NAG Fortran Library Routine Document

D02SAF

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

1 Purpose

D02SAF solves a two-point boundary-value problem for a system of first-order ordinary differential equations with boundary conditions, combined with additional algebraic equations. It uses initial value techniques and a modified Newton iteration in a shooting and matching method.

2 Specification

```
      SUBROUTINE D02SAF(P, M, N, N1, PE, PF, E, DP, NPOINT, WP, IWP, ICOUNT,
     1                  RANGE, BC, FCN, EQN, CONSTR, YMAX, MONIT, PRSOL, W,
     2                  IW1, IW2, IFAIL)
      INTEGER           M, N, N1, NPOINT, IWP, ICOUNT, IW1, IW2, IFAIL
      real              P(M), PE(M), PF(M), E(N), DP(M), WP(IWP,6), YMAX,
     1                  W(IW1,IW2)
      LOGICAL           CONSTR
      EXTERNAL          RANGE, BC, FCN, EQN, CONSTR, MONIT, PRSOL
```

3 Description

D02SAF solves a two-point boundary-value problem for a system of n first-order ordinary differential equations with separated boundary conditions by determining certain unknown parameters $p_1, p_2, \ldots, p_m$. (There may also be additional algebraic equations to be solved in the determination of the parameters and, if so, these equations are defined by the routine EQN.) The parameters may be, but need not be, boundary values; they may include eigenvalues, parameters in the coefficients of the differential equations, coefficients in series expansions or asymptotic expansions for boundary values, the length of the range of definition of the system of differential equations etc.

It is assumed that we have a system of n differential equations of the form

$$y' = f(x, y, p) \tag{1}$$

where $p = (p_1, p_2, \ldots, p_m)^T$ is the vector of parameters, and that the derivative f is evaluated by a routine FCN. Also, n_1 of the equations are assumed to depend on p. For $n_1 < n$ the $n - n_1$ equations of the system are not involved in the matching process. These are the driving equations; they should be independent of p and of the solution of the other n_1 equations. In numbering the equations in FCN and BC the driving equations must be put **first** (as they naturally occur in most applications). The range of definition $[a, b]$ of the differential equations is defined by the routine RANGE and may depend on the parameters $p_1, p_2, \ldots, p_m$ (that is, on p). RANGE must define the points $x_1, x_2, \ldots, x_{\text{NPOINT}}$, NPOINT ≥ 2, which must satisfy

$$a = x_1 < x_2 < \cdots < x_{\text{NPOINT}} = b \tag{2}$$

(or a similar relationship with all the inequalities reversed).

If NPOINT>2 the points $x_1, x_2, \ldots, x_{\text{NPOINT}}$ can be used to break up the range of definition. Integration is restarted at each of these points. This means that the differential equations (1) can be defined differently in each sub-interval $[x_i, x_{i+1}]$, for $i = 1, 2, \ldots, \text{NPOINT} - 1$. Also, since initial and maximum integration step sizes can be supplied on each sub-interval (via the array WP), the user can indicate parts of the range $[a, b]$ where the solution $y(x)$ may be difficult to obtain accurately and can take appropriate action.

The boundary conditions may also depend on the parameters and are applied at $a = x_1$ and $b = x_{\text{NPOINT}}$. They are defined (in the routine BC) in the form

$$y(a) = g_1(p),\ y(b) = g_2(p). \tag{3}$$

The boundary-value problem is solved by determining the unknown parameters p by a shooting and matching technique. The differential equations are always integrated from a to b with initial values $y(a) = g_1(p)$. The solution vector thus obtained at $x = b$ is subtracted from the vector $g_2(p)$ to give the n_1 residuals $r_1(p)$, ignoring the first $n - n_1$, driving equations. Because the direction of integration is always from a to b, it is unnecessary, in BC, to supply values for the first $n - n_1$ boundary values at b, that is the first $n - n_1$ components of g_2 in (3). For $n_1 < m$ then $r_1(p)$. Together with the $m - n_1$ equations defined by routine EQN,

$$r_2(p) = 0, \tag{4}$$

these give a vector of residuals r, which at the solution, p, must satisfy

$$\begin{pmatrix} r_1(p) \\ r_2(p) \end{pmatrix} = 0. \tag{5}$$

These equations are solved by a pseudo-Newton iteration which uses a modified singular value decomposition of $J = \frac{\partial r}{\partial p}$ when solving the linear equations which arise. The Jacobian J used in Newton's method is obtained by numerical differentiation. The parameters at each Newton iteration are accepted only if the norm $\|\mathrm{D}^{-1}\tilde{J}^{+}r\|_2$ is much reduced from its previous value. Here $\tilde{J}^{+}$ is the pseudo-inverse, calculated from the singular value decomposition, of a modified version of the Jacobian J (J^{+} is actually the inverse of the Jacobian in well-conditioned cases). D is a diagonal matrix with

$$d_{ii} = \max(|p_i|, \mathrm{PF}(i)) \tag{6}$$

where PF is an array of floor values.

See Deuflhard (1974) for further details of the variants of Newton's method used, Gay (1976) for the modification of the singular value decomposition and Gladwell (1979a) for an overview of the method used.

Two facilities are provided to prevent the pseudo-Newton iteration running into difficulty. First, the user is permitted to specify constraints on the values of the parameters p via a logical function CONSTR. These constraints are only used to prevent the Newton iteration using values for p which would violate them; that is, they are not used to determine the values of p. Secondly, the user is permitted to specify a maximum value $y_{\max}$ for $\|y(x)\|_\infty$ at all points in the range $[a, b]$. It is intended that this facility be used to prevent machine 'overflow' in the integrations of equation (1) due to poor choices of the parameters p which might arise during the Newton iteration. When using this facility, it is presumed that the user has an estimate of the likely size of $\|y(x)\|_\infty$ at all points $x \in [a, b]$. $y_{\max}$ should then be chosen rather larger (say by a factor of 10) than this estimate.

