NAG Toolbox for MATLAB

d03ec

1 Purpose

d03ec uses the Strongly Implicit Procedure to calculate the solution to a system of simultaneous algebraic equations of seven-point molecule form on a three-dimensional topologically-rectangular mesh. ('Topological' means that a polar grid, for example, can be used if it is equivalent to a rectangular box.)

2 Syntax

```
[t, itcoun, itused, resids, chngs, ifail] = d03ec(n1, n2, n3, a, b, c,
d, e, f, g, q, t, aparam, itmax, itcoun, ndir, ixn, iyn, izn, conres,
conchn, 'sda', sda)
```

3 Description

Given a set of simultaneous equations

$$Mt = q \tag{1}$$

(which could be nonlinear) derived, for example, from a finite difference representation of a three-dimensional elliptic partial differential equation and its boundary conditions, the function determines the values of the dependent variable t. M is a square $(n_1 \times n_2 \times n_3)$ by $(n_1 \times n_2 \times n_3)$ matrix and q is a known vector of length $(n_1 \times n_2 \times n_3)$.

The equations must be of seven-diagonal form:

$$a_{ijk}t_{ij,k-1} + b_{ijk}t_{i,j-1,k} + c_{ijk}t_{i-1,jk} + d_{ijk}t_{ijk} + e_{ijk}t_{i+1,jk} + f_{ijk}t_{i,j+1,k} + g_{ijk}t_{ij,k+1} = q_{ijk}$$

for $i = 1, 2, \ldots, n_1$; $j = 1, 2, \ldots, n_2$ and $k = 1, 2, \ldots, n_3$, provided that $d_{ijk} \neq 0.0$.

Indeed, if $d_{ijk} = 0.0$, then the equation is assumed to be:

$$t_{ijk} = q_{ijk}.$$

The system is solved iteratively from a starting approximation $t^{(1)}$ by the formulae:

$$\begin{aligned} r^{(n)} &= q - Mt^{(n)} \\ Ms^{(n)} &= r^{(n)} \\ t^{(n+1)} &= t^{(n)} + s^{(n)}. \end{aligned}$$

Thus $r^{(n)}$ is the residual of the nth approximate solution $t^{(n)}$, and $s^{(n)}$ is the up-date change vector.

The calling program supplies an initial approximation for the values of the dependent variable in the array **t**, the coefficients of the seven-point molecule system of equations in the arrays **a**, **b**, **c**, **d**, **e**, **f** and **g**, and the source terms in the array **q**. The function derives the residual of the latest approximate solution, and then uses the approximate LU factorization of the Strongly Implicit Procedure with the necessary acceleration parameter adjustment by calling d03ub at each iteration. d03ec combines the newly derived change with the old approximation to obtain the new approximate solution for t. The new solution is checked for convergence against the user-supplied convergence criteria, and if these have not been satisfied, the iterative cycle is repeated. Convergence is based on both the maximum absolute normalized residuals (calculated with reference to the previous approximate solution as these are calculated at the commencement of each iteration) and on the maximum absolute change made to the values of t.

Problems in topologically non-rectangular-box-shaped regions can be solved using the function by surrounding the region by a circumscribing topologically rectangular box. The equations for the nodal values external to the region of interest are set to zero (i.e., $d_{ijk} = t_{ijk} = 0$) and the boundary conditions are incorporated into the equations for the appropriate nodes.

If there is no better initial approximation when starting the iterative cycle, one can use an array of zeros as the initial approximation.

The function can be used to solve linear elliptic equations in which case the arrays **a**, **b**, **c**, **d**, **e**, **f**, **g** and **q** remain constant and for which a single call provides the required solution. It can also be used to solve nonlinear elliptic equations, in which case some or all of these arrays may require updating during the progress of the iterations as more accurate solutions are derived. The function will then have to be called repeatedly in an outer iterative cycle. Dependent on the nonlinearity, some under-relaxation of the coefficients and/or source terms may be needed during their recalculation using the new estimates of the solution.

The function can also be used to solve each step of a time-dependent parabolic equation in three space dimensions. The solution at each time step can be expressed in terms of an elliptic equation if the Crank–Nicolson or other form of implicit time integration is used.

Neither diagonal dominance, nor positive-definiteness, of the matrix M formed from the arrays **a**, **b**, **c**, **d**, **e**, **f**, **g** is necessary to ensure convergence.

