NAG Fortran Library Manual
Mark 18

Volume 2

D01 – D02N

D01 – Quadrature
D02 – Ordinary Differential Equations (cont'd in Volume 3)

NAG Fortran Library Manual, Mark 18

The copyright owner gives no warranties and makes no representations about the contents of this manual and specifically disclaims any implied warranties or merchantability or fitness for any purpose.

The copyright owner reserves the right to revise this manual and to make changes from time to time in its contents without notifying any person of such revisions or changes.

Printed and produced by NAG

1st Edition – September 1997 ISBN 1-85206-147-2

NAG Ltd
Wilkinson House
Jordan Hill Road
OXFORD
United Kingdom OX2 8DR

Tel: +44 (0)1865 511245
Fax: +44 (0)1865 310139

NAG GmbH
Schleißheimerstraße 5
D-85748 Garching
Deutschland

Tel: +49 (0)89 3207395
Fax: +49 (0)89 3207396

NAG Inc
1400 Opus Place, Suite 200
Downers Grove, IL 60515-5702
USA

Tel: +1 630 971 2337
Fax: +1 630 971 2706

NAG also has a number of distributors throughout the world. Please contact NAG for further details.

Chapter D01 – Quadrature

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
D01AHF	8	1-D quadrature, adaptive, finite interval, strategy due to Patterson, suitable for well-behaved integrands
D01AJF	8	1-D quadrature, adaptive, finite interval, strategy due to Piessens and de Doncker, allowing for badly-behaved integrands
D01AKF	8	1-D quadrature, adaptive, finite interval, method suitable for oscillating functions
D01ALF	8	1-D quadrature, adaptive, finite interval, allowing for singularities at user-specified break-points
D01AMF	8	1-D quadrature, adaptive, infinite or semi-infinite interval
D01ANF	8	1-D quadrature, adaptive, finite interval, weight function $\cos(\omega x)$ or $\sin(\omega x)$
D01APF	8	1-D quadrature, adaptive, finite interval, weight function with end-point singularities of algebraico-logarithmic type
D01AQF	8	1-D quadrature, adaptive, finite interval, weight function $1/(x-c)$, Cauchy principal value (Hilbert transform)
D01ARF	10	1-D quadrature, non-adaptive, finite interval with provision for indefinite integrals
D01ASF	13	1-D quadrature, adaptive, semi-infinite interval, weight function $\cos(\omega x)$ or $\sin(\omega x)$
D01ATF	13	1-D quadrature, adaptive, finite interval, variant of D01AJF efficient on vector machines
D01AUF	13	1-D quadrature, adaptive, finite interval, variant of D01AKF efficient on vector machines
D01BAF	7	1-D Gaussian quadrature
D01BBF	7	Pre-computed weights and abscissae for Gaussian quadrature rules, restricted choice of rule
D01BCF	8	Calculation of weights and abscissae for Gaussian quadrature rules, general choice of rule
D01BDF	8	1-D quadrature, non-adaptive, finite interval
D01DAF	5	2-D quadrature, finite region
D01EAF	12	Multi-dimensional adaptive quadrature over hyper-rectangle, multiple integrands
D01FBF	8	Multi-dimensional Gaussian quadrature over hyper-rectangle
D01FCF	8	Multi-dimensional adaptive quadrature over hyper-rectangle
D01FDF	10	Multi-dimensional quadrature, Sag–Szekeres method, general product region or n-sphere
D01GAF	5	1-D quadrature, integration of function defined by data values, Gill–Miller method
D01GBF	10	Multi-dimensional quadrature over hyper-rectangle, Monte Carlo method
D01GCF	10	Multi-dimensional quadrature, general product region, number-theoretic method
D01GDF	14	Multi-dimensional quadrature, general product region, number-theoretic method, variant of D01GCF efficient on vector machines
D01GYF	10	Korobov optimal coefficients for use in D01GCF or D01GDF, when number of points is prime
D01GZF	10	Korobov optimal coefficients for use in D01GCF or D01GDF, when number of points is product of two primes

D01JAF	10	Multi-dimensional quadrature over an n-sphere, allowing for badly-behaved integrands
D01PAF	10	Multi-dimensional quadrature over an n-simplex

Chapter D01

Quadrature

Contents

1 Scope of the Chapter **2**

2 Background to the Problems **2**

2.1 One-dimensional Integrals 2

2.2 Multi-dimensional Integrals 3

3 Recommendations on Choice and Use of Available Routines **5**

3.1 One-dimensional Integrals over a Finite Interval 5

3.2 One-dimensional Integrals over a Semi-infinite or Infinite Interval 6

3.3 Multi-dimensional Integrals 7

4 Decision Trees **10**

5 References **12**

1 Scope of the Chapter

This chapter provides routines for the numerical evaluation of definite integrals in one or more dimensions and for evaluating weights and abscissae of integration rules.

2 Background to the Problems

The routines in this chapter are designed to estimate:

(a) the value of a one-dimensional definite integral of the form:

$$\int_a^b f(x)\,dx \tag{1}$$

where $f(x)$ is defined by the user, either at a set of points $(x_i, f(x_i))$, for $i = 1, 2, \ldots, n$ where $a = x_1 < x_2 < \ldots < x_n = b$, or in the form of a function; and the limits of integration a, b may be finite or infinite.

Some methods are specially designed for integrands of the form

$$f(x) = w(x)g(x) \tag{2}$$

which contain a factor $w(x)$, called the weight-function, of a specific form. These methods take full account of any peculiar behaviour attributable to the $w(x)$ factor.

(b) the values of the one-dimensional indefinite integrals arising from (1) where the ranges of integration are interior to the interval $[a, b]$.

(c) the value of a multi-dimensional definite integral of the form:

$$\int_{R_n} f(x_1, x_2, \ldots, x_n)\,dx_n \ldots dx_2\,dx_1 \tag{3}$$

where $f(x_1, x_2, \ldots, x_n)$ is a function defined by the user and R_n is some region of n-dimensional space.

The simplest form of R_n is the n-rectangle defined by

$$a_i \leq x_i \leq b_i, \quad i = 1, 2, \ldots, n \tag{4}$$

where a_i and b_i are constants. When a_i and b_i are functions of x_j $(j < i)$, the region can easily be transformed to the rectangular form (see Davis and Rabinowitz [1], page 266). Some of the methods described incorporate the transformation procedure.

2.1 One-dimensional Integrals

To estimate the value of a one-dimensional integral, a quadrature rule uses an approximation in the form of a weighted sum of integrand values, i.e.,

$$\int_a^b f(x)\,dx \simeq \sum_{i=1}^{N} w_i f(x_i). \tag{5}$$

The points x_i within the interval $[a, b]$ are known as the abscissae, and the w_i are known as the weights.

More generally, if the integrand has the form (2), the corresponding formula is

$$\int_a^b w(x)g(x)\,dx \simeq \sum_{i=1}^{N} w_i g(x_i). \tag{6}$$

If the integrand is known only at a fixed set of points, these points must be used as the abscissae, and the weighted sum is calculated using finite-difference methods. However, if the functional form of the integrand is known, so that its value at any abscissa is easily obtained, then a wide variety of quadrature rules are available, each characterised by its choice of abscissae and the corresponding weights.

The appropriate rule to use will depend on the interval $[a, b]$ – whether finite or otherwise – and on the form of any $w(x)$ factor in the integrand. A suitable value of N depends on the general behaviour of $f(x)$; or of $g(x)$, if there is a $w(x)$ factor present.

Among possible rules, we mention particularly the Gaussian formulae, which employ a distribution of abscissae which is optimal for $f(x)$ or $g(x)$ of polynomial form.

The choice of basic rules constitutes one of the principles on which methods for one-dimensional integrals may be classified. The other major basis of classification is the implementation strategy, of which some types are now presented.

(a) Single rule evaluation procedures

A fixed number of abscissae, N, is used. This number and the particular rule chosen uniquely determine the weights and abscissae. No estimate is made of the accuracy of the result.

(b) Automatic procedures

The number of abscissae, N, within $[a, b]$ is gradually increased until consistency is achieved to within a level of accuracy (absolute or relative) requested by the user. There are essentially two ways of doing this; hybrid forms of these two methods are also possible:

(i) whole interval procedures (non-adaptive)

A series of rules using increasing values of N are successively applied over the whole interval $[a, b]$. It is clearly more economical if abscissae already used for a lower value of N can be used again as part of a higher-order formula. This principle is known as **optimal extension**. There is no overlap between the abscissae used in Gaussian formulae of different orders. However, the Kronrod formulae are designed to give an optimal $(2N + 1)$-point formula by adding $(N + 1)$ points to an N-point Gauss formula. Further extensions have been developed by Patterson.

(ii) adaptive procedures

The interval $[a, b]$ is repeatedly divided into a number of sub-intervals, and integration rules are applied separately to each sub-interval. Typically, the subdivision process will be carried further in the neighbourhood of a sharp peak in the integrand, than where the curve is smooth. Thus, the distribution of abscissae is adapted to the shape of the integrand.

Subdivision raises the problem of what constitutes an acceptable accuracy in each sub-interval. The usual **global acceptability criterion** demands that the sum of the absolute values of the error estimates in the sub-intervals should meet the conditions required of the error over the whole interval. Automatic extrapolation over several levels of subdivision may eliminate the effects of some types of singularities.

An ideal general-purpose method would be an automatic method which could be used for a wide variety of integrands, was efficient (i.e., required the use of as few abscissae as possible), and was reliable (i.e., always gave results to within the requested accuracy). Complete reliability is unobtainable, and generally higher reliability is obtained at the expense of efficiency, and vice versa. **It must therefore be emphasised that the automatic routines in this chapter cannot be assumed to be 100% reliable. In general, however, the reliability is very high.**

2.2 Multi-dimensional Integrals

A distinction must be made between cases of moderately low dimensionality (say, up to 4 or 5 dimensions), and those of higher dimensionality. Where the number of dimensions is limited, a one-dimensional method may be applied to each dimension, according to some suitable strategy, and high accuracy may be obtainable (using product rules). However, the number of integrand evaluations rises very rapidly with the number of dimensions, so that the accuracy obtainable with an acceptable amount of computational labour is limited; for example a product of 3-point rules in 20 dimensions would require more than 10^9 integrand evaluations. Special techniques such as the Monte Carlo methods can be used to deal with high dimensions.

(a) Products of one-dimensional rules

Using a two-dimensional integral as an example, we have

$$\int_{a_1}^{b_1} \int_{a_2}^{b_2} f(x,y)\, dy\, dx \simeq \sum_{i=1}^{N} w_i \left[\int_{a_2}^{b_2} f(x_i,y)\, dy \right] \tag{7}$$

$$\int_{a_1}^{b_1} \int_{a_2}^{b_2} f(x,y)\, dy\, dx \simeq \sum_{i=1}^{N} \sum_{j=1}^{N} w_i v_j f(x_i,y_j) \tag{8}$$

where (w_i, x_i) and (v_i, y_i) are the weights and abscissae of the rules used in the respective dimensions.

A different one-dimensional rule may be used for each dimension, as appropriate to the range and any weight function present, and a different strategy may be used, as appropriate to the integrand behaviour as a function of each independent variable.

For a rule-evaluation strategy in all dimensions, the formula (8) is applied in a straightforward manner. For automatic strategies (i.e., attempting to attain a requested accuracy), there is a problem in deciding what accuracy must be requested in the inner integral(s). Reference to formula (7) shows that the presence of a limited but random error in the y-integration for different values of x_i can produce a 'jagged' function of x, which may be difficult to integrate to the desired accuracy and for this reason products of automatic one-dimensional routines should be used with caution (see also Lyness [3]).

(b) Monte Carlo methods

These are based on estimating the mean value of the integrand sampled at points chosen from an appropriate statistical distribution function. Usually a variance reducing procedure is incorporated to combat the fundamentally slow rate of convergence of the rudimentary form of the technique. These methods can be effective by comparison with alternative methods when the integrand contains singularities or is erratic in some way, but they are of quite limited accuracy.

(c) Number theoretic methods

These are based on the work of Korobov and Conroy and operate by exploiting implicitly the properties of the Fourier expansion of the integrand. Special rules, constructed from so-called optimal coefficients, give a particularly uniform distribution of the points throughout n-dimensional space and from their number theoretic properties minimize the error on a prescribed class of integrals. The method can be combined with the Monte Carlo procedure.

(d) Sag–Szekeres method

By transformation this method seeks to induce properties into the integrand which make it accurately integrable by the trapezoidal rule. The transformation also allows effective control over the number of integrand evaluations.

(e) Automatic adaptive procedures

An automatic adaptive strategy in several dimensions normally involves division of the region into subregions, concentrating the divisions in those parts of the region where the integrand is worst behaved. It is difficult to arrange with any generality for variable limits in the inner integral(s). For this reason, some methods use a region where all the limits are constants; this is called a hyper-rectangle. Integrals over regions defined by variable or infinite limits may be handled by transformation to a hyper-rectangle. Integrals over regions so irregular that such a transformation is not feasible may be handled by surrounding the region by an appropriate hyper-rectangle and defining the integrand to be zero outside the desired region. Such a technique should always be followed by a Monte Carlo method for integration.

The method used locally in each subregion produced by the adaptive subdivision process is usually one of three types: Monte Carlo, number theoretic or deterministic. Deterministic methods are usually the most rapidly convergent but are often expensive to use for high dimensionality and not as robust as the other techniques.

3 Recommendations on Choice and Use of Available Routines

Note. Refer to the Users' Note for your implementation to check that a routine is available.

The following three sub-sections consider in turn routines for: one-dimensional integrals over a finite interval, and over a semi-infinite or an infinite interval; and multi-dimensional integrals. Within each sub-section, routines are classified by the type of method, which ranges from simple rule evaluation to automatic adaptive algorithms. The recommendations apply particularly when the primary objective is simply to compute the value of one or more integrals, and in these cases the automatic adaptive routines are generally the most convenient and reliable, although also the most expensive in computing time.

Note however that in some circumstances it may be counter-productive to use an automatic routine. If the results of the quadrature are to be used in turn as input to a further computation (e.g. an 'outer' quadrature or an optimization problem), then this further computation may be adversely affected by the 'jagged performance profile' of an automatic routine; a simple rule-evaluation routine may provide much better overall performance. For further guidance, the article by Lyness [3] is recommended.

3.1 One-dimensional Integrals over a Finite Interval

(a) Integrand defined at a set of points

If $f(x)$ is defined numerically at four or more points, then the Gill–Miller finite difference method (D01GAF) should be used. The interval of integration is taken to coincide with the range of x-values of the points supplied. It is in the nature of this problem that any routine may be unreliable. In order to check results independently and so as to provide an alternative technique the user may fit the integrand by Chebyshev series using E02ADF and then use routines E02AJF and E02AKF to evaluate its integral (which need not be restricted to the range of the integration points, as is the case for D01GAF). A further alternative is to fit a cubic spline to the data using E02BAF and then to evaluate its integral using E02BDF.

(b) Integrand defined as a function

If the functional form of $f(x)$ is known, then one of the following approaches should be taken. They are arranged in the order from most specific to most general, hence the first applicable procedure in the list will be the most efficient. **However, if the user does not wish to make any assumptions about the integrand, the most reliable routines to use will be D01AJF (or D01ATF) and D01AHF, although these will in general be less efficient for simple integrals.**

(i) Rule-evaluation routines

If $f(x)$ is known to be sufficiently well behaved (more precisely, can be closely approximated by a polynomial of moderate degree), a Gaussian routine with a suitable number of abscissae may be used.

D01BAF may be used if it is not required to examine the weights and abscissae.

D01BBF or D01BCF with D01FBF may be used if it is required to examine the weights and abscissae.

D01BBF is faster and more accurate, whereas D01BCF is more general.

If $f(x)$ is well behaved, apart from a weight-function of the form

$$\left|x-\frac{a+b}{2}\right|^c \quad \text{or} \quad (b-x)^c(x-a)^d,$$

D01BCF with D01FBF may be used.

(ii) Automatic whole-interval routines

If $f(x)$ is reasonably smooth, and the required accuracy is not too high, the automatic whole-interval routines, D01ARF or D01BDF may be used. D01ARF incorporates high-order extensions of the Kronrod rule and is the only routine which can also be used for indefinite integration.

(iii) Automatic adaptive routines

Firstly, several routines are available for integrands of the form $w(x)g(x)$ where $g(x)$ is a 'smooth' function (i.e., has no singularities, sharp peaks or violent oscillations in the interval of integration) and $w(x)$ is a weight function of one of the following forms:

if $w(x) = (b-x)^{\alpha}(x-a)^{\beta}(\log(b-x))^{k}(\log(x-a))^{l}$, where $k, l = 0$ or 1, $\alpha, \beta > -1$: use D01APF;

if $w(x) = \frac{1}{x-c}$: use D01AQF (this integral is called the Hilbert transform of g);

if $w(x) = \cos(\omega x)$ or $\sin(\omega x)$: use D01ANF (this routine can also handle certain types of singularities in $g(x)$).

Secondly, there are some routines for general $f(x)$. If $f(x)$ is known to be free of singularities, though it may be oscillatory, D01AKF or D01AUF may be used.

The most powerful of the finite interval integration routines are D01AJF and D01ATF, which can cope with singularities of several types, and D01AHF. They may be used if none of the more specific situations described above applies. D01AHF is likely to be more efficient, whereas D01AJF and D01ATF are somewhat more reliable, particularly where the integrand has singularities other than at an end-point, or has discontinuities or cusps, and is therefore recommended where the integrand is known to be badly behaved, or where its nature is completely unknown. It may sometimes be useful to use both routines as a check.

Most of the routines in this chapter require the user to supply a function or subroutine to evaluate the integrand at a single point. D01ATF and D01AUF use the same methods as D01AJF and D01AKF respectively, but have a different user-interface which can result in faster execution, especially on vector-processing machines (see Gladwell [2]). They require the user to provide a subroutine to return an array of values of the integrand at each of an array of points. This reduces the overhead of function calls, avoids repetition of computations common to each of the integrand evaluations, and offers greater scope for vectorisation of the user's code.

If $f(x)$ has singularities of certain types, discontinuities or sharp peaks **occurring at known points**, the integral should be evaluated separately over each of the subranges or D01ALF may be used.

3.2 One-dimensional Integrals over a Semi-infinite or Infinite Interval

(a) Integrand defined at a set of points

If $f(x)$ is defined numerically at four or more points, and the portion of the integral lying outside the range of the points supplied may be neglected, then the Gill–Miller finite difference method, D01GAF, should be used.

(b) Integrand defined as a function

(i) Rule evaluation routines

If $f(x)$ behaves approximately like a polynomial in x, apart from a weight function of the form

$e^{-\beta x}, \beta > 0$ (semi-infinite interval, lower limit finite); or

$e^{-\beta x}, \beta < 0$ (semi-infinite interval, upper limit finite); or

$e^{-\beta(x-\alpha)^2}, \beta > 0$ (infinite interval);

or if $f(x)$ behaves approximately like a polynomial in $(x+b)^{-1}$ (semi-infinite range), then the Gaussian routines may be used.

D01BAF may be used if it is not required to examine the weights and abscissae.

D01BBF or D01BCF with D01FBF may be used if it is required to examine the weights and abscissae.

D01BBF is faster and more accurate, whereas D01BCF is more general.

(ii) Automatic adaptive routines

D01AMF may be used, except for integrands which decay slowly towards an infinite end-point, and oscillate in sign over the entire range. For this class, it may be possible to calculate the integral by integrating between the zeros and invoking some extrapolation process (see C06BAF).

D01ASF may be used for integrals involving weight functions of the form $\cos(\omega x)$ and $\sin(\omega x)$ over a semi-infinite interval (lower limit finite).

The following alternative procedures are mentioned for completeness, though their use will rarely be necessary.

1. If the integrand decays rapidly towards an infinite end-point, a finite cut-off may be chosen, and the finite range methods applied.
2. If the only irregularities occur in the finite part (apart from a singularity at the finite limit, with which D01AMF can cope), the range may be divided, with D01AMF used on the infinite part.
3. A transformation to finite range may be employed, e.g.

$$x = \frac{1-t}{t} \text{ or } x = -\log_e t$$

will transform $(0,\infty)$ to (1,0) while for infinite ranges we have

$$\int_{-\infty}^{\infty} f(x)\,dx = \int_0^{\infty} [f(x) + f(-x)]\,dx.$$

If the integrand behaves badly on $(-\infty,0)$ and well on $(0,\infty)$ or vice versa it is better to compute it as $\int_{-\infty}^{0} f(x)\,dx + \int_0^{\infty} f(x)\,dx$. This saves computing unnecessary function values in the semi-infinite range where the function is well behaved.

3.3 Multi-dimensional Integrals

A number of techniques are available in this area and the choice depends to a large extent on the dimension and the required accuracy. It can be advantageous to use more than one technique as a confirmation of accuracy particularly for high dimensional integrations. Many of the routines incorporate the transformation procedure REGION which allows general product regions to be easily dealt with in terms of conversion to the standard n-cube region.

(a) Products of one-dimensional rules (suitable for up to about 5 dimensions)

If $f(x_1, x_2, \ldots, x_n)$ is known to be a sufficiently well behaved function of each variable x_i, apart possibly from weight functions of the types provided, a product of Gaussian rules may be used. These are provided by D01BBF or D01BCF with D01FBF. Rules for finite, semi-infinite and infinite ranges are included.

For two-dimensional integrals only, unless the integrand is very badly-behaved, the automatic whole-interval product procedure of D01DAF may be used. The limits of the inner integral may be user-specified functions of the outer variable. Infinite limits may be handled by transformation (see Section 3.2); end-point singularities introduced by transformation should not be troublesome, as the integrand value will not be required on the boundary of the region.

If none of these routines proves suitable and convenient, the one-dimensional routines may be used recursively. For example, the two-dimensional integral

$$I = \int_{a_1}^{b_1} \int_{a_2}^{b_2} f(x,y)\,dy\,dx$$

may be expressed as

$$I = \int_{a_1}^{b_1} F(x)\,dx, \quad \text{where } F(x) = \int_{a_2}^{b_2} f(x,y)\,dy.$$

The user segment to evaluate $F(x)$ will call the integration routine for the y-integration, which will call another user segment for $f(x,y)$ as a function of y (x being effectively a constant). Note that, as Fortran 77 is not a recursive language, **a different library integration routine must be used for each dimension**. Apart from this restriction, the following combinations are not permitted: D01AJF and D01ALF, D01ANF and D01APF, D01APF and D01AQF, D01AQF and D01ANF, D01ASF and D01ANF, D01ASF and D01AMF, D01AUF and D01ATF. Otherwise the full range of one-dimensional routines are available, for finite/infinite intervals, constant/variable limits, rule evaluation/automatic strategies etc.

(b) Sag–Szekeres method

Two routines are based on this method.

D01FDF is particularly suitable for integrals of very large dimension although the accuracy is generally not high. It allows integration over either the general product region (with built-in transformation to the n-cube) or the n-sphere. Although no error estimate is provided, two adjustable parameters may be varied for checking purposes or may be used to tune the algorithm to particular integrals.

D01JAF is also based on the Sag–Szekeres method and integrates over the n-sphere. It uses improved transformations which may be varied according to the behaviour of the integrand. Although it can yield very accurate results it can only practically be employed for dimensions not exceeding 4.

(c) Number Theoretic method

Two routines are based on this method.

D01GCF carries out multiple integration using the Korobov–Conroy method over a product region with built-in transformation to the n-cube. A stochastic modification of this method is incorporated hybridising the technique with the Monte Carlo procedure. An error estimate is provided in terms of the statistical standard error. The routine includes a number of optimal coefficient rules for up to 20 dimensions; others can be computed using D01GYF and D01GZF. Like the Sag–Szekeres method it is suitable for large dimensional integrals although the accuracy is not high.

D01GDF uses the same method as D01GCF, but has a different interface which can result in faster execution, especially on vector-processing machines. The user is required to provide two subroutines, the first to return an array of values of the integrand at each of an array of points, and the second to evaluate the limits of integration at each of an array of points. This reduces the overhead of function calls, avoids repetitions of computations common to each of the evaluations of the integral and limits of integration, and offers greater scope for vectorization of the user's code.

(d) A combinatorial extrapolation method

D01PAF computes a sequence of approximations and an error estimate to the integral of a function over a multi-dimensional simplex using a combinatorial method with extrapolation.

(e) Automatic routines (D01GBF and D01FCF)

Both routines are for integrals of the form

$$\int_{a_1}^{b_1}\int_{a_2}^{b_2}\cdots\int_{a_n}^{b_n} f(x_1,x_2,\ldots,x_n)\,dx_n\,dx_{n-1}\,\ldots\,dx_1.$$

D01GBF is an adaptive Monte Carlo routine. This routine is usually slow and not recommended for high-accuracy work. It is a robust routine that can often be used for low-accuracy results with highly irregular integrands or when n is large.

D01FCF is an adaptive deterministic routine. Convergence is fast for well behaved integrands. Highly accurate results can often be obtained for n between 2 and 5, using significantly fewer integrand evaluations than would be required by D01GBF. The routine will usually work when the integrand is mildly singular and for $n \leq 10$ should be used before D01GBF. If it is known in advance that the integrand is highly irregular, it is best to compare results from at least two different routines.

There are many problems for which one or both of the routines will require large amounts of computing time to obtain even moderately accurate results. The amount of computing time is controlled by the number of integrand evaluations allowed by the user, and users should set this parameter carefully, with reference to the time available and the accuracy desired.

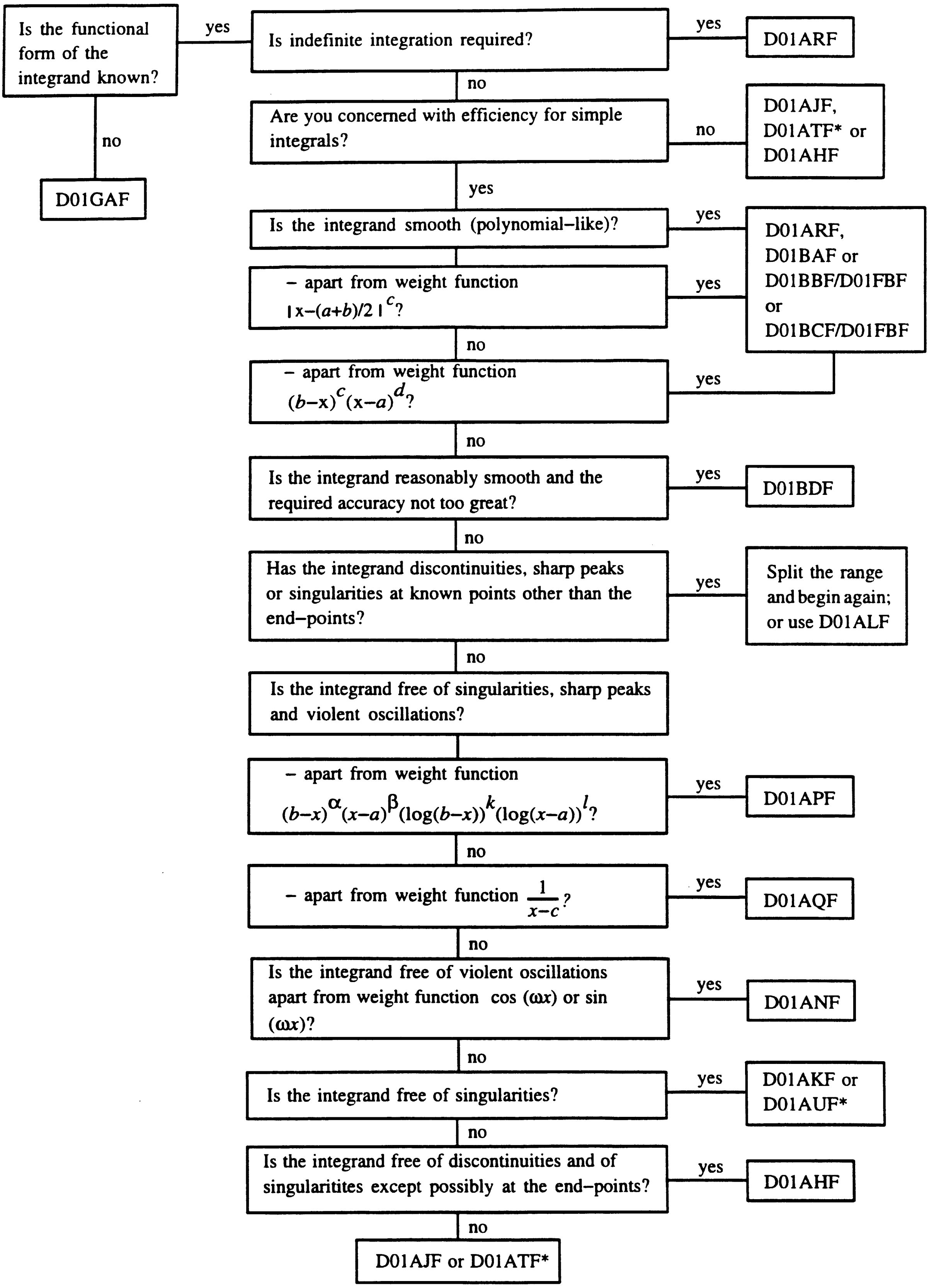
D01EAF extends the technique of D01FCF to integrate adaptively more than one integrand, that is to calculate the set of integrals

$$\int_{a_1}^{b_1} \int_{a_2}^{b_2} \cdots \int_{a_n}^{b_n} (f_1, f_2, \ldots, f_m) \, dx_n \, dx_{n-1} \cdots dx_1$$

for a set of similar integrands $f_1, f_2, \ldots, f_m$ where $f_i = f_i(x_1, x_2, \ldots, x_n)$.

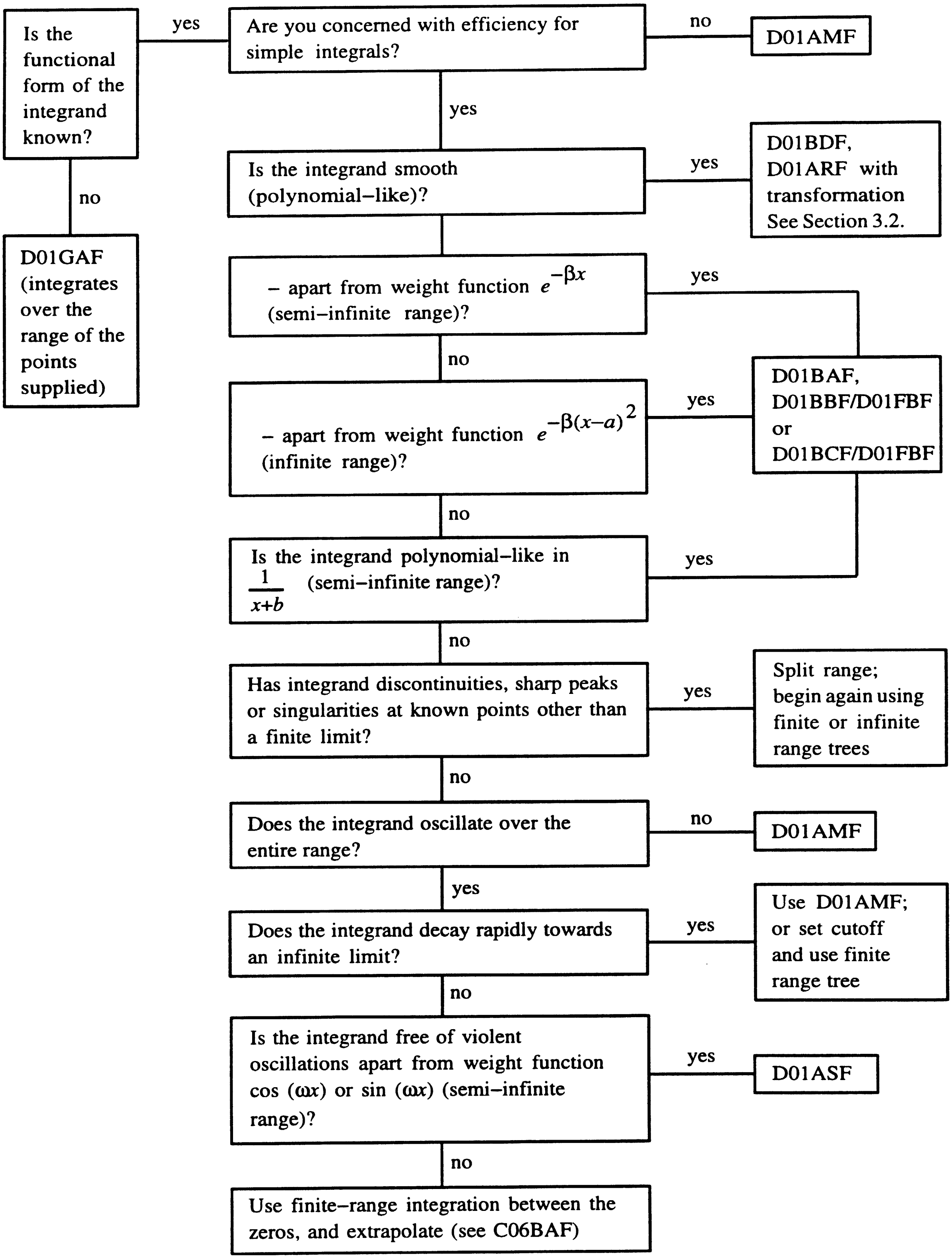
4 Decision Trees

(i) One-dimensional integrals over a finite interval
(If in doubt, follow the downward branch.)

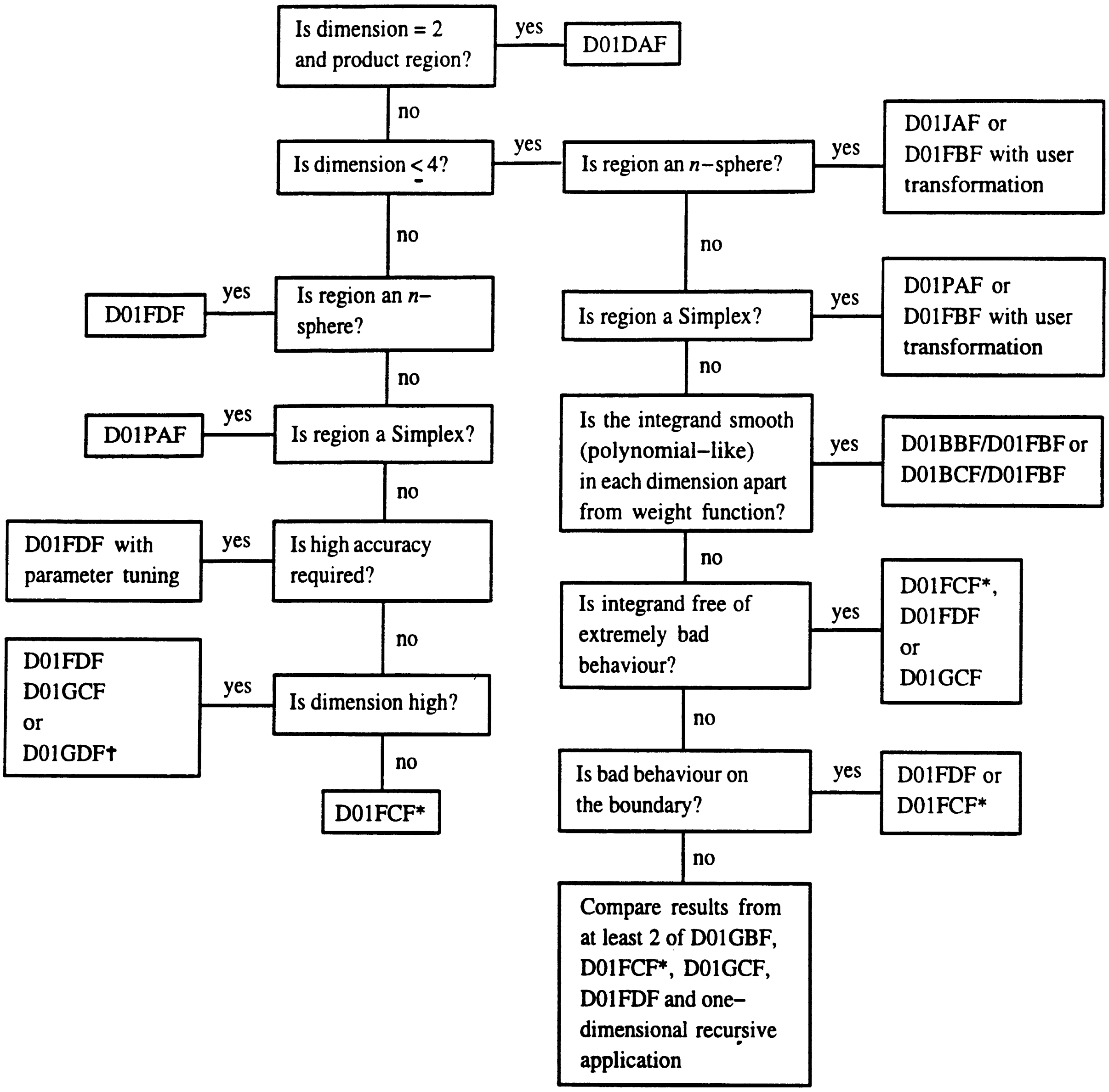

Is the functional form of the integrand known?
yes
no
D01GAF
Is indefinite integration required?
yes
D01ARF
no
Are you concerned with efficiency for simple integrals?
no
D01AJF, D01ATF* or D01AHF
yes
Is the integrand smooth (polynomial–like)?
yes
D01ARF, D01BAF or D01BBF/D01FBF or D01BCF/D01FBF
– apart from weight function $|x-(a+b)/2|^c$?
yes
no
– apart from weight function $(b-x)^c(x-a)^d$?
yes
no
Is the integrand reasonably smooth and the required accuracy not too great?
yes
D01BDF
no
Has the integrand discontinuities, sharp peaks or singularities at known points other than the end–points?
yes
Split the range and begin again; or use D01ALF
no
Is the integrand free of singularities, sharp peaks and violent oscillations?
– apart from weight function $(b-x)^\alpha(x-a)^\beta(\log(b-x))^k(\log(x-a))^l$?
yes
D01APF
no
– apart from weight function $\frac{1}{x-c}$?
yes
D01AQF
no
Is the integrand free of violent oscillations apart from weight function cos (ωx) or sin (ωx)?
yes
D01ANF
no
Is the integrand free of singularities?
yes
D01AKF or D01AUF*
no
Is the integrand free of discontinuities and of singularitites except possibly at the end–points?
yes
D01AHF
no
D01AJF or D01ATF*

*D01ATF and D01AUF are likely to be more efficient than D01AJF and D01AKF, which use a more conventional user-interface, consistent with other routines in the chapter.

(ii) One-dimensional integrals over a semi-infinite or infinite interval
(If in doubt, follow the donward branch.)


Is the functional form of the integrand known?
yes
no
D01GAF (integrates over the range of the points supplied)
Are you concerned with efficiency for simple integrals?
no
D01AMF
yes
Is the integrand smooth (polynomial–like)?
yes
D01BDF, D01ARF with transformation See Section 3.2.
– apart from weight function $e^{-\beta x}$ (semi–infinite range)?
yes
no
– apart from weight function $e^{-\beta(x-a)^2}$ (infinite range)?
yes
D01BAF, D01BBF/D01FBF or D01BCF/D01FBF
no
Is the integrand polynomial–like in $\frac{1}{x+b}$ (semi–infinite range)?
yes
no
Has integrand discontinuities, sharp peaks or singularities at known points other than a finite limit?
yes
Split range; begin again using finite or infinite range trees
no
Does the integrand oscillate over the entire range?
no
D01AMF
yes
Does the integrand decay rapidly towards an infinite limit?
yes
Use D01AMF; or set cutoff and use finite range tree
no
Is the integrand free of violent oscillations apart from weight function cos (ωx) or sin (ωx) (semi–infinite range)?
yes
D01ASF
no
Use finite–range integration between the zeros, and extrapolate (see C06BAF)

(iii) Multi-dimensional integrals


Is dimension = 2 and product region?
yes
D01DAF
no
Is dimension ≤ 4?
yes
Is region an n–sphere?
yes
D01JAF or D01FBF with user transformation
no
no
D01FDF
yes
Is region an n–sphere?
no
Is region a Simplex?
yes
D01PAF or D01FBF with user transformation
no
D01PAF
yes
Is region a Simplex?
no
Is the integrand smooth (polynomial–like) in each dimension apart from weight function?
yes
D01BBF/D01FBF or D01BCF/D01FBF
D01FDF with parameter tuning
yes
Is high accuracy required?
no
no
Is integrand free of extremely bad behaviour?
yes
D01FCF*, D01FDF or D01GCF
D01FDF D01GCF or D01GDF†
yes
Is dimension high?
no
no
D01FCF*
Is bad behaviour on the boundary?
yes
D01FDF or D01FCF*
no
Compare results from at least 2 of D01GBF, D01FCF*, D01GCF, D01FDF and one–dimensional recursive application

* In the case where there are many integrals to be evaluated D01EAF should be preferred to D01FCF.

† D01GDF is likely to be more efficient than D01GCF, which uses a more conventional user-interface, consistent with other routines in the chapter.

5 References

[1] Davis P J and Rabinowitz P (1975) *Methods of Numerical Integration* Academic Press

[2] Gladwell I (1986) Vectorisation of one dimensional quadrature codes *Numerical Integration: Recent Developments, and Applications* (ed P Keast and G Fairweather) D Reidel Publishing Company, Holland 231–238

[3] Lyness J N (1983) When not to use an automatic quadrature routine *SIAM Rev.* **25** 63–87

[4] Piessens R, De Doncker-Kapenga E, Überhuber C and Kahaner D (1983) *QUADPACK, A Subroutine Package for Automatic Integration* Springer-Verlag

[5] Sobol I M (1974) *The Monte Carlo Method* The University of Chicago Press

[6] Stroud A H (1971) *Approximate Calculation of Multiple Integrals* Prentice-Hall

D01AHF – NAG Fortran Library Routine Document

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

1. Purpose

D01AHF computes a definite integral over a finite range to a specified relative accuracy using a method described by Patterson.

2. Specification

```
real FUNCTION D01AHF (A, B, EPSR, NPTS, RELERR, F, NLIMIT, IFAIL)
INTEGER          NPTS, NLIMIT, IFAIL
real             A, B, EPSR, RELERR, F
EXTERNAL         F
```

3. Description

This routine computes a definite integral of the form

$$\int_a^b f(x)\ dx.$$

The method uses as its basis a family of interlacing high precision rules (see Patterson [1]) using 1, 3, 7, 15, 31, 63, 127 and 255 nodes. Initially the family is applied in sequence to the integrand. When two successive rules differ relatively by less than the required relative accuracy, the last rule used is taken as the value of the integral and the operation is regarded as successful. If all rules in the family have been applied unsuccessfully, subdivision is invoked. The subdivision strategy is as follows. The interval under scrutiny is divided into two subintervals (not always equal). The basic family is then applied to the first subinterval. If the required accuracy is not obtained, the interval is stored for future examination (see IFAIL = 2) and the second subinterval is examined. Should the basic family again be unsuccessful, then the subinterval is further subdivided and the whole process repeated. Successful integrations are accumulated as the partial value of the integral. When all possible successful integrations have been completed, those previously unsuccessful subintervals placed in store are examined.

A large number of refinements are incorporated to improve the performance. Some of these are:

(a) The rate of convergence of the basic family is monitored and used to make a decision to abort and subdivide before the full sequence has been applied.

(b) The ε-algorithm is applied to the basic results in an attempt to increase the convergence rate. (See Wynn [2]).

(c) An attempt is made to detect sharp end point peaks and singularities in each subinterval and to apply appropriate transformations to smooth the integrand. This consideration is also used to select interval sizes in the subdivision process.

(d) The relative accuracy sought in each subinterval is adjusted in accordance with its likely contribution to the total integral.

(e) Random transformations of the integrand are applied to improve reliability in some instances.

4. References

[1] PATTERSON, T.N.L.
The Optimum Addition of Points to Quadrature Formulae.
Math. Comp., 22, pp. 847-856, 1968.

[2] WYNN, P.
On a Device for Computing the $e_m(S_n)$ Transformation.
Math. Tables Aids Comp., 10, pp. 91-96, 1956.

5. Parameters

1: A – ***real.*** *Input*

On entry: the lower limit of integration, a.

2: B – ***real.*** *Input*

On entry: the upper limit of integration, b. It is not necessary that $a < b$.

3: EPSR – ***real.*** *Input*

On entry: the relative accuracy required.

Constraint: EPSR > 0.0.

4: NPTS – INTEGER. *Output*

On exit: the number of function evaluations used in the calculation of the integral.

5: RELERR – ***real.*** *Output*

On exit: a rough estimate of the relative error achieved.

6: F – ***real*** FUNCTION, supplied by the user. *External Procedure*

F must return the value of the integrand f at a given point.

Its specification is:

```
real FUNCTION F(X)
real          X
```

1: X – ***real.*** *Input*

On entry: the point at which the integrand must be evaluated.

F must be declared as EXTERNAL in the (sub)program from which D01AHF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: NLIMIT – INTEGER. *Input*

On entry: a limit to the number of function evaluations. If NLIMIT ≤ 0, the routine uses a default limit of 10,000.

8: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.** To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL = 1

The integral has not converged to the accuracy requested. It may be worthwhile to try increasing NLIMIT.

IFAIL = 2

Too many unsuccessful levels of subdivision have been invoked.

IFAIL = 3

On entry, EPSR ≤ 0.0.

When IFAIL = 1 or 2 a result may be obtained by continuing without further subdivision, but this is likely to be **inaccurate**.

7. Accuracy

The relative accuracy required is specified by the user in the variable EPSR. The routine will terminate whenever the relative accuracy specified by EPSR is judged to have been reached.

If on exit, IFAIL = 0, then it is most likely that the result is correct to the specified accuracy. If, on exit, IFAIL = 1 or IFAIL = 2, then it is likely that the specified accuracy has not been reached.

RELERR is a rough estimate of the relative error achieved. It is a by-product of the computation and is not used to effect the termination of the routine. The outcome of the integration must be judged by the value of IFAIL.

8. Further Comments

The time taken by the routine depends on the complexity of the integrand and the accuracy required.

9. Example

The following program evaluates the integral to a requested relative accuracy of 10^{-5}

$$\int_0^1 \frac{4}{1+x^2}dx = \pi.$$

9.1. Program Text

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

```
*     D01AHF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             A, ANS, B, EPSR, RELERR
      INTEGER          IFAIL, N, NLIMIT
*     .. External Functions ..
      real             D01AHF, FUN
      EXTERNAL         D01AHF, FUN
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01AHF Example Program Results'
      A = 0.0e0
      B = 1.0e0
      NLIMIT = 0
      EPSR = 1.0e-5
      IFAIL = 1
      ANS = D01AHF(A,B,EPSR,N,RELERR,FUN,NLIMIT,IFAIL)
      WRITE (NOUT,*)
      IF (IFAIL.NE.0) THEN
         WRITE (NOUT,99997) 'IFAIL = ', IFAIL
         WRITE (NOUT,*)
      END IF
```

```
         IF (IFAIL.LE.2) THEN
            WRITE (NOUT,99999) 'Integral = ', ANS
            WRITE (NOUT,*)
            WRITE (NOUT,99998) 'Estimated relative error = ', RELERR
            WRITE (NOUT,*)
            WRITE (NOUT,99997) 'Number of function evaluations = ', N
         END IF
         STOP
*
99999 FORMAT (1X,A,F8.5)
99998 FORMAT (1X,A,e10.2)
99997 FORMAT (1X,A,I4)
      END
*
      real FUNCTION FUN(X)
*     .. Scalar Arguments ..
      real              X
*     .. Executable Statements ..
      FUN = 4.0e0/(1.0e0+X*X)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D01AHF Example Program Results

 Integral =  3.14159

 Estimated relative error =   0.58E-08

 Number of function evaluations =   15
```

D01AJF – NAG Fortran Library Routine Document

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

1. Purpose

D01AJF is a general-purpose integrator which calculates an approximation to the integral of a function $f(x)$ over a finite interval $[a,b]$:

$$I = \int_a^b f(x)\ dx.$$

2. Specification

```
 SUBROUTINE D01AJF (F, A, B, EPSABS, EPSREL, RESULT, ABSERR, W,
1                   LW, IW, LIW, IFAIL)
 INTEGER        LW, IW(LIW), LIW, IFAIL
 real           F, A, B, EPSABS, EPSREL, RESULT, ABSERR,
1               W(LW)
 EXTERNAL       F
```

3. Description

D01AJF is based upon the QUADPACK routine QAGS (Piessens *et al.* [3]). It is an adaptive routine, using the Gauss 10-point and Kronrod 21-point rules. The algorithm, described by de Doncker [1], incorporates a global acceptance criterion (as defined by Malcolm and Simpson [2]) together with the ε–algorithm (Wynn [4]) to perform extrapolation. The local error estimation is described by Piessens *et al.* [3].

The routine is suitable as a general purpose integrator, and can be used when the integrand has singularities, especially when these are of algebraic or logarithmic type.

D01AJF requires the user to supply a function to evaluate the integrand at a single point.

The routine D01ATF uses an identical algorithm but requires the user to supply a subroutine to evaluate the integrand at an array of points. Therefore D01ATF will be more efficient if the evaluation can be performed in vector mode on a vector-processing machine.

4. References

[1] DE DONCKER, E.
An Adaptive Extrapolation Algorithm for Automatic Integration.
Signum Newsletter, 13, 2, pp. 12-18, 1978.

[2] MALCOLM, M.A. and SIMPSON, R.B.
Local Versus Global Strategies for Adaptive Quadrature.
A.C.M. Trans. Math. Software, 1, pp. 129-146, 1976.

[3] PIESSENS, R., DE DONCKER-KAPENGA, E., ÜBERHUBER, C. and KAHANER, D.
QUADPACK, A Subroutine Package for Automatic Integration.
Springer-Verlag, 1983.

[4] WYNN, P.
On a Device for Computing the $e_m(S_n)$ Transformation.
Math. Tables Aids Comp., 10, pp. 91-96, 1956.

5. Parameters

1: F – ***real*** FUNCTION, supplied by the user. *External Procedure*

F must return the value of the integrand f at a given point.

Its specification is:

```
real FUNCTION F(X)
real          X
```

1: X – ***real.*** *Input*

On entry: the point at which the integrand f must be evaluated.

F must be declared as EXTERNAL in the (sub)program from which D01AJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: A – ***real.*** *Input*

On entry: the lower limit of integration, a.

3: B – ***real.*** *Input*

On entry: the upper limit of integration, b. It is not necessary that $a < b$.

4: EPSABS – ***real.*** *Input*

On entry: the absolute accuracy required. If EPSABS is negative, the absolute value is used. See Section 7.

5: EPSREL – ***real.*** *Input*

On entry: the relative accuracy required. If EPSREL is negative, the absolute value is used. See Section 7.

6: RESULT – ***real.*** *Output*

On exit: the approximation to the integral I.

7: ABSERR – ***real.*** *Output*

On exit: an estimate of the modulus of the absolute error, which should be an upper bound for $|I\text{–RESULT}|$.

8: W(LW) – ***real*** array. *Output*

On exit: details of the computation, as described in Section 8.

9: LW – INTEGER. *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D01AJF is called. The value of LW (together with that of LIW below) imposes a bound on the number of subintervals into which the interval of integration may be divided by the routine. The number of subintervals cannot exceed LW/4. The more difficult the integrand, the larger LW should be.

Suggested value: a value in the range 800 to 2000 is adequate for most problems.

Constraint: LW ≥ 4.

10: IW(LIW) – INTEGER array. *Output*

On exit: IW(1) contains the actual number of subintervals used. The rest of the array is used as workspace.

11: LIW – INTEGER. *Input*

On entry: the dimension of the array IW as declared in the (sub)program from which D01AJF is called. The number of subintervals into which the interval of integration may be divided cannot exceed LIW.

Suggested value: LIW = LW/4.

Constraint: LIW ≥ 1.

12: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL = 1

The maximum number of subdivisions allowed with the given workspace has been reached without the accuracy requirements being achieved. Look at the integrand in order to determine the integration difficulties. If the position of a local difficulty within the interval can be determined (e.g. a singularity of the integrand or its derivative, a peak, a discontinuity, etc.) you will probably gain from splitting up the interval at this point and calling the integrator on the subranges. If necessary, another integrator, which is designed for handling the type of difficulty involved, must be used. Alternatively, consider relaxing the accuracy requirements specified by EPSABS and EPSREL, or increasing the amount of workspace.

IFAIL = 2

Roundoff error prevents the requested tolerance from being achieved. The error may be under-estimated. Consider requesting less accuracy.

IFAIL = 3

Extremely bad local integrand behaviour causes a very strong subdivision around one (or more) points of the interval. The same advice applies as in the case of IFAIL = 1.

IFAIL = 4

The requested tolerance cannot be achieved, because the extrapolation does not increase the accuracy satisfactorily; the returned result is the best which can be obtained. The same advice applies as in the case of IFAIL = 1.

IFAIL = 5

The integral is probably divergent, or slowly convergent. Please note that divergence can occur with any non-zero value of IFAIL.

IFAIL = 6

On entry, LW < 4,
or LIW < 1.

7. Accuracy

The routine cannot guarantee, but in practice usually achieves, the following accuracy:

$$|I-\text{RESULT}| \le tol$$

where

$$tol = \max\{|\text{EPSABS}|,|\text{EPSREL}|\times|I|\}$$

and EPSABS and EPSREL are user-specified absolute and relative error tolerance. Moreover it returns the quantity ABSERR which, in normal circumstances, satisfies

$$|I-\text{RESULT}| \le \text{ABSERR} \le tol.$$

8. Further Comments

The time taken by the routine depends on the integrand and the accuracy required.

If IFAIL $\neq 0$ on exit, then the user may wish to examine the contents of the array W, which contains the end-points of the subintervals used by D01AJF along with the integral contributions and error estimates over the subintervals.

Specifically, for $i = 1,2,...,n$, let r_i denote the approximation to the value of the integral over the subinterval $[a_i,b_i]$ in the partition of $[a,b]$ and e_i be the corresponding absolute error estimate. Then, $\int_{a_i}^{b_i} f(x)\ dx \simeq r_i$ and RESULT $= \sum_{i=1}^{n} r_i$, unless D01AJF terminates while testing for divergence of the integral (see Piessens *et al.* [3], Section 3.4.3). In this case, RESULT (and ABSERR) are taken to be the values returned from the extrapolation process. The value of n is returned in IW(1), and the values a_i, b_i, e_i and r_i are stored consecutively in the array W, that is:

a_i = W(i),
b_i = W($n+i$),
e_i = W($2n+i$) and
r_i = W($3n+i$).

9. Example

To compute

$$\int_0^{2\pi} \frac{x\sin(30x)}{\sqrt{\left(1-\left(\frac{x}{2\pi}\right)^2\right)}}\ dx.$$

9.1. Program Text

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

```
*     D01AJF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          LW, LIW
      PARAMETER        (LW=800,LIW=LW/4)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      real             PI
      INTEGER          KOUNT
*     .. Local Scalars ..
      real             A, ABSERR, B, EPSABS, EPSREL, RESULT
      INTEGER          IFAIL
```

```
*     .. Local Arrays ..
      real             W(LW)
      INTEGER          IW(LIW)
*     .. External Functions ..
      real             FST, X01AAF
      EXTERNAL         FST, X01AAF
*     .. External Subroutines ..
      EXTERNAL         D01AJF
*     .. Common blocks ..
      COMMON           /TELNUM/PI, KOUNT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01AJF Example Program Results'
      PI = X01AAF(PI)
      EPSABS = 0.0e0
      EPSREL = 1.0e-04
      A = 0.0e0
      B = 2.0e0*PI
      KOUNT = 0
      IFAIL = -1
*
      CALL D01AJF(FST,A,B,EPSABS,EPSREL,RESULT,ABSERR,W,LW,IW,LIW,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'A      - lower limit of integration = ', A
      WRITE (NOUT,99999) 'B      - upper limit of integration = ', B
      WRITE (NOUT,99998) 'EPSABS - absolute accuracy requested = ',
     +  EPSABS
      WRITE (NOUT,99998) 'EPSREL - relative accuracy requested = ',
     +  EPSREL
      WRITE (NOUT,*)
      IF (IFAIL.NE.0) WRITE (NOUT,99996) 'IFAIL = ', IFAIL
      IF (IFAIL.LE.5) THEN
         WRITE (NOUT,99997) 'RESULT - approximation to the integral = ',
     +     RESULT
         WRITE (NOUT,99998) 'ABSERR - estimate of the absolute error = '
     +     , ABSERR
         WRITE (NOUT,99996) 'KOUNT  - number of function evaluations = '
     +     , KOUNT
         WRITE (NOUT,99996) 'IW(1)  - number of subintervals used = ',
     +     IW(1)
      END IF
      STOP
*
99999 FORMAT (1X,A,F10.4)
99998 FORMAT (1X,A,e9.2)
99997 FORMAT (1X,A,F9.5)
99996 FORMAT (1X,A,I4)
      END
*
      real FUNCTION FST(X)
*     .. Scalar Arguments ..
      real             X
*     .. Scalars in Common ..
      real             PI
      INTEGER          KOUNT
*     .. Intrinsic Functions ..
      INTRINSIC        SIN, SQRT
*     .. Common blocks ..
      COMMON           /TELNUM/PI, KOUNT
*     .. Executable Statements ..
      KOUNT = KOUNT + 1
      FST = X*SIN(30.0e0*X)/SQRT(1.0e0-X**2/(4.0e0*PI**2))
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D01AJF Example Program Results

A      - lower limit of integration =     0.0000
B      - upper limit of integration =     6.2832
EPSABS - absolute accuracy requested =  0.00E+00
EPSREL - relative accuracy requested =  0.10E-03

RESULT - approximation to the integral =  -2.54326
ABSERR - estimate of the absolute error =  0.13E-04
KOUNT  - number of function evaluations =  777
IW(1)  - number of subintervals used =   19
```

D01AKF – NAG Fortran Library Routine Document

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

1. Purpose

D01AKF is an adaptive integrator, especially suited to oscillating, non-singular integrands, which calculates an approximation to the integral of a function $f(x)$ over a finite interval $[a,b]$:

$$I = \int_a^b f(x)\, dx.$$

2. Specification

```
      SUBROUTINE D01AKF (F, A, B, EPSABS, EPSREL, RESULT, ABSERR, W,
     1                   LW, IW, LIW, IFAIL)
      INTEGER         LW, IW(LIW), LIW, IFAIL
      real            F, A, B, EPSABS, EPSREL, RESULT, ABSERR,
     1                W(LW)
      EXTERNAL        F
```

3. Description

D01AKF is based upon the QUADPACK routine QAG (Piessens *et al.* [3]). It is an adaptive routine, using the Gauss 30-point and Kronrod 61-point rules. A 'global' acceptance criterion (as defined by Malcolm and Simpson [1]) is used. The local error estimation is described in by Piessens *et al.* [3].

Because this routine is based on integration rules of high order, it is especially suitable for non-singular oscillating integrands.

D01AKF requires the user to supply a function to evaluate the integrand at a single point.

The routine D01AUF uses an identical algorithm but requires the user to supply a subroutine to evaluate the integrand at an array of points. Therefore D01AUF will be more efficient if the evaluation can be performed in vector mode on a vector-processing machine.

D01AUF also has an additional parameter KEY which allows the user to select from six different Gauss-Kronrod rules.

4. References

[1] MALCOLM, M.A. and SIMPSON, R.B.
Local Versus Global Strategies for Adaptive Quadrature.
A.C.M. Trans. Math. Software, 1, pp. 129-146, 1975.

[2] PIESSENS, R.
An Algorithm for Automatic Integration.
Angewandte Informatik, 15, pp. 399-401, 1973.

[3] PIESSENS, R., De DONCKER-KAPENGA, E., ÜBERHUBER, C. and KAHANER, D.
QUADPACK, A Subroutine Package for Automatic Integration.
Springer-Verlag, 1983.

5. Parameters

1: F – ***real*** FUNCTION, supplied by the user. *External Procedure*

F must return the value of the integrand f at a given point.

Its specification is:

```
real FUNCTION F(X)
real          X
```

1: X – ***real.*** *Input*

On entry: the point at which the integrand f must be evaluated.

F must be declared as EXTERNAL in the (sub)program from which D01AKF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: A – ***real.*** *Input*

On entry: the lower limit of integration, a.

3: B – ***real.*** *Input*

On entry: the upper limit of integration, b. It is not necessary that $a < b$.

4: EPSABS – ***real.*** *Input*

On entry: the absolute accuracy required. If EPSABS is negative, the absolute value is used. See Section 7.

5: EPSREL – ***real.*** *Input*

On entry: the relative accuracy required. If EPSREL is negative, the absolute value is used. See Section 7.

6: RESULT – ***real.*** *Output*

On exit: the approximation to the integral I.

7: ABSERR – ***real.*** *Output*

On exit: an estimate of the modulus of the absolute error, which should be an upper bound $|I$–RESULT$|$.

8: W(LW) – ***real*** array. *Output*

On exit: details of the computation, as described in Section 8.

9: LW – INTEGER. *Input*

On entry: the dimension of W, as declared in the (sub)program from which D01AKF is called. The value of LW (together with that of LIW below) imposes a bound on the number of subintervals into which the interval of integration may be divided by the routine. The number of subintervals cannot exceed LW/4. The more difficult the integrand, the larger LW should be.

Suggested value: a value in the range 800 to 2000 is adequate for most problems.

Constraint: LW $\geq$ 4. See IW below.

10: IW(LIW) – INTEGER array. *Output*

On exit: IW(1) contains the actual number of subintervals used. The rest of the array is used as workspace.

11: LIW – INTEGER. *Input*

On entry: the dimension of the array IW as declared in the (sub)program from which D01AKF is called. The number of subintervals into which the interval of integration may be divided cannot exceed LIW.

Suggested value: LIW = LW/4.

Constraint: LIW ≥ 1.

12: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL = 1

The maximum number of subdivisions allowed with the given workspace has been reached without the accuracy requirements being achieved. Look at the integrand in order to determine the integration difficulties. Probably another integrator which is designed for handling the type of difficulty involved must be used. Alternatively, consider relaxing the accuracy requirements specified by EPSABS and EPSREL, or increasing the amount of workspace.

IFAIL = 2

Roundoff error prevents the requested tolerance from being achieved. Consider requesting less accuracy.

IFAIL = 3

Extremely bad local integrand behaviour causes a very strong subdivision around one (or more) points of the interval. The same advice applies as in the case of IFAIL = 1.

IFAIL = 4

On entry, LW < 4,
or LIW < 1.

7. Accuracy

The routine cannot guarantee, but in practice usually achieves, the following accuracy:

$$|I\text{–RESULT}| \le tol$$

where

$$tol = \max\{|\text{EPSABS}|,|\text{EPSREL}|\times|I|\},$$

and EPSABS and EPSREL are user-specified absolute and relative error tolerances. Moreover it returns the quantity ABSERR which, in normal circumstances satisfies

$$|I\text{–RESULT}| \le \text{ABSERR} \le tol.$$

8. Further Comments

The time taken by the routine depends on the integrand and the accuracy required.

If IFAIL $\neq$ 0 on exit, then the user may wish to examine the contents of the array W, which contains the end-points of the subintervals used by D01AKF along with the integral contributions and error estimates over these subintervals.

Specifically, for $i = 1,2,...,n$, let r_i denote the approximation to the value of the integral over the subinterval $[a_i,b_i]$ in the partition of $[a,b]$ and e_i be the corresponding absolute error estimate. Then, $\int_{a_i}^{b_i} f(x)dx \simeq r_i$ and RESULT $= \sum_{i=1}^{n} r_i$. The value of n is returned in IW(1), and the values a_i, b_i, e_i and r_i are stored consecutively in the array W, that is:

a_i = W(i),
b_i = $W(n+i)$,
e_i = W($2n+i$) and
r_i = W($3n+i$).

9. Example

To compute

$$\int_0^{2\pi} x \sin(30x) \cos x \, dx.$$

9.1. Program Text

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

```
*     D01AKF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          LW, LIW
      PARAMETER        (LW=800,LIW=LW/4)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      INTEGER          KOUNT
*     .. Local Scalars ..
      real             A, ABSERR, B, EPSABS, EPSREL, PI, RESULT
      INTEGER          IFAIL
*     .. Local Arrays ..
      real             W(LW)
      INTEGER          IW(LIW)
*     .. External Functions ..
      real             FST, X01AAF
      EXTERNAL         FST, X01AAF
*     .. External Subroutines ..
      EXTERNAL         D01AKF
*     .. Common blocks ..
      COMMON           /TELNUM/KOUNT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01AKF Example Program Results'
      PI = X01AAF(PI)
      EPSABS = 0.0e0
      EPSREL = 1.0e-03
      A = 0.0e0
      B = 2.0e0*PI
      KOUNT = 0
      IFAIL = -1
*
```

```
      CALL D01AKF(FST,A,B,EPSABS,EPSREL,RESULT,ABSERR,W,LW,IW,LIW,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'A      - lower limit of integration = ', A
      WRITE (NOUT,99999) 'B      - upper limit of integration = ', B
      WRITE (NOUT,99998) 'EPSABS - absolute accuracy requested = ',
     +  EPSABS
      WRITE (NOUT,99998) 'EPSREL - relative accuracy requested = ',
     +  EPSREL
      WRITE (NOUT,*)
      IF (IFAIL.NE.0) WRITE (NOUT,99996) 'IFAIL = ', IFAIL
      IF (IFAIL.LE.3) THEN
         WRITE (NOUT,99997) 'RESULT - approximation to the integral = ',
     +     RESULT
         WRITE (NOUT,99998) 'ABSERR - estimate of the absolute error = '
     +     , ABSERR
         WRITE (NOUT,99996) 'KOUNT  - number of function evaluations = '
     +     , KOUNT
         WRITE (NOUT,99996) 'IW(1)  - number of subintervals used = ',
     +     IW(1)
      END IF
      STOP
*
99999 FORMAT (1X,A,F10.4)
99998 FORMAT (1X,A,e9.2)
99997 FORMAT (1X,A,F9.5)
99996 FORMAT (1X,A,I4)
      END
*
      real FUNCTION FST(X)
*     .. Scalar Arguments ..
      real              X
*     .. Scalars in Common ..
      INTEGER           KOUNT
*     .. Intrinsic Functions ..
      INTRINSIC         COS, SIN
*     .. Common blocks ..
      COMMON            /TELNUM/KOUNT
*     .. Executable Statements ..
      KOUNT = KOUNT + 1
      FST = X*(SIN(30.0e0*X))*COS(X)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D01AKF Example Program Results

 A      - lower limit of integration =     0.0000
 B      - upper limit of integration =     6.2832
 EPSABS - absolute accuracy requested =  0.00E+00
 EPSREL - relative accuracy requested =  0.10E-02

 RESULT - approximation to the integral =  -0.20967
 ABSERR - estimate of the absolute error =  0.45E-13
 KOUNT  - number of function evaluations =  427
 IW(1)  - number of subintervals used =    4
```

D01ALF – NAG Fortran Library Routine Document

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

1. Purpose

D01ALF is a general purpose integrator which calculates an approximation to the integral of a function $f(x)$ over a finite interval $[a,b]$:

$$I = \int_a^b f(x)\ dx$$

where the integrand may have local singular behaviour at a finite number of points within the integration interval.

2. Specification

```
      SUBROUTINE D01ALF (F, A, B, NPTS, POINTS, EPSABS, EPSREL, RESULT,
     1                   ABSERR, W, LW, IW, LIW, IFAIL)
      INTEGER         NPTS, LW, IW(LIW), LIW, IFAIL
      real            F, A, B, POINTS(*), EPSABS, EPSREL, RESULT,
     1                ABSERR, W(LW)
      EXTERNAL        F
```

3. Description

D01ALF is based upon the QUADPACK routine QAGP (Piessens *et al.* [3]). It is very similar to D01AJF, but allows the user to supply 'break-points', points at which the function is known to be difficult. It is an adaptive routine, using the Gauss 10-point and Kronrod 21-point rules. The algorithm described by de Doncker [1], incorporates a global acceptance criterion (as defined by Malcolm and Simpson [2]) together with the ε-algorithm (Wynn [4]) to perform extrapolation. The user-supplied 'break-points' always occur as the end-points of some sub-interval during the adaptive process. The local error estimation is described by Piessens *et al.* [3].

4. References

[1] DE DONCKER, E.
An Adaptive Extrapolation Algorithm for Automatic Integration.
Signum Newsletter, 13, 2, pp. 12-18, 1978.

[2] MALCOLM, M.A. and SIMPSON, R.B.
Local Versus Global Strategies for Adaptive Quadrature.
ACM Trans. Math. Softw., 1, pp. 129-146, 1976.

[3] PIESSENS, R., DE DONCKER-KAPENGA, E., ÜBERHUBER, C. and KAHANER, D.
QUADPACK, A Subroutine Package for Automatic Integration.
Springer-Verlag, 1983.

[4] WYNN, P.
On a Device for Computing the $e_m(S_n)$ Transformation.
Math. Tables Aids Comput., 10, pp. 91-96, 1956.

5. Parameters

1: F – ***real*** FUNCTION, supplied by the user. *External Procedure*

F must return the value of the integrand f at a given point.

Its specification is:

```
real FUNCTION F(X)
real          X
```

1: X – ***real.*** *Input*

On entry: the point at which the integrand f must be evaluated.

F must be declared as EXTERNAL in the (sub)program from which D01ALF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: A – ***real.*** *Input*

On entry: the lower limit of integration, a.

3: B – ***real.*** *Input*

On entry: the upper limit of integration, b. It is not necessary that $a < b$.

4: NPTS – INTEGER. *Input*

On entry: the number of user-supplied break-points within the integration interval.

Constraint: NPTS ≥ 0.

5: POINTS(NPTS) – ***real*** array. *Input*

On entry: the user-specified break-points.

Constraint: the break-points must all lie within the interval of integration (but may be supplied in any order).

6: EPSABS – ***real.*** *Input*

On entry: the absolute accuracy required. If EPSABS is negative, the absolute value is used. See Section 7.

7: EPSREL – ***real.*** *Input*

On entry: the relative accuracy required. If EPSREL is negative, the absolute value is used. See Section 7.

8: RESULT – ***real.*** *Input*

On entry: the approximation to the integral I.

9: ABSERR – ***real.*** *Output*

On exit: an estimate of the modulus of the absolute error, which should be an upper bound for $|I-\text{RESULT}|$.

10: W(LW) – ***real*** array. *Output*

On exit: details of the computation, as described in Section 8.

11: LW – INTEGER. *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D01ALF is called. The value of LW (together with that of LIW below) imposes a bound on the number of sub-intervals into which the interval of integration may be divided by the routine. The number of sub-intervals cannot exceed (LW−2×NPTS−4)/4. The more difficult the integrand, the larger LW should be.

Suggested value: a value in the range 800 to 2000 is adequate for most problems.

Constraint: LW ≥ 2×NPTS + 8.

12: IW(LIW) – INTEGER array. *Output*

On exit: IW(1) contains the actual number of subintervals used. The rest of the array is used as workspace.

13: LIW – INTEGER. *Input*

On entry: the dimension of the array IW as declared in the (sub)program from which D01ALF is called. The number of subintervals into which the interval of integration may be divided cannot exceed (LIW–NPTS–2)/2.

Suggested value: LIW = LW/2.

Constraint: LIW ≥ NPTS + 4.