The user is strongly advised to supply a routine MONIT (or to call the 'default' routine D02HBX, see below) to monitor the progress of the pseudo-Newton iteration. The user can output the solution of the problem $y(x)$ by supplying a suitable routine PRSOL (an example is given in Section 9 of a routine designed to output the solution at equally spaced points).

D02SAF is designed to try all possible options before admitting failure and returning to the user. Provided the routine can start the Newton iteration from the initial point p it will exhaust all the options available to it (though the user can override this by specifying a maximum number of iterations to be taken). The fact that all its options have been exhausted is the only error exit from the iteration. Other error exits are possible, however, whilst setting up the Newton iteration and when computing the final solution.

The user who requires more background information about the solution of boundary value problems by shooting methods is recommended to read the appropriate chapters of Hall and Watt (1976), and for a detailed description of D02SAF Gladwell (1979a) is recommended.

4 References

Deuflhard P (1974) A modified Newton method for the solution of ill-conditioned systems of nonlinear equations with application to multiple shooting *Numer. Math.* **22** 289–315

Gay D (1976) On modifying singular values to solve possibly singular systems of nonlinear equations *Working Paper 125* Computer Research Centre, National Bureau for Economics and Management Science, Cambridge, MA

Gladwell I (1979a) The development of the boundary value codes in the ordinary differential equations chapter of the NAG Library *Codes for Boundary Value Problems in Ordinary Differential Equations. Lecture Notes in Computer Science* (ed B Childs, M Scott, J W Daniel, E Denman and P Nelson) **76** Springer-Verlag

Hall G and Watt J M (ed.) (1976) *Modern Numerical Methods for Ordinary Differential Equations* Clarendon Press, Oxford

5 Parameters

1: P(M) – ***real*** array *Input/Output*

On entry: P(i) must be set to an estimate of the ith parameter, p_i, for $i = 1, 2, \ldots, m$.

On exit: the corrected value for the ith parameter, unless an error has occurred, when it contains the last calculated value of the parameter.

2: M – INTEGER *Input*

On entry: the number of parameters, m.

Constraint: M > 0.

3: N – INTEGER *Input*

On entry: the total number of differential equations, n.

Constraint: N > 0.

4: N1 – INTEGER *Input*

On entry: the number of differential equations active in the matching process, n_1. The active equations must be placed last in the numbering in the routines FCN and BC (see below). The **first** N – N1 equations are used as the driving equations.

Constraint: N1 ≤ N, N1 ≤ M and N1 > 0.

5: PE(M) – ***real*** array *Input*

On entry: PE(i), for $i = 1, 2, \ldots, m$, must be set to a positive value for use in the convergence test in the ith parameter p_i. See the specification of PF below for further details.

Constraint: PE(i) > 0, for $i = 1, 2, \ldots, m$.

6: PF(M) – ***real*** array *Input/Output*

On entry: PF(i), for $i = 1, 2, \ldots, m$, should be set to a 'floor' value in the convergence test on the ith parameter p_i. If PF(i) ≤ 0.0 on entry then it is set to the small positive value $\sqrt{\epsilon}$ (where ϵ may in most cases be considered to be ***machine precision***); otherwise it is used unchanged.

The Newton iteration is presumed to have converged if a full Newton step is taken (ISTATE=1 in the specification of MONIT below), the singular values of the Jacobian are not being significantly perturbed (also see MONIT) and if the Newton correction C_i satisfies

$$|C_i| \leq \mathrm{PE}(i) \times \max(|p_i|, \mathrm{PF}(i)), \quad i = 1, 2, \ldots, m,$$

where p_i is the current value of the ith parameter. The values PF(i) are also used in determining the Newton iterates as discussed in Section 3, see equation (6).

On exit: the values actually used.

7: E(N) – ***real*** array *Input*

On entry: values for use in controlling the local error in the integration of the differential equations. If err_i is an estimate of the local error in y_i, for $i = 1, 2, \ldots, n$ then

$$|err_i| \leq \mathrm{E}(i) \times \max\{\sqrt{\epsilon}, |y_i|\}$$

where ϵ may in most cases be considered to be ***machine precision***.

Suggested value: $\mathrm{E}(i) = 10^{-5}$.

Constraint: $\mathrm{E}(i) > 0.0$, for $i = 1, 2, \ldots, \mathrm{N}$.

8: DP(M) – ***real*** array *Input/Output*

On entry: a value to be used in perturbing the parameter p_i in the numerical differentiation to estimate the Jacobian used in Newton's method. If DP(i) = 0.0 on entry, an estimate is made internally by setting

$$\mathrm{DP}(i) = \sqrt{\epsilon} \times \max(\mathrm{PF}(i), |p_i|) \tag{7}$$

where p_i is the initial value of the parameter supplied by the user and ϵ may in most cases be considered to be ***machine precision***. The estimate of the Jacobian, J, is made using forward differences, that is for each i, for $i = 1, 2, \ldots, m$, p_i is perturbed to $p_i + \mathrm{DP}(i)$ and the ith column of J is estimated as

