1 SUMMARY

DC03A/AD is the main subroutine of a package for solution of the system of ordinary differential equations

\[
\frac{dU_i}{dX} = F(X, U_1, U_2, \ldots, U_{NEQ}), \quad i = 1, 2, \ldots, NEQ
\]

(1)

from given initial values \( Y_i \) of the \( U_i \) at \( X = T \), using a variable order Backward Differentiation Formula (BDF) method, also known as Gear’s method, of order 1 to 5. The method automatically chooses stepsize and order of the integration formula, and is especially efficient on stiff systems (roughly, those involving very short time constants). The concept of stiffness is not easy to describe precisely; if you do not know your problem to be non-stiff, it may be wise to use DC03A/AD for safety, since you can waste far more computer time by using a non-stiff method on a stiff problem than by using a stiff method on most non-stiff problems; the main exception is where very high accuracy is required on non-stiff problems.

DC03A/AD can also solve some Mixed Algebraic-Differential (MAD) problems, in which the derivative on the left-hand side of some of the above equations is replaced by zero; see under INFO(7), section 3.

The subroutine integrates from \( T \) to \( TOUT \). It is easy to continue the integration to get results at additional \( TOUT \); this is the interval mode of operation. It is also easy to get the solution at each intermediate step on the way to \( TOUT \); this is the intermediate-output mode.

DC03A/AD should be considered to supersede DC01AD; it uses the highly efficient and reliable BDF integrator (ref. 1) developed for the FACSIMILE reaction kinetics code. A simpler user interface will be provided by subroutine DC04A/AD, which will call DC03A/AD and will be similar to DC02AD. The user interface to DC03A/AD itself is closely modelled, with grateful acknowledgement, on one described by Shampine and Watts (ref. 2), and this description of it also follows theirs.

Solution of stiff or MAD problems is numerically demanding; the single-precision version DC03A may not work well on 32-bit computers, e.g. I.B.M. Also, the Jacobian matrix

\[
J_i = \frac{\partial F_i}{\partial U_j}
\]

(2)
of the equations (1) will be used in solving such problems, from the beginning for a MAD problem or after a possible initial transient for a stiff problem. This need is described more fully later.


2. Calling Sequence and Argument List

The single precision version:

\[
\text{CALL } \text{DC03A}(F, \text{NEQ}, T, Y, \text{TOUT}, \text{INFO}, \text{TOL}, \text{YDER}, \text{IDID}, \text{RWORK}, \\
* \quad \text{LRW, IWORK, LIW, RPAR, IPAR, PTN, JAC})
\]

The double precision version:

\[
\text{CALL } \text{DC03AD}(F, \text{NEQ}, T, Y, \text{TOUT}, \text{INFO}, \text{TOL}, \text{YDER}, \text{IDID}, \text{RWORK}, \\
* \quad \text{LRW, IWORK, LIW, RPAR, IPAR, PTN, JAC})
\]

\( F \) is the name of a subroutine which you provide to define the differential equations. It must provide DOUBLE PRECISION values for the D version.

\( \text{NEQ} \) is an INTEGER variable set to the number of first order differential equations to be integrated.

\( \text{T} \) is a REAL (DOUBLE PRECISION in the D version) variable of the independent variable \( X \); on first entry, the
initial value.

\textbf{Y} is a REAL (DOUBLE PRECISION in the D version) array containing the components of the solution vector \( U \) at \( X = T \).

\textbf{TOUT} is a REAL (DOUBLE PRECISION in the D version) variable of \( X \) at which a solution is desired.

\textbf{INFO} is an INTEGER array of length 15 which you use to communicate how you want DC03A/AD to carry out its task (see section 3).

\textbf{TOL} is a REAL (DOUBLE PRECISION in the D version) variable representing the accuracy you want in your solution.

\textbf{YDER} is a REAL (DOUBLE PRECISION in the D version) array used to hold the components of \( dU/dX \) at \( X = T \), and in which you may at the start of a problem, if you wish, give information about the expected sizes of the components of \( U \).

\textbf{IDID} is an INTEGER variable reporting on return what the code did; you must monitor it to decide what to do next (see section 4).

\textbf{RWORK} is a REAL (DOUBLE PRECISION in the D version) work array of length \( LRW \) which provides the code with storage space.

\textbf{LRW} is an INTEGER variable the size of array RWORK; the minimum value it must have depends on the Jacobian option chosen (see INFO(6), section 3).

\textbf{IWORK} is an INTEGER work array of length \( LIW \) which provides the code with storage space.

\textbf{LIW} is an INTEGER variable set to the size of array IWORK; the minimum value it must have depends on the Jacobian option chosen (see INFO(6), section 3).

\textbf{RPAR,IPAR} are REAL (DOUBLE PRECISION in the D version) and INTEGER arrays which you can use for communication between your calling program and your \( F \) subroutine.

\textbf{PTN} is the name of a subroutine which you must supply, if you choose the sparse matrix option (see section 7), to define the pattern of non-zero elements in the Jacobian matrix of the system. If you do not choose the sparse option, you can use DC03P/PD as a dummy name.

\textbf{JAC} is the name of an optional subroutine which you may provide to define the Jacobian matrix. If you do not choose to do so, you can use DC03J/JD as a dummy name.

**3. Input on the First Call to DC03A/AD**

The first call is defined to be the start of each new problem:

\textbf{F} Provide a subroutine of the form

\[
F(X,U,UPRIME,RPAR,IPAR)
\]

to define the system of first-order differential equations to be solved. For given values of \( X \) and the array \( U \) (of dimension \( NEQ \)) it must evaluate the \( NEQ \) components of the derivatives \( dU/dX = F(X,U) \) and put them in array \( UPRIME \), that is \( UPRIME(I)=dU(I)/dX \) for \( I = 1 \) to \( NEQ \). Subroutine \( F \) must not alter \( X \) or \( U \). You must declare it \textsc{external} in your program that calls DC03A/AD. You must dimension \( U \) and \( UPRIME \) in \( F \), and declare them (and \( X \), \( RPAR \) (DOUBLE PRECISION in the D version). \textbf{RPAR} and \textbf{IPAR} are parameter arrays which you can use for communication between your calling program and \( F \). They are not used or altered by DC03A/AD. If you do not need them, treat them as dummy arguments. If you do use them, dimension them appropriately in your calling program and in \( F \).