14: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL = 1

The maximum number of subdivisions allowed with the given workspace has been reached, without the accuracy requirements being achieved. Look at the integrand in order to determine the integration difficulties. If the position of a local difficulty within the interval can be determined (e.g. a singularity of the integrand or its derivative, a peak, a discontinuity, etc.) it should be supplied to the routine as an element of the vector POINTS. If necessary, another integrator should be used, which is designed for handling the type of difficulty involved. Alternatively, consider relaxing the accuracy requirements specified by EPSABS and EPSREL, or increasing the amount of workspace.

IFAIL = 2

Roundoff error prevents the requested tolerance from being achieved. The error may be under-estimated. Consider requesting less accuracy.

IFAIL = 3

Extremely bad local integrand behaviour causes a very strong subdivision around one (or more) points of the interval. The same advice applies as in the case of IFAIL = 1.

IFAIL = 4

The requested tolerance cannot be achieved, because the extrapolation does not increase the accuracy satisfactorily; the result returned is the best which can be obtained. The same advice applies as in the case IFAIL = 1.

IFAIL = 5

The integral is probably divergent, or slowly convergent. Please note that divergence can also occur with any other non-zero value of IFAIL.

IFAIL = 6

The input is invalid: break-points are specified outside the integration range, NPTS > LIMIT or NPTS < 0. RESULT and ABSERR are set to zero.

IFAIL = 7

On entry, LW < 2×NPTS + 8,
or LIW < NPTS + 4.

7. Accuracy

The routine cannot guarantee, but in practice usually achieves, the following accuracy:

$$|I\text{–RESULT}| \le tol$$

where

$$tol = \max\{|\text{EPSABS}|,|\text{EPSREL}|\times|I|\}$$

and EPSABS and EPSREL are user-specified absolute and relative error tolerances. Moreover it returns the quantity ABSERR which, in normal circumstances, satisfies

$$|I\text{–RESULT}| \le \text{ABSERR} \le tol.$$

8. Further Comments

The time taken by the routine depends on the integrand and on the accuracy required.

If IFAIL ≠ 0 on exit, then the user may wish to examine the contents of the array W, which contains the end-points of the subintervals used by D01ALF along with the integral contributions and error estimates over these subintervals.

Specifically, for $i = 1,2,...,n$, let r_i denote the approximation to the value of the integral over the subinterval $[a_i,b_i]$ in the partition of $[a,b]$ and e_i be the corresponding absolute error estimate. Then, $\int_{a_i}^{b_i} f(x)dx \simeq r_i$ and RESULT $= \sum_{i=1}^{n} r_i$ unless D01ALF terminates while testing for divergence of the integral (see Piessens *et al.* [3] Section 3.4.3). In this case, RESULT (and ABSERR) are taken to be the values returned from the extrapolation process. The value of n is returned in IW(1), and the values a_i, b_i, e_i and r_i are stored consecutively in the array W, that is:

a_i = W(i),
b_i = W(n+i),
e_i = W(2n+i) and
r_i = W(3n+i).

9. Example

To compute

$$\int_0^1 \frac{1}{\sqrt{|x-\frac{1}{7}|}}\, dx.$$

A break-point is specified at $x = \frac{1}{7}$, at which point the integrand is infinite. (For definiteness the function FST returns the value 0.0 at this point.)

9.1. Program Text

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

```
*     D01ALF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NPTS, LW, LIW
      PARAMETER        (NPTS=1,LW=800,LIW=LW/2)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      INTEGER          KOUNT
*     .. Local Scalars ..
      real             A, ABSERR, B, EPSABS, EPSREL, RESULT
      INTEGER          IFAIL
*     .. Local Arrays ..
      real             POINTS(NPTS), W(LW)
      INTEGER          IW(LIW)
*     .. External Functions ..
      real             FST
      EXTERNAL         FST
*     .. External Subroutines ..
      EXTERNAL         D01ALF
*     .. Common blocks ..
      COMMON           /TELNUM/KOUNT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01ALF Example Program Results'
      EPSABS = 0.0e0
      EPSREL = 1.0e-03
      A = 0.0e0
      B = 1.0e0
      POINTS(1) = 1.0e0/7.0e0
      KOUNT = 0
      IFAIL = -1
*
      CALL D01ALF(FST,A,B,NPTS,POINTS,EPSABS,EPSREL,RESULT,ABSERR,W,LW,
     +            IW,LIW,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'A      - lower limit of integration = ', A
      WRITE (NOUT,99999) 'B      - upper limit of integration = ', B
      WRITE (NOUT,99998) 'EPSABS - absolute accuracy requested = ',
     +  EPSABS
      WRITE (NOUT,99998) 'EPSREL - relative accuracy requested = ',
     +  EPSREL
      WRITE (NOUT,99999) 'POINTS(1) - given break-point = ', POINTS(1)
      WRITE (NOUT,*)
      IF (IFAIL.NE.0) WRITE (NOUT,99996) 'IFAIL = ', IFAIL
      IF (IFAIL.LE.5) THEN
         WRITE (NOUT,99997)
     +     ' RESULT - approximation to the integral = ', RESULT
         WRITE (NOUT,99998)
     +     ' ABSERR - estimate of the absolute error = ', ABSERR
         WRITE (NOUT,99996)
     +     ' KOUNT  - number of function evaluations = ', KOUNT
         WRITE (NOUT,99996) 'IW(1)  - number of subintervals used = ',
     +     IW(1)
      END IF
      STOP
*
99999 FORMAT (1X,A,F10.4)
99998 FORMAT (1X,A,e9.2)
99997 FORMAT (1X,A,F9.5)
99996 FORMAT (1X,A,I4)
      END
*
```

```
      real FUNCTION FST(X)
*     .. Scalar Arguments ..
      real              X
*     .. Scalars in Common ..
      INTEGER           KOUNT
*     .. Local Scalars ..
      real              A
*     .. Intrinsic Functions ..
      INTRINSIC         ABS
*     .. Common blocks ..
      COMMON            /TELNUM/KOUNT
*     .. Executable Statements ..
      KOUNT = KOUNT + 1
      A = ABS(X-1.0e0/7.0e0)
      FST = 0.0e0
      IF (A.NE.0.0e0) FST = A**(-0.5e0)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D01ALF Example Program Results

 A      - lower limit of integration =     0.0000
 B      - upper limit of integration =     1.0000
 EPSABS - absolute accuracy requested =  0.00E+00
 EPSREL - relative accuracy requested =  0.10E-02
 POINTS(1) - given break-point =     0.1429

   RESULT - approximation to the integral =   2.60757
   ABSERR - estimate of the absolute error =  0.61E-13
   KOUNT  - number of function evaluations =  462
 IW(1)  - number of subintervals used =   12
```

D01AMF – NAG Fortran Library Routine Document

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

1. Purpose

D01AMF calculates an approximation to the integral of a function $f(x)$ over an infinite or semi-infinite interval $[a,b]$:

$$I = \int_a^b f(x)\ dx.$$

2. Specification

```
      SUBROUTINE D01AMF (F, BOUND, INF, EPSABS, EPSREL, RESULT, ABSERR,
     1                   W, LW, IW, LIW, IFAIL)
      INTEGER          INF, LW, IW(LIW), LIW, IFAIL
      real             F, BOUND, EPSABS, EPSREL, RESULT, ABSERR, W(LW)
      EXTERNAL         F
```

3. Description

D01AMF is based on the QUADPACK routine QAGI (Piessens *et al.* [3]). The entire infinite integration range is first transformed to [0,1] using one of the identities:

$$\int_{-\infty}^{a} f(x)dx = \int_0^1 f\left(a-\frac{1-t}{t}\right)\frac{1}{t^2}dt$$

$$\int_a^{\infty} f(x)dx = \int_0^1 f\left(a+\frac{1-t}{t}\right)\frac{1}{t^2}dt$$

$$\int_{-\infty}^{\infty} f(x)dx = \int_0^{\infty} (f(x)+f(-x))dx = \int_0^1 \left[f\left(\frac{1-t}{t}\right)+f\left(\frac{-1+t}{t}\right)\right]\frac{1}{t^2}dt$$

where a represents a finite integration limit. An adaptive procedure, based on the Gauss seven-point and Kronrod 15-point rules, is then employed on the transformed integral. The algorithm, described by de Doncker [1], incorporates a global acceptance criterion (as defined by Malcolm and Simpson [2]) together with the ε-algorithm (Wynn [4]) to perform extrapolation. The local error estimation is described by Piessens *et al.* [3].

4. References

[1] DE DONCKER, E.
An Adaptive Extrapolation Algorithm for Automatic Integration.
Signum Newsletter, 13, 2, pp. 12-18, 1978.

[2] MALCOLM, M.A. and SIMPSON, R.B.
Local Versus Global Strategies for Adaptive Quadrature.
ACM Trans. Math. Softw., 1, pp. 129-146, 1976.

[3] PIESSENS, R., DE DONCKER-KAPENGA, E., ÜBERHUBER, C. and KAHANER, D.
QUADPACK, A Subroutine Package for Automatic Integration.
Springer-Verlag, 1983.

[4] WYNN, P.
Device for Computing the $e_m(S_n)$ Transformation.
Math. Tables Aids Comput., 10, pp. 91-96, 1956.

5. Parameters

1: F – ***real*** FUNCTION, supplied by the user. *External Procedure*

F must return the value of the integrand f at a given point.

Its specification is:

```
real FUNCTION F(X)
real          X
```

1: X – ***real***. *Input*

On entry: the point at which the integrand f must be evaluated.

F must be declared as EXTERNAL in the (sub)program from which D01AMF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: BOUND – ***real***. *Input*

On entry: the finite limit of the integration range (if present). BOUND is not used if the interval is doubly infinite.

3: INF – INTEGER. *Input*

On entry: indicates the kind of integration range:

if INF = 1, the range is [BOUND, $+\infty$)

if INF = –1, the range is ($-\infty$, BOUND]

if INF = +2, the range is ($-\infty$, $+\infty$).

Constraint: INF = –1, 1 or 2.

4: EPSABS – ***real***. *Input*

On entry: the absolute accuracy required. If EPSABS is negative, the absolute value is used. See Section 7.

5: EPSREL – ***real***. *Input*

On entry: the relative accuracy required. If EPSREL is negative, the absolute value is used. See Section 7.

6: RESULT – ***real***. *Output*

On exit: the approximation to the integral I.

7: ABSERR – ***real***. *Output*

On exit: an estimate of the modulus of the absolute error, which should be an upper bound for $|I-\text{RESULT}|$.

8: W(LW) – ***real*** array. *Output*

On exit: details of the computation, as described in Section 8.

9: LW – INTEGER. *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D01AMF is called. The value of LW (together with that of LIW below) imposes a bound on the number of sub-intervals into which the interval of integration may be divided by the routine. The number of sub-intervals cannot exceed LW/4. The more difficult the integrand, the larger LW should be.

Suggested value: a value in the range 800 to 2000 is adequate for most problems.

Constraint: LW $\geq$ 4.

10: IW(LIW) – INTEGER array. *Output*

On exit: IW(1) contains the actual number of sub-intervals used. The rest of the array is used as workspace.

11: LIW – INTEGER. *Input*

On entry: the dimension of the array IW as declared in the (sub)program from which D01AMF is called. The number of sub-intervals into which the interval of integration may be divided cannot exceed LIW.

Suggested value: LIW = LW/4.

Constraint: LIW $\geq$ 1.

12: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq$ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL = 1

The maximum number of subdivisions allowed with the given workspace has been reached without the requested accuracy requirements being achieved. Look at the integrand in order to determine the integration difficulties. If the position of a local difficulty within the interval can be determined (e.g. a singularity of the integrand or its derivative, a peak, a discontinuity, etc.) you will probably gain from splitting up the interval at this point and calling D01AMF on the infinite subrange and an appropriate integrator on the finite subrange. Alternatively, consider relaxing the accuracy requirements specified by EPSABS and EPSREL, or increasing the amount of workspace.

IFAIL = 2

Round-off error prevents the requested tolerance from being achieved. The error may be underestimated. Consider requesting less accuracy.

IFAIL = 3

Extremely bad local integrand behaviour causes a very strong subdivision around one (or more) points of the interval. The same advice applies as in the case of IFAIL = 1.

IFAIL = 4

The requested tolerance cannot be achieved, because the extrapolation does not increase the accuracy satisfactorily; the returned result is the best which can be obtained. The same advice applies as in the case of IFAIL = 1.

IFAIL = 5

The integral is probably divergent, or slowly convergent. It must be noted that divergence can also occur with any other non-zero value of IFAIL.

IFAIL = 6

On entry, LW < 4,
or LIW < 1,
or INF ≠ –1, 1 or 2.

7. Accuracy

The routine cannot guarantee, but in practice usually achieves, the following accuracy:

$$|I\text{–RESULT}| \leq tol,$$

where

$$tol = \max\{|\text{EPSABS}|,|\text{EPSREL}|\times|I|\}$$

and EPSABS and EPSREL are user-specified absolute and relative error tolerances. Moreover it returns the quantity ABSERR, which, in normal circumstances, satisfies

$$|I\text{–RESULT}| \leq \text{ABSERR} \leq tol.$$

8. Further Comments

The time taken by the routine depends on the integrand and the accuracy required.

If IFAIL ≠ 0 on exit, then the user may wish to examine the contents of the array W, which contains the end-points of the sub-intervals used by D01AMF along with the integral contributions and error estimates over these sub-intervals.

Specifically, for $i = 1,2,...,n$, let r_i denote the approximation to the value of the integral over the sub-interval $[a_i,b_i]$ in the partition of $[a,b]$ and e_i be the corresponding absolute error estimate. Then, $\int_{a_i}^{b_i} f(x)dx \simeq r_i$ and $\text{RESULT} = \sum_{i=1}^{n} r_i$ unless D01AMF terminates while testing for divergence of the integral (see Piessens *et al.* [3] Section 3.4.3). In this case, RESULT (and ABSERR) are taken to be the values returned from the extrapolation process. The value of n is returned in IW(1), and the values a_i, b_i, e_i and r_i are stored consecutively in the array W, that is:

a_i = W(i),
b_i = W($n+i$),
e_i = W($2n+i$) and
r_i = W($3n+i$).

Note: that this information applies to the integral transformed to (0,1) as described in Section 3, not to the original integral.

9. Example

To compute

$$\int_0^\infty \frac{1}{(x+1)\sqrt{x}}dx.$$

The exact answer is π.

9.1. Program Text

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

```
*     D01AMF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          LW, LIW
      PARAMETER        (LW=800,LIW=LW/4)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      INTEGER          KOUNT
*     .. Local Scalars ..
      real             A, ABSERR, EPSABS, EPSREL, RESULT
      INTEGER          IFAIL, INF
*     .. Local Arrays ..
      real             W(LW)
      INTEGER          IW(LIW)
*     .. External Functions ..
      real             FST
      EXTERNAL         FST
*     .. External Subroutines ..
      EXTERNAL         D01AMF
*     .. Common blocks ..
      COMMON           /TELNUM/KOUNT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01AMF Example Program Results'
      EPSABS = 0.0e0
      EPSREL = 1.0e-04
      A = 0.0e0
      INF = 1
      KOUNT = 0
      IFAIL = -1
*
      CALL D01AMF(FST,A,INF,EPSABS,EPSREL,RESULT,ABSERR,W,LW,IW,LIW,
     +            IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'A      - lower limit of integration = ', A
      WRITE (NOUT,*) 'B      - upper limit of integration = infinity'
      WRITE (NOUT,99998) 'EPSABS - absolute accuracy requested = ',
     +  EPSABS
      WRITE (NOUT,99998) 'EPSREL - relative accuracy requested = ',
     +  EPSREL
      WRITE (NOUT,*)
      IF (IFAIL.NE.0) WRITE (NOUT,99996) 'IFAIL = ', IFAIL
      IF (IFAIL.LE.5) THEN
         WRITE (NOUT,99997) 'RESULT - approximation to the integral = ',
     +     RESULT
         WRITE (NOUT,99998) 'ABSERR - estimate of the absolute error = '
     +     , ABSERR
         WRITE (NOUT,99996) 'KOUNT  - number of function evaluations = '
     +     , KOUNT
         WRITE (NOUT,99996) 'IW(1)  - number of subintervals used = ',
     +     IW(1)
      END IF
      STOP
*
99999 FORMAT (1X,A,F10.4)
99998 FORMAT (1X,A,e9.2)
99997 FORMAT (1X,A,F9.5)
99996 FORMAT (1X,A,I4)
      END
*
```

```
      real FUNCTION FST(X)
*     .. Scalar Arguments ..
      real                X
*     .. Scalars in Common ..
      INTEGER             KOUNT
*     .. Intrinsic Functions ..
      INTRINSIC           SQRT
*     .. Common blocks ..
      COMMON              /TELNUM/KOUNT
*     .. Executable Statements ..
      KOUNT = KOUNT + 1
      FST = 1.0e0/((X+1.0e0)*SQRT(X))
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D01AMF Example Program Results

A      - lower limit of integration =     0.0000
B      - upper limit of integration = infinity
EPSABS - absolute accuracy requested =  0.00E+00
EPSREL - relative accuracy requested =  0.10E-03

RESULT - approximation to the integral =   3.14159
ABSERR - estimate of the absolute error =  0.27E-04
KOUNT  - number of function evaluations =  285
IW(1)  - number of subintervals used =   10
```

D01ANF – NAG Fortran Library Routine Document

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

1. Purpose

D01ANF calculates an approximation to the sine or the cosine transform of a function g over $[a,b]$:

$$I = \int_a^b g(x)\sin(\omega x)\,dx \quad \text{or} \quad I = \int_a^b g(x)\cos(\omega x)\,dx$$

(for a user-specified value of ω).

2. Specification

```
SUBROUTINE D01ANF (G, A, B, OMEGA, KEY, EPSABS, EPSREL, RESULT,
1                  ABSERR, W, LW, IW, LIW, IFAIL)
INTEGER         KEY, LW, IW(LIW), LIW, IFAIL
real            G, A, B, OMEGA, EPSABS, EPSREL, RESULT, ABSERR,
1               W(LW)
EXTERNAL        G
```

3. Description

D01ANF is based upon the QUADPACK routine QFOUR (Piessens *et al.* [3]). It is an adaptive routine, designed to integrate a function of the form $g(x)w(x)$, where $w(x)$ is either $\sin(\omega x)$ or $\cos(\omega x)$. If a subinterval has length

$$L = |b-a|2^{-l}$$

then the integration over this subinterval is performed by means of a modified Clenshaw-Curtis procedure (Piessens and Branders [2]) if $L\omega > 4$ and $l \leq 20$. In this case a Chebyshev-series approximation of degree 24 is used to approximate $g(x)$, while an error estimate is computed from this approximation together with that obtained using Chebyshev-series of degree 12. If the above conditions do not hold then Gauss 7-point and Kronrod 15-point rules are used. The algorithm, described in [3], incorporates a global acceptance criterion (as defined in Malcolm and Simpson [1]) together with the ε-algorithm Wynn [4] to perform extrapolation. The local error estimation is described in [3].

4. References

[1] MALCOLM, M.A. and SIMPSON, R.B.
Local Versus Global Strategies for Adaptive Quadrature.
A.C.M. Trans. Math. Software, 1, pp. 129-146, 1976.

[2] PIESSENS, R. and BRANDERS, M.
Algorithm 002. Computation of Oscillating Integrals.
J. Comp. Appl. Math., 1, pp. 153-164, 1975.

[3] PIESSENS, R., DE DONCKER-KAPENGA, E., ÜBERHUBER, C. and KAHANER, D.
QUADPACK, A Subroutine Package for Automatic Integration.
Springer-Verlag, 1983.

[4] WYNN, P.
On a Device for Computing the $e_m(S_n)$ Transformation.
Math. Tables Aids Comp., 10, pp. 91-96, 1956.

5. Parameters

1: G – ***real*** FUNCTION, supplied by the user. *External Procedure*

G must return the value of the function g at a given point.

Its specification is:

```
real FUNCTION G(X)
real          X
```

1: X – ***real***. *Input*

On entry: the point at which the function g must be evaluated.

G must be declared as EXTERNAL in the (sub)program from which D01ANF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: A – ***real***. *Input*

On entry: the lower limit of integration, a.

3: B – ***real***. *Input*

On entry: the upper limit of integration, b. It is not necessary that $a < b$.

4: OMEGA – ***real***. *Input*

On entry: the parameter ω in the weight function of the transform.

5: KEY – INTEGER. *Input*

On entry: indicates which integral is to be computed:

if KEY $= 1, w(x) = \cos(\omega x)$;

if KEY $= 2, w(x) = \sin(\omega x)$.

Constraint: KEY = 1 or 2.

6: EPSABS – ***real***. *Input*

On entry: the absolute accuracy required. If EPSABS is negative, the absolute value is used. See Section 7.

7: EPSREL – ***real***. *Input*

On entry: the relative accuracy required. If EPSREL is negative, the absolute value is used. See Section 7.

8: RESULT – ***real***. *Output*

On exit: the approximation to the integral I.

9: ABSERR – ***real***. *Output*

On exit: an estimate of the modulus of the absolute error, which should be an upper bound for $|I-\text{RESULT}|$.

10: W(LW) – ***real*** array. *Output*

On exit: details of the computation, as described in Section 8.

11: LW – INTEGER. *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D01ANF is called. The value of LW (together with that of LIW below) imposes a bound on the number of subintervals into which the interval of integration may be divided by the routine. The number of subintervals cannot exceed LW/4. The more difficult the integrand, the larger LW should be.

Suggested value: a value in the range 800 to 2000 is adequate for most problems.

Constraint: LW ≥ 4.

12: IW(LIW) – INTEGER array. *Output*

On exit: IW(1) contains the actual number of subintervals used. The rest of the array is used as workspace.

13: LIW – INTEGER. *Input*

On entry: the dimension of the array IW as declared in the (sub)program from which D01ANF is called. The number of subintervals into which the interval of integration may be divided cannot exceed LIW/2.

Suggested value: LIW = LW/2.

Constraint: LIW ≥ 2.

14: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL = 1

The maximum number of subdivisions allowed with the given workspace has been reached without the accuracy requested being achieved. Look at the integrand in order to determine the integration difficulties. If the position of a local difficulty within the interval can be determined (e.g. a singularity of the integrand or its derivative, a peak, a discontinuity, etc.) you will probably gain from splitting up the interval at this point and calling the integrator on the subranges. If necessary, another integrator, which is designed for handling the type of difficulty involved, must be used. Alternatively consider relaxing the accuracy requirements specified by EPSABS and EPSREL, or increasing amount of workspace.

IFAIL = 2

Roundoff error prevents the requested tolerance from being achieved. The error may be underestimated. Consider requesting less accuracy.

IFAIL = 3

Extremely bad local behaviour of $g(x)$ causes a very strong subdivision around one (or more) points of the interval. The same advice applies as in the case of IFAIL = 1.

IFAIL = 4

The requested tolerance cannot be achieved because the extrapolation does not increase the accuracy satisfactorily; the returned result is the best which can be obtained. The same advice applies as in the case of IFAIL = 1.

IFAIL = 5

The integral is probably divergent, or slowly convergent. It must be noted that divergence can occur with any non-zero value of IFAIL.

IFAIL = 6

On entry, KEY < 1,
or KEY > 2.

IFAIL = 7

On entry, LW < 4,
or LIW < 2.

7. Accuracy

The routine cannot guarantee, but in practice usually achieves, the following accuracy:

$$|I\text{–RESULT}| \leq tol,$$

where

$$tol = \max\{|\text{EPSABS}|,|\text{EPSREL}|\times|I|\},$$

and EPSABS and EPSREL are user-specified absolute and relative tolerances. Moreover it returns the quantity ABSERR, which, in normal circumstances, satisfies

$$|I\text{–RESULT}| \leq \text{ABSERR} \leq tol.$$

8. Further Comments

The time taken by the routine depends on the integrand and on the accuracy required.

If IFAIL ≠ 0 on exit, then the user may wish to examine the contents of the array W, which contains the end-points of the subintervals used by D01ANF along with the integral contributions and error estimates over these subintervals.

Specifically, for $i = 1,2,...,n$, let r_i denote the approximation to the value of the integral over the subinterval $[a_i,b_i]$ in the partition of $[a,b]$ and e_i be the corresponding absolute error estimate. Then, $\int_{a_i}^{b_i} g(x)w(x)\ dx \simeq r_i$ and RESULT $= \sum_{i=1}^{n} r_i$ unless D01ANF terminates while testing for divergence of the integral (see Piessens *et al.* [3] Section 3.4.3). In this case, RESULT (and ABSERR) are taken to be the values returned from the extrapolation process. The value of n is returned in IW(1), and the values a_i, b_i, e_i and r_i are stored consecutively in the array W, that is:

a_i = W(i),
b_i = W(n+i),
e_i = W($2n$+i) and
r_i = W($3n$+i).

9. Example

To compute

$$\int_0^1 \ln x\ \sin(10\pi x)\ dx.$$

9.1. Program Text

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

```
*     D01ANF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          LW, LIW
      PARAMETER        (LW=800,LIW=LW/2)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
```

```
*     .. Scalars in Common ..
      INTEGER          KOUNT
*     .. Local Scalars ..
      real             A, ABSERR, B, EPSABS, EPSREL, OMEGA, PI, RESULT
      INTEGER          IFAIL, KEY
*     .. Local Arrays ..
      real             W(LW)
      INTEGER          IW(LIW)
*     .. External Functions ..
      real             FST, X01AAF
      EXTERNAL         FST, X01AAF
*     .. External Subroutines ..
      EXTERNAL         D01ANF
*     .. Common blocks ..
      COMMON           /TELNUM/KOUNT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01ANF Example Program Results'
      EPSREL = 1.0e-04
      EPSABS = 0.0e+00
      A = 0.0e0
      B = 1.0e0
      OMEGA = 10.0e0*X01AAF(PI)
      KEY = 2
      KOUNT = 0
      IFAIL = -1
*
      CALL D01ANF(FST,A,B,OMEGA,KEY,EPSABS,EPSREL,RESULT,ABSERR,W,LW,IW,
     +            LIW,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'A      - lower limit of integration = ', A
      WRITE (NOUT,99999) 'B      - upper limit of integration = ', B
      WRITE (NOUT,99998) 'EPSABS - absolute accuracy requested = ',
     + EPSABS
      WRITE (NOUT,99998) 'EPSREL - relative accuracy requested = ',
     + EPSREL
      WRITE (NOUT,*)
      IF (IFAIL.NE.0) WRITE (NOUT,99996) 'IFAIL = ', IFAIL
      IF (IFAIL.LE.5) THEN
         WRITE (NOUT,99997) 'RESULT - approximation to the integral = ',
     +     RESULT
         WRITE (NOUT,99998) 'ABSERR - estimate of the absolute error = '
     +     , ABSERR
         WRITE (NOUT,99996) 'KOUNT  - number of function evaluations = '
     +     , KOUNT
         WRITE (NOUT,99996) 'IW(1)  - number of subintervals used = ',
     +     IW(1)
      END IF
      STOP
*
99999 FORMAT (1X,A,F10.4)
99998 FORMAT (1X,A,e9.2)
99997 FORMAT (1X,A,F9.5)
99996 FORMAT (1X,A,I4)
      END
*
      real  FUNCTION FST(X)
*     .. Scalar Arguments ..
      real             X
*     .. Scalars in Common ..
      INTEGER          KOUNT
*     .. Intrinsic Functions ..
      INTRINSIC        LOG
*     .. Common blocks ..
      COMMON           /TELNUM/KOUNT
```

```
*     .. Executable Statements ..
      KOUNT = KOUNT + 1
      FST = 0.0e0
      IF (X.GT.0.0e0) FST = LOG(X)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D01ANF Example Program Results

 A      - lower limit of integration =     0.0000
 B      - upper limit of integration =     1.0000
 EPSABS - absolute accuracy requested =  0.00E+00
 EPSREL - relative accuracy requested =  0.10E-03

 RESULT - approximation to the integral =  -0.12814
 ABSERR - estimate of the absolute error =  0.36E-05
 KOUNT  - number of function evaluations =  275
 IW(1)  - number of subintervals used =    8
```

D01APF – NAG Fortran Library Routine Document

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

1. Purpose

D01APF is an adaptive integrator which calculates an approximation to the integral of a function $g(x)w(x)$ over a finite interval $[a,b]$:

$$I = \int_a^b g(x)w(x)dx$$

where the weight function w has end-point singularities of algebraico-logarithmic type.

2. Specification

```
      SUBROUTINE D01APF (G, A, B, ALFA, BETA, KEY, EPSABS, EPSREL, RESULT,
     1                   ABSERR, W, LW, IW, LIW, IFAIL)
      INTEGER            KEY, LW, IW(LIW), LIW, IFAIL
      real               G, A, B, ALFA, BETA, EPSABS, EPSREL, RESULT,
     1                   ABSERR, W(LW)
      EXTERNAL           G
```

3. Description

D01APF is based upon the QUADPACK routine QAWSE (Piessens *et al.* [3]) and integrates a function of the form $g(x)w(x)$, where the weight function $w(x)$ may have algebraico-logarithmic singularities at the end-points a and/or b. The strategy is a modification of that in D01AKF. We start by bisecting the original interval and applying modified Clenshaw-Curtis integration of orders 12 and 24 to both halves. Clenshaw-Curtis integration is then used on all subintervals which have a or b as one of their end-points (Piessens *et al.* [2]). On the other subintervals Gauss-Kronrod (7-15 point) integration is carried out.

A 'global' acceptance criterion (as defined by Malcolm and Simpson [1]) is used. The local error estimation control is described by Piessens *et al.* [3].

4. References

[1] MALCOLM, M.A. and SIMPSON, R.B.
Local Versus Global Strategies for Adaptive Quadrature.
A.C.M. Trans. Math. Software, 1, pp. 129-146, 1976.

[2] PIESSENS, R., MERTENS, I. and BRANDERS, M.
Integration of Functions having Endpoint Singularities.
Angewandte Informatik, 16, pp. 65-68, 1974.

[3] PIESSENS, R., DE DONCKER-KAPENGA, E. ÜBERHUBER, C and KAHANER, D.
QUADPACK, A Subroutine Package for Automatic Integration.
Springer-Verlag, 1983.

5. Parameters

1: G – ***real*** FUNCTION, supplied by the user. *External Procedure*

G must return the value of the function g at a given point X.

Its specification is:

```
real FUNCTION G(X)
real          X
```

1: X – ***real.*** *Input*

On entry: the point at which the function g must be evaluated.

G must be declared as EXTERNAL in the (sub)program from which D01APF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: A – ***real.*** *Input*

On entry: the lower limit of integration, a.

3: B – ***real.*** *Input*

On entry: the upper limit of integration, b.

Constraint: B > A.

4: ALFA – ***real.*** *Input*

On entry: the parameter α in the weight function.

Constraint: ALFA > −1.

5: BETA – ***real.*** *Input*

On entry: the parameter β in the weight function.

Constraint: BETA > −1.

6: KEY – INTEGER. *Input*

On entry: indicates which weight function is to be used:

if KEY = 1, $w(x) = (x-a)^{\alpha}(b-x)^{\beta}$

if KEY = 2, $w(x) = (x-a)^{\alpha}(b-x)^{\beta} \ln(x-a)$

if KEY = 3, $w(x) = (x-a)^{\alpha}(b-x)^{\beta} \ln(b-x)$

if KEY = 4, $w(x) = (x-a)^{\alpha}(b-x)^{\beta} \ln(x-a) \ln(b-x)$

Constraint: KEY = 1, 2, 3 or 4

7: EPSABS – ***real.*** *Input*

On entry: the absolute accuracy required. If EPSABS is negative, the absolute value is used. See Section 7.

8: EPSREL – ***real.*** *Input*

On entry: the relative accuracy required. If EPSREL is negative, the absolute value is used. See Section 7.

9: RESULT – ***real.*** *Output*

On exit: the approximation to the integral I.

10: ABSERR – ***real.*** *Output*

On exit: an estimate of the modulus of the absolute error, which should be an upper bound for $|I-\text{RESULT}|$.

11: W(LW) – ***real*** array. *Output*

On exit: details of the computation, as described in Section 8.

12: LW – INTEGER. *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D01APF is called. The value of LW (together with that of LIW below) imposes a bound on the number of subintervals into which the interval of integration may be divided by the routine. The number of subintervals cannot exceed LW/4. The more difficult the integrand, the larger LW should be.

Suggested value: LW = 800 to 2000 is adequate for most problems.

Constraint: LW ≥ 8.

13: IW(LIW) – INTEGER array. *Output*

On exit: IW(1) contains the actual number of subintervals used. The rest of the array is used as workspace.

14: LIW – INTEGER. *Input*

On entry: the dimension of the array IW as declared in the (sub)program from which D01APF is called. The number of subintervals into which the interval of integration may be divided cannot exceed LIW.

Suggested value: LIW = LW/4.

Constraint: LIW ≥ 2.

15: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL = 1

The maximum number of subdivisions allowed with the given workspace has been reached without the accuracy requirements being achieved. Look at the integrand in order to determine the integration difficulties. If the position of a discontinuity or a singularity of algebraico-logarithmic type within the interval can be determined, the interval must be split up at this point and the integrator called on the subranges. If necessary, another integrator, which is designed for handling the difficulty involved, must be used. Alternatively consider relaxing the accuracy requirements specified by EPSABS and EPSREL, or increasing the amount of workspace.

IFAIL = 2

Roundoff error prevents the requested tolerance from being achieved. Consider requesting less accuracy.

IFAIL = 3

Extremely bad local integrand behaviour causes a very strong subdivision around one (or more) points of the interval. The same advice applies as in the case of IFAIL = 1.

IFAIL = 4

On entry, B ≤ A,
or ALFA ≤ −1,
or BETA ≤ −1,
or KEY < 1,
or KEY > 4.

IFAIL = 5

On entry, LW < 8,
or LIW < 2.

7. Accuracy

The routine cannot guarantee, but in practice usually achieves, the following accuracy:

$$|I-\text{RESULT}| \leq tol,$$

where

$$tol = \max\{|\text{EPSABS}|,|\text{EPSREL}|\times|I|\},$$

and EPSABS and EPSREL are user-specified absolute and relative error tolerances.

Moreover it returns the quantity ABSERR which, in normal circumstances, satisfies:

$$|I-\text{RESULT}| \leq \text{ABSERR} \leq tol.$$

8. Further Comments

The time taken by the routine depends on the integrand and on the accuracy required.

If IFAIL ≠ 0 on exit, then the user may wish to examine the contents of the array W, which contains the end-points of the subintervals used by D01APF along with the integral contributions and error estimates over these subintervals.

Specifically, for $i = 1,2,...,n$, let r_i denote the approximation to the value of the integral over the subinterval $[a_i,b_i]$ in the partition of $[a,b]$ and e_i be the corresponding absolute error estimate. Then, $\int_{a_i}^{b_i} f(x)w(x)dx \simeq r_i$ and RESULT $= \sum_{i=1}^{n} r_i$. The value of n is returned in IW(1), and the values a_i, b_i, e_i and r_i are stored consecutively in the array W, that is:

$$a_i = \text{W}(i),$$
$$b_i = \text{W}(n+i),$$
$$e_i = \text{W}(2n+i),$$
$$r_i = \text{W}(3n+i).$$

9. Example

To compute:

$$\int_0^1 \ln x \cos(10\pi x)dx \quad \text{and} \quad \int_0^1 \frac{\sin(10x)}{\sqrt{x(1-x)}}.$$

9.1. Program Text

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

```
*     D01APF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          LW, LIW
      PARAMETER        (LW=800,LIW=LW/4)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      INTEGER          KOUNT, NOF
*     .. Local Scalars ..
      real             A, ABSERR, B, EPSABS, EPSREL, RESULT
      INTEGER          IFAIL
*     .. Local Arrays ..
      real             ALFA(2), BETA(2), W(LW)
      INTEGER          INTEGR(2), IW(LIW)
*     .. External Functions ..
      real             FST
      EXTERNAL         FST
*     .. External Subroutines ..
      EXTERNAL         D01APF
*     .. Common blocks ..
      COMMON           /TELNUM/KOUNT, NOF
*     .. Data statements ..
      DATA             ALFA/0.0e0, -0.5e0/
      DATA             BETA/0.0e0, -0.5e0/
      DATA             INTEGR/2, 1/
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01APF Example Program Results'
      EPSABS = 0.0e0
      EPSREL = 1.0e-04
      A = 0.0e0
      B = 1.0e0
      DO 20 NOF = 1, 2
         KOUNT = 0
         IFAIL = -1
*
         CALL D01APF(FST,A,B,ALFA(NOF),BETA(NOF),INTEGR(NOF),EPSABS,
     +               EPSREL,RESULT,ABSERR,W,LW,IW,LIW,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'A      - lower limit of integration = ', A
         WRITE (NOUT,99999) 'B      - upper limit of integration = ', B
         WRITE (NOUT,99998) 'EPSABS - absolute accuracy requested = ',
     +     EPSABS
         WRITE (NOUT,99998) 'EPSREL - relative accuracy requested = ',
     +     EPSREL
         WRITE (NOUT,*)
         WRITE (NOUT,99998)
     +     'ALFA   - parameter in the weight function = ', ALFA(NOF)
         WRITE (NOUT,99998)
     +     'BETA   - parameter in the weight function = ', BETA(NOF)
         WRITE (NOUT,99997)
     +     'INTEGR - denotes which weight function is to be used = ',
     +     INTEGR(NOF)
         WRITE (NOUT,*)
         IF (IFAIL.NE.0) WRITE (NOUT,99997) 'IFAIL = ', IFAIL
         IF (IFAIL.LE.3) THEN
            WRITE (NOUT,99996)
     +        'RESULT - approximation to the integral = ', RESULT
            WRITE (NOUT,99998)
     +        'ABSERR - estimate of the absolute error = ', ABSERR
```

```
            WRITE (NOUT,99997)
     +        'KOUNT  - number of function evaluations = ', KOUNT
            WRITE (NOUT,99997) 'IW(1)  - number of subintervals used = '
     +        , IW(1)
         END IF
   20 CONTINUE
      STOP
*
99999 FORMAT (1X,A,F10.4)
99998 FORMAT (1X,A,e9.2)
99997 FORMAT (1X,A,I4)
99996 FORMAT (1X,A,F9.5)
      END
*
      real FUNCTION FST(X)
*     .. Scalar Arguments ..
      real              X
*     .. Scalars in Common ..
      INTEGER           KOUNT, NOF
*     .. Local Scalars ..
      real              A, OMEGA, PI
*     .. External Functions ..
      real              X01AAF
      EXTERNAL          X01AAF
*     .. Intrinsic Functions ..
      INTRINSIC         COS, SIN
*     .. Common blocks ..
      COMMON            /TELNUM/KOUNT, NOF
*     .. Executable Statements ..
      PI = X01AAF(PI)
      KOUNT = KOUNT + 1
      IF (NOF.EQ.1) THEN
         A = 10.0e0*PI
         FST = COS(A*X)
      ELSE
         OMEGA = 10.0e0
         FST = SIN(OMEGA*X)
      END IF
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D01APF Example Program Results

A      - lower limit of integration =     0.0000
B      - upper limit of integration =     1.0000
EPSABS - absolute accuracy requested =  0.00E+00
EPSREL - relative accuracy requested =  0.10E-03

ALFA   - parameter in the weight function =  0.00E+00
BETA   - parameter in the weight function =  0.00E+00
INTEGR - denotes which weight function is to be used =    2

RESULT - approximation to the integral =  -0.04899
ABSERR - estimate of the absolute error =  0.11E-06
KOUNT  - number of function evaluations =  110
IW(1)  - number of subintervals used =    4
```

```
A      - lower limit of integration =     0.0000
B      - upper limit of integration =     1.0000
EPSABS - absolute accuracy requested =  0.00E+00
EPSREL - relative accuracy requested =  0.10E-03

ALFA   - parameter in the weight function = -0.50E+00
BETA   - parameter in the weight function = -0.50E+00
INTEGR - denotes which weight function is to be used =    1

RESULT - approximation to the integral =   0.53502
ABSERR - estimate of the absolute error =  0.19E-11
KOUNT  - number of function evaluations =   50
IW(1)  - number of subintervals used =    2
```

D01AQF – NAG Fortran Library Routine Document

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

1. Purpose

D01AQF calculates an approximation to the Hilbert transform of a function $g(x)$ over $[a,b]$:

$$I = \int_a^b \frac{g(x)}{x-c} dx$$

for user-specified values of a, b and c.

2. Specification

```
      SUBROUTINE D01AQF (G, A, B, C, EPSABS, EPSREL, RESULT, ABSERR, W,
     1                   LW, IW, LIW, IFAIL)
      INTEGER            LW, IW(LIW), LIW, IFAIL
      real               G, A, B, C, EPSABS, EPSREL, RESULT, ABSERR,
     1                   W(LW)
      EXTERNAL           G
```

3. Description

D01AQF is based upon the QUADPACK routine QAWC (Piessens *et al.* [3]) and integrates a function of the form $g(x)w(x)$, where the weight function

$$w(x) = \frac{1}{x-c}$$

is that of the Hilbert transform. (If $a < c < b$ the integral has to be interpreted in the sense of a Cauchy principal value.) It is an adaptive routine which employs a 'global' acceptance criterion (as defined by Malcolm and Simpson [1]). Special care is taken to ensure that c is never the end-point of a subinterval (Piessens *et al.* [2]). On each subinterval (c_1,c_2) modified Clenshaw-Curtis integration of orders 12 and 24 is performed if $c_1 - d \le c \le c_2 + d$ where $d = (c_2-c_1)/20$. Otherwise the Gauss seven-point and Kronrod 15-point rules are used. The local error estimation is described by Piessens *et al.* [3].

4. References

[1] MALCOLM, M.A. and SIMPSON, R.B.
Local Versus Global Strategies for Adaptive Quadrature.
A.C.M. Trans. Math. Software, 1, pp. 129-146, 1976.

[2] PIESSENS, R., VAN ROY-BRANDERS, M. and MERTENS, I.
The Automatic Evaluation of Cauchy Principal Value Integrals.
Angewandte Informatik, 18, pp. 31-35, 1976.

[3] PIESSENS, R., DE DONCKER-KAPENGA, E., ÜBERHUBER, C. and KAHANER, D.
QUADPACK, A Subroutine Package for Automatic Integration.
Springer-Verlag, 1983.

5. Parameters

1: G – ***real*** FUNCTION, supplied by the user. *External Procedure*

G must return the value of the function g at a given point.

Its specification is:

```
real FUNCTION G(X)
real          X
```

1: X – ***real.*** *Input*

On entry: the point at which the function g must be evaluated.

G must be declared as EXTERNAL in the (sub)program from which D01AQF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: A – ***real.*** *Input*

On entry: the lower limit of integration, a.

3: B – ***real.*** *Input*

On entry: the upper limit of integration, b. It is not necessary that $a < b$.

4: C – ***real.*** *Input*

On entry: the parameter c in the weight function.

Constraint: C must not equal A or B.

5: EPSABS – ***real.*** *Input*

On entry: the absolute accuracy required. If EPSABS is negative, the absolute value is used. See Section 7.

6: EPSREL – ***real.*** *Input*

On entry: the relative accuracy required. If EPSREL is negative, the absolute value is used. See Section 7.

7: RESULT – ***real.*** *Output*

On exit: the approximation to the integral I.

8: ABSERR – ***real.*** *Output*

On exit: an estimate of the modulus of the absolute error, which should be an upper bound for $|I-\text{RESULT}|$.

9: W(LW) – ***real*** array. *Output*

On exit: details of the computation, as described in Section 8.

10: LW – INTEGER. *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D01AQF is called. The value of LW (together with that of LIW below) imposes a bound on the number of subintervals into which the interval of integration may be divided by the routine. The number of subintervals cannot exceed LW/4. The more difficult the integrand, the larger LW should be.

Suggested value: LW = 800 to 2000 is adequate for most problems.

Constraint: LW ≥ 4.

11: IW(LIW) – INTEGER array. *Output*

On exit: IW(1) contains the actual number of subintervals used. The rest of the array is used as workspace.

12: LIW – INTEGER. *Input*

On entry: the dimension of the array IW as declared in the (sub)program from which D01AQF is called. The number of subintervals into which the interval of integration may be divided cannot exceed LIW.

Suggested value: LIW = LW/4.

Constraint: LIW $\geq$ 1.

13: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq$ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL = 1

The maximum number of subdivisions allowed with the given workspace has been reached without the accuracy requirements being achieved. Look at the integrand in order to determine the integration difficulties. Another integrator which is designed for handling the type of difficulty involved, must be used. Alternatively consider relaxing the accuracy requirements specified by EPSABS and EPSREL, or increasing the workspace.

IFAIL = 2

Roundoff error prevents the requested tolerance from being achieved. Consider requesting less accuracy.

IFAIL = 3

Extremely bad local behaviour of $g(x)$ causes a very strong subdivision around one (or more) points of the interval. The same advice applies as in the case of IFAIL = 1.

IFAIL = 4

On entry, C = A or C = B.

IFAIL = 5

On entry, LW < 4,
or LIW < 1.

7. Accuracy

The routine cannot guarantee, but in practice usually achieves, the following accuracy:

$$|I\text{–RESULT}| \leq tol,$$

where

$$tol = \max\{|\text{EPSABS}|,|\text{EPSREL}|\times|I|\}$$

and EPSABS and EPSREL are user-specified absolute and relative error tolerances. Moreover it returns the quantity ABSERR which, in normal circumstances satisfies:

$$|I\text{–RESULT}| \leq \text{ABSERR} \leq tol.$$

8. Further Comments

The time taken by the routine depends on the integrand and on the accuracy required.

If IFAIL $\neq$ 0 on exit, then the user may wish to examine the contents of the array W, which contains the end-points of the subintervals used by D01AQF along with the integral contributions and error estimates over these subintervals.

Specifically, for $i = 1,2,...,n$, let r_i denote the approximation to the value of the integral over the subinterval $[a_i,b_i]$ in the partition of $[a,b]$ and e_i be the corresponding absolute error estimate. Then, $\int_{a_i}^{b_i} g(x)w(x)dx \simeq r_i$ and RESULT $= \sum_{i=1}^{n} r_i$. The value of n is returned in IW(1), and the values a_i, b_i, e_i and r_i are stored consecutively in the array W, that is:

a_i = W(i),
b_i = W(n+i),
e_i = W(2n+i) and
r_i = W(3n+i).

9. Example

To compute the Cauchy principal value of

$$\int_{-1}^{1} \frac{dx}{(x^2+0.01^2)(x-\frac{1}{2})}.$$

9.1. Program Text

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

```
*     D01AQF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          LW, LIW
      PARAMETER        (LW=800,LIW=LW/4)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      INTEGER          KOUNT
*     .. Local Scalars ..
      real             A, ABSERR, B, C, EPSABS, EPSREL, RESULT
      INTEGER          IFAIL
*     .. Local Arrays ..
      real             W(LW)
      INTEGER          IW(LIW)
*     .. External Functions ..
      real             FST
      EXTERNAL         FST
*     .. External Subroutines ..
      EXTERNAL         D01AQF
*     .. Common blocks ..
      COMMON           /TELNUM/KOUNT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01AQF Example Program Results'
      EPSABS = 0.0e0
      EPSREL = 1.0e-04
      A = -1.0e0
      B = 1.0e0
      C = 0.5e0
      KOUNT = 0
      IFAIL = -1
*
```

```
      CALL D01AQF(FST,A,B,C,EPSABS,EPSREL,RESULT,ABSERR,W,LW,IW,LIW,
     +            IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'A      - lower limit of integration = ', A
      WRITE (NOUT,99999) 'B      - upper limit of integration = ', B
      WRITE (NOUT,99998) 'EPSABS - absolute accuracy requested = ',
     +  EPSABS
      WRITE (NOUT,99998) 'EPSREL - relative accuracy requested = ',
     +  EPSREL
      WRITE (NOUT,99998) 'C      - parameter in the weight function = ',
     +  C
      WRITE (NOUT,*)
      IF (IFAIL.NE.0) WRITE (NOUT,99996) 'IFAIL = ', IFAIL
      IF (IFAIL.LE.3) THEN
         WRITE (NOUT,99997) 'RESULT - approximation to the integral = ',
     +     RESULT
         WRITE (NOUT,99998) 'ABSERR - estimate of the absolute error = '
     +     , ABSERR
         WRITE (NOUT,99996) 'KOUNT  - number of function evaluations = '
     +     , KOUNT
         WRITE (NOUT,99996) 'IW(1)  - number of subintervals used = ',
     +     IW(1)
      END IF
      STOP
*
99999 FORMAT (1X,A,F10.4)
99998 FORMAT (1X,A,e9.2)
99997 FORMAT (1X,A,F9.2)
99996 FORMAT (1X,A,I4)
      END
*
      real FUNCTION FST(X)
*     .. Scalar Arguments ..
      real              X
*     .. Scalars in Common ..
      INTEGER           KOUNT
*     .. Local Scalars ..
      real              AA
*     .. Common blocks ..
      COMMON            /TELNUM/KOUNT
*     .. Executable Statements ..
      KOUNT = KOUNT + 1
      AA = 0.01e0
      FST = 1.0e0/(X**2+AA**2)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D01AQF Example Program Results

A      - lower limit of integration =    -1.0000
B      - upper limit of integration =     1.0000
EPSABS - absolute accuracy requested =  0.00E+00
EPSREL - relative accuracy requested =  0.10E-03
C      - parameter in the weight function =  0.50E+00

RESULT - approximation to the integral =   -628.46
ABSERR - estimate of the absolute error =  0.13E-01
KOUNT  - number of function evaluations =  255
IW(1)  - number of subintervals used =    8
```

D01ARF – NAG Fortran Library Routine Document

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

1. Purpose

D01ARF computes definite and indefinite integrals over a finite range to a specified relative or absolute accuracy, using the method described by Patterson.

2. Specification

```
      SUBROUTINE D01ARF (A, B, FUN, RELACC, ABSACC, MAXRUL, IPARM, ACC, ANS,
     1                   N, ALPHA, IFAIL)
      INTEGER            MAXRUL, IPARM, N, IFAIL
      real               A, B, FUN, RELACC, ABSACC, ACC, ANS, ALPHA(390)
      EXTERNAL           FUN
```

3. Description

This routine evaluates definite and indefinite integrals of the form:

$$\int_a^b f(t)\;dt$$

using the method described by Patterson [1].

3.1. Definite Integrals

In this case the routine must be called with IPARM = 0. By linear transformation the integral is changed to,

$$I = \int_{-1}^{+1} F(x)\;dx$$

where $F(x) = \frac{b-a}{2} f\left(\frac{b+a+(b-a)x}{2}\right)$

and is then approximated by an n-point quadrature rule,

$$I = \sum_{k=1}^{n} w_k\; F(x_k)$$

where w_k are the weights and x_k are the abscissae.

The routine uses a family of 9 interlacing rules based on the optimal extension of the three-point Gauss rule. These rules use 1, 3, 7, 15, 31, 63, 127, 255 and 511 points and have respective polynomial integrating degrees 1, 5, 11, 23, 47, 95, 191, 383 and 767. Each rule has the property that the next in sequence includes all the points of its predecessor and has the greatest possible increase in integrating degree.

The integration method is based on the successive application of these rules until the absolute value of the difference of two successive results differs by not more than ABSACC, or relatively by not more than RELACC. The result of the last rule used is taken as the value of the integral (ANS), and the absolute difference of the results of the last two rules used is taken as an estimate of the absolute error (ACC). Due to their interlacing form no integrand evaluations are wasted in passing from one rule to the next.

3.2. Indefinite Integrals

Suppose the value of the integral,

$$\int_c^d f(t)\ dt$$

is required for a number of subintervals $[c,d]$, all of which lie in a interval $[a,b]$.

In this case the routine should first be called with the parameter IPARM = 1 and the interval set to $[a,b]$. The routine then calculates the integral over $[a,b]$ **and** the Legendre expansion of the integrand, using the same integrand values. If the routine is subsequently called with IPARM = 2 and the interval set to $[c,d]$, the integral over $[c,d]$ is calculated by analytical integration of the Legendre expansion, without further evaluations of the integrand.

For the interval [–1,1] the expansion takes the form,

$$F(x) = \sum_{i=0}^{\infty} \alpha_i\ P_i(x)$$

where $P_i(x)$ is the order i Legendre polynomial. Assuming that the integral over the full range [–1,1] was evaluated to the required accuracy using an n-point rule, then the coefficients,

$$\alpha_i = \tfrac{1}{2}(2i-1) \int_{-1}^{+1} P_i(x)\ F(x)\ dx, \qquad i = 0,1,\ldots,m$$

are evaluated by that same rule, up to

$$m = (3n-1)/4.$$

The accuracy for indefinite integration should be of the same order as that obtained for the definite integral over the full range. The indefinite integrals will be exact when $F(x)$ is a polynomial of degree $\leq m$.

4. References

[1] PATTERSON, T.N.L.
The Optimum Addition of Points to Quadrature Formulae.
Math. Comp., 22, pp. 847-856, 1968.

5. Parameters

1: A – ***real.*** *Input*

On entry: the lower limit of integration, a.

2: B – ***real.*** *Input*

On entry: the upper limit of integration, b. It is not necessary that $a < b$.

3: FUN – ***real*** FUNCTION, supplied by the user. *External Procedure*

FUN must evaluate the integrand f at a specified point.

Its specification is:

```
real FUNCTION  FUN(X)
real           X
```

1: X – ***real.***

On entry: the point in $[a,b]$ at which the integrand must be evaluated.

FUN must be declared as EXTERNAL in the (sub)program from which D01ARF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

If IPARM = 2, FUN is not called.

4: RELACC – ***real.*** *Input*

On entry: the relative accuracy required. If convergence according to absolute accuracy is required, RELACC should be set to zero (but see also Section 7). If RELACC < 0.0, its absolute value is used.

If IPARM = 2, RELACC is not used.

5: ABSACC – ***real.*** *Input*

On entry: the absolute accuracy required. If convergence according to relative accuracy is required, ABSACC should be set to zero (but see also Section 7). If ABSACC < 0.0, its absolute value is used.

If IPARM = 2, ABSACC is not used.

6: MAXRUL – INTEGER. *Input*

On entry: the maximum number of successive rules that may be used.

Constraint: 1 ≤ MAXRUL ≤ 9. If MAXRUL is outside these limits, the value 9 is assumed.

If IPARM = 2, MAXRUL is not used.

7: IPARM – INTEGER. *Input*

On entry: IPARM indicates the task to be performed by the routine:

if IPARM = 0, only the definite integral over $[a,b]$ is evaluated.

if IPARM = 1, as well as the definite integral, the expansion of the integrand in Legendre polynomials over $[a,b]$ is calculated, using the same values of the integrand as used to compute the integral. The expansion coefficients, and some other quantities, are returned in ALPHA for later use in computing indefinite integrals.

if IPARM = 2, $f(t)$ is integrated analytically over $[a,b]$ using the previously computed expansion, stored in ALPHA. No further evaluations of the integrand are required. The routine must previously have been called with IPARM = 1 and the interval $[a,b]$ must lie within that specified for the previous call. In this case only the arguments A, B, IPARM, ANS, ALPHA and IFAIL are used.

Constraint: IPARM = 0, 1 or 2.

8: ACC – ***real.*** *Output*

On exit: if IPARM = 0 or 1, ACC contains the absolute value of the difference between the last two successive estimates of the integral. This may be used as a measure of the accuracy actually achieved.

If IPARM = 2, ACC is not used.

9: ANS – ***real.*** *Output*

On exit: the estimated value of the integral.

10: N – INTEGER. *Output*

On exit: when IPARM = 0 or 1, N contains the number of integrand evaluations used in the calculation of the integral.

If IPARM = 2, N is not used.

11: ALPHA(390) – ***real*** array. *Input/Output*

On entry: if IPARM = 2, ALPHA must contain the coefficients of the Legendre expansions of the integrand, as returned by a previous call of D01ARF with IPARM = 1 and a range containing the present range. If IPARM = 0 or 1, ALPHA need not be set on entry.

On exit: if IPARM = 1, the first m elements of ALPHA hold the coefficients of the Legendre expansion of the integrand, and the value of m is stored in ALPHA(390).

ALPHA must not be changed between a call with IPARM = 1 and subsequent calls with IPARM = 2.

If IPARM = 2, the first m elements of ALPHA are unchanged on exit.

12: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.** To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL = 1

If IPARM = 0 or 1, this indicates that all MAXRUL rules have been used and the integral has not converged to the accuracy requested. In this case ANS contains the last approximation to the integral, and ACC contains the difference between the last two approximations. To check this estimate of the integral, D01ARF could be called again to evaluate,

$$\int_a^b f(t)\,dt \quad \text{as} \quad \int_a^c f(t)\,dt + \int_c^b f(t)\,dt \quad \text{for some } a < c < b.$$

If IPARM = 2, this indicates failure of convergence during the run with IPARM = 1 in which the Legendre expansion was created.

IFAIL = 2

On entry, IPARM < 0 or IPARM > 2.

IFAIL = 3

The routine is called with IPARM = 2 but a previous call with IPARM = 1 has been omitted or was invoked with an integration interval of length zero.

IFAIL = 4

On entry, with IPARM = 2, the interval for indefinite integration is not contained within the interval specified when the routine was previously called with IPARM = 1.

7. Accuracy

The relative or absolute accuracy required is specified by the user in the variables RELACC or ABSACC. The routine will terminate whenever either the relative accuracy specified by RELACC or the absolute accuracy specified by ABSACC is reached. One or other of these criteria may be 'forced' by setting the parameter for the other to zero. If both RELACC and ABSACC are specified as zero, then the routine uses the value 10.0×(***machine precision***) for RELACC.

If on exit IFAIL = 0, then it is likely that the result is correct to one or other of these accuracies. If on exit IFAIL = 1, then it is likely that neither of the requested accuracies has been reached.

When the user has no prior idea of the magnitude of the integral, it is possible that an unreasonable accuracy may be requested, e.g. a relative accuracy for an integral which turns out to be zero, or a small absolute accuracy for an integral which turns out to be very large. Even if failure is reported in such a case, the value of the integral may still be satisfactory. The device of setting the other 'unused' accuracy parameter to a small positive value (e.g. 10^{-9} for an implementation of 11-digit precision) rather than zero, may prevent excessive calculation in such a situation.

To avoid spurious convergence, it is recommended that relative accuracies larger than about 10^{-3} be avoided.

8. Further Comments

The time taken by the routine depends on the complexity of the integrand and the accuracy required.

This routine uses the Patterson method over the whole integration interval and should therefore be suitable for well behaved functions. However, for very irregular functions it would be more efficient to submit the differently behaved regions separately for integration.

9. Example

The program evaluates the following integrals:

(i) Definite integral only (IPARM = 0) for

$$\int_0^1 \frac{4}{1+x^2}\,dx \qquad (\text{ABSACC} = 10^{-5}).$$

(ii) Definite integral together with expansion coefficients (IPARM = 1) for

$$\int_1^2 \sqrt[8]{x}\,dx \qquad (\text{ABSACC} = 10^{-5}).$$

(iii) Indefinite integral using previous expansion (IPARM = 2) for

$$\int_{1.2}^{1.8} \sqrt[8]{x}\,dx \qquad (\text{ABSACC} = 10^{-5}).$$

9.1. Program Text

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

```
*     D01ARF Example Program Text
*     Mark 16 Revised. NAG Copyright 1993.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          MAXRUL
      PARAMETER        (MAXRUL=0)
*     .. Local Scalars ..
      real             A, ABSACC, ACC, ANS, B, RELACC
      INTEGER          IFAIL, IPARM, N
*     .. Local Arrays ..
      real             ALPHA(390)
*     .. External Functions ..
      real             F1, F2
      EXTERNAL         F1, F2
*     .. External Subroutines ..
      EXTERNAL         D01ARF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01ARF Example Program Results'
      RELACC = 0.0e0
      ABSACC = 1.0e-5
*     Definite integral of F1(x) - no expansion
      IPARM = 0
      IFAIL = 1
      A = 0.0e0
      B = 1.0e0
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Definite integral of 4/(1+x*x) over (0,1)'
*
      CALL D01ARF(A,B,F1,RELACC,ABSACC,MAXRUL,IPARM,ACC,ANS,N,ALPHA,
     +            IFAIL)
*
```

```
      IF (IFAIL.NE.0) WRITE (NOUT,99997) 'D01ARF fails. IFAIL =', IFAIL
      IF (IFAIL.LE.1) THEN
         WRITE (NOUT,99999) 'Estimated value of the integral =', ANS
         WRITE (NOUT,99998) 'Estimated absolute error =', ACC
         WRITE (NOUT,99997) 'Number of points used =', N
      END IF
*     Definite integral of F2(x) - with expansion
      IPARM = 1
      IFAIL = 1
      A = 1.0e0
      B = 2.0e0
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Definite integral of x**(1/8) over (1,2)'
*
      CALL D01ARF(A,B,F2,RELACC,ABSACC,MAXRUL,IPARM,ACC,ANS,N,ALPHA,
     +            IFAIL)
*
      IF (IFAIL.NE.0) WRITE (NOUT,99997) 'D01ARF fails. IFAIL =', IFAIL
      IF (IFAIL.LE.1) THEN
         WRITE (NOUT,99999) 'Estimated value of the integral =', ANS
         WRITE (NOUT,99998) 'Estimated absolute error =', ACC
         WRITE (NOUT,99997) 'Number of points used =', N
      END IF
*     Indefinite integral of F2(x)
      IPARM = 2
      IFAIL = 0
      A = 1.2e0
      B = 1.8e0
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Indefinite integral of x**(1/8) over (1.2,1.8)'
*
      CALL D01ARF(A,B,F2,RELACC,ABSACC,MAXRUL,IPARM,ACC,ANS,N,ALPHA,
     +            IFAIL)
*
      WRITE (NOUT,99999) 'Estimated value of the integral =', ANS
      STOP
*
99999 FORMAT (1X,A,F9.5)
99998 FORMAT (1X,A,e10.2)
99997 FORMAT (1X,A,I4)
      END
*
      real FUNCTION F1(X)
*     .. Scalar Arguments ..
      real             X
*     .. Executable Statements ..
      F1 = 4.0e0/(1.0e0+X*X)
      RETURN
      END
*
      real FUNCTION F2(X)
*     .. Scalar Arguments ..
      real             X
*     .. Executable Statements ..
      F2 = X**0.125e0
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D01ARF Example Program Results

Definite integral of 4/(1+x*x) over (0,1)
Estimated value of the integral =  3.14159
Estimated absolute error =  0.18E-07
Number of points used =  15

Definite integral of x**(1/8) over (1,2)
Estimated value of the integral =  1.04979
Estimated absolute error =  0.59E-06
Number of points used =   7

Indefinite integral of x**(1/8) over (1.2,1.8)
Estimated value of the integral =  0.63073
```

D01ASF – NAG Fortran Library Routine Document

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

1. Purpose

D01ASF calculates an approximation to the sine or the cosine transform of a function g over $[a,\infty)$:

$$I = \int_a^\infty g(x)\sin(\omega x)dx \quad \text{or} \quad I = \int_a^\infty g(x)\cos(\omega x)dx$$

(for a user-specified value of ω).

2. Specification

```
      SUBROUTINE D01ASF (G, A, OMEGA, KEY, EPSABS, RESULT, ABSERR, LIMLST,
     1                   LST, ERLST, RSLST, IERLST, W, LW, IW, LIW, IFAIL)
      INTEGER          KEY, LIMLST, LST, IERLST(LIMLST), LW, IW(LIW), LIW,
     1                 IFAIL
      real             G, A, OMEGA, EPSABS, RESULT, ABSERR, ERLST(LIMLST),
     1                 RSLST(LIMLST), W(LW)
      EXTERNAL         G
```

3. Description

D01ASF is based upon the QUADPACK routine QAWFE (Piessens *et al.* [2]). It is an adaptive routine, designed to integrate a function of the form $g(x)w(x)$ over a semi-infinite interval, where $w(x)$ is either $\sin(\omega x)$ or $\cos(\omega x)$. Over successive intervals

$$C_k = [a+(k-1)c,\ a+kc], \qquad k = 1,2,...,\text{LST}$$

integration is performed by the same algorithm as is used by D01ANF. The intervals C_k are of constant length

$$c = \{2[|\omega|]+1\}\pi/|\omega|, \qquad \omega \neq 0,$$

where $[|\omega|]$ represents the largest integer less than or equal to $|\omega|$. Since c equals an odd number of half periods, the integral contributions over succeeding intervals will alternate in sign when the function g is positive and monotonically decreasing over $[a,\infty)$. The algorithm, described by [2], incorporates a global acceptance criterion (as defined by Malcolm and Simpson [1]) together with the ε-algorithm (Wynn [3]) to perform extrapolation. The local error estimation is described by Piessens *et al.* [2].

If $\omega = 0$ and KEY = 1, the routine uses the same algorithm as D01AMF (with EPSREL = 0.0).

In contrast to the other routines in Chapter D01, D01ASF works only with a user-specified **absolute** error tolerance (EPSABS). Over the interval C_k it attempts to satisfy the absolute accuracy requirement

$$\text{EPSA}_k = U_k \times \text{EPSABS},$$

where $U_k = (1-p)p^{k-1}$, for $k = 1,2,...$ and $p = 0.9$.

However, when difficulties occur during the integration over the kth sub-interval C_k such that the error flag IERLST(k) is non-zero, the accuracy requirement over subsequent intervals is relaxed. See Piessens *et al.* [2] for more details.

4. References

[1] MALCOLM, M.A. and SIMPSON, R.B.
Local Versus Global Strategies for Adaptive Quadrature.
ACM Trans. Math. Softw., 1, pp. 129-146, 1976.

[2] PIESSENS, R., DE DONCKER-KAPENGA, E., ÜBERHUBER. C. and KAHANER, D.
QUADPACK, A Subroutine Package for Automatic Integration.
Springer-Verlag, 1983.

[3] WYNN, P.
On a Device for Computing the $e_m(S_n)$ Transformation.
Math. Table Aids Comput., 10, pp. 91-96, 1956.

5. Parameters

1: G – ***real*** FUNCTION, supplied by the user. *External Procedure*

G must return the value of the function g at a given point.

Its specification is:

```
real FUNCTION G(X)
real          X
```

1: X – ***real.*** *Input*

On entry: the point at which the function g must be evaluated.

G must be declared as EXTERNAL in the (sub)program from which D01ASF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: A – ***real.*** *Input*

On entry: the lower limit of integration, a.

3: OMEGA – ***real.*** *Input*

On entry: the parameter ω in the weight function of the transform.

4: KEY – INTEGER. *Input*

On entry: indicates which integral is to be computed:

if KEY = 1, $w(x) = \cos(\omega x)$;

if KEY = 2, $w(x) = \sin(\omega x)$.

Constraint: KEY = 1 or 2.

5: EPSABS – ***real.*** *Input*

On entry: the absolute accuracy requirement. If EPSABS is negative, the absolute value is used. See Section 7.

6: RESULT – ***real.*** *Output*

On exit: the approximation to the integral I.

7: ABSERR – ***real.*** *Output*

On exit: an estimate of the modulus of the absolute error, which should be an upper bound for $|I\text{–RESULT}|$.

8: LIMLST – INTEGER. *Input*

On entry: an upper bound on the number of intervals C_k needed for the integration.

Suggested value: LIMLST = 50 is adequate for most problems.

Constraint: LIMLST ≥ 3.

9: LST – INTEGER. *Output*

On exit: the number of intervals C_k actually used for the integration.

10: ERLST(LIMLST) – ***real*** array. *Output*

On exit: ERLST(k) contains the error estimate corresponding to the integral contribution over the interval C_k, for k = 1,2,...,LST.

11: RSLST(LIMLST) – ***real*** array. *Output*

On exit: RSLST(k) contains the integral contribution over the interval C_k for k = 1,2,...,LST.

12: IERLST(LIMLST) – INTEGER array. *Output*

On exit: IERLST(k) contains the error flag corresponding to RSLST(k), for k = 1,2,...,LST. See Section 6.

13: W(LW) – ***real*** array. *Workspace*

14: LW – INTEGER. *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D01ASF is called. The value of LW (together with that of LIW below) imposes a bound on the number of sub-intervals into which each interval C_k may be divided by the routine. The number of sub-intervals cannot exceed LW/4. The more difficult the integrand, the larger LW should be.

Suggested value: a value in the range 800 to 2000 is adequate for most problems.

Constraint: LW ≥ 4.

15: IW(LIW) – INTEGER array. *Output*

On exit: IW(1) contains the maximum number of sub-intervals actually used for integrating over any of the intervals C_k. The rest of the array is used as workspace.

16: LIW – INTEGER. *Input*

On entry: the dimension of the array IW as declared in the (sub)program from which D01ASF is called. The number of sub-intervals into which each interval C_k may be divided cannot exceed LIW/2.

Suggested value: LIW = LW/2.

Constraint: LIW ≥ 2.

17: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL = 1

The maximum number of subdivisions allowed with the given workspace has been reached without the accuracy requirements being achieved. Look at the integrand in order to determine the integration difficulties. If the position of a local difficulty within the interval can be determined (e.g. a singularity of the integrand or its derivative, a peak, a discontinuity, etc.) you will probably gain from splitting up the interval at this point and calling D01ASF on the infinite subrange and an appropriate integrator on the finite subrange. Alternatively, consider relaxing the accuracy requirements specified by EPSABS or increasing the amount of workspace.

IFAIL = 2

Round-off error prevents the requested tolerance from being achieved. The error may be underestimated. Consider requesting less accuracy.

IFAIL = 3

Extremely bad local integrand behaviour causes a very strong subdivision around one (or more) points of the interval. The same advice applies as in the case of IFAIL = 1.

IFAIL = 4

The requested tolerance cannot be achieved, because the extrapolation does not increase the accuracy satisfactorily; the returned result is the best which can be obtained. The same advice applies as in the case of IFAIL = 1. Look at the integrand in order to determine the integration difficulties. If the position of a local difficulty within the interval can be determined (e.g. a singularity of the integrand or its derivative, a peak, a discontinuity ...) you will probably gain from splitting up the interval at this point and calling D01ASF on the infinite subrange and an appropriate integrator on the finite subrange. Alternatively, consider relaxing the accuracy requirements specified by EPSABS or increasing the amount of workspace.

Please note that divergence can occur with any non-zero value of IFAIL.

IFAIL = 5

The integral is probably divergent, or slowly convergent. Please note that divergence can occur with any non-zero value of IFAIL.

IFAIL = 6

On entry, KEY < 1,
or KEY > 2,
or LIMLST < 3.

IFAIL = 7

Bad integration behaviour occurs within one or more of the intervals C_k. Location and type of the difficulty involved can be determined from the vector IERLST (see below).

IFAIL = 8

Maximum number of intervals C_k (= LIMLST) allowed has been achieved. Increase the value of LIMLST to allow more cycles.

IFAIL = 9

The extrapolation table constructed for convergence acceleration of the series formed by the integral contribution over the intervals C_k, does not converge to the required accuracy.

IFAIL = 10

On entry, LW < 4,
or LIW < 2.

In the cases IFAIL = 7, 8 or 9, additional information about the cause of the error can be obtained from the array IERLST, as follows:

IERLST(k) = 1

The maximum number of subdivisions = min(LW/4,LIW/2) has been achieved on the kth interval.

IERLST(k) = 2

Occurrence of round-off error is detected and prevents the tolerance imposed on the kth interval from being achieved.

IERLST(k) = 3

Extremely bad integrand behaviour occurs at some points of the kth interval.