$$(r(p_i + \mathrm{DP}(i)) - r(p_i))/\mathrm{DP}(i)$$

where the other components of p are unchanged (see (3) for the notation used). If this fails to produce a Jacobian with significant columns, backward differences are tried by perturbing p_i to $p_i - \mathrm{DP}(i)$ and if this also fails then central differences are used with p_i perturbed to $p_i + 10.0 \times \mathrm{DP}(i)$. If this also fails then the calculation of the Jacobian is abandoned. If the Jacobian has not previously been calculated then an error exit is taken. If an earlier estimate of the Jacobian is available then the current parameter set, p_i, for $i = 1, 2, \ldots, \mathrm{M}$, is abandoned in favour of the last parameter set from which useful progress was made and the singular values of the Jacobian used at the point are modified before proceeding with the Newton iteration. The user is recommended to use the default value DP(i) = 0.0 unless he has prior knowledge of a better choice. If any of the perturbations described above are likely to lead to an unfortunate set of parameter values then the user should use the LOGICAL FUNCTION CONSTR (see below) to prevent such perturbations (all changes of parameters are checked by a call to CONSTR).

On exit: the values actually used.

9: NPOINT – INTEGER *Input*

On entry: 2 plus the number of break-points in the range of definition of the system of differential equations (1).

Constraint: NPOINT ≥ 2.

10: WP(IWP,6) – ***real*** array *Input/Output*

On entry: WP($i, 1$) must contain an estimate for an initial step size for integration across the ith sub-interval [X(i), X($i + 1$)], $i = 1, 2, \ldots,$ NPOINT $- 1$ (see RANGE below). WP($i, 1$) should have the same sign as X($i + 1$) $-$ X(i) if it is non-zero. If WP($i, 1$) = 0.0, on entry, a default value for the initial step size is calculated internally. This is the recommended mode of entry.

WP($i, 3$) must contain a lower bound for the modulus of the step size on the ith sub-interval [X(i), X($i + 1$)], for $i = 1, 2, \ldots,$ NPOINT $- 1$. If WP($i, 3$) = 0.0 on entry, a very small default value is used. By setting WP($i, 3$) > 0.0 but smaller than the expected step sizes (assuming the user has some insight into the likely step sizes) expensive integrations with parameters p far from the solution can be avoided.

WP($i, 2$) must contain an upper bound on the modulus of the step size to be used in the integration on [X(i), X($i + 1$)], $i = 1, 2, \ldots,$ NPOINT $- 1$. If WP($i, 2$) = 0.0 on entry no bound is assumed.

This is the recommended mode of entry unless the solution is expected to have important features which might be 'missed' in the integration if the step size were permitted to be chosen freely.

On exit: WP$(i,1)$ contains the initial step size used on the last integration on [X(i), X$(i+1)$], for $i = 1,2,\ldots,$NPOINT $- 1$, (excluding integrations during the calculation of the Jacobian).

WP$(i,2)$, for $i = 1,2,\ldots,$NPOINT $- 1$, is usually unchanged. If the maximum step size WP$(i,2)$ is so small or the length of the range [X(i), X$(i+1)$] is so short that on the last integration the step size was not controlled in the main by the size of the error tolerances E(i) but by these other factors, then WP(NPOINT,2) is set to the floating-point value of i if the problem last occurred in [X(i), X$(i+1)$]. Any results obtained when this value is returned as non-zero should be viewed with caution.

WP$(i,3)$, for $i = 1,2,\ldots,$NPOINT $- 1$ are unchanged.

If an error exit with IFAIL=4, 5, or 6 (see Section 6) occurs on the integration made from X(i) to X$(i+1)$ the floating-point value of i is returned in WP(NPOINT,1). The actual point $x \in$ [X(i), X$(i+1)$] where the error occurred is returned in WP(1,5) (see also the specification of W). The floating-point value of NPOINT is returned in WP(NPOINT,1) if the error exit is caused by a call to BC.

If an error exit occurs when estimating the Jacobian matrix (IFAIL=7, 8, 9, 10, 11, 12, see Section 6) and if parameter p_i was the cause of the failure then on exit WP(NPOINT,1) contains the floating-point value of i.

WP$(i,4)$ contains the point X(i), for $i = 1,2,\ldots,$NPOINT, used at the solution p or at the final values of p if an error occurred.

WP is also partly used as workspace.

11: IWP – INTEGER *Input*

On entry: the first dimension of the array WP as declared in the (sub)program from which D02SAF is called.

Constraint: IWP $\geq$ NPOINT.

12: ICOUNT – INTEGER *Input*

On entry: an upper bound on the number of Newton iterations. If ICOUNT=0 on entry, no check on the number of iterations is made (this is the recommended mode of entry).

Constraint: ICOUNT $\geq$ 0.

13: RANGE – SUBROUTINE, supplied by the user. *External Procedure*

RANGE must specify the break-points x_i, for $i = 1,2,\ldots,$NPOINT, which may depend on the parameters p_j, for $j = 1,2,\ldots,$M.

Its specification is:

```
SUBROUTINE RANGE(X, NPOINT, P, M)
INTEGER          NPOINT, M
real             X(NPOINT), P(M)
```

1: X(NPOINT) – ***real*** array *Output*

On exit: the ith break-point, for $i = 1,2,\ldots,$NPOINT. The sequence (X(i)) must be strictly monotonic, that is either

$$a = \mathrm{X}(1) < \mathrm{X}(2) < \cdots < \mathrm{X}(\mathrm{NPOINT}) = b$$

or

$$a = \mathrm{X}(1) > \mathrm{X}(2) > \cdots > \mathrm{X}(\mathrm{NPOINT}) = b.$$

2: NPOINT – INTEGER *Input*

On entry: two plus the number of break-points in (a, b).