\textbf{NEQ} Set it to the number of differential equations (must be greater than 0).

\textbf{T} Set it to the initial value of \( X \). You must use a variable because the code changes \( T \).

\textbf{Y} Set this array to the initial values of \( U(I) \), \( I = 1 \) to \( NEQ \).
TOUT Set it to the first point at which a solution is desired. If you set \( TOUT = T \), the code will evaluate the derivatives at \( T \) and then return. \( TOUT \leq T \) is not allowed (stiff systems must be integrated forward). The code advances the solution from \( T \) towards \( TOUT \) using step sizes chosen to achieve the specified accuracy. If you wish, you can make it return with the solution and its derivative at the end of each intermediate step, but you must still provide \( TOUT \) as a target for the code to aim at. \( TOUT \) should be chosen with care on the first call, as the code estimates the timescale from \( (TOUT - T) \); the first step will not exceed \( TOL \cdot (TOUT - T) \) in size. On most problems, the code can easily reduce too large a first step (unless overflows occur), but problems where most variables start off from zero with zero initial slope may need some guidance, which the value of \( (TOUT - T) \) supplies. Note that the code normally integrates past \( TOUT \) – see \( INFO(4) \) below.

INFO Use the \( INFO \) array to give more details about how you want the problem solved. This array should be of length 15 to allow possible future extensions, but only the first 10 entries are used at present. Unused entries in \( INFO \) should be set to zero; very few users are likely to need to use \( INFO(10) \). Answer the following questions and set the elements of \( INFO \) accordingly. The simplest case corresponds to setting all entries to 0.

\( INFO(1) \) – Enables the code to initialise itself. You must set it to 0 to indicate the start of each new problem (this includes any change in the subroutine \( F \) or in parameters \( RPAR, IPAR \) used by it); or to 2 if you do not want statistical counts reset to zero (e.g. on restarting after a discontinuity).

- Is this the first call for this problem?
  - YES – set \( INFO(1) = 0 \) (or =2, optionally, on a restart).
  - NO – not applicable here; on later entries, leave \( INFO(1) \) unchanged normally – but see section 5.

\( INFO(2) \) – Allows you to specify the expected sizes \( USIZ(I) \) of the components of the solution in the array \( YDER \). The error estimate on each component \( U(I) \) is compared with a number \( W(I) \) which is (approximately – see under \( TOL \)) the sum of the current absolute value of \( U(I) \), multiplied by a number \( URFAC \) whose default value is 1.0, and the larger of:
  1. the largest absolute value of \( U(I) \) so far in the integration, multiplied by a number \( UMFAC \) whose default value is \( 10^{-10} \);
  2. zero, or \( USIZ(I) \) if you decide to supply them.

You need not supply \( USIZ \), even if some of the \( U(I) \) are initially zero – the code usually copes well with this.

- Do you want the default value zero for (b)?
  - YES – set \( INFO(2) = 0 \).
  - NO – set \( INFO(2) = 1 \), and the required values \( USIZ(I) \) in \( YDER(I) \), \( I = 1 \) to \( NEQ \).

\( INFO(3) \) – The code integrates from \( T \) to \( TOUT \) in steps. If you wish, it will return the computed solution and derivatives at the next step (the intermediate-output mode) or at \( TOUT \), whichever comes first. (If you need the solution at a great many specific \( TOUT \) points, this code will compute them efficiently, without shortening its integration steps).

- Do you want the solution at \( TOUT \) only (and not at the next integration step)?
  - YES – set \( INFO(3) = 0 \).
  - NO – set \( INFO(3) = 1 \).

\( INFO(4) \) – To give output efficiently at many specific \( TOUT \) values, the code integrates past \( TOUT \) and interpolates to get the result there. Sometimes it may not be possible to integrate beyond a value \( TSTOP \) (because of an essential singularity there, such as a failure of definition of the derivatives beyond that point). Then you must tell the code this. There are better ways of dealing with ordinary discontinuities – see section 6.

- Can the integration proceed without any upper limit on the independent variable?
  - YES – set \( INFO(4) = 0 \).
  - NO – set \( INFO(4) = 1 \) and define \( TSTOP \) by setting \( RWORK(1) = TSTOP \).
INFO(5) – To solve stiff or MAD systems it is necessary to use the Jacobian matrix of partial derivatives of the functions $F(X,U)$. If you do not provide a subroutine called JAC to evaluate it analytically, it will be approximated by numerical differencing. This is less trouble to you, and is usually very successful; it may or may not be cheaper, depending on your problem. For example, if your problem is linear, i.e. if it has the form

$$\frac{dU}{dX} = F(X,U) = J(X)U + G(X)$$

then it is cheaper, and little extra trouble, to provide a subroutine JAC to evaluate $J(X)$.

Do you want the Jacobian matrix evaluated by numerical differencing?

YES – set INFO(5) = 0 and supply DC03J/JD as argument JAC.

NO – set INFO(5) = 1 and provide JAC to evaluate $\frac{\partial F}{\partial U}$.

INFO(6) – If the Jacobian matrix is banded, with bandwidth much smaller then NEQ, or sparse, with relatively few non-zeros per row on average, then DC03A/AD will perform much better if it is told this. Storage needed in RWORK will be reduced (but in the sparse case some extra storage will be needed in IWORK), numerical differencing will be done more cheaply, and several subroutines will execute faster. Really large problems are only tractable if the Jacobian is sparse or banded, and treated as such.

