1 SUMMARY

HSL_DC05 is a suite of Fortran 95 procedures for the solution of a system of ordinary differential equations (ODE) or differential algebraic equations (DAE) of index 1:

$$\mathbf{F}(t,\mathbf{Y},\mathbf{Y}') = \mathbf{0}, \tag{1.1}$$

where $\mathbf{Y}' = \mathrm{d}\mathbf{Y}/\mathrm{d}t$. It provides a powerful optional method of avoiding the incorrect occurrence of negative values for solution components that should remain positive throughout. Some of the components of $\mathbf{F}$ and $\mathbf{Y}$ are purely algebraic; the algebraic equations do not involve $\mathbf{Y}'$ and the rest do not involve the derivatives of the algebraic components of $\mathbf{Y}$. If there are $N_{\rm alg}$ such algebraic equations and variables, we can split $\mathbf{Y}$ into two vectors: $\mathbf{Y}_1$ of length $N_{\rm alg}$ and $\mathbf{Y}_2$ of length $(N_{\rm eq} - N_{\rm alg})$, where $N_{\rm eq}$ is the total number of equations (algebraic and differential). In turn, equation (1.1) can be written as:

$$\mathbf{F}_1(t,\mathbf{Y}_1,\mathbf{Y}_2) = \mathbf{0}, \tag{1.2}$$
$$\mathbf{F}_2(t,\mathbf{Y}_1,\mathbf{Y}_2,\mathbf{Y}_2') = \mathbf{0}, \tag{1.3}$$

where $\mathbf{F}_1$ and $\mathbf{F}_2$ are vectors of length $N_{\rm alg}$ and $(N_{\rm eq} - N_{\rm alg})$, respectively. Components $\mathbf{Y}_1$ of $\mathbf{Y}$ are purely algebraic, their derivatives $\mathbf{Y}_1'$ being absent. The system must be of index no higher than 1, i.e. it must be non-singular in the sense that equation (1.2) can in principle be solved for $\mathbf{Y}_1$ and equation (1.3) for $\mathbf{Y}_2'$. HSL_DC05 cannot solve DAE systems of index greater than 1, which present added difficulties not present in those of index 1. The extension to the case where the algebraic components are not ordered before differential ones is explained in Section 2.11.

A special case, which often occurs in practice, is when equation (1.3) can be inverted to find a solution for the derivatives and the equations (1.2) and (1.3) take the explicit form:

$$\mathbf{G}_1(t,\mathbf{Y}_1,\mathbf{Y}_2) = \mathbf{0}, \tag{1.4}$$
$$\mathbf{G}_2(t,\mathbf{Y}_1,\mathbf{Y}_2) - \mathbf{Y}_2' = \mathbf{0}. \tag{1.5}$$

Another special (and common) case occurs when there are no algebraic variables, $N_{\rm alg} = 0$. In this case, we have an Implicit Ordinary Differential Equation (IDE) system:

$$\mathbf{F}_2(t,\mathbf{Y}_2,\mathbf{Y}_2') = \mathbf{0}. \tag{1.6}$$

Equations of this type are classified as DAE problems of index 0. Equation (1.6) will often in practice take the simple explicit form:

$$\mathbf{G}_2(t,\mathbf{Y}_2) - \mathbf{Y}_2' = \mathbf{0}. \tag{1.7}$$

1.1 Information needed to integrate the equations

The solution starts from given initial values of $\mathbf{Y}_2$ at $t = T$, using a variable order Backward Differentiation Formula (BDF) method, also known as Gear's method, Gear (1971), of order 1 to 5. The method automatically chooses step-size (h) and order of integration formulae (k), and is especially efficient on stiff systems in particular those arising from mass action kinetics. HSL_DC05 has been designed to handle DAE problems, which can be regarded in some sense as limiting cases of ODE problems of infinite stiffness. The methods used in HSL_DC05 are not too inefficient

to be acceptable on many non-stiff problems. Function evaluation and linear algebra are carried out in the calling program by using 'reverse communication'.

Within each integration step, the variables $\mathbf{Y}$ are represented as polynomials in t of degree k, $k = 1$ to 5. The algorithm for taking a step (of length h from $t = x_{r-1}$ to $t = x_r = x_{r-1} + h$) updates these polynomials, making use of equations (1.2) and (1.3) by satisfying them at the end of the new step. As these equations are in general nonlinear, this requires iteration. Newton iterations are used; the calling program will therefore need to form the Newton Iteration matrix $\mathbf{W}$:

$$\mathbf{W} = \begin{pmatrix} \mathbf{J}_{11} & \alpha\mathbf{J}_{12} \\ \\ \mathbf{J}_{21} & \alpha\mathbf{J}_{22} + \beta\mathbf{K}/h \end{pmatrix}, \tag{1.8}$$

where the Jacobian matrices are given by:

$$\mathbf{J}_{ij} = \frac{\partial \mathbf{F}_i}{\partial \mathbf{Y}_j} \quad \text{and} \quad \mathbf{K} = \frac{\partial \mathbf{F}_2}{\partial \mathbf{Y}_2'} \tag{1.9}$$

and α, β and h are scalars returned by HSL_DC05 to the calling program. The scalar α is equal to 1 normally, but is set to zero during iterations on start-up to correct the initial guess for $\mathbf{Y}_1$. The scalar β is always of order unity, while h is the current integration step-length and may vary through many orders of magnitude. The calling program therefore needs to supply the Jacobians. The matrix $\mathbf{W}$ is used in Newton iterations, solving the correction equations:

$$\mathbf{W}\delta\mathbf{C} + \mathbf{F} = \mathbf{0} \tag{1.10}$$

for a correction vector $\delta\mathbf{C}$ that is returned to HSL_DC05. The calling program must evaluate $\mathbf{F}$, but only $\delta\mathbf{C}$ is passed to HSL_DC05. Note that for an explicit DAE system or for a simple explicit ODE system, the function $\mathbf{F}$ is equivalent to $\mathbf{G}(t, \mathbf{Y}) - \mathbf{Y}'$. The linear algebra method used to solve equation (1.10) can be chosen to reflect the structure of $\mathbf{W}$.

There is a facility for restarting after a difficulty. All stored information about $\mathbf{Y}$ is discarded except for the current value of $\mathbf{Y}_2$ and estimates of $\mathbf{Y}_1$ and $\mathbf{Y}_2'$.

HSL_DC05 also contains subsidiary algorithms for:

1. testing the convergence of these iterations;
2. testing the accuracy of a converged step;
3. changing step-size h and/or order k, to improve the rate of progress while maintaining accuracy;
4. calculating output values of $\mathbf{Y}$ and $\mathbf{Y}'$ within a completed step; and
5. obtaining or correcting starting values of $\mathbf{Y}_1$ and $\mathbf{Y}_2'$.

The last is, of course, necessary only on first starting (or on restarting after a discontinuity). The user's initial guesses of these quantities are unlikely to satisfy equations (1.2) and (1.3) to sufficient accuracy to avoid injecting an unwanted initial transient into the solution, with possible persistent failure to pass the built in accuracy tests on the first few steps.

1.2 Problem Suitability

The package is designed to handle DAE problems, which can be regarded in some sense as limiting cases of ODE problems of infinite stiffness. This means that it can also handle very efficiently stiff ODE problems. Stiffness is an operational concept indicating a particular class of problems. There are many problems in physics and, especially, in chemistry (for example, mass action kinetics) which exhibit stiffness. The solution develops in time in a perfectly stable way, in the sense that solutions starting from slightly different initial conditions do not diverge from each other,

but rather tend to coalesce as time increases (it is characteristic of these problems that there is no inherent upper limit to the time for which they may develop, unless the process is artificially interrupted). Any small perturbation tends to die out with increasing time. In many such problems, the rate at which perturbations decay varies very greatly from one type of perturbation to another and although the system being modeled is highly stable, many numerical calculation methods which may be used to model it may be unstable unless very small integration step sizes are used. The integration can therefore become computationally very expensive unless a method specifically designed for use with stiff problems is used.

HSL_DC05 is highly suitable for stiff ODE problems, for example those arising from mass action kinetics, either alone or, using the Method of Lines, with conservatively discretized diffusion and/or transport by known solvent advection. HSL_DC05 is quite unsuitable for the Method of Lines solution of problems of laminar or turbulent fluid flow, or for calculation of orbits under gravitation, or (especially) for modeling feedback control systems in order to detect possible instabilities arising from feedback becoming positive. In fact, relatively few available numerical methods are suitable for this last class of problem, unless they have been carefully designed for the purpose.