3: P(M) – ***real*** array *Input*

On entry: the current estimate of the ith parameter, for $i = 1, 2, \ldots, m$.

4: M – INTEGER *Input*

On entry: the number of parameters, m.

RANGE must be declared as EXTERNAL in the (sub)program from which D02SAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

14: BC – SUBROUTINE, supplied by the user. *External Procedure*

BC must place in G1 and G2 the boundary conditions at a and b respectively.

Its specification is:

```
SUBROUTINE BC(G1, G2, P, M, N)
INTEGER       M, N
real          G1(N), G2(N), P(M)
```

1: G1(N) – ***real*** array *Output*

On exit: the value of $y_i(a)$, (where this may be a known value or a function of the parameters p_j, for $j = 1, 2, \ldots, m$), for $i = 1, 2, \ldots, n$.

2: G2(N) – ***real*** array *Output*

On exit: the value of $y_i(b)$, for $i = 1, 2, \ldots, n$, (where these may be known values or functions of the parameters p_j, for $j = 1, 2, \ldots, m$). If $n > n_1$, so that there are some driving equations, then the first $n - n_1$ values of G2 need not be set since they are never used.

3: P(M) – ***real*** array *Input*

On entry: an estimate of the ith parameter, p_i, for $i = 1, 2, \ldots, m$.

4: M – INTEGER *Input*

On entry: the number of parameters, m.

5: N – INTEGER *Input*

On entry: the number of differential equations, n.

BC must be declared as EXTERNAL in the (sub)program from which D02SAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

15: FCN – SUBROUTINE, supplied by the user. *External Procedure*

FCN must evaluate the functions f_i (i.e., the derivatives y_i'), for $i = 1, 2, \ldots, n$.

Its specification is:

```
SUBROUTINE FCN(X, Y, F, N, P, M, I)
INTEGER       N, M, I
real          X, Y(N), F(N), P(M)
```

1: X – ***real*** *Input*

On entry: the value of the argument x.

2: Y(N) – ***real*** array *Input*

On entry: the value of the argument y_i, for $i = 1, 2, \ldots, n$.

3: F(N) – ***real*** array *Output*

On exit: the derivative of y_i evaluated at x, for $i = 1, 2, \ldots, n$. F(i) may depend upon the parameters p_j, for $j = 1, 2, \ldots, m$. If there are any driving equations (see Section 3) then these must be numbered first in the ordering of the components of F.

4: N – INTEGER *Input*

On entry: the number of equations, n.

5: P(M) – ***real*** array *Input*

On entry: the current estimate of the ith parameter, p_i, for $i = 1, 2, \ldots, m$.

6: M – INTEGER *Input*

On entry: the number of parameters, m.

7: I – INTEGER *Input*

On entry: specifies the sub-interval $[x_i, x_{i+1}]$ on which the derivatives are to be evaluated.

FCN must be declared as EXTERNAL in the (sub)program from which D02SAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

16: EQN – SUBROUTINE, supplied by the user. *External Procedure*

EQN is used to describe the additional algebraic equations to be solved in the determination of the parameters, p_i, for $i = 1, 2, \ldots, m$. If there are no additional algebraic equations (i.e., $m = n_1$) then EQN is never called and the dummy routine D02HBZ should be used as the actual argument.

Its specification is:

```
SUBROUTINE EQN(E, Q, P, M)
INTEGER        Q, M
real           E(Q), P(M)
```

1: E(Q) – ***real*** array *Output*

On exit: the vector of residuals, $r_2(p)$, that is the amount by which the current estimates of the parameters fail to satisfy the algebraic equations.

2: Q – INTEGER *Input*

On entry: the number of algebraic equations, $m - n_1$.

3: P(M) – ***real*** array *Input*

On entry: the current estimate of the ith parameter p_i, for $i = 1, 2, \ldots, m$.

4: M – INTEGER *Input*

On entry: the number of parameters, m.

EQN must be declared as EXTERNAL in the (sub)program from which D02SAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

17: CONSTR – LOGICAL FUNCTION, supplied by the user. *External Procedure*

CONSTR is used to prevent the pseudo-Newton iteration running into difficulty. CONSTR should return the value .TRUE. if the constraints are satisfied by the parameters $p_1, p_2, \ldots, p_m$. Otherwise CONSTR should return the value .FALSE.. Usually the dummy function D02HBY, which returns the value .TRUE. at all times, will suffice and in the first instance this is recommended as the actual parameter.

Its specification is:

```
LOGICAL FUNCTION CONSTR(P, M)
INTEGER                M
real                   P(M)
```

1: P(M) – ***real*** array *Input*

On entry: an estimate of the ith parameter, p_i, for $i = 1, 2, \ldots, m$.

2: M – INTEGER *Input*

On entry: the number of parameters, m.

CONSTR must be declared as EXTERNAL in the (sub)program from which D02SAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

18: YMAX – ***real*** *Input*

On entry: a non-negative value which is used as a bound on all values $\|y(x)\|_\infty$ where $y(x)$ is the solution at any point x between X(1) and X(NPOINT) for the current parameters $p_1, p_2, \ldots, p_m$. If this bound is exceeded the integration is terminated and the current parameters are rejected. Such a rejection will result in an error exit if it prevents the initial residual or Jacobian, or the final solution, being calculated. If YMAX=0 on entry, no bound on the solution y is used; that is the integrations proceed without any checking on the size of $\|y\|_\infty$.