If for each $i$ in the range 1 to NEQ equation $i$ does not involve solution components $U(j)$ for $j<i-ML$ or for $j>i+MU$, the system is said to have half-bandwidths ML and MU. If you tell the code this and if $2*ML+MU+1$ is much less than NEQ, the gain is well worth while, especially on large problems.

If the Jacobian is sparse but not banded, it will be necessary to tell the code which rows, in each column, contain non-zero elements; section 7 describes how to do this. The code need be told this only once for a given sparsity pattern, and does quite an expensive analysis of each new pattern; so even on a first entry it is worth using the pattern information left in IWORK from an earlier entry if the pattern is unchanged.

If you wish to take advantage of sparsity or bandedness and supply your own subroutine JAC, you must be careful to make it store the non-zeros in the correct form – see section 7 for the sparse case.

Do you want to use a full (dense) Jacobian matrix, not a sparse or banded one?

YES – set INFO(6) = 0 and supply DC03P/PD as argument PTN.

NO – banded – set INFO(6) = 1, supply DC03P/PD as argument PTN, and provide the half-bandwidths by setting

- IWORK(1) = ML
- IWORK(2) = MU.

NO – sparse – set

INFO(6) = 2 for a new pattern (and write your own PTN)

INFO(6) = 3 to re-use an old pattern and follow the instructions in section 7.

NO, – no Jacobian (problem not stiff or MAD) – set

INFO(6) = 4.

INFO(7) – DC03A/AD can also solve MAD problems, in which some components (the first NALGB) of $U$ are defined by algebraic equations

$$0 = F_i(X,U), i = 1 \text{ to } NALGB,$$

instead of differential equations

$$\frac{dU(i)}{dX} = F_i(X,U)$$

(In such cases, the initial values supplied in array Y for these components must be close to values satisfying the algebraic equations). INFO(7) is used to tell the code that such a MAD system is to be solved. (Subroutine F must still provide all NEQ functions $F$, but the code will iterate to make the first NALGB of them take zero values, using the others as derivatives of their corresponding components of $U$). MAD systems are harder to solve than purely differential systems, and are rare in practice, so the code has not been as thoroughly tested.
on real MAD systems as in the case of purely differential systems.

Is the set of equations purely differential (NALGB=0)?
YES – set INFO(7) = 0.
NO – set INFO(7) = NALGB.

INFO(8) – It is occasionally necessary to specify a maximum step-size HMAX to be used in the integration, to prevent the code from "stepping over" narrow features in the functions F(X,U) without "seeing" them. On most problems this is not necessary.

Are the functions F(X,U) smooth enough not to need an explicit limit on step-size?
YES – set INFO(8) = 0.
NO – set INFO(8) = NALGB.

INFO(9) – If you are sure you want to, you can completely change the calculation of scaling numbers W(I) (see INFO(2)), by changing the values of URFAC and UMFAC. For example, by making them both zero and supplying a USIZ vector, you can specify absolute error testing (i.e. relative to your specified USIZ(I)). Doing this may sacrifice much of the reliability built into the code.

Do you want to use the built-in default values of URFAC, UMFAC?
YES – set INFO(9) = 0.
NO – set INFO(9) = 1 and define HMAX by setting RWORK(2)=HMAX.

INFO(10) – Enables internal sub-arrays of RWORK, IWORK to be dimensioned MEQ > NEQ, for compatibility with special calling programs.

Do you want the default internal array size NEQ?
YES – set INFO(10) = 0.
NO – set INFO(10) = 1 and define MEQ by setting IWORK(4) = MEQ.

TOL You must specify the accuracy you want in the solution (relative to numbers W(I) described under INFO(2)) by setting TOL to a suitable positive value. If the value you specify is larger than 10^{-2} or smaller than a number TOLMIN (about 10^{-10} on most computers, but see section 10), it will be reset to the corresponding limit and the task will be suspended, the code returning to your calling program with IDID=-2. If you are sure you want to, you may yourself alter TOL before re-entry, but in general you should accept the change. TOL is used by the code in a local error test which requires roughly that (at point T)

$$|\text{local error}| \leq TOL \cdot (\max(\text{USIZ}, UMFAC \cdot (\max|U(X)|, X \leq T)) + URFAC \cdot |U(T)|)$$

for each component, where USIZ is taken as zero if you have chosen not to supply it. The default testing, with USIZ=0, has proved very reliable over a wide range of problems, and you should think carefully before deciding to change it. It has built-in refinements to deal with cases where a component of U has an initial value of zero or passes through zero during integration; this is why the inequality above is only approximate.

RWORK Dimension this array of length LRW in your calling program. If you have set INFO(4), INFO(8) or INFO(9) to 1, you must set the corresponding entries of RWORK as prescribed.

LRW Set it to the length of array RWORK. Exceptionally, on a problem which is neither MAD nor stiff, DC03A/AD may never use the Jacobian matrix, and in this case you must have

\[ LRW \geq 50 + NEQ \cdot 11 \]

Otherwise, for a full Jacobian matrix (INFO(6) = 0), you must have

\[ LRW \geq 50 + NEQ \cdot (2 \cdot NEQ + 11) \]

while for a banded matrix (INFO(6) = 1) you must have
\[
\text{LRW} \geq 50 + \text{NEQ}^* (3^{*}\text{ML} + 2^{*}\text{MU} + 13)
\]
where ML, MU are the half-bandwidths. For a sparse matrix, you must have
\[
\text{LRW} \geq 50 + \text{NEQ}^* 11 + \text{NJAC} + \text{NLUD}
\]
where NJAC is the number of non-zeros in the Jacobian sparsity pattern, and NLUD is the number in the LU decomposition of the iteration matrix (which has the same pattern as the Jacobian but with diagonal elements present for \(i > \text{NIALGB}\)). Even if NJAC is known in advance, NLUD is difficult to predict because of fill-in during decomposition. Therefore, it is wise to set LRW large for the first run of a new problem. The code always returns in IWORK(2) the maximum number of locations actually used in array RWORK, so the space needed for a later run is known.