For problems in which the solution is not unique in the sense that an arbitrary constant can be added to the solution (for example Poisson's equation with all Neumann boundary conditions), a parameter is incorporated so that the solution can be rescaled. A specified nodal value is subtracted from the whole solution t after the completion of every iteration. This keeps rounding errors to a minimum for those cases when convergence is slow. For such problems there is generally an associated compatibility condition. For the example mentioned this compatibility condition equates the total net source within the region (i.e., the source integrated over the region) with the total net outflow across the boundaries defined by the Neumann conditions (i.e., the normal derivative integrated along the whole boundary). It is very important that the algebraic equations derived to model such a problem implement accurately the compatibility condition. If they do not, a net source or sink is very likely to be represented by the set of algebraic equations and no steady-state solution of the equations exists.

4 References

Jacobs D A H 1972 The strongly implicit procedure for the numerical solution of parabolic and elliptic partial differential equations *Note RD/L/N66/72* Central Electricity Research Laboratory

Stone H L 1968 Iterative solution of implicit approximations of multi-dimensional partial differential equations *SIAM J. Numer. Anal.* **5** 530–558

Weinstein H G, Stone H L and Kwan T V 1969 Iterative procedure for solution of systems of parabolic and elliptic equations in three dimensions *Industrial and Engineering Chemistry Fundamentals* **8** 281–287

5 Parameters

5.1 Compulsory Input Parameters

1: **n1 – int32 scalar**

The number of nodes in the first co-ordinate direction, n_1.

Constraint: **n1** > 1.

2: **n2 – int32 scalar**

the number of nodes in the second co-ordinate direction, n_2.

Constraint: **n2** > 1.

3: **n3 – int32 scalar**

the number of nodes in the third co-ordinate direction, n_3.

Constraint: **n3** > 1.

4: **a(lda,sda,n3) – double array**

lda, the first dimension of the array, must be at least **n1**.

$\mathbf{a}(i,j,k)$ must contain the coefficient of $t_{ij,k-1}$ in the (i,j,k)th equation of the system (1), for $i = 1,2,\ldots,\mathbf{n1}$ and $j = 1,2,\ldots,\mathbf{n2}$ and $k = 1,2,\ldots,\mathbf{n3}$. The elements of **a** for $k = 1$ must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

5: **b(lda,sda,n3) – double array**

lda, the first dimension of the array, must be at least **n1**.

$\mathbf{b}(i,j,k)$ must contain the coefficient of $t_{i,j-1,k}$ in the (i,j,k)th equation of the system (1), for $i = 1,2,\ldots,\mathbf{n1}$ and $j = 1,2,\ldots,\mathbf{n2}$ and $k = 1,2,\ldots,\mathbf{n3}$. The elements of **b** for $j = 1$ must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

6: **c(lda,sda,n3) – double array**

lda, the first dimension of the array, must be at least **n1**.

$\mathbf{c}(i,j,k)$ must contain the coefficient of $t_{i-1,jk}$ in the (i,j,k)th equation of the system (1), for $i = 1,2,\ldots,\mathbf{n1}$ and $j = 1,2,\ldots,\mathbf{n2}$ and $k = 1,2,\ldots,\mathbf{n3}$. The elements of **c** for $i = 1$ must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

7: **d(lda,sda,n3) – double array**

lda, the first dimension of the array, must be at least **n1**.

$\mathbf{d}(i,j,k)$ must contain the coefficient of t_{ijk} (the 'central' term) in the (i,j,k)th equation of the system (1), for $i = 1,2,\ldots,\mathbf{n1}$ and $j = 1,2,\ldots,\mathbf{n2}$ and $k = 1,2,\ldots,\mathbf{n3}$. The elements of **d** are checked to ensure that they are nonzero. If any element is found to be zero, the corresponding algebraic equation is assumed to be $t_{ijk} = q_{ijk}$. This feature can be used to define the equations for nodes at which, for example, Dirichlet boundary conditions are applied, or for nodes external to the problem of interest. Setting $\mathbf{d}(i,j,k) = 0.0$ at appropriate points, and the corresponding value of $\mathbf{q}(i,j,k)$ to the appropriate value, namely the prescribed value of $\mathbf{t}(i,j,k)$ in the Dirichlet case, or to zero at an external point.