19: MONIT – SUBROUTINE, supplied by the user. *External Procedure*

MONIT enables the user to monitor the values of various quantities during the calculation. It is called by D02SAF after every calculation of the norm $\|\mathrm{D}^{-1}\tilde{J}_{+r}\|_2$ which determines the strategy of the Newton method, every time there is an internal error exit leading to a change of strategy, and before an error exit when calculating the initial Jacobian. Usually the routine will be adequate and the user is advised to use this as the actual parameter for MONIT in the first instance. (In this case a call to X04ABF must be made prior to the call of D02SAF.) If no monitoring is required, the dummy routine D02SAS may be used. (In some implementations of the Library the names D02HBX and D02SAS are changed to HBXD02 and SASD02: refer to the Users' Note for your implementation.)

Its specification is:

```
 SUBROUTINE MONIT(ISTATE, IFLAG, IFAIL1, P, M, F, PNORM, PNORM1, EPS,
1                 D)
 INTEGER          ISTATE, IFLAG, IFAIL1, M
 real             P(M), F(M), PNORM, PNORM1, EPS, D(M)
```

1: ISTATE – INTEGER *Input*

On entry: the state of the Newton iteration:

ISTATE = 0

The calculation of the residual, Jacobian and $\|\mathrm{D}^{-1}\tilde{J}^{+}r\|_2$ are taking place.

ISTATE $= 1$ to 5

During the Newton iteration a factor of $2^{(-\text{ISTATE}+1)}$ of the Newton step is being used to try to reduce the norm.

ISTATE $= 6$

The current Newton step has been rejected and the Jacobian is being re-calculated.

ISTATE $= -6$ to -1

An internal error exit has caused the rejection of the current set of parameter values, p. $-$ISTATE is the value which ISTATE would have taken if the error had not occurred.

ISTATE $= -7$

An internal error exit has occurred when calculating the initial Jacobian.

2: IFLAG – INTEGER *Input*

On entry: whether or not the Jacobian being used has been calculated at the beginning of the current iteration. If the Jacobian has been updated then IFLAG=1; otherwise IFLAG=2. The Jacobian is only calculated when convergence to the current parameter values has been slow.

3: IFAIL1 – INTEGER *Input*

On entry: if $-6 \leq$ ISTATE ≤ -1, IFAIL1 specifies the IFAIL error number that would be produced were control returned to the user. IFAIL1 is unspecified for values of ISTATE outside this range.

4: P(M) – ***real*** array *Input*

On entry: the current estimate of the ith parameter, p_i, for $i = 1, 2, \ldots, m$.

5: M – INTEGER *Input*

On entry: the number of parameters, m.

6: F(M) – ***real*** array *Input*

On entry: the residual r corresponding to the current parameter values, provided $1 \leq$ ISTATE ≤ 5 or ISTATE $= -7$. F is unspecified for other values of ISTATE.

7: PNORM – ***real*** *Input*

On entry: a quantity against which all reductions in norm are currently measured.

8: PNORM1 – ***real*** *Input*

On entry: the norm of the current parameters, p. It is set for $1 \leq$ ISTATE ≤ 5 and is undefined for other values of ISTATE.

9: EPS – ***real*** *Input*

On entry: EPS gives some indication of the convergence rate. It is the current singular value modification factor (see Gay (1976)). It is 0 initially and whenever convergence is proceeding steadily. EPS is $\epsilon^{3/8}$ or greater (where ϵ may in most cases be considered ***machine precision***) when the singular values of J are approximately zero or when convergence is not being achieved. The larger the value of EPS the worse the convergence rate. When EPS becomes too large the Newton iteration is terminated.

10: D(M) – ***real*** array *Input*

On entry: the singular values of the current modified Jacobian matrix, J. If $\mathrm{D}(m)$ is small relative to D(1) for a number of Jacobians corresponding to different parameter values then the computed results should be viewed with suspicion. It could be that the matching equations do not depend significantly on some parameter (which could be due to a programming error in FCN, BC, RANGE or EQN). Alternatively, the system of differential equations may be very ill-conditioned when viewed as an initial value problem, in which case this routine is unsuitable. This may also be indicated by some singular values being very large. These values of $\mathrm{D}(i)$, $i = 1, 2, \ldots, m$ should not be changed.

MONIT must be declared as EXTERNAL in the (sub)program from which D02SAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

20: PRSOL – SUBROUTINE, supplied by the user. *External Procedure*

PRSOL can be used to obtain values of the solution y at a selected point z by integration across the final range [X(1),X(NPOINT)]. If no output is required D02HBW can be used as the actual parameter.

Its specification is:

```
SUBROUTINE PRSOL(Z, Y, N)
INTEGER          N
real             Z, Y(N)
```

1: Z – ***real*** *Input/Output*

On entry: contains x_1 on the first call. On subsequent calls Z contains its previous output value.

On exit: the next point at which output is required. The new point must be nearer X(NPOINT) than the old.

If Z is set to a point outside [X(1),X(NPOINT)] the process stops and control returns from D02SAF to the (sub)program from which D02SAF is called. Otherwise the next call to PRSOL is made by D02SAF at the point Z, with solution values $y_1, y_2, \ldots, y_n$ at Z contained in Y. If Z is set to X(NPOINT) exactly, the final call to PRSOL is made with $y_1, y_2, \ldots, y_n$ as values of the solution at X(NPOINT) produced by the integration. In general the solution values obtained at X(NPOINT) from PRSOL will differ from the values obtained at this point by a call to routine BC. The difference between the two solutions is the residual r. The user is reminded that the points $\mathrm{X}(1), \mathrm{X}(2), \ldots, \mathrm{X}(\mathrm{NPOINT})$ are available in the locations $\mathrm{WP}(1,4), \mathrm{WP}(2,4), \ldots, \mathrm{WP}(\mathrm{NPOINT},4)$ at all times.

