morozov and O. P. Krasitskiy

(NASA-TM-75557) NUMERICAL METHOD FOR SOLUTION OF SYSTEMS OF NON-STATIONARY SPATIALLY ONE-DIMENSIONAL NONLINEAR DIFFERENTIAL EQUATIONS (National Aeronautics and Space Administration) 35 p HC A03/MF G3/64 36859

We propose a computational scheme and a standard program for solving systems of nonstationary spatially one-dimensional nonlinear differential equations using Newton's method. The proposed scheme is valuable because of its universality and the fact that it reduces to a minimum the work of programming. We give a detailed description and present the text of the standard program which realizes this computational scheme. The program is written in the FORTRAN language and can be used without change on electronic computers of type YEs and ESM-6. The standard program described permits us to find nonstationary (or stationary) solutions to systems of spatially one-dimensional nonlinear (or linear) partial differential equations. The proposed method may be used to solve a series of geophysical problems which take chemical reactions, diffusion, and heat conductivity into account; to evaluate nonstationary thermal fields in twodimensional structures when in one of the geometrical directions it can take a small number of discrete levels, and to solve problems in nonstationary gas dynamics.
We propose a computational scheme and a standard program for solving systems of nonstationary spatially one-dimensional nonlinear differential equations using Newton's method. The proposed scheme is valuable because of its universality and the fact that it reduces to a minimum the work of programming. We give a detailed description and present the text of the standard program which realizes this computational scheme. The program is written in the FORTRAN language and can be used without change on electronic computers of type YeS and BESM-6. The standard program described permits us to find nonstationary (or stationary) solutions to systems of spatially one-dimensional nonlinear (or linear) partial differential equations. The proposed method may be used to solve a series of geophysical problems which take chemical reactions, diffusion, and heat conductivity into account, to evaluate nonstationary thermal fields in two-dimensional structures when in one of the geometrical directions it can take a small number of discrete levels, and to solve problems in nonstationary gas dynamics.

INTRODUCTION

In contemporary science and engineering practice, one often encounters situations when a specialist not familiar with the methods of computational mathematics and the fine points of programming must solve on an electronic computer one or another mathematical problem. In this case, he turns to a library of standard programs. For example, standard programs have yielded a great extension of the solution to the Cauchy problem for systems
of ordinary differential equations by methods of Runge-Kutta type. The level of the developments of present-day computational engineering permits us to pose a question about the creation of standard programs for more complex mathematical problems. In this work, we propose a standard program for solving the boundary problem for a system of nonstationary spatially one-dimensional equations. A very wide class of physical problems can be reduced to such a system. For example, many models of geophysical phenomena, in which diffusion (heat conductivity) is the dominant factor, are often formulated mathematically as mixed problems for equations of parabolic type: either initial or boundary conditions are given. As a rule, the unknowns are related to each other in a nonlinear manner, which significantly complicates the solution.

At the basis of the method is the use of the implicit difference scheme. To solve nonlinear difference equations at each time step, we use Newton's iterative method. At each iteration, a linear system of algebraic equations is solved by a modified method of Gauss, taking into account the sparseness of the matrix, by choosing a pivot element by column and a normalization by row. They behave in an analogous way in problems of mathematical chemistry (see collections [1, 2]). In geophysics a similar method was used in work [3].

The system of equations under consideration can be very stiff, i.e., it can contain time constants with essentially different values. For example, in geophysical and astrophysical problems describing the distribution of atmospheric gaseous components by height (the determining processes are chemical reactions and vertical diffusion), the stiffness depends on the great difference in the rates of chemical decay of the various components, and also on the coefficients of diffusion at various heights. It is known that, on account of a stiff restriction intrinsic to them at a step of time, it is unreasonable to use explicit difference schemes to solve stiff systems of differential equations, because the utilization of such schemes requires a large outlay of machine time while, in most cases, such systems in practice cannot be solved by explicit methods on present-day electronic computers.

The proposed scheme is especially advantageous when the determination of an exact solution in time is not required, but we are required to find the steady-state behavior. In cases of the lack of the smallness of the coefficients of the spatial derivatives, as this often occurs in
problems of mathematical chemistry, the scheme can be shown to be unstable (the scheme is A-stable \( \Delta \)). In such cases, we need to watch for negativity of the eigenvalues or, at any rate, for positivity of the solution, and to control the pace of time or to use further iterations, excluding the result of the solution in the negative domain (see, for example, the article of Yu.N. Volin et al. and the article of B.V. Pavlov in collection \([1]\)).

Linearization by Newton's method permits us to provide an effective iteration process for solving nonlinear problems, to easily guarantee the conservativity of the difference scheme (a property of the scheme without which it is often impossible to obtain a numerical solution to the original system of differential equations).

It is well-known how much work and time it takes to write down and debug a program. Therefore, by writing down the program, we set up the problem, reducing to a minimum the work of programming and ensuring the possibility of an operational introduction of changes in the original model (for example, in geophysical problems the addition of new components and chemical reactions). This problem is solved by the calculation of a Jacobian matrix by means of numerical differentiation. In this way, we decrease the probability of error in the written program, while the user is freed from tiresome work and can concentrate his attention on the physical formulation of the problem and on the numbers of the experiment.

1. THE MATHEMATICAL METHOD

Let there be given the system of equations

\[
\frac{\partial \gamma_k(t, x)}{\partial t} = \int f_k(t, x, \gamma_i, \frac{\partial \gamma_i}{\partial x}, \frac{\partial^2 \gamma_i}{\partial x^2}), \tag{1}
\]

where \( k, i = 1, \ldots, M \) (i.e., into \( f_k \) enter terms with different \( i = 1, \ldots, M \)), \( M \) is the number of equations, \( \gamma_k \) is an unknown function, \( t \) is time, and \( x \) is the spatial coordinate.

We are given some initial conditions

\( \Delta \) The definition of A-stability can be found, for example, in the article by L.S. Polak et al. in the collection \([2]\).
We seek a solution on the interval $a \leq x \leq b$.

The boundary conditions have the form:

\begin{align}
\psi_{k}(y_{i}(a), \frac{\partial y_{i}(a)}{\partial x}) &= 0, \\
\psi_{k}(y_{i}(b), \frac{\partial y_{i}(b)}{\partial x}) &= 0,
\end{align}

where $i, k = 1, \ldots, M$.

We pass from continuous variables to discrete ones. For this we choose a uniform mesh in $x$:

$$x_{n} = a + (n-1) \cdot \Delta x; \quad n = 1, \ldots, N; \quad \Delta x = (b-a)/(N-1);$$

(if it is necessary to introduce a nonuniform mesh, we can usually make a substitution of $x$ by $x'$, so the mesh in $x'$ will be uniform). The time steps will be numbered by means of the index $j$, while the moments of time corresponding to the steps will be denoted $t_{j}$. By means of $y_{k,n}^{j}$ we denote approximate values of the continuous variable $y_{k}(t_{x}, x)$ at the point $(t_{j}, x_{n})$.