1.3 Numerical precision

Solution of DAE or stiff ODE problems is numerically demanding. Use of at least 8-byte arithmetic is recommended.

ATTRIBUTES — Version: 1.1.1. **Types:** Real (single, double). **Precision:** At least 8-byte arithmetic is recommended. **Date:** July 2004. **Origin:** A.R. Curtis, ARC Scientific, L.C. Daniels, Hyprotech UK Ltd., and J.K. Reid, Rutherford Appleton Laboratory. **Calls:** _GEFA, _GESL. **Language:** Fortran 95.

2 HOW TO USE THE PACKAGE

2.1 Calling sequences

Access to the package requires a USE statement.

Single precision version

```
USE HSL_DC05_SINGLE
```

Double precision version

```
USE HSL_DC05_DOUBLE
```

In HSL_DC05_SINGLE, all reals are default reals. In HSL_DC05_DOUBLE, all reals are double precision reals. In both modules, all integers are default integers.

There are three principal subroutines for user calls:

1. The subroutine DC05_INITIALIZE must be called to initialize the data structure used by HSL_DC05. The call to DC05_INITIALIZE is also used to initialize certain components of the data structure that are used to control the integration. The subroutine must be called for each structure prior to calling DC05_INTEGRATE and DC05_FINALIZE.

2. The subroutine DC05_INTEGRATE is the main routine of the package. It uses reverse communication with the calling program. The actions required by the calling program are controlled by the state variable JOB. A template of the calling sequence is in Section 2.4. The calling sequence is more complex than a simple CALL statement, involving a sequence of operations to carry out the tasks requested by the program, followed by re-entry to continue the solution.

3. The subroutine DC05_FINALIZE reallocates the pointer arrays in the data structure used by HSL_DC05 to have size zero. It should be called when the equations using these data structures have been solved unless the data structures are about to be used again to solve a new problem for which the original call of DC05_INITIALIZE is appropriate.

2.2 The derived data type

For each problem, the user is required to declare a structure type DC05_DATA_TYPE.

For example:

```
USE HSL_DC05_DOUBLE ! or USE HSL_DC05_SINGLE
 .....
 TYPE(DC05_DATA_TYPE)   :: DATA
```

2.3 Argument Lists

We use square brackets [ ] to indicate OPTIONAL arguments. We reserve the right to add additional OPTIONAL arguments in further releases of the code.

2.3.1 The initialization subroutine

The initialization routine must be called for each structure used to hold the data for a problem.

```
CALL DC05_INITIALIZE(NEQ, NALG, DATA [,KMAX])
```

NEQ is a scalar INTEGER of INTENT(IN) specifying $N_{\rm eq}$, the total number of equations (algebraic and differential) to be solved. **Restriction:** NEQ > 0.

NALG is a scalar INTEGER of INTENT(IN) that specifies the number of algebraic components (always the first $N_{\rm alg}$ components) of $\mathbf{Y},\mathbf{F}$. The special value -1 indicates a purely differential problem for which the Jacobian matrix $\mathbf{K}$ is constant (as for example, in the simple explicit form of equation (1.7)). Otherwise, the value must be non-negative and specifies $N_{\rm alg}$ directly. **Restriction:** NALG ≥ -1.

DATA is a structure of type DC05_DATA_TYPE of INTENT(INOUT) that is used to hold the data for a problem. No components need be set by the user. The subroutine allocates targets for pointer components and gives components default values. If DC05_INITIALIZE has been previously called for DATA, all data associated with the previous problem are discarded. DATA%IERR is set to zero after a successful entry and to a nonzero value otherwise (details in Section 2.12).

KMAX is an OPTIONAL, scalar INTEGER of INTENT(IN) that is used to specify the highest order of integration formulae used by HSL_DC05. For numerical stability, KMAX cannot be greater than 5. Its value should be normally be set to 5, it should be least 1, and for efficiency preferably at least 3. If KMAX is absent, the value of 5 is used. **Restriction:** $1 \leq$ KMAX ≤ 5.

2.3.2 Integration of equations

The equations are integrated by DC05_INTEGRATE:

```
CALL DC05_INTEGRATE(JOB, ITER, DATA)
```

where:

JOB is a scalar INTEGER of INTENT(INOUT) that must be set to 0 on first entry. On return, it specifies what the code did and what it now requires; it then has one of the following values:

JOB < 0, error return (see Section 2.6).

JOB=1, the code requires evaluation of new Jacobian matrices **J** and **K** at the trial point $t =$ DATA%T, $\mathbf{Y} =$ DATA%Y, $\mathbf{Y}' =$ DATA%YDER (either at the start of an integration step, or because convergence with the old matrix has been unsuccessful). See Section 2.7.1. The evaluation of the new Jacobian matrices **J** and **K** must always be followed by a Newton matrix evaluation, decomposition (factorization) and then function evaluation and solution of equation (1.10) i.e. as for JOB $= 2$.

JOB=2, the code requires evaluation and factorization of a new Newton iteration matrix **W** using the current **J**, **K** (because the step size or order has changed, or **J**, **K** have been updated). See Section 2.7.2. The evaluation and factorization of a new Newton iteration matrix **W** must always followed by function evaluation and solution of equation (1.10) i.e. as for JOB $= 3$.

JOB=3, the code requires evaluation of the functions **F** at the trial point $t =$ DATA%T, $\mathbf{Y} =$ DATA%Y, $\mathbf{Y}' =$ DATA%YDER. See Section 2.7.3. Then the Newton equations (1.10) must be solved, and the correction vector $\delta\mathbf{C}$ returned in DATA%DELTAC. See Section 2.7.3. The solution of the Newton equations (1.10) must always be followed by recall of DC05_INTEGRATE.

JOB=4, an integration step was successfully completed and output values at the end of the step are returned in DATA%T, DATA%Y, and DATA%YDER. Interpolated output at intermediate points within the step can be obtained by changing DATA%T, setting DATA%ISTART=2, and re-entering. DC05_INTEGRATE will then return with JOB=5 (see below and Section 2.7.4). Otherwise, recall DC05_INTEGRATE to take a new step or branch to further code if the end of the integration range has been reached.

JOB=5, interpolated output at an intermediate point in the step has been returned. DATA%Y and DATA%YDER will correspond to DATA%T, and DATA%ISTART will have been reset to 1. More interpolated output may be obtained by changing DATA%T, setting DATA%ISTART=2, and re-entering. Otherwise, recall DC05_INTEGRATE to take a new step or branch to further code if the end of the integration range has been reached.

JOB must not be set by the user except on an initial entry.

ITER is an INTEGER scalar of INTENT(INOUT) that must not be set by the user. It is used by the code to indicate what type of iteration is required. Its normal return value is 0. If DC05_INTEGRATE returns with ITER equal to 1, a special iteration is required to determine a consistent set of initial values at the start of a DAE problem and the calling code must supply a modified version of the Newton matrix, see Sections 2.7.2 and 2.7.3.

DATA is a structure of type DC05_DATA_TYPE of INTENT(INOUT). It will have been initialized by DC05_INITIALIZE and is used to hold the data for the problem to be solved. The components of DATA are described in Section 2.3.4. The parts of DATA that must be set before the first JOB $= 0$ call to DC05_INTEGRATE are described in Section 2.5.2. Actions on subsequent calls to DC05_INTEGRATE are described in Section 2.6. DATA%IERR is set to zero after a successful entry and to a nonzero value otherwise (details in Section 2.12).

2.3.3 The finalization subroutine

```
CALL DC05_FINALIZE(DATA)
```

where:

DATA is a structure of type DC05_DATA_TYPE which will have been initialized by DC05_INITIALIZE and used to hold the data for a problem. On exit from DC05_FINALIZE, its pointer array components will have been reallocated to have zero length. Without such finalization, the storage occupied is unavailable for other purposes. In particular, this is very wasteful and causes 'memory leaks' if DATA goes out of scope on return from a procedure. DATA%IERR is set to zero after a successful entry and to a nonzero value otherwise (details in Section 2.12).

2.3.4 The components of the structure DATA used by DC05_INTEGRATE

The structure DATA is of type DC05_DATA_TYPE and is used to hold the data for a particular problem. DATA contains a number of POINTER arrays that are treated as if they were allocated arrays and are allocated to the correct dimensions by DC05_INTIALIZE. It also contains pointer scalars that are associated with elements of these arrays by DC05_INTIALIZE. For simplicity, we do not document which are pointers. The components listed below are employed in tasks that require interaction with the calling program.

