NAG Toolbox for MATLAB

d03ub

1 Purpose

d03ub performs at each call one iteration of the Strongly Implicit Procedure. It is used to calculate on successive calls a sequence of approximate corrections to the current estimate of the solution when solving a system of simultaneous algebraic equations for which the iterative up-date matrix is of seven-point molecule form on a three-dimensional topologically-rectangular mesh. ('Topological' means that a polar grid (r, θ), for example, can be used as it is equivalent to a rectangular box.)

2 Syntax

```
[r, ifail] = d03ub(n1, n2, n3, a, b, c, d, e, f, g, aparam, it, r,
'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 solution t may be obtained iteratively from a starting approximation $t^{(1)}$ by the formulae

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

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

d03ub determines the approximate change vector s corresponding to a given residual r, i.e., it determines an approximate solution to a set of equations

$$Ms = r \tag{2}$$

where M is a square $(n_1 \times n_2 \times n_3)$ by $(n_1 \times n_2 \times n_3)$ matrix and r is a known vector of length $(n_1 \times n_2 \times n_3)$. The set of equations (2) must be of seven-diagonal form

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

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

$$s_{ijk} = r_{ijk}.$$

The calling program supplies the current residual r at each iteration and the coefficients of the seven-point molecule system of equations on which the up-date procedure is based. The function performs one iteration, using the approximate LU factorization of the Strongly Implicit Procedure with the necessary acceleration parameter adjustment, to calculate the approximate solution s of the set of equations (2). The change s overwrites the residual array for return to the calling program. The calling program must combine this change stored in r with the old approximation to obtain the new approximate solution for t. It must then recalculate the residuals and, if the accuracy requirements have not been satisfied, commence the next iterative cycle.

Clearly there is no requirement that the iterative up-date matrix passed in the form of the seven-diagonal element arrays **a**, **b**, **c**, **d**, **e**, **f**, **g** is the same as that used to calculate the residuals, and therefore the one governing the problem. However, the convergence may be impaired if they are not equal. Indeed, if the system of equations (1) is not precisely of the seven-diagonal form illustrated above but has a few additional terms, then the methods of deferred or defect correction can be employed. The residual is calculated by the calling program using the full system of equations, but the up-date formula is based on a seven-diagonal system (2) of the form given above. For example, the solution of a system of eleven-diagonal equations each involving the combination of terms with $t_{i\pm1,j\pm1,k}, t_{i\pm1,j,k}, t_{i,j\pm1,k}, t_{i,j,k\pm1}$ and t_{ijk} could use the seven-diagonal coefficients on which to base the up-date, provided these incorporate the major features of the equations.

Problems in topologically non-rectangular box-shaped regions can be solved using the function by surrounding the region with a circumscribing topologically rectangular box. The equations for the nodal values external to the region of interest are set to zero (i.e., $d_{ijk} = r_{ijk} = 0$) 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 all zeros as the initial approximation from which the first set of residuals are determined.

The function can be used to solve linear elliptic equations in which case the arrays **a**, **b**, **c**, **d**, **e**, **f**, **g** and the quantities q will be unchanged during the iterative cycles, or for solving nonlinear elliptic equations in which case some or all of these arrays may require updating as each new approximate solution is derived. Depending 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 (see Jacobs 1972).

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 or of the up-date matrix formed from the arrays **a**, **b**, **c**, **d**, **e**, **f** and **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), the calling program should subtract a typical nodal value from the whole solution t at every iteration to keep 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 accurately implement 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

Ames W F 1977 *Nonlinear Partial Differential Equations in Engineering* (2nd Edition) Academic Press

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: $\mathbf{n2} > 1$.

3: **n3 – int32 scalar**

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

Constraint: $\mathbf{n3} > 1$.

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 $s_{ij,k-1}$ in the (i,j,k)th equation of the system (2) 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 $s_{i,j-1,k}$ in the (i,j,k)th equation of the system (2) 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 $s_{i-1,j,k}$ in the (i,j,k)th equation of the system (2), 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 s_{ijk}, the 'central' term, in the (i,j,k)th equation of the system (2), 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 $s_{ijk} = r_{ijk}$. 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, by setting $\mathbf{d}(i,j,k) = 0.0$ at appropriate points. The corresponding value of r_{ijk} is set equal to the appropriate value, namely the difference between the prescribed value of t_{ijk} and the current value in the Dirichlet case, or 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 $s_{i+1,j,k}$ in the (i,j,k)th equation of the system (2), 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 $s_{i,j+1,k}$ in the (i,j,k)th equation of the system (2), 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**.

g(i,j,k) must contain the coefficient of $s_{i,j,k+1}$ in the (i,j,k)th equation of the system (2), 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: **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: $0.0 < \mathbf{aparam} \leq \left((\mathbf{n1}-1)^2 + (\mathbf{n2}-1)^2 + (\mathbf{n3}-1)^2\right)/3.0$.

12: **it – int32 scalar**

The iteration number. It must be initialized, but not necessarily to 1, before the first call, and should be incremented by one in the calling program for each subsequent call. The function uses this counter to select the appropriate acceleration parameter from a sequence of nine, each one being used twice in succession. (Note that the acceleration parameter depends on the value of **aparam**.)

13: **r(lda,sda,n3) – double array**

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

The current residual r_{ijk} on the right-hand side of the (i,j,k)th equation of the system (2), for $i = 1,2,\ldots,\mathbf{n1}$ and $j = 1,2,\ldots,\mathbf{n2}$ and $k = 1,2,\ldots,\mathbf{n3}$.

5.2 Optional Input Parameters

1: **sda – int32 scalar**

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

Constraint: $\mathbf{sda} \geq \mathbf{n2}$.

5.3 Input Parameters Omitted from the MATLAB Interface

lda, wrksp1, wrksp2, wrksp3

5.4 Output Parameters

1: **r(lda,sda,n3) – double array**

These residuals are overwritten by the corresponding components of the solution s of the system (2), i.e., the changes to be made to the vector t to reduce the residuals supplied.

2: **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,
or **n3** < 2.

ifail = 2

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

ifail = 3

On entry, **aparam** ≤ 0.0.

ifail = 4

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

7 Accuracy

The improvement in accuracy for each iteration, i.e., on each call, 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***. However, since d03ub works with residuals and the up-date vector, the calling program can, in most cases where at each iteration all the residuals are usually of about the same size, calculate the residuals from extended precision values of the function, source term and equation coefficients if greater accuracy is required. 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 can be judged approximately from the rate of convergence determined from the changes to the dependent variable t and in particular the change on the last iteration.

8 Further Comments

The time taken is approximately proportional to **n1** × **n2** × **n3** for each call.

When used with deferred or defect correction, the residual is calculated in the calling program from a different system of equations to those represented by the seven-point molecule coefficients used by d03ub as the basis of the iterative up-date procedure. When using deferred correction the overall rate of convergence depends not only on the items detailed in Section 7 but also on the difference between the two coefficient matrices used.

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 may be associated with a ill-conditioned matrix.

9 Example