The derivatives entering into $f_{k}$ are approximated by difference relations of not more than three points $x_{n}$. For example:

\begin{align}
\frac{\partial y_{i}^{j}(t_{j}, x_{n})}{\partial x} &\approx \left( \frac{y_{i,n+1}^{j} - y_{i,n-1}^{j}}{2 \Delta x} \right), \\
\frac{\partial^{2} y_{i}^{j}(t_{j}, x_{n})}{\partial x^{2}} &\approx \left( \frac{y_{i,n+1}^{j} - 2y_{i,n}^{j} + y_{i,n-1}^{j}}{\Delta x^{2}} \right),
\end{align}

while the derivatives entering into $\psi_{k}$ and $\bar{\psi}_{k}$ are approximated by the relations

\begin{align}
\frac{\partial y_{i}^{j}(t_{j}, a)}{\partial x} &\approx \left( \frac{y_{i,2}^{j} - y_{i,1}^{j}}{\Delta x} \right).
\end{align}
In order to avoid restrictions on the time step $\Delta t$ connected with instability, we make use of a two-tiered implicit scheme:

$$\frac{y_{k,n}^d - y_{k,n}^{d-1}}{\Delta t_j} = \beta^j \left( y_{k,n+1}^d - y_{k,n}^d, y_{k,n}^d, y_{k,n+1}^d \right),$$

where $f_{k,n}^j$ is an approximate value of $f_k$ at the point $n$ at moment of time $t_j$, which is obtained by the substitution of expressions (6) and (7) into $f_k$ for $n = 2, \ldots, N-1$.

The boundary conditions can be approximated by the expressions

$$\psi_k^d (y_{i,1}^d, y_{i,2}^d) = 0,$$

$$\psi_k^d (y_{i,N-1}^d, y_{i,N}^d) = 0,$$

where $\psi_k^d$ and $\psi_k^d$ are approximate expressions for $\psi_k$ and $\psi_k$, which are used in the substitution of expressions (8),(9) into (3),(4).

Equation (10) has first-order accuracy in time. To increase the accuracy in time, we can put the right-hand side in the form

$$\beta \cdot \frac{\psi_k^d}{\Delta t_j} + (1-\beta) \cdot \frac{\psi_k^d}{\Delta t_j},$$

where $\beta > 0.5$

For $\beta = 0.5$, equation (10) approximates the differential equation (1) with double precision in time. For $\beta < 0.5$, the difference scheme becomes unstable.

To increase the accuracy of approximating the boundary conditions, the derivatives entering into $\psi_k^d$ and $\psi_k^d$ can be written in the form:

$$\frac{\partial y_{i,j}(t_x,a)}{\partial x} \approx \int -3y_{i,j}^d + 4y_{i,j}^d - y_{i,j}^d \, \Delta x,$$
\[
\frac{\partial Y_i(x,t)}{\partial x} \approx \frac{Y_{i,n-2}^i - 4Y_{i,n-1}^i + 3Y_{i,n}^i}{2\Delta x}.
\]

In this case the derivatives mentioned above are approximated with double precision in \(x\).

Equations (10), (11), and (12) form a nonlinear system of algebraic equations, consisting of \(N-M\) equations and \(N\) unknowns. We will solve this system by Newton's method, i.e., at each iteration we solve the following system of linear algebraic equations:

\[
\begin{align*}
\frac{y_{k,n}^i - y_{k,n}^{i-1}}{\Delta t} &= \sum_{l=1}^{M} \sum_{i=1}^{3} \frac{\partial \psi_{k,n}}{\partial y_{i,n}^{i-1}} (y_{i,n}^{i+1} - y_{i,n}^{i-1}) + \psi_{k,n}^i = 0, \\
\sum_{i=1}^{M} \sum_{l=1}^{3} \frac{\partial \psi_{k,n}}{\partial y_{i,n}^{i+1}} (y_{i,n}^{i+1} - y_{i,n}^{i-1}) + \psi_{k,n}^i &= 0.
\end{align*}
\]

For \(S = 0\), at the zero-th approximation the values begin with the previous time period: \(y_{k,n}^i = y_{k,n}^{j-1}\). The derivatives of the functions \(\psi_{k,n}\) and \(\psi_{k,n}^{i-1}\) with respect to the unknowns (of the Jacobi matrix) are found by numerical differentiation.

The numerical differentiation is effected by the formula:

\[
\frac{\partial \psi_{k,n}}{\partial y_{i,n}^{i-1}} = \frac{\psi_{k,n}(y_{i,n}^{i+1}, y_{i,n}^{i-1}) - \psi_{k,n}(y_{i,n}^{i+1}, y_{i,n}, y_{i,n})}{\Delta Y_{i,n}}.
\]

For clarity and convenience of our further presentation, we represent the system of equations (16), (17), and (18) in matrix form:

\[
\begin{bmatrix}
A_N & B_N & C_N & 0 & 0 & 0 & 0 & 0 & 0 & F_N & D_N \\
A_{N-1} & B_{N-1} & C_{N-1} & 0 & 0 & 0 & 0 & 0 & 0 & F_{N-1} & D_{N-1} \\
0 & A_{N-2} & B_{N-2} & C_{N-2} & 0 & 0 & 0 & 0 & 0 & F_{N-2} & D_{N-2} \\
0 & 0 & 0 & A_3 & B_3 & C_3 & 0 & F_3 & D_3 \\
0 & 0 & 0 & 0 & A_2 & B_2 & C_2 & F_2 & D_2 \\
0 & 0 & 0 & 0 & A_1 & B_1 & C_1 & F_1 & D_1
\end{bmatrix}
\]
Here A, G, and C are square matrices of size $M \times M$, and F and D are $M$-dimensional vectors. In this form, the matrix has a block-tridiagonal form under the condition that each of its block elements, in turn, is a square matrix. In the upper and lower rows of the matrix equation (19) we write equations (18) and (17), while in the interior rows is equation (16). The vector

$$F_n = \left( y_{i,1}^{j,s+1}, \ldots, y_{i,M}^{j,s+1} \right), \quad n = 1, \ldots, 2N.$$ 

Expressions for the elements of the matrices A, B, and C and the components of the vector D have the form:

1) for $n = N$ and $i, k = 1, \ldots, M$

$$a_{n,i} = \frac{\partial y_{i}^{j,s}}{\partial y_{i,n}^{j,s}}, \quad b_{n,i} = \frac{\partial y_{i}^{j,s}}{\partial y_{i,n-1}^{j,s}}, \quad c_{n,i} = \frac{\partial y_{i}^{j,s}}{\partial y_{i,n}^{j,s}},$$

$$d_{n} = \sum_{i=1}^{M} \sum_{l=1}^{3} \frac{\partial y_{i}^{j,s}}{\partial y_{i,n+l}^{j,s}} y_{i,n+l-3}.$$ 

2) for $n = 2, \ldots, N-2$ and $i, k = 1, \ldots, M$

$$a_{n,i} = \frac{\partial y_{i}^{j,s}}{\partial y_{i,n}^{j,s}}, \quad b_{n,i} = \delta_{k,i} + \frac{\partial y_{i}^{j,s}}{\partial y_{i,n}^{j,s}} - \frac{\partial y_{i}^{j,s}}{\partial y_{i,n-1}^{j,s}}, \quad c_{n,i} = -\frac{\partial y_{i}^{j,s}}{\partial y_{i,n}^{j,s}},$$

$$d_{n} = \frac{y_{i}^{j,s}}{\Delta t_{j}} + \frac{\partial y_{i}^{j,s}}{\partial y_{i,n}^{j,s}} - \sum_{i=1}^{M} \sum_{l=1}^{3} \frac{\partial y_{i}^{j,s}}{\partial y_{i,n+l-2}^{j,s}}.$$ 

where $\delta_{k,i}$ is the Kronecker delta symbol;