DATA%ALFA, DATA%BETA are REAL scalars, not set on first entry, whose values are output by DC05_INTEGRATE to provide α and β in equation (1.8). Their values must not be changed by the calling program.

DATA%DELTAC is a REAL rank-one array of length NEQ. The elements of DATA%DELTAC need be set only when JOB $= 1, 2$ or 3. Once the Newton equations (1.10) have been solved the user must return in DATA%DELTAC the correction vector $\delta\mathbf{C}$, which is given by the solution of the Newton equations (1.10). On return, DATA%DELTAC holds no significant information.

DATA%HH is a REAL scalar, not set on first entry, whose value is output by DC05_INTEGRATE to provide h of equation (1.8). This is the length of the last completed integration step to DATA%T. Its value must not be changed by the calling program.

DATA%HMAX is a REAL scalar. On the first call (JOB $= 0$) this must be set to the initial trial value of the step-size h (which DC05_INTEGRATE will not exceed), as a guide to the scale of the independent variable t. A reasonable value might be DATA%TOL times the distance to the point at which the user wants output, but even smaller values may give better results as DC05_INTEGRATE can increase its step-size very rapidly, if accuracy tests allow, at the start of a run. On subsequent calls, if the user wishes to limit the step-size to prevent the code from stepping over 'narrow features' in the functions $\mathbf{F}$ without 'seeing' them then the user must set DATA%IHMAX $= 1$ and DATA%HMAX to the limit. On most problems, this is not necessary. When a restart has been requested (DATA%ISTART $= 0$) then DATA%IHMAX must be set to 0 and DATA%HMAX reset.

DATA%IACC is an INTEGER scalar. It allows the user to override the default accuracy and convergence testing. The default accuracy testing has proved very reliable over a wide range of problems. It has built in refinements to deal with cases where a component of $\mathbf{Y}$ has an initial value of zero or passes through zero during integration. If the user wishes to override the default accuracy and convergence testing set DATA%IACC = 2, see Section 2.10.

DATA%IERR is an INTEGER scalar that is set by each of the subroutines. The value zero indicates that the subroutine has performed successfully. For nonzero values, see Section 2.12.

DATA%IHMAX is an INTEGER scalar. It is initialized to 0 by DC05_INITIALIZE and must be left unchanged unless the user wishes to limit the step-size to DATA%HMAX, in which case the user must set DATA%IHMAX $= 1$ and DATA%HMAX to the limit. When a restart has been requested then DATA%IHMAX must be reset to 0.

DATA%ISTART is an INTEGER scalar used to control the action of DC05_INTEGRATE:

DATA%ISTART = 0 for a start or restart,

DATA%ISTART = 1 for continuation of normal integration, and

DATA%ISTART = 2 for additional output within a completed step.

DC05_INITIALIZE sets its value to 0 and this must not be altered before calling DC05_INTEGRATE with JOB $= 0$. On all normal returns, JOB > 0, it will have been set to 1. For normal continuation, the user must not change its value. To request additional output when JOB $= 4$ or 5, the user must change its value to 2; on return, its value will have been reset to 1. To request a restart, the user must change its value to 0. After an error return, JOB < 0, it will have been given a negative value.

DATA%ITSTOP is an INTEGER scalar that is given a default value of 0 by DC05_INITIALIZE. For efficiency, the code takes steps as large as possible and uses an interpolation routine to give results within the last completed step. Thus it will normally integrate past the end of the range. If it is not possible to integrate beyond a value $t_{\rm stop}$ (because of an essential singularity there, such as a failure of definition of the functions **F** beyond that point), DATA%ITSTOP must be given the value 1 and DATA%TSTOP must be given the value $t_{\rm stop}$. If there is no such limit DATA%ITSTOP must have the value 0.

DATA%TSTOP is a REAL scalar. If DATA%ITSTOP $= 1$, DATA%TSTOP must be set to the value of $t_{\rm stop}$.

DATA%IYLOG is an INTEGER rank-one array of length NEQ that controls the status of component Y(I) for I $= 1$ to NEQ of the solution vector **Y** with respect to logarithmic transformation. Input values on an initial JOB $= 0$ call may be -1 (never transform), 0 (transform to avoid negativity) or 1 (transform as soon as Y(I) > 0). The default for each component is -1 and this is set by DC05_INITIALIZE. On return from DC05_INTEGRATE, components of **Y** which have been transformed by DC05_INTEGRATE will be indicated by the corresponding elements of DATA%IYLOG having been incremented by 2 from their initial values, and these must not be altered; if JOB $= 4$ or after a restart (DATA%ISTART $= 0$), components that have not been incremented by 2 i.e for any DATA%IYLOG < 2 may be altered to a value -1, 0 or 1 as required.

DATA%KNTERR is an INTEGER scalar that specifies the maximum number of diagnostic print-outs on unit DATA%OUTPUT1. It has a default value of 10 set by DC05_INITIALIZE. If the value is not positive, there will be no diagnostic printing.

DATA%KOUNT is an INTEGER rank-one array of size 5 that is used to hold a number of different statistics on the various operations carried out by DC05_INTEGRATE since the previous JOB $= 0$ call.

DATA%KOUNT(1) holds the number of completed steps.

DATA%KOUNT(2) holds the number of attempted steps.

DATA%KOUNT(3) holds the number of function evaluations, which is equal to the number of solutions of the Newton equations.

DATA%KOUNT(4) holds the number of Jacobian evaluations.

DATA%KOUNT(5) holds the number of times the Newton matrix was formed and decomposed.

DATA%MAXNOSTEPS is an INTEGER scalar that specifies the maximum number of steps to be taken by DC05_INTEGRATE. Its default value is 500, set by DC05_INITIALIZE.

DATA%OUTPUT is an INTEGER scalar that specifies the unit for printing any warning messages. If negative , the printing of warnings is suppressed. Its default value is 6, set by DC05_INITIALIZE.

DATA%OUTPUT1 is an INTEGER scalar that specifies the unit for printing diagnostic messages. If negative, the printing of diagnostics is suppressed. Its default value is -1, set by DC05_INITIALIZE.

DATA%T is a REAL scalar holding the value of the independent variable t. The initial value must be supplied on first entry JOB $= 0$. DC05_INTEGRATE alters the value of t. The solution will have reached this point if JOB $= 4$, or JOB $= 5$. The value of DATA%T must not be changed except prior to a JOB $= 0$ or JOB $= 5$ entry.

DATA%TOL is a REAL scalar representing the relative accuracy required in the solution. DATA%TOL is used by the code in a local accuracy test which requires that the estimate of the local error on Y(I) satisfies:

$$\text{estimated local error} \leq \text{DATA\%TOL*DATA\%YTST(I)}.$$

On first entry (JOB $= 0$), the user must specify the accuracy required by setting DATA%TOL to a suitable positive value.

DATA%TOL remains unchanged unless DC05_INTEGRATE returns with JOB $= -2$, indicating that DATA%TOL has been reset to keep it within range, see Section 2.6.1. The value may be changed by the user between entries.

DATA%T0 is a REAL scalar. If JOB $= 4$ or 5, it holds the value of the independent variable t at the start of the last completed integration step. For other values of JOB it does not hold any significant value. Its value must not be changed by the calling program.

DATA%TT is a REAL scalar. If JOB $= 4$ or 5, it holds the value of the independent variable t at the end of the last completed integration step, for other values of JOB it does not hold any significant value. Its value must not be changed by the calling program.

DATA%WVEC is a REAL rank-one array of length NEQ. The elements of DATA%WVEC need not set on initial entry, JOB $=$ 0. On return to the calling program for Jacobian evaluation, JOB $= 1$, it holds provisional increments $\delta\mathbf{Y} =$ DATA%WVEC and $\delta\mathbf{Y}' =$ DATA%WVEC/DATA%HH for $\mathbf{Y}$ and $\mathbf{Y}'$ during numerical finite difference evaluation of the Jacobians $\mathbf{J}$ and $\mathbf{K}$. They may need to be increased by a large factor (e.g. 103 or more) to ensure numerical significance of columns corresponding to zero or very small solution components.

DATA%Y is a REAL rank-one array of length NEQ. It contains the components of the solution vector $\mathbf{Y}$ at $t =$ DATA%T. Initial values of $\mathbf{Y}_2$ and estimates of $\mathbf{Y}_1$ must be supplied on first entry, JOB $= 0$. DC05_INTEGRATE returns output values on completion of a step, JOB $= 4$, or interpolated output values at an intermediate point, JOB $= 5$. On return during a step, JOB < 4, it holds the value of $\mathbf{Y}$. It must not be changed by the calling program, except prior to a JOB $= 0$ entry.