IWORK  Dimension this array of length LIW in your calling program. If you have set INFO(6) to 1, you must set the corresponding entries of IWORK as prescribed.

LIW  Set it to the length of array IWORK. You must have
\[
\text{LIW} \geq 60
\]
if (exceptionally, see under LRW) the Jacobian matrix is not used, but normally
\[
\text{LIW} \geq 60 + \text{NEQ}
\]
for a full or banded Jacobian matrix. For a sparse matrix, you must have
\[
\text{LIW} > 76 + \text{NEQ}^* (2 + 13) + (\text{NJAC} + \text{NLUD} + \text{MLUD})
\]
where NJAC and NLUD are defined above, and MLUD is a number around halfway between them; As for LRW, it is wise to set LIW large for the first run on a new problem. The code always returns in IWORK(25) the maximum number of locations actually used in IWORK.

RPAR, IPAR  If you need to pass REAL (DOUBLE PRECISION in the D version) and/or INTEGER data through DC03A/AD to your F subroutine (and to your JAC and/or PTN subroutine, if supplied), set it in these arrays, and dimension them accordingly in your calling program and in F (and JAC and/or PTN). If you do not need them, treat them as dummy arguments. They are not used or altered by DC03A/AD.

PTN  If you have not set INFO(6)=2, you can supply the dummy subroutine DC03J (DC03JD for the double precision version) as this argument. If you have set INFO(6)=2, you must supply a subroutine of the form:
\[
\text{PTN}(\text{LENC}, \text{NEQ}, \text{IRN}, \text{LIRN}, \text{NIRN}, \text{RPAR}, \text{IPAR})
\]
The arguments and function of this subroutine are described fully in section 7. In any case, you must declare this argument EXTERNAL in your calling program.

JAC  If you have set INFO(5)=0, you can supply the dummy subroutine DC03P (DC03PD for the double precision version) as this argument. If you have set INFO(5)=1, you must provide a subroutine of the form:
\[
\text{JAC}(\text{X}, \text{U}, \text{PD}, \text{NROW}, \text{LENC}, \text{IRN}, \text{RPAR}, \text{IPAR})
\]
In any case, you must declare this argument EXTERNAL in your calling program. For the given values of \(X\) and the vector \(U=\{U(1), U(2), \ldots, U(\text{NEQ})\}\), this subroutine must evaluate the partial derivatives \(\partial F(I)/\partial U(K)\) for each \(I = 1\) to \text{NEQ} and for each \(K = 1\) to \text{NEQ}, and store these in the array PD in appropriate form. Only non-zero elements need to be defined, as \(D\) is set to zero before calling JAC.

For the full case (INFO(6)=0), NROW will have the value \text{NEQ}; you must dimension PD (NROW, 1) and must set
\[
\text{PD}(I,K) = \partial F(I)/\partial U(K)
\]
For the banded case (INFO(6)=1), NROW will have the value \((\text{ML}+\text{MU}+1)\); you must dimension PD (NROW, 1) and must set the partial derivatives in it according to
\[
\text{IROW} = I - K + \text{MU} + 1
\]
PD (IROW, K) = ∂F(I)/∂U(K)
for each I = max(1, K-MU) to min(NEQ, K+ML), for each K = 1 to NEQ.

In both the above cases, JAC must not alter X, U, NROW, LENC, IRN; the last two are not needed, but are included for compatibility with the sparse case below.

For the sparse case (INFO(6) = 2 or 3) NROW will have the value NIRN returned by PTN (see section 7); you must dimension PD (NROW), and also LENC (NEQ), IRN (NROW), and must set the nonzero partial derivatives in PD to correspond with the sparsity pattern held in LENC and IRN (see section 7). JAC must not alter X, U, LENC, IRN or NROW.

You can use RPAR, IPAR for communication between your calling program and JAC just as for F. If you do not choose to use them, treat them as dummy arguments; if you do, dimension them appropriately.

It is perhaps worth emphasising that it is usually not necessary to provide JAC, and (especially in the sparse case) you should try not doing so unless you are very confident.

4. Output on Return From DC03A/AD

The main task of the code is to return a computed solution at TOUT, although you can get intermediate results on the way. To find out whether the code finished this task or if the integration process was interrupted before completion, you must check the value of IDID. On return, parameters have values

T the solution was successfully advanced to the output value of T.

Y contains the calculated approximation to the solution U at T.

YDER The approximate derivatives dU/dX at T are contained in array YDER. They are always obtained by differentiating the interpolation formula used internally, never from subroutine F, whose output values would be very sensitive, on a stiff problem, to acceptable errors in U.

IDID reports what the code did:

Task completed reported by positive values of IDID.

IDID = 1 - A step was successfully taken in the intermediate-output mode. The code has not yet reached TOUT.

IDID = 2 - The integration to TOUT was successfully completed (T=TOUT) by stepping exactly to TOUT.

IDID = 3 - The integration to TOUT was successfully completed (T=TOUT) by stepping past TOUT. The entries in Y are obtained by interpolation.

Task interrupted, but can be restarted reported by negative values of IDID.

IDID = -1 - A large amount of work has been done (500 steps attempted).

IDID = -2 - The error tolerance TOL has been changed to bring it within range.

IDID = -3 - You omitted to change TOUT after it had been reached, or you specified TOUT = T on a new problem.

IDID = -4 - The values of LRW and/or LIW supplied were too small.

IDID = -5 - Not used.

IDID = -6 - DC03A/AD had repeated convergence test failures on the last attempted step.

IDID = -7 - DC03A/AD had repeated error test failures on the last attempted step.

IDID = -8,...,-32 - Not used.

Task terminated

IDID = -33 - The code has met trouble from which it cannot recover. A message is printed explaining the
trouble, and control is returned to the calling program. For example, invalid input data has been detected.