3) for $n = 1$ and $i, k = 1, \ldots, M$

$$a_{n,i} = \frac{\partial y_{i}^{j,s}}{\partial y_{i,n}^{j,s}}, \quad b_{n,i} = \frac{\partial y_{i}^{j,s}}{\partial y_{i,n}^{j,s}}, \quad c_{n,i} = \frac{\partial y_{i}^{j,s}}{\partial y_{i,n}^{j,s}},$$

$$d_{n} = \sum_{i=1}^{M} \sum_{l=1}^{3} \frac{\partial y_{i}^{j,s}}{\partial y_{i,n+l}^{j,s}} y_{i,n+l}.$$ 

The linear system (19) at each iteration $S$ is solved by a modification of the method Gauss, which takes into account the tridiagonality of the matrix, with the choice of a pivot element by row and a normalization by column [5]. The basic idea of the elimination method consists of the fact that we use a linear transformation to make the diagonal elements of the
matrix into ones, and the below-diagonal elements zeros. This is done sequentially from top to bottom. To demonstrate the recurrence formula for passing from \( n \) to \( n-1 \), we introduce the rectangular matrix

\[
BZ = \begin{bmatrix}
\bar{B}_n & \bar{C}_n & 0 & \bar{D}_n \\
A_{n-1} & B_{n-1} & C_{n-1} & D_{n-1}
\end{bmatrix}
\]

The meaning of the matrices \( \bar{B}, \bar{C} \) and the vector \( \bar{D} \) will become clear from further presentations.

We take the first column of the matrix \( BZ \), find in it the maximal element as a model, and place the row in which it is found in the first position of the matrix \( BZ \). We divide each element of this row by its first element. Now, subtracting the first from the other rows, after multiplying by an appropriate factor, we obtain zeros in the first column of these rows. We find the maximal element in the second column among all rows except the first, and by means of this row we obtain zeros in the second column of all rows except the first. We conduct an analogous transformation, further, with the third column, etc. As a result the matrix \( BZ \) is reduced to the form

\[
\tilde{BZ} = \begin{bmatrix}
E & P_n & Q_n & R_n \\
0 & \bar{B}_{n-1} & \bar{C}_{n-1} & \bar{D}_{n-1}
\end{bmatrix}
\]

where \( E \) is the identity matrix.

For \( n = N \), we have

\[
BZ = \begin{bmatrix}
A_N & B_N & C_N & D_N \\
A_{N-1} & B_{N-1} & C_{N-1} & D_{N-1}
\end{bmatrix}
\]

and the passage from \( N \) to \( N-1 \) is included in the standard scheme of the passage from \( n \) to \( n-1 \).

For \( n = 1 \), before the standard scheme of passage is applied, \( A_1 \) is eliminated by means of the identity matrix obtained for \( n = 2 \). In other words, the matrix
by means of the matrix

\[
\begin{bmatrix}
A_1 & B_1 & C_1 & D_1 \\
E & P_3 & Q_3 & R_3 \\
0 & B'_1 & C'_1 & D'_1 \\
\end{bmatrix}
\]

reduces to the form

\[
\begin{bmatrix}
0 & B'_1 & C'_1 & D'_1 \\
\end{bmatrix}
\]

Then the standard scheme is applied to the matrix

\[
\begin{bmatrix}
B_{2} & C_{2} & O & D_{2} \\
B'_1 & C'_1 & O & D'_1 \\
\end{bmatrix}
\]

As a result, we obtain

\[
\begin{bmatrix}
E & P_2 & O & R_2 \\
0 & C'_1 & O & D'_1 \\
\end{bmatrix}
\]

from which

\[
F_1 = (C'_1)' \cdot D'_1
\]

and then

\[
F_2 = R_2 - P_2 \cdot F_1
\]

And finally the recurrence formula for finding the solution has the form

\[
F_n = R_n - P_n \cdot F_{n-1} - Q_n \cdot F_{n-2}, \quad n = 3, \ldots, N.
\]

Thus we find an approximate solution to the boundary value problem (1-4) at the moment of time \(t_j\) in \(S+1\) iterations. If the iterations correspond to a given precision, then we can continue iterating equation (1) in time.

2. DESCRIPTION OF THE PROGRAM

The program permits us to find a nonstationary solution \(y_1(t,x)\) and a steady-state solution \(y_1(x)\). The nonstationary solution can be found by integrating equation (1) with a constant step in \(t\) or with an automatic
choice of step. In the case of an "automatic choice of step", the time step is chosen depending on the number of iterations with respect to nonlinearity at the given step. Thus, the controls on precision do not depend on time.

The steady-state solution can be found by the method of settling down, i.e. we integrate in time up to the moment of time exceeding the maximal characteristic time of the problem (better with an automatic choice of step), or by iterations without settling down in time. In the latter case, we can solve directly a system of stationary equations, i.e. equations (1) without time derivatives.

Into the standard part of the program, we introduce three subroutines: HOK, OUT, and ISCL.

The subroutine HOK controls the iterations, the choice of step in time, the printing of the result, etc.

The subroutine OUT effects the printing of the result.

In the subroutine ISCL we calculate the coefficients for the unknowns in a linear system of difference equations, and this system is solved by Gauss's method.

The user writes the subroutine FF, where the difference equations are written down, and the basic program, where values are given to the physical constants and from which originates the access to the standard part of the program.

The access to the subroutine has the form:

\[ \text{CALL: HOK(N,M,IB,ITAU,ITER,ITB,E PS,T KOH,TAU,Y0,X0,K1) } \]

Here \( N \) is the number of partition points of the x-interval.

\( M \) is the number of equations.

\( IB \) is the number of steps \( \Delta T \) in time, across which the printout of the solution is produced. When it is necessary to print out the results at fixed moments of time \( t_n \), we need to sequentially access the subroutine HOK, giving \( TKOH = t_n \).

\( ITAU = 0 \) is the signal of a calculation with constant step \( \Delta T \) in time.

\( ITAU \neq 0 \) is the signal for a calculation with automatic choice of time step. The step is increased by 1.5 times when the number of iterations at the previous step equals 1 and decreases by 2 times when it is greater than 3. Thus, there is no linear relationship between the choice of step and the accuracy of the solution in time. When we need to obtain a steady-state solution by the method of...
settling down, then we should set $\text{ITAU} \neq 0$.

**ITER** is the maximum allowable number of iterations at the step $\text{TAU}$; when the number of iterations $\text{IT}$ exceeds $\text{ITER}$, there occurs a decrease in the step $\text{TAU}$ by two times.

$\text{ITB}=1$ is the signal to print the solution at each iteration.

**EPS** is the relative accuracy of the convergence of the iterations. The magnitude of the increment $\frac{\Delta y_{i,n}}{y_{i,n}}$ in the numerical differentiation is given by the formula

$$
\frac{\Delta y_{i,n}}{y_{i,n}} = \left(\frac{1}{\text{eps}} + \frac{1}{\text{eps}^2}\right) \left(|y_{i,n-1} + y_{i,n+1} - 2y_{i,n}| + |y_{i,n}|\right),
$$

when $\frac{\Delta y_{i,n}}{y_{i,n}} = 0$ (for example, when the initial approximation $y_{i,0} = 0$) $y_{i,n} = \text{EPS}$ (in a program with double precision $\Delta y_{i,n} = (\text{EPS})^2$).

$\text{TKOH}$ is the end of the interval of integration in time; when $\text{TAU} > \text{TKOH}/10$, the step $\text{TAU}$ does not increase; when $\text{TAU} < \text{TKOH}/10^5$, the calculation ceases and the statement "THE TAU IS SMALLER THAN ALLOWED" is printed, and the values of the current moment of time and $\text{TAU}$ are given.