8: **e(lda,sda,n3) – double array**

lda, the first dimension of the array, must be at least **n1**.

$\mathbf{e}(i,j,k)$ must contain the coefficient of $t_{i+1,jk}$ in the (i,j,k)th equation of the system (1), for $i = 1,2,\ldots,\mathbf{n1}$ and $j = 1,2,\ldots,\mathbf{n2}$ and $k = 1,2,\ldots,\mathbf{n3}$. The elements of **e** for $i = \mathbf{n1}$ must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

9: **f(lda,sda,n3) – double array**

lda, the first dimension of the array, must be at least **n1**.

$\mathbf{f}(i,j,k)$ must contain the coefficient of $t_{i,j+1,k}$ in the (i,j,k)th equation of the system (1), for $i = 1,2,\ldots,\mathbf{n1}$ and $j = 1,2,\ldots,\mathbf{n2}$ and $k = 1,2,\ldots,\mathbf{n3}$. The elements of **f** for $j = \mathbf{n2}$ must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

10: **g(lda,sda,n3) – double array**

lda, the first dimension of the array, must be at least **n1**.

$\mathbf{g}(i,j,k)$ must contain the coefficient of $t_{ij,k+1}$ in the (i,j,k)th equation of the system (1), for $i = 1,2,\ldots,\mathbf{n1}$ and $j = 1,2,\ldots,\mathbf{n2}$ and $k = 1,2,\ldots,\mathbf{n3}$. The elements of **g** for $k = \mathbf{n3}$ must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

11: **q(lda,sda,n3) – double array**

lda, the first dimension of the array, must be at least **n1**.

$\mathbf{q}(i,j,k)$ must contain q_{ijk}, for $i = 1,2,\ldots,\mathbf{n1}$ and $j = 1,2,\ldots,\mathbf{n2}$ and $k = 1,2,\ldots,\mathbf{n3}$, i.e., the source-term values at the nodal points of the system (1).

12: **t(lda,sda,n3) – double array**

lda, the first dimension of the array, must be at least **n1**.

$\mathbf{t}(i,j,k)$ must contain the element t_{ijk} of an approximate solution to the equations for $i = 1,2,\ldots,\mathbf{n1}$, $j = 1,2,\ldots,\mathbf{n2}$ and $k = 1,2,\ldots,\mathbf{n3}$.

If no better approximation is known, an array of zeros can be used.

13: **aparam – double scalar**

The iteration acceleration factor. A value of 1.0 is adequate for most typical problems. However, if convergence is slow, the value can be reduced, typically to 0.2 or 0.1. If divergence is obtained, the value can be increased, typically to 2.0, 5.0 or 10.0.

Constraint: $0.0 < \mathbf{aparam} \leq \left((\mathbf{n1}-1)^2 + (\mathbf{n2}-1)^2 + (\mathbf{n3}-1)^2\right)/3.0$.

14: **itmax – int32 scalar**

the maximum number of iterations to be used by the function in seeking the solution. A reasonable value might be 20 for a problem with 3000 nodes and convergence criteria of about 10^{-3} of the original residual and change.

15: **itcoun – int32 scalar**

On the first call of d03ec, **itcoun** must be set to 0. On subsequent entries, its value must be unchanged from the previous call.

16: **ndir – int32 scalar**

Indicates whether or not the system of equations has a unique solution. For systems which have a unique solution, **ndir** must be set to any nonzero value. For systems derived from problems to which an arbitrary constant can be added to the solution, for example Poisson's equation with all Neumann boundary conditions, **ndir** should be set to 0 and the values of the next three parameters must be specified. For such problems the function subtracts the value of the function derived at the node (**ixn**, **iyn**, **izn**) from the whole solution after each iteration to reduce the possibility of large rounding errors. You must also ensure for such problems that the appropriate compatibility condition on the source terms **q** is satisfied. See the comments at the end of Section 3.

17: **ixn – int32 scalar**

Is ignored unless **ndir** is equal to zero, in which case it must specify the first index of the nodal point at which the solution is to be set to zero. The node should not correspond to a corner node, or to a node external to the region of interest.

18: **iyn – int32 scalar**

Is ignored unless **ndir** is equal to zero, in which case it must specify the second index of the nodal point at which the solution is to be set to zero. The node should not correspond to a corner node, or to a node external to the region of interest.