2: Y(N) – ***real*** array *Input*

On entry: the solution value y_i at z, for $i = 1, 2, \ldots, n$.

3: N – INTEGER *Input*

On entry: the total number of differential equations, n.

PRSOL must be declared as EXTERNAL in the (sub)program from which D02SAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

21: W(IW1,IW2) – ***real*** array *Output*

On exit: in the case of an error exit of the type where the point of failure is returned in WP(1,5), the solution at this point of failure is returned in $\mathrm{W}(i,1)$, for $i = 1, 2, \ldots, n$.

Otherwise W is used for workspace.

22: IW1 – INTEGER *Input*

On entry: the first dimension of the array W as declared in the (sub)program from which D02SAF is called.

Constraint: IW1 $\geq$ max(N, M).

23: IW2 – INTEGER *Input*

On entry: the second dimension of the array W as declared in the (sub)program from which D02SAF is called.

Constraint: IW2 $\geq 3 \times$ M $+ 12 +$ max(11, M).

24: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. Users who are unfamiliar with this parameter should refer to Chapter P01 for details.

On exit: IFAIL $= 0$ unless the routine detects an error (see Section 6).

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, for users not familiar with this parameter the recommended value is 0. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

6 Error Indicators and Warnings

If on entry IFAIL $= 0$ or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

IFAIL $= 1$

One or more of the parameters N, N1, M, IWP, NPOINT, ICOUNT, IW1, IW2, E, PE or YMAX has been incorrectly set.

IFAIL $= 2$

The constraints have been violated by the initial parameters.

IFAIL $= 3$

The condition X(1) $<$ X(2) $< \cdots <$ X(NPOINT) (or X(1) $>$ X(2) $> \cdots >$ X(NPOINT)) has been violated on a call to RANGE with the initial parameters.

IFAIL $= 4$

In the integration from X(1) to X(NPOINT) with the initial or the final parameters, the step size was reduced too far for the integration to proceed. Consider reversing the order of the points X(1), X(2), $\ldots$, X(NPOINT). If this error exit still results, it is likely that D02SAF is not a suitable method for solving the problem, or the initial choice of parameters is very poor, or the accuracy requirement specified by E(i), for $i = 1, 2, \ldots, n$, is too stringent.

IFAIL $= 5$

In the integration from X(1) to X(NPOINT) with the initial or final parameters, an initial step could not be found to start the integration on one of the intervals X(i) to X($i + 1$). Consider reversing the order of the points. If this error exit still results it is likely that D02SAF is not a suitable routine for solving the problem, or the initial choice of parameters is very poor, or the accuracy requirement specified by E(i), for $i = 1, 2, \ldots, n$, is much too stringent.

IFAIL = 6

In the integration from X(1) to X(NPOINT) with the initial or final parameters, the solution exceeded YMAX in magnitude (when YMAX > 0). It is likely that the initial choice of parameters was very poor or YMAX was incorrectly set.

Note: on an error with IFAIL = 4, 5 or 6 with the initial parameters, the interval in which failure occurs is contained in WP(NPOINT,1). If a subroutine MONIT similar to the one in Section 9 is being used then it is a simple matter to distinguish between errors using the initial and final parameters. None of the error exits IFAIL = 4, 5 or 6 should occur on the **final** integration (when computing the solution) as this integration has already been performed previously with exactly the same parameters p_i, for $i = 1, 2, \ldots, m$. Seek expert help if this error occurs.

IFAIL = 7

On calculating the initial approximation to the Jacobian, the constraints were violated.

IFAIL = 8

On perturbing the parameters when calculating the initial approximation to the Jacobian, the condition X(1) $<$ X(2) $< \ldots <$ X(NPOINT) (or X(1) $>$ X(2) $> \ldots >$ X(NPOINT)) is violated.

IFAIL = 9

On calculating the initial approximation to the Jacobian, the integration step size was reduced too far to make further progress (see IFAIL=4).

IFAIL = 10

On calculating the initial approximation to the Jacobian, the initial integration step size on some interval was too small (see IFAIL=5).

IFAIL = 11

On calculating the initial approximation to the Jacobian, the solution of the system of differential equations exceeded YMAX in magnitude (when YMAX > 0).

Note: all the error exits IFAIL=7, 8, 9, 10 and 11 can be treated by reducing the size of some or all the elements of DP.

IFAIL = 12

On calculating the initial approximation to the Jacobian, a column of the Jacobian is found to be insignificant. This could be due to an element DP(i) being too small (but non-zero) or the solution having no dependence on one of the parameters (a programming error).

Note: on an error exit with IFAIL = 7, 8, 9, 10, 11 or 12, if a perturbation of the parameter p_i is the cause of the error then WP(NPOINT,1) will contain the floating-point value of i.

IFAIL = 13

After calculating the initial approximation to the Jacobian, the calculation of its singular value decomposition failed. It is likely that the error will never occur as it is usually associated with the Jacobian having multiple singular values. To remedy the error it should only be necessary to change the initial parameters. If the error persists it is likely that the problem has not been correctly formulated.