$\text{TAU}$ is the value of the time step: the initial step or constant step, depending on the value of the parameter $\text{ITAU}$. When $\text{TAU} = 0$, we seek a steady-state solution of the system directly by iterations without a settling-down in time. In this case, when the number of iterations $\text{IT} > \text{ITER}$, the statement "IT = ITER ITERATIONS DO NOT CONVERGE WITH THE GIVEN ACCURACY EPS" is printed and the values of the next two iterations are given.

$\text{X0}$ is the dimensionality file $N \cdot M$ of the initial conditions; for the output of the subroutine HOK, $\text{X0}$ is a solution for $t = \text{TKOH}$ or in case $\text{TAU} = 0$ a steady-state solution.

$\text{X0}$ is the left endpoint of the interval of $x$'s.

$\text{XL}$ is the right endpoint of the interval of $x$'s.

The subroutine HOK works with the subroutines ISCL, OUT, FF, and the library subroutine ABS.

ISCL is the subroutine for solving the system of linear algebraic equations. An accidental result of the form "MATRIX IS DEGENERATE............." is possible. Such a result, as a rule, indicates an improper formulation of the problem (an incorrect assignment of initial or boundary conditions) or errors in the subroutine FF (improperly written equations, errors in the coefficients, etc.)
OUT is the subroutine of printing the solution. The value of the current moment of time is printed $BP = \ldots$; the column of $x$ values and the columns of values of the unknown functions $y_k$ are also printed.

**FF** is the subroutine which is written by the user. In it are given the formulas for computing the right-hand side of $f_k$ and also of $\psi_k$ and $\varphi_k$. In this subroutine we must describe the general block HIF:

\[
\text{COMMON} / \text{HIF} / BP, H, \tau_{UL}, xH(21), f(10), \\
\varphi_0(21,10), \varphi(21,10), \varphi_1(21,10)
\]

where $BP$ is the moment of time $t_j$; $H$ is the coordinate step; $\tau_{UL}$ is the reciprocal value of the time step $\tau_{UL}$; $xH$ is the file of $x$ for $n=1, \ldots, N$; $\varphi_0$ is the value of the function at level $t_{j-1}$; and $\varphi$ is the value of the function at level $t_j$ of the previous iteration.

Access to the subroutine FF originates from the standard part by means of the operator CALL FF(N,NN), where $N$ is the current index of the point $x_n$, NN is the collection of partition points $x_n$, $1 \leq N \leq NN$. Correspondingly, the first operator of the subroutine FF must have the form: SUBROUTINE FF(N,NN), where $N$, NN are formal parameters having the above-mentioned meanings. The connection between the subroutine FF and the basic program can be effected through additional general blocks (also including unmarked ones).

On the basis of the data, we calculate the

\[
\text{of } F \text{ of dimension } M, \text{ the elements of which are given by the file: numerical values of expressions representing themselves or, either, difference approximations of the right-hand sides of the equations at the point } x_n, \text{ or a difference approximation of the boundary conditions.}
\]

The variables entering into the general block HIF cannot be used in the subroutine FF as identifiers of other variables.

To economize machine time, it follows that we should avoid in the subroutine FF calculations of expressions not depending on $j$ and $n$, because to calculate the Jacobian matrix at each iteration at each point $x_n$ is the result of $M+1$ accesses to the
subroutine FF. Furthermore, if these expressions enter the subroutine FF, then they also are computed M+1 times.

In the operators of description COMMON and DIMENSION, the files of the variables are written so that we can use the program when $N \leq 21$ and $M \leq 10$. When it is necessary to increase $N$ or $M$, it follows that we should insert corresponding changes into these operators. In different programs the operators have the following form:

1) $\text{COMMON} / H_1 / B_P, H, TAU1, XH(N), F(M), Y_0(N,M), Y(N,M), Y_1(N,M)$

in the subroutines HOK, ISCL, FF and OUT;

2) $\text{DIMENSION Y0}(N,M)$ in the basic program;

3) $\text{DIMENSION Y0}(N,M)$ in the subroutine HOK;

4) $\text{DIMENSION B2}(2*3+1), PZ(N-1,M,M), QZ(N-2,M,M), C2/(2), F1(M)$

in the subroutine ISCL.

Here the variables $N$ and $M$ and expressions containing them need to be replaced by appropriate numbers because the sizes of the dimensions of the files are given by integer constants without sign.

For example, if $N = 51$ and $M = 5$, the the operator DIMENSION in the subroutine ISCL needs to be replaced by the operator

$\text{DIMENSION B2}(49,50,5,5), PZ(49,5,5), QZ(49,5,5), C2(2), F1(5)$.

We need to make analogous changes in operators 1-3. After we have made these changes, we can use the program for arbitrary $N \leq 51$ and $M \leq 5$.

The necessity of changing the operators of description COMMON and DIMENSION for increases in $N$ or $M$ creates definite inconveniences in the work and is related to the limitations of the programming language Fortran, in which, in contrast to programming in Algol, we cannot use dynamic files. To eliminate this deficiency and also to increase the economy of the program, we should transfer from multidimensional files, to one-dimensional ones, as for example is done in the IBM library of standard programs {6}. The authors hope to return to this question at some time in the immediate future.
Example

Let there be given a system of linear differential equations of parabolic type:

\[
\begin{align*}
\frac{\partial y_1}{\partial t} &= \frac{\partial^2 y_1}{\partial x^2} + y_2, \\
\frac{\partial y_2}{\partial t} &= \frac{\partial^2 y_2}{\partial x^2}
\end{align*}
\]

(20)

where \(0.5 \leq x \leq 1, \ t \geq 0\).

We put on this system of equations the following boundary and initial conditions:

\[
\begin{align*}
\frac{\partial y_1}{\partial x} (t, 0.5) &= 0 , \quad y_1(t, 1) = 0 , \\
\frac{\partial y_2}{\partial x} (t, 0.5) &= 0 , \quad y_2(t, 1) = 0 , \\
Y_1(0, x) &= 0 , \quad Y_2(0, x) = \sin(\pi x)
\end{align*}
\]

(21)

The analytic solution to the problem formulated has the form:

\[
\begin{align*}
Y_1(t, x) &= t \cdot e^{-\pi^2 t} \sin(\pi x) , \\
Y_2(t, x) &= e^{-\pi^2 t} \sin(\pi x)
\end{align*}
\]

(22)

We choose a uniform grid in \(x\) and \(t\):

\[
\begin{align*}
L_n &= (n-1) \cdot \Delta x + 0.5 ; \quad n = 1, \ldots, 21 ; \quad \Delta x = 0.025 \\
t_{ij} &= j \cdot \Delta t ; \quad j = 0, \ldots, 100 ; \quad \Delta t = 0.001
\end{align*}
\]

The differential equations (20) may be approximated by the difference relations:
The difference expressions for the boundary and initial conditions (21) have the form

\[ \frac{y_{i,n}^j - y_{i,n}^j}{\Delta t} = 0.55 \left( \frac{y_{i,n+1}^j - 2y_{i,n}^j + y_{i,n-1}^j}{\Delta x^2} + \frac{y_{j,n}^j}{2} \right) \]
\[ \frac{y_{2,n}^j - y_{1,n}^j}{\Delta t} = 0.55 \left( \frac{y_{2,n+1}^j + 2y_{1,n+1}^j - y_{1,n}^j}{\Delta x^2} \right) \]

where \( j = 0, \ldots, 100 \), \( n = 1, \ldots, 21 \).