IERLST(k) = 4

The integration procedure over the kth interval does not converge (to within the required accuracy) due to round-off in the extrapolation procedure invoked on this interval. It is assumed that the result on this interval is the best which can be obtained.

IERLST(k) = 5

The integral over the kth interval is probably divergent or slowly convergent. It must be noted that divergence can occur with any other value of IERLST(k).

7. Accuracy

The routine cannot guarantee, but in practice usually achieves, the following accuracy:

$|I\text{–RESULT}| \le |\text{EPSABS}|$.

where EPSABS is the user-specified absolute error tolerance. Moreover, it returns the quantity ABSERR, which, in normal circumstances, satisfies

$|I\text{–RESULT}| \le \text{ABSERR} \le |\text{EPSABS}|$.

8. Further Comments

None.

9. Example

To compute $\displaystyle\int_0^\infty \frac{1}{\sqrt{x}} \cos(\pi x/2)dx$.

9.1. Program Text

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

```
*     D01ASF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          LW, LIW, LIMLST
      PARAMETER        (LW=800,LIW=LW/2,LIMLST=50)
*     .. Scalars in Common ..
      INTEGER          KOUNT
```

```
*     .. Local Scalars ..
      real              A, ABSERR, EPSABS, OMEGA, RESULT
      INTEGER           IFAIL, INTEGR, LST
*     .. Local Arrays ..
      real              ERLST(LIMLST), RSLST(LIMLST), W(LW)
      INTEGER           IERLST(LIMLST), IW(LIW)
*     .. External Functions ..
      real              FST, X01AAF
      EXTERNAL          FST, X01AAF
*     .. External Subroutines ..
      EXTERNAL          D01ASF
*     .. Common blocks ..
      COMMON            /TELNUM/KOUNT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01ASF Example Program Results'
      EPSABS = 1.0e-03
      A = 0.0e0
      KOUNT = 0
      OMEGA = 0.5e0*X01AAF(0.0e0)
      INTEGR = 1
      IFAIL = -1
*
      CALL D01ASF(FST,A,OMEGA,INTEGR,EPSABS,RESULT,ABSERR,LIMLST,LST,
     +            ERLST,RSLST,IERLST,W,LW,IW,LIW,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'A      - lower limit of integration = ', A
      WRITE (NOUT,*) 'B      - upper limit of integration =  infinity'
      WRITE (NOUT,99998) 'EPSABS - absolute accuracy requested = ',
     +  EPSABS
      WRITE (NOUT,*)
      IF (IFAIL.NE.0) WRITE (NOUT,99996) 'IFAIL = ', IFAIL
      IF (IFAIL.NE.6 .AND. IFAIL.NE.10) THEN
         WRITE (NOUT,99997) 'RESULT - approximation to the integral = ',
     +     RESULT
         WRITE (NOUT,99998) 'ABSERR - estimate of the absolute error = '
     +     , ABSERR
         WRITE (NOUT,99996) 'KOUNT  - number of function evaluations = '
     +     , KOUNT
         WRITE (NOUT,99996) 'LST    - number of intervals used = ', LST
         WRITE (NOUT,99996)
     +   'IW(1)  - max. no. of subintervals used in any one interval = '
     +     , IW(1)
      END IF
      STOP
*
99999 FORMAT (1X,A,F10.4)
99998 FORMAT (1X,A,e9.2)
99997 FORMAT (1X,A,F9.5)
99996 FORMAT (1X,A,I4)
      END
*
      real FUNCTION FST(X)
*     .. Scalar Arguments ..
      real              X
*     .. Scalars in Common ..
      INTEGER           KOUNT
*     .. Intrinsic Functions ..
      INTRINSIC         SQRT
*     .. Common blocks ..
      COMMON            /TELNUM/KOUNT
*     .. Executable Statements ..
      KOUNT = KOUNT + 1
      FST = 0.0e0
      IF (X.GT.0.0e0) FST = 1.0e0/SQRT(X)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
DO1ASF Example Program Results

A      - lower limit of integration =     0.0000
B      - upper limit of integration =  infinity
EPSABS - absolute accuracy requested =  0.10E-02

RESULT - approximation to the integral =   1.00000
ABSERR - estimate of the absolute error =  0.59E-03
KOUNT  - number of function evaluations =  380
LST    - number of intervals used =    6
IW(1)  - max. no. of subintervals used in any one interval =    8
```

D01ATF – NAG Fortran Library Routine Document

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

1. Purpose

D01ATF is a general-purpose integrator which calculates an approximation to the integral of a function $f(x)$ over a finite interval $[a,b]$:

$$I = \int_a^b f(x)\ dx.$$

2. Specification

```
      SUBROUTINE D01ATF (F, A, B, EPSABS, EPSREL, RESULT, ABSERR, W,
     1                   LW, IW, LIW, IFAIL)
      INTEGER          LW, IW(LIW), LIW, IFAIL
      real             A, B, EPSABS, EPSREL, RESULT, ABSERR, W(LW)
      EXTERNAL         F
```

3. Description

D01ATF is based upon the QUADPACK routine QAGS (Piessens *et al.* [3]). It is an adaptive routine, using the Gauss 10-point and Kronrod 21-point rules. The algorithm, described by de Doncker [1], incorporates a global acceptance criterion (as defined by Malcolm and Simpson [2]) together with the ε-algorithm (Wynn [4]) to perform extrapolation. The local error estimation is described by Piessens *et al.* [3].

The routine is suitable as a general purpose integrator, and can be used when the integrand has singularities, especially when these are of algebraic or logarithmic type.

The routine requires a user-supplied subroutine to evaluate the integrand at an array of different points and is therefore particularly efficient when the evaluation can be performed in vector mode on a vector-processing machine. Otherwise the algorithm is identical to that used by D01AJF.

4. References

[1] DE DONCKER, E.
An Adaptive Extrapolation Algorithm for Automatic Integration.
Signum Newsletter, 13, 2, pp. 12-18, 1978.

[2] MALCOLM, M.A. and SIMPSON, R.B.
Local Versus Global Strategies for Adaptive Quadrature.
ACM Trans. Math. Softw., 1, pp. 129-146, 1976.

[3] PIESSENS, R., DE DONCKER-KAPENGA, E., ÜBERHUBER, C. and KAHANER, D.
QUADPACK, A Subroutine Package for Automatic Integration.
Springer-Verlag, 1983.

[4] WYNN, P.
On a Device for Computing the $e_m(S_n)$ Transformation.
Math. Tables Aids Comput., 10, pp. 91-96, 1956.

5. Parameters

1: F – SUBROUTINE, supplied by the user. *External Procedure*

F must return the values of the integrand f at a set of points.

Its specification is:

```
SUBROUTINE F(X, FV, N)
INTEGER    N
real       X(N), FV(N)
```

1: X(N) – ***real*** array. *Input*

On entry: the points at which the integrand f must be evaluated.

2: FV(N) – ***real*** array. *Output*

On exit: FV(j) must contain the value of f at the point X(j), for j = 1,2,...,N.

3: N – INTEGER. *Input*

On entry: the number of points at which the integrand is to be evaluated. The actual value of N is always 21.

F must be declared as EXTERNAL in the (sub)program from which D01ATF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: A – ***real.*** *Input*

On entry: the lower limit of integration, a.

3: B – ***real.*** *Input*

On entry: the upper limit of integration, b. It is not necessary that $a < b$.

4: EPSABS – ***real.*** *Input*

On entry: the absolute accuracy required. If EPSABS is negative, the absolute value is used. See Section 7.

5: EPSREL – ***real.*** *Input*

On entry: the relative accuracy required. If EPSREL is negative, the absolute value is used. See Section 7.

6: RESULT – ***real.*** *Output*

On exit: the approximation to the integral I.

7: ABSERR – ***real.*** *Output*

On exit: an estimate of the modulus of the absolute error, which should be an upper bound for $|I\text{–RESULT}|$.

8: W(LW) – ***real*** array. *Output*

On exit: details of the computation, as described in Section 8.

9: LW – INTEGER. *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D01ATF is called. The value of LW (together with that of LIW below) imposes a bound on the number of sub-intervals into which the interval of integration may be divided by the routine. The number of sub-intervals cannot exceed LW/4. The more difficult the integrand, the larger LW should be.

Suggested value: a value in the range of 800 to 2000 is adequate for most problems.

Constraint: LW ≥ 4.

10: IW(LIW) – INTEGER array. *Output*

On exit: IW(1) contains the actual number of sub-intervals used. The rest of the array is used as workspace.

11: LIW – INTEGER. *Input*

On entry: the dimension of the array IW as declared in the (sub)program from which D01ATF is called. The number of sub-intervals into which the interval of integration may be divided cannot exceed LIW.

Suggested value: LIW = LW/4.

Constraint: LIW ≥ 1.

12: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL = 1

The maximum number of subdivisions allowed with the given workspace has been reached without the accuracy requirements being achieved. Look at the integrand in order to determine the integration difficulties. If the position of a local difficulty within the interval can be determined (e.g. a singularity of the integrand or its derivative, a peak, a discontinuity, etc.) you will probably gain from splitting up the interval at this point and calling the integrator on the subranges. If necessary, another integrator, which is designed for handling the type of difficulty involved, must be used. Alternatively, consider relaxing the accuracy requirements specified by EPSABS and EPSREL, or increasing the amount of workspace.

IFAIL = 2

Round-off error prevents the requested tolerance from being achieved. The error may be underestimated. Consider requesting less accuracy.

IFAIL = 3

Extremely bad local integrand behaviour causes a very strong subdivision around one (or more) points of the interval. The same advice applies as in the case of IFAIL = 1.

IFAIL = 4

The requested tolerance cannot be achieved, because the extrapolation does not increase the accuracy satisfactorily; the returned result is the best which can be obtained. The same advice applies as in the case of IFAIL = 1.

IFAIL = 5

The integral is probably divergent, or slowly convergent. Please note that divergence can occur with any non-zero value of IFAIL.

IFAIL = 6

On entry, LW < 4,
or LIW < 1.

7. Accuracy

The routine cannot guarantee, but in practice usually achieves, the following accuracy:

$$|I-\text{RESULT}| \leq tol$$

where

$$tol = \max\{|\text{EPSABS}|,|\text{EPSREL}|\times|I|\}$$

and EPSABS and EPSREL are user-specified absolute and relative error tolerance. Moreover it returns the quantity ABSERR which, in normal circumstances, satisfies

$$|I-\text{RESULT}| \leq \text{ABSERR} \leq tol.$$

8. Further Comments

If IFAIL ≠ 0 on exit, then the user may wish to examine the contents of the array W, which contains the end-points of the sub-intervals used by D01ATF along with the integral contributions and error estimates over the sub-intervals.

Specifically, for $i = 1,2,...,n$, let r_i denote the approximation to the value of the integral over the sub-interval $[a_i,b_i]$, in the partition of $[a,b]$, and e_i be the corresponding absolute error estimate. Then, $\int_{a_i}^{b_i} f(x)\,dx \simeq r_i$ and RESULT $= \sum_{i=1}^{n} r_i$, unless D01ATF terminates while testing for divergence of the integral (see Piessens *et al.* [3], Section 3.4.3). In this case, RESULT (and ABSERR) are taken to be the values returned from the extrapolation process. The value of n is returned in IW(1), and the values of a_i, b_i, e_i and r_i are stored consecutively in the array W, that is:

a_i = W(i),
b_i = W($n+i$),
e_i = W($2n+i$) and
r_i = W($3n+i$).

9. Example

To compute

$$\int_0^{2\pi} \frac{x\sin(30x)}{\sqrt{1-(x/2\pi)^2}}\,dx.$$

9.1. Program Text

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

```
*     D01ATF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          LW, LIW
      PARAMETER        (LW=800,LIW=LW/4)
*     .. Scalars in Common ..
      real             PI
      INTEGER          KOUNT
```

```
*     .. Local Scalars ..
      real                 A, ABSERR, B, EPSABS, EPSREL, RESULT
      INTEGER              IFAIL
*     .. Local Arrays ..
      real                 W(LW)
      INTEGER              IW(LIW)
*     .. External Functions ..
      real                 X01AAF
      EXTERNAL             X01AAF
*     .. External Subroutines ..
      EXTERNAL             D01ATF, FST
*     .. Common blocks ..
      COMMON               /TELNUM/PI, KOUNT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01ATF Example Program Results'
      PI = X01AAF(0.0e0)
      EPSABS = 0.0e0
      EPSREL = 1.0e-04
      A = 0.0e0
      B = 2.0e0*PI
      KOUNT = 0
      IFAIL = -1
*
      CALL D01ATF(FST,A,B,EPSABS,EPSREL,RESULT,ABSERR,W,LW,IW,LIW,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'A      - lower limit of integration = ', A
      WRITE (NOUT,99999) 'B      - upper limit of integration = ', B
      WRITE (NOUT,99998) 'EPSABS - absolute accuracy requested = ',
     +  EPSABS
      WRITE (NOUT,99998) 'EPSREL - relative accuracy requested = ',
     +  EPSREL
      WRITE (NOUT,*)
      IF (IFAIL.NE.0) WRITE (NOUT,99996) 'IFAIL = ', IFAIL
      IF (IFAIL.LE.5) THEN
         WRITE (NOUT,99997) 'RESULT - approximation to the integral = ',
     +     RESULT
         WRITE (NOUT,99998) 'ABSERR - estimate of the absolute error = '
     +     , ABSERR
         WRITE (NOUT,99996)
     +     'KOUNT  - number of function evaluations   = ', KOUNT
         WRITE (NOUT,99996) 'IW(1)  - number of subintervals used = ',
     +     IW(1)
      END IF
      STOP
*
99999 FORMAT (1X,A,F10.4)
99998 FORMAT (1X,A,e9.2)
99997 FORMAT (1X,A,F9.5)
99996 FORMAT (1X,A,I4)
      END
*
      SUBROUTINE FST(X,FV,N)
*     .. Scalar Arguments ..
      INTEGER           N
*     .. Array Arguments ..
      real              FV(N), X(N)
*     .. Scalars in Common ..
      real              PI
      INTEGER           KOUNT
*     .. Local Scalars ..
      INTEGER           I
*     .. Intrinsic Functions ..
      INTRINSIC         SIN, SQRT
*     .. Common blocks ..
      COMMON            /TELNUM/PI, KOUNT
```

```
*     .. Executable Statements ..
      KOUNT = KOUNT + N
      DO 20 I = 1, N
         FV(I) = X(I)*SIN(30.0e0*X(I))/SQRT(1.0e0-X(I)**2/(4.0e0*PI**2))
   20 CONTINUE
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D01ATF Example Program Results

A      - lower limit of integration =     0.0000
B      - upper limit of integration =     6.2832
EPSABS - absolute accuracy requested =  0.00E+00
EPSREL - relative accuracy requested =  0.10E-03

RESULT - approximation to the integral =  -2.54326
ABSERR - estimate of the absolute error =  0.13E-04
KOUNT  - number of function evaluations  =  777
IW(1)  - number of subintervals used =   19
```

D01AUF – NAG Fortran Library Routine Document

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

1. Purpose

D01AUF is an adaptive integrator, especially suited to oscillating, non-singular integrands, which calculates an approximation to the integral of a function $f(x)$ over a finite interval $[a,b]$:

$$I = \int_a^b f(x)\ dx.$$

2. Specification

```
      SUBROUTINE D01AUF (F, A, B, KEY, EPSABS, EPSREL, RESULT, ABSERR,
     1                   W, LW, IW, LIW, IFAIL)
      INTEGER        KEY, LW, IW(LIW), LIW, IFAIL
      real           A, B, EPSABS, EPSREL, RESULT, ABSERR, W(LW)
      EXTERNAL       F
```

3. Description

D01AUF is based upon the QUADPACK routine QAG (Piessens *et al.* [3]). It is an adaptive routine, offering a choice of six Gauss-Kronrod rules. A global acceptance criterion (as defined by Malcolm and Simpson [1]) is used. The local error estimation is described by Piessens *et al.* [3].

Because this routine is based on integration rules of high order, it is especially suitable for non-singular oscillating integrands.

The routine requires a user-supplied subroutine to evaluate the integrand at an array of different points and is therefore particularly efficient when the evaluation can be performed in vector mode on a vector-processing machine. Otherwise this algorithm with KEY = 6 is identical to that used by D01AKF.

4. References

[1] MALCOLM, M.A. and SIMPSON, R.B.
Local Versus Global Strategies for Adaptive Quadrature.
A.C.M. Trans. Math. Software, 1, pp. 129-146, 1975.

[2] PIESSENS, R.
An Algorithm for Automatic Integration.
Angewandte Informatik, 15, pp. 399-401, 1973.

[3] PIESSENS, R., DE DONCKER-KAPENGA, E., ÜBERHUBER, C. and KAHANER, D.
QUADPACK, A Subroutine Package for Automatic Integration.
Springer-Verlag, 1983.

5. Parameters

1: F – SUBROUTINE, supplied by the user. *External Procedure*

F must return the values of the integrand f at a set of points.

Its specification is:

```
SUBROUTINE F(X, FV, N)
INTEGER    N
real       X(N), FV(N)
```

1: X(N) – ***real*** array. *Input*

On entry: the points at which the integrand f must be evaluated..

2: FV(N) – ***real*** array. *Output*

On exit: FV(j) must contain the value of f at the point X(j), for j = 1,2,...,N.

3: N – INTEGER. *Input*

On entry: the number of points at which the integrand is to be evaluated. The actual value of N is equal to the number of points in the Kronrod rule (see specification of KEY below).

F must be declared as EXTERNAL in the (sub)program from which D01AUF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: A – ***real.*** *Input*

On entry: the lower limit of integration, a.

3: B – ***real.*** *Input*

On entry: the upper limit of integration, b. It is not necessary that $a < b$.

4: KEY – INTEGER. *Input*

On entry: which integration rule is to be used:

if KEY = 1 for the Gauss 7-point and Kronrod 15-point rule,
if KEY = 2 for the Gauss 10-point and Kronrod 21-point rule,
if KEY = 3 for the Gauss 15-point and Kronrod 31-point rule,
if KEY = 4 for the Gauss 20-point and Kronrod 41-point rule,
if KEY = 5 for the Gauss 25-point and Kronrod 51-point rule,
if KEY = 6 for the Gauss 30-point and Kronrod 61-point rule.

Suggested value: KEY = 6.

Constraint: KEY = 1, 2, 3, 4, 5 or 6.

5: EPSABS – ***real.*** *Input*

On entry: the absolute accuracy required. If EPSABS is negative, the absolute value is used. See Section 7.

6: EPSREL – ***real.*** *Input*

On entry: the relative accuracy required. If EPSREL is negative, the absolute value is used. See Section 7.

7: RESULT – ***real.*** *Output*

On exit: the approximation to the integral I.

8: ABSERR – ***real.*** *Output*

On exit: an estimate of the modulus of the absolute error, which should be an upper bound $|I$–RESULT$|$.

9: W(LW) – ***real*** array. *Output*

On exit: details of the computation, as described in Section 8.

10: LW – INTEGER. *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D01AUF is called. The value of LW (together with that of LIW below) imposes a bound on the number of subintervals into which the interval of integration may be divided by the routine. The number of subintervals cannot exceed LW/4. The more difficult the integrand, the larger LW should be.

Suggested value: a value in the range 800 to 2000 is adequate for most problems.

Constraint: LW ≥ 4.

11: IW(LIW) – INTEGER array. *Output*

On exit: IW(1) contains the actual number of subintervals. The rest of the array is used as workspace.

12: LIW – INTEGER. *Input*

On entry: the dimension of the array IW as declared in the (sub)program from which D01AUF is called.

The number of subintervals into which the interval of integration may be divided cannot exceed LIW.

Suggested value: LIW = LW/4.

Constraint: LIW ≥ 1.

13: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL = 1

The maximum number of subdivisions allowed with the given workspace has been reached without the accuracy requirements being achieved. Look at the integrand in order to determine the integration difficulties. Probably another integrator which is designed for handling the type of difficulty involved must be used. Alternatively, consider relaxing the accuracy requirements specified by EPSABS and EPSREL, or increasing the amount of workspace.

IFAIL = 2

Roundoff error prevents the requested tolerance from being achieved. Consider requesting less accuracy.

IFAIL = 3

Extremely bad local integrand behaviour causes a very strong subdivision around one (or more) points of the interval. The same advice applies as in the case of IFAIL = 1.

IFAIL = 4

On entry, KEY < 1,
or KEY > 6.

IFAIL = 5

On entry, LW < 4,
or LIW < 1.

7. Accuracy

The routine cannot guarantee, but in practice usually achieves, the following accuracy:

$$|I-\text{RESULT}| \leq tol$$

where

$$tol = \max\{|\text{EPSABS}|,|\text{EPSREL}|\times|I|\},$$

and EPSABS and EPSREL are user-specified absolute and relative error tolerances. Moreover it returns the quantity ABSERR which, in normal circumstances satisfies

$$|I-\text{RESULT}| \leq \text{ABSERR} \leq tol.$$

8. Further Comments

If IFAIL $\neq$ 0 on exit, then the user may wish to examine the contents of the array W, which contains the end-points of the subintervals used by D01AUF along with the integral contributions and error estimates over these subintervals.

Specifically, for $i = 1,2,...,n$, let r_i denote the approximation to the value of the integral over the subinterval $[a_i,b_i]$ in the partition of $[a,b]$, and e_i be the corresponding absolute error estimate. Then, $\int_{a_i}^{b_i} f(x)dx \simeq r_i$ and RESULT $= \sum_{i=1}^{n} r_i$. The value of n is returned in IW(1), and the values a_i, b_i, e_i and r_i are stored consecutively in the array W, that is:

a_i = W(i),
b_i = W(n+i),
e_i = W(2n+i) and
r_i = W(3n+i).

9. Example

To compute

$$\int_0^{2\pi} x \sin(30x) \cos x \, dx.$$

9.1. Program Text

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

```
*     D01AUF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          LW, LIW
      PARAMETER        (LW=800,LIW=LW/4)
*     .. Scalars in Common ..
      INTEGER          KOUNT
*     .. Local Scalars ..
      real             A, ABSERR, B, EPSABS, EPSREL, PI, RESULT
      INTEGER          IFAIL, KEY
*     .. Local Arrays ..
      real             W(LW)
      INTEGER          IW(LIW)
*     .. External Functions ..
      real             X01AAF
      EXTERNAL         X01AAF
*     .. External Subroutines ..
      EXTERNAL         D01AUF, FST
*     .. Common blocks ..
      COMMON           /TELNUM/KOUNT
```

```
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01AUF Example Program Results'
      PI = X01AAF(0.0e0)
      EPSABS = 0.0e0
      EPSREL = 1.0e-03
      A = 0.0e0
      B = 2.0e0*PI
      KEY = 6
      KOUNT = 0
      IFAIL = -1
*
      CALL D01AUF(FST,A,B,KEY,EPSABS,EPSREL,RESULT,ABSERR,W,LW,IW,LIW,
     +            IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'A      - lower limit of integration = ', A
      WRITE (NOUT,99999) 'B      - upper limit of integration = ', B
      WRITE (NOUT,99998) 'EPSABS - absolute accuracy requested = ',
     +  EPSABS
      WRITE (NOUT,99998) 'EPSREL - relative accuracy requested = ',
     +  EPSREL
      WRITE (NOUT,*)
      IF (IFAIL.NE.0) WRITE (NOUT,99996) 'IFAIL = ', IFAIL
      IF (IFAIL.LE.3) THEN
         WRITE (NOUT,99997) 'RESULT - approximation to the integral = ',
     +     RESULT
         WRITE (NOUT,99998) 'ABSERR - estimate of the absolute error = '
     +     , ABSERR
         WRITE (NOUT,99996) 'KOUNT  - number of function evaluations = '
     +     , KOUNT
         WRITE (NOUT,99996) 'IW(1)  - number of subintervals used = ',
     +     IW(1)
      END IF
      STOP
*
99999 FORMAT (1X,A,F10.4)
99998 FORMAT (1X,A,e9.2)
99997 FORMAT (1X,A,F9.5)
99996 FORMAT (1X,A,I4)
      END
*
      SUBROUTINE FST(X,FV,N)
*     .. Scalar Arguments ..
      INTEGER         N
*     .. Array Arguments ..
      real            FV(N), X(N)
*     .. Scalars in Common ..
      INTEGER         KOUNT
*     .. Local Scalars ..
      INTEGER         I
*     .. Intrinsic Functions ..
      INTRINSIC       COS, SIN
*     .. Common blocks ..
      COMMON          /TELNUM/KOUNT
*     .. Executable Statements ..
      KOUNT = KOUNT + N
      DO 20 I = 1, N
         FV(I) = X(I)*(SIN(30.0e0*X(I)))*COS(X(I))
   20 CONTINUE
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D01AUF Example Program Results

A      - lower limit of integration =     0.0000
B      - upper limit of integration =     6.2832
EPSABS - absolute accuracy requested =  0.00E+00
EPSREL - relative accuracy requested =  0.10E-02

RESULT - approximation to the integral =  -0.20967
ABSERR - estimate of the absolute error =  0.45E-13
KOUNT  - number of function evaluations =  427
IW(1)  - number of subintervals used =    4
```

D01BAF – NAG Fortran Library Routine Document

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

1. Purpose

D01BAF computes an estimate of the definite integral of a function of known analytical form, using a Gaussian quadrature formula with a specified number of abscissae. Formulae are provided for a finite interval (Gauss-Legendre), a semi-infinite interval (Gauss-Laguerre, Gauss-Rational), and an infinite interval (Gauss-Hermite).

2. Specification

```
real FUNCTION D01BAF (D01XXX, A, B, N, FUN, IFAIL)
INTEGER        N, IFAIL
real           A, B, FUN
EXTERNAL       D01XXX, FUN
```

3. Description

3.1. General

This routine evaluates an estimate of the definite integral of a function $f(x)$, over a finite or infinite range, by n-point Gaussian quadrature (see Davis and Rabinowitz [1], Froberg [2], Ralston [3] or Stroud and Secrest [4]). The integral is approximated by a summation

$$\sum_{i=1}^{n} w_i\, f(x_i)$$

where the w_i are called the weights, and the x_i the abscissae. A selection of values of n is available. (See Section 5.)

3.2. Both Limits Finite

$$\int_a^b f(x)dx$$

The Gauss-Legendre weights and abscissae are used, and the formula is exact for any function of the form:

$$f(x) = \sum_{i=0}^{2n-1} c_i x^i$$

The formula is appropriate for functions which can be well approximated by such a polynomial over $[a,b]$. It is inappropriate for functions with algebraic singularities at one or both ends of the interval, such as $(1+x)^{-1/2}$ on $[-1,1]$.

3.3. One Limit Infinite

$$\int_a^{\infty} f(x)dx \quad \text{or} \quad \int_{-\infty}^{a} f(x)dx$$

Two quadrature formulae are available for these integrals.

(a) The Gauss-Laguerre formula is exact for any function of the form:

$$f(x) = e^{-bx} \sum_{i=0}^{2n-1} c_i x^i$$

This formula is appropriate for functions decaying exponentially at infinity; the parameter b should be chosen if possible to match the decay rate of the function.

(b) The Gauss-Rational formula is exact for any function of the form:

$$f(x) = \sum_{i=2}^{2n+1} \frac{c_i}{(x+b)^i} = \frac{\sum_{i=0}^{2n-1} c_{2n+1-i}(x+b)^i}{(x+b)^{2n+1}}.$$

This formula is likely to be more accurate for functions having only an inverse power rate of decay for large x. Here the choice of a suitable value of b may be more difficult; unfortunately a poor choice of b can make a large difference to the accuracy of the computed integral.

3.4. Both Limits Infinite

$$\int_{-\infty}^{+\infty} f(x)dx$$

The Gauss-Hermite weights and abscissae are used, and the formula is exact for any function of the form:

$$f(x) = e^{-b(x-a)^2} \sum_{i=0}^{2n-1} c_i x^i$$

Again, for general functions not of this exact form, the parameter b should be chosen to match if possible the decay rate at ±∞.

4. References

[1] DAVIS, P.J. and RABINOWITZ, P.
Numerical Integration.
Blaisdell Publishing Company, pp. 33-52, 1967.

[2] FROBERG, C.E.
Introduction to Numerical Analysis.
Addison-Wesley, pp. 181-187, 1965.

[3] RALSTON, A.
A First Course in Numerical Analysis.
McGraw-Hill, pp. 87-90, 1965.

[4] STROUD, A.H. and SECREST, D.
Gaussian Quadrature Formulas.
Prentice-Hall, 1966.

5. Parameters

1: D01XXX – SUBROUTINE, supplied by the NAG Fortran Library. *External Procedure*

The name of the routine indicates the quadrature formula:

D01BAZ, for Gauss-Legendre quadrature on a finite interval;
D01BAY, for Gauss-Rational quadrature on a semi-infinite interval;
D01BAX, for Gauss-Laguerre quadrature on a semi-infinite interval;
D01BAW, for Gauss-Hermite quadrature on an infinite interval.

The name used must be declared as EXTERNAL in the (sub)program from which D01BAF is called.

In certain implementations, to avoid name clashes between single and double precision versions, names of auxiliary routines have been changed, e.g. D01BAX to BAXD01. Please refer to the Users' Note for your implementation.

2: A – ***real.*** *Input*

3: B – ***real.*** *Input*

On entry: the parameters a and b which occur in the integration formulae:

Gauss-Legendre:

a is the lower limit and b is the upper limit of the integral. It is not necessary that $a < b$.

Gauss-Rational:

b must be chosen so as to make the integrand match as closely as possible the exact form given in Section 3.3(b). The range of integration is $[a,\infty)$ if $a + b > 0$, and $(-\infty,a]$ if $a + b < 0$.

Gauss-Laguerre:

b must be chosen so as to make the integrand match as closely as possible the exact form given in Section 3.3(a). The range of integration is $[a,\infty)$ if $b > 0$, and $(-\infty,a]$ is $b < 0$.

Gauss-Hermite:

a and b must be chosen so as to make the integrand match as closely as possible the exact form given in Section 3.4.

Constraints: Gauss-Rational: A + B ≠ 0,
Gauss-Laguerre: B ≠ 0,
Gauss-Hermite: B > 0.

4: N – INTEGER. *Input*

On entry: the number of abscissae to be used, n.

Constraint: N = 1,2,3,4,5,6,8,10,12,14,16,20,24,32,48 or 64.

5: FUN – ***real*** FUNCTION, supplied by the user. *External Procedure*

FUN must return the value of the integrand f at a given point.

Its specification is:

```
real FUNCTION FUN(X)
real          X
```

1: X – ***real.*** *Input*

On entry: the point at which the integrand must be evaluated.

FUN must be declared as EXTERNAL in the (sub)program from which D01BAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

Some points to bear in mind when coding FUN are mentioned in Section 7.

6: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.** To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL = 1

The N-point rule is not among those stored. If the soft fail option is used, the answer is evaluated for the largest valid value of N less than the requested value.

IFAIL = 2

The value of A and/or B is invalid.

Gauss-Rational: A + B = 0.
Gauss-Laguerre: B = 0.
Gauss-Hermite: B ≤ 0.

If the soft fail option is used, the answer is returned as zero.

7. Accuracy

The accuracy depends on the behaviour of the integrand, and on the number of abscissae used. No tests are carried out in the routine to estimate the accuracy of the result. If such an estimate is required, the routine may be called more than once, with a different number of abscissae each time, and the answers compared. It is to be expected that for sufficiently smooth functions a larger number of abscissae will give improved accuracy.

Alternatively, the range of integration may be subdivided, the integral estimated separately for each sub-interval, and the sum of these estimates compared with the estimate over the whole range.

The coding of the function FUN may also have a bearing on the accuracy. For example, if a high-order Gauss-Laguerre formula is used, and the integrand is of the form

$$f(x) = e^{-bx}g(x)$$

it is possible that the exponential term may underflow for some large abscissae. Depending on the machine, this may produce an error, or simply be assumed to be zero. In any case, it would be better to evaluate the expression as:

$$f(x) = \exp(-bx+\ln g(x))$$

Another situation requiring care is exemplified by

$$\int_{-\infty}^{+\infty} e^{-x^2}x^m dx = 0, \qquad m \text{ odd.}$$

The integrand here assumes very large values; for example, for $m = 63$, the peak value exceeds 3×10^{33}. Now, if the machine holds floating-point numbers to an accuracy of k significant decimal digits, we could not expect such terms to cancel in the summation leaving an answer of much less than 10^{33-k} (the weights being of order unity); that is instead of zero, we obtain a rather large answer through rounding error. Fortunately, such situations are characterised by great variability in the answers returned by formulae with different values of n. In general, the user should be aware of the order of magnitude of the integrand, and should judge the answer in that light.

8. Further Comments

The time taken by the routine depends on the complexity of the expression for the integrand and on the number of abscissae required.

9. Example

This example program evaluates the integrals

$$\int_0^1 \frac{4}{1+x^2}\,dx = \pi$$

by Gauss-Legendre quadrature;

$$\int_2^\infty \frac{1}{x^2 \ln x}\,dx = 0.378671$$

by Gauss-Rational quadrature with $b = 0$;

$$\int_2^\infty \frac{e^{-x}}{x}\,dx = 0.048901$$

by Gauss-Laguerre quadrature with $b = 1$; and

$$\int_{-\infty}^{+\infty} e^{-3x^2-4x-1} dx = \int_{-\infty}^{+\infty} e^{-3(x+1)^2} e^{2x+2}\,dx = 1.428167$$

by Gauss-Hermite quadrature with $a = -1$ and $b = 3$.

The formulae with $n = 4,8,16$ are used in each case.

9.1. Program Text

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

```
*     D01BAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             A, ANS, B
      INTEGER          I, IFAIL
*     .. Local Arrays ..
      INTEGER          NSTOR(3)
*     .. External Functions ..
      real             D01BAF, FUN1, FUN2, FUN3, FUN4
      EXTERNAL         D01BAF, FUN1, FUN2, FUN3, FUN4
*     .. External Subroutines ..
      EXTERNAL         D01BAW, D01BAX, D01BAY, D01BAZ
*     .. Data statements ..
      DATA             NSTOR/4, 8, 16/
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01BAF Example Program Results'
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Gauss-Legendre example'
      DO 20 I = 1, 3
         A = 0.0e0
         B = 1.0e0
         IFAIL = 1
*
         ANS = D01BAF(D01BAZ,A,B,NSTOR(I),FUN1,IFAIL)
*
         IF (IFAIL.NE.0) THEN
            WRITE (NOUT,99998) 'IFAIL = ', IFAIL
            WRITE (NOUT,*)
         END IF
         IF (IFAIL.LE.1) WRITE (NOUT,99999) NSTOR(I),
     +       ' Points     Answer = ', ANS
   20 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Gauss-Rational example'
      DO 40 I = 1, 3
         A = 2.0e0
         B = 0.0e0
         IFAIL = 1
*
         ANS = D01BAF(D01BAY,A,B,NSTOR(I),FUN2,IFAIL)
*
         IF (IFAIL.NE.0) THEN
            WRITE (NOUT,99998) 'IFAIL = ', IFAIL
            WRITE (NOUT,*)
         END IF
         IF (IFAIL.LE.1) WRITE (NOUT,99999) NSTOR(I),
     +       ' Points     Answer = ', ANS
   40 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Gauss-Laguerre example'
      DO 60 I = 1, 3
         IFAIL = 1
         A = 2.0e0
         B = 1.0e0
*
         ANS = D01BAF(D01BAX,A,B,NSTOR(I),FUN3,IFAIL)
*
         IF (IFAIL.NE.0) THEN
            WRITE (NOUT,99998) 'IFAIL = ', IFAIL
            WRITE (NOUT,*)
         END IF
         IF (IFAIL.LE.1) WRITE (NOUT,99999) NSTOR(I),
     +       ' Points     Answer = ', ANS
   60 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Gauss-Hermite  example'
      DO 80 I = 1, 3
         A = -1.0e0
         B = 3.0e0
         IFAIL = 1
*
         ANS = D01BAF(D01BAW,A,B,NSTOR(I),FUN4,IFAIL)
*
         IF (IFAIL.NE.0) THEN
            WRITE (NOUT,99998) 'IFAIL = ', IFAIL
            WRITE (NOUT,*)
         END IF
         IF (IFAIL.LE.1) WRITE (NOUT,99999) NSTOR(I),
     +       ' Points     Answer = ', ANS
   80 CONTINUE
      STOP
*
99999 FORMAT (1X,I5,A,F10.5)
99998 FORMAT (1X,A,I2)
      END
*
      real FUNCTION FUN1(X)
*     .. Scalar Arguments ..
      real              X
*     .. Executable Statements ..
      FUN1 = 4.0e0/(1.0e0+X*X)
      RETURN
      END
*
```

```
      real  FUNCTION FUN2(X)
*     .. Scalar Arguments ..
      real                X
*     .. Intrinsic Functions ..
      INTRINSIC           LOG
*     .. Executable Statements ..
      FUN2 = 1.0e0/(X*X*LOG(X))
      RETURN
      END
*
      real  FUNCTION FUN3(X)
*     .. Scalar Arguments ..
      real                X
*     .. Intrinsic Functions ..
      INTRINSIC           EXP
*     .. Executable Statements ..
      FUN3 = EXP(-X)/X
      RETURN
      END
*
      real  FUNCTION FUN4(X)
*     .. Scalar Arguments ..
      real                X
*     .. Intrinsic Functions ..
      INTRINSIC           EXP
*     .. Executable Statements ..
      FUN4 = EXP(-3.0e0*X*X-4.0e0*X-1.0e0)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D01BAF Example Program Results

 Gauss-Legendre example
     4 Points     Answer =    3.14161
     8 Points     Answer =    3.14159
    16 Points     Answer =    3.14159


 Gauss-Rational example
     4 Points     Answer =    0.37910
     8 Points     Answer =    0.37876
    16 Points     Answer =    0.37869


 Gauss-Laguerre example
     4 Points     Answer =    0.04887
     8 Points     Answer =    0.04890
    16 Points     Answer =    0.04890


 Gauss-Hermite  example
     4 Points     Answer =    1.42803
     8 Points     Answer =    1.42817
    16 Points     Answer =    1.42817
```

D01BBF – NAG Fortran Library Routine Document

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

1. Purpose

D01BBF returns the weights and abscissae appropriate to a Gaussian quadrature formula with a specified number of abscissae. The formulae provided are Gauss-Legendre, Gauss-Rational, Gauss-Laguerre and Gauss-Hermite.

2. Specification

```
SUBROUTINE D01BBF (D01XXX, A, B, ITYPE, N, WEIGHT, ABSCIS, IFAIL)
INTEGER          ITYPE, N, IFAIL
real             A, B, WEIGHT(N), ABSCIS(N)
EXTERNAL         D01XXX
```

3. Description

This routine returns the weights and abscissae for use in the Gaussian quadrature of a function $f(x)$. The quadrature takes the form

$$S = \sum_{i=1}^{n} w_i \ f(x_i)$$

where w_i are the weights and x_i are the abscissae (see Davis and Rabinowitz [1], Froberg [2], Ralston [3] or Stroud and Secrest [4]).

Weights and abscissae are available for Gauss-Legendre, Gauss-Rational, Gauss-Laguerre and Gauss-Hermite quadrature, and for a selection of values of n (see Section 5).

(a) Gauss-Legendre Quadrature:

$$S \simeq \int_a^b f(x) \ dx$$

where a and b are finite and it will be exact for any function of the form

$$f(x) = \sum_{i=0}^{2n-1} c_i x^i$$

(b) Gauss-Rational quadrature:

$$S \simeq \int_a^{\infty} f(x) \ dx \quad (a+b>0) \qquad \text{or} \qquad S \simeq \int_{-\infty}^{a} f(x) \ dx \quad (a+b<0)$$

and will be exact for any function of the form

$$f(x) = \sum_{i=2}^{2n+1} \frac{c_i}{(x+b)^i} = \frac{\sum_{i=0}^{2n-1} c_{2n+1-i}(x+b)^i}{(x+b)^{2n+1}}$$

(c) Gauss-Laguerre quadrature, adjusted weights option:

$$S \simeq \int_a^{\infty} f(x) \ dx \quad (b>0) \qquad \text{or} \qquad S \simeq \int_{-\infty}^{a} f(x) \ dx \quad (b<0)$$

and will be exact for any function of the form

$$f(x) = e^{-bx} \sum_{i=0}^{2n-1} c_i x^i$$

(d) Gauss-Hermite quadrature, adjusted weights option:

$$S \simeq \int_{-\infty}^{+\infty} f(x)\ dx$$

and will be exact for any function of the form

$$f(x) = e^{-b(x-a)^2} \sum_{i=0}^{2n-1} c_i x^i \quad (b>0)$$

(e) Gauss-Laguerre quadrature, normal weights option:

$$S \simeq \int_{a}^{\infty} e^{-bx} f(x)\ dx \quad (b>0) \quad \text{or} \quad S \simeq \int_{-\infty}^{a} e^{-bx} f(x)\ dx \quad (b<0)$$

and will be exact for any function of the form

$$f(x) = \sum_{i=0}^{2n-1} c_i x^i$$

(f) Gauss-Hermite quadrature, normal weights option:

$$S \simeq \int_{-\infty}^{+\infty} e^{-b(x-a)^2} f(x)\ dx$$

and will be exact for any function of the form:

$$f(x) = \sum_{i=0}^{2n-1} c_i x^i$$

Note: that the Gauss-Legendre abscissae, with $a = -1$, $b = +1$, are the zeros of the Legendre polynomials; the Gauss-Laguerre abscissae, with $a = 0$, $b = 1$, are the zeros of the Laguerre polynomials; and the Gauss-Hermite abscissae, with $a = 0$, $b = 1$, are the zeros of the Hermite polynomials.

4. References

[1] DAVIS, P.J. and RABINOWITZ, P.
Numerical Integration.
Blaisdell Publishing Company, pp. 33-52, 1967.

[2] FROBERG, C.E.
Introduction to Numerical Analysis.
Addison-Wesley, pp. 181-187, 1965.

[3] RALSTON, A.
A First Course in Numerical Analysis.
McGraw-Hill, pp. 87-90, 1965.

[4] STROUD, A.H. and SECREST, D.
Gaussian Quadrature Formulas.
Prentice-Hall, 1966.

5. Parameters

1: D01XXX – SUBROUTINE, supplied by the NAG Fortran Library. *External Procedure*

The name of the routine indicates the quadrature formula:

D01BAZ, for Gauss-Legendre weights and abscissae;

D01BAY, for Gauss-Rational weights and abscissae;

D01BAX, for Gauss-Laguerre weights and abscissae;

D01BAW, for Gauss-Hermite weights and abscissae.

The name used must be declared as EXTERNAL in the (sub)program from which D01BBF is called.

In certain implementations, to avoid name clashes between single and double precision versions, names of auxiliary routines have been changed, e.g. D01BAX to BAXD01. Please refer to the Users' Note for your implementation.

2: A – ***real.*** *Input*

3: B – ***real.*** *Input*

On entry: the quantities a and b as described in the appropriate subsection of Section 3.

4: ITYPE – INTEGER. *Input*

On entry: indicates the type of weights for Gauss-Laguerre or Gauss-Hermite quadrature (see Section 3):

if ITYPE = 1, adjusted weights will be returned;

if ITYPE = 0, normal weights will be returned.

Constraint: ITYPE = 0 or 1.

For Gauss-Legendre or Gauss-Rational quadrature, this parameter is not used.

5: N – INTEGER. *Input*

On entry: the number of weights and abscissae to be returned, n.

Constraint: N = 1,2,3,4,5,6,8,10,12,14,16,20,24,32,48 or 64.

6: WEIGHT(N) – ***real*** array. *Output*

On exit: the N weights. For Gauss-Laguerre and Gauss-Hermite quadrature, these will be the adjusted weights if ITYPE = 1, and the normal weights if ITYPE = 0.

7: ABSCIS(N) – ***real*** array. *Output*

On exit: the N abscissae.

8: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

The N-point rule is not among those stored. If the soft fail option is used, the weights and abscissae returned will be those for the largest valid value of N less than the requested value, and the excess elements of WEIGHT and ABSCIS (i.e. up to the requested N) will be filled with zeros.

IFAIL = 2

The value of A and/or B is invalid.

Gauss-Rational: A + B = 0
Gauss-Laguerre: B = 0
Gauss-Hermite: B ≤ 0

If the soft fail option is used the weights and abscissae are returned as zero.

IFAIL = 3

Laguerre and Hermite normal weights only: underflow is occurring in evaluating one or more of the normal weights. If the soft fail option is used, the underflowing weights are returned as zero. A smaller value of N must be used; or adjusted weights should be used (ITYPE = 1). In the latter case, take care that underflow does not occur when evaluating the integrand appropriate for adjusted weights.

7. Accuracy

The weights and abscissae are stored for standard values of A and B to full machine accuracy.

8. Further Comments

Timing is negligible.

9. Example

This example program returns the abscissae and (adjusted) weights for the six-point Gauss-Laguerre formula.

9.1. Program Text

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

```
*     D01BBF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N
      PARAMETER        (N=6)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             A, B
      INTEGER          IFAIL, ITYPE, J
*     .. Local Arrays ..
      real             ABSCIS(N), WEIGHT(N)
*     .. External Subroutines ..
      EXTERNAL         D01BAX, D01BBF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01BBF Example Program Results'
      A = 0.0e0
      B = 1.0e0
      ITYPE = 1
      IFAIL = 0
*
      CALL D01BBF(D01BAX,A,B,ITYPE,N,WEIGHT,ABSCIS,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99998) 'Laguerre formula,', N, ' points'
      WRITE (NOUT,*)
      WRITE (NOUT,*) '    Abscissae        Weights'
      WRITE (NOUT,*)
      WRITE (NOUT,99999) (ABSCIS(J),WEIGHT(J),J=1,N)
      STOP
*
99999 FORMAT (1X,2e15.6)
99998 FORMAT (1X,A,I3,A)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D01BBF Example Program Results

Laguerre formula,  6 points

    Abscissae          Weights

   0.222847E+00   0.573536E+00
   0.118893E+01   0.136925E+01
   0.299274E+01   0.226068E+01
   0.577514E+01   0.335052E+01
   0.983747E+01   0.488683E+01
   0.159829E+02   0.784902E+01
```

D01BCF – NAG Fortran Library Routine Document

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

1. Purpose

D01BCF returns the weights (normal or adjusted) and abscissae for a Gaussian integration rule with a specified number of abscissae. Six different types of Gauss rule are allowed.

2. Specification

```
SUBROUTINE D01BCF (ITYPE, A, B, C, D, N, WEIGHT, ABSCIS, IFAIL)
INTEGER        ITYPE, N, IFAIL
real           A, B, C, D, WEIGHT(N), ABSCIS(N)
```

3. Description

This routine returns the weights w_i and abscissae x_i for use in the summation

$$S = \sum_{i=1}^{n} w_i \, f(x_i)$$

which approximates a definite integral (see Davis and Rabinowitz [1], or Stroud and Secrest [2]). The following types are provided:

(a) Gauss-Legendre:

$$S \simeq \int_a^b f(x)dx, \quad \text{exact for} \quad f(x) = P_{2n-1}(x).$$

Constraint: $b > a$.

(b) Gauss-Jacobi:

normal weights:

$$S \simeq \int_a^b (b-x)^c (x-a)^d f(x)dx, \quad \text{exact for} \quad f(x) = P_{2n-1}(x),$$

adjusted weights:

$$S \simeq \int_a^b f(x)dx, \quad \text{exact for} \quad f(x) = (b-x)^c (x-a)^d P_{2n-1}(x).$$

Constraint: $c > -1, d > -1, b > a$.

(c) Gauss-Exponential:

normal weights:

$$S \simeq \int_a^b \left| x-\frac{a+b}{2} \right|^c f(x)dx, \quad \text{exact for} \quad f(x) = P_{2n-1}(x),$$

adjusted weights:

$$S \simeq \int_a^b f(x)dx, \quad \text{exact for} \quad f(x) = \left| x-\frac{a+b}{2} \right|^c P_{2n-1}(x).$$

Constraint: $c > -1, b > a$.

(d) Gauss-Laguerre:

normal weights:

$$S \simeq \int_a^{\infty} |x-a|^c e^{-bx} f(x)dx \quad (b>0),$$

$$\simeq \int_{-\infty}^{a} |x-a|^c e^{-bx} f(x)dx \quad (b<0), \quad \text{exact for } f(x) = P_{2n-1}(x),$$

adjusted weights:

$$S \simeq \int_a^{\infty} f(x)dx \quad (b>0),$$

$$\simeq \int_{-\infty}^{a} f(x)dx \quad (b<0), \quad \text{exact for } f(x) = |x-a|^c e^{-bx} P_{2n-1}(x).$$

Constraint: $c > -1, b \neq 0$.

(e) Gauss-Hermite:

normal weights:

$$S \simeq \int_{-\infty}^{+\infty} |x-a|^c \ e^{-b(x-a)^2} f(x)dx, \quad \text{exact for } f(x) = P_{2n-1}(x),$$

adjusted weights:

$$S \simeq \int_{-\infty}^{+\infty} f(x)dx, \quad \text{exact for } f(x) = |x-a|^c \ e^{-b(x-a)^2} P_{2n-1}(x).$$

Constraint: $c > -1, b > 0$.

(f) Gauss-Rational:

normal weights:

$$S \simeq \int_a^{\infty} \frac{|x-a|^c}{|x+b|^d} f(x)dx \quad (a+b>0),$$

$$\simeq \int_{-\infty}^{a} \frac{|x-a|^c}{|x+b|^d} f(x)dx \quad (a+b<0), \quad \text{exact for } f(x) = P_{2n-1}\left(\frac{1}{x+b}\right),$$

adjusted weights:

$$S \simeq \int_a^{\infty} f(x)dx \quad (a+b>0),$$

$$\simeq \int_{-\infty}^{a} f(x)dx \quad (a+b<0), \quad \text{exact for } f(x) = \frac{|x-a|^c}{|x+b|^d} P_{2n-1}\left(\frac{1}{x+b}\right).$$

Constraint: $c > -1, d > c + 1, a + b \neq 0$.

In the above formulae, $P_{2n-1}(x)$ stands for any polynomial of degree $2n - 1$ or less in x.

The method used to calculate the abscissae involves finding the eigenvalues of the appropriate tridiagonal matrix (see Golub and Welsch [3]). The weights are then determined by the formula:

$$w_i = \left\{ \sum_{j=0}^{n-1} P_j^*(x_i)^2 \right\}^{-1}$$

where $P_j^*(x)$ is the jth orthogonal polynomial with respect to the weight function over the appropriate interval.

The weights and abscissae produced by D01BCF may be passed to D01FBF, which will evaluate the summations in one or more dimensions.

4. References

[1] DAVIS, P.J. and RABINOWITZ, P.
Methods of Numerical Integration.
Academic Press, pp. 73-105, 1975.

[2] STROUD, A.H. and SECREST, D.
Gaussian Quadrature Formulas.
Prentice-Hall, 1966.

[3] GOLUB, G.H. and WELSCH, J.H.
Calculation of Gauss Quadrature Rules.
Math. Comput, 23, pp. 221-230, 1969.

5. Parameters

1: ITYPE – INTEGER. *Input*

On entry: indicates the type of quadrature rule.

ITYPE	= 0	Gauss-Legendre
	= 1	Gauss-Jacobi
	= 2	Gauss-Exponential
	= 3	Gauss-Laguerre
	= 4	Gauss-Hermite
	= 5	Gauss-Rational

The above values give the normal weights; the adjusted weights are obtained if the value of ITYPE above is negated.

Constraint: $-5 \le$ ITYPE ≤ 5.

2: A – ***real***. *Input*
3: B – ***real***. *Input*
4: C – ***real***. *Input*
5: D – ***real***. *Input*

On entry: the parameters a, b, c and d which occur in the quadrature formulae. C is not used if ITYPE = 0; D is not used unless ITYPE = ±1 or ±5. For some rules C and D must not be too large (See Section 6).

Constraints: if ITYPE = 0, A < B
if ITYPE = ±1, A < B, C > –1 and D > –1
if ITYPE = ±2, A < B, and C > –1
if ITYPE = ±3, B ≠ 0, and C > –1
if ITYPE = ±4, B > 0, and C > –1
if ITYPE = ±5, A + B ≠ 0, C > –1 and D > C + 1.

6: N – INTEGER. *Input*

On entry: the number of weights and abscissae to be returned, n. If ITYPE = –2 or –4 and C ≠ 0.0, an odd value of N may raise problems – see Section 6, IFAIL = 6.

Constraint: N > 0.

7: WEIGHT(N) – ***real*** array. *Output*

On exit: the N weights.

8: ABSCIS(N) – ***real*** array. *Output*

On exit: the N abscissae.

9: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

The algorithm for computing eigenvalues of a tridiagonal matrix has failed to obtain convergence. If the soft fail option is used, the values of the weights and abscissae on return are indeterminate.

IFAIL = 2

On entry, N < 1,
or ITYPE < –5,
or ITYPE > 5.

If the soft fail option is used, weights and abscissae are returned as zero.

IFAIL = 3

A, B, C or D is not in the allowed range:

if ITYPE = 0, A ≥ B

if ITYPE = ±1, A ≥ B or C ≤ –1.0 or D ≤ –1.0 or C + D + 2.0 > GMAX

if ITYPE = ±2, A ≥ B or C ≤ –1.0

if ITYPE = ±3, B = 0.0 or C ≤ –1.0 or C + 1.0 > GMAX

if ITYPE = ±4, B ≤ 0.0 or C ≤ –1.0 or (C+1.0)/2.0 > GMAX

if ITYPE = ±5, A + B = 0.0 or C ≤ –1.0 or D ≤ C + 1.0

Here GMAX is the (machine-dependent) largest integer value such that Γ(GMAX) can be computed without overflow (see the Users' Note for your implementation for S14AAF).

If the soft fail option is used, weights and abscissae are returned as zero.

IFAIL = 4

One or more of the weights are larger than RMAX, the largest floating-point number on this machine. RMAX is given by the function X02ALF. If the soft fail option is used, the overflowing weights are returned as RMAX. Possible solutions are to use a smaller value of N; or, if using adjusted weights, to change to normal weights.

IFAIL = 5

One or more of the weights are too small to be distinguished from zero on this machine. If the soft fail option is used, the underflowing weights are returned as zero, which may be a usable approximation. Possible solutions are to use a smaller value of N; or, if using normal weights, to change to adjusted weights.

IFAIL = 6

Gauss-Exponential or Gauss-Hermite adjusted weights with N odd and C ≠ 0.0. Theoretically, in these cases:

for C > 0.0, the central adjusted weight is infinite, and the exact function $f(x)$ is zero at the central abscissa.

for C < 0.0, the central adjusted weight is zero, and the exact function $f(x)$ is infinite at the central abscissa.

In either case, the contribution of the central abscissa to the summation is indeterminate.

In practice, the central weight may not have overflowed or underflowed, if there is sufficient rounding error in the value of the central abscissa.

If the soft fail option is used, the weights and abscissa returned may be usable; the user must be particularly careful not to 'round' the central abscissa to its true value without simultaneously 'rounding' the central weight to zero or ∞ as appropriate, or the summation will suffer. It would be preferable to use normal weights, if possible.

Note: Remember that, when switching from normal weights to adjusted weights or vice versa, redefinition of $f(x)$ is involved.

7. Accuracy

The accuracy depends mainly on n, with increasing loss of accuracy for larger values of n. Typically, one or two decimal digits may be lost from machine accuracy with $n \simeq 20$, and three or four decimal digits may be lost for $n \simeq 100$.

8. Further Comments

The major portion of the time is taken up during the calculation of the eigenvalues of the appropriate tridiagonal matrix, where the time is roughly proportional to n^3.

9. Example

This example program returns the abscissae and (adjusted) weights for the seven-point Gauss-Laguerre formula.

9.1. Program Text

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

```
*     D01BCF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N
      PARAMETER        (N=7)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             A, B, C, D
      INTEGER          IFAIL, ITYPE, J
*     .. Local Arrays ..
      real             ABSCIS(N), WEIGHT(N)
*     .. External Subroutines ..
      EXTERNAL         D01BCF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01BCF Example Program Results'
      A = 0.0e0
      B = 1.0e0
      C = 0.0e0
      D = 0.0e0
      ITYPE = -3
      IFAIL = 0
*
      CALL D01BCF(ITYPE,A,B,C,D,N,WEIGHT,ABSCIS,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'Laguerre formula,', N, ' points'
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      Abscissae             Weights'
      WRITE (NOUT,*)
      WRITE (NOUT,99998) (ABSCIS(J),WEIGHT(J),J=1,N)
      STOP
*
99999 FORMAT (1X,A,I3,A)
99998 FORMAT (1X,e15.5,5X,e15.5)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D01BCF Example Program Results

 Laguerre formula,  7 points

        Abscissae              Weights

      0.19304E+00          0.49648E+00
      0.10267E+01          0.11776E+01
      0.25679E+01          0.19182E+01
      0.49004E+01          0.27718E+01
      0.81822E+01          0.38412E+01
      0.12734E+02          0.53807E+01
      0.19396E+02          0.84054E+01
```

D01BDF – NAG Fortran Library Routine Document

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

1. Purpose

D01BDF calculates an approximation to the integral of a function over a finite interval $[a,b]$:

$$I = \int_a^b f(x)dx.$$

It is non-adaptive and as such is recommended for the integration of 'smooth' functions. These **exclude** integrands with singularities, derivative singularities or high peaks on $[a,b]$, or which oscillate too strongly on $[a,b]$.

2. Specification

```
SUBROUTINE D01BDF (F, A, B, EPSABS, EPSREL, RESULT, ABSERR)
real           F, A, B, EPSABS, EPSREL, RESULT, ABSERR
EXTERNAL       F
```

3. Description

D01BDF is based on the QUADPACK routine QNG (Piessens *et al.* [2]). It is a non-adaptive routine which uses as its basic rules, the Gauss 10-point and 21-point formulae. If the accuracy criterion is not met, formulae using 43 and 87 points are used successively, stopping whenever the accuracy criterion is satisfied.

This routine is designed for smooth integrands only.

4. References

[1] PATTERSON, T.N.L.
The Optimum Addition of Points to Quadrature Formulae.
Math. Comput., 22, pp. 847-856, 1968.

[2] PIESSENS, R., DE DONCKER, E., ÜBERHUBER, C. and KAHANER, D.
QUADPACK, A Subroutine Package for Automatic Integration.
Springer-Verlag, 1983.

5. Parameters

1: F – ***real*** FUNCTION, supplied by the user. *External Procedure*

F must return the value of the integrand f at a given point.

Its specification is:

```
real FUNCTION F(X)
real          X
```

1: X – ***real***. *Input*

On entry: the point at which the integrand f must be evaluated.

F must be declared as EXTERNAL in the (sub)program from which D01BDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: A – ***real***. *Input*

On entry: the lower limit of integration, a.

3: B – ***real.*** *Input*

On entry: the upper limit of integration, b. It is not necessary that $a < b$.

4: EPSABS – ***real.*** *Input*

On entry: the absolute accuracy required. If EPSABS is negative, the absolute value is used. See Section 7.

5: EPSREL – ***real.*** *Input*

On entry: the relative accuracy required. If EPSREL is negative, the absolute value is used. See Section 7.

6: RESULT – ***real.*** *Output*

On exit: the approximation to the integral I.

7: ABSERR – ***real.*** *Output*

On exit: an estimate of the modulus of the absolute error, which should be an upper bound for $|I-\text{RESULT}|$.

6. Error Indicators and Warnings

There are no specific errors detected by the routine. However, if ABSERR is greater than

$$\max\{\text{EPSABS},\text{EPSREL}\times|\text{RESULT}|\}$$

this indicates that the routine has probably failed to achieve the requested accuracy within 87 function evaluations.

7. Accuracy

The routine attempts to compute an approximation, RESULT, such that:

$$|I-\text{RESULT}| \le tol,$$

where

$$tol = \max\{|\text{EPSABS}|,|\text{EPSREL}|\times|I|\}$$

and EPSABS and EPSREL are user-specified absolute and relative error tolerances. There can be no guarantee that this is achieved, and users are advised to subdivide the interval if they have any doubts about the accuracy obtained. Note that ABSERR contains an estimated bound on $|I-\text{RESULT}|$.

8. Further Comments

The time taken by the routine depends on the integrand and on the accuracy required.

9. Example

To compute

$$\int_0^1 x^2 \sin(10\pi x)dx.$$

9.1. Program Text

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

```
*     D01BDF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
```

```
*     .. Scalars in Common ..
      real             PI
      INTEGER          KOUNT
*     .. Local Scalars ..
      real             A, ABSERR, B, EPSABS, EPSREL, RESULT
*     .. External Functions ..
      real             FST, X01AAF
      EXTERNAL         FST, X01AAF
*     .. External Subroutines ..
      EXTERNAL         D01BDF
*     .. Intrinsic Functions ..
      INTRINSIC        ABS, MAX
*     .. Common blocks ..
      COMMON          /TELNUM/PI, KOUNT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01BDF Example Program Results'
      PI = X01AAF(0.0e0)
      EPSABS = 0.0e0
      EPSREL = 1.0e-04
      A = 0.0e0
      B = 1.0e0
      KOUNT = 0
*
      CALL D01BDF(FST,A,B,EPSABS,EPSREL,RESULT,ABSERR)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'A      - lower limit of integration = ', A
      WRITE (NOUT,99999) 'B      - upper limit of integration = ', B
      WRITE (NOUT,99998) 'EPSABS - absolute accuracy requested = ',
     +  EPSABS
      WRITE (NOUT,99998) 'EPSREL - relative accuracy requested = ',
     +  EPSREL
      WRITE (NOUT,*)
      WRITE (NOUT,99997) 'RESULT - approximation to the integral = ',
     +  RESULT
      WRITE (NOUT,99998) 'ABSERR - estimate to the absolute error = ',
     +  ABSERR
      WRITE (NOUT,99996) 'KOUNT  - number of function evaluations = ',
     +  KOUNT
      WRITE (NOUT,*)
      IF (KOUNT.GT.87 .OR. ABSERR.GT.MAX(EPSABS,EPSREL*ABS(RESULT)))
     +    THEN
         WRITE (NOUT,*)
     +      'Warning - requested accuracy may not have been achieved'
      END IF
      STOP
*
99999 FORMAT (1X,A,F10.4)
99998 FORMAT (1X,A,e9.2)
99997 FORMAT (1X,A,F9.5)
99996 FORMAT (1X,A,I4)
      END
*
      real FUNCTION FST(X)
*     .. Scalar Arguments ..
      real              X
*     .. Scalars in Common ..
      real              PI
      INTEGER           KOUNT
*     .. Intrinsic Functions ..
      INTRINSIC         SIN
*     .. Common blocks ..
      COMMON           /TELNUM/PI, KOUNT
*     .. Executable Statements ..
      KOUNT = KOUNT + 1
      FST = (X**2)*SIN(10.0e0*PI*X)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D01BDF Example Program Results

A      - lower limit of integration =     0.0000
B      - upper limit of integration =     1.0000
EPSABS - absolute accuracy requested =  0.00E+00
EPSREL - relative accuracy requested =  0.10E-03

RESULT - approximation to the integral =  -0.03183
ABSERR - estimate to the absolute error =  0.13E-10
KOUNT  - number of function evaluations =   43
```

D01DAF – NAG Fortran Library Routine Document

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

1. Purpose

D01DAF attempts to evaluate a double integral to a specified absolute accuracy by repeated applications of the method described by Patterson.

2. Specification

```
SUBROUTINE D01DAF (YA, YB, PHI1, PHI2, F, ABSACC, ANS, NPTS, IFAIL)
INTEGER          NPTS, IFAIL
real             YA, YB, PHI1, PHI2, F, ABSACC, ANS
EXTERNAL         PHI1, PHI2, F
```

3. Description

This routine attempts to evaluate a definite integral of the form

$$I = \int_a^b \int_{\phi_1(y)}^{\phi_2(y)} f(x,y)dx\ dy$$

where a and b are constants and $\phi_1(y)$ and $\phi_2(y)$ are functions of the variable y.

The integral is evaluated by expressing it as

$$I = \int_a^b F(y)dy, \quad \text{where } F(y) = \int_{\phi_1(y)}^{\phi_2(y)} f(x,y)dx.$$

Both the outer integral I and the inner integrals $F(y)$ are evaluated by the method, described by Patterson [1] and [2], of the optimum addition of points to Gauss quadrature formulae.

This method uses a family of interlacing common point formulae. Beginning with the three-point Gauss rule, formulae using 7, 15, 31, 63, 127 and finally 255 points are derived. Each new formula contains all the pivots of the earlier formulae so that no function evaluations are wasted. Each integral is evaluated by applying these formulae successively until two results are obtained which differ by less than the specified absolute accuracy.

4. References

[1] PATTERSON, T.N.L.
The optimum addition of points to quadrature formulae.
Math. Comp., 22, pp. 847-856, 1968.
Errata, Math. Comp., 23, p. 892, 1969.

[2] PATTERSON, T.N.L.
On some Gauss and Lobatto based integration formulae.
Math. Comp., 22, pp. 877-881, 1968.

5. Parameters

1: YA – ***real.*** *Input*

On entry: the lower limit of the integral, a.

2: YB – ***real.*** *Input*

On entry: the upper limit of the integral, b. It is not necessary that $a < b$.

3: PHI1 – ***real*** FUNCTION, supplied by the user. *External Procedure*

PHI1 must return the lower limit of the inner integral for a given value of y.

Its specification is:

```
real FUNCTION PHI1(Y)
real          Y
```

1: Y – ***real.*** *Input*

On entry: the value of y for which the lower limit must be evaluated.

PHI1 must be declared as EXTERNAL in the (sub)program from which D01DAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

4: PHI2 – ***real*** FUNCTION, supplied by the user. *External Procedure*

PHI2 must return the upper limit of the inner integral for a given value of y.

Its specification is:

```
real FUNCTION PHI2(Y)
real          Y
```

1: Y – ***real.*** *Input*

On entry: the value of y for which the upper limit must be evaluated.

PHI2 must be declared as EXTERNAL in the (sub)program from which D01DAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: F – ***real*** FUNCTION, supplied by the user. *External Procedure*

F must return the value of the integrand f at a given point.

Its specification is:

```
real FUNCTION F(X, Y)
real          X, Y
```

1: X – ***real.*** *Input*

2: Y – ***real.*** *Input*

On entry: the co-ordinates of the point (x,y) at which the integrand must be evaluated.

F must be declared as EXTERNAL in the (sub)program from which D01DAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: ABSACC – ***real.*** *Input*

On entry: the absolute accuracy requested.

7: ANS – ***real.*** *Output*

On exit: the estimate of the integral.

8: NPTS – INTEGER. *Output*

On exit: the total number of function evaluations.

9: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq$ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.** To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL = 1

This indicates that 255 points have been used in the outer integral and convergence has not been obtained. All the inner integrals have, however, converged. In this case ANS may still contain an approximate estimate of the integral.

IFAIL = 10×N

This indicates that the outer integral has converged but N inner integrals have failed to converge with the use of 255 points. In this case ANS may still contain an approximate estimate of the integral, but its reliability will decrease as N increases.

IFAIL = 10×N + 1

This indicates that both the outer integral and N of the inner integrals have not converged. ANS may still contain an approximate estimate of the integral, but its reliability will decrease as N increases.

7. Accuracy

The absolute accuracy is specified by the variable ABSACC. If, on exit, IFAIL = 0 then the result is most likely correct to this accuracy. Even if IFAIL is non-zero on exit, it is still possible that the calculated result could differ from the true value by less than the given accuracy.

8. Further Comments

The time taken by the routine depends upon the complexity of the integrand and the accuracy requested.

With Patterson's method accidental convergence may occasionally occur, when two estimates of an integral agree to within the requested accuracy, but both estimates differ considerably from the true result. This could occur in either the outer integral or in one or more of the inner integrals.

If it occurs in the outer integral then apparent convergence is likely to be obtained with considerably fewer integrand evaluations than may be expected. If it occurs in an inner integral, the incorrect value could make the function $F(y)$ appear to be badly behaved, in which case a very large number of pivots may be needed for the overall evaluation of the integral. Thus both unexpectedly small and unexpectedly large numbers of integrand evaluations should be considered as indicating possible trouble. If accidental convergence is suspected, the integral may be recomputed, requesting better accuracy; if the new request is more stringent than the degree of accidental agreement (which is of course unknown), improved results should be obtained. This is only possible when the accidental agreement is not better than machine accuracy. It should be noted that the routine requests the same accuracy for the inner integrals as for the outer integral. In practice it has been found that in the vast majority of cases this has proved to be adequate for the overall result of the double integral to be accurate to within the specified value.

The routine is not well-suited to non-smooth integrands, i.e. integrands having some kind of analytic discontinuity (such as a discontinuous or infinite partial derivative of some low order) in, on the boundary of, or near, the region of integration. **Warning**: such singularities may be induced by incautiously presenting an apparently smooth interval over the positive quadrant of the unit circle, R

$$I = \int_R (x+y)dxdy.$$

This may be presented to D01DAF as

$$I = \int_0^1 dy \int_0^{\sqrt{1-y^2}} (x+y)dx = \int_0^1 \left(\tfrac{1}{2}(1-y^2)+y\sqrt{1-y^2}\right)dy$$

but here the outer integral has an induced square-root singularity stemming from the way the region has been presented to D01DAF. This situation should be avoided by re-casting the problem. For the example given, the use of polar co-ordinates would avoid the difficulty:

$$I = \int_0^1 dr \int_0^{\frac{\pi}{2}} r^2(\cos v + \sin v)dv.$$

9. Example

The following program evaluates the integral discussed in Section 8, presenting it to D01DAF first as

$$\int_0^1 \int_0^{\sqrt{1-y^2}} (x+y)dxdy$$

and then as

$$\int_0^1 \int_0^{\frac{\pi}{2}} r^2(\cos v + \sin v)dvdr.$$

Note the difference in the number of function evaluations.

9.1. Program Text

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

```
*     D01DAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             ABSACC, ANS, YA, YB
      INTEGER          IFAIL, NPTS
*     .. External Functions ..
      real             FA, FB, P1, P2A, P2B
      EXTERNAL         FA, FB, P1, P2A, P2B
*     .. External Subroutines ..
      EXTERNAL         D01DAF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01DAF Example Program Results'
      YA = 0.0e0
      YB = 1.0e0
      ABSACC = 1.0e-6
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'First formulation'
      IFAIL = 1
*
```

```
      CALL D01DAF(YA,YB,P1,P2A,FA,ABSACC,ANS,NPTS,IFAIL)
*
      WRITE (NOUT,99999) 'Integral =', ANS
      WRITE (NOUT,99998) 'Number of function evaluations =', NPTS
      IF (IFAIL.GT.0) WRITE (NOUT,99997) 'IFAIL = ', IFAIL
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Second formulation'
      IFAIL = 1
*
      CALL D01DAF(YA,YB,P1,P2B,FB,ABSACC,ANS,NPTS,IFAIL)
*
      WRITE (NOUT,99999) 'Integral =', ANS
      WRITE (NOUT,99998) 'Number of function evaluations =', NPTS
      IF (IFAIL.GT.0) WRITE (NOUT,99997) 'IFAIL = ', IFAIL
      STOP
*
99999 FORMAT (1X,A,F9.4)
99998 FORMAT (1X,A,I5)
99997 FORMAT (1X,A,I2)
      END
*
      real FUNCTION P1(Y)
*     .. Scalar Arguments ..
      real              Y
*     .. Executable Statements ..
      P1 = 0.0e0
      RETURN
      END
*
      real FUNCTION P2A(Y)
*     .. Scalar Arguments ..
      real               Y
*     .. Intrinsic Functions ..
      INTRINSIC         SQRT
*     .. Executable Statements ..
      P2A = SQRT(1.0e0-Y*Y)
      RETURN
      END
*
      real FUNCTION FA(X,Y)
*     .. Scalar Arguments ..
      real             X, Y
*     .. Executable Statements ..
      FA = X + Y
      RETURN
      END
*
      real FUNCTION P2B(Y)
*     .. Scalar Arguments ..
      real               Y
*     .. External Functions ..
      real              X01AAF
      EXTERNAL          X01AAF
*     .. Executable Statements ..
      P2B = 0.5e0*X01AAF(0.0e0)
      RETURN
      END
*
      real FUNCTION FB(X,Y)
*     .. Scalar Arguments ..
      real             X, Y
*     .. Intrinsic Functions ..
      INTRINSIC        COS, SIN
*     .. Executable Statements ..
      FB = Y*Y*(COS(X)+SIN(X))
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D01DAF Example Program Results

First formulation
Integral =   0.6667
Number of function evaluations =  189

Second formulation
Integral =   0.6667
Number of function evaluations =   89
```

D01EAF – NAG Fortran Library Routine Document

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

1. Purpose

D01EAF computes approximations to the integrals of a vector of similar functions, each defined over the same multi-dimensional hyper-rectangular region. The routine uses an adaptive subdivision strategy, and also computes absolute error estimates.

