NAG Toolbox for MATLAB

d03eb

1 Purpose

d03eb uses the Strongly Implicit Procedure to calculate the solution to a system of simultaneous algebraic equations of five-point molecule form on a two-dimensional topologically-rectangular mesh. ('Topological' means that a polar grid, for example (r, θ), can be used, being equivalent to a rectangular box.)

2 Syntax

```
[t, itcoun, itused, resids, chngs, ifail] = d03eb(n1, a, b, c, d, e, q,
t, aparam, itmax, itcoun, ndir, ixn, iyn, conres, conchn, 'n2', n2)
```

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 two-dimensional elliptic partial differential equation and its boundary conditions, the function determines the values of the dependent variable t. q is a known vector of length $n_1 \times n_2$ and M is a square $(n_1 \times n_2)$ by $(n_1 \times n_2)$ matrix.

The equations must be of five-diagonal form:

$$a_{ij}t_{i,j-1} + b_{ij}t_{i-1,j} + c_{ij}t_{ij} + d_{ij}t_{i+1,j} + e_{ij}t_{i,j+1} = q_{ij}$$

for $i = 1, 2, \ldots, n_1$; $j = 1, 2, \ldots, n_2$, provided $c_{ij} \neq 0.0$. Indeed, if $c_{ij} = 0.0$, then the equation is assumed to be

$$t_{ij} = q_{ij}.$$

For example, if $n_1 = 3$ and $n_2 = 2$, the equations take the form:

$$\begin{bmatrix} c_{11} & d_{11} & & e_{11} & & \\ b_{21} & c_{21} & d_{21} & & e_{21} & \\ & b_{31} & c_{31} & & & e_{31} \\ a_{12} & & & c_{12} & d_{12} & \\ & a_{22} & & b_{22} & c_{22} & d_{22} \\ & & a_{32} & & b_{32} & c_{32} \end{bmatrix} \begin{bmatrix} t_{11} \\ t_{21} \\ t_{31} \\ t_{12} \\ t_{22} \\ t_{32} \end{bmatrix} = \begin{bmatrix} q_{11} \\ q_{21} \\ q_{31} \\ q_{12} \\ q_{22} \\ q_{32} \end{bmatrix}.$$

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 five-point molecule system of equations in the arrays **a**, **b**, **c**, **d** and **e**, 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 d03ua at each iteration. d03eb 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 achieved 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 regions can be solved using the function by surrounding the region by a circumscribing topological rectangle. The equations for the nodal values external to the region of interest are set to zero (i.e., $c_{ij} = t_{ij} = 0$) 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, an array of all zeros can be used as the initial approximation.

The function can be used to solve linear elliptic equations in which case the arrays **a**, **b**, **c**, **d**, **e** and **q** are constants 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 two 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** 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 Laplace's equation with all Neumann boundary conditions, a parameter is incorporated so that the solution can be rescaled by subtracting a specified nodal value from the whole solution t after the completion of every iteration to keep rounding errors to a minimum for those cases when the convergence is slow.

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

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: **a(lda,n2) – double array**

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

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

3: **b(lda,n2) – double array**

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

b(i,j) must contain the coefficient of the 'westerly' term involving $t_{i-1,j}$ in the (i,j)th equation of the system (1), for $i = 1,2,\ldots,$ **n1** and $j = 1,2,\ldots,$ **n2**. The elements of **b** for $i = 1$ must be zero

after incorporating the boundary conditions, since they involve nodal values from outside the rectangle.

4: **c(lda,n2) – double array**

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

c(i,j) must contain the coefficient of the 'central' term involving t_{ij} in the (i,j)th equation of the system (1), for $i = 1,2,\ldots,$ **n1** and $j = 1,2,\ldots,$ **n2**. The elements of **c** are checked to ensure that they are nonzero. If any element is found to be zero, the corresponding algebraic equation is assumed to be $t_{ij} = q_{ij}$. 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, by setting **c**$(i,j) = 0.0$ at appropriate points, and the corresponding value of **q**(i,j) to the appropriate value, namely the prescribed value of **t**(i,j) in the Dirichlet case or zero at an external point.

5: **d(lda,n2) – double array**

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

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

6: **e(lda,n2) – double array**

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

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

7: **q(lda,n2) – double array**

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

q(i,j) must contain q_{ij}, for $i = 1,2,\ldots,$ **n1** and $j = 1,2,\ldots,$ **n2**, i.e., the source term values at the nodal points for the system (1).

8: **t(lda,n2) – double array**

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

t(i,j) must contain the element t_{ij} of the approximate solution to the equations supplied by the calling program as an initial starting value, for $i = 1,2,\ldots,$ **n1** and $j = 1,2,\ldots,$ **n2**.