When we accessed the standard program HOK, the following actual parameters were given:

\[ N = 21, \ M = 2, \ IB = 50, \ ITAU = 0, \ ITER = 7, \ ITB = 0, \]
\[ EPS = 0.001, \ TKOH = 0.1, \ TAU = 0.001, \ X \phi = 0.5, \ Xf = 1. \]

In the subroutine FF, the right-hand sides of equations (23) and the boundary conditions (24) are written down. In the appendix we present the text of the subroutine FF, where \( H = \Delta x \), \( H2 = (\Delta x)^2 \), \( \Phi(k,N) = y_{k,n}^j \), \( \Phi(k,n) = y_{k,n}^{j-1} \), \( N = n, \ N = 1 \) corresponds to \( x = x \phi \), and \( N = NN \) corresponds to \( x = x_f, 1 \leq N \leq NN \).

The initial conditions are assigned in the basic program (see the text of the basic program in the appendix).

In the table we cite the approximate and exact solutions at the moment of time \( t = 0.1 \).
<table>
<thead>
<tr>
<th>$x$ Coordinate</th>
<th>$y_1$ Approximate</th>
<th>$y_1$ Exact</th>
<th>$y_2$ Approximate</th>
<th>$y_2$ Exact</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.50000</td>
<td>0.037264</td>
<td>0.037271</td>
<td>0.37300</td>
<td>0.37271</td>
</tr>
<tr>
<td>0.52500</td>
<td>0.037149</td>
<td>0.037156</td>
<td>0.37186</td>
<td>0.37156</td>
</tr>
<tr>
<td>0.55000</td>
<td>0.036805</td>
<td>0.036812</td>
<td>0.36842</td>
<td>0.36812</td>
</tr>
<tr>
<td>0.57500</td>
<td>0.036235</td>
<td>0.036241</td>
<td>0.36271</td>
<td>0.36241</td>
</tr>
<tr>
<td>0.60000</td>
<td>0.035441</td>
<td>0.035447</td>
<td>0.35476</td>
<td>0.35447</td>
</tr>
<tr>
<td>0.62500</td>
<td>0.034428</td>
<td>0.034434</td>
<td>0.34462</td>
<td>0.34434</td>
</tr>
<tr>
<td>0.65000</td>
<td>0.033203</td>
<td>0.033209</td>
<td>0.33236</td>
<td>0.33209</td>
</tr>
<tr>
<td>0.67500</td>
<td>0.031773</td>
<td>0.031779</td>
<td>0.31705</td>
<td>0.31779</td>
</tr>
<tr>
<td>0.70000</td>
<td>0.030148</td>
<td>0.030153</td>
<td>0.30178</td>
<td>0.30153</td>
</tr>
<tr>
<td>0.72500</td>
<td>0.028396</td>
<td>0.028341</td>
<td>0.28365</td>
<td>0.28341</td>
</tr>
<tr>
<td>0.75000</td>
<td>0.026350</td>
<td>0.026355</td>
<td>0.26376</td>
<td>0.26355</td>
</tr>
<tr>
<td>0.77500</td>
<td>0.024202</td>
<td>0.024206</td>
<td>0.24226</td>
<td>0.24206</td>
</tr>
<tr>
<td>0.80000</td>
<td>0.021904</td>
<td>0.021907</td>
<td>0.21926</td>
<td>0.21907</td>
</tr>
<tr>
<td>0.82500</td>
<td>0.019471</td>
<td>0.019474</td>
<td>0.19490</td>
<td>0.19474</td>
</tr>
<tr>
<td>0.85000</td>
<td>0.016918</td>
<td>0.016921</td>
<td>0.16935</td>
<td>0.16921</td>
</tr>
<tr>
<td>0.87500</td>
<td>0.014261</td>
<td>0.014263</td>
<td>0.14275</td>
<td>0.14263</td>
</tr>
<tr>
<td>0.90000</td>
<td>0.011515</td>
<td>0.011517</td>
<td>0.11527</td>
<td>0.11517</td>
</tr>
<tr>
<td>0.92500</td>
<td>0.008699</td>
<td>0.008701</td>
<td>0.08708</td>
<td>0.08701</td>
</tr>
<tr>
<td>0.95000</td>
<td>0.005829</td>
<td>0.005831</td>
<td>0.05835</td>
<td>0.05831</td>
</tr>
<tr>
<td>0.97500</td>
<td>0.002924</td>
<td>0.002924</td>
<td>0.02927</td>
<td>0.02924</td>
</tr>
<tr>
<td>1.00000</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>
Before solving complicated problems by the standard program described here, we recommend to the user that he solve one or more simple problems, the solutions are which are well-known.

CONCLUSION

The method described here was shown to be effective in solving stiff systems of differential equations for modeling the chemical composition of the atmosphere of Mars.\(7\)\, System(11) of equations of parabolic type was solved. An implicit conservative difference scheme of second-order accuracy in \(\Delta x\) was used. Steady-state and nonstationary solutions were found. The characteristic times of chemical decay for different components in this problem varied between the limits \(10^{-8}\) to \(10^{12}\) sec., while the characteristic times for the establishment of diffusion equilibrium varied depending on \(\cdot\) height within the limits \(10^{2}\) to \(10^{7}\) sec. In calculations with usual accuracy on electronic computers of type \(\text{Y}e\text{S}\) and \(\text{BESM-6}\), we obtain larger roundoff errors for time steps \(\Delta t\) of \(10^{5}\) to \(10^{7}\) sec. These errors can be explained by poor conditioning on the system of equations (19), which is a consequence of the great difference in characteristic times of the problem. It turns out that a steady-state solution is obtained only when calculating to \(\text{double precision}\). In this case there are no restrictions on the time step \(\Delta t\); because the corresponding linear problem (19), which is obtained as a result of linearizing the nonlinear difference equations, is stable. We consider the steady-state solution to be obtained when \(t \sim 10^{16}\) sec. (the maximal characteristic time of the problem \(\sim 10^{12}\) sec.).

Using the given method to solve the described system of equations permits us to obtain a series of new results on the chemical composition of the atmosphere of Mars.

\(\text{\texttt{\textdagger}}\) For an abrupt change in the solution at a given time \(\Delta t\), the emergence of negative concentrations and instability is possible. In this case, the iterations do not converge, and the program automatically changes the time step until the concentrations become positive.
The proposed method is adaptable to the solution of problems of nonstationary gas dynamics. For example, to solve two-dimensional problems of the atmospheric heat circulation of planets [8,9,10], an implicit scheme was used in which the angular derivatives were taken with preceding iterations. However, to solve the one-dimensional boundary problems obtained from it along vertical lines, a variant of the proposed method in which the Jacobian matrix was calculated analytically was used. This method is also adaptable to the calculation of thermal fields in two-dimensional structures and other problems.

In conclusion, we enumerate the basic merits and defects of the proposed method, and we give recommendations about its use. Regarding the merits of the method, we can list

1. The possibility of using the program for solving a wide variety of physical problems, among which are those having a large range of characteristic times.
2. Rapid convergence of the iterations for nonlinearity, owing to the use of Newton's method.
3. The possibility of easily constructing a conservative difference scheme.
4. The automatic linearization of nonlinear difference equations by means of numerical differentiation, which significantly reduces the work of programming.

It follows that we should consider as an essential defect of the method the fact that the calculational time (the number of arithmetic operations) in solving a system of M equations is proportional to \( M^3 \). It follows that we should stress that this defect is related to the use of Gauss' method, i.e. it turns out to be closely interrelated with the merits of our method.