DATA%YDER is a REAL rank-one array of length NEQ. It contains the components of $\mathbf{Y}' = \mathrm{d}\mathbf{Y}/\mathrm{d}t$ at $t =$ DATA%T. The user must supply initial guesses on first entry, JOB $= 0$, if equations are implicit and non-linear in $\mathbf{Y}'$, or zero for components occurring linearly. On return with JOB $= 4$, DC05_INTEGRATE returns the output components of $\mathbf{Y}'$ at $t =$ DATA%T. On return with JOB $= 5$, it gives the values of $\mathbf{Y}'$ obtained by differentiating the interpolation formulas used for DATA%Y. It must not be changed by the calling program.

DATA%YTST is a REAL rank-one array of length NEQ. It is used to hold an estimate of the largest absolute value of the each solution component in an interval around the current value of t (DATA%T). The components of DATA%YTST are used for comparison values against which accuracy estimates on the components of $\mathbf{Y}$ are assessed. By default, the comparison values are initialized on a JOB $= 0$ entry , and then updated at the end of each integration step in readiness for the next step. The user may (but is not advised to) set DATA%IACC = 2 and supply his or her own comparison values in DATA%YTST. In this case DC05_INTEGRATE does not alter them.

2.4 Calling Template

The user must include code similar to the following in the calling program, inserting problem-dependent sections where indicated, see Section 2.7

```
! Access via USE statement and declare data structures
 USE HSL_DC05_DOUBLE ! or USE HSL_DC05_SINGLE
 TYPE(DC05_DATA_TYPE)   :: DATA
 INTEGER :: KMAX,NALG, NEQ
 INTEGER :: JOB, ITER
! Insertion: set the values of NEQ, NALG and the optional argument KMAX (if required).
! Initialize the data structure (see Section 2.3.1)
 CALL DC05_INITIALIZE(NEQ,NALG,DATA,KMAX)

! Insertion A: provide data, as described in Section 2.5.2
 JOB = 0
 DO
    CALL DC05_INTEGRATE(JOB, ITER, DATA)

    IF(JOB <= 0) THEN ! Negative value of job indicates error
! Insertion B: either handle error and cycle or exit - see Section 2.6.1
    END IF

    IF(JOB == 1) THEN ! New Jacobian matrices required
! Insertion C: compute J, K at t = DATA%T, Y = DATA%Y, Y' = DATA%YDER,
! retaining them for later use, see Section 2.7.1.
    END IF

    IF(JOB <=2) THEN ! decomposition of new Newton matrix required
! Insertion D:
       IF(ITER.GT.0)THEN ! Handle special iteration
         ! Replace rows NALG+1 to NEQ of W as described in Section 2.7.2
       END IF
! Compute and factorize W, retaining for later re-use, see Section 2.7.2
    END IF

    IF(JOB <= 3) THEN ! function evaluation and Newton correction
! Insertion E: compute F at t = DATA%T, Y=DATA%Y, Y'=DATA%YDER,
!  - see Section 2.7.3
       IF(ITER.GT.0)THEN !Handle special iteration
         ! Replace elements NALG+1 to NEQ of F by those of YDER - see Section 2.7.3
       ENDIF
       ! Compute DELTAC from equation (1.10) - see Section 2.7.3
       CYCLE ! Call code again

    ELSE IF (JOB >= 4) THEN ! step completed or intermediate output point found.
! Insertion F: process any end of step output required;
       IF (further output needed) THEN
          DATA%T = next output point in step (see Section 2.7.4)
          DATA%ISTART = 2
          CYCLE ! Call code again
       ELSE IF (end of range reached) THEN
          EXIT ! Finish integration
       ELSE ! Set new output point
```

```
          CYCLE ! Call code again
        END IF

    END IF
  END DO
  CALL DC05_FINALIZE(DATA) ! Clean up data structures
```

An example using the above coding structure is given in Section 5.1.

2.5 Starting the calculation

In this section, we explain how to commence solving a new problem.

2.5.1 Inititialization of data structure

DC05_INITIALIZE (see Section 2.3.1) must be called to initialize the data structure.

2.5.2 Initialization of problem data (insertion A in calling template)

The following components of DATA (Section 2.3.4) must be set before DC05_INTEGRATE is called.

DATA%HMAX, DATA%ITSTOP, DATA%IACC, DATA%T, DATA%TOL, DATA%Y, DATA%YDER, and DATA%YTST. The other components of DATA need not be set.

2.6 Action on error return from DC05_INTEGRATE

DC05_INTEGRATE returns to the calling program with JOB set to indicate what the calling program must do next. Negative values indicate that an error condition has occurred.

2.6.1 Integration interrupted, but resumable — reported by negative values of JOB (insertion B in calling template)

If the user wishes to inform the code that the integration should continue despite being interrupted, appropriate action should be taken and DATA%ISTART reset. If DATA%ISTART is not reset, the integration will be terminated; no further calls of DC05_INTEGRATE are permitted without another call of DC05_INITIALIZE.

JOB=-1, a large amount of work has been done (DATA%MAXNOSTEPS steps attempted). To enable the code to continue, set DATA%ISTART=1, and call the code again. A further DATA%MAXNOSTEPS steps will be allowed.

JOB=-2, the tolerance on the accuracy DATA%TOL has been changed to bring it within range. The user may change it but should be sure that the new value is suitable. To continue with the changed value, set DATA%ISTART to 1 and call the code again. To restart, set DATA%ISTART to 0, DATA%IHMAX to 0, reset DATA%HMAX, call the code again. Note that if the user has used DATA%TOL to calculate a value for DATA%HMAX, the value will have to be recomputed.

JOB=-3, not used.

JOB=-4, the user asked for interpolated output (DATA%ISTART=2) at a point outside the last completed step, i.e. before DATA%T0 or after DATA%TT. The values of DATA%Y, DATA%YDER returned are those obtained from the interpolation formula at the user's value of DATA%T. DATA%ISTART=1 must be reset before calling DC05_INTEGRATE to

continue integration. If instead further interpolated output is required, the user should set DATA%ISTART=2 and change DATA%T to the required output point.

JOB=-5, the Newton iterations at the start of a DAE problem have failed to achieve $\mathbf{F} = \mathbf{0}$ starting from the user supplied initial values of $\mathbf{Y}, \mathbf{Y}'$. If the values of $\mathbf{Y}, \mathbf{Y}'$ returned are an improvement on the initial estimates, further iterations may be successful. If this is required, the user should then set DATA%ISTART to 0, DATA%IHMAX to 0, reset DATA%HMAX, and call the code again, which forces DC05_INTEGRATE to restart.

JOB=-6, DC05_INTEGRATE had repeated convergence test failures on the last attempted step. Inaccuracy of the Jacobian matrix may be the cause of the convergence problems. The calculation may be continued, perhaps at greatly reduced efficiency, by setting DATA%ISTART to 0, DATA%IHMAX to 0, resetting DATA%HMAX, and calling the code again, which forces DC05_INTEGRATE to restart.

JOB=-7, DC05_INTEGRATE had repeated accuracy test failures on the last attempted step. There may be unrecognized discontinuities in the equations, or the **F** functions may be providing noisy derivative values. The calculation may be continued by setting DATA%ISTART to 0, DATA%IHMAX to 0, resetting DATA%HMAX, and calling the code again, which forces DC05_INTEGRATE to restart.

JOB=-8,...,-32, are not used at present.

2.6.2 Integration terminated

JOB=-33, the code has met trouble from which it cannot recover. A message is printed on DATA%OUTPUT explaining the cause of the trouble, and control is returned to the calling program. For example, invalid input data have been detected. An attempt to continue will cause DC05_INTEGRATE to return immediately.

2.7 Main part the code (insertions C to F in the calling template)

Insertion A has been described in Section 2.5, and insertion B in Section 2.6.1. The remaining insertions are now described, in the sections below. The code in these insertions should not change any quantity not specifically mentioned.