2. Specification

```
      SUBROUTINE D01EAF (NDIM, A, B, MINCLS, MAXCLS, NFUN, FUNSUB, ABSREQ,
     1                   RELREQ, LENWRK, WRKSTR, FINEST, ABSEST, IFAIL)
      INTEGER          NDIM, MINCLS, MAXCLS, NFUN, LENWRK, IFAIL
      real             A(NDIM), B(NDIM), ABSREQ, RELREQ, WRKSTR(LENWRK),
     1                 FINEST(NFUN), ABSEST(NFUN)
      EXTERNAL         FUNSUB
```

3. Description

The subroutine uses a globally adaptive method based on the algorithm described by van Dooren and de Ridder [1] and Genz and Malik [2]. It is implemented for integrals in the form:

$$\int_{a_1}^{b_1}\int_{a_2}^{b_2}\cdots\int_{a_n}^{b_n}(f_1,f_2,\ldots,f_m)\;dx_n\;\ldots\;dx_2dx_1,$$

where $f_i = f_i(x_1,x_2,\ldots,x_n)$, for $i = 1,2,\ldots,m$.

Upon entry, unless MINCLS has been set to a value less than or equal to 0, the subroutine divides the integration region into a number of subregions with randomly selected volumes. Inside each subregion the integrals and their errors are estimated. The initial number of subregions is chosen to be as large as possible without using more than MINCLS calls to FUNSUB. The results are stored in a partially ordered list (a heap). The routine then proceeds in stages. At each stage the subregion with the largest error (measured using the maximum norm) is halved along the co-ordinate axis where the integrands have largest absolute fourth differences. The basic rule is applied to each half of this subregion and the results are stored in the list. The results from the two halves are used to update the global integral and error estimates (FINEST and ABSEST) and the routine continues unless $\|\mathrm{ABSEST}\| \le \max(\mathrm{ABSREQ},\|\mathrm{FINEST}\|\times\mathrm{RELREQ})$ where the norm $\|.\|$ is the maximum norm, or further subdivision would use more than MAXCLS calls to FUNSUB. If at some stage there is insufficient working storage to keep the results for the next subdivision, the routine switches to a less efficient mode; only if this mode of operation breaks down is insufficient storage reported.

4. References

[1] VAN DOOREN, P. and DE RIDDER, L.
An Adaptive Algorithm for Numerical Integration over an N-dimensional Cube.
J. Comput. Appl. Math., Vol. 2, pp. 207-217, 1976.

[2] GENZ, A.C. and MALIK, A.A.
An Adaptive Algorithm for Numerical Integration over an N-dimensional Rectangular Region.
J. Comput. Appl. Math., Vol. 6, pp. 295-302, 1980.

5. Parameters

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integrals, n.

Constraint: NDIM $\geq$ 1.

2: A(NDIM) – ***real*** array. *Input*

On entry: the lower limits of integration, a_i, for $i = 1,2,...,n$.

3: B(NDIM) – ***real*** array. *Input*

On entry: the upper limits of integration, b_i, for $i = 1,2,...,n$.

4: MINCLS – INTEGER. *Input/Output*

On entry: MINCLS must be set:

either to the minimum number of FUNSUB calls to be allowed, in which case MINCLS $\geq$ 0;

or to a negative value. In this case, the routine continues the calculation started in a previous call with the same integrands and integration limits: no parameters other than MINCLS, MAXCLS, ABSREQ, RELREQ or IFAIL must be changed between the calls.

On exit: MINCLS gives the number of FUNSUB calls actually used by D01EAF. For the continuation case (MINCLS < 0 on entry) this is the number of new FUNSUB calls on the current call to D01EAF.

5: MAXCLS – INTEGER. *Input*

On entry: the maximum number of FUNSUB calls to be allowed. In the continuation case this is the number of new FUNSUB calls to be allowed.

Constraints: MAXCLS $\geq$ MINCLS
MAXCLS $\geq r$,
where $r = 2^n + 2n^2 + 2n + 1$, if $n < 11$,
or $r = 1 + n(4n^2-6n+14)/3$, if $n \geq 11$.

6: NFUN – INTEGER. *Input*

On entry: the number of integrands, m.

Constraint: NFUN $\geq$ 1.

7: FUNSUB – SUBROUTINE, supplied by the user. *External Procedure*

FUNSUB must evaluate the integrands f_i at a given point.

Its specification is:

```
SUBROUTINE FUNSUB(NDIM, Z, NFUN, F)
INTEGER    NDIM, NFUN
real       Z(NDIM), F(NFUN)
```

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integrals, n.

2: Z(NDIM) – ***real*** array. *Input*

On entry: the co-ordinates of the point at which the integrands must be evaluated.

3: NFUN – INTEGER. *Input*

On entry: the number of integrands, m.

4: F(NFUN) – ***real*** array. *Output*

On exit: the value of the *i*th integrand at the given point.

FUNSUB must be declared as EXTERNAL in the (sub)program from which D01EAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

8: ABSREQ – ***real***. *Input*

On entry: the absolute accuracy required by the user.

Constraint: ABSREQ ≥ 0.0.

9: RELREQ – ***real***. *Input*

On entry: the relative accuracy required by the user.

Constraint: RELREQ ≥ 0.0.

10: LENWRK – INTEGER. *Input*

On entry: the dimension of the array WRKSTR as declared in the (sub)program from which D01EAF is called.

Suggested value: LENWRK $\geq 6n + 9m + (n{+}m{+}2)(1{+}p/r)$, where p is the value of MAXCLS and r is defined under MAXCLS. If LENWRK is significantly smaller than this, the routine will not work as efficiently and may even fail.

Constraint: LENWRK ≥ 8×NDIM + 11×NFUN + 3.

11: WRKSTR(LENWRK) – ***real*** array. *Input/Output*

On entry: if MINCLS < 0, WRKSTR must be unchanged from the previous call of D01EAF.

On exit: WRKSTR contains information about the current subdivision which could be used in a continuation call.

12: FINEST(NFUN) – ***real*** array. *Output*

On exit: FINEST(i) specifies the best estimate obtained from the ith integral, for $i = 1,2,...,m$.

13: ABSEST(NFUN) – ***real*** array. *Output*

On exit: ABSEST(i) specifies the estimated absolute accuracy of FINEST(i), for $i = 1,2,...,m$.

14: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to −1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL = 1

MAXCLS was too small for D01EAF to obtain the required accuracy. The arrays FINEST and ABSEST respectively contain current estimates for the integrals and errors.

IFAIL = 2

LENWRK is too small for the routine to continue. The arrays FINEST and ABSEST respectively contain current estimates for the integrals and errors.

IFAIL = 3

On a continuation call, MAXCLS was set too small to make any progress. Increase MAXCLS before calling D01EAF again.

IFAIL = 4

On entry, NDIM < 1,
or NFUN < 1,
or MAXCLS < MINCLS,
or MAXCLS < r (see MAXCLS),
or ABSREQ < 0.0,
or RELREQ < 0.0,
or LENWRK < 8×NDIM + 11×NFUN + 3.

7. Accuracy

An absolute error estimate for each integrand is output in the array ABSEST. The routine exits with IFAIL = 0 if

$$\max_i (\mathrm{ABSEST}(i)) \le \max(\mathrm{ABSREQ}, \mathrm{RELREQ} \times \max_i |\mathrm{FINEST}(i)|).$$

8. Further Comments

Usually the running time for D01EAF will be dominated by the time in the user-supplied subroutine FUNSUB, so the maximum time that could be used by D01EAF will be proportional to MAXCLS multiplied by the cost of a call to FUNSUB.

On a normal call, the user should set MINCLS = 0 on entry.

For some integrands, particularly those that are poorly behaved in a small part of the integration region, D01EAF may terminate prematurely with values of ABSEST that are significantly smaller than the actual absolute errors. This behaviour should be suspected if the returned value of MINCLS is small relative to the expected difficulty of the integrals. When this occurs D01EAF should be called again, but with an entry value of MINCLS ≥ $2r$, (see specification of MAXCLS) and the results compared with those from the previous call.

If the routine is called with MINCLS ≥ $2r$, the exact values of FINEST and ABSEST on return will depend (within statistical limits) on the sequence of random numbers generated internally within D01EAF by calls to G05CAF. Separate runs will produce identical answers unless the part of the program executed prior to calling D01EAF also calls (directly or indirectly) routines from the G05 chapter, and, in addition, the series of such calls differs between runs.

Because of moderate instability in the application of the basic integration rule, approximately the last $1 + \log_{10}(n^3)$ decimal digits may be inaccurate when using D01EAF for large values of n.

9. Example

To compute

$$\int_0^1 \int_0^1 \int_0^1 \int_0^1 (f_1, f_2, \ldots, f_{10}) \; dx_4 dx_3 dx_2 dx_1,$$

where, for $j = 1,2,\ldots,10$ $f_j = \ln(x_1+2x_2+3x_3+4x_4)\sin(j+x_1+2x_2+3x_3+4x_4)$. The program is intended to show how to exploit the continuation facility provided with D01EAF: the routine exits with IFAIL = 1 (printing an explanatory error message) and is re-entered with MAXCLS reset to a larger value. The program can be used with any values of NDIM and NFUN, except that the expression for IRCLS must be changed if NDIM > 10 (see specification of MAXCLS).

9.1. Program Text

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

```
*     D01EAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NDIM, NFUN, IRCLS, MXCLS, LENWRK
      PARAMETER        (NDIM=4,NFUN=10,
     +                 IRCLS=2**NDIM+2*NDIM*NDIM+2*NDIM+1,MXCLS=IRCLS,
     +                 LENWRK=6*NDIM+9*NFUN+(NDIM+NFUN+2)
     +                 *(1+MXCLS/IRCLS))
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             ABSREQ, RELREQ
      INTEGER          I, IFAIL, MAXCLS, MINCLS, MULFAC, N
*     .. Local Arrays ..
      real             A(NDIM), ABSEST(NFUN), B(NDIM), FINEST(NFUN),
     +                 WRKSTR(LENWRK)
*     .. External Subroutines ..
      EXTERNAL         D01EAF, FUNSUB
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01EAF Example Program Results'
      DO 20 N = 1, NDIM
         A(N) = 0.0e0
         B(N) = 1.0e0
   20 CONTINUE
      MINCLS = 0
      MAXCLS = MXCLS
      ABSREQ = 0.0e0
      RELREQ = 1.0e-3
      IF (NDIM.LE.10) THEN
         MULFAC = 2**NDIM
      ELSE
         MULFAC = 2*NDIM**3
      END IF
   40 IFAIL = -1
*
      CALL D01EAF(NDIM,A,B,MINCLS,MAXCLS,NFUN,FUNSUB,ABSREQ,RELREQ,
     +            LENWRK,WRKSTR,FINEST,ABSEST,IFAIL)
*
      WRITE (NOUT,*)
      IF (IFAIL.GT.0) THEN
         IF (IFAIL.EQ.1 .OR. IFAIL.EQ.3) THEN
            WRITE (NOUT,99999) 'Results so far (', MINCLS,
     +        ' FUNSUB calls in last call of D01EAF)'
            WRITE (NOUT,*)
            WRITE (NOUT,*) '    I       Integral   Estimated error'
            DO 60 I = 1, NFUN
               WRITE (NOUT,99998) I, FINEST(I), ABSEST(I)
   60       CONTINUE
            WRITE (NOUT,*)
            MINCLS = -1
            MAXCLS = MAXCLS*MULFAC
            GO TO 40
         END IF
      ELSE
         WRITE (NOUT,99999) 'Final results (', MINCLS,
     +     ' FUNSUB calls in last call of D01EAF)'
         WRITE (NOUT,*)
         WRITE (NOUT,*) '    I       Integral   Estimated error'
         DO 80 I = 1, NFUN
            WRITE (NOUT,99998) I, FINEST(I), ABSEST(I)
   80    CONTINUE
      END IF
      STOP
*
```

```
99999 FORMAT (1X,A,I7,A)
99998 FORMAT (1X,I4,2F14.4)
      END
*
      SUBROUTINE FUNSUB(NDIM,Z,NFUN,F)
*     .. Scalar Arguments ..
      INTEGER           NDIM, NFUN
*     .. Array Arguments ..
      real              F(NFUN), Z(NDIM)
*     .. Local Scalars ..
      real              SUM
      INTEGER           I, N
*     .. Intrinsic Functions ..
      INTRINSIC         LOG, real, SIN
*     .. Executable Statements ..
      SUM = 0.0e0
      DO 20 N = 1, NDIM
         SUM = SUM + real(N)*Z(N)
   20 CONTINUE
      DO 40 I = 1, NFUN
         F(I) = LOG(SUM)*SIN(real(I)+SUM)
   40 CONTINUE
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D01EAF Example Program Results
 ** MAXCLS too small to obtain required accuracy
 ** ABNORMAL EXIT from NAG Library routine D01EAF: IFAIL =     1
 ** NAG soft failure - control returned

 Results so far (     57 FUNSUB calls in last call of D01EAF)

    I       Integral   Estimated error
    1        0.0422        0.0086
    2        0.3998        0.0038
    3        0.3898        0.0127
    4        0.0214        0.0099
    5       -0.3666        0.0020
    6       -0.4176        0.0120
    7       -0.0846        0.0110
    8        0.3261        0.0001
    9        0.4371        0.0112
   10        0.1461        0.0119

 ** MAXCLS too small to obtain required accuracy
 ** ABNORMAL EXIT from NAG Library routine D01EAF: IFAIL =     1
 ** NAG soft failure - control returned

 Results so far (    798 FUNSUB calls in last call of D01EAF)

    I       Integral   Estimated error
    1        0.0384        0.0006
    2        0.4012        0.0006
    3        0.3952        0.0006
    4        0.0258        0.0006
    5       -0.3673        0.0006
    6       -0.4227        0.0006
    7       -0.0895        0.0006
    8        0.3260        0.0006
    9        0.4417        0.0006
   10        0.1514        0.0006
```

```
Final results (    912 FUNSUB calls in last call of D01EAF)

   I        Integral   Estimated error
   1         0.0384        0.0004
   2         0.4012        0.0003
   3         0.3952        0.0003
   4         0.0258        0.0003
   5        -0.3672        0.0003
   6        -0.4227        0.0003
   7        -0.0895        0.0003
   8         0.3260        0.0003
   9         0.4417        0.0003
  10         0.1514        0.0003
```

D01FBF – NAG Fortran Library Routine Document

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

1. Purpose

D01FBF computes an estimate of a multi-dimensional integral (from 1 to 20 dimensions), given the analytic form of the integrand and suitable Gaussian weights and abscissae.

2. Specification

```
real FUNCTION D01FBF (NDIM, NPTVEC, LWA, WEIGHT, ABSCIS, FUN, IFAIL)
INTEGER        NDIM, NPTVEC(NDIM), LWA, IFAIL
real           WEIGHT(LWA), ABSCIS(LWA), FUN
EXTERNAL       FUN
```

3. Description

This routine approximates a multi-dimensional integral by evaluating the summation

$$\sum_{i_1=1}^{l_1} w_{1,i_1} \sum_{i_2=1}^{l_2} w_{2,i_2} \dots \sum_{i_n=1}^{l_n} w_{n,i_n} \; f(x_{1,i_1},x_{2,i_2},\dots,x_{n,i_n})$$

given the weights w_{j,i_j} and abscissae x_{j,i_j} for a multi-dimensional product integration rule (see Davis and Rabinowitz [1]). The number of dimensions may be anything from 1 to 20.

The weights and abscissae for each dimension must have been placed in successive segments of the arrays WEIGHT and ABSCIS; for example, by calling D01BBF or D01BCF once for each dimension using a quadrature formula and number of abscissae appropriate to the range of each x_j and to the functional dependence of f on x_j.

If normal weights are used, the summation will approximate the integral

$$\int w_1(x_1) \int w_2(x_2) \dots \int w_n(x_n) \; f(x_1,x_2,\dots,x_n) dx_n \dots dx_2 \; dx_1$$

where $w_j(x)$ is the weight function associated with the quadrature formula chosen for the jth dimension; while if adjusted weights are used, the summation will approximate the integral

$$\int \int \dots \int f(x_1,x_2,\dots,x_n) \; dx_n \dots dx_2 \; dx_1.$$

The user must supply a routine to evaluate

$$f(x_1,x_2,\dots,x_n)$$

at any values of $x_1,x_2,\dots,x_n$ within the range of integration.

4. References

[1] DAVIS, P.J., and RABINOWITZ, P.
Methods of Numerical Integration.
Academic Press, pp. 268-275, 1975.

5. Parameters

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integral, n.

Constraint: $1 \le$ NDIM ≤ 20.

2: NPTVEC(NDIM) – INTEGER array. *Input*

On entry: NPTVEC(j) must specify the number of points in the jth dimension of the summation, for $j = 1,2,\dots,n$.

3: LWA – INTEGER. *Input*

On entry: the dimension of the arrays WEIGHT and ABSCIS as declared in the (sub)program from which D01FBF is called.

Constraint: LWA ≥ NPTVEC(1) + NPTVEC(2) + ... + NPTVEC(NDIM).

4: WEIGHT(LWA) – ***real*** array. *Input*

On entry: WEIGHT must contain in succession the weights for the various dimensions, i.e. WEIGHT(k) contains the ith weight for the jth dimension, with

$$k = \text{NPTVEC}(1) + \text{NPTVEC}(2) + ... + \text{NPTVEC}(j-1) + i.$$

5: ABSCIS(LWA) – ***real*** array. *Input*

On entry: ABSCIS must contain in succession the abscissae for the various dimensions, i.e. ABSCIS(k) contains the ith abscissa for the jth dimension, with

$$k = \text{NPTVEC}(1) + \text{NPTVEC}(2) + ... + \text{NPTVEC}(j-1) + i.$$

6: FUN – ***real*** FUNCTION, supplied by the user. *External Procedure*

FUN must return the value of the integrand f at a given point.

Its specification is:

```
real FUNCTION FUN(NDIM, X)
INTEGER       NDIM
real          X(NDIM)
```

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integral, n.

2: X(NDIM) – ***real*** array. *Input*

On entry: the co-ordinates of the point at which the integrand must be evaluated.

FUN must be declared as EXTERNAL in the (sub)program from which D01FBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, NDIM < 1,
or NDIM > 20,
or LWA < NPTVEC(1) + NPTVEC(2) + ... + NPTVEC(NDIM).

7. Accuracy

The accuracy of the computed multi-dimensional sum depends on the weights and the integrand values at the abscissae. If these numbers vary significantly in size and sign then considerable accuracy could be lost. If these numbers are all positive, then little accuracy will be lost in computing the sum.

8. Further Comments

The total time taken by the routine will be proportional to

T×NPTVEC(1)×NPTVEC(2)×...×NPTVEC(NDIM),

where T is the time taken for one evaluation of FUN.

9. Example

This example program evaluates the integral

$$\int_1^2\int_0^\infty\int_{-\infty}^\infty\int_1^\infty \frac{(x_1x_2x_3)^6}{(x_4+2)^8}e^{-2x_2}e^{-0.5x_3^2}\,dx_4dx_3dx_2dx_1$$

using adjusted weights. The quadrature formulae chosen are:

x_1 : Gauss-Legendre, $a = 1.0$, $b = 2.0$,
x_2 : Gauss-Laguerre, $a = 0.0$, $b = 2.0$,
x_3 : Gauss-Hermite, $a = 0.0$, $b = 0.5$,
x_4 : Gauss-Rational, $a = 1.0$, $b = 2.0$.

Four points are sufficient in each dimension, as this integral in is in fact a product of four one-dimensional integrals, for each of which the chosen four-point formula is exact.

9.1. Program Text

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

```
*     D01FBF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NDIM, LWAMAX
      PARAMETER        (NDIM=4,LWAMAX=16)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             A, ANS, B
      INTEGER          I, IFAIL, ITYPE, IW, LWA
*     .. Local Arrays ..
      real             ABSCIS(LWAMAX), WEIGHT(LWAMAX)
      INTEGER          NPTVEC(NDIM)
*     .. External Functions ..
      real             D01FBF, FUN
      EXTERNAL         D01FBF, FUN
*     .. External Subroutines ..
      EXTERNAL         D01BAW, D01BAX, D01BAY, D01BAZ, D01BBF
*     .. Data statements ..
      DATA             NPTVEC/4, 4, 4, 4/
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01FBF Example Program Results'
      LWA = 0
      DO 20 I = 1, NDIM
         LWA = LWA + NPTVEC(I)
   20 CONTINUE
      IF (LWAMAX.GE.LWA) THEN
         ITYPE = 1
         IW = 1
         A = 1.0e0
         B = 2.0e0
         IFAIL = 0
*
```

```
         CALL D01BBF(D01BAZ,A,B,ITYPE,NPTVEC(1),WEIGHT(IW),ABSCIS(IW),
     +               IFAIL)
*
         IW = IW + NPTVEC(1)
         A = 0.0e0
         B = 2.0e0
*
         CALL D01BBF(D01BAX,A,B,ITYPE,NPTVEC(2),WEIGHT(IW),ABSCIS(IW),
     +               IFAIL)
*
         IW = IW + NPTVEC(2)
         A = 0.0e0
         B = 0.5e0
*
         CALL D01BBF(D01BAW,A,B,ITYPE,NPTVEC(3),WEIGHT(IW),ABSCIS(IW),
     +               IFAIL)
*
         IW = IW + NPTVEC(3)
         A = 1.0e0
         B = 2.0e0
*
         CALL D01BBF(D01BAY,A,B,ITYPE,NPTVEC(4),WEIGHT(IW),ABSCIS(IW),
     +               IFAIL)
*
         IFAIL = 0
*
         ANS = D01FBF(NDIM,NPTVEC,LWA,WEIGHT,ABSCIS,FUN,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Answer = ', ANS
      END IF
      STOP
*
99999 FORMAT (1X,A,F10.5)
      END
*
      real FUNCTION FUN(NDIM,X)
*     .. Scalar Arguments ..
      INTEGER             NDIM
*     .. Array Arguments ..
      real                X(NDIM)
*     .. Intrinsic Functions ..
      INTRINSIC           EXP
*     .. Executable Statements ..
      FUN = (X(1)*X(2)*X(3))**6/(X(4)+2.0e0)**8*EXP(-2.0e0*X(2)
     +      -0.5e0*X(3)*X(3))
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D01FBF Example Program Results

Answer =    0.25065
```

D01FCF – NAG Fortran Library Routine Document

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

1. Purpose

D01FCF attempts to evaluate a multi-dimensional integral (up to 15 dimensions), with constant and finite limits, to a specified relative accuracy, using an adaptive subdivision strategy.

2. Specification

```
 SUBROUTINE D01FCF (NDIM, A, B, MINPTS, MAXPTS, FUNCTN, EPS, ACC,
1                   LENWRK, WRKSTR, FINVAL, IFAIL)
 INTEGER          NDIM, MINPTS, MAXPTS, LENWRK, IFAIL
 real             A(NDIM), B(NDIM), FUNCTN, EPS, ACC,
1                 WRKSTR(LENWRK), FINVAL
 EXTERNAL         FUNCTN
```

3. Description

The routine returns an estimate of a multi-dimensional integral over a hyper-rectangle (i.e. with constant limits), and also an estimate of the relative error. The user sets the relative accuracy required, supplies the integrand as a function subprogram (FUNCTN), and also sets the minimum and maximum acceptable number of calls to FUNCTN (in MINPTS and MAXPTS).

The routine operates by repeated subdivision of the hyper-rectangular region into smaller hyper-rectangles. In each subregion, the integral is estimated using a seventh-degree rule, and an error estimate is obtained by comparison with a fifth-degree rule which uses a subset of the same points. The fourth differences of the integrand along each co-ordinate axis are evaluated, and the subregion is marked for possible future subdivision in half along that co-ordinate axis which has the largest absolute fourth difference.

If the estimated errors, totalled over the subregions, exceed the requested relative error (or if fewer than MINPTS calls to FUNCTN have been made), further subdivision is necessary, and is performed on the subregion with the largest estimated error, that subregion being halved along the appropriate co-ordinate axis.

The routine will fail if the requested relative error level has not been attained by the time MAXPTS calls to FUNCTN have been made; or, if the amount LENWRK of working storage is insufficient. A formula for the recommended value of LENWRK is given in Section 5. If a smaller value is used, and is exhausted in the course of execution, the routine switches to a less efficient mode of operation; only if this mode also breaks down is insufficient storage reported.

D01FCF is based on the HALF subroutine developed by van Dooren and de Ridder [1]. It uses a different basic rule, described by Genz and Malik [2].

4. References

[1] VAN DOOREN, P. and DE RIDDER, L.
An Adaptive Algorithm for Numerical Integration over an N-dimensional Cube.
J. Comput. Appl. Math. 2, No. 3, pp. 207-217, 1976.

[2] GENZ, A.C. and MALIK, A.A.
An Adaptive Algorithm for Numerical Integration over an N-dimensional Rectangular Region.
J. Comput. Appl. Math. 6, pp. 295-302, 1980.

5. Parameters

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integral, n.

Constraint: $2 \le \text{NDIM} \le 15$.

2: A(NDIM) – ***real*** array. *Input*

On entry: the lower limits of integration, a_i, for $i = 1,2,...,n$.

3: B(NDIM) – ***real*** array. *Input*

On entry: the upper limits of integration, b_i, for $i = 1,2,...,n$.

4: MINPTS – INTEGER. *Input/Output*

On entry: MINPTS must be set to the minimum number of integrand evaluations to be allowed.

On exit: MINPTS contains the actual number of integrand evaluations used by D01FCF.

5: MAXPTS – INTEGER. *Input*

On entry: the maximum number of integrand evaluations to be allowed.

Constraints: MAXPTS ≥ MINPTS
MAXPTS ≥ α,
where $\alpha = 2^{\text{NDIM}} + 2\times\text{NDIM}^2 + 2\times\text{NDIM} + 1$.

6: FUNCTN – ***real*** FUNCTION, supplied by the user. *External Procedure*

FUNCTN must return the value of the integrand f at a given point.

Its specification is:

```
real FUNCTION FUNCTN(NDIM, Z)
INTEGER       NDIM
real          Z(NDIM)
```

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integral, n.

2: Z(NDIM) – ***real*** array. *Input*

On entry: the co-ordinates of the point at which the integrand must be evaluated.

FUNCTN must be declared as EXTERNAL in the (sub)program from which D01FCF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: EPS – ***real.*** *Input*

On entry: the relative error acceptable to the user. When the solution is zero or very small relative accuracy may not be achievable but the user may still set EPS to a reasonable value and check for the error exit IFAIL = 2.

Constraint: EPS > 0.0.

8: ACC – ***real.*** *Output*

On exit: the estimated relative error in FINVAL.

9: LENWRK – INTEGER. *Input*

On entry: the dimension of the array WRKSTR as declared in the (sub)program from which D01FCF is called.

Suggested value: for maximum efficiency, LENWRK ≥ (NDIM+2)×(1+MAXPTS/α) (see parameter MAXPTS for α).

If LENWRK is less than this, the routine will usually run less efficiently and may fail.

Constraint: LENWRK $\geq$ 2×NDIM + 4.

10: WRKSTR(LENWRK) – ***real*** array. *Workspace*

11: FINVAL – ***real.*** *Output*

On exit: the best estimate obtained for the integral.

12: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. It is then essential to test the value of IFAIL on exit. To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL = 1

On entry, NDIM < 2,
or NDIM > 15,
or MAXPTS is too small,
or LENWRK < 2×NDIM + 4,
or EPS $\leq$ 0.0.

IFAIL = 2

MAXPTS was too small to obtain the required relative accuracy EPS. On soft failure, FINVAL and ACC contain estimates of the integral and the relative error, but ACC will be greater than EPS.

IFAIL = 3

LENWRK was too small. On soft failure, FINVAL and ACC contain estimates of the integral and the relative error, but ACC will be greater than EPS.

7. Accuracy

A relative error estimate is output through the parameter ACC.

8. Further Comments

Execution time will usually be dominated by the time taken to evaluate the integrand FUNCTN, and hence the maximum time that could be taken will be proportional to MAXPTS.

9. Example

This example program estimates the integral

$$\int_0^1\int_0^1\int_0^1\int_0^1 \frac{4z_1z_3^2\exp(2z_1z_3)}{(1+z_2+z_4)^2}dz_4dz_3dz_2dz_1 = 0.575364$$

The accuracy requested is one part in 10,000.

9.1. Program Text

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

```
*     D01FCF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NDIM, MAXPTS, LENWRK
      PARAMETER        (NDIM=4,MAXPTS=1000*NDIM,LENWRK=(NDIM+2)
     +                 *(1+MAXPTS/(2**NDIM+2*NDIM*NDIM+2*NDIM+1)))
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             ACC, EPS, FINVAL
      INTEGER          IFAIL, K, MINPTS
*     .. Local Arrays ..
      real             A(NDIM), B(NDIM), WRKSTR(LENWRK)
*     .. External Functions ..
      real             FUNCTN
      EXTERNAL         FUNCTN
*     .. External Subroutines ..
      EXTERNAL         D01FCF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01FCF Example Program Results'
      DO 20 K = 1, NDIM
         A(K) = 0.0e0
         B(K) = 1.0e0
   20 CONTINUE
      EPS = 0.0001e0
      MINPTS = 0
      IFAIL = 1
*
      CALL D01FCF(NDIM,A,B,MINPTS,MAXPTS,FUNCTN,EPS,ACC,LENWRK,WRKSTR,
     +            FINVAL,IFAIL)
*
      WRITE (NOUT,*)
      IF (IFAIL.NE.0) THEN
         WRITE (NOUT,99999) 'IFAIL =', IFAIL
         WRITE (NOUT,*)
      END IF
      IF (IFAIL.EQ.0 .OR. IFAIL.GE.2) THEN
         WRITE (NOUT,99998) 'Requested accuracy = ', EPS
         WRITE (NOUT,99997) 'Estimated value    = ', FINVAL
         WRITE (NOUT,99998) 'Estimated accuracy = ', ACC
      END IF
      STOP
*
99999 FORMAT (1X,A,I5)
99998 FORMAT (1X,A,e12.2)
99997 FORMAT (1X,A,F12.4)
      END
*
      real FUNCTION FUNCTN(NDIM,Z)
*     .. Scalar Arguments ..
      INTEGER               NDIM
*     .. Array Arguments ..
      real                  Z(NDIM)
*     .. Intrinsic Functions ..
      INTRINSIC             EXP
*     .. Executable Statements ..
      FUNCTN = 4.0e0*Z(1)*Z(3)*Z(3)*EXP(2.0e0*Z(1)*Z(3))/(1.0e0+Z(2)
     +         +Z(4))**2
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D01FCF Example Program Results

Requested accuracy =     0.10E-03
Estimated value    =       0.5754
Estimated accuracy =     0.99E-04
```

D01FDF – NAG Fortran Library Routine Document

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

1. Purpose

D01FDF calculates an approximation to a definite integral in up to 30 dimensions, using the method of Sag and Szekeres. The region of integration is an n-sphere, or by built-in transformation via the unit n-cube, any product region.

2. Specification

```
      SUBROUTINE D01FDF (NDIM, FUNCTN, SIGMA, REGION, LIMIT, R0, U, RESULT,
     1                   NCALLS, IFAIL)
      INTEGER            NDIM, LIMIT, NCALLS, IFAIL
      real               FUNCTN, SIGMA, R0, U, RESULT
      EXTERNAL           FUNCTN, REGION
```

3. Description

This subroutine calculates an approximation to

$$\int_{\substack{n\text{-sphere}\\ \text{of radius } \sigma}} f(x_1,x_2,\ldots,x_n)\ dx_1 dx_2 \ldots dx_n \tag{1}$$

or, more generally,

$$\int_{c_1}^{d_1} dx_1 \ldots \int_{c_n}^{d_n} dx_n\ f(x_1,\ldots,x_n) \tag{2}$$

where each c_i and d_i may be functions of x_j $(j<i)$.

The routine uses the method of Sag and Szekeres [1], which exploits a property of the shifted p-point trapezoidal rule, namely, that it integrates exactly all polynomials of degree $< p$ (Krylov [2]). An attempt is made to induce periodicity in the integrand by making a parameterised transformation to the unit n-sphere. The Jacobian of the transformation and all its direct derivatives vanish rapidly towards the surface of the unit n-sphere, so that, except for functions which have strong singularities on the boundary, the resulting integrand will be pseudo-periodic. In addition, the variation in the integrand can be considerably reduced, causing the trapezoidal rule to perform well.

Integrals of the form (1) are transformed to the unit n-sphere by the change of variables:

$$x_i = y_i \frac{\sigma}{r} \tanh\left(\frac{ur}{1-r^2}\right)$$

where $r^2 = \sum_{i=1}^{n} y_i^2$ and u is an adjustable parameter.

Integrals of the form (2) are first of all transformed to the n-cube $[-1,1]^n$ by a linear change of variables

$$x_i = ((d_i+c_i)+(d_i-c_i)y_i)/2$$

and then to the unit sphere by a further change of variables

$$y_i = \tanh\left(\frac{uz_i}{1-r}\right)$$

where $r^2 = \sum_{i=1}^{n} z_i^2$ and u is again an adjustable parameter.

The parameter u in these transformations determines how the transformed integrand is distributed between the origin and the surface of the unit n-sphere. A typical value of u is 1.5. For larger u, the integrand is concentrated toward the centre of the unit n-sphere, while for smaller u it is concentrated toward the perimeter.

In performing the integration over the unit n-sphere by the trapezoidal rule, a displaced equidistant grid of size h is constructed. The points of the mesh lie on concentric layers of radius

$$r_i = \frac{h}{4}\sqrt{n+8(i-1)} \qquad \text{for } i = 1,2,3,\ldots.$$

The routine requires the user to specify an approximate maximum number of points to be used, and then computes the largest number of whole layers to be used, subject to an upper limit of 400 layers.

In practice, the rapidly-decreasing Jacobian makes it unnecessary to include the whole unit n-sphere and the integration region is limited by a user-specified cut-off radius $r_0 < 1$. The grid-spacing h is determined by r_0 and the number of layers to be used. A typical value of r_0 is 0.8.

Some experimentation may be required with the choice of r_0 (which determines how much of the unit n-sphere is included) and u (which determines how the transformed integrand is distributed between the origin and surface of the unit n-sphere), to obtain best results for particular families of integrals. This matter is discussed further in Section 8.

4. References

[1] SAG, T.W. and SZEKERES, G.
Numerical Evaluation of High-Dimensional Integrals.
Math. Comput. 18, pp. 245-253, 1964.

[2] KRYLOV, V.I.
Approximate Calculation of Integrals (trans. A.H. Stroud).
Macmillan, 1962.

5. Parameters

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integral, n.

Constraint: $1 \le$ NDIM ≤ 30.

2: FUNCTN – ***real*** FUNCTION, supplied by the user. *External Procedure*

FUNCTN must return the value of the integrand f at a given point.

Its specification is:

```
real FUNCTION FUNCTN(NDIM, X)
INTEGER       NDIM
real          X(NDIM)
```

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integral, n.

2: X(NDIM) – ***real*** array. *Input*

On entry: the co-ordinates of the point at which the integrand must be evaluated.

FUNCTN must be declared as EXTERNAL in the (sub)program from which D01FDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

3: SIGMA – ***real.*** *Input*

On entry: SIGMA indicates the region of integration:

if SIGMA ≥ 0.0, the integration is carried out over the n-sphere of radius SIGMA, centred at the origin;

if SIGMA < 0.0, the integration is carried out over the product region described by the user-specified subroutine REGION.

4: REGION – SUBROUTINE, supplied by the user. *External Procedure*

If SIGMA < 0.0, REGION must evaluate the limits of integration in any dimension.

Its specification is:

```
SUBROUTINE REGION(NDIM, X, J, C, D)
INTEGER    NDIM, J
real       X(NDIM), C, D
```

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integral, n.

2: X(NDIM) – ***real*** array. *Input*

On entry: X(1),...,X(j–1) contain the current values of the first (j–1) variables, which may be used if necessary in calculating c_j and d_j.

3: J – INTEGER. *Input*

On entry: the index j for which the limits of the range of integration are required.

4: C – ***real.*** *Output*

On exit: the lower limit c_j of the range of x_j.

5: D – ***real.*** *Output*

On exit: the upper limit d_j of the range of x_j.

If SIGMA ≥ 0.0, REGION is not called by D01FDF, but a dummy routine must be supplied (NAG Fortran Library auxiliary routine D01FDV may be used).

REGION must be declared as EXTERNAL in the (sub)program from which D01FDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: LIMIT – INTEGER. *Input*

On entry: the approximate maximum number of integrand evaluations to be used.

Constraint: LIMIT ≥ 100.

6: R0 – ***real.*** *Input*

On entry: the cutoff radius on the unit n-sphere, which may be regarded as an adjustable parameter of the method.

Suggested value: a typical value is R0 = 0.8. (See also Section 8.)

Constraint: 0.0 < R0 < 1.0.

7: U – ***real.*** *Input*

On entry: U must specify an adjustable parameter of the transformation to the unit n-sphere.

Suggested value: a typical value is U = 1.5. (See also Section 8.)

Constraint: U > 0.0.

8: RESULT – ***real.*** *Output*

On exit: an estimate of the value of the integral.

9: NCALLS – INTEGER. *Output*

On exit: the actual number of integrand evaluations used. (See also Section 8.)

10: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, NDIM < 1
or NDIM > 30.

IFAIL = 2

On entry, LIMIT < 100.

IFAIL = 3

On entry, R0 ≤ 0.0
or R0 ≥ 1.0.

IFAIL = 4

On entry, U ≤ 0.0.

7. Accuracy

No error estimate is returned, but results may be verified by repeating with an increased value of LIMIT (provided that this causes an increase in the returned value of NCALLS).

8. Further Comments

The time taken by the routine will be approximately proportional to the returned value of NCALLS, which, except in the circumstances outlined in (b) below, will be close to the given value of LIMIT.

(a) Choice of R0 and U

If the chosen combination of r_0 and u is too large in relation to the machine accuracy it is possible that some of the points generated in the original region of integration may transform into points in the unit n-sphere which lie too close to the boundary surface to be distinguished from it to machine accuracy (despite the fact that $r_0 < 1$). To be specific, the combination of r_0 and u is too large if

$$\frac{ur_0}{1-r_0^2} > 0.3465(t-1), \text{ if SIGMA} \geq 0.0,$$

or

$$\frac{ur_0}{1-r_0} > 0.3465(t-1), \text{ if SIGMA} < 0.0,$$

where t is the number of bits in the mantissa of a ***real*** number.

The contribution of such points to the integral is neglected. This may be justified by appeal to the fact that the Jacobian of the transformation rapidly approaches zero towards the surface. Neglect of these points avoids the occurrence of overflow with integrands which are infinite on the boundary.

(b) Values of LIMIT and NCALLS

LIMIT is an approximate upper limit to the number of integrand evaluations, and may not be chosen less than 100. There are two circumstances when the returned value of NCALLS (the actual number of evaluations used) may be significantly less than LIMIT.

Firstly, as explained in Section 8(a), an unsuitably large combination of R0 and U may result in some of the points being unusable. Such points are not included in the returned value of NCALLS.

Secondly, no more than 400 layers will ever be used, no matter how high LIMIT is set. This places an effective upper limit on NCALLS as follows:

$n = 1$:	56
$n = 2$:	1252
$n = 3$:	23690
$n = 4$:	394528
$n = 5$:	5956906

9. Example

This example program calculates the integral

$$\iiint_s \frac{dx_1 dx_2 dx_3}{\sqrt{\sigma^2 - r^2}} = 22.2066$$

where s is the 3-sphere of radius σ, $r^2 = x_1^2 + x_2^2 + x_3^2$ and $\sigma = 1.5$. Both sphere-to-sphere and general product region transformations are used. For the former, we use $r_0 = 0.9$ and $u = 1.5$; for the latter, $r_0 = 0.8$ and $u = 1.5$.

9.1. Program Text

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

```
*     D01FDF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             R0, RESULT, SIGMA, U
      INTEGER          IFAIL, LIMIT, NCALLS, NDIM
*     .. External Functions ..
      real             FUNCTN
      EXTERNAL         FUNCTN
*     .. External Subroutines ..
      EXTERNAL         D01FDF, D01FDV, REGION
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01FDF Example Program Results'
      NDIM = 3
      LIMIT = 8000
      U = 1.5e0
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Sphere-to-sphere transformation'
      SIGMA = 1.5e0
      R0 = 0.9e0
      IFAIL = 0
*
      CALL D01FDF(NDIM,FUNCTN,SIGMA,D01FDV,LIMIT,R0,U,RESULT,NCALLS,
     +            IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'Estimated value of the integral =', RESULT
      WRITE (NOUT,99998) 'Number of integrand evaluations =', NCALLS
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Product region transformation'
      SIGMA = -1.0e0
      R0 = 0.8e0
      IFAIL = 0
*
      CALL D01FDF(NDIM,FUNCTN,SIGMA,REGION,LIMIT,R0,U,RESULT,NCALLS,
     +            IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'Estimated value of the integral =', RESULT
      WRITE (NOUT,99998) 'Number of integrand evaluations =', NCALLS
      STOP
*
99999 FORMAT (1X,A,F9.3)
99998 FORMAT (1X,A,I4)
      END
*
      real FUNCTION FUNCTN(NDIM,X)
*     .. Scalar Arguments ..
      INTEGER              NDIM
*     .. Array Arguments ..
      real                 X(NDIM)
*     .. Local Scalars ..
      INTEGER              I
*     .. Intrinsic Functions ..
      INTRINSIC            ABS, SQRT
*     .. Executable Statements ..
      FUNCTN = 2.25e0
      DO 20 I = 1, NDIM
         FUNCTN = FUNCTN - X(I)*X(I)
   20 CONTINUE
      FUNCTN = 1.0e0/SQRT(ABS(FUNCTN))
      RETURN
      END
*
      SUBROUTINE REGION(NDIM,X,J,C,D)
*     .. Scalar Arguments ..
      real             C, D
      INTEGER          J, NDIM
*     .. Array Arguments ..
      real             X(NDIM)
*     .. Local Scalars ..
      real             SUM
      INTEGER          I, J1
*     .. Intrinsic Functions ..
      INTRINSIC        ABS, SQRT
*     .. Executable Statements ..
      C = -1.5e0
      D = 1.5e0
      IF (J.GT.1) THEN
         SUM = 2.25e0
         J1 = J - 1
         DO 20 I = 1, J1
            SUM = SUM - X(I)*X(I)
   20    CONTINUE
         D = SQRT(ABS(SUM))
         C = -D
      END IF
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D01FDF Example Program Results

Sphere-to-sphere transformation

Estimated value of the integral =   22.168
Number of integrand evaluations =8026

Product region transformation

Estimated value of the integral =   22.137
Number of integrand evaluations =8026
```

D01GAF – NAG Fortran Library Routine Document

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

1. Purpose

D01GAF integrates a function which is specified numerically at four or more points, over the whole of its specified range, using third-order finite-difference formulae with error estimates, according to a method due to Gill and Miller.

2. Specification

```
SUBROUTINE D01GAF (X, Y, N, ANS, ER, IFAIL)
INTEGER          N, IFAIL
real             X(N), Y(N), ANS, ER
```

3. Description

This routine evaluates the definite integral

$$I = \int_{x_1}^{x_n} y(x)\ dx,$$

where the function y is specified at the n-points $x_1,x_2,...,x_n$, which should be all distinct, and in either ascending or descending order. The integral between successive points is calculated by a four-point finite-difference formula centred on the interval concerned, except in the case of the first and last intervals, where four-point forward and backward difference formulae respectively are employed. If n is less than 4, the routine fails. An approximation to the truncation error is integrated and added to the result. It is also returned separately to give an estimate of the uncertainty in the result. The method is due to Gill and Miller.

4. References

[1] GILL, P.E. and MILLER, G.F.
An Algorithm for the Integration of Unequally Spaced Data.
Comput. J., 15, pp. 80-83, 1972.

5. Parameters

1: X(N) – ***real*** array. *Input*

On entry: the values of the independent variable, i.e. the $x_1,x_2,...,x_n$.

Constraint: either X(1) < X(2) < ... < X(N) or X(1) > X(2) > ... > X(N).

2: Y(N) – ***real*** array. *Input*

On entry: the values of the dependent variable y_i at the points x_i, for $i = 1,2,...,n$.

3: N – INTEGER. *Input*

On entry: the number of points, n.

Constraint: N $\geq$ 4.

4: ANS – ***real***. *Output*

On exit: the estimate of the integral.

5: ER – ***real***. *Output*

On exit: an estimate of the uncertainty in ANS.

6: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

Indicates that fewer than four-points have been supplied to the routine.

IFAIL = 2

Values of X are neither strictly increasing nor strictly decreasing.

IFAIL = 3

Two points have the same X-value.

No error is reported arising from the relative magnitudes of ANS and ER on return, due to the difficulty when the true answer is zero.

7. Accuracy

No accuracy level is specified by the user before calling the routine but on return ABS(ER) is an approximation to, but not necessarily a bound for, $|I-\text{ANS}|$. If on exit IFAIL > 0, both ANS and ER are returned as zero.

8. Further Comments

The time taken by the routine depends on the number of points supplied, n.

In their paper, Gill and Miller [1] do not add the quantity ER to ANS before return. However, extensive tests have shown that a dramatic reduction in the error often results from such addition. In other cases, it does not make an improvement, but these tend to be cases of low accuracy in which the modified answer is not significantly inferior to the unmodified one. The user has the option of recovering the Gill-Miller answer by subtracting ER from ANS on return from the routine.

9. Example

The following example program evaluates the integral

$$\int_0^1 \frac{4}{1+x^2}\,dx = \pi$$

reading in the function values at 21 unequally-spaced points.

9.1. Program Text

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

```
*     D01GAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NMAX
      PARAMETER        (NMAX=21)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             ANS, ERROR
      INTEGER          I, IFAIL, N
```

```
*     .. Local Arrays ..
      real             X(NMAX), Y(NMAX)
*     .. External Subroutines ..
      EXTERNAL         D01GAF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01GAF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) N
      WRITE (NOUT,*)
      IF (N.LE.NMAX) THEN
         READ (NIN,*) (X(I),Y(I),I=1,N)
         IFAIL = 1
*
         CALL D01GAF(X,Y,N,ANS,ERROR,IFAIL)
*
         IF (IFAIL.EQ.0) THEN
            WRITE (NOUT,99999) 'Integral = ', ANS,
     +        '     Estimated error = ', ERROR
         ELSE IF (IFAIL.EQ.1) THEN
            WRITE (NOUT,*) 'Less than 4 points supplied'
         ELSE IF (IFAIL.EQ.2) THEN
            WRITE (NOUT,*)
     +        'Points not in increasing or decreasing order'
         ELSE IF (IFAIL.EQ.3) THEN
            WRITE (NOUT,*) 'Points not all distinct'
         END IF
      ELSE
         WRITE (NOUT,*) 'More than NMAX data points'
      END IF
      STOP
*
99999 FORMAT (1X,A,F7.4,A,F7.4)
      END
```

9.2. Program Data

```
D01GAF Example Program Data
   21
   0.00   4.0000
   0.04   3.9936
   0.08   3.9746
   0.12   3.9432
   0.22   3.8153
   0.26   3.7467
   0.30   3.6697
   0.38   3.4943
   0.39   3.4719
   0.42   3.4002
   0.45   3.3264
   0.46   3.3014
   0.60   2.9412
   0.68   2.7352
   0.72   2.6344
   0.73   2.6094
   0.83   2.3684
   0.85   2.3222
   0.88   2.2543
   0.90   2.2099
   1.00   2.0000
```

9.3. Program Results

```
D01GAF Example Program Results

Integral =  3.1414     Estimated error = -0.0001
```

D01GBF – NAG Fortran Library Routine Document

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

1. Purpose

D01GBF returns an approximation to the integral of a function over a hyper-rectangular region, using a Monte-Carlo method. An approximate relative error estimate is also returned. This routine is suitable for low accuracy work.

2. Specification

```
      SUBROUTINE D01GBF (NDIM, A, B, MINCLS, MAXCLS, FUNCTN, EPS,
     1                   ACC, LENWRK, WRKSTR, FINEST, IFAIL)
      INTEGER        NDIM, MINCLS, MAXCLS, LENWRK, IFAIL
      real           A(NDIM), B(NDIM), FUNCTN, EPS, ACC,
     1               WRKSTR(LENWRK), FINEST
      EXTERNAL       FUNCTN
```

3. Description

D01GBF uses an adaptive Monte-Carlo method based on the algorithm described by Lautrup [1]. It is implemented for integrals of the form:

$$\int_{a_1}^{b_1} \int_{a_2}^{b_2} \dots \int_{a_n}^{b_n} f(x_1,x_2,\dots,x_n)\ dx_n\ \dots\ dx_2\ dx_1.$$

Upon entry, unless LENWRK has been set to the minimum value 10×NDIM, the routine subdivides the integration region into a number of equal volume subregions. Inside each subregion the integral and the variance are estimated by means of pseudo-random sampling. All contributions are added together to produce an estimate for the whole integral and total variance. The variance along each co-ordinate axis is determined and the routine uses this information to increase the density and change the widths of the subintervals along each axis, so as to reduce the total variance. The total number of subregions is then increased by a factor of two and the program recycles for another iteration. The program stops when a desired accuracy has been reached or too many integral evaluations are needed for the next cycle.

4. References

[1] LAUTRUP, B.
An Adaptive Multi-dimensional Integration Procedure.
Proc. 2nd Coll. on Advanced Methods in Theoretical Physics.
Marseille, 1971.

5. Parameters

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integral, n.

Constraint: NDIM ≥ 1.

2: A(NDIM) – ***real*** array. *Input*

On entry: the lower limits of integration, a_i, for $i = 1,2,\dots,n$.

3: B(NDIM) – ***real*** array. *Input*

On entry: the upper limits of integration, b_i, for $i = 1,2,\dots,n$.

4: MINCLS – INTEGER. *Input/Output*

On entry: MINCLS must be set:

either to the minimum number of integrand evaluations to be allowed, in which case MINCLS ≥ 0;

or to a negative value. In this case the routine assumes that a previous call had been made with the same parameters NDIM, A and B and with either the same integrand (in which case D01GBF continues calculation) or a similar integrand (in which case D01GBF begins the calculation with the subdivision used in the last iteration of the previous call). See also WRKSTR.

On exit: MINCLS contains the number of integrand evaluations actually used by D01GBF.

5: MAXCLS – INTEGER. *Input*

On entry: the maximum number of integrand evaluations to be allowed. In the continuation case this is the number of new integrand evaluations to be allowed. These counts do not include zero integrand values.

Constraints: MAXCLS > MINCLS,
MAXCLS ≥ 4×(NDIM+1).

6: FUNCTN – ***real*** FUNCTION, supplied by the user. *External Procedure*

FUNCTN must return the value of the integrand *f* at a given point.

Its specification is:

```
real FUNCTION  FUNCTN(NDIM, X)
INTEGER        NDIM
real           X(NDIM)
```

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integral, *n*.

2: X(NDIM) – ***real*** array. *Input*

On entry: the co-ordinates of the point at which the integrand must be evaluated.

FUNCTN must be declared as EXTERNAL in the (sub)program from which D01GBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: EPS – ***real.*** *Input*

On entry: the relative accuracy required.

Constraint: EPS ≥ 0.0.

8: ACC – ***real.*** *Output*

On exit: the estimated relative accuracy of FINEST.

9: LENWRK – INTEGER. *Input*

On entry: the dimension of the array WRKSTR as declared in the (sub)program from which D01GBF is called.

For maximum efficiency, LENWRK should be about

$$3\times\text{NDIM}\times(\text{MAXCLS}/4)^{1/\text{NDIM}} + 7\times\text{NDIM}.$$

If LENWRK is given the value 10×NDIM then the subroutine uses only one iteration of a crude Monte-Carlo method with MAXCLS sample points.

Constraint: LENWRK ≥ 10×NDIM.

10: WRKSTR(LENWRK) – ***real*** array. *Input/Output*

On entry: if MINCLS < 0.0, WRKSTR must be unchanged from the previous call of D01GBF – except that for a new integrand WRKSTR(LENWRK) must be set to 0.0. See MINCLS.

On exit: WRKSTR contains information about the current subinterval structure which could be used in later calls of D01GBF. In particular, WRKSTR(j) gives the number of subintervals used along the jth co-ordinate axis.

11: FINEST – ***real.*** *Output*

On exit: the best estimate obtained for the integral.

12: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.** To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL = 1

On entry, NDIM < 1,
or MINCLS ≥ MAXCLS,
or LENWRK < 10×NDIM,
or MAXCLS < 4×(NDIM+1),
or EPS < 0.0.

IFAIL = 2

MAXCLS was too small for D01GBF to obtain the required relative accuracy EPS. In this case D01GBF returns a value of FINEST with estimated relative error ACC, but ACC will be greater than EPS. This error exit may be taken before MAXCLS non-zero integrand evaluations have actually occurred, if the routine calculates that the current estimates could not be improved before MAXCLS was exceeded.

7. Accuracy

A relative error estimate is output through the parameter ACC. The confidence factor is set so that the actual error should be less than ACC 90% of the time. If a user desires a higher confidence level then a smaller value of EPS should be used.

8. Further Comments

The running time for D01GBF will usually be dominated by the time used to evaluate the integrand FUNCTN, so the maximum time that could be used is approximately proportional to MAXCLS.

For some integrands, particularly those that are poorly behaved in a small part of the integration region, D01GBF may terminate with a value of ACC which is significantly smaller than the actual relative error. This should be suspected if the returned value of MINCLS is small relative to the expected difficulty of the integral. Where this occurs, D01GBF should be called again, but with a higher entry value of MINCLS (e.g. twice the returned value) and the results compared with those from the previous call.

The exact values of FINEST and ACC on return will depend (within statistical limits) on the sequence of random numbers generated within D01GBF by calls to G05CAF. Separate runs will produce identical answers unless the part of the program executed prior to calling D01GBF also calls (directly or indirectly) routines from the G05 chapter, and the series of such calls differs between runs. If desired, the user may ensure the identity or difference between runs of the results returned by D01GBF, by calling G05CBF or G05CCF respectively, immediately before calling D01GBF.

9. Example

This example program calculates the integral

$$\int_0^1\int_0^1\int_0^1\int_0^1 \frac{4x_1x_3^2\exp(2x_1x_3)}{(1+x_2+x_4)^2}dx_1dx_2dx_3dx_4 = 0.575364.$$

9.1. Program Text

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

```
*     D01GBF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NDIM, MAXCLS, LENWRK
      PARAMETER        (NDIM=4,MAXCLS=20000,LENWRK=500)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             ACC, EPS, FINEST
      INTEGER          IFAIL, K, MINCLS
*     .. Local Arrays ..
      real             A(NDIM), B(NDIM), WRKSTR(LENWRK)
*     .. External Functions ..
      real             FUNCTN
      EXTERNAL         FUNCTN
*     .. External Subroutines ..
      EXTERNAL         D01GBF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01GBF Example Program Results'
      DO 20 K = 1, NDIM
         A(K) = 0.0e0
         B(K) = 1.0e0
   20 CONTINUE
      EPS = 0.01e0
      MINCLS = 1000
      IFAIL = 1
*
      CALL D01GBF(NDIM,A,B,MINCLS,MAXCLS,FUNCTN,EPS,ACC,LENWRK,WRKSTR,
     +            FINEST,IFAIL)
*
      WRITE (NOUT,*)
      IF (IFAIL.GT.0) THEN
         WRITE (NOUT,99999) 'D01GBF fails. IFAIL =', IFAIL
         WRITE (NOUT,*)
      END IF
      IF (IFAIL.EQ.0 .OR. IFAIL.EQ.2) THEN
         WRITE (NOUT,99998) 'Requested accuracy    = ', EPS
         WRITE (NOUT,99997) 'Estimated value       = ', FINEST
         WRITE (NOUT,99998) 'Estimated accuracy    = ', ACC
         WRITE (NOUT,99999) 'Number of evaluations = ', MINCLS
      END IF
      STOP
*
99999 FORMAT (1X,A,I5)
99998 FORMAT (1X,A,e13.2)
99997 FORMAT (1X,A,F13.5)
      END
*
```

```
      real FUNCTION FUNCTN(NDIM,X)
*     .. Scalar Arguments ..
      INTEGER              NDIM
*     .. Array Arguments ..
      real                 X(NDIM)
*     .. Intrinsic Functions ..
      INTRINSIC            EXP
*     .. Executable Statements ..
      FUNCTN = 4.0e0*X(1)*X(3)**2*EXP(2.0e0*X(1)*X(3))/(1.0e0+X(2)+X(4))
     +         **2
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D01GBF Example Program Results

 Requested accuracy    =      0.10E-01
 Estimated value       =       0.57554
 Estimated accuracy    =      0.82E-02
 Number of evaluations =  1728
```

D01GCF – NAG Fortran Library Routine Document

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

1. Purpose

D01GCF calculates an approximation to a definite integral in up to 20 dimensions, using the Korobov-Conroy number theoretic method.

2. Specification

```
      SUBROUTINE D01GCF (NDIM, FUNCTN, REGION, NPTS, VK, NRAND, ITRANS,
     1                   RES, ERR, IFAIL)
      INTEGER            NDIM, NPTS, NRAND, ITRANS, IFAIL
      real               FUNCTN, VK(NDIM), RES, ERR
      EXTERNAL           FUNCTN, REGION
```

3. Description

This routine calculates an approximation to the integral,

$$I = \int_{c_1}^{d_1} dx_1,...,\int_{c_n}^{d_n} dx_n\, f(x_1,x_2,...,x_n) \qquad (1)$$

using the Korobov-Conroy number theoretic method ([1], [2], [3]). The region of integration defined in (1) is such that generally c_i and d_i may be functions of $x_1,x_2,...,x_{i-1}$, for $i = 2,3,...,n$, with c_1 and d_1 constants. The integral is first of all transformed to an integral over the n-cube $[0,1]^n$ by the change of variables

$$x_i = c_i + (d_i - c_i)y_i, \qquad i = 1,2,...,n.$$

The method then uses as its basis the number theoretic formula for the n-cube, $[0,1]^n$:

$$\int_0^1 dx_1 \ ... \int_0^1 dx_n\ g(x_1,x_2,...,x_n) = \frac{1}{p}\sum_{k=1}^{p} g\left(\left\{k\frac{a_1}{p}\right\},...,\left\{k\frac{a_n}{p}\right\}\right) - E \qquad (2)$$

where $\{x\}$ denotes the fractional part of x, $a_1,a_2,...,a_n$ are the so-called optimal coefficients, E is the error and p is a prime integer. (It is strictly only necessary that p be relatively prime to all $a_1,a_2,...,a_n$ and is in fact chosen to be even for some cases in Conroy [3].) The method makes use of properties of the Fourier expansion of $g(x_1,x_2,...,x_n)$ which is assumed to have some degree of periodicity. Depending on the choice of $a_1,a_2,...,a_n$ the contributions from certain groups of Fourier coefficients are eliminated from the error, E. Korobov shows that $a_1,a_2,...,a_n$ can be chosen so that the error satisfies

$$E \le CK\, p^{-\alpha}\ln^{\alpha\beta}p \qquad (3)$$

where α and C are real numbers depending on the convergence rate of the Fourier series, β is a constant depending on n and K is a constant depending on α and n. There are a number of procedures for calculating these optimal coefficients. Korobov imposes the constraint that

$$a_1 = 1$$
$$a_i = a^{i-1}\ (\mathrm{mod}\,p) \qquad (4)$$

and gives a procedure for calculating the parameter, a, to satisfy the optimal conditions.

In this routine the periodisation is achieved by the simple transformation

$$x_i = y_i^2(3-2y_i), \qquad i = 1,2,...,n.$$

More sophisticated periodisation procedures are available but in practice the degree of periodisation does not appear to be a critical requirement of the method.

An easily calculable error estimate is not available apart from repetition with an increasing sequence of values of p which can yield erratic results. The difficulties have been studied by

Cranley and Patterson [4] who have proposed a Monte Carlo error estimate arising from converting (2) into a stochastic integration rule by the inclusion of a random origin shift which leaves the form of the error (3) unchanged; i.e. in the formula (2), $\left\{k\frac{a_i}{p}\right\}$ is replaced by $\left\{\alpha_i+k\frac{a_i}{p}\right\}$, for $i = 1,2,...,n$, where each α_i, is uniformly distributed over [0,1]. Computing the integral for each of a sequence of random vectors α allows a 'standard error' to be estimated.

This routine provides built-in sets of optimal coefficients, corresponding to six different values of p. Alternatively the optimal coefficients may be supplied by the user. Routines D01GYF and D01GZF compute the optimal coefficients for the cases where p is a prime number or p is a product of 2 primes, respectively.

4. References

[1] KOROBOV, N.M.
The Approximate Calculation of Multiple Integrals Using Number Theoretic Methods.
Dokl. Acad. Nauk. SSSR, 115, pp. 1062-1065, 1957.

[2] KOROBOV, N.M.
Number Theoretic Methods in Approximate Analysis.
Fizmatgiz, Moscow, 1963.

[3] CONROY, H.
Molecular Schroedinger Equation VIII. A New Method for Evaluating Multidimensional Integrals.
J. Chem. Phys., 47, pp. 5307-5318, 1967.

[4] CRANLEY, R. and PATTERSON, T.N.L.
Randomisation of Number Theoretic Methods for Multiple Integration.
SIAM J. Numer. Anal., 13, pp. 904-914, 1976.

5. Parameters

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integral, n.

Constraint: $1 \le$ NDIM ≤ 20.

2: FUNCTN – ***real*** FUNCTION, supplied by the user. *External Procedure*

FUNCTN must return the value of the integrand f at a given point.

Its specification is:

```
real FUNCTION FUNCTN(NDIM, X)
INTEGER       NDIM
real          X(NDIM)
```

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integral, n.

2: X(NDIM) – ***real*** array. *Input*

On entry: the co-ordinates of the point at which the integrand must be evaluated.

FUNCTN must be declared as EXTERNAL in the (sub)program from which D01GCF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

3: REGION – SUBROUTINE, supplied by the user. *External Procedure*

REGION must evaluate the limits of integration in any dimension.

Its specification is:

```
SUBROUTINE REGION(NDIM, X, J, C, D)
INTEGER    NDIM, J
real       X(NDIM), C, D
```

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integral, n.

2: X(NDIM) – ***real*** array. *Input*

On entry: X(1),...,X(j–1) contain the current values of the first j–1 variables, which may be used if necessary in calculating c_j and d_j.

3: J – INTEGER. *Input*

On entry: the index j for which the limits of the range of integration are required.

4: C – ***real.*** *Output*

On exit: the lower limit c_j of the range of x_j.

5: D – ***real.*** *Output*

On exit: the upper limit d_j of the range of x_j.

REGION must be declared as EXTERNAL in the (sub)program from which D01GCF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

4: NPTS – INTEGER. *Input*

On entry: the Korobov rule to be used. There are two alternatives depending on the value of NPTS.

(a) 1 ≤ NPTS ≤ 6.
In this case one of six preset rules is chosen using 2129, 5003, 10007, 20011, 40009 or 80021 points depending on the respective value of NPTS being 1, 2, 3, 4, 5 or 6.

(b) NPTS > 6.
NPTS is the number of actual points to be used with corresponding optimal coefficients supplied in the array VK.

Constraint: NPTS ≥ 1.

5: VK(NDIM) – ***real*** array. *Input/Output*

On entry: if NPTS > 6, VK must contain the n optimal coefficients (which may be calculated using D01GYF or D01GZF); if NPTS ≤ 6, VK need not be set.

On exit: if NPTS > 6, VK is unchanged; if NPTS ≤ 6, VK contains the n optimal coefficients used by the preset rule.

6: NRAND – INTEGER. *Input*

On entry: the number of random samples to be generated in the error estimation (generally a small value, say 3 to 5 is sufficient). The total number of integrand evaluations will be NRAND×NPTS.

Constraint: NRAND ≥ 1.

7: ITRANS – INTEGER. *Input*

On entry: indicates whether the periodising transformation is to be used:

if ITRANS = 0, the transformation is to be used.

if ITRANS ≠ 0, the transformation is to be suppresssed (to cover cases where the integrand may already be periodic or where the user desires to specify a particular transformation in the definition of FUNCTN).

Suggested value: ITRANS = 0.

8: RES – ***real.*** *Output*

On exit: an estimate of the value of the integral.

9: ERR – ***real.*** *Output*

On exit: the standard error as computed from NRAND sample values. If NRAND = 1, then ERR contains zero.

10: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, NDIM < 1,
or NDIM > 20.

IFAIL = 2

On entry, NPTS < 1.

IFAIL = 3

On entry, NRAND < 1.

7. Accuracy

An estimate of the absolute standard error is given by the value, on exit, of ERR.

8. Further Comments

The time taken by the routine will be approximately proportional to NRAND×p, where p is the number of points used.

The exact values of RES and ERR returned by D01GCF will depend (within statistical limits) on the sequence of random numbers generated within the routine by calls to G05CAF. To ensure that the results returned by D01GCF in separate runs are identical, users should call G05CBF immediately before calling D01GCF; to ensure that they are different, call G05CCF.

9. Example

This example calculates the integral

$$\int_0^1\int_0^1\int_0^1\int_0^1 \cos(0.5+2(x_1+x_2+x_3+x_4)-4)\ dx_1dx_2dx_3dx_4.$$

9.1. Program Text

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

```
*     D01GCF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NDIM
      PARAMETER        (NDIM=4)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             ERR, RES
      INTEGER          IFAIL, ITRANS, NPTS, NRAND
*     .. Local Arrays ..
      real             VK(NDIM)
*     .. External Functions ..
      real             FUNCT
      EXTERNAL         FUNCT
*     .. External Subroutines ..
      EXTERNAL         D01GCF, REGION
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01GCF Example Program Results'
      NPTS = 2
      ITRANS = 0
      NRAND = 4
      IFAIL = 0
*
      CALL D01GCF(NDIM,FUNCT,REGION,NPTS,VK,NRAND,ITRANS,RES,ERR,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'Result =', RES, '  Standard error =', ERR
      STOP
*
99999 FORMAT (1X,A,F13.5,A,e10.2)
      END
*
      SUBROUTINE REGION(N,X,J,A,B)
*     .. Scalar Arguments ..
      real              A, B
      INTEGER           J, N
*     .. Array Arguments ..
      real               X(N)
*     .. Executable Statements ..
      A = 0.0e0
      B = 1.0e0
      RETURN
      END
*
      real FUNCTION FUNCT(NDIM,X)
*     .. Scalar Arguments ..
      INTEGER              NDIM
*     .. Array Arguments ..
      real                 X(NDIM)
*     .. Local Scalars ..
      real                 SUM
      INTEGER              J
*     .. Intrinsic Functions ..
      INTRINSIC            COS, real
*     .. Executable Statements ..
      SUM = 0.0e0
      DO 20 J = 1, NDIM
         SUM = SUM + X(J)
   20 CONTINUE
      FUNCT = COS(0.5e0+2.0e0*SUM-real(NDIM))
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D01GCF Example Program Results

Result =       0.43999  Standard error =  0.18E-05
```

D01GDF – NAG Fortran Library Routine Document

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

1. Purpose

D01GDF calculates an approximation to a definite integral in up to 20 dimensions, using the Korobov-Conroy number theoretic method. This routine is designed to be particularly efficient on vector processors.

2. Specification

```
      SUBROUTINE D01GDF (NDIM, VECFUN, VECREG, NPTS, VK, NRAND, ITRANS,
     1                   RES, ERR, IFAIL)
      INTEGER            NDIM, NPTS, NRAND, ITRANS, IFAIL
      real               VK(NDIM), RES, ERR
      EXTERNAL           VECFUN, VECREG
```

3. Description

This routine calculates an approximation to the integral,

$$I = \int_{c_1}^{d_1} \dots \int_{c_n}^{d_n} f(x_1,\dots,x_n)\; dx_n \;\dots\; dx_1 \tag{1}$$

using the Korobov-Conroy number theoretic method ([1], [2], [3]). The region of integration defined in (1) is such that generally c_i and d_i may be functions of $x_1,x_2,\dots,x_{i-1}$, for $i = 2,3,\dots,n$, with c_1 and d_1 constants. The integral is first of all transformed to an integral over the n–cube $[0,1]^n$ by the change of variables

$$x_i = c_i + (d_i - c_i)y_i, \qquad i = 1,2,\dots,n.$$

The method then uses as its basis the number theoretic formula for the n–cube, $[0,1]^n$:

$$\int_0^1 \dots \int_0^1 g(x_1,\dots,x_n)\; dx_n \;\dots\; dx_1 = \frac{1}{p}\sum_{k=1}^{p} g\left(\left\{k\frac{a_1}{p}\right\},\dots,\left\{k\frac{a_n}{p}\right\}\right) - E \tag{2}$$

where $\{x\}$ denotes the fractional part of x, $a_1,\dots,a_n$ are the so-called optimal coefficients, E is the error and p is a prime integer. (It is strictly only necessary that p be relatively prime to all $a_1,\dots,a_n$ and is in fact chosen to be even for some cases in Conroy, [3].) The method makes use of properties of the Fourier expansion of $g(x_1,\dots,x_n)$ which is assumed to have some degree of periodicity. Depending on the choice of $a_1,\dots,a_n$ the contributions from certain groups of Fourier coefficients are eliminated from the error, E. Korobov shows that $a_1,\dots,a_n$ can be chosen so that the error satisfies

$$E \le CK\, p^{-\alpha} \ln^{\alpha\beta} p \tag{3}$$

where α and C are real numbers depending on the convergence rate of the Fourier series, β is a constant depending on n and K is a constant depending on α and n. There are a number of procedures for calculating these optimal coefficients. Korobov imposes the constraint that

$$\begin{aligned} a_1 &= 1 \\ a_i &= a^{i-1} \pmod{p} \end{aligned} \tag{4}$$

and gives a procedure for calculating the parameter, a, to satisfy the optimal conditions.

In this routine the periodisation is achieved by the simple transformation

$$x_i = y_i^2(3-2y_i), \qquad i = 1,2,\dots,n.$$

More sophisticated periodisation procedures are available but in practice the degree of periodisation does not appear to be a critical requirement of the method.