Owing to, on the one hand, the stability and rapidity of convergence of the iterations, and on the other hand, the reduction to a minimum of the programming work of the described method, it can find wide application in scientific and engineering computations. We note that more economical methods, in which the calculational time is proportional to \( M^2 \), for example, often possess slower rates of convergence of the iterations and, as a result, can prove less economical.
Recommendations for Use

1. Linear equations.
   a. Stationary equations.

   To solve such problems, it is necessary to solve only once a corresponding system of difference equations, i.e. we need one iteration. For that one iteration to be made, it follows that we should set \( \text{ITER} = 1 \). However, by convention, it is useful to set \( \text{ITER} > 1 \) (for example, \( \text{ITER} = 5 \)) and \( \text{ITB} = 1 \). Then when there are no errors, one superfluous iteration will be made. If, however, there is an error, then with a significant probability, the iterations will not converge. The result of each iteration will be presented in print.

   b. Nonstationary equations.

   Because there is in the program no control over the time precision, then in order to guarantee it, it follows that we should solve the problem several times with different constant time steps \( \Delta t \) and should equalize the results obtained.

2. Nonlinear equations.
   a. Stationary equations.

   If the equations are strongly nonlinear and the form of the solution is unknown beforehand, then it is necessary to solve the problem by the method of bisection. In other words, it is necessary to give an arbitrary initial approximation and time step \( \Delta t \), which must be smaller than the minimal characteristic time of the problem. We need to undertake the calculation with an automatic choice of step up to a moment of time larger than the maximal characteristic time of the problem.

   b. Nonstationary equations.

   Here the same remarks as those in item "b" for linear equations are valid.

   We can use the method to solve multidimensional problems by methods of coordinate-wise decomposition.

   In the case of solving systems of equations having a large difference in characteristic times for different functions, i.e. stiff systems, we recommend the use of a program with double precision, the text of which can be found in the appendix. It follows, in view of this, that such a program will occupy twice as much space in the machine's memory. On an electronic computer of type YeS, the calculation time for such a program cannot be
practically increased, but on a BESM-6, the space is increased five-fold. On the BESM-6, a number with ordinary precision has 12 decimal places, but on a computer of type YeS only 7. Therefore, for calculations on a computer of type YeS, we always recommend the use of double precision, because, on account of the restricted number of spaces, round-off errors which are too large can occur in numerical differentiation and in solving a system of algebraic equations.

To solve systems of differential equations with derivatives no higher than first order as part of the boundary conditions, we can use difference equations written on one of the boundaries. In this case, the difference analogue of the time derivative is written down in a corresponding expression for \( \psi \) or \( \psi \) (see formulas (11),(12)).

If an equation not containing a time derivative enters the system, then it follows that we should multiply this equation by a sufficiently small number.

We recall that the choice of a difference scheme belongs to the user. We have only the following requirements for it:

a) a scheme of ordered pairs \((t_{j-1}, t_j)\) in time,

b) in the coordinate \(x\), we establish at each partition point \(x_n\) not more than three values of each of the unknown functions \(y_{k,n-1}^j, y_{k,n}^j, y_{k,n+1}^j\).

Programs with ordinary and double precision were prepared for electronic computers of type YeS and BESM-6 and can be used on them without any changes.
REFERENCES

1. Proceedings of the All-Union Symposium "Mathematical Problems in Chemistry," (Trydy vsesoyuznogo simpoziuma "matematicheskiye problemy khimii"), Computa­

2. Proceedings of the All-Union Symposium "Mathematical Problems in Chemistry," (Trudy vsesoyuznogo simpoziuma "matematicheskiye problemy khimii"), Computa­
tional Center of the Siberian Branch of the Academy of Sciences of the USSR, Novosibirsk, 1975.


6. Sbornik Nauchnykh Program na Fortranе. Collection of Scientific Programs in Fortran, No. 2, Moscow, Statistika


APPENDIX

Program with Ordinary Accuracy

Basic Program

```
DIMENSION YO(21,10)
T=0
ITER=7
ITAU=0
IKC=0
TAU=0.001
TKOH=0.01
EPS=0.001
X0=0.5
X1=1.
M=2.
N=23
DO 2 I=9,N
   Y0(I,1)=SIN(3.14159*(X0*(X1-X0)*FLOAT(I-1)/FLOAT(N-1)))
   CALL HOK(N,M,18,ITAU,ITER,ITER,EPS,TKOH,TAU,Y0,X0,X1)
2      PRINT 3
      FORMAT (*CALCULATION OF THE SOLUTION OF A CONTROL EXAMPLE AT MOMENT TKOH*,")
      FORMAT (32H COMPARISON OF THE OBTAINED AND EXACT \ ,20H SOLUTIONS FOR BP = TKOH*)
      FORMAT (4E15.5)
      PRINT 8
      PRINT 9(Y0(I,J),J=1,4)
      FORMAT (4E15.5)
      END

Subroutine FF

SUBROUTINE FF(N,N)
COMMON (HIF,BP,TAU,IKC),FCIO),YO(21,10),Y(21,10)
N2=4lN
IF(N.EQ.N) GOTO 1
F(2)=Y(N,2)
GOTO 3
1   F(2)=Y(N,2)
2   CONTINUE
   F(2)=Y(N-1,1)-2.*Y(N,1)+Y(N-2,1)/H2+Y(N-2,2)
   Y0(N,2)=0.85*(Y(N-1,1)-2.*Y(N,1)+Y(N-2,1))/H2+Y(N-2,2)*0.45
   CONTINUE
   RETURN
   END
```

Boundary conditions on the right end
- $F(2) = Y(N,2)$

Boundary conditions on the left end
- $F(2) = 3.*Y(1,2)+4.*Y(2,2)-Y(3,2)$

Right-hand sides of equations
- $2 \text{ CONTINUE}$
- $F(2) = Y(N-1,1)-2.*Y(N,1)+Y(N-2,1)/H2+Y(N-2,2)$
- $Y0(N,2) = 0.85*(Y(N-1,1)-2.*Y(N,1)+Y(N-2,1))/H2+Y(N-2,2)*0.45$
- $3 \text{ CONTINUE}$
Subroutine H0K

SUBROUTINE H0K(N, M, IB, ITAU, ITER, ITB, EPS, TKOH, TAU, Y00, X0, X1)

COMMON /HP, BP/ K, TAU,

DIMENSION Y00(21,10), Y1(21,10)

COMMENTS ON

1 FORMAT(/)

2 FORMAT(/)

3 FORMAT(/)

FORMAT(/)

FORMAT(/)

JBR=0

/ XH(1)=X0

IF(ITAU.EQ.0.) GOTO 72

CALL IN/.

IF(IT ,EQ.1.) GOTO 50

DO 28 I=1, N

DO 28 J=1, M

Y(I,J)=Y0(I,J)

IF(ITB .EQ.1.) GOTO 55

GOTO 52

c Comparing the convergence of iterations

DO 27 I=1,N

DO 27 J=1,M

IF(ABS(Y(I,J)).LE.1.E-10) GOTO 27

IF(ABS(Y(I,J)-Y1(I,J))/Y1(I,J)).LE.EPS) GOTO 27, 27, 32

CONTINUE

IF(ITAU .EQ.0.) GOTO 57

Preparation for the next step

DO 26 I=1, N

DO 26 J=1, M

Y(I,J)=Y1(I,J)

26.