IFAIL = 14

The Newton iteration has failed to converge after exercising all its options. The user is strongly recommended to monitor the progress of the iteration via the parameter MONIT. There are many possible reasons for the iteration not converging. Amongst the most likely are:

(a) there is no solution;

(b) the initial parameters are too far away from the correct parameters;

(c) the problem is too ill-conditioned as an initial value problem for Newton's method to choose suitable corrections;

(d) the accuracy requirements for convergence are too restrictive, that is some of the components of PE (and maybe PF) are too small – in this case the final value of this norm output via MONIT will usually be very small; or

(e) the initial parameters are so close to the solution parameters p that the Newton iteration cannot find improved parameters. The norm output by MONIT should be very small.

IFAIL = 15

The number of iterations permitted by ICOUNT has been exceeded (in the case when ICOUNT > 0 on entry).

IFAIL = 16
IFAIL = 17
IFAIL = 18
IFAIL = 19

These indicate that there has been a serious error in one of the auxiliary routines D02SAZ, D02SAW, D02SAX or D02SAU respectively. Check all subroutine calls and array dimensions. Seek expert help.

7 Accuracy

If the iteration converges, the accuracy to which the unknown parameters are determined is usually close to that specified by the user. The accuracy of the solution (output via PRSOL) depends on the error tolerances E(i), for $i = 1, 2, \ldots, n$. The user is strongly recommended to vary all tolerances to check the accuracy of the parameters p and the solution y.

8 Further Comments

The time taken by the routine depends on the complexity of the system of differential equations and on the number of iterations required. In practice, the integration of the differential system (1) is usually by far the most costly process involved. The computing time for integrating the differential equations can sometimes depend critically on the quality of the initial estimates for the parameters p. If it seems that too much computing time is required and, in particular, if the values of the residuals (output in MONIT) are much larger than expected given the user's knowledge of the expected solution, then the coding of the subroutines FCN, EQN, RANGE and BC should be checked for errors. If no errors can be found then an independent attempt should be made to improve the initial estimates p.

In the case of an error exit in the integration of the differential system indicated by IFAIL = 4, 5, 9 or 10 the user is strongly recommended to perform trial integrations with D02PDF to determine the effects of changes of the local error tolerances and of changes to the initial choice of the parameters p_i, for $i = 1, 2 \ldots, m$ (that is the initial choice of p).

It is possible that by following the advice given in Section 6 an error exit with IFAIL = 7, 8, 9, 10 or 11 might be followed by one with IFAIL=12 (or vice-versa) where the advice given is the opposite. If the user is unable to refine the choice of DP(i), for $i = 1, 2, \ldots, n$, such that both these types of exits are avoided then the problem should be rescaled if possible or the method must be abandoned.

The choice of the 'floor' values PF(i), for $i = 1, 2, \ldots, m$, may be critical in the convergence of the Newton iteration. For each value i, the initial choice of p_i and the choice of PF(i) should not both be very small unless it is expected that the final parameter p_i will be very small and that it should be determined accurately in a **relative** sense.

For many problems it is critical that a good initial estimate be found for the parameters p or the iteration will not converge or may even break down with an error exit. There are many mathematical techniques which obtain good initial estimates for p in simple cases but which may fail to produce useful estimates in harder cases. If no such technique is available it is recommended that the user try a continuation (homotopy) technique preferably based on a physical parameter (e.g., the Reynolds or Prandtl number is often a suitable continuation parameter). In a continuation method a sequence of problems is solved, one for each choice of the continuation parameter, starting with the problem of interest. At each stage the parameters p calculated at earlier stages are used to compute a good initial estimate for the parameters at the current stage (see Hall and Watt (1976) for more details).

9 Example

The following example program is intended to illustrate the use of the break-point and equation solving facilities of D02SAF. Most of the facilities which are common to D02SAF and D02HBF are illustrated in the example in the specification of D02HBF (which should also be consulted).

The program solves a projectile problem in two media determining the position of change of media, p_3, and the gravity and viscosity in the second medium (p_2 represents gravity and p_4 represents viscosity).

9.1 Program Text

Note: the listing of the example program presented below uses ***bold italicised*** terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     D02SAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, M, NPOINT, IWP, NMMAX, IW1, IW2, N1
      PARAMETER        (N=3,M=4,NPOINT=3,IWP=NPOINT,NMMAX=M,IW1=NMMAX,
     +                 IW2=3*M+23,N1=N)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      real             XEND
      INTEGER          ICAP
*     .. Local Scalars ..
      real             YMAX
      INTEGER          I, ICOUNT, IFAIL, J
*     .. Local Arrays ..
      real             DP(M), E(N), P(M), PE(M), PF(M), W(IW1,IW2),
     +                 WP(IWP,6)
*     .. External Functions ..
      LOGICAL          CONSTR
      EXTERNAL         CONSTR
*     .. External Subroutines ..
      EXTERNAL         BC, D02SAF, D02SAS, EQN, FCN, PRSOL, RANGE,
     +                 X04ABF
*     .. Common blocks ..
      COMMON           /ENDDAT/XEND, ICAP
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02SAF Example Program Results'
      ICAP = 0
      ICOUNT = 0
      YMAX = 0.0e0
      XEND = 5.0e0
      DO 20 I = 1, M
         PE(I) = 1.0e-3
         PF(I) = 1.0e-6
         DP(I) = 0.0e0
   20 CONTINUE
      DO 40 I = 1, N
         E(I) = 1.0e-5
   40 CONTINUE
      CALL X04ABF(1,NOUT)
      DO 80 I = 1, NPOINT - 1
         DO 60 J = 1, 3
            WP(I,J) = 0.0e0
   60    CONTINUE
   80 CONTINUE
```