2.7.1 Jacobian calculation (insertion C in calling template)

Insertion C (JOB = 1) requires evaluation of the Jacobian matrices:

$$\mathbf{J}_{ij} = \frac{\partial \mathbf{F}_i}{\partial \mathbf{Y}_j} \quad \text{and} \quad \mathbf{K} = \frac{\partial \mathbf{F}_2}{\partial \mathbf{Y}'_2}. \tag{2.1}$$

If any special action (for example determining the sparsity pattern) is required on the first Jacobian evaluation, code must be written to detect the first return with JOB=1 and take the necessary action. Storage must also be provided for the Jacobian matrices **J**, **K** and for their sparsity pattern description. Evaluation by finite differences or automatic differentiation is recommended, except perhaps for linear problems. The calling program is responsible for choosing good increments for finite differences, although the suggested increments in DATA%WVEC may be found useful (DATA%WVEC for the increments in $\mathbf{Y}$ and DATA%WVEC$/h$ for the increments in $\mathbf{Y}'$). The significance of the calculated matrix columns should be checked, especially those relating to zero or small solution components, and if necessary the corresponding component of DATA%WVEC should be increased and the column re-computed. Increases by a factor of (say) 105 may be suitable. For simple problems of the form (1.7), $\mathbf{K} = -\mathbf{I}$, and so does not need be explicitly evaluated and stored. If sparsity allows differencing for more than one column at a time, the user is responsible for finding and implementing a suitable column grouping, for example by the use of the 'CPR'algorithm (Curtis et al., 1974) which is implemented in the HSL routine TD22. The values of **J** and **K** should be retained for later use, as well as supplied immediately to the following insertion D.

If the user has set NALG $= -1$ indicating that **K** is constant and that $N_{\rm alg} = 0$, DC05_INTEGRATE will return with JOB $= 2$ before the first return with JOB $= 1$. Therefore, care must be taken to ensure that **J** is initialized to zero and **K** to its constant value before the first call to DC05_INTEGRATE to ensure that the initial iterations are carried out correctly.

2.7.2 Newton matrix generation and decomposition (insertion D in calling template)

Insertion D ($1 \leq$ JOB ≤ 2) requires the formation and decomposition (factorization) of the Newton iteration matrix **W**, which is normally given by equation (1.8), but is:

$$\mathbf{W} = \begin{pmatrix} \mathbf{J}_{11} & \alpha\mathbf{J}_{12} \\ \mathbf{0} & \mathbf{I}/h \end{pmatrix} \tag{2.2}$$

if ITER $= 1$. The values of h, α, β, needed to determine **W** will be found as the scalar variables DATA%HH, DATA%ALFA, and DATA%BETA.

The precise nature of the process referred to as factorization or decomposition of **W** is up to the user. All that DC05_INTEGRATE needs is that **W** is stored in some form which enables the user to solve equations (1.10) efficiently for many successive values of **F**. The user is responsible for providing necessary storage for this form of **W** and for any indexing information needed. As a practical point, it may be convenient to store and factorize $-\mathbf{W}$ rather than **W**, since it is the equations (1.10) which have to be solved for $\delta\mathbf{C}$.

If any special action (for example, analysis of the sparsity pattern to choose the pivotal sequence) is required on the first decomposition of **W**, the user must write code to detect the first occasion and take the necessary action. Since this occasion will follow on from the first return with JOB=1, detection can be shared with the insertion C above. If the chosen decomposition routine chooses pivots for stability and sparseness, it may be advisable to replace by zero the components of $\mathbf{J}_{12}$ and $\mathbf{J}_{22}$ of **W** for a trial factorization, and then force use of the same pivotal sequence thereafter, to cope with the expected wide variation in h. This will in fact be automatic for a DAE system, since the first return with JOB=1 will have $\alpha = 0$. For a purely differential system, the first trial decomposition could well omit $\mathbf{J}_{22}$ and simply use the **K** matrix. In either case, of course, the full pattern of logical nonzeros must be supplied to the decomposition routine.

2.7.3 Function or correction calculation (insertion E in calling template)

Insertion E ($1 \leq$ JOB ≤ 3) defines the problem. If ITER $= 0$ the functions **F** must be evaluated at the current trial point $t =$ DATA%T, with **Y** and $\mathbf{Y}'$ in DATA%Y and DATA%YDER. DATA%WVEC must not be altered. In practice, it is probably best to call a subroutine to evaluate **F**, since the same code will be needed in computing the Jacobians (by finite differences or automatic differentiation) when JOB=1.

In the case of a DAE problem, DC05_INTEGRATE uses a modified form of (1.10) to obtain initial values of $\mathbf{Y}'_1$. In this case it returns with ITER $= 1$, to indicate that the components $\mathbf{F}_2$ must be replaced by those of $\mathbf{Y}'_2$ in DATA%YDER, before solving equations (1.10). The function evaluation is followed by solution of the correction equation (1.10) to obtain a correction vector $\delta\mathbf{C}$, which must be placed in DATA%DELTAC. The coding needed to solve these equations will, of course, depend on the chosen decomposition of **W**. If an approximate method of solution is chosen, it must be accurate enough not to upset the convergence test in DC05_INTEGRATE, which is to an accuracy of 0.1*DATA%TOL, relative to the components of DATA%YTST. It follows that $\delta\mathbf{C}$ should be found to considerably better accuracy than this.

2.7.4 End of integration step, output handling (insertion F in calling template)

If output is required only at the end of each integration step, it need only be processed when JOB=4. If however, output is required at one or more intermediate points in the step just completed, from DATA%T0 to DATA%TT, set DATA%T to the next value of t at which output is wanted, change DATA%ISTART to 2, and re-enter. DC05_INTEGRATE will interpolate

the values of $\mathbf{Y}$ and $\mathbf{Y}'$, returning them in arrays DATA%Y and DATA%YDER, with JOB=5 and DATA%ISTART reset to 1. This may be done as often as required, so long as the requested values of DATA%T are within the step.

It is safe to test whether a point has been passed, and to iterate if necessary to find it, if an output point is determined by the value of a function of t, $\mathbf{Y}$ and $\mathbf{Y}'$, since the interpolated values in DATA%Y and DATA%YDER are continuous functions of DATA%T.

It should be noted that DC05_INTEGRATE maintains an internal variable TIME which may differ from the external DATA%T. The reason for this is to reduce the risk that when t is large the step size h may become so small that an end-of-step value $t+h$ may be indistinguishable from t, causing failure of some internal algorithms. At each start or restart, DC05_INTEGRATE sets an internal quantity TZERO = DATA%T and TIME = 0, thereafter advancing TIME by h (DATA%HH) on each step and returning DATA%T = TIME+TZERO. The internal tests require only that DATA%HH differs from zero, which is more rugged if TZERO is large.

There is no need to restore end-of-step DATA%Y and DATA%YDER values before re-entering for a new step. However, if the problem is restarted by re-entering with DATA%ISTART $= 0$ it will of course start with the current contents of DATA%Y, DATA%T and DATA%YDER. See Section 2.9, for information about more advanced techniques, including the handling of discontinuities in the definition of $\mathbf{F}$.

2.8 How to continue integration by steps after the first step

The code is organized so that steps after the first step involve little (if any) additional effort on the part of the user.

To carry out another integration step, re-enter DC05_INTEGRATE without altering any quantity except as described in the next three paragraphs. In principle, a change in any numerical parameter, other than those described in the next three paragraphs, counts as a change to the differential equations and should be treated by restarting with a new DATA%ISTART $= 0$ call. However, in practice DC05_INTEGRATE can often cope quite well with ordinary discontinuities in $\mathbf{F}$, at little greater cost than if a restart is forced.

DATA%TOL can be changed 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. The user may switch to his or her own control of DATA%YTST by setting DATA%IACC $= 2$, but cannot thereafter switch back again.

The value of HMAX can be changed, if DATA%IHMAX $= 1$ is set.

DATA%TSTOP can be set at any time, provided DATA%ITSTOP is set to 1; the code will refuse to integrate past the current DATA%TSTOP. Once it has been reached, DATA%TSTOP must be changed or DATA%ITSTOP set to 0 in order to continue.

2.8.1 To continue integration after a completed step

If JOB=4 or 5, call the code again to continue the integration, or first call it with DATA%ISTART=2 and DATA%T changed (see Section 2.7.4 above) to get interpolated output within the step just completed.