IF(ITAU .EQ.0.) GOTO 58

IF(IT ,EQ.1.) GOTO 43

IF(IB.NE.0) GOTO 30

Increase in step by TAU

if(IT

IF(IB .EQ.0.) GOTO 43

IF(IT ,EQ.1.) AND..TAU.LT.TKOH/1n..) GOTO 41

Decrease in step by TAU

c Delivery of a result with period IB

JB=0

CALL IN/.

GOTO 30

72 TAU=TKOH-BP+TAU

BP=TKOH

TAU=TAU/2.
FORMAT (34H, Step TAU is smaller than the permitted time = )

STOP

IF (TAU .EQ. 0) GO TO 52
DO 34 J = 1, M
31 Y (1, J) = YO (1, J)
GO TO 42

FORMAT (39H, IT = ITER Iterations do not converge with accuracy EPS)

STOP

FORMAT (10H, Iteration = )

PRINT 56, I
GO TO 31

CONTINUE

PRINT 56, I
DO 59 J = 1, M
DO 59 J = 1, M
59 Y0 (1, J) = YV (1, J)
CALL OUT (H, N)
RETURN
END

Subroutine ISCL

SUBROUTINE ISCL (NM, NN, EPS)

DIMENSION BZ (20, 20), Z (20, 10, 10), Q (10, 10, 10), C (10)
COMMON (H, EPS) / BP, W, TAU /

C Cycle in N

DO 102 J = 1, M / 102 BZ (1, J) = 0

DO 101 J = 1, M / 101 BZ (1, J) = 0

C Search for the Pivot Element

DO 113 K = 1, M / 113 K = ABS (C (1, 1))

C Pivot element is close to zero

IF (ABS (C (1, 1)) .LE. EPS) GO TO 113

C Rearrangement of the rows and elimination

DO 114 K = 2, M / 114 K = C (1, 1) / C (1, 1)

C Return

RETURN
END
Formation of the matrices P, Q, R

DO J=1,N
IF(J.EQ.1) GOTO 116
IF(J.EQ.N) GOTO 117
CONTINUE
CONTINUE
CONTINUE
CONTINUE
DO J=1,N
DO J=1,N
DO J=1,N
DO J=1,N

End of the cycle in N

DO J=1,N
DO J=1,N
DO J=1,N
DO J=1,N

Block for calculating coefficients at interior point of N

CALL F(K+1,NN)
DO K=M+1,N

Normalization of the row

DO J=1,N
DO J=1,N
DO J=1,N
DO J=1,N

End of the cycle in N

DO J=1,N
DO J=1,N
DO J=1,N
DO J=1,N
CONTINUE
**END**

**Block for calculating coefficients at the right end**

```fortran
CALL F(K,K)
DO 32 I=1,N2
DY=ABS(Y(NN+1,I))+ABS(Y(NN,1,I))
*+ABS(Y(NN+1,I))+ABS(Y(NN,1,I))=EPS
IF(DY.EQ.0.,DY=EPS)
Y(NN,1,I)=Y(NN,1,I)+DY
CALL F(K,K)
Y(NN+1,1)=Y(NN+1,1)+DY
DO 32 K=1,N2
```

**Normalization of the row**

```fortran
DO 37 I=1,N2
CZ(I)=Z(I)
DO 38 K=1,N2
Z(K,I)=ABS(Y(K,I))
*+ABS(Y(K,I))=EPS
IF(DY.EQ.0.,DY=EPS)
Y(K,I)=Y(K,I)+DY
CALL F(K,K)
Y(K,1)=Y(K,1)+DY
DO 37 K=1,N2
```

**Block for calculating coefficients at the left end**

```fortran
CALL F(1,1)
DO 41 K=1,N2
DO 40 I=1,N2
DY=ABS(Y(K,I))
*+ABS(Y(K,I))=EPS
IF(DY.EQ.0.,DY=EPS)
Y(K,I)=Y(K,I)+DY
CALL F(1,1)
Y(K,1)=Y(K,1)+DY
DO 41 K=1,N2
```

**End of the procedure**
Program with Double Precision

{

DOUBLE PRECISION Y0, EPS, TKOH, TAU, XO, X1

DIMENSION Y0(21, 10)

ITEM = 0
IITER = 7
ITAU = 0
ITEPS = 0.04
TKOH = 0.1
EPS = 0.001

X0 = 0.5
X1 = 1
N = 21

Calculation of exact solution of control example at moment TKOH

\[ Y(U, V) = \exp(-0.8690 \times TKOH) \times \sin(3.14159 \times (X0 \times (X1 - X0)) \times (U - V)) \]

\[ Y(0, 0) = \exp(0) \times \sin(0) \]

\[ Y(N, 0) = \exp(-0.8690 \times TKOH) \times \sin(3.14159 \times (N \times (N - 1))) \]

PRINT 8

20H solutions at BP = TKOH

PRINT (32H Comparison of obtained and exact)

Subroutine FF

SUBROUTINE FF(N, NN)

DOUBLE PRECISION BP, H, TAU, XH, F, Y0, Y1

COMMON /H/I, BP, TAU, XH(21, 10), F(10, 0(21, 10), 1(21, 10)

H = M

Boundary conditions on the right end

\[ F(2) = y(x + h, 2) \]

GOTO 3

Boundary conditions on the left end

\[ F(1) = -5 \times y(1, 1) + 4 \times y(2, 1) - y(3, 1) \]

F(2) = -5 \times y(1, 2) + 4 \times y(2, 2) - y(3, 2)

GOTO 2

Right-hand side of the equation

\[ \text{CON} = 1 \]

\[ \text{CONT} = (y(N + 1, 1) + 2 \times y(N, 1) + y(N - 1, 1) \times H) \times H + y(N, 2) \times H \]

\[ \text{CONT} = (y(N + 1, 2) + 2 \times y(N, 2) + y(N - 1, 2) \times H) \times H \]

\[ \text{CONT} = (y(N + 1, 2) - 2 \times y(N, 2) + y(N - 1, 2) \times H) \times H \]

CONTINUE

RETURN

END
Subroutine DHOK

```
! Subroutine DHOK(N,M,IS,ITAU,ITER,ITB,EPS,TKOH,TAU,Y0,X0,X1)
! DOUBLF Precision Y00,EPS,TKOH,TAU,X0,X1
! DOUBLF Precision BPH,TAU1,XH,F,Y0,Y1
! DIMENSION Y00(21,10)
! COMMON //HIF/ BPH,TAU1,0
! FORMAT(/)
! X H ( 21)
! F 1 0 ) xy(2 11 ,1 0 ) ,Y ( 2 11 ,1 0 )
! PRINT /I
! 2 FORMAT(3H N=',13,3H M=',13,6H ITAU=',13,6H ITAU=',13
! PRINT N,M,ITAU,ITAU,ITB
! 3 FORMAT(5H EPS=',D13.4,6H TKOH=',D13.4,5H TAU=',D13.4)
! PRINT EPS,TKOH,TAU,X0,X1
! IF (TAU.EQ.0) TAUI1=/TAU
! DO 29 J=1,M
! 29 Y(J,J)=Y(J,J)
! CALL DOUT(M,N)
! IF (BP.EQ.0) GOTO 72
! LE TP.EQ.1) GOTO 55
! ! One compulsory iteration
! DO 30 J=1,M
! 30 CONTINUE
! ! Comparing the convergence of iterations
! DO 33 J=1,M
! IF (ABS(Y(J,J)).LE.10-32) GOTO 27
! CONTINUE
! ! Preparation for the next step
! DO 36 J=1,M
! 36 IF (TAU.EQ.0) GOTO 43
! IF (TB.EQ.1) GOTO 44
! ! Increase in the step by TAU
! IF (TB.EQ.1) GOTO 41
! IF (TAU.EQ.0) GOTO 43
! IF (TB.EQ.1) GOTO 44
! ! Decrease in the step by TAU
! IF (TB.EQ.1) GOTO 41
! IF (TB.EQ.0) GOTO 43
! IF (TB.EQ.1) GOTO 44
! ! Delivery of the result with period IB
! IB=0
! CALL DOUT(M,N)
! GOTO 30
! TAUI=TAUI-TAU
! GOTO 31
! TAU=1.5*TAU
```
TAU = TAU / 2
GO TO 42

FORMAT (34H Step TAU is smaller than the permitted time =.