**TOL** Remains unchanged except when \( IDID=-2 \), when **TOL** has been altered to bring it within range.

**RWORK, IWORK** contain information which is often of no interest to the user, but is necessary for subsequent calls and must be left undisturbed. Copies of some of this information, with mnemonic names, are held in COMMON block **DC03Y/YD** (see section 10). The following values, whose mnemonic names are also shown, may be of interest:

- **RWORK(6)** - contains **RELR**E, the ratio of error estimate on the last step to error tolerance.
- **RWORK(8)** - contains **STIFNS**, a number measuring roughly how stiff the differential equations are (algebraic equations in a MAD system can be thought of as infinitely stiff).
- **RWORK(11)** - contains **HNEW**, the proposed length of the next step.
- **RWORK(12)** - contains \( X_0 \), the value of \( X \) at the start of the last completed step.
- **RWORK(13)** - contains **XX**, the current value of \( X \), at the end of the last completed step.
- **RWORK(14)** - contains **H**, the length of the last completed step.
- **IWORK(24)** - contains the number of locations used in **RWORK** (or the estimated number needed, if \( IDID=-4 \)).
- **IWORK(25)** - contains the number of locations used in **IWORK** (or the estimated number needed, if \( IDID=-4 \)).
- **IWORK(K+25)** - for \( K = 1 \) to 6 contain statistical counts **KOUNT(K)**:
  - \( K=1 \): Number of completed steps.
  - \( K=2 \): Number of attempted steps.
  - \( K=3 \): Number of derivative calls to subroutine \( F \).
  - \( K=4 \): Number of Jacobian calls, to subroutine \( JAC \) or by numerical differencing.
  - \( K=5 \): Number of LU decompositions of the Newton iteration matrix, \( (I-\gamma H J) \), where \( \gamma \) is a number of order 1.
  - \( K=6 \): Number of starts and error re-starts.
- **IWORK(32)** - contains **KDEQ**, the order of integration formula used on the last step.

5. How to Continue the Integration by Calls After the First

The code is organised so that calls after the first involve little (if any) additional effort on your part. You must monitor **IDID** to determine what to do next. In the usual case, all you need do is specify a new **TOUT** when the current one is reached.

Do not alter any quantity not specifically permitted below. In particular, do not change \( NEQ, T, Y, RWORK, IWORK \), or the differential equations as specified by \( F \); a change to **RPAR** or **IPAR**, if used, counts as a change to the differential equations. Any such change constitutes a new problem, and must be treated as such by a new first call, even if in fact the same problem is being continued after a discontinuity. (However, it is possible to restart after such a change with \( INFO(1)=2 \), instead of 0; the only difference is that the statistical counts in **IWORK(J+25)**, \( J=1 \) to 6, are not reset to zero).

The values **USIZ(I)**, if used (**INFO(2)=1**), are referenced only on the first call, and so cannot effectively be changed. The same is true of any non-default values of **URFAC**, **UMFAC** if provided (**INFO(9)=1**). However, you can change **TOL** at any time; increasing it may make the equations easier to integrate, but decreasing it makes them harder to integrate and so should usually be avoided. You can also change the value of **HMAX**, if you set **INFO(8)=1**.

You can switch the intermediate-output mode (**INFO(3)**) on or off at any time. If you have set **TSTOP** (**INFO(4)=1**), the code will refuse to integrate past the current **TSTOP**. Once it has been reached, you must change **RWORK(1)** or set **INFO(4)=0** in order to continue. You may do this at any time, but whenever you set **INFO(4)=1** you must supply **TSTOP** in **RWORK(1)**.
The parameter $INFO(1)$ determines whether the call is the first of a new problem (including a restart after a discontinuity) or a continuation call. You must set $INFO(1)$ to 0, or to 2 for a restart without resetting the statistical counts, to start a new problem; on return, $INFO(1)$ will have the value 1, to indicate a continuation call, unless the task was interrupted. You can therefore continue a completed task without changing $INFO(1)$, but must reset it yourself to continue an interrupted task, or to start a new problem.

If you are using the sparse Jacobian option ($INFO(6)=2$), the code will reset $INFO(6)$ to 3 after it has analysed the sparsity structure; thus, if a new problem has the same structure (e.g. if it is a restart after a discontinuity), you need not change $INFO(6)$, but if the structure is different you must reset it to 2.

After a completed task:

If

- $IDID = 1$ - call the code again to continue the integration towards $TOUT$, or change $TOUT$ to get interpolated output within the step just completed.
- $IDID = 2$ or $3$ - define a new $TOUT$ and call the code again. $TOUT$ must be different from $T$. If the new $TOUT$ falls within the current step (from $RWORK(9)$ to $RWORK(13)$), output at $TOUT$ will be obtained by interpolation without any more integration. If $TOUT$ exceeds $RWORK(13)$, at least one more integration step will be attempted. If $TOUT$ precedes $RWORK(9)$, or exceeds $TSTOP$ if set, the task will be terminated.

After an interrupted task:

To show the code that you realise that the task was interrupted and nevertheless you want to continue, you must take appropriate action and reset $INFO(1)$. If you do not reset $INFO(1)$, the task will be terminated.

- $IDID = -1$ - the code has attempted 500 steps. If you want to continue, set $INFO(1)=1$ and call the code again. A further 500 steps will be allowed.
- $IDID = -2$ - the error tolerance $TOL$ has been changed to bring it within range. You may change it yourself if you are sure that your value will work. To continue with the changed value, set $INFO(1)=1$ and call the code again.
- $IDID = -3$ - you omitted to change $TOUT$ after it had been reached. Set a new $TOUT$, set $INFO(1)=1$ and call the code again.
- $IDID = -4$ - the values of $LRW$ and/or $LIW$ were inadequate to handle the Jacobian matrix on a non-MAD problem (on a MAD problem, lack of space for matrix handling causes task termination). If the problem is only mildly stiff, it may be worth continuing, at reduced step size, without Jacobian matrix handling. If you are sure you want to do this, set $INFO(6)=4$, set $INFO(1)=1$ and call the code again. You may set $INFO(6)=4$ from the start, if you are prepared to accept the efficiency cost of not using the Jacobian matrix.
- $IDID = -6$ - repeated convergence test failures occurred. Inaccuracy of the Jacobian matrix may be the problem. If you are sure you want to continue, perhaps at greatly reduced efficiency, do a restart by setting $INFO(1)=2$ and calling the code again.
- $IDID = -7$ - repeated error test failures occurred on a single step. There may be unrecognised discontinuities in the equations, or your $F$ subroutine may be providing noisy derivative values (e.g. by failure to declare some variable DOUBLE PRECISION in the D version). If you are sure you want to continue, do a restart by setting $INFO(1)=2$ and calling the code again.