2.9 Advanced output testing; handling discontinuities

The user may require output when some function of the solution $\mathbf{Y}$, $\mathbf{Y}'$ and the independent variable t takes a certain value, e.g. when $P(t,\mathbf{Y},\mathbf{Y}')=0$ for a known function P. As an example, suppose output is required at $Y_1 - tY_3 = 0$. Start integrating and, at each return with JOB=4, monitor:

$$P\big(\texttt{DATA\%T,DATA\%Y,DATA\%YDER}\big) = \quad \texttt{DATA\%Y(1)-DATA\%T*DATA\%Y(3)} \tag{2.3}$$

to see if it has changed sign; if not, call the code again. When P has changed sign in the most recent step, carry out an inverse interpolation to find a value DATA%T at which $P = 0$. This has to be done iteratively. When the point has been found to sufficient accuracy (for example to within DATA%TOL*DATA%HH), 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 functions. By ordinary discontinuities we mean those where a smooth continuation would exist were it not for a change of definition of some parameter value. Suppose a switching function $P(t,\mathbf{Y},\mathbf{Y}')$ is constructed, such that the discontinuity occurs at $P=0$. Using the old **F** coding (including the smooth continuation past the switching point), locate the latter by the technique of the previous paragraph. Then make the necessary discontinuous change, force a restart (by setting DATA%ISTART to 0, DATA%IHMAX to 0, and resetting DATA%HMAX) and call the code again. The restart will occur to 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. DC05_INTEGRATE can in fact integrate through ordinary discontinuities or near-discontinuities without serious loss of efficiency in many cases.

To do the necessary inverse interpolation we can use Newton's method. Consider our example function, $P = Y_1 - tY_3$. We start with the interval $(t_l,t_u) =$ (DATA%T0,DATA%TT). If we evaluate

$$\frac{\mathrm{d}P}{\mathrm{d}t} = \frac{\mathrm{d}Y_1}{\mathrm{d}t} - Y_3 - t\frac{\mathrm{d}Y_3}{\mathrm{d}t}$$

as applied to interpolated values within the current step, the values of $\mathrm{d}Y_i/\mathrm{d}t$ returned in DATA%YDER are exact. Thus we can use them in the above expression, calculating (DATA%YDER(1)-DATA%Y(3)-DATA%T*DATA%YDER(3)). The Newton estimate for the point at which $P=0$ is:

$$\begin{aligned} t &= \texttt{DATA\%T} - \left(\frac{P}{\mathrm{d}P/\mathrm{d}t}\right) \quad \text{evaluated at } t = \texttt{DATA\%T} \\ &= \texttt{DATA\%T} - \big(\texttt{DATA\%Y(1)} - \texttt{DATA\%T} * \texttt{DATA\%Y(3)}\big) \big/ \big(\texttt{DATA\%YDER(1)} - \texttt{DATA\%Y(3)} - \texttt{DATA\%T} * \texttt{DATA\%YDER(3)}\big). \end{aligned}$$

If this lies within the current interval, set DATA%ISTART=2 and re-enter to get interpolated values there, iterating until the change is within tolerance. If we find that the above t is outside the current interval, we carry out a binary subdivision of either $(t_l,$DATA%T$)$ or $($DATA%T$,t_u)$ choosing the sub-interval in which P 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.

2.10 Overriding the default accuracy and convergence testing

DC05_INTEGRATE uses a relative accuracy test which requires that the estimate of the local error on Y(I) satisfies:

$$\text{estimated local error} \leq \texttt{DATA\%TOL} * \texttt{DATA\%YTST(I)}$$

for each component. By default, the values of the components of DATA%YTST are generated as the calculation procedes. The process has proved very reliable over a wide range of problems, and the user should think carefully before deciding to change it. It has built-in refinements to deal with cases where a component of **Y** has an initial value of zero or passes through zero during integration. DATA%YTST may removed from control by DC05_INTEGRATE and its values set by the user, although such course is not recommended as departing from the default may sacrifice much of the reliability built into the accuracy and convergence testing.

The user should therefore consider carefully whether to override the default calculation of DATA%YTST(I). If user wishes to:

Accept the defaults — the user need do nothing DC05_INITIALIZE has set DATA%IACC = 0 as a default.

Override the defaults — set DATA%IACC = 2 and set DATA%YTST(I), I = 1 to NEQ.

2.11 Ordering of vectors in DAE problems

When using HSL_DC05 on DAE problems one may encounter a minor difficulty in that the problem as received may have one or more of the algebraic components of **F** and **Y** ordered away from the first $N_{\rm alg}$ positions. The recommended solution is to call the given subroutines for these evaluations through others, which carry out the necessary permutations of vector components, before calling and after returning from the ready-coded subroutines. The simplest way is to buffer the vectors into and from others which are actually passed to the supplied code. There is then no danger of permuted vectors reaching HSL_DC05, with disastrous results.

2.12 Error diagnostics

If DC05_INTEGRATE encounters no errors or complications, the value of DATA%IERR will be zero on return from the subroutine. The errors and complications that can occur are as follows:

-1 a negative value was given for DATA%TOL

-2 DATA%TOL was larger than the maximum value, and was reset to 10^{-2}.

-3 DATA%TOL was smaller than the minimum value, and was reset to 10^{-10}.

-5 the number of equations NEQ was less than or equal to zero.

-6 the array components of DATA are too small, probably because it is being used for a larger problem without a call to DC05_INITIALIZE.

-7 the value of DATA%TSTOP is less than DATA%T.

-8 a negative initial step estimate was given in DATA%HMAX.

-12 re-entry with illegally changed JOB.

-14 re-entry without resetting DATA%ISTART.

-15 a negative value was given for DATA%HMAX.

-16 integration was requested beyond DATA%TSTOP.

-17 to -25 the POINTER arrays in DATA have not been ASSOCIATED correctly, probably because it is being used without a call to DC05_INITIALIZE.

-33 fatal error return.

-47 to -54 the POINTER arrays in DATA have not been correctly sized and they are too small for the problem.

-60 DATA%IHMAX has been illegally reset.

-61 DATA%ITSTOP has been illegally reset.

-62 NALG has been illegally reset.

-63 DATA%IACC has been illegally reset.

-70 a DEALLOCATE failed in DC05_INITIALIZE.

-80 an ALLOCATE failed in DC05_INITIALIZE.

-90 a DEALLOCATE failed in DC05_FINALIZE.

2.13 Warning and error messages

Explanatory messages are printed out before an integration is terminated. This is done on the Fortran unit specified by DATA%OUTPUT, if it has a non-negative value. The default value is 6. All the integration termination messages finish with:

```
***** HSL_DC05 INTEGRATION TERMINATED FOR ABOVE REASON
```

The messages which precede the above termination message are self-explanatory.

2.14 Information printed

Diagnostic messages are output on unit DATA%OUTPUT1 (if non-negative). A message is output if DC05_INTEGRATE experiences one of the difficulties: when the corrections on the first corrector iteration are so large that it is unlikely that the accuracy will be passed; when convergence failure occurs in spite of new Jacobian matrices; and when accuracy test failure occurs so many times in the same step that DC05_INTEGRATE reduces the order of the integration formula. They do not necessarily indicate an error condition but merely that the integration step size has had to be reduced and that efficiency may be impaired.

The user can control the amount of diagnostic print-out that is produced by setting DATA%KNTERR to any non-negative value. The default value is 10.

3 GENERAL INFORMATION

Use of COMMON: None
Workspace: Provided automatically by HSL_DC05.
Other subroutines: None.
Restrictions: NEQ > 0, $-1 \leq$ NALG $\leq$ NEQ and $1 \leq$ KMAX ≤ 5.
Portability: ISO Fortran 95.

4 METHOD

The method used is especially efficient on stiff problems defined by equation (1.6) or the more general DAE problems of equations (1.2) and (1.3).

4.1 Outline

The solution is advanced step by step in t, using (in principle) steps of fixed length h by means of a variable order Backward Differentiation Formula (BDF) method, Gear (1971). Within each integration step, the variables $\mathbf{Y}$ are represented as polynomials in t of degree k, $k = 1$, to 5. The algorithm for taking a step (of length h from $t = x_{r-1}$ to $t = x_r = x_{r-1} + h$) updates these polynomials, making use of equations (1.2) – (1.3) by satisfying them at the end of the new step. As these equations are in general nonlinear, this requires iteration. Newton iterations are used.