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

9: **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: $0.0 <$ **aparam** $\leq \left((\mathbf{n1}-1)^2 + (\mathbf{n2}-1)^2\right)/2.0$.

10: **itmax – int32 scalar**

the maximum number of iterations to be used by the function in seeking the solution. A reasonable value might be 30 if **n1** $=$ **n2** $= 10$ or 100 if **n1** $=$ **n2** $= 50$.

11: **itcoun – int32 scalar**

On the first call of d03eb, **itcoun** must be set to 0. On subsequent entries, its value must be unchanged from the previous call.

12: **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, for example, Laplace's equation with all Neumann boundary conditions, i.e., problems in which an arbitrary constant can be added to the solution, **ndir** should be set to 0 and the values of the next two parameters must be specified. For such problems the function subtracts the value of the function derived at the node (**ixn**, **iyn**) from the whole solution after each iteration to reduce the possibility of large rounding errors. You must also ensure that for such problems the appropriate consistency condition on the source terms **q** is satisfied.

13: **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.

14: **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.

15: **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.

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

16: **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. Clearly **conchn** should not be less than a reasonable multiple of the ***machine precision*** 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: **n2 – int32 scalar**

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

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

Constraint: **n2** > 1.

5.3 Input Parameters Omitted from the MATLAB Interface

lda, wrksp1, wrksp2, wrksp3

5.4 Output Parameters

1: **t(lda,n2) – 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** and **n2** but possibly different coefficients and/or source terms, as occur with nonlinear systems or with time-dependent systems, **itcoun** is the accumulated number of iterations. This applies to the second and subsequent calls to d03eb. In this way a suitable cycling of the sequence of iteration parameters is obtained in the calls to d03ua.

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 you want to know the maximum absolute residual of the solution which is returned you must calculate this in the calling program. 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

Errors or warnings detected by the function:

ifail $= 1$

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

ifail $= 2$

On entry, **lda** $<$ **n1**.

ifail $= 3$

On entry, **aparam** ≤ 0.0.

ifail $= 4$

On entry, **aparam** $> \left((\mathbf{n1}-1)^2 + (\mathbf{n2}-1)^2\right)/2.0$.

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 five-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, for example when $\mathbf{n1} = \mathbf{n2} = 60$. The final accuracy may be judged approximately from the rate of convergence determined from the sequence of values returned in **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 $\mathbf{n1} \times \mathbf{n2}$.