An easily calculable error estimate is not available apart from repetition with an increasing sequence of values of p which can yield erratic results. The difficulties have been studied by Cranley and Patterson [4] who have proposed a Monte-Carlo error estimate arising from converting (2) into a stochastic integration rule by the inclusion of a random origin shift which leaves the form of the error (3) unchanged; i.e. in the formula (2), $\left\{k\frac{a_i}{p}\right\}$ is replaced by $\left\{\alpha_i+k\frac{a_i}{p}\right\}$, for $i = 1,2,...,n$, where each α_i, is uniformly distributed over [0,1]. Computing the integral for each of a sequence of random vectors α allows a 'standard error' to be estimated.

This routine provides built-in sets of optimal coefficients, corresponding to six different values of p. Alternatively, the optimal coefficients may be supplied by the user. D01GYF and D01GZF compute the optimal coefficients for the cases where p is a prime number or p is a product of two primes, respectively.

This routine is designed to be particularly efficient on vector processors, although it is very important that the user also codes the subroutines VECFUN and VECREG efficiently.

4. References

[1] KOROBOV, N.M.
The Approximate Calculation of Multiple Integrals Using Number Theoretic Methods.
Dokl. Acad. Nauk. SSSR, 115, pp. 1062-1065, 1957.

[2] KOROBOV, N.M.
Number Theoretic Methods in Approximate Analysis.
Fizmatgiz, Moscow, 1963.

[3] CONROY, H.
Molecular Schroedinger Equation VIII. A new method for evaluating multidimensional integrals.
J. Chem. Phys., 47, pp. 5307-5318, 1967.

[4] CRANLEY, R. and PATTERSON, T.N.L.
Randomisation of number theoretic methods for multiple integration.
SIAM J. Numer. Anal., 13, pp. 904-914, 1976.

5. Parameters

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integral, n.

Constraint: $1 \leq$ NDIM ≤ 20.

2: VECFUN – SUBROUTINE, supplied by the user. *External Procedure*

VECFUN must evaluate the integrand at a specified set of points.

Its specification is:

```
SUBROUTINE VECFUN (NDIM, X, FV, M)
INTEGER    NDIM, M
real       X(M,NDIM), FV(M)
```

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integral, n.

2: X(M,NDIM) – ***real*** array. *Input*

On entry: the co-ordinates of the m points at which the integrand must be evaluated. X(i,j) contains the jth co-ordinate of the ith point.

3: FV(M) – ***real*** array. *Output*

On exit: FV(i) must contain the value of the integrand of the ith point. i.e. FV(i) = f(X(i,1),X(i,2),...,X(i,NDIM)), for $i = 1,2,...,$M.

4: M – INTEGER. *Input*

On entry: the number of points m at which the integrand is to be evaluated.

VECFUN must be declared as EXTERNAL in the (sub)program from which D01GDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

3: VECREG – SUBROUTINE, supplied by the user. *External Procedure*

VECREG must evaluate the limits of integration in any dimension for a set of points.

Its specification is:

```
SUBROUTINE VECREG (NDIM, X, J, C, D, M)
INTEGER    NDIM, J, M
real       X(M,NDIM), C(M), D(M)
```

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integral, n.

2: X(M,NDIM) – ***real*** array. *Input*

On entry: for $i = 1,2,...,m$, X(i,1), X(i,2), ... ,X(i,j–1) contain the current values of the first j–1 co-ordinates of the ith point, which may be used if necessary in calculating the m values of c_j and d_j.

3: J – INTEGER. *Input*

On entry: the index, j, of the dimension for which the limits of the range of integration are required.

4: C(M) – ***real*** array. *Output*

On exit: C(i) must be set to the lower limit of the range for X(i,j), for $i = 1,2,...,m$.

5: D(M) – ***real*** array. *Output*

On exit: D(i) must be set to the upper limit of the range for X(i,j), for $i = 1,2,...,m$.

6: M – INTEGER. *Input*

On entry: the number of points m at which the limits of integration must be specified.

VECREG must be declared as EXTERNAL in the (sub)program from which D01GDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

4: NPTS – INTEGER. *Input*

On entry: the Korobov rule to be used. There are two alternatives depending on the value of NPTS.

(a) 1 ≤ NPTS ≤ 6.
In this case one of six preset rules is chosen using 2129, 5003, 10007, 20011, 40009 or 80021 points depending on the respective value of NPTS being 1, 2, 3, 4, 5 or 6.

(b) NPTS > 6.
NPTS is the number of actual points to be used with corresponding optimal coefficients supplied in the array VK.

Constraint: NPTS ≥ 1

5: VK(NDIM) – ***real*** array. *Input/Output*

On entry: If NPTS > 6, VK must contain the n optimal coefficients (which may be calculated using D01GYF or D01GZF); if NPTS ≤ 6, VK need not be set.

On exit: if NPTS > 6, VK is unchanged; if NPTS ≤ 6, VK contains the n optimal coefficients used by the preset rule.

6: NRAND – INTEGER. *Input*

On entry: the number of random samples to be generated (generally a small value, say 3 to 5, is sufficient). The estimate, RES, of the value of the integral returned by the routine is then the average of NRAND calculations with different random origin shifts. If NPTS > 6, the total number of integrand evaluations will be NRAND×NPTS. If 1 ≤ NPTS ≤ 6, then the number of integrand evaluations will be NRAND×p, where p is the number of points corresponding to the six preset rules. For reasons of efficiency, these values are calculated a number at a time in VECFUN.

Constraint: NRAND ≥ 1

7: ITRANS – INTEGER. *Input*

On entry: indicates whether the periodising transformation is to be used:

if ITRANS = 0, the transformation is to be used.

if ITRANS ≠ 0, the transformation is to be suppressed (to cover cases where the integrand may already be periodic or where the user desires to specify a particular transformation in the definition of VECFUN).

Suggested value: ITRANS = 0.

8: RES – ***real.*** *Output*

On exit: an estimate of the value of the integral.

9: ERR – ***real.*** *Output*

On exit: the standard error as computed from NRAND sample values. If NRAND = 1, then ERR contains zero.

10: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, NDIM < 1,
or NDIM > 20.

IFAIL = 2

On entry, NPTS < 1.

IFAIL = 3

On entry, NRAND < 1.

7. Accuracy

If NRAND > 1, an estimate of the absolute standard error is given by the value, on exit, of ERR.

8. Further Comments

This routine performs the same computation as the D01GCF. However, the interface has been modified so that it can perform more efficiently on machines with vector processing capabilities. In particular, the routines VECFUN and VECREG must calculate the integrand and limits of integration at a *set* of points. For some problems the amount of time spent in these two subroutines, which must be supplied by the user, may account for a significant part of the total computation time. For this reason it is vital that the user considers the possibilities for vectorization in the code supplied for these two subroutines.

The time taken will be approximately proportional to NRAND×p, where p is the number of points used, but may depend significantly on the efficiency of the code provided by the user in subroutines VECFUN and VECREG.

The exact values of RES and ERR returned by D01GDF will depend (within statistical limits) on the sequence of random numbers generated within the routine by calls to G05CAF. To ensure that the results returned by D01GDF in separate runs are identical, users should call G05CBF immediately before calling D01GDF; to ensure that they are different, call G05CCF.

9. Example

This example calculates the integral

$$\int_0^1\int_0^1\int_0^1\int_0^1 \cos(0.5+2(x_1+x_2+x_3+x_4)-4)\ dx_1dx_2dx_3dx_4.$$

9.1. Program Text

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

```
*     D01GDF Example Program Text
*     Mark 14 Release.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NDIM
      PARAMETER        (NDIM=4)
*     .. Local Scalars ..
      real             ERR, RES
      INTEGER          IFAIL, ITRANS, NPTS, NRAND
*     .. Local Arrays ..
      real             VK(NDIM)
*     .. External Subroutines ..
      EXTERNAL         D01GDF, VECFUN, VECREG
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01GDF Example Program Results'
      WRITE (NOUT,*)
      NPTS = 2
      ITRANS = 0
      NRAND = 4
      IFAIL = 0
*
      CALL D01GDF(NDIM,VECFUN,VECREG,NPTS,VK,NRAND,ITRANS,RES,ERR,IFAIL)
*
      WRITE (NOUT,99999) 'Result = ', RES, ', standard error = ', ERR
      STOP
*
99999 FORMAT (1X,A,F13.5,A,e10.2)
      END
*
      SUBROUTINE VECFUN(NDIM,X,FV,M)
*     .. Scalar Arguments ..
      INTEGER          M, NDIM
*     .. Array Arguments ..
      real             FV(M), X(M,NDIM)
*     .. Local Scalars ..
      INTEGER          I, J
*     .. Intrinsic Functions ..
      INTRINSIC        COS, real
```

```
*     .. Executable Statements ..
      DO 20 I = 1, M
         FV(I) = 0.0e0
   20 CONTINUE
      DO 60 J = 1, NDIM
         DO 40 I = 1, M
            FV(I) = FV(I) + X(I,J)
   40    CONTINUE
   60 CONTINUE
      DO 80 I = 1, M
         FV(I) = COS(0.5e0+2.0e0*FV(I)-real(NDIM))
   80 CONTINUE
      RETURN
      END
*
      SUBROUTINE VECREG(NDIM,X,J,C,D,M)
*     .. Scalar Arguments ..
      INTEGER           J, M, NDIM
*     .. Array Arguments ..
      real              C(M), D(M), X(M,NDIM)
*     .. Local Scalars ..
      INTEGER           I
*     .. Executable Statements ..
      DO 20 I = 1, M
         C(I) = 0.0e0
         D(I) = 1.0e0
   20 CONTINUE
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D01GDF Example Program Results

Result =       0.43999, standard error =   0.18E-05
```

D01GYF – NAG Fortran Library Routine Document

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

1. Purpose

D01GYF calculates the optimal coefficients, for use by D01GCF and D01GDF for prime numbers of points.

2. Specification

```
SUBROUTINE D01GYF (NDIM, NPTS, VK, IFAIL)
INTEGER       NDIM, NPTS, IFAIL
real          VK(NDIM)
```

3. Description

The Korobov procedure [1] for calculating the optimal coefficients $a_1,a_2,...,a_n$ for p-point integration over the n-cube $[0,1]^n$ imposes the constraint

$$a_1 = 1$$

$$a_i = a^{i-1} \ (\mathrm{mod}\, p), \qquad i = 1,2,...,n \tag{1}$$

where p is a prime number and a is an adjustable parameter. This parameter is computed to minimize the error in the integral

$$3^n \int_0^1 dx_1 \ ... \int_0^1 dx_n \prod_{i=1}^{n} (1-2x_i)^2, \tag{2}$$

when computed using the number theoretic rule, and the resulting coefficients can be shown to fit the Korobov definition of optimality.

The computation for large values of p is extremely time consuming (the number of elementary operations varying as p^2) and there is a practical upper limit to the number of points that can be used. Routine D01GZF is computationally more economical in this respect but the associated error is likely to be larger.

4. References

[1] KOROBOV, N.M.
Number Theoretic Methods in Approximate Analysis.
Fizmatgiz, Moscow, 1963.

5. Parameters

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integral, n.

Constraint: NDIM $\geq$ 1.

2: NPTS – INTEGER. *Input*

On entry: the number of points to be used, p.

Constraint: NPTS must be a prime number $\geq$ 5.

3: VK(NDIM) – ***real*** array. *Output*

On exit: the n optimal coefficients.

4: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, NDIM < 1.

IFAIL = 2

On entry, NPTS < 5.

IFAIL = 3

On entry, NPTS is not a prime number.

IFAIL = 4

The precision of the machine is insufficient to perform the computation exactly. Try a smaller value of NPTS, or use an implementation of higher precision.

7. Accuracy

The optimal coefficients are returned as exact integers (though stored in a ***real*** array).

8. Further Comments

The time taken is approximately proportional to p^2 (see Section 3).

9. Example

This example program calculates the Korobov optimal coefficients where the number of dimensions is 4 and the number of points is 631.

9.1. Program Text

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

```
*     D01GYF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NDIM
      PARAMETER        (NDIM=4)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      INTEGER          I, IFAIL, NPTS
*     .. Local Arrays ..
      real             VK(20)
*     .. External Subroutines ..
      EXTERNAL         D01GYF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01GYF Example Program Results'
      NPTS = 631
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'NDIM =', NDIM, ' NPTS =', NPTS
      IFAIL = 0
*
```

```
      CALL D01GYF(NDIM,NPTS,VK,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99998) 'Coefficients =', (VK(I),I=1,NDIM)
      STOP
*
99999 FORMAT (1X,A,I3,A,I6)
99998 FORMAT (1X,A,4F6.0)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D01GYF Example Program Results

 NDIM =  4 NPTS =    631

 Coefficients =    1.  198.   82.  461.
```

D01GZF – NAG Fortran Library Routine Document

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

1. Purpose

D01GZF calculates the optimal coefficients, for use by D01GCF and D01GDF, when the number of points is the product of two primes.

2. Specification

```
SUBROUTINE D01GZF (NDIM, NP1, NP2, VK, IFAIL)
INTEGER        NDIM, NP1, NP2, IFAIL
real           VK(NDIM)
```

3. Description

Korobov [1] gives a procedure for calculating optimal coefficients for p-point integration over the n-cube $[0,1]^n$, when the number of points is

$$p = p_1 p_2 \qquad (1)$$

where p_1 and p_2 are distinct prime numbers.

The advantage of this procedure is that if p_1 is chosen to be the nearest prime integer to p_2^2, then the number of elementary operations required to compute the rule is of the order of $p^{4/3}$ which grows less rapidly than the number of operations required by D01GYF. The associated error is likely to be larger although it may be the only practical alternative for high values of p.

4. References

[1] KOROBOV, N.M.
Number Theoretic Methods in Approximate Analysis.
Fizmatgiz, Moscow, 1963.

5. Parameters

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integral, n.

Constraint: NDIM $\geq$ 1.

2: NP1 – INTEGER. *Input*

On entry: the larger prime factor p_1 of the number of points in the integration rule.

Constraint: NP1 must be a prime number $\geq$ 5.

3: NP2 – INTEGER. *Input*

On entry: the smaller prime factor p_2 of the number of points in the integration rule. For maximum efficiency, p_2^2 should be close to p_1.

Constraint: NP2 must be a prime number such that NP1 > NP2 $\geq$ 2.

4: VK(NDIM) – ***real*** array. *Output*

On exit: the n optimal coefficients.

5: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, NDIM < 1.

IFAIL = 2

On entry, NP1 < 5,
or NP2 < 2,
or NP1 ≤ NP2.

IFAIL = 3

The value NP1×NP2 exceeds the largest integer representable on the machine, and hence the optimal coefficients could not be used in a valid call of D01GCF.

IFAIL = 4

On entry, NP1 is not a prime number.

IFAIL = 5

On entry, NP2 is not a prime number.

IFAIL = 6

The precision of the machine is insufficient to perform the computation exactly. Try smaller values of NP1 or NP2, or use an implementation with higher precision.

7. Accuracy

The optimal coefficients are returned as exact integers (though stored in a ***real*** array).

8. Further Comments

The time taken by the routine grows at least as fast as $(p_1 p_2)^{4/3}$. (See Section 3.)

9. Example

This example program calculates the Korobov optimal coefficients where the number of dimensons is 4 and the number of points is the product of the two prime numbers, 89 and 11.

9.1. Program Text

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

```
*     D01GZF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NDIM
      PARAMETER        (NDIM=4)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      INTEGER          I, IFAIL, NP1, NP2
*     .. Local Arrays ..
      real             VK(NDIM)
*     .. External Subroutines ..
      EXTERNAL         D01GZF
```

```
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01GZF Example Program Results'
      NP1 = 89
      NP2 = 11
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'NDIM =', NDIM, ' NP1 =', NP1, ' NP2 =', NP2
      IFAIL = 0
*
      CALL D01GZF(NDIM,NP1,NP2,VK,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99998) 'Coefficients =', (VK(I),I=1,NDIM)
      STOP
*
99999 FORMAT (1X,A,I3,A,I6,A,I6)
99998 FORMAT (1X,A,4F6.0)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D01GZF Example Program Results

 NDIM =  4 NP1 =     89 NP2 =     11

 Coefficients =    1.  102.  614.  951.
```

D01JAF – NAG Fortran Library Routine Document

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

1. Purpose

D01JAF attempts to evaluate an integral over an n-dimensional sphere ($n = 2, 3$, or 4), to a user specified absolute or relative accuracy, by means of a modified Sag-Szekeres method. The routine can handle singularities on the surface or at the centre of the sphere, and returns an error estimate.

2. Specification

```
      SUBROUTINE D01JAF (F, NDIM, RADIUS, EPSA, EPSR, METHOD, ICOORD, RESULT,
     1                   ESTERR, NEVALS, IFAIL)
      INTEGER            NDIM, METHOD, ICOORD, NEVALS, IFAIL
      real               F, RADIUS, EPSA, EPSR, RESULT, ESTERR
      EXTERNAL           F
```

3. Description

This routine calculates an approximation to the n-dimensional integral

$$I = \int \dots \int_S F(x_1,\dots x_n) dx_1 \dots dx_n, \qquad 2 \le n \le 4,$$

where S is the hypersphere

$$\sqrt{(x_1^2+\dots+x_n^2)} \le \alpha < \infty$$

(the integrand function may also be defined in spherical co-ordinates). The algorithm is based on the Sag-Szekeres method [1], applying the product trapezoidal formula after a suitable radial transformation. An improved transformation technique is developed: depending on the behaviour of the function and on the required accuracy, different transformations can be used, some of which are 'double exponential', as defined by Takahasi and Mori [2]. The resulting technique allows the routine to deal with integrand singularities on the surface or at the centre of the sphere. When the estimated error of the approximation with mesh size h is larger than the tolerated error, the trapezoidal formula with mesh size $h/2$ is calculated. A drawback of this method is the exponential growth of the number of function evaluations in the successive approximations (this number grows with a factor $\approx 2^n$). This introduces the restriction $n \le 4$. Because the convergence rate of the successive approximations is normally better than linear, the error estimate is based on the linear extrapolation of the difference between the successive approximations [3,4]. For further details of the algorithm, see Roose and de Doncker [4].

4. References

[1] SAG, T.W. and SZEKERES, G.
Numerical Evaluation of High-dimensional Integrals.
Math. Comp. 18, pp. 245-253, 1964.

[2] TAKAHASI, H. and MORI, M.
Double Exponential Formulas for Numerical Integration.
Publ. RIMS, Kyoto Univ. 9, pp. 721-741, 1974.

[3] ROBINSON, I and DE DONCKER, E.
Automatic Computation of Improper Integrals over a Bounded or Unbounded Planar Region.
Computing, 27, pp. 89-284, 1981.

[4] ROOSE, D. and DE DONCKER, E.
Automatic Integration over a Sphere.
J. Comp. Appl. Math., 7, pp. 203-224, 1981.

5. Parameters

1: F – ***real*** FUNCTION, supplied by the user. *External Procedure*

F must return the value of the integrand f at a given point.

Its specification is:

```
real FUNCTION F(NDIM, X)
INTEGER       NDIM
real          X(NDIM)
```

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integral, n.

2: X(NDIM) – ***real*** array. *Input*

On entry: the co-ordinates of the point at which the integrand must be evaluated. These co-ordinates are given in Cartesian or spherical polar form according to the value of ICOORD (see below).

F must be declared as EXTERNAL in the (sub)program from which D01JAF is called. Parameters denoted as *Input* must **not** be changed by this procedure. See also Section 8.

2: NDIM – INTEGER. *Input*

On entry: the dimension of the sphere, n.

Constraint: 2 ≤ NDIM ≤ 4.

3: RADIUS – ***real***. *Input*

On entry: the radius of the sphere, α.

Constraint: RADIUS ≥ 0.0.

4: EPSA – ***real***. *Input*

On entry: the requested absolute tolerance. If EPSA < 0.0, its absolute value is used. See Section 7.

5: EPSR – ***real***. *Input*

On entry: the requested relative tolerance. If EPSR < 0.0, its absolute value is used. If EPSR < 10×(***machine precision***), the latter value is used as EPSR by the routine. See Section 7.

6: METHOD – INTEGER. *Input*

On entry: METHOD must specify the transformation to be used by the routine. The choice depends on the behaviour of the integrand and on the required accuracy.

For well-behaved functions and functions with mild singularities on the surface of the sphere only:

low accuracy required: METHOD = 1
high accuracy required: METHOD = 2

for functions with severe singularities on the surface of the sphere only:

low accuracy required: METHOD = 3
high accuracy required: METHOD = 4

(in this case ICOORD must be set to 2, and the function defined in special spherical co-ordinates).

For functions with a singularity at the centre of the sphere (and possibly with singularities on the surface as well):

low accuracy required: METHOD = 5
high accuracy required: METHOD = 6

METHOD = 0 can be used as a default value and is equivalent to METHOD = 1 if EPSR > 10^{-6}, and to METHOD = 2 if EPSR ≤ 10^{-6}.

The distinction between low and high required accuracies, as mentioned above, depends also on the behaviour of the function. Roughly one may assume the critical value of EPSA and EPSR to be 10^{-6}, but the critical value will be smaller for a well-behaved integrand and larger for an integrand with severe singularities.

Suggested value: METHOD = 0.

Constraint: 0 ≤ METHOD ≤ 6. If ICOORD = 2, METHOD = 3 or 4.

7: ICOORD – INTEGER. *Input*

On entry: ICOORD must specify which kind of co-ordinates are used in the user-supplied function F.

ICOORD = 0,

Cartesian co-ordinates x_i, for $i = 1,2,...,n$.

ICOORD = 1,

spherical co-ordinates (see Section 8.2): X(1) = ρ; X(i) = θ_{i-1}, for $i = 2,3,...,n$.

ICOORD = 2,

special spherical polar co-ordinates (see Section 8.3), with the additional transformation $\rho = \alpha - \lambda$: X(1) = $\lambda = \alpha - \rho$; $X(i) = \theta_{i-1}$, for $i = 2,3,...,n$.

Constraint: ICOORD = 0, 1 or 2. If METHOD = 3 or 4, ICOORD = 2.

8: RESULT – ***real.*** *Output*

On exit: the approximation to the integral.

9: ESTERR – ***real.*** *Output*

On exit: an estimate of the modulus of the absolute error.

10: NEVALS – INTEGER. *Output*

On exit: the number of function evaluations used.

11: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.** To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL = 1

The required accuracy cannot be achieved within a limiting number of function evaluations (which is set by the routine).

IFAIL = 2

The required accuracy cannot be achieved because of roundoff error.

IFAIL = 3

The required accuracy cannot be achieved because the maximum accuracy with respect to the machine constants X02AJF and X02AMF has been attained. If this maximum accuracy is rather low (compared with X02AJF), the cause of the problem is a severe singularity on the boundary or at the centre of the sphere. If METHOD = 0, 1 or 2, then setting METHOD = 3 or 4 may help.

IFAIL = 4

On entry, NDIM < 2 or > 4,
or RADIUS < 0.0,
or METHOD < 0 or > 6,
or ICOORD < 0 or > 2,
or ICOORD = 2 and METHOD ≠ 3 or 4,
or METHOD = 3 or 4 and ICOORD ≠ 2.

No calculations have been performed. RESULT and ESTERR are set to 0.0.

7. Accuracy

The user can specify an absolute and/or a relative tolerance, setting EPSA and EPSR. The routine attempts to calculate an approximation RESULT such that

$$|I-\text{RESULT}| \leq \max\{\text{EPSA},\text{EPSR}\times|I|\}.$$

If $0 \leq \text{IFAIL} \leq 3$, ESTERR returns an estimate of, but not necessarily a bound for, $|I-\text{RESULT}|$.

8. Further Comments

8.1. Timing

Timing depends on the integrand and the accuracy required.

8.2. Spherical Polar Co-ordinates

Cartesian co-ordinates are related to the spherical polar co-ordinates by:

$$\begin{aligned}
x_1 &= \rho.\sin\theta_1 \;\ldots\; \sin\theta_{n-2}.\sin\theta_{n-1} \\
x_2 &= \rho.\sin\theta_1 \;\ldots\; \sin\theta_{n-2}.\cos\theta_{n-1} \\
x_3 &= \rho.\sin\theta_1 \;\ldots\; \cos\theta_{n-2} \\
&\;\;. \\
&\;\;. \\
&\;\;. \\
x_n &= \rho.\cos\theta_1
\end{aligned}$$

where $0 < \theta_i < \pi$, for $i = 1,2,\ldots,n-2$ and $0 < \theta_{n-1} < 2\pi$.

8.3. Machine Dependencies

As a consequence of the transformation technique, the severity of the singularities which can be handled by the routine depends on the precision and range of ***real*** numbers on the machine. METHOD = 3 or 4 must be used when the singularity on the surface is 'severe' in view of the requested accuracy and machine precision. In practice one has to set METHOD = 3 or 4 if D01JAF terminates with IFAIL = 3 when called with METHOD = 0, 1 or 2.

When integrating a function with a severe singular behaviour on the surface of the sphere, the additional transformation $\rho = \alpha - \lambda$ helps to avoid the loss of significant figures due to round-off error in the calculation of the integration nodes which are very close to the surface. For these points, the value of λ can be computed more accurately than the value of ρ. Naturally, care must be taken that the function subprogram does not contain expressions of the form $\alpha - \lambda$, which could cause a large round-off error in the calculation of the integrand at the boundary of the sphere.

Care should be taken to avoid underflow and/or overflow problems in the function subprogram, because some of the integration nodes used by D01JAF may be very close to the surface or to the centre of the sphere.

Example:

suppose the function

$$f(\varrho) = (1-\varrho^2)^{-0.7}$$

is to be integrated over the unit sphere, with METHOD = 3 or 4. Then ICOORD should be set to 2; the transformation $\varrho = 1-\lambda$ gives $f(\varrho) = (2\lambda-\lambda^2)^{-0.7}$; and F could be coded thus:

```
      F = 1.0
      A = X(1)
      IF (A.GT.0.0) F = 1.0/(A*(2.0-A))**0.7
      RETURN
```

Note that D01JAF ensures that $\lambda = X(1) > X02AMF$, but underflow could occur in the computation of λ^2.

9. Example

The program following evaluates the integrals

$$\int \cdots \int_S \frac{1}{\sqrt{1-\varrho^2}}\, dx_1 \ldots dx_n$$

where $\varrho = \sqrt{\sum_{i=1}^{n} x_i^2}$, and S is the unit sphere of dimension $n = 2$ or 4.

The exact values (to 12 decimal places) are 6.28318530718 and 13.1594725348.

9.1. Program Text

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

```
*     D01JAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             EPSA, EPSR, ESTERR, RADIUS, RELEST, RESULT
      INTEGER          ICOORD, IFAIL, ITEST, METHOD, NDIM, NEVALS
*     .. Local Arrays ..
      INTEGER          ND(2)
*     .. External Functions ..
      real             F
      EXTERNAL         F
*     .. External Subroutines ..
      EXTERNAL         D01JAF
*     .. Data statements ..
      DATA             ND/2, 4/
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01JAF Example Program Results'
      RADIUS = 1.0e0
      METHOD = 0
      ICOORD = 1
      EPSA = 0.0e0
      EPSR = 0.5e-4
      DO 20 ITEST = 1, 2
         NDIM = ND(ITEST)
         IFAIL = 1
*
```

```
         CALL D01JAF(F,NDIM,RADIUS,EPSA,EPSR,METHOD,ICOORD,RESULT,
     +               ESTERR,NEVALS,IFAIL)
*
         WRITE (NOUT,*)
         IF (IFAIL.NE.0) THEN
            WRITE (NOUT,99999) 'IFAIL =', IFAIL
            WRITE (NOUT,*)
         END IF
         IF (IFAIL.LE.3) THEN
            RELEST = ESTERR/RESULT
            WRITE (NOUT,99999) 'Dimension of the sphere       =', NDIM
            WRITE (NOUT,99998) 'Requested relative tolerance  =', EPSR
            WRITE (NOUT,99997) 'Approximation to the integral =', RESULT
            WRITE (NOUT,99999) 'No. of function evaluations   =', NEVALS
            WRITE (NOUT,99998) 'Estimated relative error      =', RELEST
         END IF
   20 CONTINUE
      STOP
*
99999 FORMAT (1X,A,I5)
99998 FORMAT (1X,A,e9.2)
99997 FORMAT (1X,A,F9.5)
      END
*
      real FUNCTION F(NDIM,X)
*     .. Scalar Arguments ..
      INTEGER         NDIM
*     .. Array Arguments ..
      real            X(NDIM)
*     .. Local Scalars ..
      real            A, RHO
*     .. Intrinsic Functions ..
      INTRINSIC       SQRT
*     .. Executable Statements ..
      RHO = X(1)
      F = 0.0e0
      A = (1.0e0-RHO)*(1.0e0+RHO)
      IF (A.NE.0.0e0) F = 1.0e0/SQRT(A)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D01JAF Example Program Results

 Dimension of the sphere       =    2
 Requested relative tolerance  = 0.50E-04
 Approximation to the integral =  6.28319
 No. of function evaluations   =  193
 Estimated relative error      = 0.31E-04

 Dimension of the sphere       =    4
 Requested relative tolerance  = 0.50E-04
 Approximation to the integral = 13.16004
 No. of function evaluations   = 2873
 Estimated relative error      = 0.40E-04
```

D01PAF – NAG Fortran Library Routine Document

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

1. Purpose

D01PAF returns a sequence of approximations to the integral of a function over a multi-dimensional simplex, together with an error estimate for the last approximation.

2. Specification

```
      SUBROUTINE D01PAF (NDIM, VERTEX, IV1, IV2, FUNCTN, MINORD, MAXORD,
     1                   FINVLS, ESTERR, IFAIL)
      INTEGER          NDIM, IV1, IV2, MINORD, MAXORD, IFAIL
      real             VERTEX(IV1,IV2), FUNCTN, FINVLS(MAXORD), ESTERR
      EXTERNAL         FUNCTN
```

3. Description

The subroutine computes a sequence of approximations FINVLS(j), for j = MINORD+1,MINORD+2,...,MAXORD, to an integral

$$\int_S f(x_1,x_2,...,x_n)dx_1\ dx_2\ ...\ dx_n$$

where S is an n-dimensional simplex defined in terms of its $n+1$ vertices. FINVLS(j) is an approximation which will be exact (except for rounding errors) whenever the integrand is a polynomial of total degree $2j-1$ or less.

The type of method used has been described by Grundmann and Moller [1], and is implemented in an extrapolated form using the theory from de Doncker [2].

4. References

[1] GRUNDMANN, A. and MOLLER, H.M.
Invariant Integration Formulas for the n-simplex by Combinatorial Methods.
SIAM J. Numer. Anal., 15, pp. 282-290, 1978.

[2] DE DONCKER, E.
New Euler Maclaurin Expansions and their Application to Quadrature over the s-dimensional Simplex.
Math. Comp., 33, pp. 1003-1018, 1979.

5. Parameters

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integral, n.

Constraint: NDIM ≥ 2.

2: VERTEX(IV1,IV2) – ***real*** array. *Input/Output*

On entry: VERTEX(i,j) must be set to the jth component of the ith vertex for the simplex integration region, for $i = 1,2,...,n+1$; $j = 1,2,...,n$. If MINORD > 0, VERTEX must be unchanged since the previous call of D01PAF.

On exit: these value are unchanged. The rest of the array VERTEX is used for workspace and contains information to be used if another call of D01PAF is made with MINORD > 0. In particular VERTEX$(n+1,2n+2)$ contains the volume of the simplex.

3: IV1 – INTEGER. *Input*

On entry: the first dimension of the array VERTEX as declared in the (sub)program from which D01PAF is called.

Constraint: IV1 ≥ NDIM + 1.

4: IV2 – INTEGER. *Input*

On entry: the second dimension of the array VERTEX as declared in the (sub)program from which D01PAF is called.

Constraint: IV2 ≥ 2×(NDIM+1).

5: FUNCTN – ***real*** FUNCTION, supplied by the user. *External Procedure*

FUNCTN must return the value of the integrand f at a given point.

Its specification is:

```
real FUNCTION FUNCTN(NDIM, X)
INTEGER       NDIM
real          X(NDIM)
```

1: NDIM – INTEGER. *Input*

On entry: the number of dimensions of the integral, n.

2: X(NDIM) – ***real*** array. *Input*

On entry: the co-ordinates of the point at which the integrand must be evaluated.

FUNCTN must be declared as EXTERNAL in the (sub)program from which D01PAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: MINORD – INTEGER. *Input/Output*

On entry: MINORD must specify the highest order of the approximations currently available in the array FINVLS. MINORD = 0 indicates an initial call; MINORD > 0 indicates that FINVLS(1),FINVLS(2),...,FINVLS(MINORD) have already been computed in a previous call of D01PAF.

Constraint: MINORD ≥ 0.

On exit: MINORD = MAXORD.

7: MAXORD – INTEGER. *Input*

On entry: the highest order of approximation to the integral to be computed.

Constraint: MAXORD > MINORD.

8: FINVLS(MAXORD) – ***real*** array. *Input/Output*

On entry: FINVLS(1),FINVLS(2),...,FINVLS(MINORD) must contain approximations to the integral previously computed by D01PAF.

On exit: FINVLS contains these values unchanged, and the newly computed values FINVLS(MINORD+1),FINVLS(MINORD+2),...,FINVLS(MAXORD). FINVLS(j) is an approximation to the integral of polynomial degree $2j - 1$.

9: ESTERR – ***real***. *Output*

On exit: an absolute error estimate for FINVLS(MAXORD).

10: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, NDIM < 2,
or IV1 < NDIM + 1,
or IV2 < 2×(NDIM+1),
or MINORD < 0,
or MAXORD ≤ MINORD.

IFAIL = 2

The volume of the simplex integration region (computed as a determinant by F03AAF) is too large or too small to be representable in the machine.

7. Accuracy

An absolute error estimate is output through the parameter ESTERR.

8. Further Comments

The running time for D01PAF will usually be dominated by the time used to evaluate the integrand FUNCTN. The maximum time that could be used by D01PAF will be approximately given by

$$T \times \frac{(\text{MAXORD}+\text{NDIM})!}{(\text{MAXORD}-1)!(\text{NDIM}+1)!}$$

where T is the time needed for one call of FUNCTN.

9. Example

A program demonstrating the use of the subroutine with the integral

$$\int_0^1 \int_0^{1-x} \int_0^{1-x-y} \exp(x+y+z)\cos(x+y+z) \; dz \; dy \; dx = \tfrac{1}{4}$$

is given below.

9.1. Program Text

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

```
*     D01PAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NDIM, IV1, IV2, MXORD
      PARAMETER        (NDIM=3,IV1=NDIM+1,IV2=2*(NDIM+1),MXORD=5)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             ESTERR
      INTEGER          IFAIL, J, K, MAXORD, MINORD, NEVALS
*     .. Local Arrays ..
      real             FINVLS(MXORD), VERTEX(IV1,IV2)
*     .. External Functions ..
      real             FUNCTN
      EXTERNAL         FUNCTN
*     .. External Subroutines ..
      EXTERNAL         D01PAF
```

```
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01PAF Example Program Results'
      DO 40 J = 1, IV1
         DO 20 K = 1, NDIM
            VERTEX(J,K) = 0.0e0
   20    CONTINUE
         IF (J.GT.1) VERTEX(J,J-1) = 1.0e0
   40 CONTINUE
      MINORD = 0
      NEVALS = 1
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     + 'MAXORD   Estimated      Estimated        Integrand'
      WRITE (NOUT,*)
     + '           value         accuracy        evaluations'
      DO 60 MAXORD = 1, MXORD
         IFAIL = 0
*
         CALL D01PAF(NDIM,VERTEX,IV1,IV2,FUNCTN,MINORD,MAXORD,FINVLS,
     +               ESTERR,IFAIL)
*
         WRITE (NOUT,99999) MAXORD, FINVLS(MAXORD), ESTERR, NEVALS
         NEVALS = (NEVALS*(MAXORD+NDIM+1))/MAXORD
   60 CONTINUE
      STOP
*
99999 FORMAT (1X,I4,F13.5,e16.3,I15)
      END
*
      real FUNCTION FUNCTN(NDIM,X)
*     .. Scalar Arguments ..
      INTEGER              NDIM
*     .. Array Arguments ..
      real                 X(NDIM)
*     .. Intrinsic Functions ..
      INTRINSIC            COS, EXP
*     .. Executable Statements ..
      FUNCTN = EXP(X(1)+X(2)+X(3))*COS(X(1)+X(2)+X(3))
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D01PAF Example Program Results

MAXORD   Estimated      Estimated        Integrand
           value         accuracy       evaluations
   1      0.25816       0.258E+00              1
   2      0.25011       0.806E-02              5
   3      0.25000       0.107E-03             15
   4      0.25000       0.410E-06             35
   5      0.25000       0.173E-08             70
```

Chapter D02 – Ordinary Differential Equations

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
D02AGF	2	ODEs, boundary value problem, shooting and matching technique, allowing interior matching point, general parameters to be determined
D02BGF	7	ODEs, IVP, Runge–Kutta–Merson method, until a component attains given value (simple driver)
D02BHF	7	ODEs, IVP, Runge–Kutta–Merson method, until function of solution is zero (simple driver)
D02BJF	18	ODEs, IVP, Runge–Kutta method, until function of solution is zero, integration over range with intermediate output (simple driver)
D02CJF	13	ODEs, IVP, Adams method, until function of solution is zero, intermediate output (simple driver)
D02EJF	12	ODEs, stiff IVP, BDF method, until function of solution is zero, intermediate output (simple driver)
D02GAF	8	ODEs, boundary value problem, finite difference technique with deferred correction, simple nonlinear problem
D02GBF	8	ODEs, boundary value problem, finite difference technique with deferred correction, general linear problem
D02HAF	8	ODEs, boundary value problem, shooting and matching, boundary values to be determined
D02HBF	8	ODEs, boundary value problem, shooting and matching, general parameters to be determined
D02JAF	8	ODEs, boundary value problem, collocation and least-squares, single *n*th order linear equation
D02JBF	8	ODEs, boundary value problem, collocation and least-squares, system of 1st order linear equations
D02KAF	7	2nd order Sturm–Liouville problem, regular system, finite range, eigenvalue only
D02KDF	7	2nd order Sturm–Liouville problem, regular/singular system, finite/infinite range, eigenvalue only, user-specified break-points
D02KEF	8	2nd order Sturm–Liouville problem, regular/singular system, finite/infinite range, eigenvalue and eigenfunction, user-specified break-points
D02LAF	13	2nd order ODEs, IVP, Runge–Kutta–Nystrom method
D02LXF	13	2nd order ODEs, IVP, set-up for D02LAF
D02LYF	13	2nd order ODEs, IVP, diagnostics for D02LAF
D02LZF	13	2nd order ODEs, IVP, interpolation for D02LAF
D02MVF	14	ODEs, IVP, DASSL method, set-up for D02M-N routines
D02MZF	14	ODEs, IVP, interpolation for D02M-N routines, natural interpolant
D02NBF	12	Explicit ODEs, stiff IVP, full Jacobian (comprehensive)
D02NCF	12	Explicit ODEs, stiff IVP, banded Jacobian (comprehensive)
D02NDF	12	Explicit ODEs, stiff IVP, sparse Jacobian (comprehensive)
D02NGF	12	Implicit/algebraic ODEs, stiff IVP, full Jacobian (comprehensive)
D02NHF	12	Implicit/algebraic ODEs, stiff IVP, banded Jacobian (comprehensive)
D02NJF	12	Implicit/algebraic ODEs, stiff IVP, sparse Jacobian (comprehensive)
D02NMF	12	Explicit ODEs, stiff IVP (reverse communication, comprehensive)
D02NNF	12	Implicit/algebraic ODEs, stiff IVP (reverse communication, comprehensive)
D02NRF	12	ODEs, IVP, for use with D02M-N routines, sparse Jacobian, enquiry routine

D02NSF	12	ODEs, IVP, for use with D02M-N routines, full Jacobian, linear algebra set-up
D02NTF	12	ODEs, IVP, for use with D02M-N routines, banded Jacobian, linear algebra set-up
D02NUF	12	ODEs, IVP, for use with D02M-N routines, sparse Jacobian, linear algebra set-up
D02NVF	12	ODEs, IVP, BDF method, set-up for D02M-N routines
D02NWF	12	ODEs, IVP, Blend method, set-up for D02M-N routines
D02NXF	12	ODEs, IVP, sparse Jacobian, linear algebra diagnostics, for use with D02M-N routines
D02NYF	12	ODEs, IVP, integrator diagnostics, for use with D02M-N routines
D02NZF	12	ODEs, IVP, set-up for continuation calls to integrator, for use with D02M-N routines
D02PCF	16	ODEs, IVP, Runge–Kutta method, integration over range with output
D02PDF	16	ODEs, IVP, Runge–Kutta method, integration over one step
D02PVF	16	ODEs, IVP, set-up for D02PCF and D02PDF
D02PWF	16	ODEs, IVP, resets end of range for D02PDF
D02PXF	16	ODEs, IVP, interpolation for D02PDF
D02PYF	16	ODEs, IVP, integration diagnostics for D02PCF and D02PDF
D02PZF	16	ODEs, IVP, error assessment diagnostics for D02PCF and D02PDF
D02QFF	13	ODEs, IVP, Adams method with root-finding (forward communication, comprehensive)
D02QGF	13	ODEs, IVP, Adams method with root-finding (reverse communication, comprehensive)
D02QWF	13	ODEs, IVP, set-up for D02QFF and D02QGF
D02QXF	13	ODEs, IVP, diagnostics for D02QFF and D02QGF
D02QYF	13	ODEs, IVP, root-finding diagnostics for D02QFF and D02QGF
D02QZF	13	ODEs, IVP, interpolation for D02QFF or D02QGF
D02RAF	8	ODEs, general nonlinear boundary value problem, finite difference technique with deferred correction, continuation facility
D02SAF	8	ODEs, boundary value problem, shooting and matching technique, subject to extra algebraic equations, general parameters to be determined
D02TGF	8	nth order linear ODEs, boundary value problem, collocation and least-squares
D02TKF	17	ODEs, general nonlinear boundary value problem, collocation technique
D02TVF	17	ODEs, general nonlinear boundary value problem, set-up for D02TKF
D02TXF	17	ODEs, general nonlinear boundary value problem, continuation facility for D02TKF
D02TYF	17	ODEs, general nonlinear boundary value problem, interpolation for D02TKF
D02TZF	17	ODEs, general nonlinear boundary value problem, diagnostics for D02TKF
D02XJF	12	ODEs, IVP, interpolation for D02M-N routines, natural interpolant
D02XKF	12	ODEs, IVP, interpolation for D02M-N routines, C_1 interpolant
D02ZAF	12	ODEs, IVP, weighted norm of local error estimate for D02M-N routines

Chapter D02

Ordinary Differential Equations

Contents

1 Scope of the Chapter **2**

2 Background to the Problems **2**
- 2.1 Initial Value Problems 3
- 2.2 Boundary Value Problems 3
  - 2.2.1 Collocation methods 4
  - 2.2.2 Shooting methods 4
  - 2.2.3 Finite-difference methods 4
- 2.3 Chebyshev Collocation for Linear Differential Equations 4
- 2.4 Eigenvalue Problems 4

3 Recommendations on Choice and Use of Available Routines **5**
- 3.1 Initial Value Problems 5
  - 3.1.1 Runge–Kutta routines 5
  - 3.1.2 Adams routines 5
  - 3.1.3 BDF routines 6
  - 3.1.4 Runge–Kutta–Nystrom routines 6
- 3.2 Boundary Value Problems 6
  - 3.2.1 Collocation methods 6
  - 3.2.2 Shooting methods 6
  - 3.2.3 Finite-difference methods 7
- 3.3 Chebyshev Collocation Method 7
- 3.4 Eigenvalue Problems 7
- 3.5 Summary of Recommended Routines 8

4 Routines Withdrawn or Scheduled for Withdrawal **9**

5 References **9**

1 Scope of the Chapter

This chapter is concerned with the numerical solution of ordinary differential equations. There are two main types of problem, those in which all boundary conditions are specified at one point (initial value problems), and those in which the boundary conditions are distributed between two or more points (boundary value problems and eigenvalue problems). Routines are available for initial value problems, two-point boundary value problems and Sturm–Liouville eigenvalue problems.

2 Background to the Problems

For most of the routines in this chapter a system of ordinary differential equations must be written in the form

$$y_1' = f_1(x, y_1, y_2, \ldots, y_n),$$

$$y_2' = f_2(x, y_1, y_2, \ldots, y_n),$$

$$\ldots\ldots$$

$$y_n' = f_n(x, y_1, y_2, \ldots, y_n),$$

that is the system must be given in first-order form. The n dependent variables (also, the solution) $y_1, y_2, \ldots, y_n$ are functions of the independent variable x, and the differential equations give expressions for the first derivatives $y_i' = \frac{dy_i}{dx}$ in terms of x and $y_1, y_2, \ldots, y_n$. For a system of n first-order equations, n associated boundary conditions are usually required to define the solution.

A more general system may contain derivatives of higher order, but such systems can almost always be reduced to the first-order form by introducing new variables. For example, suppose we have the third-order equation

$$z''' + zz'' + k(l - z'^2) = 0.$$

We write $y_1 = z$, $y_2 = z'$, $y_3 = z''$, and the third-order equation may then be written as the system of first-order equations

$$y_1' = y_2$$

$$y_2' = y_3$$

$$y_3' = -y_1y_3 - k(l - y_2^2).$$

For this system $n = 3$ and we require 3 boundary conditions in order to define the solution. These conditions must specify values of the dependent variables at certain points. For example, we have an **initial value problem** if the conditions are:

$$\begin{array}{lll} y_1 = 0 & \text{at} & x = 0 \\ y_2 = 0 & \text{at} & x = 0 \\ y_3 = 0.1 & \text{at} & x = 0. \end{array}$$

These conditions would enable us to integrate the equations numerically from the point $x = 0$ to some specified end-point. We have a **boundary value problem** if the conditions are:

$$\begin{array}{lll} y_1 = 0 & \text{at} & x = 0 \\ y_2 = 0 & \text{at} & x = 0 \\ y_2 = 1 & \text{at} & x = 10. \end{array}$$

These conditions would be sufficient to define a solution in the range $0 \le x \le 10$, but the problem could not be solved by direct integration (see Section 2.2). More general boundary conditions are permitted in the boundary value case.

It is sometimes advantageous to solve higher-order systems directly. In particular, there is an initial value routine to solve a system of second-order ordinary differential equations of the special form

$$y_1'' = f_1(x, y_1, y_2, \ldots, y_n),$$

$$y_2'' = f_2(x, y_1, y_2, \ldots, y_n),$$

$$\ldots\ldots$$

$$y_n'' = f_n(x, y_1, y_2, \ldots, y_n).$$

For this second-order system initial values of the derivatives of the dependent variables, y_i', for $i = 1, 2, \ldots, n$, are required.

There is also a boundary value routine that can treat directly a mixed order system of ordinary differential equations.

There is a broader class of initial value problems known as differential algebraic systems which can be treated. Such a system may be defined as

$$\begin{aligned} y' &= f(x, y, z) \\ 0 &= g(x, y, z) \end{aligned}$$

where y and f are vectors of length n and g and z are vectors of length m. The functions g represent the algebraic part of the system.

In addition implicit systems can also be solved, that is systems of the form

$$A(x, y)y' = f(x, y)$$

where A is a matrix of functions; such a definition can also incorporate algebraic equations. Note that general systems of this form may contain higher-order derivatives and that they can usually be transformed to first-order form, as above.

2.1 Initial Value Problems

To solve first-order systems, initial values of the dependent variables y_i, for $i = 1, 2, \ldots, n$, must be supplied at a given point, a. Also a point, b, at which the values of the dependent variables are required, must be specified. The numerical solution is then obtained by a step-by-step calculation which approximates values of the variables y_i, for $i = 1, 2, \ldots, n$, at finite intervals over the required range $[a, b]$. The routines in this chapter adjust the step length automatically to meet specified accuracy tolerances. Although the accuracy tests used are reliable over each step individually, in general an accuracy requirement cannot be guaranteed over a long range. For many problems there may be no serious accumulation of error, but for unstable systems small perturbations of the solution will often lead to rapid divergence of the calculated values from the true values. A simple check for stability is to carry out trial calculations with different tolerances; if the results differ appreciably the system is probably unstable. Over a short range, the difficulty may possibly be overcome by taking sufficiently small tolerances, but over a long range it may be better to try to reformulate the problem.

A special class of initial value problems are those for which the solutions contain rapidly decaying transient terms. Such problems are called **stiff**; an alternative way of describing them is to say that certain eigenvalues of the Jacobian matrix $\left(\frac{\partial f_i}{\partial y_j}\right)$ have large negative real parts when compared to others. These problems require special methods for efficient numerical solution; the methods designed for non-stiff problems when applied to stiff problems tend to be very slow, because they need small step lengths to avoid numerical instability. A full discussion is given in Hall and Watt [9] and a discussion of the methods for stiff problems is given in Berzins *et al.* [4].

2.2 Boundary Value Problems

In general, a system of nonlinear differential equations with boundary conditions at two or more points cannot be guaranteed to have a solution. The solution, if it exists, has to be determined iteratively. A comprehensive treatment of the numerical solution of boundary value problems can be found in [1] and [10]. The methods for this chapter are discussed in [3], [2] and [7].

2.2.1 Collocation methods

In the collocation method, the solution components are approximated by piecewise polynomials on a mesh. The coefficients of the polynomials form the unknowns to be computed. The approximation to the solution must satisfy the boundary conditions and the differential equations at collocation points in each mesh subinterval. A modified Newton method is used to solve the nonlinear equations. The mesh is refined by trying to equidistribute the estimated error over the whole interval. An initial estimate of the solution across the mesh is required.

2.2.2 Shooting methods

In the shooting method, the unknown boundary values at the initial point are estimated to form an initial value problem, and the equations are then integrated to the final point. At the final point the computed solution and the known boundary conditions should be equal. The condition for equality gives a set of nonlinear equations for the estimated values, which can be solved by Newton's method or one of its variants. The iteration cannot be guaranteed to converge, but it is usually successful if:

- the system has a solution,
- the system is not seriously unstable or very stiff for step-by-step solution, and
- good initial estimates can be found for the unknown boundary conditions.

It may be necessary to simplify the problem and carry out some preliminary calculations, in order to obtain suitable starting values. A fuller discussion is given in Chapters 16, 17 and 18 of Hall and Watt [9], Chapter 11 of Gladwell and Sayers [8] and Chapter 8 of Childs *et al.* [5].

2.2.3 Finite-difference methods

If a boundary value problem seems insoluble by the above methods and a good estimate for the solution of the problem is known at all points of the range then a finite-difference method may be used. Finite-difference equations are set up on a mesh of points and estimated values for the solution at the grid points are chosen. Using these estimated values as starting values a Newton iteration is used to solve the finite-difference equations. The accuracy of the solution is then improved by deferred corrections or the addition of points to the mesh or a combination of both. The method does not suffer from the difficulties associated with the shooting method but good initial estimates of the solution may be required in some cases and the method is unlikely to be successful when the solution varies very rapidly over short ranges. A discussion is given in Chapters 9 and 11 of Gladwell and Sayers [8] and Chapter 4 of Childs *et al.* [5].

2.3 Chebyshev Collocation for Linear Differential Equations

The collocation method gives a different approach to the solution of ordinary differential equations. It can be applied to problems of either initial value or boundary value type. Suppose the approximate solution is represented in polynomial form, say as a series of Chebyshev polynomials. The coefficients may be determined by matching the series to the boundary conditions, and making it satisfy the differential equation at a number of selected points in the range. The calculation is straightforward for linear differential equations (nonlinear equations may also be solved by an iterative technique based on linearisation). The result is a set of Chebyshev coefficients, from which the solution may be evaluated at any point using E02AKF. A fuller discussion is given in Chapter 24 of Childs *et al.* [5] and Chapter 11 of Gladwell and Sayers [8].

This method can be useful for obtaining approximations to standard mathematical functions. For example, suppose we require values of the Bessel function $J_{\frac{1}{3}}(x)$ over the range (0,5), for use in another calculation. We solve the Bessel differential equation by collocation and obtain the Chebyshev coefficients of the solution, which we can use to construct a function for $J_{\frac{1}{3}}(x)$. (Note that routines for many common standard functions are already available in the NAG Library, Chapter S).

2.4 Eigenvalue Problems

Sturm–Liouville problems of the form

$$(p(x)y')' + q(x,\lambda)y = 0$$

with appropriate boundary conditions given at two points, can be solved by a Scaled Prüfer method. In this method the differential equation is transformed to another which can be solved for a specified eigenvalue by a shooting method. A discussion is given in Chapter 11 of Gladwell and Sayers [8] and a complete description is given in Pryce [11]. Some more general eigenvalue problems can be solved by the methods described in Section 2.2.

3 Recommendations on Choice and Use of Available Routines

Note. Refer to the Users' Note for your implementation to check that a routine is available.

There are no routines which deal directly with COMPLEX equations. These may however be transformed to larger systems of real equations of the required form. Split each equation into its real and imaginary parts and solve for the real and imaginary parts of each component of the solution. Whilst this process doubles the size of the system and may not always be appropriate it does make available for use the full range of routines provided presently.

3.1 Initial Value Problems

In general, for non-stiff first-order systems, Runge–Kutta (RK) routines should be used. For the usual requirement of integrating across a range the appropriate routines are D02PVF and D02PCF; D02PVF is a setup routine for D02PCF. For more complex tasks there are a further five related routines, D02PDF, D02PWF, D02PXF, D02PYF and D02PZF. When a system is to be integrated over a long range or with relatively high accuracy requirements the variable-order, variable-step Adams codes may be more efficient. The appropriate routine in this case is D02CJF. For more complex tasks using an Adams code there are a further six related routines: D02QFF, D02QGF, D02QXF, D02QWF, D02QYF and D02QZF.

For stiff systems, that is those which usually contain rapidly decaying transient components, the Backward Differentiation Formula (BDF) variable-order, variable-step codes should be used. The appropriate routine in this case is D02EJF. For more complex tasks using a BDF code there are a collection of routines in the D02M–D02N Subchapter. These routines can treat implicit differential algebraic systems and contain methods alternative to BDF techniques which may be appropriate in some circumstances.

If users are not sure how to classify a problem, they are advised to perform some preliminary calculations with D02PCF, which can indicate whether the system is stiff. We also advise performing some trial calculations with D02PCF (RK), D02CJF (Adams) and D02EJF (BDF) so as to determine which type of routine is best applied to the problem. The conclusions should be based on the computer time used and the number of evaluations of the derivative function f_i. See Gladwell [6] for more details.

For second-order systems of the special form described in Section 2 the Runge–Kutta–Nystrom (RKN) routine D02LAF should be used.

3.1.1 Runge–Kutta routines

The basic RK routine is D02PDF which takes one integration step at a time. An alternative is D02PCF which provides output at user-specified points. The initialisation of either D02PCF or D02PDF and the setting of optional inputs, including choice of method, is made by a call to the setup routine D02PVF. Optional output information about the integration and about error assessment, if selected, can be obtained by calls to the diagnostic routines D02PYF and D02PZF respectively. D02PXF may be used to interpolate on information produced by D02PDF to give solution and derivative values between the integration points. D02PWF may be used to reset the end of the integration range whilst integrating using D02PDF.

There is a simple driving routine D02BJF which integrates a system over a range and, optionally, computes intermediate output and/or determines the position where a specified function of the solution is zero.

3.1.2 Adams routines

The general Adams variable-order variable-step routine is D02QFF which provides a choice of automatic error control and the option of a sophisticated root-finding technique. Reverse communication for both the differential equation and root definition function is provided in D02QGF, which otherwise has the same facilities as D02QFF. A reverse communication routine makes a return to the calling (sub)program for evaluations of equations rather than calling a user-supplied procedure. The initialisation of either of D02QFF and D02QGF and the setting of optional inputs is made by a call to the setup routine D02QWF. Optional output information about the integration and any roots detected can be obtained by calls to the diagnostic routines D02QXF and D02QYF respectively. D02QZF may be used to interpolate on information produced by D02QFF or D02QGF to give solution and derivative values between the integration points.

There is a simple driving routine D02CJF which integrates a system over a range and, optionally, computes intermediate output and/or determines the position where a specified function of the solution is zero.

3.1.3 BDF routines

General routines for explicit and implicit ordinary differential equations with a wide range of options for integrator choice and special forms of numerical linear algebra are provided in the D02M–D02N Subchapter. A separate document describing the use of this subchapter is given immediately before the routines of the subchapter.

There is a simple driving routine D02EJF which integrates a system over a range and, optionally, computes intermediate output and/or determines the position where a specified function of the solution is zero. It has a specification similar to the Adams routine D02CJF except that to solve the equations arising in the BDF method an approximation to the Jacobian $\left(\frac{\partial f_i}{\partial y_j}\right)$ is required. This approximation can be calculated internally but the user may supply an analytic expression. In most cases supplying a correct analytic expression will reduce the amount of computer time used.

3.1.4 Runge–Kutta–Nystrom routines

The Runge–Kutta–Nystrom routine D02LAF uses either a low- or high-order method (chosen by the user). The choice of method and error control and the setting of optional inputs is made by a call to the setup routine D02LXF. Optional output information about the integration can be obtained by a call to the diagnostic routine D02LYF. When the low-order method has been employed D02LZF may be used to interpolate on information produced by D02LAF to give solution and derivative values between the integration points.

3.2 Boundary Value Problems

In general, for a nonlinear system of mixed order with separated boundary conditions, the collocation method (D02TKF and its associated routines) can be used. Problems of a more general nature can often be transformed into a suitable form for treatment by D02TKF, for example nonseparated boundary conditions or problems with unknown parameters (see Section 8 of D02TVF for details).

For simple boundary value problems with assigned boundary values the user may prefer to use a code based on the shooting method or finite difference method for which there are routines with simple calling sequences (D02HAF and D02GAF).

For difficult boundary value problems, where the user needs to exercise some control over the calculation, and where the collocation method proves unsuccessful, the user may wish to try the alternative methods of shooting (D02SAF) or finite-differences (D02RAF).

Note that it is not possible to make a fully automatic boundary value routine, and the user should be prepared to experiment with different starting values or a different routine if the problem is at all difficult.

3.2.1 Collocation methods

The collocation routine D02TKF solves a nonlinear system of mixed order boundary value problems with separated boundary conditions. The initial mesh and accuracy requirements must be specified by a call to the setup routine D02TVF. Optional output information about the final mesh and estimated maximum error can be obtained by a call to the diagnostic routine D02TZF. The solution anywhere on the mesh can be computed by a call to the interpolation routine D02TYF. If D02TKF is being used to solve a sequence of related problems then the continuation routine D02TXF should also be used.

3.2.2 Shooting methods

D02HAF may be used for simple boundary value problems, where the unknown parameters are the missing boundary conditions. More general boundary value problems may be handled by using D02HBF.

This routine allows for a generalised parameter structure, and is fairly complicated. The older routine D02AGF has been retained for use when an interior matching-point is essential; otherwise the newer routine D02HBF should be preferred.

For particularly complicated problems where, for example, the parameters must be constrained or the range of integration must be split to enable the shooting method to succeed, the recommended routine is D02SAF which extends the facilities provided by D02HBF. D02SAF permits the sophisticated user much more control over the calculation than does D02HBF; in particular the user is permitted precise control of solution output and intermediate monitoring information.

3.2.3 Finite-difference methods

D02GAF may be used for simple boundary value problems with assigned boundary values. The calling sequence of D02GAF is very similar to that for D02HAF discussed above.

The user may find that convergence is difficult to achieve using D02GAF since only specifying the unknown boundary values and the position of the finite-difference mesh is permitted. In such cases the user may use D02RAF which permits specification of an initial estimate for the solution at all mesh points and allows the calculation to be influenced in other ways too. D02RAF is designed to solve a general nonlinear two-point boundary value problem with nonlinear boundary conditions.

A routine, D02GBF, is also supplied specifically for the general linear two-point boundary value problem written in a standard 'textbook' form.

The user is advised to use interpolation routines from the E01 Chapter to obtain solution values at points not on the final mesh.

3.3 Chebyshev Collocation Method

D02TGF may be used to obtain the approximate solution of a system of differential equations in the form of a Chebyshev series. The routine treats linear differential equations directly, and makes no distinction between initial value and boundary value problems. This routine is appropriate for problems where it is known that the solution is smooth and well-behaved over the range, so that each component can be represented by a single polynomial. Singular problems can be solved using D02TGF as long as their polynomial-like solutions are required.

D02TGF permits the differential equations to be specified in higher order form; that is without conversion to a first-order system. This type of specification leads to a complicated calling sequence. For the inexperienced user two simpler routines are supplied. D02JAF solves a single regular linear differential equation of any order whereas D02JBF solves a system of regular linear first-order differential equations.

3.4 Eigenvalue Problems

Two routines, D02KAF and D02KDF, may be used to find the eigenvalues of second-order Sturm–Liouville problems. D02KAF is designed to solve simple problems with regular boundary conditions. D02KAF calls D02KDF which is designed to solve more difficult problems, for example with singular boundary conditions or on infinite ranges or with discontinuous coefficients.

If the eigenfunctions of the Sturm–Liouville problem are also required, D02KEF should be used. (D02KEF solves the same types of problem as D02KDF.)

3.5 Summary of Recommended Routines

Problem	Routine		
	R K method	Adams method	BDF method
Initial–value Problems			
Driver Routines			
Integration over a range with optional intermediate output and optional determination of position where a function of the solution becomes zero	D02BJF	D02CJF	D02EJF
Integration over a range			
–with intermediate output	D02BJF	D02CJF	D02EJF
–until function of solution becomes zero	D02BJF	D02CJF	D02EJF
Comprehensive Integration routines	D02PCF, D02PDF D02PVF, D02PWF D02PXF, D02PYF	D02QFF, D02QGF D02QWF, D02QXF D02QYF, D02QZF	D02M routines D02N routines D02XKF, D02XJF and D02ZAF
Package for Solving Stiff Equations	D02M–D02N Subchapter		
Package for Solving Second–order Systems of Special Form	D02L routines		
Boundary–value Problems Collocation Method, Mixed Order	D02TKF, D02TVF, D02TXF, D02TYF, D02TZF		
Boundary–value Problems			
Shooting Method			
simple parameter	D02HAF		
generalised parameters	D02HBF, D02AGF		
additional facilities	D02SAF		
Boundary–value Problems			
Finite–difference Method			
simple parameter	D02GAF		
linear problem	D02GBF		
full nonlinear problem	D02RAF		
Chebyshev Collocation, Linear Problems			
single equation	D02JAF		
first–order system	D02JBF		
general system	D02TGF		
Sturm–Liouville Eigenvalue Problems			
regular problems	D02KAF		
general problems	D02KDF		
eigenfunction calculation	D02KEF		

4 Routines Withdrawn or Scheduled for Withdrawal

Since Mark 13 the following routines have been withdrawn. Advice on replacing calls to these routines is given in the document 'Advice on Replacement Calls for Withdrawn/Superseded Routines'.

D02BAF D02BBF D02BDF D02CAF D02CBF D02CGF
D02CHF D02EAF D02EBF D02EGF D02EHF D02PAF
D02QAF D02QBF D02QDF D02QQF D02XAF D02XBF
D02XGF D02XHF D02YAF

5 References

[1] Ascher U M, Mattheij R M M and Russell R D (1988) *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations* Prentice Hall, Englewood Cliffs, NJ

[2] Ascher U M and Bader G (1987) A new basis implementation for a mixed order boundary value ODE solver *SIAM J. Sci. Stat. Comput.* **8** 483–500

[3] Ascher U M, Christiansen J and Russell R D (1979) A collocation solver for mixed order systems of boundary value problems *Math. Comput.* **33** 659–679

[4] Berzins M, Brankin R W and Gladwell I (1988) Design of the stiff integrators in the NAG Library *SIGNUM Newsl.* **23** 16–23

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

[6] Gladwell I (1979) Initial value routines in the NAG Library *ACM Trans. Math. Software* **5** 386–400

[7] Gladwell I (1987) The NAG Library boundary value codes *Numerical Analysis Report* **134** Manchester University

[8] Gladwell I and Sayers D K (ed.) (1980) *Computational Techniques for Ordinary Differential Equations* Academic Press

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

[10] Keller H B (1992) *Numerical Methods for Two-point Boundary-value Problems* Dover, New York

[11] Pryce J D (1986) Error estimation for phase-function shooting methods for Sturm–Liouville problems *IMA J. Numer. Anal.* **6** 103–123

D02AGF – NAG Fortran Library Routine Document

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

1. Purpose

D02AGF solves the two-point boundary-value problem for a system of ordinary differential equations, using initial value techniques and Newton iteration; it generalizes D02HAF to include the case where parameters other than boundary values are to be determined.

2. Specification

```
 SUBROUTINE D02AGF (H, ERROR, PARERR, PARAM, C, N, N1, M1, AUX,
1                   BCAUX, RAAUX, PRSOL, MAT, COPY, WSPACE, WSPAC1,
2                   WSPAC2, IFAIL)
 INTEGER          N, N1, M1, IFAIL
 real             H, ERROR(N), PARERR(N1), PARAM(N1), C(M1,N),
1                 MAT(N1,N1), COPY(N1,N1), WSPACE(N,9), WSPAC1(N),
2                 WSPAC2(N)
 EXTERNAL         AUX, BCAUX, RAAUX, PRSOL
```

3. Description

The routine solves the two-point boundary-value problem by determining the unknown parameters $p_1,p_2,...,p_{n_1}$ of the problem. These parameters may be, but need not be, boundary values (as they are in D02HAF); they may include eigenvalue parameters in the coefficients of the differential equations, length of the range of integration, etc. The notation and methods used are similar to those of D02HAF and the user is advised to study this first. (There the parameters $p_1,p_2,...,p_{n_1}$ correspond to the unknown boundary conditions.) It is assumed that we have a system of n first-order ordinary differential equations of the form:

$$\frac{dy_i}{dx} = f_i(x,y_1,y_2,...,y_n), \qquad i = 1,2,...,n,$$

and that derivatives f_i are evaluated by a subroutine AUX supplied by the user. The system, including the boundary conditions given by BCAUX, and the range of integration and matching point, r, given by RAAUX, involves the n_1 unknown parameters $p_1,p_2,...,p_{n_1}$ which are to be determined, and for which initial estimates must be supplied. The number of unknown parameters n_1 must not exceed the number of equations n. If $n_1 < n$, we assume that $(n-n_1)$ equations of the system are not involved in the matching process. These are usually referred to as 'driving equations'; they are independent of the parameters and of the solutions of the other n_1 equations. In numbering the equations for the subroutine AUX, the driving equations must be put last.

The estimated values of the parameters are corrected by a form of Newton iteration. The Newton correction on each iteration is calculated using a matrix whose (i,j)th element depends on the derivative of the ith component of the solution, y_i, with respect to the jth parameter, p_j. This matrix is calculated by a simple numerical differentiation technique which requires n_1 evaluations of the differential system.

4. References

None.

5. Parameters

Users are strongly recommended to read Sections 3 and 8 in conjunction with this section.

1: H – ***real***. *Input/Output*

On entry: H must be set to an estimate of the step size needed for integration, h.

On exit: the last step length used.

2: ERROR(N) – ***real*** array. *Input*

On entry: ERROR(i) must be set to a small quantity to control the ith solution component. The element ERROR(i) is used:

(i) in the bound on the local error in the ith component of the solution y_i during integration,

(ii) in the convergence test on the ith component of the solution y_i at the matching point in the Newton iteration.

The elements ERROR(i) should not be chosen too small. They should usually be several orders of magnitude larger than ***machine precision***.

3: PARERR(N1) – ***real*** array. *Input*

On entry: PARERR(i) must be set to a small quantity to control the ith parameter component. The element PARERR(i) is used:

(i) in the convergence test on the ith parameter in the Newton iteration,

(ii) in perturbing the ith parameter when approximating the derivatives of the components of the solution with respect to the ith parameter, for use in the Newton iteration.

The elements PARERR(i) should not be chosen too small. They should usually be several orders of magnitude larger than ***machine precision***.

4: PARAM(N1) – ***real*** array. *Input/Output*

On entry: PARAM(i) must be set to an estimate for the ith parameter, p_i, for i = 1,2,...,N1.

On exit: the corrected value for the ith parameter, unless an error has occurred, when it contains the last calculated value of the parameter (possibly perturbed by PARERR(i)×(1+|PARAM(i)|) if the error occurred when calculating the approximate derivatives).

5: C(M1,N) – ***real*** array. *Output*

On exit: the solution when M1 > 1 (see below).

If M1 = 1 then the elements of C are not used.

6: N – INTEGER. *Input*

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

7: N1 – INTEGER. *Input*

On entry: the number of parameters, n_1.

If N1 < N, the last N – N1 differential equations (in the subroutine AUX below) are driving equations (see Section 3).

Constraint: N1 ≤ N.

8: M1 – INTEGER. *Input*

On entry: determines whether or not the final solution is computed as well as the parameter values. For

M1 = 1

the final solution is not calculated;

M1 > 1

the final values of the solution at interval (length of range)/(M1−1) are calculated and stored sequentially in the array C starting with the values of y_i evaluated at the first end-point (see subroutine RAAUX below) stored in C(1,i).

9: AUX – SUBROUTINE, supplied by the user. *External Procedure*

AUX must evaluate the functions f_i (i.e. the derivatives y_i') for given values of its arguments, $x,y_1,...,y_n, p_1,...,p_{n_1}$

Its specification is:

```
SUBROUTINE AUX(F, Y, X, PARAM)
real         F(n), Y(n), X, PARAM(n1)
```

where n and n1 are the numerical values of N and N1 in the call of D02AGF.

1: F(n) – ***real*** array. *Output*

On exit: the value of f_i, for $i = 1,2,...,n$.

2: Y(n) – ***real*** array. *Input*

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

3: X – ***real.*** *Input*

On entry: the value of the argument x.

4: PARAM(n1) – ***real*** array. *Input*

On entry: the value of the argument p_i, for $i = 1,2,...,n_1$.

AUX must be declared as EXTERNAL in the (sub)program from which D02AGF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

10: BCAUX – SUBROUTINE, supplied by the user. *External Procedure*

BCAUX must evaluate the values of y_i at the end-points of the range given the values of $p_1,...,p_{n_1}$.

Its specification is:

```
SUBROUTINE BCAUX(G0, G1, PARAM)
real         G0(n), G1(n), PARAM(n1)
```

where n and n1 are the numerical values of N and N1 in the call of D02AGF.

1: G0(n) – ***real*** array. *Output*

On exit: the values y_i, for $i = 1,2,...,n$, at the boundary point x_0 (see RAAUX below).

2: G1(n) – ***real*** array. *Output*

On exit: the values y_i, for $i = 1,2,...,n$, at the boundary point x_1 (see RAAUX below).

3: PARAM(n1) – ***real*** array. *Input*

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

BCAUX must be declared as EXTERNAL in the (sub)program from which D02AGF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

11: RAAUX – SUBROUTINE, supplied by the user. *External Procedure*

RAAUX must evaluate the end-points, x_0 and x_1, of the range and the matching point, r, given the values $p_1,p_2,...,p_{n_1}$.

Its specification is:

```
SUBROUTINE RAAUX(X0, X1, R, PARAM)
real       X0, X1, R, PARAM(n1)
```

where n1 is the numerical value of N1 in the call of D02AGF.