After a terminated task

If

- $IDID = -33$ - you cannot continue this problem. An attempt to do so will cause your job to be terminated.
6. Advanced Output Testing: Handling Discontinuities

You may want output when some function of the solution \( U \) and the independent variable \( X \) takes a certain value, e.g. when \( G(X,U)=0 \) for a known function \( G \). As an example, suppose you want output where \( U(1)=X\times U(3) \). Start integrating in intermediate-output mode, with any reasonable \( TOUT \). At each return, monitor \( G(U)=U(1)-X\times U(3) \) to see if it has changed sign; if not, call the code again, first increasing \( TOUT \) if \( IDID=2 \) or 3. When you find that \( G(U) \) has changed sign in the most recent step, carry out an inverse interpolation to find a value \( T \) of \( X \) at which \( G(U)=0 \); this has to be done iteratively. When the point has been found to sufficient accuracy (e.g. to within \( TOL\times H \)), the necessary output action can be taken. We discuss below how to do the inverse interpolation.

This technique may be used to handle ordinary discontinuities in the \( F \) functions; by ordinary discontinuities we mean those where a smooth continuation does exist, but is not followed because of a change of definition or of some parameter value. A switching function \( G(X,U) \) is constructed, such that the discontinuity occurs at \( G=0 \). Using the old \( F \) coding (including the smooth continuation past the switching point), locate the latter by the above technique. Then make the necessary discontinuous change, set \( INFO(1)=2 \) to force a re-start, and call the code again; the restart will occur from the point in the smooth solution at which the switching function becomes zero, which is exactly what is required. This is the most efficient means of dealing with this kind of discontinuity, but it is by no means necessary to use it in all cases. \texttt{DC03A/AD} can in fact integrate through ordinary discontinuities or near-discontinuities without serious loss of efficiency in most cases.

To do the necessary inverse interpolation we can use Newton’s method. Consider our example function, \( G=U(1)-X\times U(3) \). We can evaluate

\[
\frac{dG}{dX} = \frac{dU(1)}{dX} - U(3) - X\times \frac{dU(3)}{dX}.
\]

As applied to interpolated values within the current step, the values of \( dU(I)/dX \) stored in \( YDER \) are exact; thus we can use them in the above expression. The Newton estimate for \( X \) where \( G=0 \) is

\[
X=TOUT-(G/(dG/dX))_{X=TOUT}.
\]

If this lies within the range \( RWORK(9) \) to \( RWORK(13) \), we set it as a new \( TOUT \) and call the code again to get interpolated values at \( X \), iterating until the change is within tolerance. If we find that \( X \) is outside the current step, we carry out a binary subdivision of either \( (RWORK(9),TOUT) \) or \( (TOUT,RWORK(13)) \), choosing the sub-interval in which \( G \) changes sign, until we find a point at which the Newton iterate is within the step, or until the subdivision is already fine enough to be within tolerance.

7. The sparse matrix interface

This interface consists of eight subroutines of the package, whose names are all of the form \texttt{DC03x} (or \texttt{DC03xD} for the double precision version), where \( x = Q, R, S, T, U, V, N \) or \( X \). The \( Q, V \) and \( N \) subroutines call subroutines of the \texttt{HSL MA30} sparse matrix package, and together with this package these subroutines make up quite a lot of code. This space can be saved by replacing these subroutines by dummies, if you wish to use only the full or banded option. There is also a dummy subroutine \texttt{DC03P/PD}, which you can name as argument \texttt{PTN} if you are not using the stiff option.

If you do wish to use the sparse option, you must supply a subroutine called \texttt{PTN} (the name you gave as the last argument but one of the call to \texttt{DC03A/AD}) to specify the pattern of non-zero elements of the Jacobian matrix of your system. Experience has shown that trying to find this pattern by changing one component of \( U \) at a time, and calling \( F \) to see which derivative components change, can be unreliable, as well as expensive. This is because, in many problems, many solution components start from zero values with zero initial slope; such a component gives no indication of its scale, so an automatic method cannot choose a good increment. Accordingly, this method is not recommended for your use either, unless you are sure you know how to choose suitable increments.

The subroutine you write must be of the form

```
SUBROUTINE PTN(LENC, NEQ, IRN, LIRN, NIRN, RPAR, IPAR)
```

where
LENC is an INTEGER array of size NEQ, in which the subroutine must set LENC(I) to the number of non-zeros in column I of the Jacobian (i.e. the number of components of F which depend on U(I)).

NEQ is an INTEGER variable set to the number of equations, as supplied to DC03A/AD.

IRN is an INTEGER array of length LIRN, into which the subroutine must put the row numbers of the non-zero elements; first those of column 1 in increasing order, then those of column 2 in increasing order, and so on. It is quite permissible, and not unusual, for a column to have no non-zero entries; then the corresponding entry of LENC should be set to zero, and no gap should be left between entries in IRN.

LIRN is the length of array IRN. It must not be changed by the subroutine.

NIRN is an INTEGER which the subroutine must set to the number of non-zero elements, i.e. to the sum of the values in LENC. If LIRN is too small, the subroutine should set NIRN to a number greater than LIRN (if possible, to an estimate of the space needed), and must not, of course, try to set elements of IRN beyond IRN(LIRN).