19: **izn – int32 scalar**

Is ignored unless **ndir** is equal to zero, in which case it must specify the third index of the nodal point at which the solution is to be set to zero. The node should not correspond to a corner node, or to a node external to the region of interest.

20: **conres – double scalar**

The convergence criterion to be used on the maximum absolute value of the normalized residual vector components. The latter is defined as the residual of the algebraic equation divided by the central coefficient when the latter is not equal to 0.0, and defined as the residual when the central coefficient is zero.

conres should not be less than a reasonable multiple of the ***machine precision***.

21: **conchn – double scalar**

The convergence criterion to be used on the maximum absolute value of the change made at each iteration to the elements of the array **t**, namely the dependent variable. **conchn** should not be less than a reasonable multiple of the machine accuracy multiplied by the maximum value of **t** attained.

Convergence is achieved when both the convergence criteria are satisfied. You can therefore set convergence on either the residual or on the change, or (as is recommended) on a requirement that both are below prescribed limits.

5.2 Optional Input Parameters

1: **sda – int32 scalar**

Default: The second dimension of the arrays **a**, **b**, **c**, **d**, **e**, **f**, **g**, **q**, **t**. (An error is raised if these dimensions are not equal.)

Constraint: **sda** $\geq$ **n2**.

5.3 Input Parameters Omitted from the MATLAB Interface

lda, wrksp1, wrksp2, wrksp3, wrksp4

5.4 Output Parameters

1: **t**(**lda,sda,n3**) **– double array**

The solution derived by the function.

2: **itcoun – int32 scalar**

Its value is increased by the number of iterations used on this call (namely **itused**). It therefore stores the accumulated number of iterations actually used.

For subsequent calls for the same problem, i.e., with the same **n1**, **n2** and **n3** but possibly different coefficients and/or source terms, as occur with nonlinear systems or with time-dependent systems, **itcoun** should not be reset, i.e., it must contain the accumulated number of iterations. In this way a suitable cycling of the sequence of iteration parameters is obtained in the calls to d03ub.

3: **itused – int32 scalar**

The number of iterations actually used on that call.

4: **resids**(**itmax**) **– double array**

The maximum absolute value of the residuals calculated at the ith iteration, for $i = 1, 2, \ldots,$ **itused**. If the residual of the solution is sought you must calculate this in the (sub)program from which d03ec is called. The sequence of values **resids** indicates the rate of convergence.

5: **chngs**(**itmax**) **– double array**

The maximum absolute value of the changes made to the components of the dependent variable **t** at the ith iteration, for $i = 1, 2, \ldots,$ **itused**. The sequence of values **chngs** indicates the rate of convergence.

6: **ifail – int32 scalar**

0 unless the function detects an error (see Section 6).

6 Error Indicators and Warnings

Note: d03ec may return useful information for one or more of the following detected errors or warnings.

ifail $= 1$

On entry, **n1** < 2,
or **n2** < 2,
or **n3** < 2.

ifail $= 2$

On entry, **lda** $<$ **n1**,
or **sda** $<$ **n2**.

ifail $= 3$

On entry, **aparam** ≤ 0.0.

ifail $= 4$

On entry, **aparam** $> \left((\mathbf{n1}-1)^2+(\mathbf{n2}-1)^2+(\mathbf{n3}-1)^2\right)/3.0$.

ifail $= 5$

Convergence was not achieved after **itmax** iterations.

7 Accuracy

The improvement in accuracy for each iteration depends on the size of the system and on the condition of the up-date matrix characterised by the seven-diagonal coefficient arrays. The ultimate accuracy obtainable depends on the above factors and on the ***machine precision***. The rate of convergence obtained with the Strongly Implicit Procedure is not always smooth because of the cyclic use of nine acceleration parameters. The convergence may become slow with very large problems. The final accuracy obtained may be judged approximately from the rate of convergence determined from the sequence of values returned in the arrays **resids** and **chngs** and the magnitude of the maximum absolute value of the change vector on the last iteration stored in **chngs**(**itused**).

8 Further Comments

The time taken per iteration is approximately proportional to **n1** $\times$ **n2** $\times$ **n3**.

Convergence may not always be obtained when the problem is very large and/or the coefficients of the equations have widely disparate values. The latter case is often associated with a near ill-conditioned matrix.