1: X0 – ***real.*** *Output*

On exit: must contain the left-hand end of the range, x_0.

2: X1 – ***real.*** *Output*

On exit: must contain the right-hand end of the range x_1.

3: R – ***real.*** *Output*

On exit: must contain the matching point, r.

4: PARAM(n1) – ***real*** array. *Input*

On entry: the value of the argument p_i, for $i = 1,2,...,n_1$.

RAAUX must be declared as EXTERNAL in the (sub)program from which D02AGF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

PRSOL is called at each iteration of the Newton method and can be used to print the current values of the parameters p_i, for $i = 1,2,...,n_1$, their errors, e_i, and the sum of squares of the errors at the matching point, r.

Its specification is:

```
SUBROUTINE PRSOL(PARAM, RES, N1, ERR)
INTEGER    N1
real       PARAM(N1), RES, ERR(N1)
```

1: PARAM(N1) – ***real*** array. *Input*

On entry: the current value of the parameters p_i, for $i = 1,2,...,n_1$.

2: RES – ***real.*** *Input*

On entry: the sum of squares of the errors in the parameters, $\sum_{i=1}^{n_1} e_i^2$.

3: N1 – INTEGER. *Input*

On entry: the number of parameters, n_1.

4: ERR(N1) – ***real*** array. *Input*

On entry: the errors in the parameters, e_i, for $i = 1,2,...,n_1$.

PRSOL must be declared as EXTERNAL in the (sub)program from which D02AGF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

13: MAT(N1,N1) – ***real*** array. *Workspace*

14: COPY(N1,N1) – ***real*** array. *Workspace*

15: WSPACE(N,9) – ***real*** array. *Workspace*

16: WSPAC1(N) – ***real*** array. *Workspace*

17: WSPAC2(N) – ***real*** array. *Workspace*

18: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

This indicates that N1 > N on entry, that is the number of parameters is greater than the number of differential equations.

IFAIL = 2

As for IFAIL = 4 (below) except that the integration failed while calculating the matrix for use in the Newton iteration.

IFAIL = 3

The current matching point r does not lie between the current end-points x_0 and x_1. If the values x_0, x_1 and r depend on the parameters p_i, this may occur at any time in the Newton iteration if care is not taken to avoid it when coding subroutine RAAUX.

IFAIL = 4

The step length for integration H has halved more than 13 times (or too many steps were needed to reach the end of the range of integration) in attempting to control the local truncation error whilst integrating to obtain the solution corresponding to the current values p_i. If, on failure, H has the sign of $r - x_0$ then failure has occurred whilst integrating from x_0 to r, otherwise it has occurred whilst integrating from x_1 to r.

IFAIL = 5

The matrix of the equations to be solved for corrections to the variable parameters in the Newton method is singular (as determined by F03AFF).

IFAIL = 6

A satisfactory correction to the parameters was not obtained on the last Newton iteration employed. A Newton iteration is deemed to be unsatisfactory if the sum of the squares of the residuals (which can be printed using PRSOL) has not been reduced after three iterations using a new Newton correction.

IFAIL = 7

Convergence has not been obtained after 12 satisfactory iterations of the Newton method.

A further discussion of these errors and the steps which might be taken to correct them is given in Section 8.

7. Accuracy

If the process converges, the accuracy to which the unknown parameters are determined is usually close to that specified by the user; and the solution, if requested, is usually determined to the accuracy specified.

8. Further Comments

The time taken by the routine depends on the complexity of the system, and on the number of iterations required. In practice, integration of the differential equations is by far the most costly process involved.

There may be particular difficulty in integrating the differential equations in one direction (indicated by IFAIL = 2 or 4). The value of r should be adjusted to avoid such difficulties.

If the matching point r is at one of the end-points x_0 or x_1 and some of the parameters are used **only** to determine the boundary values at this point, then good initial estimates for these parameters are not required, since they are completely determined by the routine (for example, see p_2 in example (i) of Section 9).

Wherever they occur in the procedure, the error parameters contained in the arrays ERROR and PARERR are used in 'mixed' form; that is ERROR(i) always occurs in expressions of the form ERROR(i)$\times(1+|y_i|)$, and PARERR(i) always occurs in expressions of the form PARERR(i)$\times(1+|p_i|)$. Though not ideal for every application, it is expected that this mixture of absolute and relative error testing will be adequate for most purposes.

Note that **convergence is not guaranteed**. The user is strongly advised to provide an output subroutine PRSOL, as shown in the example (i) of Section 9, in order to monitor the progress of the iteration. Failure of the Newton iteration to converge (IFAIL = 6 or IFAIL = 7) usually results from poor starting approximations to the parameters, though occasionally such failures occur because the elements of one or both of the arrays PARERR or ERROR are too small. (It should be possible to distinguish these cases by studying the output from PRSOL.) Poor starting approximations can also result in the failure described under IFAIL = 4 and IFAIL = 5 in Section 6 (especially if these errors occur after some Newton iterations have been completed, that is, after two or more calls of PRSOL). More frequently, a singular matrix in the Newton method (monitored as IFAIL = 5) occurs because the mathematical problem has been posed incorrectly. The case IFAIL = 4 usually occurs because h or r has been poorly estimated, so these values should be checked first. If IFAIL = 2 is monitored, the solution $y_1, y_2, \ldots, y_n$ is sensitive to perturbations in the parameters p_i. Reduce the size of one or more values PARERR(i) to reduce the perturbations. Since only one value p_i is perturbed at any time when forming the matrix, the perturbation which is too large can be located by studying the final output from PRSOL and the values of the parameters returned by D02AGF. If this change leads to other types of failure improve the initial values of p_i by other means.

The computing time for integrating the differential equations can sometimes depend critically on the quality of the initial estimates for the parameters p_i. If it seems that too much computing time is required and, in particular, if the values ERR(i) (available on each call of PRSOL) are much larger than the expected values of the solution at the matching point r, then the coding of the subroutines AUX, BCAUX and RAAUX should be checked for errors. If no errors can be found, an independent attempt should be made to improve the initial estimates for PARAM(i).

The subroutine can be used to solve a very wide range of problems, for example:

(a) eigenvalue problems, including problems where the eigenvalue occurs in the boundary conditions;

(b) problems where the differential equations depend on some parameters which are to be determined so as to satisfy certain boundary conditions (see example (ii) in Section 9);

(c) problems where one of the end-points of the range of integration is to be determined as the point where a variable y_i takes a particular value (see (ii) in Section 9);

(d) singular problems and problems on infinite ranges of integration where the values of the solution at x_0 or x_1 or both are determined by a power series or an asymptotic expansion (or a more complicated expression) and where some of the coefficients in the expression are to be determined (see example (i) in Section 9); and

(e) differential equations with certain terms defined by other independent (driving) differential equations.

9. Example

For this routine two examples are presented, in Sections 9.1 and 9.2. In the example programs distributed to sites, there is a single example program for D02AGF, with a main program:

```
*     D02AGF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. External Subroutines ..
      EXTERNAL         EX1, EX2
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02AGF Example Program Results'
      CALL EX1
      CALL EX2
      STOP
      END
```

The code to solve the two example problems is given in the subroutines EX1 and EX2, in Sections 9.1.1 and 9.2.1 respectively.

9.1. Example 1

To find the solution of the differential equation

$$y'' = \frac{y^3-y'}{2x}$$

on the range $0 \leq x \leq 16$, with boundary conditions $y(0) = 0.1$ and $y(16) = 1/6$.

We cannot use the differential equation at $x = 0$ because it is singular, so we take the truncated series expansion

$$y(x) = \frac{1}{10} + p_1\frac{\sqrt{x}}{10} + \frac{x}{100}$$

near the origin (which is correct to the number of terms given in this case). Where p_1 is one of the parameters to be determined. We choose the range as [0.1,16] and setting $p_2 = y'(16)$, we can determine all the boundary conditions. We take the matching point to be 16, the end of the range, and so a good initial guess for p_2 is not necessary. We write $y =$ Y(1), $y' =$ Y(2), and estimate $p_1 =$ PARAM(1) $= 0.2$, $p_2 =$ PARAM(2) $= 0.0$.

9.1.1. Program text

```
      SUBROUTINE EX1
*     .. Parameters ..
      INTEGER          N, M1
      PARAMETER        (N=2,M1=6)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      INTEGER          IPRINT
*     .. Local Scalars ..
      real             DUM, H, R, X, X1
      INTEGER          I, IFAIL, J, N1
*     .. Local Arrays ..
      real             C(M1,N), COPY(N,N), ERROR(N), G(N), G1(N),
     +                 MAT(N,N), PARAM(N), PARERR(N), WSPACE(N,9)
*     .. External Subroutines ..
      EXTERNAL         AUX1, BCAUX1, D02AGF, PRSOL, RNAUX1
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Common blocks ..
      COMMON           /BLOCK1/IPRINT
```

```
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 1'
      WRITE (NOUT,*)
*     * Set IPRINT to 1 to obtain output from PRSOL at each iteration *
      IPRINT = 0
      PARAM(1) = 0.2e0
      PARAM(2) = 0.0e0
      N1 = 2
      H = 0.1e0
      PARERR(1) = 1.0e-5
      PARERR(2) = 1.0e-3
      ERROR(1) = 1.0e-4
      ERROR(2) = 1.0e-4
      IFAIL = 1
*
      CALL D02AGF(H,ERROR,PARERR,PARAM,C,N,N1,M1,AUX1,BCAUX1,RNAUX1,
     +            PRSOL,MAT,COPY,WSPACE,G,G1,IFAIL)
*
      IF (IFAIL.EQ.0) THEN
         WRITE (NOUT,*) 'Final parameters'
         WRITE (NOUT,99998) (PARAM(I),I=1,N1)
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Final solution'
         WRITE (NOUT,*) 'X-value     Components of solution'
         CALL RNAUX1(X,X1,R,PARAM)
         H = (X1-X)/5.0e0
         DO 20 I = 1, 6
            DUM = X + real(I-1)*H
            WRITE (NOUT,99997) DUM, (C(I,J),J=1,N)
   20    CONTINUE
      ELSE
         WRITE (NOUT,99999) 'IFAIL = ', IFAIL
      END IF
      RETURN
*
99999 FORMAT (1X,A,I3)
99998 FORMAT (1X,3e16.6)
99997 FORMAT (1X,F7.2,3e13.4)
      END
*
      SUBROUTINE AUX1(F,Y,X,PARAM)
*     .. Scalar Arguments ..
      real              X
*     .. Array Arguments ..
      real              F(2), PARAM(2), Y(2)
*     .. Executable Statements ..
      F(1) = Y(2)
      F(2) = (Y(1)**3-Y(2))/(2.0e0*X)
      RETURN
      END
*
      SUBROUTINE RNAUX1(X,X1,R,PARAM)
*     .. Scalar Arguments ..
      real              R, X, X1
*     .. Array Arguments ..
      real              PARAM(2)
*     .. Executable Statements ..
      X = 0.1e0
      X1 = 16.0e0
      R = 16.0e0
      RETURN
      END
*
```

```
      SUBROUTINE BCAUX1(G,G1,PARAM)
*     .. Array Arguments ..
      real                    G(2), G1(2), PARAM(2)
*     .. Local Scalars ..
      real                    Z
*     .. Intrinsic Functions ..
      INTRINSIC               SQRT
*     .. Executable Statements ..
      Z = 0.1e0
      G(1) = 0.1e0 + PARAM(1)*SQRT(Z)*0.1e0 + 0.01e0*Z
      G(2) = PARAM(1)*0.05e0/SQRT(Z) + 0.01e0
      G1(1) = 1.0e0/6.0e0
      G1(2) = PARAM(2)
      RETURN
      END
*
      SUBROUTINE PRSOL(PARAM,RESID,N1,ERR)
*     .. Parameters ..
      INTEGER                NOUT
      PARAMETER              (NOUT=6)
*     .. Scalar Arguments ..
      real                   RESID
      INTEGER                N1
*     .. Array Arguments ..
      real                   ERR(N1), PARAM(N1)
*     .. Scalars in Common ..
      INTEGER                IPRINT
*     .. Local Scalars ..
      INTEGER                I
*     .. Common blocks ..
      COMMON                 /BLOCK1/IPRINT
*     .. Executable Statements ..
      IF (IPRINT.NE.0) THEN
         WRITE (NOUT,99999) 'Current parameters   ', (PARAM(I),I=1,N1)
         WRITE (NOUT,99998) 'Residuals   ', (ERR(I),I=1,N1)
         WRITE (NOUT,99998) 'Sum of residuals squared  ', RESID
         WRITE (NOUT,*)
      END IF
      RETURN
*
99999 FORMAT (1X,A,6(e14.6,2X))
99998 FORMAT (1X,A,6(e12.4,1X))
      END
```

9.1.2. Program data

None.

9.1.3. Program results

```
D02AGF Example Program Results


Case 1

Final parameters
    0.464269E-01    0.349429E-02

Final solution
X-value     Components of solution
   0.10   0.1025E+00   0.1734E-01
   3.28   0.1217E+00   0.4180E-02
   6.46   0.1338E+00   0.3576E-02
   9.64   0.1449E+00   0.3418E-02
  12.82   0.1557E+00   0.3414E-02
  16.00   0.1667E+00   0.3494E-02
```

With IPRINT set to 1 in the example program, monitoring information similar to that below is obtained:

```
Current parameters     0.200000E+00    0.000000E+00
Residuals    -0.8426E-01  -0.8408E-02
Sum of residuals squared    0.7171E-02

Current parameters     0.577582E-01    0.278452E-02
Residuals    -0.5802E-02  -0.1005E-02
Sum of residuals squared    0.3467E-04

Current parameters     0.479643E-01    0.340256E-02
Residuals    -0.7841E-03  -0.1313E-03
Sum of residuals squared    0.6321E-06

Current parameters     0.466406E-01    0.348158E-02
Residuals    -0.1092E-03  -0.1825E-04
Sum of residuals squared    0.1225E-07

Current parameters     0.464562E-01    0.349254E-02
Residuals    -0.1526E-04  -0.2551E-05
Sum of residuals squared    0.2395E-09

Current parameters     0.464305E-01    0.349407E-02
Residuals    -0.2135E-05  -0.3568E-06
Sum of residuals squared    0.4685E-11
```

9.2 Example 2

To find the gravitational constant p_1 and the range p_2 over which a projectile must be fired to hit the target with a given velocity. The differential equations are

$$y' = \tan\phi$$

$$v' = \frac{-(p_1 \sin\phi + 0.00002v^2)}{v\cos\phi}$$

$$\phi' = \frac{-p_1}{v^2}$$

on the range $0 < x < p_2$ with boundary conditions

$$y = 0, \quad v = 500, \quad \phi = 0.5 \quad \text{at} \quad x = 0$$

$$y = 0, \quad v = 450, \quad \phi = p_3 \quad \text{at} \quad x = p_2.$$

We write y = Y(1), v = Y(2), ϕ = Y(3), and we take the matching point $r = p_2$. We estimate p_1 = PARAM(1) = 32, p_2 = PARAM(2) = 6000 and p_2 = PARAM(3) = 0.54 (though this estimate is not important).

9.2.1. Program text

```
      SUBROUTINE EX2
*     .. Parameters ..
      INTEGER          N, M1
      PARAMETER        (N=3,M1=6)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      INTEGER          IPRINT
*     .. Local Scalars ..
      real             DUM, H, R, X, X1
      INTEGER          I, IFAIL, J
*     .. Local Arrays ..
      real             C(M1,N), COPY(N,N), ERROR(N), G(N), G1(N),
     +                 MAT(N,N), PARAM(N), PARERR(N), WSPACE(N,9)
*     .. External Subroutines ..
      EXTERNAL         AUX2, BCAUX2, D02AGF, PRSOL, RNAUX2
*     .. Intrinsic Functions ..
      INTRINSIC        real
```

```
*     .. Common blocks ..
      COMMON           /BLOCK1/IPRINT
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 2'
      WRITE (NOUT,*)
*
*     Set IPRINT to 1 to obtain output from PRSOL at each iteration *
      IPRINT = 0
      H = 10.0e0
      PARAM(1) = 32.0e0
      PARAM(2) = 6000.0e0
      PARAM(3) = 0.54e0
      PARERR(1) = 1.0e-5
      PARERR(2) = 1.0e-4
      PARERR(3) = 1.0e-4
      ERROR(1) = 1.0e-2
      ERROR(2) = 1.0e-2
      ERROR(3) = 1.0e-2
      IFAIL = 1
*
      CALL D02AGF(H,ERROR,PARERR,PARAM,C,N,N,M1,AUX2,BCAUX2,RNAUX2,
     +            PRSOL,MAT,COPY,WSPACE,G,G1,IFAIL)
*
      IF (IFAIL.EQ.0) THEN
         WRITE (NOUT,*) 'Final parameters'
         WRITE (NOUT,99998) (PARAM(I),I=1,N)
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Final solution'
         WRITE (NOUT,*) 'X-value     Components of solution'
         CALL RNAUX2(X,X1,R,PARAM)
         H = (X1-X)/5.0e0
         DO 20 I = 1, 6
            DUM = X + real(I-1)*H
            WRITE (NOUT,99997) DUM, (C(I,J),J=1,N)
   20    CONTINUE
      ELSE
         WRITE (NOUT,99999) 'IFAIL = ', IFAIL
      END IF
      RETURN
*
99999 FORMAT (1X,A,I3)
99998 FORMAT (1X,3e16.6)
99997 FORMAT (1X,F7.0,3e13.4)
      END
*
      SUBROUTINE AUX2(F,Y,X,PARAM)
*     .. Scalar Arguments ..
      real             X
*     .. Array Arguments ..
      real             F(3), PARAM(3), Y(3)
*     .. Local Scalars ..
      real             C, S
*     .. Intrinsic Functions ..
      INTRINSIC        COS, SIN
*     .. Executable Statements ..
      C = COS(Y(3))
      S = SIN(Y(3))
      F(1) = S/C
      F(2) = -(PARAM(1)*S+0.00002e0*Y(2)*Y(2))/(Y(2)*C)
      F(3) = -PARAM(1)/(Y(2)*Y(2))
      RETURN
      END
*
```

```
      SUBROUTINE RNAUX2(X,X1,R,PARAM)
*     .. Scalar Arguments ..
      real              R, X, X1
*     .. Array Arguments ..
      real              PARAM(3)
*     .. Executable Statements ..
      X = 0.0e0
      X1 = PARAM(2)
      R = PARAM(2)
      RETURN
      END
*
      SUBROUTINE BCAUX2(G,G1,PARAM)
*     .. Array Arguments ..
      real              G(3), G1(3), PARAM(3)
*     .. Executable Statements ..
      G(1) = 0.0e0
      G(2) = 500.0e0
      G(3) = 0.5e0
      G1(1) = 0.0e0
      G1(2) = 450.0e0
      G1(3) = PARAM(3)
      RETURN
      END
*
      SUBROUTINE PRSOL(PARAM,RESID,N1,ERR)
*     .. Parameters ..
      INTEGER           NOUT
      PARAMETER         (NOUT=6)
*     .. Scalar Arguments ..
      real              RESID
      INTEGER           N1
*     .. Array Arguments ..
      real              ERR(N1), PARAM(N1)
*     .. Scalars in Common ..
      INTEGER           IPRINT
*     .. Local Scalars ..
      INTEGER           I
*     .. Common blocks ..
      COMMON            /BLOCK1/IPRINT
*     .. Executable Statements ..
      IF (IPRINT.NE.0) THEN
         WRITE (NOUT,99999) 'Current parameters   ', (PARAM(I),I=1,N1)
         WRITE (NOUT,99998) 'Residuals    ', (ERR(I),I=1,N1)
         WRITE (NOUT,99998) 'Sum of residuals squared  ', RESID
         WRITE (NOUT,*)
      END IF
      RETURN
*
99999 FORMAT (1X,A,6(e14.6,2X))
99998 FORMAT (1X,A,6(e12.4,1X))
      END
```

9.2.2. Program data

None.

9.2.3. Program results

```
Case 2

Final parameters
    0.323729E+02    0.596317E+04   -0.535231E+00

Final solution
X-value      Components of solution
     0.   0.0000E+00  0.5000E+03   0.5000E+00
  1193.   0.5298E+03  0.4516E+03   0.3281E+00
  2385.   0.8076E+03  0.4203E+03   0.1231E+00
  3578.   0.8208E+03  0.4094E+03  -0.1032E+00
  4771.   0.5563E+03  0.4200E+03  -0.3296E+00
  5963.   0.0000E+00  0.4500E+03  -0.5352E+00
```

D02BGF – NAG Fortran Library Routine Document

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

1. Purpose

D02BGF integrates a system of first-order ordinary differential equations over an interval with suitable initial conditions, using a Runge-Kutta-Merson method, until a specified component attains a given value.

2. Specification

```
SUBROUTINE D02BGF (X, XEND, N, Y, TOL, HMAX, M, VAL, FCN, W, IFAIL)

INTEGER        N, M, IFAIL
real           X, XEND, Y(N), TOL, HMAX, VAL, W(N,10)
EXTERNAL       FCN
```

3. Description

The routine advances the solution of a system of ordinary differential equations

$$y_i' = f_i(x,y_1,y_2,...,y_n), \qquad i = 1,2,...,n,$$

from x = X towards x = XEND using a Merson form of the Runge-Kutta method. The system is defined by a subroutine FCN supplied by the user, which evaluates f_i in terms of x and $y_1,y_2,...,y_n$ (see Section 5), and the values of $y_1,y_2,...,y_n$ must be given at x = X.

As the integration proceeds, a check is made on the specified component y_m of the solution to determine an interval where it attains a given value α. The position where this value is attained is then determined accurately by interpolation on the solution and its derivative. It is assumed that the solution of $y_m = \alpha$ can be determined by searching for a change in sign in the function $y_m-\alpha$.

The accuracy of the integration and, indirectly, of the determination of the position where $y_m = \alpha$ is controlled by the parameter TOL.

For a description of Runge-Kutta methods and their practical implementation see Hall and Watt [1].

4. References

[1] HALL, G. and WATT, J.M. (eds.)
Modern Numerical Methods for Ordinary Differential Equations.
Clarendon Press, Oxford, p. 59, 1976.

5. Parameters

1: X – ***real***. *Input/Output*

On entry: X must be set to the initial value of the independent variable x.

On exit: the point where the component y_m attains the value α unless an error has occurred, when it contains the value of x at the error. In particular, if $y_m \neq \alpha$ anywhere on the range x = X to x = XEND, it will contain XEND on exit.

2: XEND – ***real***. *Input*

On entry: the final value of the independent variable x.

If XEND < X on entry integration will proceed in the negative direction.

3: N – INTEGER. *Input*

On entry: the number of differential equations, n.

Constraint: N > 0.

4: Y(N) – ***real*** array. *Input/Output*

On entry: the initial values of the solution $y_1,y_2,...,y_n$.

On exit: the computed values of the solution at a point near the solution X, unless an error has occurred when they contain the computed values at the final value of X.

5: TOL – ***real.*** *Input/Output*

On entry: TOL must be set to a positive tolerance for controlling the error in the integration and in the determination of the position where $y_m = \alpha$.

D02BGF has been designed so that, for most problems, a reduction in TOL leads to an approximately proportional reduction in the error in the solution obtained in the integration. The relation between changes in TOL and the error in the determination of the position where $y_m = \alpha$ is less clear, but for TOL small enough the error should be approximately proportional to TOL. However, the actual relation between TOL and the accuracy cannot be guaranteed. The user is strongly recommended to call D02BGF with more than one value for TOL and to compare the results obtained to estimate their accuracy. In the absence of any prior knowledge the user might compare results obtained by calling D02BGF with TOL = 10.0^{-p} and TOL = 10.0^{-p-1} if p correct decimal digits in the solution are required.

Constraint: TOL > 0.0.

On exit: normally unchanged. However if the range from X to the position where $y_m = \alpha$ (or to the final value of X if an error occurs) is so short that a small change in TOL is unlikely to make any change in the computed solution then, on return, TOL has its sign changed. To check results returned with TOL < 0.0, D02BGF should be called again with a positive value of TOL whose magnitude is considerably smaller than that of the previous call.

6: HMAX – ***real.*** *Input*

On entry: controls how the sign of $y_m-\alpha$ is checked.

If HMAX = 0.0, $y_m-\alpha$ is checked at every internal integration step.

If HMAX ≠ 0.0, the computed solution is checked for a change in sign of $y_m-\alpha$ at steps of not greater than ABS(HMAX). This facility should be used if there is any chance of 'missing' the change in sign by checking too infrequently. For example, if two changes of sign of $y_m-\alpha$ are expected within a distance h, say, of each other then a suitable value for HMAX might be HMAX = $h/2$. If only one change of sign in $y_m-\alpha$ is expected on the range X to XEND then HMAX = 0.0 is most appropriate.

7: M – INTEGER. *Input*

On entry: the index m of the component of the solution whose value is to be checked.

Constraint: 1 ≤ M ≤ N.

8: VAL – ***real.*** *Input*

On entry: the value of α in the equation $y_m = \alpha$ to be solved for X.

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

FCN must evaluate the functions f_i (i.e. the derivatives y_i') for given values of its arguments $x,y_1,...,y_n$.

Its specification is:

```
SUBROUTINE FCN(X, Y, F)
real         X, Y(n), F(n)
```

where n is the actual value of N in the call of D02BGF.

1: X – ***real***. *Input*

On entry: the value of the argument x.

2: Y(n) – ***real*** array. *Input*

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

3: F(n) – ***real*** array. *Output*

On exit: the value of f_i, for $i = 1,2,...,n$.

FCN must be declared as EXTERNAL in the (sub)program from which D02BGF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

10: W(N,10) – ***real*** array. *Workspace*

11: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, TOL ≤ 0.0,
or N ≤ 0,
or M ≤ 0,
or M > N.

IFAIL = 2

With the given value of TOL, no further progress can be made across the integration range from the current point x = X, or dependence of the error on TOL would be lost if further progress across the integration range were attempted (see Section 8 for a discussion of this error exit). The components Y(1),Y(2),...,Y(n) contain the computed values of the solution at the current point x = X. No point at which $y_m - \alpha$ changes sign has been located up to the point x = X.

IFAIL = 3

TOL is too small for the routine to take an initial step (see Section 8). X and Y(1),Y(2),...,Y(n) retain their initial values.

IFAIL = 4

At no point in the range X to XEND did the function $y_m - \alpha$ change sign. It is assumed that $y_m - \alpha$ has no solution.

IFAIL = 5

A serious error has occurred in an internal call to C05AZF. Check all subroutine calls and array dimensions. Seek expert help.

IFAIL = 6

A serious error has occurred in an internal call to an integration routine. Check all subroutine calls and array dimensions. Seek expert help.

IFAIL = 7

A serious error has occurred in an internal call to an interpolation routine. Check all subroutine calls and array dimensions. Seek expert help.

7. Accuracy

The accuracy depends on TOL, on the mathematical properties of the differential system, on the position where $y_m = \alpha$ and on the method. It can be controlled by varying TOL but the approximate proportionality of the error to TOL holds only for a restricted range of values of TOL. For TOL too large, the underlying theory may break down and the result of varying TOL may be unpredictable. For TOL too small, rounding error may affect the solution significantly and an error exit with IFAIL = 2 or IFAIL = 3 is possible.

8. Further Comments

The time taken by the routine depends on the complexity and mathematical properties of the system of differential equations defined by FCN, on the range, the position of solution and the tolerance. There is also an overhead of the form $a + b \times n$ where a and b are machine-dependent computing times.

For some problems it is possible that D02BGF will exit with IFAIL = 4 due to inaccuracy of the computed value y_m. For example, consider a case where the component y_m has a maximum in the integration range and α is close to the maximum value. If TOL is too large, it is possible that the maximum might be estimated as less than α, or even that the integration step length chosen might be so long that the maximum of y_m and the (two) positions where $y_m = \alpha$ are all in the same step and so the position where $y_m = \alpha$ remains undetected. Both these difficulties can be overcome by reducing TOL sufficiently and, if necessary, by choosing HMAX sufficiently small. For similar reasons, care should be taken when choosing XEND. If possible, the user should choose XEND well beyond the point where y_m is expected to equal α, for example |XEND−X| should be made about 50% longer than the expected range. As a simple check, if, with XEND fixed, a change in TOL does not lead to a significant change in y_m at XEND, then inaccuracy is not a likely source of error.

If the routine fails with IFAIL = 3, then it could be called again with a larger value of TOL if this has not already been tried. If the accuracy requested is really needed and cannot be obtained with this routine, the system may be very stiff (see below) or so badly scaled that it cannot be solved to the required accuracy.

If the routine fails with IFAIL = 2, it is likely that it has been called with a value of TOL which is so small that a solution cannot be obtained on the range X to XEND. This can happen for well-behaved systems and very small values of TOL. The user should, however, consider whether there is a more fundamental difficulty. For example:

(a) in the region of a singularity (infinite value) of the solution, the routine will usually stop with IFAIL = 2, unless overflow occurs first. If overflow occurs using D02BGF, routine D02PDF can be used instead to detect the increasing solution before overflow occurs. In any case, numerical integration cannot be continued through a singularity, and analytical treatment should be considered;

(b) for 'stiff' equations, where the solution contains rapidly decaying components the routine will use very small steps in x (internally to D02BGF) to preserve stability. This will usually exhibit itself by making the computing time excessively long, or occasionally by an exit with IFAIL = 2. Merson's method is not efficient in such cases, and the user should try the method D02EJF which uses a Backward Differentiation Formula. To determine whether a problem is stiff, D02PCF may be used.

For well-behaved systems with no difficulties such as stiffness or singularities, the Merson method should work well for low accuracy calculations (three or four figures). For high accuracy calculations or where FCN is costly to evaluate, Merson's method may not be appropriate and a computationally less expensive method may be D02CJF which uses an Adams method.

For problems for which D02BGF is not sufficiently general, the user should consider the routines D02PDF and D02BHF. Routine D02BHF can be used to solve an equation involving the components $y_1, y_2, ..., y_n$ and their derivatives (for example, to find where a component passes through zero or to find the maximum value of a component). It also permits a more general form of error control and may be preferred to D02BGF if the component whose value is to be determined is very small in modulus on the integration range. D02BHF can always be used in place of D02BGF, but will usually be computationally more expensive for solving the same problem. D02PDF is a more general routine with many facilities including a more general error control criteron. D02PDF can be combined with the root-finder C05AZF and the interpolation routine D02PXF to solve equations involving $y_1, y_2, ..., y_n$ and their derivatives.

This routine is only intended to be used to locate the **first** zero of the function $y_m - \alpha$. If later zeros are required users are strongly advised to construct their own more general root finding routines as discussed above.

9. Example

To find the value X > 0.0 where $y = 0.0$, where y, v, ϕ, are defined by

$$y' = \tan\phi$$

$$v' = \frac{-0.032\tan\phi}{v} - \frac{0.02v}{\cos\phi}$$

$$\phi' = \frac{-0.032}{v^2}$$

and where at X = 0.0 we are given $y = 0.5$, $v = 0.5$ and $\phi = \pi/5$. We write y = Y(1), v = Y(2) and ϕ = Y(3) and we set TOL = 1.0E–4 and TOL = 1.0E–5 in turn so that we can compare the solutions obtained. We expect the solution X $\simeq$ 7.3 and we set XEND = 10.0 so that the point where $y = 0.0$ is not too near the end of the range of integration. The value of π is obtained by using X01AAF.

9.1. Program Text

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

```
*     D02BGF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          N, M
      PARAMETER        (N=3,M=1)
*     .. Local Scalars ..
      real             HMAX, PI, TOL, VAL, X, XEND
      INTEGER          I, IFAIL
*     .. Local Arrays ..
      real             W(N,10), Y(N)
*     .. External Functions ..
      real             X01AAF
      EXTERNAL         X01AAF
*     .. External Subroutines ..
      EXTERNAL         D02BGF, FCN
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02BGF Example Program Results'
      XEND = 10.0e0
      HMAX = 0.0e0
      VAL = 0.0e0
      PI = X01AAF(X)
      DO 20 I = 4, 5
         TOL = 10.0e0**(-I)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Calculation with TOL =', TOL
         X = 0.0e0
         Y(1) = 0.5e0
         Y(2) = 0.5e0
         Y(3) = PI/5.0e0
         IFAIL = 0
*
         CALL D02BGF(X,XEND,N,Y,TOL,HMAX,M,VAL,FCN,W,IFAIL)
*
         WRITE (NOUT,99998) ' Y(M) changes sign at X = ', X
         IF (TOL.LT.0.0e0) WRITE (NOUT,*)
     +       ' Over one-third steps controlled by HMAX'
   20 CONTINUE
      STOP
*
99999 FORMAT (1X,A,e8.1)
99998 FORMAT (1X,A,F7.4)
      END
*
      SUBROUTINE FCN(T,Y,F)
*     .. Parameters ..
      INTEGER        N
      PARAMETER     (N=3)
*     .. Scalar Arguments ..
      real           T
*     .. Array Arguments ..
      real           F(N), Y(N)
*     .. Intrinsic Functions ..
      INTRINSIC      COS, TAN
*     .. Executable Statements ..
      F(1) = TAN(Y(3))
      F(2) = -0.032e0*TAN(Y(3))/Y(2) - 0.02e0*Y(2)/COS(Y(3))
      F(3) = -0.032e0/Y(2)**2
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D02BGF Example Program Results

Calculation with TOL = 0.1E-03
 Y(M) changes sign at X =  7.2884

Calculation with TOL = 0.1E-04
 Y(M) changes sign at X =  7.2883
```

D02BHF – NAG Fortran Library Routine Document

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

1. Purpose

D02BHF integrates a system of first-order ordinary differential equations over an interval with suitable initial conditions, using a Runge-Kutta-Merson method, until a user-specified function of the solution is zero.

2. Specification

```
 SUBROUTINE D02BHF (X, XEND, N, Y, TOL, IRELAB, HMAX, FCN, G, W,
1                   IFAIL)
 INTEGER         N, IRELAB, IFAIL
 real            X, XEND, Y(N), TOL, HMAX, G, W(N,7)
 EXTERNAL        FCN, G
```

3. Description

The routine advances the solution of a system of ordinary differential equations

$$y_i' = f_i(x,y_1,y_2,...,y_n), \qquad i = 1,2,...,n,$$

from x = X towards x = XEND using a Merson form of the Runge-Kutta method. The system is defined by a subroutine FCN supplied by the user, which evaluates f_i in terms of x and $y_1,y_2,...,y_n$ (see Section 5), and the values of $y_1,y_2,...,y_n$ must be given at x = X.

As the integration proceeds, a check is made on the function $g(x,y)$ specified by the user, to determine an interval where it changes sign. The position of this sign change is then determined accurately by interpolating for the solution and its derivative. It is assumed that $g(x,y)$ is a continuous function of the variables, so that a solution of $g(x,y) = 0$ can be determined by searching for a change in sign in $g(x,y)$.

The accuracy of the integration and, indirectly, of the determination of the position where $g(x,y) = 0$, is controlled by the parameter TOL.

For a description of Runge-Kutta methods and their practical implementation see Hall and Watt [1].

4. References

[1] HALL, G. and WATT, J.M. (eds.)
Modern Numerical Methods for Ordinary Differential Equations.
Clarendon Press, Oxford, p. 59, 1976.

5. Parameters

1: X – ***real***. *Input/Output*

On entry: X must be set to the initial value of the independent variable x.

On exit: the point where $g(x,y) = 0.0$ unless an error has occurred, when it contains the value of x at the error. In particular, if $g(x,y) \neq 0.0$ anywhere on the range X to XEND, it will contain XEND on exit.

2: XEND – ***real***. *Input*

On entry: the final value of the independent variable x.

If XEND < X on entry, integration proceeds in a negative direction.

3: N – INTEGER. *Input*

On entry: the number of differential equations, n.

Constraint: N > 0.

4: Y(N) – ***real*** array. *Input/Output*

On entry: the initial values of the solution $y_1, y_2, ..., y_n$.

On exit: the computed values of the solution at the final point x = X.

5: TOL – ***real.*** *Input/Output*

On entry: TOL must be set to a **positive** tolerance for controlling the error in the integration and in the determination of the position where $g(x,y)$ = 0.0.

D02BHF has been designed so that, for most problems, a reduction in TOL leads to an approximately proportional reduction in the error in the solution obtained in the integration. The relation between changes in TOL and the error in the determination of the position where $g(x,y)$ = 0.0 is less clear, but for TOL small enough the error should be approximately proportional to TOL. However, the actual relation between TOL and the accuracy cannot be guaranteed. The user is strongly recommended to call D02BHF with more than one value for TOL and to compare the results obtained to estimate their accuracy. In the absence of any prior knowledge the user might compare results obtained by calling D02BHF with TOL = 10.0^{-p} and TOL = 10.0^{-p-1} if p correct decimal digits in the solution are required.

Constraint: TOL > 0.0.

On exit: normally unchanged. However if the range from x = X to the position where $g(x,y)$ = 0.0 (or to the final value of x if an error occurs) is so short that a small change in TOL is unlikely to make any change in the computed solution, then TOL is returned with its sign changed. To check results returned with TOL < 0.0, D02BHF should be called again with a positive value of TOL whose magnitude is considerably smaller than that of the previous call.

6: IRELAB – INTEGER. *Input*

On entry: IRELAB determines the type of error control. At each step in the numerical solution an estimate of the local error, EST, is made. For the current step to be accepted the following condition must be satisfied:

IRELAB = 0

$\mathrm{EST} \le \mathrm{TOL}\times\max\{1.0,|y_1|,|y_2|,...,|y_n|\}$;

IRELAB = 1

$\mathrm{EST} \le \mathrm{TOL}$;

IRELAB = 2

$\mathrm{EST} \le \mathrm{TOL}\times\max\{\varepsilon,|y_1|,|y_2|,...,|y_n|\}$,

where ε is ***machine precision***.

If the appropriate condition is not satisfied, the stepsize is reduced and the solution recomputed on the current step.

If the user wishes to measure the error in the computed solution in terms of the number of correct decimal places, then IRELAB should be given the value 1 on entry, whereas if the error requirement is in terms of the number of correct significant digits, then IRELAB should be given the value 2. Where there is no preference in the choice of error test, IRELAB = 0 will result in a mixed error test. It should be borne in mind that the computed solution will be used in evaluating $g(x,y)$.

Constraint: 0 ≤ IRELAB ≤ 2.

7: HMAX – ***real***. *Input*

On entry: if HMAX = 0.0, no special action is taken.

If HMAX ≠ 0.0, a check is made for a change in sign of $g(x,y)$ at steps not greater than |HMAX|. This facility should be used if there is any chance of 'missing' the change in sign by checking too infrequently. For example, if two changes of sign of $g(x,y)$ are expected within a distance h, say, of each other, then a suitable value for HMAX might be HMAX = $h/2$. If only one change of sign in $g(x,y)$ is expected on the range X to XEND, then the choice HMAX = 0.0 is most appropriate.

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

FCN must evaluate the functions f_i (i.e. the derivatives y_i') for given values of its arguments $x,y_1,...,y_n$.

Its specification is:

```
SUBROUTINE FCN(X, Y, F)
real          X, Y(n), F(n)
```

where n is the actual value of N in the call of D02BHF.

1: X – ***real***. *Input*

On entry: the value of the argument x.

2: Y(n) – ***real*** array. *Input*

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

3: F(n) – ***real*** array. *Output*

On exit: the value of f_i, for $i = 1,2,...,n$.

FCN must be declared as EXTERNAL in the (sub)program from which D02BHF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

9: G – ***real*** FUNCTION, supplied by the user. *External Procedure*

G must evaluate the function $g(x,y)$ at a specified point.

Its specification is:

```
real FUNCTION G(X, Y)
real          X, Y(n)
```

where n is the actual value of N in the call of D02BHF.

1: X – ***real***. *Input*

On entry: the value of the independent variable x.

2: Y(n) – ***real*** array. *Input*

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

G must be declared as EXTERNAL in the (sub)program from which D02BHF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

10: W(N,7) – ***real*** array. *Workspace*

11: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, TOL ≤ 0.0,
or N ≤ 0,
or IRELAB ≠ 0, 1 or 2.

IFAIL = 2

With the given value of TOL, no further progress can be made across the integration range from the current point x = X, or dependence of the error on TOL would be lost if further progress across the integration range were attempted (see Section 8 for a discussion of this error exit). The components Y(1),Y(2),...,Y(n) contain the computed values of the solution at the current point x = X. No point at which $g(x,y)$ changes sign has been located up to the point x = X.

IFAIL = 3

TOL is too small for the routine to take an initial step (see Section 8). X and Y(1),Y(2),...,Y(n) retain their initial values.

IFAIL = 4

At no point in the range X to XEND did the function $g(x,y)$ change sign. It is assumed that $g(x,y)$ = 0.0 has no solution.

IFAIL = 5

A serious error has occurred an internal call to C05AZF. Check all subroutine calls and array dimensions. Seek expert help.

IFAIL = 6

A serious error has occurred an internal call to an integration routine. Check all subroutine calls and array dimensions. Seek expert help.

IFAIL = 7

A serious error has occurred an internal call to an interpolation routine. Check all subroutine calls and array dimensions. Seek expert help.

7. Accuracy

The accuracy depends on TOL, on the mathematical properties of the differential system, on the position where $g(x,y)$ = 0.0 and on the method. It can be controlled by varying TOL but the approximate proportionality of the error to TOL holds only for a restricted range of values of TOL. For TOL too large, the underlying theory may break down and the result of varying TOL may be unpredictable. For TOL too small, rounding error may affect the solution significantly and an error exit with IFAIL = 2 or IFAIL = 3 is possible.

The accuracy may also be restricted by the properties of $g(x,y)$. The user should try to code G without introducing any unnecessary cancellation errors.

8. Further Comments

The time taken by the routine depends on the complexity and mathematical properties of the system of differential equations defined by FCN, the complexity of G, on the range, the position of the solution and the tolerance. There is also an overhead of the form $a + b \times n$ where a and b are machine-dependent computing times.

For some problems it is possible that D02BHF will return IFAIL = 4 because of inaccuracy of the computed values Y, leading to inaccuracy in the computed values of $g(x,y)$ used in the search for the solution of $g(x,y) = 0.0$. This difficulty can be overcome by reducing TOL sufficiently, and if necessary, by choosing HMAX sufficiently small. If possible, the user should choose XEND well beyond the expected point where $g(x,y) = 0.0$; for example make |XEND−X| about 50% larger than the expected range. As a simple check, if, with XEND fixed, a change in TOL does not lead to a significant change in Y at XEND, then inaccuracy is not a likely source of error.

If the routine fails with IFAIL = 3, then it could be called again with a larger value of TOL if this has not already been tried. If the accuracy requested is really needed and cannot be obtained with this routine, the system may be very stiff (see below) or so badly scaled that it cannot be solved to the required accuracy.

If the routine fails with IFAIL = 2, it is likely that it has been called with a value of TOL which is so small that a solution cannot be obtained on the range X to XEND. This can happen for well-behaved systems and very small values of TOL. The user should, however, consider whether there is a more fundamental difficulty. For example:

(a) in the region of a singularity (infinite value) of the solution, the routine will usually stop with IFAIL = 2, unless overflow occurs first. If overflow occurs using D02BHF, D02PDF can be used instead to detect the increasing solution, before overflow occurs. In any case, numerical integration cannot be continued through a singularity, and analytical treatment should be considered;

(b) for 'stiff' equations, where the solution contains rapidly decaying components, the routine will compute in very small steps in x (internally to D02BHF) to preserve stability. This will usually exhibit itself by making the computing time excessively long, or occasionally by an exit with IFAIL = 2. Merson's method is not efficient in such cases, and the user should try D02EJF which uses a Backward Differentiation Formula method. To determine whether a problem is stiff, D02PCF may be used.

For well-behaved systems with no difficulties such as stiffness or singularities, the Merson method should work well for low accuracy calculations (three or four figures). For high accuracy calculations or where FCN is costly to evaluate, Merson's method may not be appropriate and a computationally less expensive method may be D02CJF which uses an Adams method.

For problems for which D02BHF is not sufficiently general, the user should consider D02PDF. D02PDF is a more general routine with many facilities including a more general error control criterion. D02PDF can be combined with the rootfinder C05AZF and the interpolation routine D02PXF to solve equations involving $y_1, y_2, \ldots, y_n$ and their derivatives.

D02BHF can also be used to solve an equation involving $x, y_1, y_2, \ldots, y_n$ and the derivatives of $y_1, y_2, \ldots, y_n$. For example in Section 9, D02BHF is used to find a value of X > 0.0 where Y(1) = 0.0. It could instead be used to find a turning-point of y_1 by replacing the function $g(x,y)$ in the program by:

```
real FUNCTION G(X,Y)
real X,Y(3),F(3)
CALL FCN(X,Y,F)
G = F(1)
RETURN
END
```

This routine is only intended to locate the **first** zero of $g(x,y)$. If later zeros are required, users are strongly advised to construct their own more general root finding routines as discussed above.

9. Example

To find the value X > 0.0 at which $y = 0.0$, where y, v, ϕ are defined by

$$y' = \tan\phi$$

$$v' = \frac{-0.032\tan\phi}{v} - \frac{0.02v}{\cos\phi}$$

$$\phi' = \frac{-0.032}{v^2}$$

and where at X = 0.0 we are given $y = 0.5$, $v = 0.5$ and $\phi = \pi/5$. We write $y =$ Y(1), $v =$ Y(2) and $\phi =$ Y(3) and we set TOL = 1.0E−4 and TOL = 1.0E−5 in turn so that we can compare the solutions. We expect the solution X $\simeq$ 7.3 and so we set XEND = 10.0 to avoid determining the solution of $y = 0.0$ too near the end of the range of integration. The value of π is obtained by using X01AAF.

9.1. Program Text

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

```
*     D02BHF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          N
      PARAMETER        (N=3)
*     .. Local Scalars ..
      real             HMAX, PI, TOL, X, XEND
      INTEGER          I, IFAIL, IRELAB, J
*     .. Local Arrays ..
      real             W(N,7), Y(N)
*     .. External Functions ..
      real             G, X01AAF
      EXTERNAL         G, X01AAF
*     .. External Subroutines ..
      EXTERNAL         D02BHF, FCN
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02BHF Example Program Results'
      XEND = 10.0e0
      HMAX = 0.0e0
      IRELAB = 0
      PI = X01AAF(X)
      DO 20 J = 4, 5
         TOL = 10.0e0**(-J)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Calculation with TOL =', TOL
         X = 0.0e0
         Y(1) = 0.5e0
         Y(2) = 0.5e0
         Y(3) = 0.2e0*PI
         IFAIL = 0
*
         CALL D02BHF(X,XEND,N,Y,TOL,IRELAB,HMAX,FCN,G,W,IFAIL)
*
         WRITE (NOUT,99998) ' Root of Y(1) at', X
         WRITE (NOUT,99997) ' Solution is', (Y(I),I=1,N)
         IF (TOL.LT.0.0e0) WRITE (NOUT,*)
     +       ' Over one-third steps controlled by HMAX'
   20 CONTINUE
      STOP
*
99999 FORMAT (1X,A,e8.1)
99998 FORMAT (1X,A,F7.4)
99997 FORMAT (1X,A,3F13.5)
      END
```

```
*
      SUBROUTINE FCN(T,Y,F)
*     .. Parameters ..
      INTEGER        N
      PARAMETER     (N=3)
*     .. Scalar Arguments ..
      real           T
*     .. Array Arguments ..
      real           F(N), Y(N)
*     .. Intrinsic Functions ..
      INTRINSIC     COS, TAN
*     .. Executable Statements ..
      F(1) = TAN(Y(3))
      F(2) = -0.032e0*TAN(Y(3))/Y(2) - 0.02e0*Y(2)/COS(Y(3))
      F(3) = -0.032e0/Y(2)**2
      RETURN
      END
*
      real FUNCTION G(T,Y)
*     .. Parameters ..
      INTEGER        N
      PARAMETER     (N=3)
*     .. Scalar Arguments ..
      real           T
*     .. Array Arguments ..
      real           Y(N)
*     .. Executable Statements ..
      G = Y(1)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D02BHF Example Program Results

Calculation with TOL = 0.1E-03
 Root of Y(1) at 7.2884
 Solution is      0.00000      0.47485     -0.76010

Calculation with TOL = 0.1E-04
 Root of Y(1) at 7.2883
 Solution is      0.00000      0.47486     -0.76011
```

D02BJF – NAG Fortran Library Routine Document

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

1 Purpose

D02BJF integrates a system of first-order ordinary differential equations over an interval with suitable initial conditions, using a fixed order Runge–Kutta method (RK), until a user-specified function, if supplied, of the solution is zero, and returns the solution at points specified by the user, if desired.

2 Specification

```
   SUBROUTINE D02BJF(X, XEND, N, Y, FCN, TOL, RELABS, OUTPUT, G, W,
  1                  IFAIL)
   real              X, XEND, Y(N), TOL, G, W(20*N)
   INTEGER           N, IFAIL
   CHARACTER*1       RELABS
   EXTERNAL          FCN, OUTPUT, G
```

3 Description

The routine advances the solution of a system of ordinary differential equations

$$y_i' = f_i(x, y_1, y_2, \ldots, y_n), \quad i = 1, 2, \ldots, n,$$

from $x = \mathrm{X}$ to $x = \mathrm{XEND}$ using a fixed order Runge–Kutta method. The system is defined by a subroutine FCN supplied by the user, which evaluates f_i in terms of x and $y = (y_1, y_2, \ldots, y_n)$. The initial values of $y = (y_1, y_2, \ldots, y_n)$ must be given at $x = \mathrm{X}$.

The solution is returned via the user-supplied subroutine OUTPUT at points specified by the user, if desired: this solution is obtained by C^1 interpolation on solution values produced by the method. As the integration proceeds a check can be made on the user-specified function $g(x, y)$ to determine an interval where it changes sign. The position of this sign change is then determined accurately by C^1 interpolation to the solution. It is assumed that $g(x, y)$ is a continuous function of the variables, so that a solution of $g(x, y) = 0$ can be determined by searching for a change in sign in $g(x, y)$. The accuracy of the integration, the interpolation and, indirectly, of the determination of the position where $g(x, y) = 0$, is controlled by the parameters TOL and RELABS.

4 References

[1] Shampine L F (1994) *Numerical solution of ordinary differential equations* Chapman and Hall

5 Parameters

1: X — ***real*** *Input/Output*

On entry: the initial value of the independent variable x.

On exit: if g is supplied by the user, it contains the point where $g(x, y) = 0$, unless $g(x, y) \neq 0$ anywhere on the range X to XEND, in which case, X will contain XEND (and the error indicator IFAIL = 6 is set); if g is not supplied by the user it contains XEND. However, if an error has occurred, it contains the value of x at which the error occurred.

2: XEND — ***real*** *Input*

On entry: the final value of the independent variable. If XEND < X, integration will proceed in the negative direction.

Constraint: XEND $\neq$ X.

3: N — INTEGER *Input*

On entry: the number of equations, n.

Constraint: N > 0.

4: Y(N) — ***real*** array *Input/Output*

On entry: the initial values of the solution $y_1, y_2, \ldots, y_n$ at $x = $ X.

On exit: the computed values of the solution at the final point $x = $ X.

5: FCN — SUBROUTINE, supplied by the user. *External Procedure*

FCN must evaluate the functions f_i (i.e., the derivatives y_i') for given values of its arguments $x, y_1, \ldots, y_n$.

Its specification is:

```
SUBROUTINE FCN(X, Y, F)
real           X, Y(*), F(*)
```

1: X — ***real*** *Input*

On entry: the value of the independent variable x.

2: Y(∗) — ***real*** array *Input*

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

3: F(∗) — ***real*** array *Output*

On exit: the value of f_i, for $i = 1, 2, \ldots, n$.

FCN must be declared as EXTERNAL in the (sub)program from which D02BJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: TOL — ***real*** *Input*

On entry: a **positive** tolerance for controlling the error in the integration. Hence TOL affects the determination of the position where $g(x, y) = 0$, if g is supplied.

D02BJF has been designed so that, for most problems, a reduction in TOL leads to an approximately proportional reduction in the error in the solution. However, the actual relation between TOL and the accuracy achieved cannot be guaranteed. The user is strongly recommended to call D02BJF with more than one value for TOL and to compare the results obtained to estimate their accuracy. In the absence of any prior knowledge, the user might compare the results obtained by calling D02BJF with RELABS set to 'D' and with each of TOL $= 10.0^{-p}$ and TOL $= 10.0^{-p-1}$ where p correct significant digits are required in the solution, y. The accuracy of the value x such that $g(x, y) = 0$ is indirectly controlled by varying TOL. The user should experiment to determine this accuracy.

Constraint: 10.0 × ***machine precision*** < TOL < 0.01.

7: RELABS — CHARACTER*1 *Input*

On entry: the type of error control. At each step in the numerical solution an estimate of the local error, EST, is made. For the current step to be accepted the following condition must be satisfied:

$$\mathrm{EST} = \max(e_i/(\tau_r \times \max(|y_i|, \tau_a))) \leq 1.0$$

where τ_r and τ_a are defined by

RELABS	τ_r	τ_a
'M'	TOL	1.0
'A'	ϵ_r	TOL/ϵ_r
'R'	TOL	ϵ_a
'D'	TOL	ϵ_a

where ϵ_r and ϵ_a are small machine-dependent numbers and e_i is an estimate of the local error at y_i, computed internally. If the condition is not satisfied, the step size is reduced and the solution is recomputed on the current step. If the user wishes to measure the error in the computed solution in terms of the number of correct decimal places, then RELABS should be set to 'A' on entry, whereas if the error requirement is in terms of the number of correct significant digits, then RELABS should be set to 'R'. If the user prefers a mixed error test, then RELABS should be set to 'M', otherwise if the user has no preference, RELABS should be set to the default 'D'. Note that in this case 'D' is taken to be 'R'.

Constraint: RELABS = 'M', 'A', 'R', 'D'.

8: OUTPUT — SUBROUTINE, supplied by the user. *External Procedure*

OUTPUT permits access to intermediate values of the computed solution (for example to print or plot them), at successive user-specified points. It is initially called by D02BJF with XSOL = X (the initial value of x). The user must reset XSOL to the next point (between the current XSOL and XEND) where OUTPUT is to be called, and so on at each call to OUTPUT. If, after a call to OUTPUT, the reset point XSOL is beyond XEND, D02BJF will integrate to XEND with no further calls to OUTPUT; if a call to OUTPUT is required at the point XSOL = XEND, then XSOL must be given precisely the value XEND.

Its specification is:

```
SUBROUTINE OUTPUT(XSOL, Y)
real              XSOL, Y(*)
```

1: XSOL — ***real*** *Input/Output*

On entry: the output value of the independent variable x.

On exit: the user must set XSOL to the next value of x at which OUTPUT is to be called.

2: Y(*) — ***real*** array *Input*

On entry: the computed solution at the point XSOL.

If the user does not wish to access intermediate output, the actual argument OUTPUT **must** be the dummy routine D02BJX. (D02BJX is included in the NAG Fortran Library and so need not be supplied by the user. The name may be implementation-dependent: see the Users' Note for your implementation for details.)

OUTPUT must be declared as EXTERNAL in the (sub)program from which D02BJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

9: G — ***real*** FUNCTION, supplied by the user. *External Procedure*

G must evaluate the function $g(x,y)$ for specified values x,y. It specifies the function g for which the first position x where $g(x,y) = 0$ is to be found.

Its specification is:

```
real FUNCTION G(X, Y)
real            X, Y(*)
```

1: X — ***real*** *Input*

On entry: the value of the independent variable x.

2: Y(*) — ***real*** array *Input*

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

If the user does not require the root finding option, the actual argument G **must** be the dummy routine D02BJW. (D02BJW is included in the NAG Fortran Library and so need not be supplied by the user. The name may be implementation-dependent: see the Users' Note for your implementation for details.)
G must be declared as EXTERNAL in the (sub)program from which D02BJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

10: W(20*N) — ***real*** array *Workspace*

11: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Errors and Warnings

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

Errors detected by the routine:

IFAIL = 1

On entry, TOL $\geq$ 0.01,
or TOL is too small
or N $\leq$ 0,
or RELABS $\neq$ 'M', 'A', 'R' or 'D',
or X = XEND.

IFAIL = 2

With the given value of TOL, no further progress can be made across the integration range from the current point x = X. (See Section 8 for a discussion of this error exit.) The components Y(1),Y(2),...,Y(N) contain the computed values of the solution at the current point x = X. If the user has supplied g, then no point at which $g(x,y)$ changes sign has been located up to the point x = X.

IFAIL = 3

TOL is too small for D02BJF to take an initial step. X and Y(1),Y(2),...,Y(N) retain their initial values.

IFAIL = 4

XSOL has not been reset or XSOL lies behind X in the direction of integration, after the initial call to OUTPUT, if the OUTPUT option was selected.

IFAIL = 5

A value of XSOL returned by OUTPUT has not been reset or lies behind the last value of XSOL in the direction of integration, if the OUTPUT option was selected.

IFAIL = 6

At no point in the range X to XEND did the function $g(x,y)$ change sign, if g was supplied. It is assumed that $g(x,y) = 0$ has no solution.

IFAIL = 7

A serious error has occurred in an internal call to an interpolation routine. Check all subroutine calls and array dimensions. Seek expert help.

7 Accuracy

The accuracy of the computation of the solution vector Y may be controlled by varying the local error tolerance TOL. In general, a decrease in local error tolerance should lead to an increase in accuracy. Users are advised to choose RELABS = 'D' unless they have a good reason for a different choice.

If the problem is a root-finding one, then the accuracy of the root determined will depend on the properties of $g(x, y)$ and on the values of TOL and RELABS. The user should try to code G without introducing any unnecessary cancellation errors.

8 Further Comments

If more than one root is required, then to determine the second and later roots D02BJF may be called again starting a short distance past the previously determined roots. Alternatively the user may construct his own root finding code using D02PDF, D02PXF and C05AZF.

If the routine fails with IFAIL = 3, then it can be called again with a larger value of TOL if this has not already been tried. If the accuracy requested is really needed and cannot be obtained with this routine, the system may be very stiff (see below) or so badly scaled that it cannot be solved to the required accuracy.

If the routine fails with IFAIL = 2, it is probable that it has been called with a value of TOL which is so small that a solution cannot be obtained on the range X to XEND. This can happen for well-behaved systems and very small values of TOL. The user should, however, consider whether there is a more fundamental difficulty. For example:

(a) in the region of a singularity (infinite value) of the solution, the routine will usually stop with IFAIL = 2, unless overflow occurs first. Numerical integration cannot be continued through a singularity, and analytic treatment should be considered;

(b) for 'stiff' equations where the solution contains rapidly decaying components, the routine will use very small steps in x (internally to D02BJF) to preserve stability. This will exhibit itself by making the computing time excessively long, or occasionally by an exit with IFAIL = 2. Runge–Kutta methods are not efficient in such cases, and the user should try D02EJF.

9 Example

We illustrate the solution of four different problems. In each case the differential system (for a projectile) is

$$\begin{aligned} y' &= \tan\phi \\ v' &= \frac{-0.032\tan\phi}{v} - \frac{0.02v}{\cos\phi} \\ \phi' &= \frac{-0.032}{v^2} \end{aligned}$$

over an interval X = 0.0 to XEND = 10.0 starting with values $y = 0.5$, $v = 0.5$ and $\phi = \pi/5$. We solve each of the following problems with local error tolerances 1.0E−4 and 1.0E−5.

(i) To integrate to $x = 10.0$ producing intermediate output at intervals of 2.0 until a root is encountered where $y = 0.0$.

(ii) As (i) but with no intermediate output.

(iii) As (i) but with no termination on a root-finding condition.

(iv) As (i) but with no intermediate output and no root-finding termination condition.

9.1 Program Text

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

```
*     D02BJF Example Program Text
*     Mark 18 Release. NAG Copyright 1997.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          N, IW
      PARAMETER        (N=3,IW=20*N)
*     .. Scalars in Common ..
      real             H, XEND
      INTEGER          K
*     .. Local Scalars ..
      real             PI, TOL, X
      INTEGER          I, IFAIL, J
*     .. Local Arrays ..
      real             W(IW), Y(N)
*     .. External Functions ..
      real             D02BJW, G, X01AAF
      EXTERNAL         D02BJW, G, X01AAF
*     .. External Subroutines ..
      EXTERNAL         D02BJF, D02BJX, FCN, OUT
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Common blocks ..
      COMMON           XEND, H, K
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02BJF Example Program Results'
      XEND = 10.0e0
      PI = X01AAF(0.0e0)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 1: intermediate output, root-finding'
      DO 20 J = 4, 5
         TOL = 10.0e0**(-J)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) ' Calculation with TOL =', TOL
         X = 0.0e0
         Y(1) = 0.5e0
         Y(2) = 0.5e0
         Y(3) = PI/5.0e0
         K = 4
         H = (XEND-X)/real(K+1)
         WRITE (NOUT,*) '     X         Y(1)         Y(2)         Y(3)'
         IFAIL = 0
*
         CALL D02BJF(X,XEND,N,Y,FCN,TOL,'Default',OUT,G,W,IFAIL)
*
         WRITE (NOUT,99998) '  Root of Y(1) = 0.0 at', X
         WRITE (NOUT,99997) '  Solution is', (Y(I),I=1,N)
   20 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 2: no intermediate output, root-finding'
      DO 40 J = 4, 5
         TOL = 10.0e0**(-J)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) ' Calculation with TOL =', TOL
         X = 0.0e0
         Y(1) = 0.5e0
         Y(2) = 0.5e0
         Y(3) = PI/5.0e0
         IFAIL = 0
*
         CALL D02BJF(X,XEND,N,Y,FCN,TOL,'Default',D02BJX,G,W,IFAIL)
*
         WRITE (NOUT,99998) '  Root of Y(1) = 0.0 at', X
         WRITE (NOUT,99997) '  Solution is', (Y(I),I=1,N)
   40 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 3: intermediate output, no root-finding'
      DO 60 J = 4, 5
         TOL = 10.0e0**(-J)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) ' Calculation with TOL =', TOL
         X = 0.0e0
         Y(1) = 0.5e0
         Y(2) = 0.5e0
         Y(3) = PI/5.0e0
         K = 4
         H = (XEND-X)/real(K+1)
         WRITE (NOUT,*) '     X         Y(1)         Y(2)         Y(3)'
         IFAIL = 0
*
         CALL D02BJF(X,XEND,N,Y,FCN,TOL,'Default',OUT,D02BJW,W,IFAIL)
*
   60 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     +'Case 4: no intermediate output, no root-finding ( integrate to XE
     +ND)'
      DO 80 J = 4, 5
         TOL = 10.0e0**(-J)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) ' Calculation with TOL =', TOL
         X = 0.0e0
         Y(1) = 0.5e0
         Y(2) = 0.5e0
         Y(3) = PI/5.0e0
         WRITE (NOUT,*) '     X         Y(1)         Y(2)         Y(3)'
         WRITE (NOUT,99996) X, (Y(I),I=1,N)
         IFAIL = 0
*
         CALL D02BJF(X,XEND,N,Y,FCN,TOL,'Default',D02BJX,D02BJW,W,IFAIL)
*
         WRITE (NOUT,99996) X, (Y(I),I=1,N)
   80 CONTINUE
      STOP
*
```

```
99999 FORMAT (1X,A,e8.1)
99998 FORMAT (1X,A,F7.3)
99997 FORMAT (1X,A,3F13.4)
99996 FORMAT (1X,F8.2,3F13.4)
      END
*
      SUBROUTINE OUT(X,Y)
*     .. Parameters ..
      INTEGER        NOUT
      PARAMETER      (NOUT=6)
      INTEGER        N
      PARAMETER      (N=3)
*     .. Scalar Arguments ..
      real           X
*     .. Array Arguments ..
      real           Y(N)
*     .. Scalars in Common ..
      real           H, XEND
      INTEGER        I
*     .. Local Scalars ..
      INTEGER        J
*     .. Intrinsic Functions ..
      INTRINSIC      real
*     .. Common blocks ..
      COMMON         XEND, H, I
*     .. Executable Statements ..
      WRITE (NOUT,99999) X, (Y(J),J=1,N)
      X = XEND - real(I)*H
      I = I - 1
      RETURN
*
99999 FORMAT (1X,F8.2,3F13.4)
      END
*
      SUBROUTINE FCN(T,Y,F)
*     .. Parameters ..
      INTEGER        N
      PARAMETER      (N=3)
*     .. Scalar Arguments ..
      real           T
*     .. Array Arguments ..
      real           F(N), Y(N)
*     .. Intrinsic Functions ..
      INTRINSIC      COS, TAN
*     .. Executable Statements ..
      F(1) = TAN(Y(3))
      F(2) = -0.032e0*TAN(Y(3))/Y(2) - 0.02e0*Y(2)/COS(Y(3))
      F(3) = -0.032e0/Y(2)**2
      RETURN
      END
*
      real FUNCTION G(T,Y)
*     .. Parameters ..
      INTEGER        N
      PARAMETER      (N=3)
*     .. Scalar Arguments ..
      real           T
```

```
*     .. Array Arguments ..
      real              Y(N)
*     .. Executable Statements ..
      G = Y(1)
      RETURN
      END
```

9.2 Program Data

None.

9.3 Program Results

```
 D02BJF Example Program Results

 Case 1: intermediate output, root-finding

  Calculation with TOL = 0.1E-03
      X          Y(1)          Y(2)          Y(3)
     0.00        0.5000        0.5000        0.6283
     2.00        1.5493        0.4055        0.3066
     4.00        1.7423        0.3743       -0.1289
     6.00        1.0055        0.4173       -0.5507
   Root of Y(1) = 0.0 at  7.288
   Solution is       0.0000        0.4749       -0.7601

  Calculation with TOL = 0.1E-04
      X          Y(1)          Y(2)          Y(3)
     0.00        0.5000        0.5000        0.6283
     2.00        1.5493        0.4055        0.3066
     4.00        1.7423        0.3743       -0.1289
     6.00        1.0055        0.4173       -0.5507
   Root of Y(1) = 0.0 at  7.288
   Solution is       0.0000        0.4749       -0.7601


 Case 2: no intermediate output, root-finding

  Calculation with TOL = 0.1E-03
   Root of Y(1) = 0.0 at  7.288
   Solution is       0.0000        0.4749       -0.7601

  Calculation with TOL = 0.1E-04
   Root of Y(1) = 0.0 at  7.288
   Solution is       0.0000        0.4749       -0.7601


 Case 3: intermediate output, no root-finding

  Calculation with TOL = 0.1E-03
      X          Y(1)          Y(2)          Y(3)
     0.00        0.5000        0.5000        0.6283
     2.00        1.5493        0.4055        0.3066
     4.00        1.7423        0.3743       -0.1289
     6.00        1.0055        0.4173       -0.5507
     8.00       -0.7460        0.5130       -0.8537
    10.00       -3.6283        0.6333       -1.0515
 Calculation with TOL = 0.1E-04
     X          Y(1)          Y(2)         Y(3)
   0.00        0.5000        0.5000       0.6283
   2.00        1.5493        0.4055       0.3066
   4.00        1.7423        0.3743      -0.1289
   6.00        1.0055        0.4173      -0.5507
   8.00       -0.7459        0.5130      -0.8537
  10.00       -3.6282        0.6333      -1.0515

Case 4: no intermediate output, no root-finding ( integrate to XEND)

 Calculation with TOL = 0.1E-03
     X          Y(1)          Y(2)         Y(3)
   0.00        0.5000        0.5000       0.6283
  10.00       -3.6283        0.6333      -1.0515

 Calculation with TOL = 0.1E-04
     X          Y(1)          Y(2)         Y(3)
   0.00        0.5000        0.5000       0.6283
  10.00       -3.6282        0.6333      -1.0515
```

D02CJF – NAG Fortran Library Routine Document

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

1. Purpose

D02CJF integrates a system of first-order ordinary differential equations over a range with suitable initial conditions, using a variable-order, variable-step Adams method until a user-specified function, if supplied, of the solution is zero, and returns the solution at points specified by the user, if desired.

2. Specification

```
      SUBROUTINE D02CJF (X, XEND, N, Y, FCN, TOL, RELABS, OUTPUT, G,
     1                   W, IFAIL)
      INTEGER            N, IFAIL
      real               X, XEND, Y(N), TOL, G, W(28+21*N)
      CHARACTER*1        RELABS
      EXTERNAL           FCN, OUTPUT, G
```

3. Description

The routine advances the solution of a system of ordinary differential equations

$$y_i' = f_i(x,y_1,y_2,...,y_n), \qquad i = 1,2,...,n,$$

from x = X to x = XEND using a variable-order, variable-step Adams method. The system is defined by a subroutine FCN supplied by the user, which evaluates f_i in terms of x and $y_1,y_2,...,y_n$. The initial values of $y_1,y_2,...,y_n$ must be given at x = X.

The solution is returned via the user-supplied routine OUTPUT at points specified by the user, if desired: this solution is obtained by C^1 interpolation on solution values produced by the method. As the integration proceeds a check can be made on the user-specified function $g(x,y)$ to determine an interval where it changes sign. The position of this sign change is then determined accurately by C^1 interpolation to the solution. It is assumed that $g(x,y)$ is a continuous function of the variables, so that a solution of $g(x,y) = 0.0$ can be determined by searching for a change in sign in $g(x,y)$. The accuracy of the integration, the interpolation and, indirectly, of the determination of the position where $g(x,y) = 0.0$, is controlled by the parameters TOL and RELABS.

For a description of Adams methods and their practical implementation see Hall and Watt [1].

4. References

[1] HALL, G. and WATT, J.M. (eds).
Modern Numerical Methods for Ordinary Differential Equations.
Clarendon Press, Oxford, 1976.

5. Parameters

1: X – ***real***. *Input/Output*

On entry: the initial value of the independent variable x.

Constraint: X ≠ XEND.

On exit: if g is supplied by the user, it contains the point where $g(x,y) = 0.0$, unless $g(x,y) \neq 0.0$ anywhere on the range X to XEND, in which case, X will contain XEND. If g is not supplied by the user it contains XEND, unless an error has occurred, when it contains the value of x at the error.

2: XEND – ***real***. *Input*

On entry: the final value of the independent variable. If XEND < X, integration proceeds in the negative direction.

Constraint: XEND ≠ X.

3: N – INTEGER. *Input*

On entry: the number of differential equations.

Constraint: N ≥ 1.

4: Y(N) – ***real*** array. *Input/Output*

On entry: the initial values of the solution $y_1,y_2,...,y_n$ at x = X.

On exit: the computed values of the solution at the final point x = XEND.

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

FCN must evaluate the functions f_i (i.e. the derivatives y_i') for given values of their arguments $x,y_1,y_2,...,y_n$.

Its specification is:

```
SUBROUTINE FCN (X, Y, F)
real         X, Y(n), F(n)
```

where n is the actual value of N in the call of D02CJF.

1: X – ***real***. *Input*

On entry: the value of the independent variable x.

2: Y(n) – ***real*** array. *Input*

On entry: the value of the variable y_i, for i = 1,2,...,n.

3: F(n) – ***real*** array. *Output*

On exit: the value of f_i, for i = 1,2,...,n.

FCN must be declared as EXTERNAL in the (sub)program from which D02CJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: TOL – ***real***. *Input*

On entry: a **positive** tolerance for controlling the error in the integration. Hence TOL affects the determination of the position where $g(x,y) = 0.0$, if g is supplied.