RPAR, IPAR are the REAL (DOUBLE PRECISION in the D version) and INTEGER arrays you passed to DC03A/AD, which you can use, if you want, for communication between your calling program and PTN.

If you supply your own JAC subroutine to calculate the Jacobian in the sparse case, the arguments LENC, IRN passed to it will be as computed by your PTN subroutine, while the argument NROW will have the value NIRN returned by PTN. If IRN(K) refers to a non-zero in column J, i.e. if

\[ \sum_{L=1}^{J-1} \sum_{L=1}^{J} \text{LENC}(L) < K \leq \sum_{L=1}^{J} \text{LENC}(L), \]

then PD(K) must be set according to

\[ I = \text{IRN}(K) \]
\[ \text{PD}(K) = \frac{\partial F(I)}{\partial U(J)}. \]

To sum up, if you do not intend to use the sparse matrix option you can save a lot of space by replacing eight of the subroutines in the DC03 package by dummies; if you do use it, then as well as setting INFO(6)=2 you must provide your own subroutine PTN to specify in the form described above, which elements of the Jacobian matrix are non-zero.

8. Error messages and other communication

Explanatory messages are put out before task termination, and also on failure to update TOUT. This is done on the Fortran unit specified by integer LP in COMMON block DC03Z/ZD, whose default value is 6; setting LP to zero switches off these messages. All the task termination messages finish with

***** DC03A TASK TERMINATED FOR ABOVE REASON

Most of the messages which precede the above termination message are self-explanatory, but two may need some comment:

***** SINGULAR OR ILL-CONDITIONED MATRIX

The Newton iteration matrix could not be successfully decomposed by the appropriate library subroutine; the author has no experience of this occurring on properly posed problems.

***** DC03A/AD ERROR 1 - PLEASE REPORT

The sparse linear algebra code has incorrectly been entered after the user has set INFO(6)=4 to specify no Jacobian matrix; this is a logical error which, it is believed, cannot occur.

No message is put out before aborting the user’s job if he attempts to continue a terminated task.

The message put out if the user fails to update TOUT (before return with IDID=-3) is self-explanatory.

Calls are made to a communication subroutine, DC03I/ID, at various significant stages during the integration process; these are mainly intended to facilitate integration of the package into the FACSIMILE code, and probably

All use is subject to licence.
the user will not need to concern himself with them. The subroutine DC03I/ID supplied with the package is a dummy. It is called with a single INTEGER argument, whose value on entry defines the occasion for the call; the subroutine must not change this argument. The significance of the argument values is as follows:

1. A trial starting step size has been chosen.
2. The trial starting step size has had to be reduced.
3. A very small lower bound on step size causes TOL relaxation.
4. A subsequent increase in step size removes TOL relaxation.
5. The Jacobian sparseness pattern has been found by PTN.
6. The pivotal sequence has been chosen for sparse LU decomposition.

It is possible to replace DC03I/ID by a subroutine which can give further information to a calling package, by examining COMMON block DC03Y/YD, on each of the above occasions.

9. General Information

Use of common: COMMON blocks DC03Y/YD, DC03Z/ZD, of which the latter is initialised in BLOCK DATA. Also refers to COMMON blocks MA30E/ED (in which it sets LP, the message unit number, to zero) and MA30F/FD of the MA30 sparse matrix package.

Workspace: As described.

Package subroutines: All names of the form DC03x/xD are used as subroutine or COMMON block names, where x is a letter other than J, K, or L. A list is given in section 10.

Other routines called directly: Calls FD05A/AD, _GEFA, _GBFA, _GESL, _GBSL, MA30A/AD, MA30B/BD, MA30C/CD, and user-supplied subroutines F and JAC.

Input/output: See section 8.

10. COMMON Block and Subroutine Names

The contents of the two COMMON blocks are as follows:

COMMON /DC03Z / ZERO,ONE,TWO,TEN,PT1,PT5,ROUND,TOLMIN
1,VARMIN,VSMAL,VMAX,BIASD,BIASL,BIASU,ETAFAC,CNVFAC
2,RMINIT,RMCONV,RATMIN,HFINIT,HFRUN,RHOMAX,RHOFAC,TSTINT
3,KMAXX,NRETRY,LP,NNI2

These are various absolute or method-determining constants, set in a BLOCK DATA subprogram, of which only the following may possibly need to be changed:

ROUND, VARMIN, VSMAL, VMAX have values set by calls to subroutine FD05. These values may be computer-dependent in the sense that the precision implied by ROUND, or the range implied by the others, may not be attainable on some computers. The temptation to widen the range because the computer in use makes it possible should be resisted in general – it can be very expensive on problems where many solution components start from zero initial values with zero initial slopes. The value of TOLMIN, the tightest normally acceptable tolerance, should be about $10^5$ times ROUND. VARMIN is a lower bound on absolute values of solution components, below which they are treated as zero and error testing on them is omitted. VSMAL is used for two purposes: as a fictitious size for such small components when calculating Jacobians by finite differencing; and as an initial lower bound for error testing on these components, when they first exceed VARMIN – the testing gradually relaxes to the relative error test already described. The value used for VSMAL has been found to be a good compromise over a wide range of problems. If it was smaller, the error testing on such initially zero components would be more stringent, but the range of component size for which relative error control is used would be wider.

KMAXX may be reduced (from 5) to reduce the maximum order of BDF integration formula used; it should not be increased.
LP (=6) specifies the output unit number for diagnostic messages, and may be changed if necessary; setting it to zero switches off the messages.