So DC05_INTEGRATE can return to the beginning of an integration step which fails its accuracy test, and repeat it with a smaller step-size, the package defines the values of the vectors $\mathbf{Y}$, $\mathbf{Y}'$ to the calling program in terms of internal vectors $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ and scalar arguments α, β, and h by:

$$\mathbf{Y}_1 = \mathbf{A}_1 + \mathbf{C}_1, \qquad \mathbf{Y}_2 = \mathbf{A}_2 + \alpha\mathbf{C}_2, \qquad \mathbf{Y}' = (\mathbf{B} + \beta\mathbf{C})/h. \tag{4.4}$$

The vectors **A** and **B** are constant during iterations and **C** is updated at each iteration. The scalar α is equal to 1 normally, but is set to 0 during iterations on start-up to correct the initial guess for $\mathbf{Y}_1$. Thus except in this preliminary phase the first two equations of (4.4) can be written simply as:

$$\mathbf{Y} = \mathbf{A} + \mathbf{C}. \tag{4.5}$$

The scalar β is always of order unity, while h is the current integration step-length and may vary through many orders of magnitude. Then the Newton iteration matrix:

$$\mathbf{W} = \begin{pmatrix} \partial \mathbf{F}_1/\partial \mathbf{C} \\ \\ \partial \mathbf{F}_2/\partial \mathbf{C} \end{pmatrix} = \begin{pmatrix} \mathbf{J}_{11} & \alpha \mathbf{J}_{12} \\ \\ \mathbf{J}_{21} & \alpha \mathbf{J}_{22} + \beta \mathbf{K}/h \end{pmatrix} \tag{4.6}$$

of equations (1.2) and (1.3) is used in solving such problems, from the beginning of a DAE problem or after a possible initial transient for an explicit stiff problem. In equation (4.6) the Jacobian matrices are given by:

$$\mathbf{J}_{ij} = \frac{\partial \mathbf{F}_i}{\partial \mathbf{Y}_j} \quad \text{and} \quad \mathbf{K} = \frac{\partial \mathbf{F}_2}{\partial \mathbf{Y}_2'}. \tag{4.7}$$

The matrix **W** is used in Newton iterations, solving the correction equations:

$$\mathbf{W}\delta\mathbf{C} + \mathbf{F} = \mathbf{0} \tag{4.8}$$

for the correction vector $\delta\mathbf{C}$ to be added to **C**.

During each integration step DC05_INTEGRATE iterates to generate the correction vector $\delta\mathbf{C}$ which causes the equations to be satisfied at the end of the step. During these, it sets $\alpha = 1, \beta = O(1)$ (dependent on the order of the integration formula) and $h =$ current step size. The user must solve the equations (4.8) for $\delta\mathbf{C}$.

On the first convergence failure, the step is repeated to obtain a new Newton iteration matrix **W** from the same Jacobians **J**, **K**, on the assumption that the convergence failure is due to a change of h. On subsequent convergence failures, the step is repeated, on the assumption that the Jacobians **J**, **K** are inaccurate as a result of changes in t and **Y**. On successful convergence but failure of the accuracy testing, the step is repeated with a reduced step-size, h.

DC05_INTEGRATE also uses (4.8) to perform preliminary iterations at each start or restart, to set initial values of $\mathbf{Y}_2'$ and correct initial values of $\mathbf{Y}_1$. During these, it sets $\alpha = 0, \beta = 1, h =$ initial trial value.

4.2 Preliminary Iterations

Especially in non-linear problems, it is very common for the user's initial estimates of algebraic components $\mathbf{Y}_1$, and of the derivatives $\mathbf{Y}_2^{'}$ of differential components, to be insufficiently accurate to satisfy the equations well enough to ensure a reliable start to the solution. It is even possible for errors in $\mathbf{Y}_1$ to be so large that the first integration step cannot converge to a solution satisfying the accuracy test. A few preliminary iterations are therefore carried out to correct any such errors. The preliminary iterations are carried out using the iteration matrix with $\alpha = 0$, applying the corrections only to $\mathbf{Y}_1$ and $\mathbf{Y}_2^{'}$. The iteration matrix is recomputed at the start of each iteration, to give true Newton iterations. When these are complete, a single additional iteration can compute good initial derivatives $\mathbf{Y}_1^{'}$ of the algebraic components (if any, using the equation:

$$0 = \frac{\mathrm{d}\mathbf{F}_1}{\mathrm{d}t} = \frac{\partial \mathbf{F}_1}{\partial t} + \mathbf{J}_{11}\mathbf{Y}_1' + \mathbf{J}_{12}\mathbf{Y}_2' \tag{4.9}$$

this iteration is based on the fact that, for the solution, the total derivative with respect to t of the $\mathbf{F}_1$ components must vanish, since they are identically zero.

This is also valuable in ensuring that accuracy testing on the first integration step can succeed. It is, of course, omitted on purely differential equation problems.

5 EXAMPLES OF USE

In the examples shown below we show how HSL_DC05 can be used to solve a stiff systems of ODEs and a DAE problem.

5.1 Example 1 — A simple stiff problem

As a simple example of solving a stiff problem we take an example from Gear (1971). The equations solved are:

$$\frac{dy_1}{dt} = 998y_1 + 1998y_2, \qquad y_1(0) = 1, \tag{5.10a}$$

$$\frac{dy_2}{dt} = -999y_1 - 1999y_2, \qquad y_2(0) = 0. \tag{5.10b}$$

5.1.1 Program

```
    PROGRAM GEAR_SPEC
      USE HSL_DC05_DOUBLE
      IMPLICIT NONE

      INTEGER, PARAMETER :: WP = KIND(0.0D+0) ! Double Precision
      REAL (WP), PARAMETER :: ZERO = 0.0_WP

!     Local arrays for Jacobian, decomposition of Newton matrix and K Jacobian
      REAL (WP) :: FVAL(2), AJAC(2,2), ALUD(2,2), AKAC(2,2)

      REAL (WP) :: T0, T1, TOUT, TFINAL ! Various times
      INTEGER :: I, IERROR, IINFO, ITER, JOB, KMAX, LLP, NEQ, NALG
      INTEGER, DIMENSION (2) :: IPVT ! Used by DGETRF and DGETRS
      TYPE (DC05_DATA_TYPE) :: DATA ! Problem data

! Initialize Data Structure
      NEQ  = 2;  NALG = 0; KMAX = 5
      CALL DC05_INITIALIZE(NEQ,NALG,DATA,KMAX=KMAX)

! Set up output stream
      LLP  = 16; DATA%OUTPUT = LLP
      OPEN (UNIT=LLP,FILE='temp',STATUS='UNKNOWN',ACTION='READWRITE', &
        IOSTAT=IERROR)

!  Prepare to start the integration
      DATA%TOL = 1.E-2_WP !   Set relative error tolerance
      JOB = 0
      TOUT = 1.E-3_WP ! Initial output point
      DATA%HMAX = TOUT*DATA%TOL ! Initial step size
      DATA%T       = ZERO
      DATA%Y(:)    = (/1.E0_WP,ZERO/)
      DATA%YDER(:) = (/998.0_WP,-999.0_WP/)
      TFINAL = 200.0_WP ! Time for end of output
```

```
! Set K matrix here as it is constant
      AKAC(:,:) = ZERO; AKAC(1,1) = -1.E0_WP; AKAC(2,2) = -1.E0_WP
      AJAC(:,:) = ZERO ! Clear AJAC for safety first time
      WRITE (LLP,'(a)') &
        '       DATA%KOUNT(1:5)        TIME           Y1           Y2'
      T0 = DATA%T
      DO  ! Main loop
        CALL DC05_INTEGRATE(JOB,ITER,DATA)
        IF (JOB<=0 .OR. JOB>5) THEN
! Error reporting
          WRITE (LLP,'(/A,I3,2(A,ES13.5))') &
                '!!! ERROR JOB = ', JOB,' !   T0 = ', T0, '   T = ', DATA%T
          WRITE (LLP,65) (I,DATA%Y(I),I=1,2)
65        FORMAT (/,3(:,3X,'Y(',I2,')=',D13.5,3X))
! Error handling
          IF (JOB<-30) STOP ! Fatal Error
          DATA%ISTART = 1
          IF (JOB<=-5) DATA%ISTART = 0 ! Force Restart
          IF (JOB==-2) THEN
            DATA%ISTART = 0
            DATA%HMAX = TOUT*DATA%TOL ! Reset step size
          END IF
          CYCLE ! Call code again
        END IF
        IF (JOB==1) THEN ! Construct Jacobian
!  AJAC contains DF/DY
          AJAC(1,1) = 998.0_WP + 1998.0_WP*DATA%Y(2)
          AJAC(2,1) = -999.0_WP - 1999.0_WP*DATA%Y(2)
          AJAC(1,2) = 998.0_WP*DATA%Y(1) + 1998.0_WP
          AJAC(2,2) = -999.0_WP*DATA%Y(1) - 1999.0_WP
        END IF
        IF (JOB<=2) THEN
!  Construct and factor Newton matrix
!  ALUD contains - ALFA*DF/DY - (BETA/H)*DF/DY' = - DF/DC = - W
          ALUD(:,:) = -AJAC(:,:)*DATA%ALFA - AKAC(:,:)*DATA%BETA/DATA%HH
          CALL DGETRF(2,2,ALUD,2,IPVT,IINFO) !  Get LU factors of W
        END IF
        IF (JOB<=3) THEN
!  Calculate functions and correction
          FVAL(1) = 998.0_WP*DATA%Y(1) + 1998.0_WP*DATA%Y(2) - DATA%YDER(1)
          FVAL(2) = -999.0_WP*DATA%Y(1) - 1999.0_WP*DATA%Y(2) - DATA%YDER(2)
!  Replace F by DELTAC = - WINVERSE*F, solving  W*DELTAC + F = 0
          CALL DGETRS('N',2,1,ALUD,2,IPVT,FVAL,2,IINFO)
! Copy values of correction vector returned by DGETRS in FVAL into DELTAC
          DATA%DELTAC(:) = FVAL(:)
          CYCLE
        ELSE IF (JOB==4) THEN
!  Completed step; test for output
          IF (TOUT>DATA%T) THEN ! Interpolate for output
```

```
          T0 = DATA%T
          CYCLE
        ELSE IF (TOUT==DATA%T) THEN ! Produce output
          WRITE (LLP,'(5I5,4ES13.5)') DATA%KOUNT, DATA%T, DATA%Y
        ELSE IF (TOUT<DATA%T) THEN ! Interpolate
          IF (TOUT<T0) THEN ! Error condition
            WRITE (LLP,'(/A,I3,2(A,ES13.5))') &
                '!!! ERROR JOB = ', JOB,' !   T0 = ', T0, '   T = ', DATA%T
            WRITE (LLP,65) (I,DATA%Y(I),I=1,2)
          END IF
          T1 = DATA%T
          DATA%T = TOUT;  DATA%ISTART = 2
          CYCLE
        END IF
      ELSE IF (JOB==5) THEN ! Output point reached
        WRITE (LLP,'(5I5,4ES13.5)') DATA%KOUNT, DATA%T, DATA%Y
        TOUT = TOUT*1.0E1_WP !  Increment TOUT
        IF (TOUT<=T1) THEN ! Another interpolation is required
          DATA%T = TOUT; DATA%ISTART = 2
        END IF
      END IF
      IF (TOUT>=TFINAL) EXIT
    END DO
    CALL DC05_FINALIZE(DATA)  ! Tidy up
  END PROGRAM GEAR_SPEC
```