D02CJF has been designed so that, for most problems, a reduction in TOL leads to an approximately proportional reduction in the error in the solution. However, the actual relation between TOL and the accuracy achieved cannot be guaranteed. The user is strongly recommended to call D02CJF with more than one value for TOL and to compare the results obtained to estimate their accuracy. In the absence of any prior knowledge, the user might compare the results obtained by calling D02CJF with TOL = 10.0^{-p} and TOL = 10.0^{-p-1} where p correct decimal digits are required in the solution.

Constraint: TOL > 0.0.

7: RELABS – CHARACTER*1. *Input*

On entry: the type of error control. At each step in the numerical solution an estimate of the local error, EST, is made. For the current step to be accepted the following condition must be satisfied:

$$\mathrm{EST} = \sqrt{\sum_{i=1}^{n} (e_i/(\tau_r \times |y_i| + \tau_a))^2} \le 1.0$$

where τ_r and τ_a are defined by

RELABS	τ_r	τ_a
'M'	TOL	TOL
'A'	0.0	TOL
'R'	TOL	ε
'D'	TOL	TOL

where ε is a small machine-dependent number and e_i is an estimate of the local error at y_i, computed internally. If the appropriate condition is not satisfied, the stepsize is reduced and the solution is recomputed on the current step. If the user wishes to measure the error in the computed solution in terms of the number of correct decimal places, then RELABS should be set to 'A' on entry, whereas if the error requirement is in terms of the number of correct significant digits, then RELABS should be set to 'R'. If the user prefers a mixed error test, then RELABS should be set to 'M', otherwise if the user has no preference, RELABS should be set to the default 'D'. Note that in this case 'D' is taken to be 'M'.

Constraint: RELABS = 'M', 'A', 'R' or 'D'.

8: OUTPUT – SUBROUTINE, supplied by the user. *External Procedure*

OUTPUT permits access to intermediate values of the computed solution (for example to print or plot them), at successive user specified points. It is initially called by D02CJF with XSOL = X (the initial value of x). The user must reset XSOL to the next point (between the current XSOL and XEND) where OUTPUT is to be called, and so on at each call to OUTPUT. If, after a call to OUTPUT, the reset point XSOL is beyond XEND, D02CJF will integrate to XEND with no further calls to OUTPUT; if a call to OUTPUT is required at the point XSOL = XEND, then XSOL must be given precisely the value XEND.

Its specification is:

```
SUBROUTINE OUTPUT (XSOL, Y)
real         XSOL, Y(n)
```

where n is the actual value of N in the call of D02CJF.

1: XSOL – ***real***. *Input/Output*

On entry: the output value of the independent variable x.

On exit: the user must set XSOL to the next value of x at which OUTPUT is to be called.

2: Y(n) – ***real*** array. *Input*

On entry: the computed solution at the point XSOL.

If the user does not wish to access intermediate output, the actual argument OUTPUT **must** be the dummy routine D02CJX. (D02CJX is included in the NAG Fortran Library and so need not be supplied by the user. The name may be implementation-dependent: see the Users' Note for your implementation for details.)

OUTPUT must be declared as EXTERNAL in the (sub)program from which D02CJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

9: G – ***real*** FUNCTION, supplied by the user. *External Procedure*

G must evaluate the function $g(x,y)$ for specified values x,y. It specifies the function g for which the first position x where $g(x,y) = 0$ is to be found.

Its specification is:

```
real FUNCTION G(X, Y)
real          X, Y(n)
```

where n is the actual value of N in the call of D02CJF.

1: X – ***real.*** *Input*

On entry: the value of the independent variable x.

2: Y(n) – ***real*** array. *Input*

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

If the user does not require the root finding option, the actual argument G **must** be the dummy routine D02CJW. (D02CJW is included in the NAG Fortran Library and so need not be supplied by the user. The name may be implementation-dependent: see the Users' Note for your implementation for details.)

G must be declared as EXTERNAL in the (sub)program from which D02CJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

10: W(28+21*N) – ***real*** array. *Workspace*

11: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, TOL ≤ 0.0,
or N ≤ 0,
or RELABS ≠ 'M', 'A', 'R' or 'D',
or X = XEND.

IFAIL = 2

With the given value of TOL, no further progress can be made across the integration range from the current point x = X. (See Section 8 for a discussion of this error exit.) The components Y(1),Y(2),...,Y(N) contain the computed values of the solution at the current point x = X. If the user has supplied g, then no point at which $g(x,y)$ changes sign has been located up to the point x = X.

IFAIL = 3

TOL is too small for D02CJF to take an initial step. X and Y(1),Y(2),...,Y(N) retain their initial values.

IFAIL = 4

XSOL has not been reset or XSOL lies behind X in the direction of integration, after the initial call to OUTPUT, if the OUTPUT option was selected.

IFAIL = 5

A value of XSOL returned by OUTPUT has not been reset or lies behind the last value of XSOL in the direction of integration, if the OUTPUT option was selected.

IFAIL = 6

At no point in the range X to XEND did the function $g(x,y)$ change sign, if g was supplied. It is assumed that $g(x,y) = 0$ has no solution.

IFAIL = 7

A serious error has occurred in an internal call. Check all subroutine calls and array sizes. Seek expert help.

7. Accuracy

The accuracy of the computation of the solution vector Y may be controlled by varying the local error tolerance TOL. In general, a decrease in local error tolerance should lead to an increase in accuracy. Users are advised to choose RELABS = 'M' unless they have a good reason for a different choice.

If the problem is a root-finding one, then the accuracy of the root determined will depend on the properties of $g(x,y)$. The user should try to code G without introducing any unnecessary cancellation errors.

8. Further Comments

If more than one root is required then D02QFF should be used.

If the routine fails with IFAIL = 3, then it can be called again with a larger value of TOL if this has not already been tried. If the accuracy requested is really needed and cannot be obtained with this routine, the system may be very stiff (see below) or so badly scaled that it cannot be solved to the required accuracy.

If the routine fails with IFAIL = 2, it is probable that it has been called with a value of TOL which is so small that a solution cannot be obtained on the range X to XEND. This can happen for well-behaved systems and very small values of TOL. The user should, however, consider whether there is a more fundamental difficulty. For example:

(a) in the region of a singularity (infinite value) of the solution, the routine will usually stop with IFAIL = 2, unless overflow occurs first. Numerical integration cannot be continued through a singularity, and analytic treatment should be considered;

(b) for 'stiff' equations where the solution contains rapidly decaying components, the routine will use very small steps in x (internally to D02CJF) to preserve stability. This will exhibit itself by making the computing time excessively long, or occasionally by an exit with IFAIL = 2. Adams methods are not efficient in such cases, and the user should try D02EJF.

9. Example

We illustrate the solution of four different problems. In each case the differential system (for a projectile) is

$$y' = \tan\phi$$

$$v' = \frac{-0.032\tan\phi}{v} - \frac{0.02v}{\cos\phi}$$

$$\phi' = \frac{-0.032}{v^2}$$

over an interval X = 0.0 to XEND = 10.0 starting with values $y = 0.5$, $v = 0.5$ and $\phi = \pi/5$. We solve each of the following problems with local error tolerances 1.0E−4 and 1.0E−5.

(i) To integrate to $x = 10.0$ producing output at intervals of 2.0 until a point is encountered where $y = 0.0$.

(ii) As (i) but with no intermediate output.

(iii) As (i) but with no termination on a root-finding condition.

(iv) As (i) but with no intermediate output and no root-finding termination condition.

9.1. Program Text

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

```
*     D02CJF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          N, IW
      PARAMETER        (N=3,IW=21*N+28)
*     .. Scalars in Common ..
      real             H, XEND
      INTEGER          K
*     .. Local Scalars ..
      real             PI, TOL, X
      INTEGER          I, IFAIL, J
*     .. Local Arrays ..
      real             W(IW), Y(N)
*     .. External Functions ..
      real             D02CJW, G, X01AAF
      EXTERNAL         D02CJW, G, X01AAF
*     .. External Subroutines ..
      EXTERNAL         D02CJF, D02CJX, FCN, OUT
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Common blocks ..
      COMMON           XEND, H, K
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02CJF Example Program Results'
      XEND = 10.0e0
      PI = X01AAF(0.0e0)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 1: intermediate output, root-finding'
      DO 20 J = 4, 5
         TOL = 10.0e0**(-J)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) ' Calculation with TOL =', TOL
         X = 0.0e0
         Y(1) = 0.5e0
         Y(2) = 0.5e0
         Y(3) = PI/5.0e0
         K = 4
         H = (XEND-X)/real(K+1)
         WRITE (NOUT,*) '     X         Y(1)         Y(2)         Y(3)'
         IFAIL = 0
*
         CALL D02CJF(X,XEND,N,Y,FCN,TOL,'Default',OUT,G,W,IFAIL)
*
         WRITE (NOUT,99998) '  Root of Y(1) = 0.0 at', X
         WRITE (NOUT,99997) '  Solution is', (Y(I),I=1,N)
   20 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 2: no intermediate output, root-finding'
      DO 40 J = 4, 5
         TOL = 10.0e0**(-J)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) ' Calculation with TOL =', TOL
         X = 0.0e0
         Y(1) = 0.5e0
         Y(2) = 0.5e0
         Y(3) = PI/5.0e0
         IFAIL = 0
*
```

```
         CALL D02CJF(X,XEND,N,Y,FCN,TOL,'Default',D02CJX,G,W,IFAIL)
*
         WRITE (NOUT,99998) '  Root of Y(1) = 0.0 at', X
         WRITE (NOUT,99997) '  Solution is', (Y(I),I=1,N)
   40 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 3: intermediate output, no root-finding'
      DO 60 J = 4, 5
         TOL = 10.0e0**(-J)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) ' Calculation with TOL =', TOL
         X = 0.0e0
         Y(1) = 0.5e0
         Y(2) = 0.5e0
         Y(3) = PI/5.0e0
         K = 4
         H = (XEND-X)/real(K+1)
         WRITE (NOUT,*)     '  X         Y(1)         Y(2)         Y(3)'
         IFAIL = 0
*
         CALL D02CJF(X,XEND,N,Y,FCN,TOL,'Default',OUT,D02CJW,W,IFAIL)
*
   60 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     +'Case 4: no intermediate output, no root-finding ( integrate to XE
     +ND)'
      DO 80 J = 4, 5
         TOL = 10.0e0**(-J)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) ' Calculation with TOL =', TOL
         X = 0.0e0
         Y(1) = 0.5e0
         Y(2) = 0.5e0
         Y(3) = PI/5.0e0
         WRITE (NOUT,*)     '  X         Y(1)         Y(2)         Y(3)'
         WRITE (NOUT,99996) X, (Y(I),I=1,N)
         IFAIL = 0
*
         CALL D02CJF(X,XEND,N,Y,FCN,TOL,'Default',D02CJX,D02CJW,W,IFAIL)
*
         WRITE (NOUT,99996) X, (Y(I),I=1,N)
   80 CONTINUE
      STOP
*
99999 FORMAT (1X,A,e8.1)
99998 FORMAT (1X,A,F7.3)
99997 FORMAT (1X,A,3F13.5)
99996 FORMAT (1X,F8.2,3F13.5)
      END
*
      SUBROUTINE OUT(X,Y)
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          N
      PARAMETER        (N=3)
*     .. Scalar Arguments ..
      real             X
*     .. Array Arguments ..
      real             Y(N)
*     .. Scalars in Common ..
      real             H, XEND
      INTEGER          I
*     .. Local Scalars ..
      INTEGER          J
```

```
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Common blocks ..
      COMMON           XEND, H, I
*     .. Executable Statements ..
      WRITE (NOUT,99999) X, (Y(J),J=1,N)
      X = XEND - real(I)*H
      I = I - 1
      RETURN
*
99999 FORMAT (1X,F8.2,3F13.5)
      END
*
      SUBROUTINE FCN(T,Y,F)
*     .. Parameters ..
      INTEGER          N
      PARAMETER        (N=3)
*     .. Scalar Arguments ..
      real             T
*     .. Array Arguments ..
      real             F(N), Y(N)
*     .. Intrinsic Functions ..
      INTRINSIC        COS, TAN
*     .. Executable Statements ..
      F(1) = TAN(Y(3))
      F(2) = -0.032e0*TAN(Y(3))/Y(2) - 0.02e0*Y(2)/COS(Y(3))
      F(3) = -0.032e0/Y(2)**2
      RETURN
      END
*
      real FUNCTION G(T,Y)
*     .. Parameters ..
      INTEGER          N
      PARAMETER        (N=3)
*     .. Scalar Arguments ..
      real             T
*     .. Array Arguments ..
      real             Y(N)
*     .. Executable Statements ..
      G = Y(1)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D02CJF Example Program Results

Case 1: intermediate output, root-finding

 Calculation with TOL = 0.1E-03
     X         Y(1)         Y(2)         Y(3)
    0.00      0.50000      0.50000      0.62832
    2.00      1.54931      0.40548      0.30662
    4.00      1.74229      0.37433     -0.12890
    6.00      1.00554      0.41731     -0.55068
  Root of Y(1) = 0.0 at  7.288
  Solution is      0.00000      0.47486     -0.76011

 Calculation with TOL = 0.1E-04
     X         Y(1)         Y(2)         Y(3)
    0.00      0.50000      0.50000      0.62832
    2.00      1.54933      0.40548      0.30662
    4.00      1.74232      0.37433     -0.12891
    6.00      1.00552      0.41731     -0.55069
  Root of Y(1) = 0.0 at  7.288
  Solution is      0.00000      0.47486     -0.76010
```

```
Case 2: no intermediate output, root-finding

 Calculation with TOL = 0.1E-03
  Root of Y(1) = 0.0 at  7.288
  Solution is       0.00000       0.47486     -0.76011

 Calculation with TOL = 0.1E-04
  Root of Y(1) = 0.0 at  7.288
  Solution is       0.00000       0.47486     -0.76010


Case 3: intermediate output, no root-finding

 Calculation with TOL = 0.1E-03
     X           Y(1)          Y(2)          Y(3)
    0.00       0.50000       0.50000       0.62832
    2.00       1.54931       0.40548       0.30662
    4.00       1.74229       0.37433      -0.12890
    6.00       1.00554       0.41731      -0.55068
    8.00      -0.74589       0.51299      -0.85371
   10.00      -3.62813       0.63325      -1.05152

 Calculation with TOL = 0.1E-04
     X           Y(1)          Y(2)          Y(3)
    0.00       0.50000       0.50000       0.62832
    2.00       1.54933       0.40548       0.30662
    4.00       1.74232       0.37433      -0.12891
    6.00       1.00552       0.41731      -0.55069
    8.00      -0.74601       0.51299      -0.85372
   10.00      -3.62829       0.63326      -1.05153


Case 4: no intermediate output, no root-finding ( integrate to XEND)

 Calculation with TOL = 0.1E-03
     X           Y(1)          Y(2)          Y(3)
    0.00       0.50000       0.50000       0.62832
   10.00      -3.62813       0.63325      -1.05152

 Calculation with TOL = 0.1E-04
     X           Y(1)          Y(2)          Y(3)
    0.00       0.50000       0.50000       0.62832
   10.00      -3.62829       0.63326      -1.05153
```

D02EJF – NAG Fortran Library Routine Document

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

1. Purpose

D02EJF integrates a stiff system of first-order ordinary differential equations over an interval with suitable initial conditions, using a variable-order, variable-step method implementing the Backward Differentiation Formulae (BDF), until a user-specified function, if supplied, of the solution is zero, and returns the solution at points specified by the user, if desired.

2. Specification

```
      SUBROUTINE D02EJF (X, XEND, N, Y, FCN, PEDERV, TOL, RELABS, OUTPUT,
     1                   G, W, IW, IFAIL)
      INTEGER          N, IW, IFAIL
      real             X, XEND, Y(N), TOL, G, W(IW)
      CHARACTER*1      RELABS
      EXTERNAL         FCN, PEDERV, OUTPUT, G
```

3. Description

The routine advances the solution of a system of ordinary differential equations

$$y_i' = f_i(x,y_1,y_2,...,y_n), \qquad i = 1,2,...,n,$$

from x = X to x = XEND using a variable-order, variable-step method implementing the BDF. The system is defined by a subroutine FCN supplied by the user, which evaluates f_i in terms of x and $y_1,y_2,...,y_n$ (see Section 5). The initial values of $y_1,y_2,...,y_n$ must be given at x = X.

The solution is returned via the user-supplied routine OUTPUT at points specified by the user, if desired: this solution is obtained by C^1 interpolation on solution values produced by the method. As the integration proceeds a check can be made on the user-specified function $g(x,y)$ to determine an interval where it changes sign. The position of this sign change is then determined accurately by C^1 interpolation to the solution. It is assumed that $g(x,y)$ is a continuous function of the variables, so that a solution of $g(x,y) = 0.0$ can be determined by searching for a change in sign in $g(x,y)$. The accuracy of the integration, the interpolation and, indirectly, of the determination of the position where $g(x,y) = 0.0$, is controlled by the parameters TOL and RELABS. The Jacobian of the system $y' = f(x,y)$ may be supplied in routine PEDERV, if it is available.

For a description of BDF and their practical implementation see Hall and Watt [1].

4. References

[1] HALL, G. and WATT, J.M. (eds).
Modern Numerical Methods for Ordinary Differential Equations.
Clarendon Press, Oxford, 1976.

5. Parameters

1: X – ***real***. *Input/Output*

On entry: the initial value of the independent variable x.

Constraint: X ≠ XEND

On exit: if G is supplied by the user, X contains the point where $g(x,y) = 0.0$, unless $g(x,y) \neq 0.0$ anywhere on the range X to XEND, in which case, X will contain XEND. If G is not supplied X contains XEND, unless an error has occured, when it contains the value of x at the error.

2: XEND – ***real.*** *Input*

On entry: the final value of the independent variable. If XEND < X, integration proceeds in the negative direction.

Constraint: XEND ≠ X.

3: N – INTEGER. *Input*

On entry: the number of differential equations, n.

Constraint: N ≥ 1.

4: Y(N) – ***real*** array. *Input/Output*

On entry: the initial values of the solution $y_1,y_2,...,y_n$ at x = X.

On exit: the computed values of the solution at the final point x = X.

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

FCN must evaluate the functions f_i (i.e. the derivatives y_i') for given values of their arguments $x,y_1,y_2,...,y_n$.

Its specification is:

```
SUBROUTINE  FCN (X, Y, F)
real           X, Y(n), F(n)
```

where n is the actual value of N in the call of D02EJF.

1: X – ***real.*** *Input*

On entry: the value of the independent variable x.

2: Y(n) – ***real*** array. *Input*

On entry: the value of the variable y_i, for i = 1,2,...,n.

3: F(n) – ***real*** array. *Output*

On exit: the value of f_i, for i = 1,2,...,n.

FCN must be declared as EXTERNAL in the (sub)program from which D02EJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: PEDERV – SUBROUTINE, supplied by the user. *External Procedure*

PEDERV must evaluate the Jacobian of the system (that is, the partial derivatives $\frac{\partial f_i}{\partial y_j}$) for given values of the variables $x,y_1,y_2,...,y_n$.

Its specification is:

```
SUBROUTINE  PEDERV (X, Y, PW)
real           X, Y(n), PW(n, n)
```

where n is the actual value of N in the call of D02EJF.

1: X – ***real.*** *Input*

On entry: the value of the independent variable x.

2: Y(n) – ***real*** array. *Input*

On entry: the value of the variable y_i, for i = 1,2,...,n.

3: PW(n,n) – ***real*** array. *Output*

On exit: the value of $\frac{\partial f_i}{\partial y_j}$, for $i,j = 1,2,\ldots,n$.

If the user does not wish to supply the Jacobian, the actual argument PEDERV **must** be the dummy routine D02EJY. (D02EJY is included in the NAG Fortran Library and so need not be supplied by the user. The name may be implementation dependent: see the Users' Note for your implementation for details.)

PEDERV must be declared as EXTERNAL in the (sub)program from which D02EJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: TOL – ***real***. *Input/Output*

On entry: TOL must be set to a **positive** tolerance for controlling the error in the integration. Hence TOL affects the determination of the position where $g(x,y) = 0.0$, if G is supplied.

D02EJF has been designed so that, for most problems, a reduction in TOL leads to an approximately proportional reduction in the error in the solution. However, the actual relation between TOL and the accuracy achieved cannot be guaranteed. The user is strongly recommended to call D02EJF with more than one value for TOL and to compare the results obtained to estimate their accuracy. In the absence of any prior knowledge, the user might compare the results obtained by calling D02EJF with TOL $= 10^{-p}$ and TOL $= 10^{-p-1}$ if p correct decimal digits are required in the solution.

Constraint: TOL > 0.0.

On exit: normally unchanged. However if the range X to XEND is so short that a small change in TOL is unlikely to make any change in the computed solution, then, on return, TOL has its sign changed.

8: RELABS – CHARACTER*1. *Input*

On entry: the type of error control. At each step in the numerical solution an estimate of the local error, EST, is made. For the current step to be accepted the following condition must be satisfied:

$$\text{EST} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(e_i/(\tau_r \times |y_i| + \tau_a))^2} \leq 1.0$$

where τ_r and τ_a are defined by

RELABS	τ_r	τ_a
'M'	TOL	TOL
'A'	0.0	TOL
'R'	TOL	ε
'D'	TOL	ε

where ε is a small machine dependent number and e_i is an estimate of the local error at y_i, computed internally. If the appropriate condition is not satisfied, the stepsize is reduced and the solution is recomputed on the current step. If the user wishes to measure the error in the computed solution in terms of the number of correct decimal places, then RELABS should be set to 'A' on entry, whereas if the error requirement is in terms of the number of correct significant digits, then RELABS should be set to 'R'. If the user prefers a mixed error test, then RELABS should be set to 'M', otherwise if the user has no preference, RELABS should be set to the default 'D'. Note that in this case 'D' is taken to be 'R'.

Constraint: RELABS = 'A', 'M', 'R' or 'D'.

9: OUTPUT – SUBROUTINE, supplied by the user. *External Procedure*

OUTPUT permits access to intermediate values of the computed solution (for example to print or plot them), at successive user specified points. It is initially called by D02EJF with XSOL = X (the initial value of x). The user must reset XSOL to the next point (between the current XSOL and XEND) where OUTPUT is to be called, and so on at each call to OUTPUT. If, after a call to OUTPUT, the reset point XSOL is beyond XEND, D02EJF will integrate to XEND with no further calls to OUTPUT; if a call to OUTPUT is required at the point XSOL = XEND, then XSOL must be given precisely the value XEND.

Its specification is:

```
SUBROUTINE OUTPUT (XSOL, Y)
real         XSOL, Y(n)
```

where n is the actual value of N in the call of D02EJF.

1: XSOL – ***real.*** *Input/Output*

On entry: the value of the independent variable x.

On exit: the user must set XSOL to the next value of x at which OUTPUT is to be called.

2: Y(n) – ***real*** array. *Input*

On entry: the computed solution at the point XSOL.

If the user does not wish to access intermediate output, the actual argument OUTPUT **must** be the dummy routine D02EJX. (D02EJX is included in the NAG Fortran Library and so need not be supplied by the user. The name may be implementation dependent: see the Users' Note for your implementation for details.)

OUTPUT must be declared as EXTERNAL in the (sub)program from which D02EJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

10: G – ***real*** FUNCTION, supplied by the user. *External Procedure*

G must evaluate the function $g(x,y)$ for specified values x,y. It specifies the function g for which the first position x where $g(x,y) = 0$ is to be found.

Its specification is:

```
real FUNCTION G(X, Y)
real          X, Y(n)
```

where n is the actual value of N in the call of D02EJF.

1: X – ***real.*** *Input*

On entry: the value of the independent variable x.

2: Y(n) – ***real*** array. *Input*

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

If the user does not require the root finding option, the actual argument G **must** be the dummy routine D02EJW. (D02EJW is included in the NAG Fortran Library and so need not be supplied by the user. The name may be implementation dependent: see the Users' Note for your implementation for details.)

G must be declared as EXTERNAL in the (sub)program from which D02EJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

11: W(IW) – ***real*** array. *Workspace*

12: IW – INTEGER. *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D02EJF is called.

Constraint: IW ≥ (12+N)×N + 50.

13: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, TOL ≤ 0.0,
or X = XEND,
or N ≤ 0,
or RELABS ≠ 'M', 'A', 'R' or 'D',
or IW < (12+N)×N + 50.

IFAIL = 2

With the given value of TOL, no further progress can be made across the integration range from the current point x = X. (See Section 5 for a discussion of this error test.) The components Y(1),Y(2),...,Y(n) contain the computed values of the solution at the current point x = X. If the user has supplied G, then no point at which $g(x,y)$ changes sign has been located up to the point x = X.

IFAIL = 3

TOL is too small for D02EJF to take an initial step. X and Y(1),Y(2),...,Y(n) retain their initial values.

IFAIL = 4

XSOL lies behind X in the direction of integration, after the initial call to OUTPUT, if the OUTPUT option was selected.

IFAIL = 5

A value of XSOL returned by OUTPUT lies behind the last value of XSOL in the direction of integration, if the OUTPUT option was selected.

IFAIL = 6

At no point in the range X to XEND did the function $g(x,y)$ change sign, if G was supplied. It is assumed that $g(x,y)$ = 0 has no solution.

IFAIL = 7 (C05AZF)
IFAIL = 8 (D02XKF)
IFAIL = 9 (D02NMF)

A serious error has occurred in an internal call to the specified routine. Check all subroutine calls and array sizes. Seek expert help.

7. Accuracy

The accuracy of the computation of the solution vector Y may be controlled by varying the local error tolerance TOL. In general, a decrease in local error tolerance should lead to an increase in accuracy. Users are advised to choose RELABS = 'R' unless they have a good reason for a different choice. It is particularly appropriate if the solution decays.

If the problem is a root-finding one, then the accuracy of the root determined will depend strongly on $\frac{\partial g}{\partial x}$ and $\frac{\partial g}{\partial y_i}$, for $i = 1,2,...,n$. Large values for these quantities may imply large errors in the root.

8. Further Comments

If more than one root is required, then to determine the second and later roots D02EJF may be called again starting a short distance past the previously determined roots. Alternatively the user may construct his own root finding code using D02NBF (and other routines of the subchapter D02M-D02N), D02XKF and C05AZF.

If it is easy to code, the user should supply the routine PEDERV. However, it is important to be aware that if PEDERV is coded incorrectly, a very inefficient integration may result and possibly even a failure to complete the integration (IFAIL = 2).

9. Example

We illustrate the solution of five different problems. In each case the differential system is the well-known stiff Robertson problem.

$$\begin{aligned} a' &= -0.04a + 10^4bc \\ b' &= \;\;0.04a - 10^4bc - 3\times10^7b^2 \\ c' &= \qquad\qquad\qquad\quad\; 3\times10^7b^2 \end{aligned}$$

with initial conditions $a = 1.0$, $b = c = 0.0$ at $x = 0.0$. We solve each of the following problems with local error tolerances 1.0E−3 and 1.0E−4.

(i) To integrate to $x = 10.0$ producing output at intervals of 2.0 until a point is encountered where $a = 0.9$. The Jacobian is calculated numerically.

(ii) As (i) but with the Jacobian calculated analytically.

(iii) As (i) but with no intermediate output.

(iv) As (i) but with no termination on a root-finding condition.

(v) Integrating the equations as in (i) but with no intermediate output and no root-finding termination condition.

9.1. Program Text

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

```
*     D02EJF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, IW
      PARAMETER        (N=3,IW=(12+N)*N+50)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      real             H, XEND
      INTEGER          K
*     .. Local Scalars ..
      real             TOL, X
      INTEGER          I, IFAIL, J
*     .. Local Arrays ..
      real             W(IW), Y(N)
*     .. External Functions ..
      real             D02EJW, G
      EXTERNAL         D02EJW, G
```

```
*     .. External Subroutines ..
      EXTERNAL         D02EJF, D02EJX, D02EJY, FCN, OUT, PEDERV
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Common blocks ..
      COMMON           XEND, H, K
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02EJF Example Program Results'
      XEND = 10.0e0
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 1: calculating Jacobian internally,'
      WRITE (NOUT,*) ' intermediate output, root-finding'
      DO 20 J = 3, 4
         TOL = 10.0e0**(-J)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) ' Calculation with TOL =', TOL
         X = 0.0e0
         Y(1) = 1.0e0
         Y(2) = 0.0e0
         Y(3) = 0.0e0
         K = 4
         H = (XEND-X)/real(K+1)
         WRITE (NOUT,*) '     X          Y(1)           Y(2)           Y(3)'
         IFAIL = 0
*
         CALL D02EJF(X,XEND,N,Y,FCN,D02EJY,TOL,'Default',OUT,G,W,IW,
     +               IFAIL)
*
         WRITE (NOUT,99998) '  Root of Y(1)-0.9 at', X
         WRITE (NOUT,99997) '  Solution is', (Y(I),I=1,N)
         IF (TOL.LT.0.0e0) WRITE (NOUT,*) '  Range too short for TOL'
   20 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 2: calculating Jacobian by PEDERV,'
      WRITE (NOUT,*) ' intermediate output, root-finding'
      DO 40 J = 3, 4
         TOL = 10.0e0**(-J)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) ' Calculation with TOL =', TOL
         X = 0.0e0
         Y(1) = 1.0e0
         Y(2) = 0.0e0
         Y(3) = 0.0e0
         K = 4
         H = (XEND-X)/real(K+1)
         WRITE (NOUT,*) '     X          Y(1)           Y(2)           Y(3)'
         IFAIL = 0
*
         CALL D02EJF(X,XEND,N,Y,FCN,PEDERV,TOL,'Default',OUT,G,W,IW,
     +               IFAIL)
*
         WRITE (NOUT,99998) '  Root of Y(1)-0.9 at', X
         WRITE (NOUT,99997) '  Solution is', (Y(I),I=1,N)
         IF (TOL.LT.0.0e0) WRITE (NOUT,*) '  Range too short for TOL'
   40 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 3: calculating Jacobian internally,'
      WRITE (NOUT,*) ' no intermediate output, root-finding'
      DO 60 J = 3, 4
         TOL = 10.0e0**(-J)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) ' Calculation with TOL =', TOL
         X = 0.0e0
         Y(1) = 1.0e0
         Y(2) = 0.0e0
         Y(3) = 0.0e0
         IFAIL = 0
*
```

```
         CALL D02EJF(X,XEND,N,Y,FCN,D02EJY,TOL,'Default',D02EJX,G,W,IW,
     +               IFAIL)
*
         WRITE (NOUT,99998) '  Root of Y(1)-0.9 at', X
         WRITE (NOUT,99997) '  Solution is', (Y(I),I=1,N)
         IF (TOL.LT.0.0e0) WRITE (NOUT,*) '  Range too short for TOL'
   60 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 4: calculating Jacobian internally,'
      WRITE (NOUT,*) ' intermediate output, no root-finding'
      DO 80 J = 3, 4
         TOL = 10.0e0**(-J)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) ' Calculation with TOL =', TOL
         X = 0.0e0
         Y(1) = 1.0e0
         Y(2) = 0.0e0
         Y(3) = 0.0e0
         K = 4
         H = (XEND-X)/real(K+1)
         WRITE (NOUT,*) '     X         Y(1)        Y(2)         Y(3)'
         IFAIL = 0
*
         CALL D02EJF(X,XEND,N,Y,FCN,D02EJY,TOL,'Default',OUT,D02EJW,W,
     +               IW,IFAIL)
*
         IF (TOL.LT.0.0e0) WRITE (NOUT,*) '  Range too short for TOL'
   80 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 5: calculating Jacobian internally,'
      WRITE (NOUT,*)
     + ' no intermediate output, no root-finding (integrate to XEND)'
      DO 100 J = 3, 4
         TOL = 10.0e0**(-J)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) ' Calculation with TOL =', TOL
         X = 0.0e0
         Y(1) = 1.0e0
         Y(2) = 0.0e0
         Y(3) = 0.0e0
         WRITE (NOUT,*) '     X         Y(1)        Y(2)         Y(3)'
         WRITE (NOUT,99996) X, (Y(I),I=1,N)
         IFAIL = 0
*
         CALL D02EJF(X,XEND,N,Y,FCN,D02EJY,TOL,'Default',D02EJX,D02EJW,
     +               W,IW,IFAIL)
*
         WRITE (NOUT,99996) X, (Y(I),I=1,N)
         IF (TOL.LT.0.0e0) WRITE (NOUT,*) '  Range too short for TOL'
  100 CONTINUE
      STOP
*
99999 FORMAT (1X,A,e8.1)
99998 FORMAT (1X,A,F7.3)
99997 FORMAT (1X,A,3F13.5)
99996 FORMAT (1X,F8.2,3F13.5)
      END
*
      SUBROUTINE FCN(T,Y,F)
*     .. Parameters ..
      INTEGER         N
      PARAMETER       (N=3)
*     .. Scalar Arguments ..
      real            T
*     .. Array Arguments ..
      real            F(N), Y(N)
```

```
*     .. Executable Statements ..
      F(1) = -0.04e0*Y(1) + 1.0e4*Y(2)*Y(3)
      F(2) = 0.04e0*Y(1) - 1.0e4*Y(2)*Y(3) - 3.0e7*Y(2)*Y(2)
      F(3) = 3.0e7*Y(2)*Y(2)
      RETURN
      END
*
      SUBROUTINE PEDERV(X,Y,PW)
*     .. Parameters ..
      INTEGER           N
      PARAMETER         (N=3)
*     .. Scalar Arguments ..
      real              X
*     .. Array Arguments ..
      real              PW(N,N), Y(N)
*     .. Executable Statements ..
      PW(1,1) = -0.04e0
      PW(1,2) = 1.0e4*Y(3)
      PW(1,3) = 1.0e4*Y(2)
      PW(2,1) = 0.04e0
      PW(2,2) = -1.0e4*Y(3) - 6.0e7*Y(2)
      PW(2,3) = -1.0e4*Y(2)
      PW(3,1) = 0.0e0
      PW(3,2) = 6.0e7*Y(2)
      PW(3,3) = 0.0e0
      RETURN
      END
*
      real FUNCTION G(T,Y)
*     .. Parameters ..
      INTEGER           N
      PARAMETER         (N=3)
*     .. Scalar Arguments ..
      real              T
*     .. Array Arguments ..
      real              Y(N)
*     .. Executable Statements ..
      G = Y(1) - 0.9e0
      RETURN
      END
*
      SUBROUTINE OUT(X,Y)
*     .. Parameters ..
      INTEGER           N
      PARAMETER         (N=3)
      INTEGER           NOUT
      PARAMETER         (NOUT=6)
*     .. Scalar Arguments ..
      real              X
*     .. Array Arguments ..
      real              Y(N)
*     .. Scalars in Common ..
      real              H, XEND
      INTEGER           I
*     .. Local Scalars ..
      INTEGER           J
*     .. Intrinsic Functions ..
      INTRINSIC         real
*     .. Common blocks ..
      COMMON            XEND, H, I
*     .. Executable Statements ..
      WRITE (NOUT,99999) X, (Y(J),J=1,N)
      X = XEND - real(I)*H
      I = I - 1
      RETURN
*
99999 FORMAT (1X,F8.2,3F13.5)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D02EJF Example Program Results

Case 1: calculating Jacobian internally,
 intermediate output, root-finding

 Calculation with TOL = 0.1E-02
     X        Y(1)          Y(2)         Y(3)
   0.00      1.00000       0.00000      0.00000
   2.00      0.94163       0.00003      0.05835
   4.00      0.90551       0.00002      0.09447
  Root of Y(1)-0.9 at  4.377
  Solution is     0.90000      0.00002      0.09998

 Calculation with TOL = 0.1E-03
     X        Y(1)          Y(2)         Y(3)
   0.00      1.00000       0.00000      0.00000
   2.00      0.94161       0.00003      0.05837
   4.00      0.90551       0.00002      0.09446
  Root of Y(1)-0.9 at  4.377
  Solution is     0.90000      0.00002      0.09998


Case 2: calculating Jacobian by PEDERV,
 intermediate output, root-finding

 Calculation with TOL = 0.1E-02
     X        Y(1)          Y(2)         Y(3)
   0.00      1.00000       0.00000      0.00000
   2.00      0.94163       0.00003      0.05835
   4.00      0.90551       0.00002      0.09447
  Root of Y(1)-0.9 at  4.377
  Solution is     0.90000      0.00002      0.09998

 Calculation with TOL = 0.1E-03
     X        Y(1)          Y(2)         Y(3)
   0.00      1.00000       0.00000      0.00000
   2.00      0.94161       0.00003      0.05837
   4.00      0.90551       0.00002      0.09446
  Root of Y(1)-0.9 at  4.377
  Solution is     0.90000      0.00002      0.09998


Case 3: calculating Jacobian internally,
 no intermediate output, root-finding

 Calculation with TOL = 0.1E-02
  Root of Y(1)-0.9 at  4.377
  Solution is     0.90000      0.00002      0.09998

 Calculation with TOL = 0.1E-03
  Root of Y(1)-0.9 at  4.377
  Solution is     0.90000      0.00002      0.09998
```

```
Case 4: calculating Jacobian internally,
 intermediate output, no root-finding

 Calculation with TOL = 0.1E-02
      X          Y(1)          Y(2)          Y(3)
     0.00       1.00000       0.00000       0.00000
     2.00       0.94163       0.00003       0.05835
     4.00       0.90551       0.00002       0.09447
     6.00       0.87929       0.00002       0.12069
     8.00       0.85858       0.00002       0.14141
    10.00       0.84136       0.00002       0.15862

 Calculation with TOL = 0.1E-03
      X          Y(1)          Y(2)          Y(3)
     0.00       1.00000       0.00000       0.00000
     2.00       0.94161       0.00003       0.05837
     4.00       0.90551       0.00002       0.09446
     6.00       0.87926       0.00002       0.12072
     8.00       0.85854       0.00002       0.14145
    10.00       0.84136       0.00002       0.15863


Case 5: calculating Jacobian internally,
 no intermediate output, no root-finding (integrate to XEND)

 Calculation with TOL = 0.1E-02
      X          Y(1)          Y(2)          Y(3)
     0.00       1.00000       0.00000       0.00000
    10.00       0.84136       0.00002       0.15862

 Calculation with TOL = 0.1E-03
      X          Y(1)          Y(2)          Y(3)
     0.00       1.00000       0.00000       0.00000
    10.00       0.84136       0.00002       0.15863
```

D02GAF – NAG Fortran Library Routine Document

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

1. Purpose

D02GAF solves the two-point boundary-value problem with assigned boundary values for a system of ordinary differential equations, using a deferred correction technique and a Newton iteration.

2. Specification

```
 SUBROUTINE D02GAF (U, V, N, A, B, TOL, FCN, MNP, X, Y, NP, W, LW,
1                    IW, LIW, IFAIL)
 INTEGER          N, MNP, NP, LW, IW(LIW), LIW, IFAIL
 real             U(N,2), V(N,2), A, B, TOL, X(MNP), Y(N,MNP),
1                 W(LW)
 EXTERNAL         FCN
```

3. Description

D02GAF solves a two-point boundary-value problem for a system of n differential equations in the interval $[a,b]$. The system is written in the form

$$y_i' = f_i(x,y_1,y_2,...,y_n), \qquad i = 1,2,...,n \qquad (1)$$

and the derivatives are evaluated by a subroutine FCN supplied by the user. Initially, n boundary values of the variables y_i must be specified (assigned), some at a and some at b. The user also supplies estimates of the remaining n boundary values and all the boundary values are used in constructing an initial approximation to the solution. This approximate solution is corrected by a finite-difference technique with deferred correction allied with a Newton iteration to solve the finite-difference equations. The technique used is described fully in Pereyra [1]. The Newton iteration requires a Jacobian matrix $\frac{\partial f_i}{\partial y_j}$ and this is calculated by numerical differentiation using an algorithm described in Curtis, *et al.* [2].

The user supplies an absolute error tolerance and may also supply an initial mesh for the construction of the finite-difference equations (alternatively a default mesh is used). The algorithm constructs a solution on a mesh defined by adding points to the initial mesh. This solution is chosen so that the error is everywhere less than the user's tolerance and so that the error is approximately equidistributed on the final mesh. The solution is returned on this final mesh.

If the solution is required at a few specific points then these should be included in the initial mesh. If on the other hand the solution is required at several specific points then the user should use the interpolation routines provided in the E01 chapter if these points do not themselves form a convenient mesh.

4. References

[1] PEREYRA, V.
PASVA3: An Adaptive Finite-Difference Fortran Program for First Order Nonlinear, Ordinary Boundary Problems.
In: 'Codes for Boundary Value Problems in Ordinary Differential Equations',
B. Childs, M. Scott, J.W. Daniel, E. Denman and P. Nelson. (eds.)
Springer-Verlag, Lecture Notes in Computer Science, 76, 1979.

[2] CURTIS, A.R., POWELL, M.J.D. and REID, J.K.
On The Estimation of Sparse Jacobian Matrices.
J. Inst. Maths. Applics, 13, pp. 117-119. 1974.

5. Parameters

1: U(N,2) – ***real*** array. *Input*

On entry: U$(i,1)$ must be set to the known (assigned) or estimated values of y_i at a and U$(i,2)$ must be set to the known or estimated values of y_i at b, for $i = 1,2,...,n$.

2: V(N,2) – ***real*** array. *Input*

On entry: V(i,j) must be set to 0.0 if U(i,j) is a known (assigned) value and to 1.0 if U(i,j) is an estimated value, $i = 1,2,...,n$; $j = 1,2$.

Constraint: precisely N of the $V(i,j)$ must be set to 0.0, i.e. precisely N of the U(i,j) must be known values, and these must not be all at a or all at b.

3: N – INTEGER. *Input*

On entry: the number of equations.

Constraint: N $\geq$ 2.

4: A – ***real.*** *Input*

On entry: the left-hand boundary point, a.

5: B – ***real.*** *Input*

On entry: the right-hand boundary point, b.

Constraint: B > A.

6: TOL – ***real.*** *Input*

On entry: a positive absolute error tolerance. If

$$a = x_1 < x_2 < ... < x_{\mathrm{NP}} = b$$

is the final mesh, $z_j(x_i)$ is the jth component of the approximate solution at x_i, and $y_j(x)$ is the jth component of the true solution of equation (1) (see Section 3) and the boundary conditions, then, except in extreme cases, it is expected that

$$|z_j(x_i)-y_j(x_i)| \leq \mathrm{TOL}, \qquad i = 1,2,...,\mathrm{NP}; j = 1,2,...,n. \qquad (2)$$

Constraint: TOL > 0.0.

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

FCN must evaluate the functions f_i (i.e. the derivatives y_i') at the general point x.

Its specification is:

```
SUBROUTINE FCN(X, Y, F)
real         X, Y(n), F(n)
```

where n is the actual value of N in the call of D02GAF.

1: X – ***real.*** *Input*

On entry: the value of the argument x.

2: Y(n) – ***real*** array. *Input*

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

3: F(n) – ***real*** array. *Output*

On exit: the values of f_i, for $i = 1,2,...,n$.

FCN must be declared as EXTERNAL in the (sub)program from which D02GAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

8: MNP – INTEGER. *Input*

On entry: the maximum permitted number of mesh-points.

Constraint: MNP ≥ 32.

9: X(MNP) – ***real*** array. *Input/Output*

On entry: if NP ≥ 4 (see NP below), the first NP elements must define an initial mesh. Otherwise the elements of X need not be set.

Constraint: A = X(1) < X(2) < ... < X(NP) = B for NP ≥ 4 (3)

On exit: X(1),X(2),...,X(NP) define the final mesh (with the returned value of NP) satisfying the relation (3).

10: Y(N,MNP) – ***real*** array. *Output*

On exit: the approximate solution $z_j(x_i)$ satisfying (2), on the final mesh, that is

$$Y(j,i) = z_j(x_i), \qquad i = 1,2,\dots,NP; j = 1,2,\dots,n,$$

where NP is the number of points in the final mesh.

The remaining columns of Y are not used.

11: NP – INTEGER. *Input/Output*

On entry: determines whether a default or user-supplied mesh is used. If NP = 0, a default value of 4 for NP and a corresponding equispaced mesh X(1),X(2),...,X(NP) are used. If NP ≥ 4, then the user must define an initial mesh using the array X as described.

Constraint: NP = 0 or 4 ≤ NP ≤ MNP.

On exit: the number of points in the final (returned) mesh.

12: W(LW) – ***real*** array. *Workspace*
13: LW – INTEGER. *Input*

On entry: the length of the array W as declared in the calling (sub)program.

Constraint: LW ≥ MNP×($3N^2$+6N+2) + $4N^2$ + 4N

14: IW(LIW) – INTEGER array. *Workspace*
15: LIW – INTEGER. *Input*

On entry: the length of the array IW as declared in the calling (sub)program.

Constraint: LIW ≥ MNP×(2N+1) + N^2 + 4N + 2.

16: IFAIL – INTEGER. *Input/Output*

For this routine, the normal use of IFAIL is extended to control the printing of error and warning messages as well as specifying hard or soft failure (see Chapter P01 for details).

Before entry, IFAIL must be set to a value with the decimal expansion *cba*, where each of the decimal digits *c*, *b* and *a* must have the value 0 or 1.

a = 0 specifies hard failure, otherwise soft failure;

b = 0 suppresses error messages, otherwise error messages will be printed (see Section 6);

c = 0 suppresses warning messages, otherwise warning messages will be printed (see Section 6).

The recommended value for inexperienced users is 110 (i.e. hard failure with all messages printed).

Unless the routine detects an error (see Section 6), IFAIL contains 0 on exit.

6. Error Indicators and Warnings

Errors detected by the routine:

For each error, an explanatory error message is output on the current error message unit (as defined by X04AAF), unless suppressed by the value of IFAIL on entry.

IFAIL = 1

One or more of the parameters N, TOL, NP, MNP, LW or LIW has been incorrectly set, or B ≤ A, or the condition (3) on X is not satisfied, or the number of known boundary values (specified by V) is not N.

IFAIL = 2

The Newton iteration has failed to converge. This could be due to there being too few points in the initial mesh or to the initial approximate solution being too inaccurate. If this latter reason is suspected the user should use subroutine D02RAF instead. If the warning 'Jacobian matrix is singular' is printed this could be due to specifying zero estimated boundary values and these should be varied. This warning could also be printed in the unlikely event of the Jacobian matrix being calculated inaccurately. If the user cannot make changes to prevent the warning then subroutine D02RAF should be used.

IFAIL = 3

The Newton iteration has reached roundoff level. It could be, however, that the answer returned is satisfactory. This error might occur if too much accuracy is requested.

IFAIL = 4

A finer mesh is required for the accuracy requested; that is MNP is not large enough.

IFAIL = 5

A serious error has occurred in a call to D02GAF. Check all array subscripts and subroutine parameter lists in calls to D02GAF. Seek expert help.

7. Accuracy

The solution returned by the routine will be accurate to the user's tolerance as defined by the relation (2) except in extreme circumstances. If too many points are specified in the initial mesh, the solution may be more accurate than requested and the error may not be approximately equidistributed.

8. Further Comments

The time taken by the routine depends on the difficulty of the problem, the number of mesh points used (and the number of different meshes used), the number of Newton iterations and the number of deferred corrections.

The user is strongly recommended to set IFAIL to obtain self-explanatory error messages, and also monitoring information about the course of the computation. The user may select the channel numbers on which this output is to appear by calls of X04AAF (for error messages) or X04ABF (for monitoring information) – see Section 9 for an example. Otherwise the default channel numbers will be used, as specified in the implementation document.

A common cause of convergence problems in the Newton iteration is the user specifying too few points in the initial mesh. Although the routine adds points to the mesh to improve accuracy it is unable to do so until the solution on the initial mesh has been calculated in the Newton iteration.

If the user specifies zero known **and** estimated boundary values, the routine constructs a zero initial approximation and in many cases the Jacobian is singular when evaluated for this approximation, leading to the breakdown of the Newton iteration.

The user may be unable to provide a sufficiently good choice of initial mesh and estimated boundary values, and hence the Newton iteration may never converge. In this case the continuation facility provided in D02RAF is recommended.

In the case where the user wishes to solve a sequence of similar problems, the final mesh from solving one case is strongly recommended as the initial mesh for the next.

9. Example

We solve the differential equation

$$y''' = -yy'' - \beta(1-y'^2)$$

with boundary conditions

$$y(0) = y'(0) = 0, \quad y'(10) = 1$$

for $\beta = 0.0$ and $\beta = 0.2$ to an accuracy specified by TOL = 1.0E−3. We solve first the simpler problem with $\beta = 0.0$ using an equi-spaced mesh of 26 points and then we solve the problem with $\beta = 0.2$ using the final mesh from the first problem.

Note the call to X04ABF prior to the call to D02GAF.

9.1. Program Text

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

```
*     D02GAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, MNP, LW, LIW
      PARAMETER        (N=3,MNP=40,LW=MNP*(3*N*N+6*N+2)+4*N*N+4*N,
     +                 LIW=MNP*(2*N+1)+N*N+4*N+2)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      real             BETA
*     .. Local Scalars ..
      real             A, B, TOL
      INTEGER          I, IFAIL, J, K, NP
*     .. Local Arrays ..
      real             U(N,2), V(N,2), W(LW), X(MNP), Y(N,MNP)
      INTEGER          IW(LIW)
*     .. External Subroutines ..
      EXTERNAL         D02GAF, FCN, X04ABF
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Common blocks ..
      COMMON           BETA
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02GAF Example Program Results'
      TOL = 1.0e-3
      NP = 26
      A = 0.0e0
      B = 10.0e0
      CALL X04ABF(1,NOUT)
      BETA = 0.0e0
      DO 40 I = 1, N
         DO 20 J = 1, 2
            U(I,J) = 0.0e0
            V(I,J) = 0.0e0
   20    CONTINUE
   40 CONTINUE
      V(3,1) = 1.0e0
      V(1,2) = 1.0e0
      V(3,2) = 1.0e0
      U(2,2) = 1.0e0
      U(1,2) = B
      X(1) = A
      DO 60 I = 2, NP - 1
         X(I) = (real(NP-I)*A+real(I-1)*B)/real(NP-1)
   60 CONTINUE
```

```
         X(NP) = B
         DO 80 K = 1, 2
            WRITE (NOUT,*)
            WRITE (NOUT,99999) 'Problem with BETA = ', BETA
*           * Set IFAIL to 111 to obtain monitoring information *
            IFAIL = 11
*
            CALL D02GAF(U,V,N,A,B,TOL,FCN,MNP,X,Y,NP,W,LW,IW,LIW,IFAIL)
*
            IF (IFAIL.EQ.0 .OR. IFAIL.EQ.3) THEN
               WRITE (NOUT,*)
               IF (IFAIL.EQ.3) WRITE (NOUT,99996) ' IFAIL = ', IFAIL
               WRITE (NOUT,99998) 'Solution on final mesh of ', NP,
     +           ' points'
               WRITE (NOUT,*)
     +           '        X(I)          Y1(I)          Y2(I)          Y3(I)'
               WRITE (NOUT,99997) (X(I),(Y(J,I),J=1,N),I=1,NP)
               BETA = BETA + 0.2e0
            ELSE
               STOP
            END IF
      80 CONTINUE
         STOP
*
   99999 FORMAT (1X,A,F7.2)
   99998 FORMAT (1X,A,I2,A)
   99997 FORMAT (1X,F11.3,3F13.4)
   99996 FORMAT (1X,A,I3)
         END
*
         SUBROUTINE FCN(X,Y,F)
*        .. Parameters ..
         INTEGER          N
         PARAMETER        (N=3)
*        .. Scalar Arguments ..
         real             X
*        .. Array Arguments ..
         real             F(N), Y(N)
*        .. Scalars in Common ..
         real             BETA
*        .. Common blocks ..
         COMMON           BETA
*        .. Executable Statements ..
         F(1) = Y(2)
         F(2) = Y(3)
         F(3) = -Y(1)*Y(3) - BETA*(1.0e0-Y(2)*Y(2))
         RETURN
         END
```

9.2. Program Data

None.

9.3. Program Results

```
 D02GAF Example Program Results

 Problem with BETA =    0.00

 Solution on final mesh of 26 points
        X(I)          Y1(I)          Y2(I)          Y3(I)
       0.000         0.0000         0.0000         0.4695
       0.400         0.0375         0.1876         0.4673
       0.800         0.1497         0.3719         0.4511
       1.200         0.3336         0.5450         0.4104
       1.600         0.5828         0.6963         0.3424
       2.000         0.8864         0.8163         0.2558
       2.400         1.2309         0.9009         0.1678
       2.800         1.6026         0.9529         0.0953
       3.200         1.9900         0.9805         0.0464
       3.600       2.3851       0.9930       0.0193
       4.000       2.7834       0.9978       0.0069
       4.400       3.1829       0.9994       0.0021
       4.800       3.5828       0.9999       0.0006
       5.200       3.9828       1.0000       0.0001
       5.600       4.3828       1.0000       0.0000
       6.000       4.7828       1.0000       0.0000
       6.400       5.1828       1.0000       0.0000
       6.800       5.5828       1.0000       0.0000
       7.200       5.9828       1.0000       0.0000
       7.600       6.3828       1.0000       0.0000
       8.000       6.7828       1.0000       0.0000
       8.400       7.1828       1.0000       0.0000
       8.800       7.5828       1.0000       0.0000
       9.200       7.9828       1.0000       0.0000
       9.600       8.3828       1.0000       0.0000
      10.000       8.7828       1.0000       0.0000

 Problem with BETA =    0.20

 Solution on final mesh of 26 points
        X(I)        Y1(I)        Y2(I)        Y3(I)
       0.000       0.0000       0.0000       0.6865
       0.400       0.0528       0.2584       0.6040
       0.800       0.2020       0.4814       0.5091
       1.200       0.4324       0.6636       0.4001
       1.600       0.7268       0.8007       0.2860
       2.000       1.0670       0.8939       0.1821
       2.400       1.4368       0.9498       0.1017
       2.800       1.8233       0.9791       0.0492
       3.200       2.2180       0.9924       0.0206
       3.600       2.6162       0.9976       0.0074
       4.000       3.0157       0.9993       0.0023
       4.400       3.4156       0.9998       0.0006
       4.800       3.8155       1.0000       0.0001
       5.200       4.2155       1.0000       0.0000
       5.600       4.6155       1.0000       0.0000
       6.000       5.0155       1.0000       0.0000
       6.400       5.4155       1.0000       0.0000
       6.800       5.8155       1.0000       0.0000
       7.200       6.2155       1.0000       0.0000
       7.600       6.6155       1.0000       0.0000
       8.000       7.0155       1.0000       0.0000
       8.400       7.4155       1.0000       0.0000
       8.800       7.8155       1.0000       0.0000
       9.200       8.2155       1.0000       0.0000
       9.600       8.6155       1.0000       0.0000
      10.000       9.0155       1.0000       0.0000
```

With IFAIL set to 111 in the example program, monitoring information similar to the following is printed when BETA = 0.0:

```
 D02GAF MONITORING INFORMATION

  MONITORING NEWTON ITERATION
   CORRECTION NUMBER     0   RESIDUAL SHOULD BE .LE.   1.60E-02
    ITERATION NUMBER     0   RESIDUAL =  1.17E+00
    ITERATION NUMBER     1   RESIDUAL =  2.04E-01
    ITERATION NUMBER     2   RESIDUAL =  2.45E-02
    ITERATION NUMBER     3   RESIDUAL =  7.57E-04
  NUMBER OF POINTS IN CURRENT MESH =   26

  CORRECTION NUMBER    0   ESTIMATED MAXIMUM ERROR =   1.68E-02
  ESTIMATED ERROR BY COMPONENTS
    1.68E-02  5.45E-03  2.60E-03
```

```
MONITORING NEWTON ITERATION
 CORRECTION NUMBER    1   RESIDUAL SHOULD BE .LE.  1.68E-05
  ITERATION NUMBER    0   RESIDUAL =  4.00E-03
  ITERATION NUMBER    1   RESIDUAL =  2.18E-04
  ITERATION NUMBER    2   RESIDUAL =  9.40E-06
NUMBER OF POINTS IN CURRENT MESH =   26

CORRECTION NUMBER    1   ESTIMATED MAXIMUM ERROR =  5.96E-04
ESTIMATED ERROR BY COMPONENTS
  5.96E-04  3.11E-04  2.22E-04
```

D02GBF – NAG Fortran Library Routine Document

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

1. Purpose

D02GBF solves a general linear two-point boundary value problem for a system of ordinary differential equations using a deferred correction technique.

2. Specification

```
      SUBROUTINE D02GBF (A, B, N, TOL, FCNF, FCNG, C, D, GAM, MNP, X, Y,
     1                   NP, W, LW, IW, LIW, IFAIL)
      INTEGER          N, MNP, NP, LW, IW(LIW), LIW, IFAIL
      real             A, B, TOL, C(N,N), D(N,N), GAM(N), X(MNP),
     1                 Y(N,MNP), W(LW)
      EXTERNAL         FCNF, FCNG
```

3. Description

D02GBF solves the linear two-point boundary value problem for a system of n ordinary differential equations in the interval $[a,b]$. The system is written in the form

$$y' = F(x)y + g(x) \qquad (1)$$

and the boundary conditions are written in the form

$$Cy(a) + Dy(b) = \gamma \qquad (2)$$

Here $F(x)$, C and D are n by n matrices, and $g(x)$ and γ are n-component vectors. The approximate solution to (1) and (2) is found using a finite-difference method with deferred correction. The algorithm is a specialisation of that used in subroutine D02RAF which solves a nonlinear version of (1) and (2). The nonlinear version of the algorithm is described fully in Pereyra [1].

The user supplies an absolute error tolerance and may also supply an initial mesh for the construction of the finite-difference equations (alternatively a default mesh is used). The algorithm constructs a solution on a mesh defined by adding points to the initial mesh. This solution is chosen so that the error is everywhere less than the user's tolerance and so that the error is approximately equidistributed on the final mesh. The solution is returned on this final mesh.

If the solution is required at a few specific points then these should be included in the initial mesh. If, on the other hand, the solution is required at several specific points, then the user should use the interpolation routines provided in the E01 chapter if these points do not themselves form a convenient mesh.

4. References

[1] PEREYRA, V.
PASVA3: An Adaptive Finite-Difference Fortran Program for First Order
Nonlinear, Ordinary Boundary Problems.
In: 'Codes for Boundary Value Problems in Ordinary Differential Equations',
B. Childs, M. Scott, J.W. Daniel, E. Denman and P. Nelson, P. (eds.)
Springer-Verlag, Lecture Notes in Computer Science, 76, 1979.

5. Parameters

1: A – ***real***. *Input*

On entry: the left-hand boundary point, a.

2: B – ***real.*** *Input*

On entry: the right-hand boundary point, b.

Constraint: B > A.

3: N – INTEGER. *Input*

On entry: the number of equations; that is n is the order of system (1).

Constraint: N ≥ 2.

4: TOL – ***real.*** *Input*

On entry: a positive absolute error tolerance. If

$$a = x_1 < x_2 < \ldots < x_{\mathrm{NP}} = b$$

is the final mesh, $z(x)$ is the approximate solution from D02GBF and $y(x)$ is the true solution of equations (1) and (2) then, except in extreme cases, it is expected that

$$\|z-y\| \le \mathrm{TOL} \qquad (3)$$

where

$$\|u\| = \max_{1\le i\le N}\max_{1\le j\le \mathrm{NP}} |u_i(x_j)|.$$

Constraint: TOL > 0.0.

5: FCNF – SUBROUTINE, supplied by the user. *External Procedure*

FCNF must evaluate the matrix $F(x)$ in (1) at a general point x.

Its specification is:

```
SUBROUTINE FCNF (X, F)
real           X, F(n,n)
```

where n is the actual value of N in the call of D02GBF.

1: X – ***real.*** *Input*

On entry: the value of the independent variable x.

2: F(n,n) – ***real*** array. *Output*

On exit: the (i,j)th element of the matrix $F(x)$, for $i,j = 1,2,\ldots,n$. (See Section 9 for an example.)

FCNF must be declared as EXTERNAL in the (sub)program from which D02GBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: FCNG – SUBROUTINE, supplied by the user. *External Procedure*

FCNG must evaluate the vector $g(x)$ in (1) at a general point x.

Its specification is:

```
SUBROUTINE FCNG(X, G)
real           X, G(n)
```

where n is the actual value of N in the call of D02GBF.

1: X – ***real.*** *Input*

On entry: the value of the independent variable x.

2: G(n) – ***real*** array. *Output*

On exit: the ith element of the vector $g(x)$, for $i = 1,2,\ldots,n$. (See Section 9 for an example.)

FCNG must be declared as EXTERNAL in the (sub)program from which D02GBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: C(N,N) – ***real*** array. *Input/Output*

8: D(N,N) – ***real*** array. *Input/Output*

9: GAM(N) – ***real*** array. *Input/Output*

On entry: the arrays C and D must be set to the matrices C and D in (2). GAM must be set to the vector γ in (2).

On exit: the rows of C and D and the components of GAM are re-ordered so that the boundary conditions are in the order:

(i) conditions on $y(a)$ only;

(ii) condition involving $y(a)$ and $y(b)$; and

(iii) conditions on $y(b)$ only.

The routine will be slightly more efficient if the arrays C, D and GAM are ordered in this way before entry, and in this event they will be unchanged on exit.

Note that the problems (1) and (2) must be of boundary value type, that is neither C nor D may be identically zero. Note also that the rank of the matrix $[C,D]$ must be n for the problem to be properly posed. Any violation of these conditions will lead to an error exit.

10: MNP – INTEGER. *Input*

On entry: the maximum permitted number of mesh points.

Constraint: MNP $\geq$ 32.

11: X(MNP) – ***real*** array. *Input/Output*

On entry: if NP $\geq$ 4 (see NP below), the first NP elements must define an initial mesh. Otherwise the elements of x need not be set.

Constraint: A = X(1) < X(2) < ... < X(NP) = B, for NP $\geq$ 4. (4)

On exit: X(1),X(2),...,X(NP) define the final mesh (with the returned value of NP) satisfying the relation (4).

12: Y(N,MNP) – ***real*** array. *Output*

On exit: the approximate solution $z(x)$ satisfying (3), on the final mesh, that is

$$Y(j,i) = z_j(x_i), \qquad i = 1,2,...,NP; j = 1,2,...,n$$

where NP is the number of points in the final mesh.

The remaining columns of Y are not used.

13: NP – INTEGER. *Input/Output*

On entry: determines whether a default mesh or user-supplied mesh is used. If NP = 0, a default value of 4 for NP and a corresponding equispaced mesh X(1),X(2),...,X(NP) are used. If NP $\geq$ 4, then the user must define an initial mesh X as in (4) above.

On exit: the number of points in the final (returned) mesh.

14: W(LW) – ***real*** array. *Workspace*

15: LW – INTEGER. *Input*

On entry: the length of the array W,

Constraint: LW $\geq$ MNP×(3N^2+5N+2) + 3N^2 + 5N.

16: IW(LIW) – INTEGER array. *Workspace*

17: LIW – INTEGER. *Input*

On entry: the length of the array IW.

Constraint: LIW $\geq$ MNP×(2N+1) + N .

18: IFAIL – INTEGER. *Input/Output*

For this routine, the normal use of IFAIL is extended to control the printing of error and warning messages as well as specifying hard or soft failure (see Chapter P01 for details).

Before entry, IFAIL must be set to a value with the decimal expansion *cba*, where each of the decimal digits *c*, *b* and *a* must have the value 0 or 1.

$a = 0$ specifies hard failure, otherwise soft failure;

$b = 0$ suppresses error messages, otherwise error messages will be printed (see Section 6);

$c = 0$ suppresses warning messages, otherwise warning messages will be printed (see Section 6).

The recommended value for inexperienced users is 110 (i.e. hard failure with all messages printed).

Unless the routine detects an error (see Section 6), IFAIL contains 0 on exit.

6. Error Indicators and Warnings

Errors detected by the routine:

For each error, an explanatory error message is output on the current error message unit (as defined by X04AAF), unless suppressed by the value of IFAIL on entry.

IFAIL = 1

One or more of the parameters N, TOL, NP, MNP, LW or LIW is incorrectly set, B ≤ A or the condition (4) on X is not satisfied.

IFAIL = 2

There are three possible reasons for this error exit to be taken:

(i) one of the matrices C or D is identically zero (that is the problem is of initial value and not boundary value type). In this case, IW(1) = 0 on exit;

(ii) a row of C and the corresponding row of D are identically zero (that is the boundary conditions are rank deficient). In this case, on exit IW(1) contains the index of the first such row encountered; and

(iii) more than n of the columns of the n by $2n$ matrix $[C,D]$ are identically zero (that is the boundary conditions are rank deficient). In this case, on exit IW(1) contains minus the number of non-identically zero columns.

IFAIL = 3

The routine has failed to find a solution to the specified accuracy. There are a variety of possible reasons including:

(i) the boundary conditions are rank deficient, which may be indicated by the message that the Jacobian is singular. However this is an unlikely explanation for the error exit as all rank deficient boundary conditions should lead instead to error exits with either IFAIL = 2 or IFAIL = 5; see also (iv) below;

(ii) not enough mesh points are permitted in order to attain the required accuracy. This is indicated by NP = MNP on return from a call to D02GBF. This difficulty may be aggravated by a poor initial choice of mesh points;

(iii) the accuracy requested cannot be attained on the computer being used; and

(iv) an unlikely combination of values of $F(x)$ has led to a singular Jacobian. The error should not persist if more mesh points are allowed.

IFAIL = 4

A serious error has occurred in a call to D02GBF. Check all array subscripts and subroutine parameter lists in calls to D02GBF. Seek expert help.

IFAIL = 5

There are two possible reasons for this error exit which occurs when checking the rank of the boundary conditions by reduction to a row echelon form:

(i) at least one row of the n by $2n$ matrix $[C,D]$ is a linear combination of the other rows and hence the boundary conditions are rank deficient. The index of the first such row encountered is given by IW(1) on exit; and

(ii) as (i) but the rank deficiency implied by this error exit has only been determined up to a numerical tolerance. Minus the index of the first such row encountered is given by IW(1) on exit.

In case (ii) above there is some doubt as to the rank deficiency of the boundary conditions. However even if the boundary conditions are not rank deficient they are not posed in a suitable form for use with this routine.

For example, if

$$C = \begin{pmatrix} 1 & 0 \\ 1 & \varepsilon \end{pmatrix}, \quad D = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}, \quad \gamma = \begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix}$$

and ε is small enough, this error exit is likely to be taken. A better form for the boundary conditions in this case would be

$$C = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad D = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \quad \gamma = \begin{pmatrix} \gamma_1 \\ \varepsilon^{-1}(\gamma_2-\gamma_1) \end{pmatrix}$$

7. Accuracy

The solution returned by the routine will be accurate to the user's tolerance as defined by the relation (3) except in extreme circumstances. If too many points are specified in the initial mesh, the solution may be more accurate than requested and the error may not be approximately equidistributed.

8. Further Comments

The time taken by the routine depends on the difficulty of the problem, the number of mesh points (and meshes) used and the number of deferred corrections.

The user is strongly recommended to set IFAIL to obtain self-explanatory error messages, and also monitoring information about the course of the computation. The user may select the channel numbers on which this output is to appear by calls of X04AAF (for error messages) or X04ABF (for monitoring information) – see Section 9 for an example. Otherwise the default channel numbers will be used, as specified in the implementation document.

In the case where the user wishes to solve a sequence of similar problems, the use of the final mesh from one case is strongly recommended as the initial mesh for the next.

9. Example

We solve the problem (written as a first order system)

$$\varepsilon y'' + y' = 0$$

with boundary conditions

$$y(0) = 0, \quad y(1) = 1$$

for the cases $\varepsilon = 10^{-1}$ and $\varepsilon = 10^{-2}$ using the default initial mesh in the first case, and the final mesh of the first case as initial mesh for the second (more difficult) case. We give the solution and the error at each mesh point to illustrate the accuracy of the method given the accuracy request TOL = 1.0E–3.

Note the call to X04ABF prior to the call to D02GBF.

9.1. Program Text

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

```
*     D02GBF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, MNP, LW, LIW
      PARAMETER        (N=2,MNP=70,LW=MNP*(3*N*N+5*N+2)+3*N*N+5*N,
     +                 LIW=MNP*(2*N+1)+N)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      real             EPS
*     .. Local Scalars ..
      real             A, B, TOL
      INTEGER          I, IFAIL, J, NP
*     .. Local Arrays ..
      real             C(N,N), D(N,N), GAM(N), W(LW), X(MNP), Y(N,MNP)
      INTEGER          IW(LIW)
*     .. External Subroutines ..
      EXTERNAL         D02GBF, FCNF, FCNG, X04ABF
*     .. Common blocks ..
      COMMON           EPS
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02GBF Example Program Results'
      TOL = 1.0e-3
      NP = 0
      A = 0.0e0
      B = 1.0e0
      CALL X04ABF(1,NOUT)
      DO 40 I = 1, N
         GAM(I) = 0.0e0
         DO 20 J = 1, N
            C(I,J) = 0.0e0
            D(I,J) = 0.0e0
   20    CONTINUE
   40 CONTINUE
      C(1,1) = 1.0e0
      D(2,1) = 1.0e0
      GAM(2) = 1.0e0
      DO 60 I = 1, 2
         EPS = 10.0e0**(-I)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Problem with epsilon = ', EPS
*        * Set IFAIL to 111 to obtain monitoring information *
         IFAIL = 11
*
         CALL D02GBF(A,B,N,TOL,FCNF,FCNG,C,D,GAM,MNP,X,Y,NP,W,LW,IW,LIW,
     +               IFAIL)
*
         WRITE (NOUT,*)
         IF (IFAIL.EQ.0) THEN
            WRITE (NOUT,99998) 'Approximate solution on final mesh of ',
     +        NP, ' points'
            WRITE (NOUT,*) '   X(I)     Y(1,I)'
            WRITE (NOUT,99997) (X(J),Y(1,J),J=1,NP)
         ELSE
            WRITE (NOUT,99996) ' IFAIL = ', IFAIL
            STOP
         END IF
   60 CONTINUE
   80 STOP
*
```

```
99999 FORMAT (1X,A,e10.2)
99998 FORMAT (1X,A,I2,A)
99997 FORMAT (1X,2F11.4)
99996 FORMAT (1X,A,I3)
      END
*
      SUBROUTINE FCNF(X,F)
*     .. Parameters ..
      INTEGER          N
      PARAMETER        (N=2)
*     .. Scalar Arguments ..
      real             X
*     .. Array Arguments ..
      real             F(N,N)
*     .. Scalars in Common ..
      real             EPS
*     .. Common blocks ..
      COMMON           EPS
*     .. Executable Statements ..
      F(1,1) = 0.0e0
      F(1,2) = 1
      F(2,1) = 0.0e0
      F(2,2) = -1.0e0/EPS
      RETURN
      END
*
      SUBROUTINE FCNG(X,G)
*     .. Parameters ..
      INTEGER          N
      PARAMETER        (N=2)
*     .. Scalar Arguments ..
      real             X
*     .. Array Arguments ..
      real             G(N)
*     .. Executable Statements ..
      G(1) = 0.0e0
      G(2) = 0.0e0
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D02GBF Example Program Results

 Problem with epsilon =   0.10E+00

 Approximate solution on final mesh of 15 points
       X(I)      Y(1,I)
      0.0000     0.0000
      0.0278     0.2425
      0.0556     0.4263
      0.1111     0.6708
      0.1667     0.8112
      0.2222     0.8917
      0.2778     0.9379
      0.3333     0.9644
      0.4444     0.9883
      0.5556     0.9962
      0.6667     0.9988
      0.7500     0.9995
      0.8333     0.9998
      0.9167     0.9999
      1.0000     1.0000
 Problem with epsilon =   0.10E-01

 Approximate solution on final mesh of 49 points
      X(I)      Y(1,I)
     0.0000     0.0000
     0.0009     0.0884
     0.0019     0.1690
     0.0028     0.2425
     0.0037     0.3095
     0.0046     0.3706
     0.0056     0.4262
     0.0065     0.4770
     0.0074     0.5232
     0.0083     0.5654
     0.0093     0.6038
     0.0111     0.6708
     0.0130     0.7265
     0.0148     0.7727
     0.0167     0.8111
     0.0185     0.8431
     0.0204     0.8696
     0.0222     0.8916
     0.0241     0.9100
     0.0259     0.9252
     0.0278     0.9378
     0.0306     0.9529
     0.0333     0.9643
     0.0361     0.9730
     0.0389     0.9795
     0.0417     0.9845
     0.0444     0.9883
     0.0472     0.9911
     0.0500     0.9933
     0.0528     0.9949
     0.0556     0.9961
     0.0648     0.9985
     0.0741     0.9994
     0.0833     0.9998
     0.0926     0.9999
     0.1019     1.0000
     0.1111     1.0000
     0.1389     1.0000
     0.1667     1.0000
     0.2222     1.0000
     0.2778     1.0000
     0.3333     1.0000
     0.4444     1.0000
     0.5556     1.0000
     0.6667     1.0000
     0.7500     1.0000
     0.8333     1.0000
     0.9167     1.0000
     1.0000     1.0000
```

With IFAIL set to 111 in the example program, monitoring information similar to the following is printed when $\varepsilon = 10^{-1}$:

```
 D02GBF MONITORING INFORMATION
  NUMBER OF POINTS IN CURRENT MESH =   15

  CORRECTION NUMBER    0   ESTIMATED MAXIMUM ERROR =  6.59E-02
  ESTIMATED ERROR BY COMPONENTS
    6.57E-03  6.59E-02
  NUMBER OF POINTS IN CURRENT MESH =   15

  CORRECTION NUMBER    1   ESTIMATED MAXIMUM ERROR =  3.60E-03
  ESTIMATED ERROR BY COMPONENTS
    3.61E-04  3.60E-03
  NUMBER OF POINTS IN CURRENT MESH =   15

  CORRECTION NUMBER    2   ESTIMATED MAXIMUM ERROR =  4.36E-04
  ESTIMATED ERROR BY COMPONENTS
    4.45E-05  4.36E-04
```

D02HAF – NAG Fortran Library Routine Document

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

1 Purpose

D02HAF solves the two-point boundary-value problem for a system of ordinary differential equations, using a Runge–Kutta–Merson method and a Newton iteration in a shooting and matching technique.