NII2 is set to 1.

```
COMMON /DC03YD/ TSTOP, HMAX0, URFAC, UMFAC, SPARES
1, RELERR, BSCALE, STIFNS, HFAC, HMIN, HNEW, X0, XX, H, RATMAX
2, ACTOL, ETA, HDOWN, HSAME, HUP, AOH, AO, FKDI, PD, QD, PL, QL, PU, QU
3, ECON, ECON1, TSTFAC, ERNORM, veck(7), HMAX, HREC
4, ML, MU, NALGB, MZNOR, ISPAR5, NROA, LSTRT, NROB, LSTILL, NCORR
5, KZNOR, KSPARE, KYTST, KDSAV, KVEC1, KYLIM, KLUD, KJAC, LRUSED
6, LIUSED, KOUNT(6), KDEQ, IUPH, ICORR, NSCONV, LJAC, LLUD, LNEWT
7, JACSIZ, LUDSIZ, KIPTR, NY, KMAX, KMAX1, JSWICH
8, KLENB, KICNB, KICNG, KICGP, KICN, LENOFF(1), KLENR, KLENRL
9, KIPA, KIQA, IDISPX(2), NCG, KSPR1, KSPR2, NALG1, IFLAG, KD1
```

The main purpose of this block is to give mnemonic names to variables held in arrays RWORK and IWORK; this is done by saving these variables in the arrays before each return to the calling program, and restoring them on re-entry. The first 40 variables (up to and including VECK) correspond to those of RWORK; the remaining reals (HMAX and HREC) are reset at the start of each integration step. The first 45 integers correspond to those of IWORK, and the next 15, which are used only for the sparse option, to IWORK(46) to IWORK(60) are used to hold a copy of array INFO. The variables which may be of interest to the user have been described in section 4.

The package contains 22 subroutines, whose single-precision names and functions are as follows:

- **DC03A/AD** checks input giving error returns if necessary, implements options in INFO, calls DC03D/DD to initialise on first entry, calls DC03B/BD for each integration step, tests for output on return, calls DC03C/CD to interpolate within completed step at TOUT.

- **DC03B/BD** carries the integration forward one step (unless it returns for 500 attempted steps, or for space shortage first time Jacobian is used). On entry, it changes stepsize and order, if necessary, to accord with new values chosen on previous step; this enables TSTOP and HMAX constraints to be observed, even if the user has changed them.

- **DC03C/CD** interpolates within a step just completed to get values at TOUT.

- **DC03D/DD** initialises at the start of a run. For non-MAD problems, this is mainly a matter of setting up an initial Nordsieck array for a first-order method, checking whether a stepsize reduction is desirable. For MAD problems, a preliminary iteration adjusts the algebraic variables, if necessary, to satisfy their defining equations, and then computes their initial rates of change.

- **DC03E/ED** resets the Nordsieck array to start-of-step values, before the stepsize is reduced following step failure.

- **DC03F/FD** tests HNEW, a tentative new stepsize, against bounds HMIN (an extremely small number) and, if applicable, an upper bound; sets it within these limits if outside. The upper bound is chosen to reflect any HMAX and/or TSTOP specifications.

- **DC03G/GD** is entered at the start of each step to set up numbers depending on the order to be used on the step.

- **DC03H/HD** solves the linear equations for each correction, when Newton iteration is in use. For the full or banded case, LINPACK subroutine _GESL or _GBSL is called; for the sparse case, package subroutine DC03W/WD.

- **DC03I/ID** is the behind-the-scenes communication subroutine already described.

- **DC03J/JD** is a dummy subroutine which the user may use, if he wants the standard option of finite-difference Jacobian calculation, as the argument JAC of DC03A/AD.

- **DC03K/KD** computes the LU decomposition of a new Newton iteration matrix \((I - \gamma HJ)\), where \(\gamma\) is a number of order 1, \(h\) is the stepsize, and \(J\) is the Jacobian matrix (except that rows corresponding to algebraic variables, if any, are just those of \(-J\)). If a valid Jacobian is available, it uses it; if DC03B/BD has marked the current Jacobian...
outdated, it calls DC03N/ND to evaluate a new one. For the full or banded case, it calls the LINPACK subroutine
_GEFA or _GBFA to carry out the decomposition. For the sparse case, if a pivotal sequence has not yet been
chosen it first calls DC03Q/QD to choose one, and calls DC03V/VD to carry out the decomposition.

DC03N/ND computes the Jacobian matrix. In the sparse case, if the sparsity pattern is not yet known it first calls
DC03U/UD to find it. It tests INFO(5), calling JAC if specified; otherwise it sets up finite difference increments
and calls DC03O/OD for the full or banded case, or DC03X/XD for the sparse case.

DC03O/OD computes a full or banded Jacobian matrix by finite differences. In the banded case only (ML+MU+2) calls
to F are needed, compared with (NEQ+1) in the full case.

DC03P/PD is a dummy subroutine which the user may use, in the non-sparse case, as the argument PTN of
DC03A/AD.

DC03Q/QD computes the pivotal sequence for decomposition of a sparse Newton iteration matrix. It first calls
DC03S/SD to construct a skeleton matrix, which has the full sparsity pattern, but (except for any algebraic
submatrix) has numerical values of a unit matrix. It then calls MA30A, which is thus constrained to choose
pivots only from the diagonal (but in an order chosen to preserve sparseness) except in the algebraic submatrix
(if any), where diagonal elements may be missing.

DC03R/RD loads a new Newton iteration matrix, ready for LU decomposition by MA30B using the pivotal sequence
found by MA30A.

DC03S/SD loads a skeleton Newton iteration matrix ready for LU decomposition by MA30A.

DC03T/TD sorts the columns of a sparse Jacobian matrix into groups, such that all columns in a group may be
simultaneously evaluated by finite differences.

DC03U/UD calls PTN to evaluate the sparsity pattern of the Jacobian matrix, then calls DC03T/TD (unless the user has
set INFO(5)=1) to choose a column grouping for efficient numerical evaluation.

DC03V/VD calls DC03R/RD to load a new sparse Newton iteration matrix, and then calls MA30B to decompose it.

DC03W/WD simply calls MA30C to solve for a new Newton iterate in the sparse case; it exists to buffer the MA30 sparse
matrix package from DC03H/HD, so that the user can avoid having MA30 loaded if he wishes.

DC03X/XD computes a sparse Jacobian by finite differences.

11. References