The output produced by GEAR_SPEC is as follows:

```
     DATA%KOUNT(1:5)         TIME           Y1           Y2
  13   13   23    2    6  1.00000E-03  1.62985E+00 -6.30851E-01
  27   27   50    2    9  1.00000E-02  1.97989E+00 -9.89841E-01
  35   35   60    2   11  1.00000E-01  1.80876E+00 -9.03907E-01
  55   55  169    9   23  1.00000E+00  7.35478E-01 -3.67564E-01
 130  135  411   16   45  1.00000E+01  9.12276E-05 -4.56052E-05
 218  232  562   16   68  1.00000E+02  7.08661E-16 -1.03648E-15
```

5.2 Example 2 — A stiff ODE Problem from chemical kinetics

As an example of a system of stiff ODE's, we use a model of a chemical reaction which was first described by Robertson (1966), and has become a model for a stiff system of ODE's.

The mass action kinetics are represented by the following equations:

$$\frac{dy_1}{dt} = -0.04y_1 + 10^4 y_2 y_3, \qquad y_1(0) = 1, \tag{5.11a}$$

$$\frac{dy_2}{dt} = 0.04y_1 - 10^4 y_2 y_3 - 3 \times 107 y_2 y_2, \qquad y_2(0) = 0, \tag{5.11b}$$

$$\frac{dy_3}{dt} = 3 \times 10^7 y_2 y_2, \qquad y_3(0) = 0. \tag{5.11c}$$

We use the LINPACK routines DGEFA and DGESL Dongarra et al. (1979) to factorize, by performing a LU decomposition, and solve the Newton iteration matrix. The code and its output is not reproduced here, but is available with the source code for HSL_DC05. From the solution it can be seen that $t \to \infty, y_1 \to 0, y_2 \to 0$ and $y_3 \to 1$..

5.3 Example 3 — a DAE problem

As an example of a DAE problem we use the 'Chemical Akzo Nobel'problem from Lioen and de Swart (1999). The problem originates from Akzo Nobel Central research in Arnhem, The Netherlands.

The mathematical formulation of the problem is given below in equations (5.12a – 5.12f):

$$\frac{dy_1}{dt} = -2r_1 + r_2 - r_3 - r_4, \tag{5.12a}$$
$$\frac{dy_2}{dt} = -\frac{1}{2}r_1 - r_4 - \frac{1}{2}r_5 + F_{\text{in}}, \tag{5.12b}$$
$$\frac{dy_3}{dt} = r_1 - r_2 + r_3, \tag{5.12c}$$
$$\frac{dy_4}{dt} = -r_2 + r_3 - 2r_4, \tag{5.12d}$$
$$\frac{dy_5}{dt} = r_2 - r_3 + r_5, \tag{5.12e}$$
$$0 = K_s \cdot y_1 \cdot y_4 - y_6, \tag{5.12f}$$

where the $r_i, i = 1, \ldots 5$, and F_{in} are given by:

$$r_1 = k_1 \cdot y_1 4 \cdot y_2^{\frac{1}{2}}, \tag{5.13a}$$
$$r_2 = k_2 \cdot y_3 \cdot y_4, \tag{5.13b}$$
$$r_3 = \frac{k_2}{K} \cdot y_1 \cdot y_5, \tag{5.13c}$$
$$r_4 = k_3 \cdot y_1 \cdot y_4 2, \tag{5.13d}$$
$$r_5 = k_4 \cdot y_6 2 \cdot y_2^{\frac{1}{2}}, \tag{5.13e}$$
$$F_{\text{in}} = klA \cdot \big(\frac{p(CO_2)}{H} - y_2\big). \tag{5.13f}$$

The parameters in equations (5.13a – 5.13f) are as follows:

$$k_1 = 18.7, \quad k_4 = 0.42, \quad K_s = 115.83,$$
$$k_2 = 0.58, \quad K = 34.4, \quad p(CO_2) = 0.9,$$
$$k_3 = 0.09, \quad klA = 3.3, \quad H = 737.$$

The consistent initial vectors are as follows:

$$\mathbf{y}(0) = \big(0.444, 0.00123, 0, 0.007, 0, K_s \cdot y_1(0) \cdot y_4(0)\big)^{\mathrm{T}},$$
$$\frac{d\mathbf{y}}{dt} = \mathbf{F}(\mathbf{0}).$$

where $\mathbf{F}$ is the right hand sides of equations (5.13a – 5.13f). Full details of the problem can be found in Lioen and de Swart (1999),

The code and its output is not reproduced here, but is available with the source code for HSL_DC05. We use HSL_DC05 to solve the 'Chemical Akzo Nobel' problem. Once again we use the LINPACK routines DGEFA and DGESL

(Dongarra et al., 1979) to factorize, by performing a LU decomposition, and solve the Newton iteration matrix. For small dense Newton iteration matrices the LINPACK routines perform very well. The values of `Y(i)` obtained, $i = 1, \ldots 6$, are in agreement with the solution in (Lioen and de Swart, 1999).

5.3.1 Program

An alternative to the use of a series of `IF` constructs, as in Section 5.1, is to use the `SELECT CASE` construct. Note however, that the Fortran 90/95 `SELECT CASE` construct does not allow 'a drop through' between the individual cases. If the user prefers to use a `SELECT CASE` construct care must be taken to ensure that the correct sequence events is followed i.e. the evaluation of a new Jacobian is always followed by, the formation and decomposition of a new Newton iteration matrix, which is always followed by a function evaluation and Newton correction, which is always followed by a call to `DC05_INTEGRATE`. One way achieving this is to combine a `DO` construct with a `SELECT CASE` construct together with the introduction of an additional 'state variable', `IDID`, see the code itself.

REFERENCES

A. R. Curtis, M. J. D. Powell, and J. K. Reid. On the estimation of sparse jacobian matrices. *J. Inst. Maths. Applics.*, 13:117–120, 1974.

J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart. *LINPACK Users' Guide*. SIAM, Philadelphia, 1979.

C. W. Gear. *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice Hall, Engelwood Cliffs, New Jersey, 1971.

Walter M. Lioen and Jacques J. B. de Swart. Test set for initial value problem solvers. Technical report, CWI, Amsterdam, The Netherlands, 1999. URL `http://pitagora.dm.uniba.it/ testset//`.

H. H. Robertson. *The Solution of a Set of Reaction Rate Equations*, page 178. Academic Press, New York, 1966.