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

```
n1 = int32(6);
a = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0;
                  0,   0.6666666666666666,   0.2,   0.09523809523809523,
0.05555555555555555, ...
    0.03636363636363636, 0.02564102564102564, 0.01904761904761905, ...
    0.01470588235294118, 0;
                  0,   0.6666666666666666,   0.2,   0.09523809523809523,
0.05555555555555555, ...
    0.03636363636363636, 0.02564102564102564, 0.01904761904761905, ...
    0.01470588235294118, 0;
                  0,   0.6666666666666666,   0.2,   0.09523809523809523,
0.05555555555555555, ...
    0.03636363636363636, 0.02564102564102564, 0.01904761904761905, ...
    0.01470588235294118, 0;
                  0,   0.6666666666666666,   0.2,   0.09523809523809523,
0.05555555555555555, ...
    0.03636363636363636, 0.02564102564102564, 0.01904761904761905, ...
    0.01470588235294118, 0;
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
b = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0;
     0, 0.6666666666666666, 0.6666666666666666, 0.6666666666666666, ...
    0.6666666666666666, 0.6666666666666666, 0.6666666666666666, ...
    0.6666666666666666, 0.6666666666666666, 0;
     0, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0;
       0,  0.09523809523809523,  0.09523809523809523,  0.09523809523809523,
...
    0.09523809523809523, 0.09523809523809523, 0.09523809523809523, ...
    0.09523809523809523, 0.09523809523809523, 0;
       0,  0.05555555555555555,  0.05555555555555555,  0.05555555555555555,
...
    0.05555555555555555, 0.05555555555555555, 0.05555555555555555, ...
    0.05555555555555555, 0.05555555555555555, 0;
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
c = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0;
            0,  -2,  -1.333333333333333,  -1.166666666666667,  -1.1,  -
1.066666666666667, ...
    -1.047619047619048, -1.035714285714286, -1.027777777777778, 0;
               0,   -1.333333333333333,   -0.6666666666666667,   -0.5,   -
0.4333333333333333, ...
      -0.4, -0.380952380952381, -0.3690476190476191, -0.3611111111111111,
0;
               0,   -1.166666666666667,   -0.5,   -0.3333333333333333,   -
0.2666666666666667, ...
```

```
    -0.2333333333333333, -0.2142857142857143, -0.2023809523809524, ...
    -0.1944444444444444, 0;
     0, -1.1, -0.4333333333333333, -0.2666666666666667, -0.2, ...
    -0.1666666666666667, -0.1476190476190476, -0.1357142857142857, ...
    -0.1277777777777778, 0;
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
d = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0;
     0, 0.3333333333333333, 0.3333333333333333, 0.3333333333333333, ...
    0.3333333333333333, 0.3333333333333333, 0.3333333333333333, ...
    0.3333333333333333, 0.3333333333333333, 0;
     0, 0.1333333333333333, 0.1333333333333333, 0.1333333333333333, ...
    0.1333333333333333, 0.1333333333333333, 0.1333333333333333, ...
    0.1333333333333333, 0.1333333333333333, 0;
       0, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142,
...
    0.07142857142857142, 0.07142857142857142, 0.07142857142857142, ...
    0.07142857142857142, 0.07142857142857142, 0;
       0, 0.04444444444444445, 0.04444444444444445, 0.04444444444444445,
...
    0.04444444444444445, 0.04444444444444445, 0.04444444444444445, ...
    0.04444444444444445, 0.04444444444444445, 0;
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
e = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0;
     0, 0.3333333333333333, 0.1333333333333333, 0.07142857142857142, ...
    0.04444444444444445, 0.0303030303030303, 0.02197802197802198, ...
    0.01666666666666667, 0.0130718954248366, 0;
     0, 0.3333333333333333, 0.1333333333333333, 0.07142857142857142, ...
    0.04444444444444445, 0.0303030303030303, 0.02197802197802198, ...
    0.01666666666666667, 0.0130718954248366, 0;
     0, 0.3333333333333333, 0.1333333333333333, 0.07142857142857142, ...
    0.04444444444444445, 0.0303030303030303, 0.02197802197802198, ...
    0.01666666666666667, 0.0130718954248366, 0;
     0, 0.3333333333333333, 0.1333333333333333, 0.07142857142857142, ...
    0.04444444444444445, 0.0303030303030303, 0.02197802197802198, ...
    0.01666666666666667, 0.0130718954248366, 0;
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
q = [1.022470974998333, 1.022218523418408, 1.020199658691349, ...
    1.013395800771301, 0.9973285010313558, 0.9661910502298774, ...
    0.9131411732014387, 0.8308448419408389, 0.7123623883940765, ...
    0.5524434254748445;
     1.045446894714042, 0, 0, 0, 0, 0, 0, 0, 0, 0.5648573678766832;
     1.092959209667233, 0, 0, 0, 0, 0, 0, 0, 0, 0.5905283812030258;
     1.168306840199909, 0, 0, 0, 0, 0, 0, 0, 0, 0.6312388797215314;
     1.276911720713716, 0, 0, 0, 0, 0, 0, 0, 0, 0.6899183470916745;
     1.426973196997562, 1.426620872435886, 1.423803319738234, ...
    1.414307771086494, 1.391884047931845, 1.348428234707795, ...
    1.274391167177617, 1.159537384731323, 0.9941818004067758, ...
    0.770996908749814];
t = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0;
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0;
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0;
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0;
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0;
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
aparam = 1;
itmax = int32(18);
itcoun = int32(0);
ndir = int32(1);
ixn = int32(8185080);
iyn = int32(-1076188144);
conres = 1e-06;
conchn = 1e-06;
[tOut, itcounOut, itused, resids, chngs, ifail] = ...
      d03eb(n1, a, b, c, d, e, q, t, aparam, itmax, itcoun, ndir, ixn,
iyn, conres, conchn)
```

```
tOut =
  Columns 1 through 7
    1.0225    1.0222    1.0202    1.0134    0.9973    0.9662    0.9131
    1.0454    1.0452    1.0431    1.0362    1.0198    0.9879    0.9337
```

```
    1.0930    1.0927    1.0906    1.0833    1.0662    1.0329    0.9762
    1.1683    1.1680    1.1658    1.1580    1.1397    1.1042    1.0436
    1.2769    1.2766    1.2741    1.2657    1.2457    1.2068    1.1406
    1.4270    1.4266    1.4238    1.4143    1.3919    1.3484    1.2744
  Columns 8 through 10
    0.8308    0.7124    0.5524
    0.8496    0.7284    0.5649
    0.8882    0.7616    0.5905
    0.9495    0.8141    0.6312
    1.0378    0.8898    0.6899
    1.1595    0.9942    0.7710
itcounOut =
           7
itused =
           7
resids =
    1.4270
    0.0067
    0.0008
    0.0001
    0.0000
    0.0000
    0.0000
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
chngs =
    1.4270
    0.0218
    0.0016
    0.0002
    0.0000
    0.0000
    0.0000
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
ifail =
           0
```