9 Example

```
x = [0; 1; 3; 6];
y = [0; 1; 3; 6; 10];
z = [0; 1; 3; 6; 10; 15];
n1 = int32(4);
n2 = int32(5);
n3 = int32(6);
a = zeros(4, 5, 6);
b = zeros(4, 5, 6);
```

```
c = zeros(4, 5, 6);
d = zeros(4, 5, 6);
e = zeros(4, 5, 6);
f = zeros(4, 5, 6);
g = zeros(4, 5, 6);
q = zeros(4, 5, 6);
t = zeros(4, 5, 6);
aparam = 1;
itmax = int32(18);
itcoun = int32(0);
ndir = int32(1);
ixn = int32(0);
iyn = int32(0);
izn = int32(0);
conres = 1e-06;
conchn = 1e-06;
% Set up difference equation coefficients, source terms and
% initial approximation
for k = 1:n3
  for j = 1:n2
    for i = 1:n1
      if (i ~= 1 && i ~= n1 && j ~= 1 && j ~= n2 && k ~= 1 && k ~= n3)
        % Specification for internal nodes
        a(i,j,k) = 2/((z(k)-z(k-1))*(z(k+1)-z(k-1)));
        g(i,j,k) = 2/((z(k+1)-z(k))*(z(k+1)-z(k-1)));
        b(i,j,k) = 2/((y(j)-y(j-1))*(y(j+1)-y(j-1)));
        f(i,j,k) = 2/((y(j+1)-y(j))*(y(j+1)-y(j-1)));
        c(i,j,k) = 2/((x(i)-x(i-1))*(x(i+1)-x(i-1)));
        e(i,j,k) = 2/((x(i+1)-x(i))*(x(i+1)-x(i-1)));
              d(i,j,k)  =  -a(i,j,k)-b(i,j,k)-c(i,j,k)-e(i,j,k)-f(i,j,k)-
g(i,j,k);
      else
        % Specification for boundary nodes
           q(i,j,k)  =  exp((x(i)+1)/y(n2))*cos(sqrt(2)*y(j)/y(n2))*exp((-
z(k)-1)/y(n2));
      end
    end
  end
end

[tOut, itcounOut, itused, resids, chngs, ifail] = ...
     d03ec(n1, n2, n3, a, b, c, d, e, f, g, q, t, aparam, itmax, itcoun,
...
    ndir, ixn, iyn, izn, conres, conchn)
```

```
tOut =
(:,:,1) =
    1.0000    0.9900    0.9113    0.6611    0.1559
    1.1052    1.0941    1.0072    0.7306    0.1723
    1.3499    1.3364    1.2302    0.8924    0.2105
    1.8221    1.8039    1.6606    1.2046    0.2841
(:,:,2) =
    0.9048    0.8958    0.8246    0.5982    0.1411
    1.0000    0.9903    0.9120    0.6621    0.1559
    1.2214    1.2097    1.1142    0.8091    0.1905
    1.6487    1.6323    1.5025    1.0900    0.2571
(:,:,3) =
    0.7408    0.7334    0.6751    0.4897    0.1155
    0.8187    0.8110    0.7472    0.5427    0.1277
    1.0000    0.9908    0.9132    0.6637    0.1559
    1.3499    1.3364    1.2302    0.8924    0.2105
(:,:,4) =
    0.5488    0.5433    0.5002    0.3628    0.0856
    0.6065    0.6009    0.5537    0.4023    0.0946
    0.7408    0.7341    0.6769    0.4921    0.1155
    1.0000    0.9900    0.9113    0.6611    0.1559
(:,:,5) =
    0.3679    0.3642    0.3353    0.2432    0.0574
    0.4066    0.4028    0.3712    0.2697    0.0634
    0.4966    0.4921    0.4538    0.3299    0.0774
```

```
    0.6703    0.6636    0.6109    0.4431    0.1045
(:,:,6) =
    0.2231    0.2209    0.2033    0.1475    0.0348
    0.2466    0.2441    0.2247    0.1630    0.0385
    0.3012    0.2982    0.2745    0.1991    0.0470
    0.4066    0.4025    0.3705    0.2688    0.0634
itcounOut =
           6
itused =
           6
resids =
    1.8221
    0.0090
    0.0014
    0.0000
    0.0000
    0.0000
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
chngs =
    1.8221
    0.0197
    0.0015
    0.0000
    0.0000
    0.0000
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
ifail =
           0
```