```
x = [0; 1; 3; 6];
y = [0; 1; 3; 6; 10];
z = [0; 1; 3; 6; 10; 15];
nits = 10;
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;
it = int32(1);
r = zeros(4, 5, 6);
% 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
for k = 1:n3
  for j = 1:n2
    for i = 1:n1
      if (d(i,j,k) ~= 0)
        % Seven point molecule formula
        r(i,j,k) = q(i,j,k) - a(i,j,k)*t(i,j,k-1) - b(i,j,k)*t(i,j-1,k) -
...
                     c(i,j,k)*t(i-1,j,k) - d(i,j,k)*t(i,j,k) - ...
                     e(i,j,k)*t(i+1,j,k) - f(i,j,k)*t(i,j+1,k) - ...
                     g(i,j,k)*t(i,j,k+1);
      else
        % Explicit equation
        r(i,j,k) = q(i,j,k) - t(i,j,k);
      end
    end
  end
end

[rOut, ifail] = d03ub(n1, n2, n3, a, b, c, d, e, f, g, aparam, it, r)
```

```
rOut =
(:,:,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    0.9886    0.9070    0.6579    0.1559
    1.2214    1.2045    1.1036    0.8001    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    0.8056    0.7361    0.5331    0.1277
    1.0000    0.9801    0.8931    0.6456    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    0.5963    0.5435    0.3928    0.0946
    0.7408    0.7246    0.6575    0.4735    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    0.4002    0.3649    0.2635    0.0634
    0.4966    0.4864    0.4413    0.3172    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
ifail =
           0
```