STOP

IF (TAU.EQ.0) GO TO 52

DO 51 I = 1, M

DO 51 J = 1, M

Y(I, J) = YO(I, J)

GO TO 42

PRINT

FORMAT (39H IT = ITER Iterations do not converge with precision EPS)

PRINT 56, IT

CALL DOUT(M, N)

PRINT 56, IT

DO 53 J = 1, M

STOP

PRINT 56, IT

CONTINUE

CALL DOUT(M, N)

DO 59 J = 1, M

Y00(I, J) = Y1(I, J)

RETURN

END

Subroutine DSCL

SUBROUTINE DSCL(M, NN, EPS)

DOUBLE PRECISION B,RTAU1,XK, Y, Y1

DIMENSION B(20, 31), Z(20, 10), F(10)

COMMON YH, H, TAU1

XH(21), F(10), YO(21, 10), Y(21, 10), Y1(21, 10)

M2 = M

M21 = M + 1

M20 = 2 * M

M21 = 3 * M + 1

N = M - 3

Cycle in N

10 IF (N) 10, 10, 100

10 DO 11, 1, M, M20

11 DO 11 J = 1, M21

110 DO K = 1, M21

111 DO 12, L = 1, M

Z(K,L) = 0

C Search for a pivot element

YF = ABS(Z(K,L))

DABS(ABS(CZ(1))) GOTO 113

K = I

CZ(1) = Z(I,L)

113

C Pivot element is close to zero

IF YFABS(Z(K,L), LE. 1, D=50) GOTO 100

30
Rearrangement of the rows and elimination

BZC(K J) = BZ(K J)/CZ(I)

END

Formation of the matrices P, Q, R

NR(N.EQ.-1) = 0

DO 10 J = I, MZ

PZ(NI-1, IJ) = BZ(IJ, MZ)

IF(N.EQ.0) GOTO 110

QZ(NJ) = 0

CONTINUE

CONTINUE

IF(N.EQ.-1) GOTO 105

BM (M, J) = Z(I, MZ = 1, MZ),

DO 121 J = I, MZ

BZ(IJ) = BZ(I J, MZ = 1, MZ)

DO 120 J = MZ + 1, MZ

CONTINUE

END of the cycle in N

Receipt of the solution

105 DO 124 K = 2, N

DO 123 J = I, MZ

C(1) = C(1) * P(ZK).P(J) + C(J)

DO 122 K = 2, N

C(K) = C(K) * P(ZK).P(J) + C(J)

END of the cycle in N

Rearrangement of the rows and elimination
\begin{verbatim}
DO 56 K=MZ1,MZ2
BZ(K,K) = BZ(K,MZ3) + \sum_{j=1}^{M} Y_{0,n} MZ_1 \times MZ_2
56 Normalization of the row
DO 58 J=1,MZ0
IF(DABS(BZ(J,J),LT,DABS(CZ(J,J))) GOTO 58
CZ(J,J) = BZ(J,J)
58 CONTINUE
IF(CZ(1,1).EQ.0.) GOTO 100
DO 57 J=1,MZ2
BZ(J,J) = BZ(J,J)/CZ(J,J)
GOTO 100

Block for calculating coefficients at the right end
51 CALL FF(NN,NN)
DO 33 K=1,MZ
DY = DABS(Y(NN+2,1)) + DABS(Y(NN-1,1))
* DABS(Y(NN+1,1)) + EPS
IF(DY.EQ.0.) DY = EPS
Y(NN+1,1) = Y(NN-1,1) + DY
CALL FF(NN,NN)
Y(NN-1,1) = Y(NN-1,1) + DY
DO 31 J=1,MZ
DY = DABS(Y(NN+1,1)) + DABS(Y(NN-1,1))
* DABS(Y(NN+2,1)) + EPS
IF(DY.EQ.0.) DY = EPS
Y(NN-1,1) = Y(NN-1,1) + DY
CALL FF(NN,NN)
Y(NN-1,1) = Y(NN-1,1) + DY
DO 35 K=MZ1,MZ2
BZ(K,J) = BZ(K,MZ2) + BZ(K,1)*Y(NN-1,1)
1+ BZ(K,J) = BZ(K,J) + BZ(K,MZ3)*Y(NN-2,1)
35 Normalization of the row
DO 37 J=1,MZ2
CZ(J,J) = 0
DO 33 J=1,MZ2
IF(DABS(CZ(J,J),LT,DABS(CZ(J,J))) GOTO 33
CZ(J,J) = BZ(J,J)
33 CONTINUE
IF(CZ(1,1).EQ.0.) GOTO 100
DO 37 J=1,MZ2
BZ(J,J) = BZ(J,J)/CZ(J,J)
GOTO 100

Block for calculating coefficients at the left end
54 CALL FF(1,NN)
DO 41 K=1,MZ
DY = DABS(Y(3,1)) + DABS(Y(2,1))
* DABS(Y(3,1)) + EPS
IF(DY.EQ.0.) DY = EPS
Y(3,1) = Y(3,1) + DY
CALL FF(1,NN)
Y(3,1) = Y(3,1) + DY
DO 43 K=MZ1,MZ2
BZ(K,J) = BZ(K,MZ2) - BZ(K,MZ3)*Y(3,1)
43 Normalization of the row
\end{verbatim}
```
Elimination of A(0)

DO 60 I = MZ11, MZ20
IF (CZ(I).EQ.0.) GOTO 100
       CONTINUE
DO 61 J = 1, MZ
       BZ(J,MZ11) = BZ(J,I)*PZ(2K,I)*KZ(J,K)
       BZ(J,MZ20) = BZ(J,I)*QZ(1K,J)*KZ(J,K)
       CONTINUE
DO 62 J = 1, N
       BZ(I,J) = BZ(I,J)/CZ(I)
       CONTINUE
60      CONTINUE
61      CONTINUE
62      CONTINUE

Failure in Delivery

IF (CZ(I).EQ.0.) GOTO 100

100     PRINT ,,,.,R
       DO 3 J = 1, MZ
       PRINT 3, Y(J)
       PRINT 2, X(J)
       PRINT 1
       DO 10 J = 1, N
       PRINT 2, X(T(I,J)), Y(J), J = J+1
       PRINT 1
       END

Subroutine DOUT

SUBROUTINE DOUT(N,N)
DOUBLE PRECISION BP,H,TAU1,XH,F,Y0,Y1
COMMON BP,H,TAU1,XH,F,Y0,Y1
1 FORMAT (/,23H Matrix is singular: N =,I3, N =)
2 FORMAT (10D12.4)
STOP
END

Printed at the ICR of the AS of the USSR
Sent to press 3/20/78
33
```