```
      P(1) = 1.2e0
      P(2) = 0.032e0
      P(3) = 2.5e0
      P(4) = 0.02e0
      IFAIL = 1
*
*     * To obtain monitoring information, replace the name D02SAS
*     by D02HBX in the next statement and declare D02HBX as external *
*
      CALL D02SAF(P,M,N,N1,PE,PF,E,DP,NPOINT,WP,IWP,ICOUNT,RANGE,BC,FCN,
     +            EQN,CONSTR,YMAX,D02SAS,PRSOL,W,IW1,IW2,IFAIL)
*
      IF (IFAIL.NE.0) THEN
         WRITE (NOUT,99999) 'IFAIL = ', IFAIL
         IF (IFAIL.GE.4) THEN
            IF (IFAIL.LE.12) WRITE (NOUT,99998) 'WP(NPOINT,1) = ',
     +          WP(NPOINT,1)
            IF (IFAIL.LE.6) THEN
               WRITE (NOUT,99998) 'WP(1,5) = ', WP(1,5)
               WRITE (NOUT,99997) 'W(.,1) ', (W(I,1),I=1,N)
            END IF
         END IF
      END IF
      STOP
*
99999 FORMAT (1X,A,I3)
99998 FORMAT (1X,A,F10.4)
99997 FORMAT (1X,A,10e10.3)
      END
*
      SUBROUTINE EQN(F,Q,P,M)
*     .. Scalar Arguments ..
      INTEGER        M, Q
*     .. Array Arguments ..
      real           F(Q), P(M)
*     .. Executable Statements ..
      F(1) = 0.02e0 - P(4) - 1.0e-5*P(3)
      RETURN
      END
*
      SUBROUTINE FCN(X,Y,F,N,P,M,I)
*     .. Scalar Arguments ..
      real           X
      INTEGER        I, M, N
*     .. Array Arguments ..
      real           F(N), P(M), Y(N)
*     .. Intrinsic Functions ..
      INTRINSIC      COS, TAN
*     .. Executable Statements ..
      F(1) = TAN(Y(3))
      IF (I.EQ.1) THEN
         F(2) = -0.032e0*TAN(Y(3))/Y(2) - 0.02e0*Y(2)/COS(Y(3))
         F(3) = -0.032e0/Y(2)**2
      ELSE
         F(2) = -P(2)*TAN(Y(3))/Y(2) - P(4)*Y(2)/COS(Y(3))
         F(3) = -P(2)/Y(2)**2
      END IF
      RETURN
      END
*
      SUBROUTINE BC(F,G,P,M,N)
*     .. Scalar Arguments ..
      INTEGER       M, N
*     .. Array Arguments ..
      real          F(N), G(N), P(M)
*     .. Executable Statements ..
      F(1) = 0.0e0
      F(2) = 0.5e0
      F(3) = P(1)
      G(1) = 0.0e0
      G(2) = 0.45e0
      G(3) = -1.2e0
      RETURN
      END
*
      SUBROUTINE RANGE(X,NPOINT,P,M)
*     .. Scalar Arguments ..
      INTEGER          M, NPOINT
*     .. Array Arguments ..
      real             P(M), X(NPOINT)
*     .. Scalars in Common ..
      real             XEND
      INTEGER          ICAP
*     .. Common blocks ..
      COMMON           /ENDDAT/XEND, ICAP
*     .. Executable Statements ..
      X(1) = 0.0e0
      X(2) = P(3)
      X(3) = XEND
      RETURN
      END
*
      SUBROUTINE PRSOL(X,Y,N)
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalar Arguments ..
      real             X
      INTEGER          N
*     .. Array Arguments ..
      real             Y(N)
*     .. Scalars in Common ..
      real             XEND
      INTEGER          ICAP
*     .. Local Scalars ..
      INTEGER          I
*     .. Intrinsic Functions ..
      INTRINSIC        ABS
*     .. Common blocks ..
      COMMON           /ENDDAT/XEND, ICAP
*     .. Executable Statements ..
      IF (ICAP.NE.1) THEN
         ICAP = 1
         WRITE (NOUT,*)
         WRITE (NOUT,*) '      X       Y(1)      Y(2)      Y(3)'
      END IF
      WRITE (NOUT,99999) X, (Y(I),I=1,N)
      X = X + 0.5e0
      IF (ABS(X-XEND).LT.0.25e0) X = XEND
      RETURN
*
99999 FORMAT (1X,F9.3,3F10.4)
      END
*
      LOGICAL FUNCTION CONSTR(P,M)
*     .. Scalar Arguments ..
      INTEGER                 M
*     .. Array Arguments ..
      real                    P(M)
*     .. Local Scalars ..
      INTEGER                 I
*     .. Executable Statements ..
      CONSTR = .TRUE.
      DO 20 I = 1, M
         IF (P(I).LT.0.0e0) CONSTR = .FALSE.
   20 CONTINUE
      IF (P(3).GT.5.0e0) CONSTR = .FALSE.
      RETURN
      END
```

9.2 Program Data

None.

9.3 Program Results

```
 D02SAF Example Program Results

      X        Y(1)       Y(2)       Y(3)
     0.000    0.0000     0.5000     1.1753
     0.500    1.0881     0.4127     1.0977
     1.000    1.9501     0.3310     0.9802
     1.500    2.5768     0.2582     0.7918
     2.000    2.9606     0.2019     0.4796
     2.500    3.0958     0.1773     0.0245
     3.000    2.9861     0.1935    -0.4353
     3.500    2.6289     0.2409    -0.7679
     4.000    2.0181     0.3047    -0.9767
     4.500    1.1454     0.3759    -1.1099
     5.000   -0.0000     0.4500    -1.2000
```