2 Specification

```
SUBROUTINE D02HAF(U, V, N, A, B, TOL, FCN, SOLN, M1, W, IW, IFAIL)
INTEGER         N, M1, IW, IFAIL
real            U(N,2), V(N,2), A, B, TOL, SOLN(N,M1), W(N,IW)
EXTERNAL        FCN
```

3 Description

D02HAF solves the two-point boundary-value problem for a system of n ordinary differential equations in the range a, b. The system is written in the form:

$$y_i' = f_i(x, y_1, y_2, \ldots, y_n), \qquad i = 1, 2, \ldots, n \tag{1}$$

and the derivatives f_i are evaluated by a subroutine FCN supplied by the user. Initially, n boundary values of the variables y_i must be specified, some of which will be specified at a and some at b. The user must supply estimates of the remaining n boundary values (called parameters below), and the subroutine corrects them by a form of Newton iteration. It also calculates the complete solution on an equispaced mesh if required.

Starting from the known and estimated values of y_i at a, the subroutine integrates the equations from a to b (using a Runge–Kutta–Merson method). The differences between the values of y_i at b from integration and those specified initially should be zero for the true solution. (These differences are called residuals below.) The subroutine uses a generalized Newton method to reduce the residuals to zero, by calculating corrections to the estimated boundary values. This process is repeated iteratively until convergence is obtained, or until the routine can no longer reduce the residuals. See Hall and Watt [1] for a simple discussion of shooting and matching techniques.

4 References

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

5 Parameters

1: U(N,2) — ***real*** array *Input/Output*

On entry: U(i,1) must be set to the known or estimated value of y_i at a, and U(i,2) must be set to the known or estimated value of y_i at b, for $i = 1, 2, \ldots, n$.

On exit: the known values unaltered, and corrected values of the estimates, unless an error has occurred. If an error has occurred, U contains the known values and the latest values of the estimates.

2: V(N,2) — ***real*** array *Input*

On entry: V(i,j) must be set to 0.0 if U(i,j) is a known value, and 1.0 if U(i,j) is an estimated value to be corrected.

Constraint: precisely n of the V(i,j) must be set to 0.0, i.e., precisely n of the U(i,j) must be known values, and these must not be all at a or all at b.

3: N — INTEGER *Input*

On entry: the number of equations, n.

Constraint: N $\geq$ 2

4: A — ***real*** *Input*

On entry: the initial point of the interval of integration, a.

5: B — ***real*** *Input*

On entry: the final point of the interval of integration, b.

6: TOL — ***real*** *Input*

On entry: TOL must be set to a small quantity suitable for

(1) testing the local error in y_i during integration,
(2) testing for the convergence of y_i at b,
(3) calculating the perturbation in estimated boundary values for y_i, which are used to obtain the approximate derivatives of the residuals for use in the Newton iteration.

The user is advised to check his results by varying TOL.

Constraint: TOL > 0.0.

7: FCN — SUBROUTINE, supplied by the user. *External Procedure*

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

Its specification is:

```
SUBROUTINE FCN(X, Y, F)
real            X, Y(n), F(n)
```

where n is the actual value of N on the call of D02HAF.

1: X — ***real*** *Input*

On entry: the value of the argument, x.

2: Y(n) — ***real*** array *Input*

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

3: F(n) — ***real*** array *Output*

On exit: the value of f_i, x, for $i = 1,2,\ldots,n$.

FCN must be declared as EXTERNAL in the (sub)program from which D02HAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

8: SOLN(N,M1) — ***real*** array *Output*

On exit: the solution when M1 > 1 (see below).

9: M1 — INTEGER *Input*

On entry: a value which controls output:

M1 = 1

the final solution is not evaluated.

M1 > 1

the final values of y_i at interval $(b-a)/(\mathrm{M1}-1)$ are calculated and stored in the array SOLN by columns, starting with values y_i at a stored in SOLN(i,1), for $i = 1,2,\ldots,n$.

Constraint: M1 $\geq$ 1.

10: W(N,IW) — ***real*** array *Output*

On exit: if IFAIL = 2, 3, 4 or 5, W(i,1) contains the solution at the point where the integration fails, for $i = 1, 2, \ldots, n$, and the point of failure is returned in W(1,2).

11: IW — INTEGER *Input*

On entry: the second dimension of W.

Constraint: IW $\geq$ 3N + 17 + max(11,N).

12: IFAIL — INTEGER *Input/Output*

For this routine, the normal use of IFAIL is extended to control the printing of error and warning messages as well as specifying hard or soft failure (see Chapter P01).

Before entry, IFAIL must be set to a value with the decimal expansion cba, where each of the decimal digits c, b and a must have a value of 0 or 1.

$a = 0$ specifies hard failure, otherwise soft failure;
$b = 0$ suppresses error messages, otherwise error messages will be printed (see Section 6);
$c = 0$ suppresses warning messages, otherwise warning messages will be printed (see Section 6).

The recommended value for inexperienced users is 110 (i.e., hard failure with all messages printed).

Unless the routine detects an error (see Section 6), IFAIL contains 0 on exit.

6 Error Indicators and Warnings

For each error, an explanatory error message is output on the current error message unit (as defined by X04AAF), unless suppressed by the value of IFAIL on entry.

Errors detected by the routine:

IFAIL = 1

One or more of the parameters V, N, M1, IW, or TOL is incorrectly set.

IFAIL = 2

The step length for the integration is too short whilst calculating the residual (see Section 8).

IFAIL = 3

No initial step length could be chosen for the integration whilst calculating the residual.

Note: IFAIL = 2 or 3 can occur due to choosing too small a value for TOL or due to choosing the wrong direction of integration. Try varying TOL and interchanging a and b. These error exits can also occur for very poor initial estimates of the unknown initial values and, in extreme cases, because this routine cannot be used to solve the problem posed.

IFAIL = 4

As for IFAIL = 2 but the error occurred when calculating the Jacobian of the derivatives of the residuals with respect to the parameters.

IFAIL = 5

As for IFAIL = 3 but the error occurred when calculating the derivatives of the residuals with respect to the parameters.

IFAIL = 6

The calculated Jacobian has an insignificant column.

Note: IFAIL = 4, 5 or 6 usually indicate a badly scaled problem. The user may vary the size of TOL or change to one of the more general routines D02HBF or D02SAF which afford more control over the calculations.

IFAIL = 7

The linear algebra routine (F02WEF) used has failed. This error exit should not occur and can be avoided by changing the estimated initial values.

IFAIL = 8

The Newton iteration has failed to converge.

Note: IFAIL = 8 can indicate poor initial estimates or a very difficult problem. Consider varying TOL if the residuals are small in the monitoring output. If the residuals are large try varying the initial estimates.

IFAIL = 9, 10, 11, 12 or 13

Indicate that a serious error has occurred in D02SAZ, D02SAW, D02SAX, D02SAU or D02SAV respectively. Check all array subscripts and subroutine parameter lists in calls to D02HAF. Seek expert help.

7 Accuracy

If the process converges, the accuracy to which the unknown parameters are determined is usually close to that specified by the user; the solution, if requested, may be determined to a required accuracy by varying the parameter TOL.

8 Further Comments

The time taken by the routine depends on the complexity of the system, and on the number of iterations required. In practice, integration of the differential equations is by far the most costly process involved.

Wherever it occurs in the routine, the error parameter TOL is used in 'mixed' form; that is TOL always occurs in expressions of the form TOL × $(1+|y_i|)$. Though not ideal for every application, it is expected that this mixture of absolute and relative error testing will be adequate for most purposes.

The user is strongly recommended to set IFAIL to obtain self-explanatory error messages, and also monitoring information about the course of the computation. The user may select the channel numbers on which this output is to appear by calls of X04AAF (for error messages) or X04ABF (for monitoring information) – see Section 9 for an example. Otherwise the default channel numbers will be used, as specified in the implementation document. The monitoring information produced at each iteration includes the current parameter values, the residuals and two norms: a basic norm and a current norm. At each iteration the aim is to find parameter values which make the current norm less than the basic norm. Both these norms should tend to zero as should the residuals. (They would all be zero if the exact parameters were used as input.) For more details, the user may consult the specification of D02SAF, and especially the description of the parameter MONIT there.

The computing time for integrating the differential equations can sometimes depend critically on the quality of the initial estimates. If it seems that too much computing time is required and, in particular, if the values of the residuals printed by the monitoring routine are much larger than the expected values of the solution at b, then the coding of the subroutine FCN should be checked for errors. If no errors can be found, an independent attempt should be made to improve the initial estimates. In practical problems it is not uncommon for the differential equation to have a singular point at one or both ends of the range. Suppose a is a singular point; then the derivatives y_i' in (1) (in Section 3) cannot be evaluated at a, usually because one or more of the expressions for f_i give overflow. In such a case it is necessary for the user to take a a short distance away from the singularity, and to find values for y_i at the new value of a (e.g. use the first one or two terms of an analytical (power series) solution). The user should experiment with the new positon of a; if it is taken too close to the singular point, the derivatives f_i will be inaccurate, and the routine may sometimes fail with IFAIL = 2 or 3 or, in extreme cases, with an overflow condition. A more general treatment of singular solutions is provided by the subroutine D02HBF.

Another difficulty which often arises in practice is the case when one end of the range, b say, is at infinity. The user must approximate the end-point by taking a finite value for b, which is obtained by estimating where the solution will reach its asymptotic state. The estimate can be checked by repeating the calculation with a larger value of b. If b is very large, and if the matching point is also at b, the numerical solution may suffer a considerable loss of accuracy in integrating across the range, and the program may fail with IFAIL = 6 or 8. (In the former case, solutions from all initial values at a are tending to the same curve at infinity.) The simplest remedy is to try to solve the equations with a smaller value of b, and then to increase b in stages, using each solution to give boundary value estimates for the next calculation. For problems where some terms in the asymptotic form of the solution are known, D02HBF will be more successful.

If the unknown quantities are not boundary values, but are eigenvalues or the length of the range or some other parameters occurring in the differential equations, D02HBF may be used.

9 Example

To find the angle at which a projectile must be fired for a given range.

The differential equations are:

$$\begin{aligned} y' &= \tan\phi \\ v' &= \frac{-0.032\tan\phi}{v} - \frac{0.02v}{\cos\phi} \\ \phi' &= \frac{-0.032}{v^2}, \end{aligned}$$

with the following boundary conditions:

$$y = 0,\quad v = 0.5 \quad \text{at } x = 0,$$

$$y = 0 \qquad\qquad \text{at } x = 5.$$

The remaining boundary conditions are estimated as:

$$\phi = 1.15 \qquad\qquad \text{at } x = 0,$$

$$\phi = 1.2,\quad v = 0.46 \quad \text{at } x = 5.$$

We write y = Z(1), v= Z(2), ϕ = Z(3). To check the accuracy of the results the problem is solved twice with TOL = 5.0E−3 and 5.0E−4 respectively. Note the call to X04ABF before the call to D02HAF.

9.1 Program Text

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

```
*     D02HAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
*     N.B the definition of IW must be changed for N.GT.11
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          N, IW, M1
      PARAMETER        (N=3,IW=3*N+17+11,M1=6)
*     .. Local Scalars ..
      real             TOL, X, X1
      INTEGER          I, IFAIL, J, L
*     .. Local Arrays ..
      real             U(N,2), V(N,2), W(N,IW), Y(N,M1)
*     .. External Subroutines ..
      EXTERNAL         D02HAF, DERIV, X04ABF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02HAF Example Program Results'
      CALL X04ABF(1,NOUT)
      DO 40 L = 3, 4
         TOL = 5.0e0*10.0e0**(-L)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Results with TOL = ', TOL
         U(1,1) = 0.0e0
         V(1,1) = 0.0e0
         U(1,2) = 0.0e0
         V(1,2) = 0.0e0
         U(2,1) = 0.5e0
         V(2,1) = 0.0e0
         U(2,2) = 0.46e0
         V(2,2) = 1.0e0
         U(3,1) = 1.15e0
         V(3,1) = 1.0e0
         U(3,2) = -1.2e0
         V(3,2) = 1.0e0
         X = 0.0e0
         X1 = 5.0e0
*        * Set IFAIL to 111 to obtain monitoring information *
         IFAIL = 11
*
         CALL D02HAF(U,V,N,X,X1,TOL,DERIV,Y,M1,W,IW,IFAIL)
*
         WRITE (NOUT,*)
         IF (IFAIL.EQ.0) THEN
            WRITE (NOUT,*) ' X-value and final solution'
            DO 20 I = 1, M1
               WRITE (NOUT,99998) I - 1, (Y(J,I),J=1,N)
   20       CONTINUE
         ELSE
            WRITE (NOUT,99997) ' IFAIL =', IFAIL
         END IF
   40 CONTINUE
      STOP
*
99999 FORMAT (1X,A,e10.3)
99998 FORMAT (1X,I3,3F10.4)
99997 FORMAT (1X,A,I4)
      END
*
      SUBROUTINE DERIV(X,Z,G)
*     .. Parameters ..
      INTEGER          N
      PARAMETER        (N=3)
*     .. Scalar Arguments ..
      real             X
*     .. Array Arguments ..
      real             G(N), Z(N)
*     .. Intrinsic Functions ..
      INTRINSIC        COS, TAN
*     .. Executable Statements ..
      G(1) = TAN(Z(3))
      G(2) = -0.032e0*TAN(Z(3))/Z(2) - 0.02e0*Z(2)/COS(Z(3))
      G(3) = -0.032e0/Z(2)**2
      RETURN
      END
```

9.2 Program Data

None.

9.3 Program Results

```
D02HAF Example Program Results

Results with TOL =  0.500E-02

 X-value and final solution
  0     0.0000    0.5000    1.1680
  1     1.9172    0.3343    0.9746
  2     2.9293    0.2067    0.4916
  3     2.9762    0.1956   -0.4214
  4     2.0177    0.3099   -0.9756
  5    -0.0088    0.4602   -1.2020

Results with TOL =  0.500E-03

 X-value and final solution
  0     0.0000    0.5000    1.1681
  1     1.9177    0.3343    0.9749
  2     2.9280    0.2070    0.4929
  3     2.9769    0.1955   -0.4194
  4     2.0210    0.3095   -0.9751
  5     0.0000    0.4597   -1.2014
```

D02HBF – NAG Fortran Library Routine Document

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

1 Purpose

D02HBF solves the two-point boundary-value problem for a system of ordinary differential equations, using initial value techniques and Newton iteration; it generalises subroutine D02HAF to include the case where parameters other than boundary values are to be determined.

2 Specification

```
      SUBROUTINE D02HBF(P, N1, PE, E, N, SOLN, M1, FCN, BC, RANGE, W,
     1                  IW, IFAIL)
      INTEGER           N1, N, M1, IW, IFAIL
      real              P(N1), PE(N1), E(N), SOLN(N,M1), W(N,IW)
      EXTERNAL          FCN, BC, RANGE
```

3 Description

D02HBF solves the two-point boundary-value problem by determining the unknown parameters $p_1, p_2, \ldots, p_{n_1}$ of the problem. These parameters may be, but need not be, boundary values; they may include eigenvalue parameters in the coefficients of the differential equations, length of the range of integration, etc. The notation and methods used are similar to those of D02HAF and the user is advised to study this first. (The parameters $p_1, p_2, \ldots, p_{n_1}$ correspond precisely to the unknown boundary conditions in D02HAF.) It is assumed that we have a system of n first-order ordinary differential equations of the form:

$$\frac{dy_i}{dx} = f_i(x, y_1, y_2, \ldots, y_n), \quad i = 1, 2, \ldots, n,$$

and that the derivatives f_i are evaluated by a subroutine FCN supplied by the user. The system, including the boundary conditions given by BC and the range of integration given by RANGE, involves the n_1 unknown parameters $p_1, p_2, \ldots, p_{n_1}$ which are to be determined, and for which initial estimates must be supplied. The number of unknown parameters n_1 must not exceed the number of equations n. If $n_1 < n$, we assume that $(n - n_1)$ equations of the system are not involved in the matching process. These are usually referred to as 'driving equations'; they are independent of the parameters and of the solutions of the other n_1 equations. In numbering the equations for the subroutine FCN, the driving equations must be put **first**.

The estimated values of the parameters are corrected by a form of Newton iteration. The Newton correction on each iteration is calculated using a Jacobian matrix whose (i, j)th element depends on the derivative of the ith component of the solution, y_i, with respect to the jth parameter, p_j. This matrix is calculated by a simple numerical differentiation technique which requires n_1 evaluations of the differential system.

If the parameter IFAIL is set appropriately, the routine automatically prints messages to inform the user of the flow of the calculation. These messages are discussed in detail in Section 8.

D02HBF is a simplified version of D02SAF which is described in detail in Gladwell [1].

4 References

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

5 Parameters

Users are strongly recommended to read Section 3 and Section 8 in conjunction with this section.

1: P(N1) — ***real*** array *Input/Output*

On entry: an estimate for the ith parameter, p_i, for $i = 1, 2, \ldots, n_1$.

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

2: N1 — INTEGER *Input*

On entry: the number of parameters, n_1.

Constraint: $1 \leq \mathrm{N1} \leq \mathrm{N}$.

3: PE(N1) — ***real*** array *Input*

On entry: the elements of PE must be given small positive values. The element PE(i) is used

(i) in the convergence test on the ith parameter in the Newton iteration, and
(ii) in perturbing the ith parameter when approximating the derivatives of the components of the solution with respect to this parameter for use in the Newton iteration.

The elements PE(i) should not be chosen too small. They should usually be several orders of magnitude larger than ***machine precision***.

Constraint: PE(i) > 0.0, for $i = 1, 2, \ldots$,N1.

4: E(N) — ***real*** array *Input*

On entry: the elements of E must be given positive values.The element E(i) is used in the bound on the local error in the ith component of the solution y_i during integration.

The elements E(i) should not be chosen too small. They should usually be several orders of magnitude larger than ***machine precision***.

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

5: N — INTEGER *Input*

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

Constraint: $\mathrm{N} \geq 2$.

6: SOLN(N,M1) — ***real*** array *Output*

On exit: the solution when M1 > 1 (see below).

7: M1 — INTEGER *Input*

On entry: a value which controls exit values as follows:

M1 = 1

The final solution is not calculated;

M1 > 1

The final values of the solution at interval (length of range)/(M1−1) are calculated and stored sequentially in the array SOLN starting with the values of the solutions evaluated at the first end-point (see subroutine RANGE below) stored in the first column of SOLN.

Constraint: $\mathrm{M1} \geq 1$.

8: FCN — SUBROUTINE, supplied by the user. *External Procedure*

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

Its specification is:

```
SUBROUTINE FCN(X, Y, F, P)
real           X, Y(n), F(n), P(n1)
```

where n and n1 are the actual values of N and N1 in the call of D02HBF.

1: X — ***real*** *Input*

On entry: the value of the argument x.

2: Y(n) — ***real*** array *Input*

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

3: F(n) — ***real*** array *Output*

On exit: the value of f_i, for $i = 1,2,\ldots,n$. The f_i may depend upon the parameters p_j, for $j = 1,2,\ldots,n_1$. If there are any driving equations (see Section 3) then these must be numbered first in the ordering of the components of F in FCN.

4: P(n1) — ***real*** array *Input*

On entry: the current estimate of the parameter p_i, for $i = 1,2,\ldots,n_1$.

FCN must be declared as EXTERNAL in the (sub)program from which D02HBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

9: BC — SUBROUTINE, supplied by the user. *External Procedure*

BC must place in G1 and G2 the boundary conditions at a and b respectively (see RANGE below).

Its specification is:

```
SUBROUTINE BC(G1, G2, P)
real           G1(n), G2(n), P(n1)
```

where n and n1 are the actual values of N and N1 in the call of D02HBF.

1: G1(n) — ***real*** array *Output*

On exit: the value of $y_i(a)$, (where this may be a known value or a function of the parameters p_j, for $j = 1,2,\ldots,n_1$); $i = 1,2,\ldots,n$.

2: G2(n) — ***real*** array *Output*

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

3: P(n1) — ***real*** array *Input*

On entry: an estimate of the parameter p_i, for $i = 1,2,\ldots,n_1$.

BC must be declared as EXTERNAL in the (sub)program from which D02HBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

10: RANGE — SUBROUTINE, supplied by the user. *External Procedure*

RANGE must evaluate the boundary points a and b, each of which may depend on the parameters $p_1, p_2, \ldots, p_{n_1}$. The integrations in the shooting method are always from a to b.

Its specification is:

```
      SUBROUTINE RANGE(A, B, P)
      real            A, B, P(n1)
```

where n1 is the actual value of N1 in the call of D02HBF.

1: A — ***real*** *Output*

On exit: one of the boundary points, a.

2: B — ***real*** *Output*

On exit: the second boundary point, b. Note that B > A forces the direction of integration to be that of increasing X. If A and B are interchanged the direction of integration is reversed.

3: P(n1) — ***real*** array *Input*

On entry: the current estimate of the ith parameter, p_i, for $i = 1, 2, \ldots, n_1$.

RANGE must be declared as EXTERNAL in the (sub)program from which D02HBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

11: W(N,IW) — ***real*** array *Output*

Used mainly for workspace.

On exit: with IFAIL = 2, 3, 4 or 5 (see Section 6), W(i, 1), for $i = 1, 2, \ldots, n$ contains the solution at the point x when the error occurred. W(1,2) contains x.

12: IW — INTEGER *Input*

On entry: the second dimension of the array W as declared in the (sub)program from which D02HBF is called.

Constraint: IW $\geq$ 3N + 14 + max(11,N).

13: IFAIL — INTEGER *Input/Output*

For this routine, the normal use of IFAIL is extended to control the printing of error and warning messages as well as specifying hard or soft failure (see Chapter P01).

Before entry, IFAIL must be set to a value with the decimal expansion *cba*, where each of the decimal digits c, b and a must have a value of 0 or 1.

$a = 0$ specifies hard failure, otherwise soft failure;

$b = 0$ suppresses error messages, otherwise error messages will be printed (see Section 6);

$c = 0$ suppresses warning messages, otherwise warning messages will be printed (see Section 6).

The recommended value for inexperienced users is 110 (i.e., hard failure with all messages printed).

Unless the routine detects an error (see Section 6), IFAIL contains 0 on exit.

6 Error Indicators and Warnings

For each error, an explanatory error message is output on the current error message unit (as defined by X04AAF), unless suppressed by the value of IFAIL on entry.

Errors detected by the routine:

IFAIL = 1

One or more of the parameters N, N1, M1, IW, E or PE is incorrectly set.

IFAIL = 2

The step length for the integration became too short whilst calculating the residual (see Section 8).

IFAIL = 3

No initial step length could be chosen for the integration whilst calculating the residual.

Note: IFAIL = 2 or 3 can occur due to choosing too small a value for E or due to choosing the wrong direction of integration. Try varying E and interchanging a and b. These error exits can also occur for very poor initial choices of the parameters in the array P and, in extreme cases, because this routine cannot be used to solve the problem posed.

IFAIL = 4

As for IFAIL = 2 but the error occurred when calculating the Jacobian.

IFAIL = 5

As for IFAIL = 3 but the error occurred when calculating the Jacobian.

IFAIL = 6

The calculated Jacobian has an insignificant column. This can occur because a parameter p_i is incorrectly entered when posing the problem.

Note: IFAIL = 4, 5 or 6 usually indicate a badly scaled problem. The user may vary the size of PE. Otherwise the use of the more general D02SAF which affords more control over the calculations is advised.

IFAIL = 7

The linear algebra routine used (F02WEF) has failed. This error exit should not occur and can be avoided by changing the initial estimates p_i.

IFAIL = 8

The Newton iteration has failed to converge. This can indicate a poor initial choice of parameters p_i or a very difficult problem. Consider varying the elements PE(i) if the residuals are small in the monitoring output. If the residuals are large, try varying the initial parameters p_i.

IFAIL = 9, 10, 11, 12 or 13

Indicate that a serious error has occurred in D02SAZ, D02SAW, D02SAX, D02SAU or D02SAV respectively. Check all array subscripts and subroutine parameter lists in the call to D02HBF. Seek expert help.

7 Accuracy

If the process converges, the accuracy to which the unknown parameters are determined is usually close to that specified by the user; and the solution, if requested, may be determined to a required accuracy by varying the parameter E.

8 Further Comments

The time taken by the routine depends on the complexity of the system, and on the number of iterations required. In practice, integration of the differential equations is by far the most costly process involved.

Wherever they occur in the routine, the error parameters contained in the arrays E and PE are used in 'mixed' form; that is E(i) always occurs in expressions of the form

$$\mathrm{E}(i) \times (1 + |y_i|)$$

and PE(i) always occurs in expressions of the form

$$\mathrm{PE}(i) \times (1 + |p_i|)$$

Though not ideal for every application, it is expected that this mixture of absolute and relative error testing will be adequate for most purposes.

The user may determine a suitable direction of integration a to b and suitable values for E(i) by integrations with D02PCF. The best direction of integration is usually the direction of decreasing solutions. The user is strongly recommended to set IFAIL to obtain self-explanatory error messages, and also monitoring information about the course of the computation. The user may select the channel numbers on which this output is to appear by calls of X04AAF (for error messages) or X04ABF (for monitoring information) – see Section 9 for an example. Otherwise the default channel numbers will be used, as specified in the implementation document. The monitoring information produced at each iteration includes the current parameter values, the residuals and two norms: a basic norm and a current norm. At each iteration the aim is to find parameter values which make the current norm less than the basic norm. Both these norms should tend to zero as should the residuals. (They would all be zero if the exact parameters were used as input.) For more details, in particular about the other monitoring information printed, the user is advised to consult the specification of D02SAF and, especially, the description of the parameter MONIT there.

The computing time for integrating the differential equations can sometimes depend critically on the quality of the initial estimates for the parameters p_i. If it seems that too much computing time is required and, in particular, if the values of the residuals printed by the monitoring routine are much larger than the expected values of the solution at b then the coding of the subroutines FCN, BC and RANGE should be checked for errors. If no errors can be found, an independent attempt should be made to improve the initial estimates for p_i.

The subroutine can be used to solve a very wide range of problems, for example:

(a) eigenvalue problems, including problems where the eigenvalue occurs in the boundary conditions;

(b) problems where the differential equations depend on some parameters which are to be determined so as to satisfy certain boundary conditions (see example (ii) in Section 9);

(c) problems where one of the end-points of the range of integration is to be determined as the point where a variable y_i takes a particular value (see example (ii) in Section 9);

(d) singular problems and problems on infinite ranges of integration where the values of the solution at a or b or both are determined by a power series or an asymptotic expansion (or a more complicated expression) and where some of the coefficients in the expression are to be determined (see example (i) in Section 9); and

(e) differential equations with certain terms defined by other independent (driving) differential equations.

9 Example

For this routine two examples are presented, in Section 9.1 and Section 9.2. In the example programs distributed to sites, there is a single example program for D02HBF, with a main program:

```
*     D02HBF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. External Subroutines ..
      EXTERNAL         EX1, EX2
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02HBF Example Program Results'
      CALL EX1
      CALL EX2
      STOP
      END
```

The code to solve the two example problems is given in the subroutines EX1 and EX2, in Section 9.1.1 and Section 9.2.1 respectively.

9.1 Example 1

To find the solution of the differential equation

$$y'' = (y^3 - y')/2x$$

on the range $0 \leq x \leq 16$, with boundary conditions $y(0) = 0.1$ and $y(16) = 1/6$. We cannot use the differential equation at $x = 0$ because it is singular, so we take a truncated power series expansion

$$y(x) = 1/10 + p_1 \times \sqrt{x}/10 + x/100$$

near the origin where p_1 is one of the parameters to be determined. We choose the interval as [0.1,16] and setting $p_2 = y'(16)$, we can determine all the boundary conditions. We take X1 = 16. We write y = Y(1), y' = Y(2), and estimate PARAM(1) = 0.2, PARAM(2) = 0.0. Note the call to X04ABF before the call to D02HBF.

9.1.1 Program Text

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

```
*
      SUBROUTINE EX1
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          N, N1, IW, M1
      PARAMETER        (N=2,N1=2,IW=3*N+14+11,M1=6)
*     .. Local Scalars ..
      real             H, X, X1
      INTEGER          I, IFAIL, J
*     .. Local Arrays ..
      real             C(N,M1), ERROR(N), PARAM(N1), PARERR(N1), W(N,IW)
*     .. External Subroutines ..
      EXTERNAL         AUX1, BCAUX1, D02HBF, RNAUX1, X04ABF
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 1'
      CALL X04ABF(1,NOUT)
      PARAM(1) = 0.2e0
      PARAM(2) = 0.0e0
      PARERR(1) = 1.0e-5
      PARERR(2) = 1.0e-3
      ERROR(1) = 1.0e-4
      ERROR(2) = 1.0e-4
*     * Set IFAIL to 111 to obtain monitoring information *
      IFAIL = 11
*
      CALL D02HBF(PARAM,N1,PARERR,ERROR,N,C,M1,AUX1,BCAUX1,RNAUX1,W,IW,
     +            IFAIL)
*
      WRITE (NOUT,*)
      IF (IFAIL.NE.0) THEN
```

```
         WRITE (NOUT,99999) 'IFAIL = ', IFAIL
         IF (IFAIL.LE.5 .AND. IFAIL.NE.1) THEN
            WRITE (NOUT,*)
            WRITE (NOUT,99996) 'W(1,2) = ', W(1,2), ' W(.,1) = ',
     +        (W(I,1),I=1,N)
         END IF
      ELSE
         WRITE (NOUT,*) 'Final parameters'
         WRITE (NOUT,99998) (PARAM(I),I=1,N1)
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Final solution'
         WRITE (NOUT,*) 'X-value     Components of solution'
         CALL RNAUX1(X,X1,PARAM)
         H = (X1-X)/real(M1-1)
         DO 20 I = 1, M1
            WRITE (NOUT,99997) X + (I-1)*H, (C(J,I),J=1,N)
   20    CONTINUE
      END IF
      RETURN
*
99999 FORMAT (1X,A,I3)
99998 FORMAT (1X,1P,3e15.3)
99997 FORMAT (1X,F7.2,2F13.4)
99996 FORMAT (1X,A,F9.4,A,10e10.3)
      END
*
      SUBROUTINE AUX1(X,Y,F,PARAM)
*     .. Parameters ..
      INTEGER         N
      PARAMETER       (N=2)
*     .. Scalar Arguments ..
      real            X
*     .. Array Arguments ..
      real           F(N), PARAM(N), Y(N)
*     .. Executable Statements ..
      F(1) = Y(2)
      F(2) = (Y(1)**3-Y(2))/(2.0e0*X)
      RETURN
      END
*
      SUBROUTINE RNAUX1(X,X1,PARAM)
*     .. Scalar Arguments ..
      real              X, X1
*     .. Array Arguments ..
      real              PARAM(2)
*     .. Executable Statements ..
      X = 0.1e0
      X1 = 16.0e0
      RETURN
      END
*
      SUBROUTINE BCAUX1(G,G1,PARAM)
*     .. Parameters ..
      INTEGER           N
      PARAMETER         (N=2)
*     .. Array Arguments ..
      real              G(N), G1(N), PARAM(N)
```

```
*     .. Local Scalars ..
      real             Z
*     .. Intrinsic Functions ..
      INTRINSIC        SQRT
*     .. Executable Statements ..
      Z = 0.1e0
      G(1) = 0.1e0 + PARAM(1)*SQRT(Z)*0.1e0 + 0.01e0*Z
      G(2) = PARAM(1)*0.05e0/SQRT(Z) + 0.01e0
      G1(1) = 1.0e0/6.0e0
      G1(2) = PARAM(2)
      RETURN
      END
```

9.1.2 Program Data

None.

9.1.3 Program Results

```
 D02HBF Example Program Results


 Case 1

 Final parameters
       4.629E-02      3.494E-03

 Final solution
 X-value     Components of solution
    0.10        0.1025        0.0173
    3.28        0.1217        0.0042
    6.46        0.1338        0.0036
    9.64        0.1449        0.0034
   12.82        0.1557        0.0034
   16.00        0.1667        0.0035
```

9.2 Example 2

To find the gravitational constant p_1 and the range p_2 over which a projectile must be fired to hit the target with a given velocity.

The differential equations are

$$y' = \tan\phi$$
$$v' = \frac{-(p_1 \sin\phi + 0.00002v^2)}{v\cos\phi}$$
$$\phi' = \frac{-p_1}{v^2}$$

on the range $0 < x < p_2$, with boundary conditions

$$\begin{array}{llll} y=0, & v=500, & \phi=0.5 & \text{at } x=0, \\ y=0, & v=450, & \phi=p_3 & \text{at } x=p_2 \end{array}.$$

We write $y = \mathrm{Y}(1)$, $v = \mathrm{Y}(2)$, $\phi = \mathrm{Y}(3)$. We estimate $p_1 = \mathrm{PARAM}(1) = 32$, $p_2 = \mathrm{PARAM}(2) = 6000$ and $p_3 = \mathrm{PARAM}(3) = 0.54$ (though this last estimate is not important).

9.2.1 Program Text

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

```
*
      SUBROUTINE EX2
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          N, N1, IW, M1
      PARAMETER        (N=3,N1=3,IW=3*N+14+11,M1=6)
*     .. Local Scalars ..
      real             H, X, X1
      INTEGER          I, IFAIL, J
*     .. Local Arrays ..
      real             C(N,M1), ERROR(N), PARAM(N1), PARERR(N1), W(N,IW)
*     .. External Subroutines ..
      EXTERNAL         AUX2, BCAUX2, D02HBF, RNAUX2, X04ABF
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Case 2'
      CALL X04ABF(1,NOUT)
      PARAM(1) = 32.0e0
      PARAM(2) = 6000.0e0
      PARAM(3) = 0.54e0
      PARERR(1) = 1.0e-5
      PARERR(2) = 1.0e-4
      PARERR(3) = 1.0e-4
      ERROR(1) = 1.0e-2
      ERROR(2) = 1.0e-2
      ERROR(3) = 1.0e-2
*     * Set IFAIL to 111 to obtain monitoring information *
      IFAIL = 11
*
      CALL D02HBF(PARAM,N1,PARERR,ERROR,N,C,M1,AUX2,BCAUX2,RNAUX2,W,IW,
     +            IFAIL)
*
      WRITE (NOUT,*)
      IF (IFAIL.NE.0) THEN
         WRITE (NOUT,99999) 'IFAIL = ', IFAIL
         IF (IFAIL.LE.5 .AND. IFAIL.NE.1) THEN
            WRITE (NOUT,*)
            WRITE (NOUT,99996) 'W(1,2) = ', W(1,2), ' W(.,1) = ',
     +        (W(I,1),I=1,N)
         END IF
      ELSE
         WRITE (NOUT,*) 'Final parameters'
         WRITE (NOUT,99998) (PARAM(I),I=1,N1)
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Final solution'
         WRITE (NOUT,*) 'X-value     Components of solution'
         CALL RNAUX2(X,X1,PARAM)
         H = (X1-X)/real(M1-1)
         DO 20 I = 1, M1
            WRITE (NOUT,99997) X + (I-1)*H, (C(J,I),J=1,N)
```

```
   20    CONTINUE
      END IF
      RETURN
*
99999 FORMAT (1X,A,I3)
99998 FORMAT (1X,1P,3e15.3)
99997 FORMAT (1X,F7.0,2F13.1,F13.3)
99996 FORMAT (1X,A,F9.4,A,10e10.3)
      END
*
      SUBROUTINE AUX2(X,Y,F,PARAM)
*     .. Parameters ..
      INTEGER         N
      PARAMETER       (N=3)
*     .. Scalar Arguments ..
      real            X
*     .. Array Arguments ..
      real            F(N), PARAM(N), Y(N)
*     .. Intrinsic Functions ..
      INTRINSIC       COS, TAN
*     .. Executable Statements ..
      F(1) = TAN(Y(3))
      F(2) = -PARAM(1)*TAN(Y(3))/Y(2) - 0.00002e0*Y(2)/COS(Y(3))
      F(3) = -PARAM(1)/Y(2)**2
      RETURN
      END
*
      SUBROUTINE RNAUX2(X,X1,PARAM)
*     .. Parameters ..
      INTEGER           N
      PARAMETER         (N=3)
*     .. Scalar Arguments ..
      real              X, X1
*     .. Array Arguments ..
      real              PARAM(N)
*     .. Executable Statements ..
      X = 0.0e0
      X1 = PARAM(2)
      RETURN
      END
*
      SUBROUTINE BCAUX2(G,G1,PARAM)
*     .. Parameters ..
      INTEGER           N
      PARAMETER         (N=3)
*     .. Array Arguments ..
      real              G(N), G1(N), PARAM(N)
*     .. Executable Statements ..
      G(1) = 0.0e0
      G(2) = 500.0e0
      G(3) = 0.5e0
      G1(1) = 0.0e0
      G1(2) = 450.0e0
      G1(3) = PARAM(3)
      RETURN
      END
```

9.2.2 Program Data

None.

9.2.3 Program Results

```
 Case 2

 Final parameters
       3.239E+01      5.962E+03     -5.353E-01

 Final solution
 X-value     Components of solution
      0.          0.0          500.0          0.500
   1192.        529.6          451.6          0.328
   2385.        807.2          420.3          0.123
   3577.        820.4          409.4         -0.103
   4769.        556.1          420.0         -0.330
   5962.          0.0          450.0         -0.535
```

D02JAF – NAG Fortran Library Routine Document

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

1. Purpose

D02JAF solves a regular linear two-point boundary value problem for a single *n*th order ordinary differential equation by Chebyshev-series using collocation and least-squares.

2. Specification

```
SUBROUTINE D02JAF (N, CF, BC, X0, X1, K1, KP, C, W, LW, IW, IFAIL)
INTEGER        N, K1, KP, LW, IW(K1), IFAIL
real           CF, X0, X1, C(K1), W(LW)
EXTERNAL       CF, BC
```

3. Description

This routine calculates the solution of a regular two-point boundary value problem for a single *n*th order linear ordinary differential equation as a Chebyshev-series in the range (x_0,x_1). The differential equation

$$f_{n+1}(x)y^{(n)}(x) + f_n(x)y^{(n-1)}(x) + \ldots + f_1(x)y(x) = f_0(x)$$

is defined by the user-supplied function CF, and the boundary conditions at the points x_0 and x_1 are defined by the user-supplied subroutine BC.

The user specifies the degree of Chebyshev-series required, K1 − 1, and the number of collocation points, KP. The routine sets up a system of linear equations for the Chebyshev coefficients, one equation for each collocation point and one for each boundary condition. The boundary conditions are solved exactly, and the remaining equations are then solved by a least-squares method. The result produced is a set of coefficients for a Chebyshev-series solution of the differential equation on a range normalised to the range $(-1,1)$.

E02AKF can be used to evaluate the solution at any point on the range (x_0,x_1) – see Section 9 for an example. E02AHF followed by E02AKF can be used to evaluate its derivatives.

4. References

[1] PICKEN, S.M.
Algorithms for the solution of differential equations in Chebyshev-series by the selected points method.
Report Math. 94, National Physical Laboratory, Teddington, 1970.

5. Parameters

1: N – INTEGER. *Input*

On entry: the order *n* of the differential equation.

Constraint: N ≥ 1.

2: CF – ***real*** FUNCTION, supplied by the user. *External Procedure*

CF defines the differential equation (see Section 3). It must return the value of a function $f_j(x)$ at a given point x, where, for $1 \le j \le n+1, f_j(x)$ is the coefficient of $y^{(j-1)}(x)$ in the equation, and $f_0(x)$ is the right-hand side.

Its specification is:

```
real FUNCTION CF(J, X)
INTEGER       J
real          X
```

1: J – INTEGER. *Input*

On entry: the index of the function f_j to be evaluated.

2: X – ***real.*** *Input*

On entry: the point at which f_j is to be evaluated.

CF must be declared as EXTERNAL in the (sub)program from which D02JAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

BC defines the boundary conditions, each of which has the form $y^{(k-1)}(x_1) = s_k$ or $y^{(k-1)}(x_0) = s_k$. The boundary conditions may be specified in any order.

Its specification is:

```
SUBROUTINE BC(I, J, RHS)
INTEGER    I, J
real       RHS
```

1: I – INTEGER. *Input*

On entry: the index of the boundary condition to be defined.

2: J – INTEGER. *Output*

On exit: J must be set to $-k$ if the boundary condition is $y^{(k-1)}(x_0) = s_k$, and to $+k$ if it is $y^{(k-1)}(x_1) = s_k$,

J must not be set to the same value k for two different values of I.

3: RHS – ***real.*** *Output*

On exit: RHS must be set to the value s_k.

BC must be declared as EXTERNAL in the (sub)program from which D02JAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

4: X0 – ***real.*** *Input*
5: X1 – ***real.*** *Input*

On entry: the left- and right-hand boundaries, x_0 and x_1, respectively.

Constraint: X1 > X0.

6: K1 – INTEGER. *Input*

On entry: the number of coefficients to be returned in the Chebyshev-series representation of the solution (hence the degree of the polynomial approximation is K1 – 1).

Constraint: K1 ≥ N + 1.

7: KP – INTEGER. *Input*

On entry: the number of collocation points to be used.

Constraint: KP ≥ K1 – N.

8: C(K1) – ***real*** array. *Output*

On exit: the computed Chebyshev coefficients; that is, the computed solution is:

$$\sum_{i=1}^{K1}{}' C(i)\ T_{i-1}(x)$$

where $T_i(x)$ is the ith Chebyshev polynomial of the first kind, and $\sum'$ denotes that the first coefficient, C(1), is halved.

9: W(LW) – ***real*** array. *Workspace*

10: LW – INTEGER. *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D02JAF is called.

Constraint: LW ≥ 2×(KP+N)×(K1+1) + 7×K1.

11: IW(K1) – INTEGER array. *Workspace*

12: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, N < 1,
or X0 ≥ X1,
or K1 < N + 1,
or KP < K1 – N.

IFAIL = 2

On entry LW < 2×(KP+N)×(K1+1) + 7×K1 (insufficient workspace).

IFAIL = 3

Either the boundary conditions are not linearly independent (that is, in the subroutine BC the variable j is set to the same value k for two different values of i), or the rank of the matrix of equations for the coefficients is less than the number of unknowns. Increasing KP may overcome this problem.

IFAIL = 4

The least-squares routine F04AMF has failed to correct the first approximate solution (see the routine document for F04AMF).

7. Accuracy

The Chebyshev coefficients are determined by a stable numerical method. The accuracy of the approximate solution may be checked by varying the degree of the polynomial and the number of collocation points (see Section 8).

8. Further Comments

The time taken by the routine depends on the complexity of the differential equation, the degree of the polynomial solution, and the number of matching points.

The collocation points in the range (x_0,x_1) are chosen to be the extrema of the appropriate shifted Chebyshev polynomial. If KP = K1 – N, then the least-squares solution reduces to the solution of a system of linear equations, and true collocation results. The accuracy of the solution may be checked by repeating the calculation with different values of K1 and with KP fixed but KP ≫ K1 – N. If the Chebyshev coefficients decrease rapidly (and consistently for various K1 and KP), the size of the last two or three gives an indication of the error. If the Chebyshev coefficients do not decay rapidly, it is likely that the solution cannot be well-represented by Chebyshev-series. Note that the Chebyshev coefficients are calculated for the range (–1,1).

Systems of regular linear differential equations can be solved using D02JBF. It is necessary before using this routine to write the differential equations as a first order system. Linear systems of high order equations in their original form, singular problems, and, indirectly, nonlinear problems can be solved using D02TGF.

9. Example

To solve the equation

$$y'' + y = 1$$

with boundary conditions

$$y(-1) = y(1) = 0.$$

We use K1 = 4,6,8 and KP = 10 and 15, so that the different Chebyshev-series may be compared. The solution for K1 = 8 and KP = 15 is evaluated by E02AKF at 9 equally spaced points over the interval $(-1,1)$.

9.1. Program Text

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

```
*     D02JAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, K1MAX, KPMAX, LW
      PARAMETER        (N=2,K1MAX=8,KPMAX=15,LW=2*(KPMAX+N)*(K1MAX+1)
     +                 +7*K1MAX)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             X, X0, X1, Y
      INTEGER          I, IA1, IFAIL, K1, KP, M
*     .. Local Arrays ..
      real             C(K1MAX), W(LW)
      INTEGER          IW(K1MAX)
*     .. External Subroutines ..
      EXTERNAL         BC, CF, D02JAF, E02AKF
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02JAF Example Program Results'
      X0 = -1.0e0
      X1 = 1.0e0
      WRITE (NOUT,*)
      WRITE (NOUT,*) ' KP  K1   Chebyshev coefficients'
      DO 40 KP = 10, KPMAX, 5
         DO 20 K1 = 4, K1MAX, 2
            IFAIL = 1
*
            CALL D02JAF(N,CF,BC,X0,X1,K1,KP,C,W,LW,IW,IFAIL)
*
            IF (IFAIL.NE.0) THEN
               WRITE (NOUT,99999) KP, K1, '  D02JAF fails with IFAIL =',
     +           IFAIL
               STOP
            ELSE
               WRITE (NOUT,99998) KP, K1, (C(I),I=1,K1)
            END IF
   20    CONTINUE
   40 CONTINUE
      K1 = 8
      M = 9
      IA1 = 1
      WRITE (NOUT,*)
      WRITE (NOUT,99997) 'Last computed solution evaluated at', M,
     +  ' equally spaced points'
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y'
      DO 60 I = 1, M
         X = (X0*real(M-I)+X1*real(I-1))/real(M-1)
         IFAIL = 0
*
         CALL E02AKF(K1,X0,X1,C,IA1,K1MAX,X,Y,IFAIL)
*
         WRITE (NOUT,99996) X, Y
   60 CONTINUE
      STOP
*
99999 FORMAT (1X,2(I3,1X),A,I4)
99998 FORMAT (1X,2(I3,1X),8F8.4)
99997 FORMAT (1X,A,I3,A)
99996 FORMAT (1X,2F10.4)
      END
*
      real FUNCTION CF(J,X)
*     .. Scalar Arguments ..
      real              X
      INTEGER           J
*     .. Executable Statements ..
      IF (J.EQ.2) THEN
         CF = 0.0e0
      ELSE
         CF = 1.0e0
      END IF
      RETURN
      END
*
      SUBROUTINE BC(I,J,RHS)
*     .. Scalar Arguments ..
      real            RHS
      INTEGER         I, J
*     .. Executable Statements ..
      RHS = 0.0e0
      IF (I.EQ.1) THEN
         J = 1
      ELSE
         J = -1
      END IF
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D02JAF Example Program Results

  KP  K1   Chebyshev coefficients
  10   4  -0.6108  0.0000  0.3054  0.0000
  10   6  -0.8316  0.0000  0.4246  0.0000 -0.0088  0.0000
  10   8  -0.8325  0.0000  0.4253  0.0000 -0.0092  0.0000  0.0001  0.0000
  15   4  -0.6174  0.0000  0.3087  0.0000
  15   6  -0.8316  0.0000  0.4246  0.0000 -0.0088  0.0000
  15   8  -0.8325  0.0000  0.4253  0.0000 -0.0092  0.0000  0.0001  0.0000
 Last computed solution evaluated at  9 equally spaced points

       X          Y
    -1.0000     0.0000
    -0.7500    -0.3542
    -0.5000    -0.6242
    -0.2500    -0.7933
     0.0000    -0.8508
     0.2500    -0.7933
     0.5000    -0.6242
     0.7500    -0.3542
     1.0000     0.0000
```

D02JBF – NAG Fortran Library Routine Document

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

1. Purpose

D02JBF solves a regular linear two-point boundary value problem for a system of ordinary differential equations by Chebyshev-series using collocation and least-squares.

2. Specification

```
      SUBROUTINE D02JBF (N, CF, BC, X0, X1, K1, KP, C, IC, W, LW, IW, LIW,
     1                   IFAIL)
      INTEGER            N, K1, KP, IC, LW, IW(LIW), LIW, IFAIL
      real               CF, X0, X1, C(IC,N), W(LW)
      EXTERNAL           CF, BC
```

3. Description

This routine calculates the solution of a regular two-point boundary value problem for a regular linear nth order system of first order ordinary differential equations in Chebyshev-series in the range (x_0,x_1). The differential equation

$$y' = A(x)y + r(x)$$

is defined by the user-supplied function CF and the boundary conditions at the points x_0 and x_1 are defined by the user-supplied routine BC (see Section 5).

The user specifies the degree of the Chebyshev-series required, K1 – 1, and the number of collocation points, KP. The routine sets up a system of linear equations for the Chebyshev coefficients, n equations for each collocation point and one for each boundary condition. The boundary conditions are solved exactly, and the remaining equations are then solved by a least-squares method. The result produced is a set of coefficients for a Chebyshev-series solution for each component of the solution of the system of differential equations on a range normalised to $(-1,1)$.

E02AKF can be used to evaluate the components of the solution at any point on the interval (x_0,x_1) – see Section 9 for an example. E02AHF followed by E02AKF can be used to evaluate their derivatives.

4. References

[1] PICKEN, S.M.
Algorithms for the solution of differential equations in Chebyshev-series by the selected points method.
Report Math. 94, National Physical Laboratory, Teddington, Middlesex, 1970.

5. Parameters

1: N – INTEGER. *Input*

On entry: the order of the system of differential equations, n.

Constraint: N ≥ 1.

2: CF – ***real*** FUNCTION, supplied by the user. *External Procedure*

CF defines the system of differential equations (see Section 3). It must return the value of a coefficient function $a_{i,j}(x)$, of A, at a given point x, or of a right-hand side function $r_i(x)$ if J = 0.

Its specification is:

```
real FUNCTION CF(I, J, X)
INTEGER       I, J
real          X
```

1: I – INTEGER. *Input*

2: J – INTEGER. *Input*

On entry: indicate the function to be evaluated, namely $a_{i,j}(x)$ if $1 \leq \mathrm{J} \leq n$, or $r_i(x)$ if $\mathrm{J} = 0$.

$1 \leq \mathrm{I} \leq n, 0 \leq \mathrm{J} \leq n.$

3: X – ***real.*** *Input*

On entry: the point at which the function is to be evaluated.

CF must be declared as EXTERNAL in the (sub)program from which D02JBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

BC defines the n boundary conditions, which have the form $y_k(x_0) = s$ or $y_k(x_1) = s$. The boundary conditions may be specified in any order.

Its specification is:

```
SUBROUTINE BC(I, J, RHS)
INTEGER      I, J
real         RHS
```

1: I – INTEGER. *Input*

On entry: the index of the boundary condition to be defined.

2: J – INTEGER. *Output*

On exit: J must be set to $-k$ if the ith boundary condition is $y_k(x_0) = s$, or to $+k$ if it is $y_k(x_1) = s$.

J must not be set to the same value k for two different values of I.

3: RHS – ***real.*** *Output*

On exit: the value s.

BC must be declared as EXTERNAL in the (sub)program from which D02JBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

4: X0 – ***real.*** *Input*

5: X1 – ***real.*** *Input*

On entry: the left- and right-hand boundaries, x_0 and x_1, repectively.

Constraint: X1 > X0.

6: K1 – INTEGER. *Input*

On entry: the number of coefficients to be returned in the Chebyshev-series representation of the components of the solution (hence the degree of the polynomial approximation is $\mathrm{K1} - 1$).

Constraint: $\mathrm{K1} \geq 2$.

7: KP – INTEGER. *Input*

On entry: the number of collocation points to be used.

Constraint: $\mathrm{KP} \geq \mathrm{K1} - 1$.

8: C(IC,N) – ***real*** array. *Output*

On exit: the computed Chebyshev coefficients of the *k*th component of the solution, y_k; that is, the computed solution is:

$$y_k = \sum_{i=1}^{\mathrm{K1}}{}' \mathrm{C}(i,k)T_{i-1}(x), \qquad 1 \le k \le n$$

where $T_i(x)$ is the *i*th Chebyshev polynomial of the first kind, and $\sum'$ denotes that the first coefficient, C(1,*k*), is halved.

9: IC – INTEGER. *Input*

On entry: the first dimension of the array C as declared in the (sub)program from which D02JBF is called.

Constraint: IC ≥ K1.

10: W(LW) – ***real*** array. *Workspace*
11: LW – INTEGER. *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D02JBF is called.

Constraint: LW ≥ 2×N×(KP+1)×(N×K1+1) + 7×N×K1.

12: IW(LIW) – INTEGER array. *Workspace*
13: LIW – INTEGER. *Input*

On entry: the dimension of the array IW as declared in the (sub)program from which D02JBF is called.

Constraint: LIW ≥ N×(K1+2).

14: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, N < 1,
or X0 ≥ X1,
or K1 < 2,
or KP < K1 – 1,
or IC < K1.

IFAIL = 2

On entry, LW < 2×N×(KP+1)×(N×K1+1) + 7×N×K1,
or LIW < N×(K1+2) (i.e. insufficient workspace).

IFAIL = 3

Either the boundary conditions are not linearly independent, (that is, in the subroutine BC the variable J is set to the same value *k* for two different values of I), or the rank of the matrix of equations for the coefficients is less than the number of unknowns. Increasing KP may overcome this latter problem.

IFAIL = 4

The least-squares routine F04AMF has failed to correct the first approximate solution (see routine document F04AMF).

7. Accuracy

The Chebyshev coefficients are determined by a stable numerical method. The accuracy of the approximate solution may be checked by varying the degree of the polynomials and the number of collocation points (see Section 8).

8. Further Comments

The time taken by the routine depends on the size and complexity of the differential system, the degree of the polynomial solution and the number of matching points.

The collocation points in the range (x_0,x_1) are chosen to be the extrema of the appropriate shifted Chebyshev polynomial. If KP = K1 − 1, then the least-squares solution reduces to the solution of a system of linear equations and true collocation results.

The accuracy of the solution may be checked by repeating the calculation with different values of K1 and with KP fixed but KP $\gg$ K1 − 1. If the Chebyshev coefficients decrease rapidly for each component (and consistently for various K1 and KP), the size of the last two or three gives an indication of the error. If the Chebyshev coefficients do not decay rapidly, it is likely that the solution cannot be well-represented by Chebyshev-series. Note that the Chebyshev coefficients are calculated for the range (−1,1).

Linear systems of high order equations in their original form, singular problems, and, indirectly, nonlinear problems can be solved using D02TGF.

9. Example

To solve the equation

$$y'' + y = 1$$

with boundary conditions

$$y(-1) = y(1) = 0$$

The equation is written as the first order system

$$\begin{pmatrix} y_1' \\ y_2' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

for solution by D02JBF and the boundary conditions are written

$$y_1(-1) = y_1(1) = 0.$$

We use K1 = 4, 6, 8 and KP = 10 and 15, so that the different Chebyshev-series may be compared. The solution for K1 = 8 and KP = 15 is evaluated by E02AKF at nine equally spaced points over interval (−1,1).

9.1. Program Text

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

```
*     D02JBF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, K1MAX, KPMAX, IC, LW, LIW
      PARAMETER        (N=2,K1MAX=8,KPMAX=15,IC=K1MAX,LW=2*N*(KPMAX+1)
     +                 *(N*K1MAX+1)+7*N*K1MAX,LIW=N*(K1MAX+2))
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             X, X0, X1
      INTEGER          I, IA1, IFAIL, J, K1, KP, M
*     .. Local Arrays ..
      real             C(IC,N), W(LW), Y(N)
      INTEGER          IW(LIW)
```

```
*     .. External Functions ..
      real             CF
      EXTERNAL         CF
*     .. External Subroutines ..
      EXTERNAL         BC, D02JBF, E02AKF
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02JBF Example Program Results'
      X0 = -1.0e0
      X1 = 1.0e0
      WRITE (NOUT,*)
      WRITE (NOUT,*) ' KP  K1   Chebyshev coefficients'
      DO 60 KP = 10, KPMAX, 5
         DO 40 K1 = 4, K1MAX, 2
            IFAIL = 1
*
            CALL D02JBF(N,CF,BC,X0,X1,K1,KP,C,IC,W,LW,IW,LIW,IFAIL)
*
            IF (IFAIL.NE.0) THEN
               WRITE (NOUT,99999) KP, K1, '  D02JBF fails with IFAIL =',
     +           IFAIL
               STOP
            ELSE
               WRITE (NOUT,99998) KP, K1, (C(I,1),I=1,K1)
               DO 20 J = 2, N
                  WRITE (NOUT,99997) (C(I,J),I=1,K1)
   20          CONTINUE
               WRITE (NOUT,*)
            END IF
   40    CONTINUE
   60 CONTINUE
      K1 = 8
      M = 9
      IA1 = 1
      WRITE (NOUT,99996) 'Last computed solution evaluated at', M,
     +  '  equally spaced points'
      WRITE (NOUT,*)
      WRITE (NOUT,99995) '      X ', (J,J=1,N)
      DO 100 I = 1, M
         X = (X0*real(M-I)+X1*real(I-1))/real(M-1)
         DO 80 J = 1, N
            IFAIL = 0
*
            CALL E02AKF(K1,X0,X1,C(1,J),IA1,IC,X,Y(J),IFAIL)
*
   80    CONTINUE
         WRITE (NOUT,99994) X, (Y(J),J=1,N)
  100 CONTINUE
      STOP
*
99999 FORMAT (1X,2(I3,1X),A,I4)
99998 FORMAT (1X,2(I3,1X),8F8.4)
99997 FORMAT (9X,8F8.4)
99996 FORMAT (1X,A,I3,A)
99995 FORMAT (1X,A,2('      Y(',I1,')'))
99994 FORMAT (1X,3F10.4)
      END
*
      real FUNCTION CF(I,J,X)
*     .. Parameters ..
      INTEGER          N
      PARAMETER        (N=2)
*     .. Scalar Arguments ..
      real             X
      INTEGER          I, J
*     .. Local Arrays ..
      real             A(N,N), R(N)
```

```
*     .. Data statements ..
      DATA              A(1,1), A(2,1), A(1,2), A(2,2)/0.0e0, -1.0e0,
     +                  1.0e0, 0.0e0/
      DATA              R(1), R(2)/0.0e0, 1.0e0/
*     .. Executable Statements ..
      IF (J.GT.0) CF = A(I,J)
      IF (J.EQ.0) CF = R(I)
      RETURN
      END
*
      SUBROUTINE BC(I,J,RHS)
*     .. Scalar Arguments ..
      real              RHS
      INTEGER           I, J
*     .. Executable Statements ..
      RHS = 0.0e0
      IF (I.GT.1) THEN
         J = -1
      ELSE
         J = 1
      END IF
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D02JBF Example Program Results

 KP  K1   Chebyshev coefficients
 10   4 -0.7798  0.0000  0.3899  0.0000
          0.0000  1.5751  0.0000 -0.0629

 10   6 -0.8326  0.0000  0.4253  0.0000 -0.0090  0.0000
          0.0000  1.6290  0.0000 -0.0724  0.0000  0.0009

 10   8 -0.8325  0.0000  0.4253  0.0000 -0.0092  0.0000  0.0001  0.0000
          0.0000  1.6289  0.0000 -0.0724  0.0000  0.0009  0.0000  0.0000

 15   4 -0.7829  0.0000  0.3914  0.0000
          0.0000  1.5778  0.0000 -0.0631

 15   6 -0.8326  0.0000  0.4253  0.0000 -0.0090  0.0000
          0.0000  1.6290  0.0000 -0.0724  0.0000  0.0009

 15   8 -0.8325  0.0000  0.4253  0.0000 -0.0092  0.0000  0.0001  0.0000
          0.0000  1.6289  0.0000 -0.0724  0.0000  0.0009  0.0000  0.0000

Last computed solution evaluated at  9  equally spaced points

       X         Y(1)       Y(2)
   -1.0000     0.0000    -1.5574
   -0.7500    -0.3542    -1.2616
   -0.5000    -0.6242    -0.8873
   -0.2500    -0.7933    -0.4579
    0.0000    -0.8508     0.0000
    0.2500    -0.7933     0.4579
    0.5000    -0.6242     0.8873
    0.7500    -0.3542     1.2616
    1.0000     0.0000     1.5574
```

D02KAF – NAG Fortran Library Routine Document

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

1. Purpose

D02KAF finds a specified eigenvalue of a regular second-order Sturm-Liouville system defined on a finite range, using a Pruefer transformation and a shooting method.

2. Specification

```
    SUBROUTINE D02KAF (XL, XR, COEFFN, BCOND, K, TOL, ELAM, DELAM,
  1                    MONIT, IFAIL)
    INTEGER          K, IFAIL
    real             XL, XR, BCOND(3,2), TOL, ELAM, DELAM
    EXTERNAL         COEFFN, MONIT
```

3. Description

D02KAF finds a specified eigenvalue $\tilde{\lambda}$ of a Sturm-Liouville system defined by a self-adjoint differential equation of the second order

$$(p(x)y')' + q(x;\lambda)y = 0, \quad a < x < b$$

together with boundary conditions of the form

$$a_2y(a) = a_1p(a)y'(a)$$
$$b_2y(b) = b_1p(b)y'(b)$$

at the two, finite, endpoints a and b. The functions p and q, which are real-valued, are defined by a subroutine COEFFN supplied by the user.

For the theoretical basis of the numerical method to be valid, the following conditions should hold on the coefficient functions:

(a) $p(x)$ must be non-zero and of constant sign throughout the closed interval $[a,b]$;

(b) $\frac{\partial q}{\partial \lambda}$ must be of constant sign and non-zero throughout the open interval (a,b) and for all relevant values of λ, and must not be identically zero as x varies, for any relevant value λ;

(c) p and q should (as functions of x) have continuous derivatives, preferably up to the fourth order, on $[a,b]$. The differential equation code used will integrate through mild discontinuities, but probably with severely reduced efficiency. Therefore, if p and q violate this condition, D02KDF should be used.

The eigenvalue $\tilde{\lambda}$ is determined by a shooting method based on a Pruefer transformation of the differential equations. Providing certain assumptions are met, the computed value of $\tilde{\lambda}$ will be correct to within a mixed absolute/relative error specified by the user-supplied value TOL. D02KAF is a driver routine for the more complicated routine D02KDF whose specification provides more details of the techniques used.

A good account of the Sturm-Liouville systems, with some description of Pruefer transformations, is given in Birkhoff [1], Chapter X. The best introduction to the use of Pruefer transformations for the numerical solution of eigenvalue problems arising from physics and chemistry is given in Bailey [2].

4. References

[1] BIRKHOFF, G. and ROTA, G.C.
Ordinary Differential Equations.
Ginn & Co., Boston and New York, 1962.

[2] BAILEY, P.B.
Sturm-Liouville Eigenvalue via a Phase Function.
SIAM J. Appl. Math., 14, pp. 242-249, 1966.

5. Parameters

1: XL – ***real.*** *Input*

2: XR – ***real.*** *Input*

On entry: the left-hand and right-hand endpoints a and b respectively, of the interval of definition of the problem.

Constraint: XL < XR.

3: COEFFN – SUBROUTINE, supplied by the user. *External Procedure*

COEFFN must compute the values of the coefficient functions $p(x)$ and $q(x;\lambda)$ for given values of x and λ. Section 3 states conditions which p and q must satisfy.

Its specification is:

```
SUBROUTINE COEFFN(P, Q, DQDL, X, ELAM, JINT)
INTEGER    JINT
real       P, Q, DQDL, X, ELAM
```

1: P – ***real.*** *Output*

On exit: the value of $p(x)$ for the current value of x.

2: Q – ***real.*** *Output*

On exit: the value of $q(x;\lambda)$ for the current value of x and the current trial value of λ.

3: DQDL – ***real.*** *Output*

On exit: the value of $\frac{\partial q}{\partial \lambda}(x;\lambda)$ for the current value of x and the current trial value of λ. However DQDL is only used in error estimation and, in the rare cases where it may be difficult to evaluate, an approximation (say to within 20 percent) will suffice.

4: X – ***real.*** *Input*

On entry: the current value of x.

5: ELAM – ***real.*** *Input*

On entry: the current trial value of the eigenvalue parameter λ.

6: JINT – INTEGER. *Input*

This parameter is included for compatibility with the more complex routine D02KDF (which is called by D02KAF).

COEFFN must be declared as EXTERNAL in the (sub)program from which D02KAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

4: BCOND(3,2) – ***real*** array. *Input/Output*

On entry: BCOND(1,1) and BCOND (2,1) must contain the numbers a_1, a_2 specifying the left-hand boundary condition in the form

$$a_2y(a) = a_1p(a)y'(a)$$

where $|a_2| + |a_1p(a)| \neq 0$

while BCOND(1,2) and BCOND(2,2) must contain b_1, b_2 such that

$$b_2y(b) = b_1p(b)y'(b)$$

where $|b_2| + |b_1p(b)| \neq 0$.

Note the occurrence of $p(a), p(b)$ in these formulae.

On exit: BCOND(3,1) and BCOND(3,2) hold values σ_l, σ_r estimating the sensitivity of the computed eigenvalue to changes in the boundary conditions. These values should only be of interest if the boundary conditions are, in some sense, an approximation to some 'true' boundary conditions. For example, if the range [XL, XR] should really be [0, ∞] but instead XR has been given a large value and the boundary conditions at infinity applied at XR, then the sensitivity parameter σ_r may be of interest. Refer to the specification of D02KDF, Section 8.5, for the actual meaning of σ_r and σ_l.

5: K – INTEGER. *Input*

On entry: the index, k, of the desired eigenvalue when these are ordered:

$\lambda_0 < \lambda_1 < ... < \lambda_k ...$

Constraint: K ≥ 0.

6: TOL – ***real***. *Input*

On entry: the tolerance parameter which determines the accuracy of the computed eigenvalue. The error estimate held in DELAM on exit will satisfy the 'mixed absolute/relative error test'

$$\text{DELAM} \leq \text{TOL}\times\max(1.0, |\text{ELAM}|), \qquad (*)$$

where ELAM has its exit value; DELAM will usually be somewhat smaller than the right-hand side of (*) but not several orders of magnitude smaller.

Constraint: TOL > 0.0.

7: ELAM – ***real***. *Input/Output*

On entry: an initial estimate of the eigenvalue.

On exit: the final computed estimate, whether or not an error occurred.

8: DELAM – ***real***. *Input/Output*

On entry: an indication of the scale of the problem in the λ-direction. DELAM holds the initial 'search step' (positive or negative). Its value is not critical, but the first two trial evaluations are made at ELAM and ELAM + DELAM, so the routine will work most efficiently if the eigenvalue lies between these values. A reasonable choice (if a closer bound is not known) is about half the distance between adjacent eigenvalues in the neighbourhood of the one sought. Often there will be a problem, similar to the one in hand but with known eigenvalues, which will help one to choose initial values for ELAM and DELAM.

If DELAM = 0.0 on entry, it is given the default value of 0.25×max(1.0, |ELAM|).

On exit: if IFAIL = 0, DELAM holds an estimate of the absolute error in the computed eigenvalue, that is $|\tilde{\lambda} - \text{ELAM}| \simeq \text{DELAM}$, where $\tilde{\lambda}$ is the true eigenvalue.

With IFAIL ≠ 0, DELAM may hold an estimate of the error, or its initial value, depending on the value of IFAIL. See Section 6 for further details.

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

MONIT is called by D02KAF at the end of each iteration for λ and allows the user to monitor the course of the computation by printing out the parameters (see Section 9 for an example). **The parameters must not be altered by the routine.**

If no monitoring is required, the dummy subroutine D02KAY may be used. (D02KAY is included in the NAG Fortran Library. In some implementations of the Library the name is changed to KAYD02: refer to the Users' Note for your implementation.)

Its specification is:

```
SUBROUTINE MONIT(NIT, IFLAG, ELAM, FINFO)
INTEGER    NIT, IFLAG
real       ELAM, FINFO(15)
```

1: NIT – INTEGER. *Input*

On entry: 15 minus the number of iterations used so far in the search for $\tilde{\lambda}$. (Up to 15 iterations are permitted.)

2: IFLAG – INTEGER. *Input*

On entry: IFLAG describes what phase the computation is in, as follows:

IFLAG < 0

an error occurred during the computation at this iteration; an error exit from D02KAF will follow.

IFLAG = 1

the routine is trying to bracket the eigenvalue $\tilde{\lambda}$.

IFLAG = 2

the routine is converging to the eigenvalue $\tilde{\lambda}$ (having already bracketed it).

Normally, the iteration will terminate after a sequence of iterates with IFLAG = 2, but occasionally the bracket on $\tilde{\lambda}$ thus determined will not be sufficiently small and the iteration will be repeated with tighter accuracy control.

3: ELAM – ***real***. *Input*

On entry: the current trial value of $\tilde{\lambda}$.

4: FINFO(15) – ***real*** array. *Input*

On entry: information about the behaviour of the shooting method, and diagnostic information in the case of errors. It should not normally be printed in full if no error has occurred (that is, if IFLAG ≥ 0), though the first few components may be of interest to the user. In case of an error (IFLAG < 0) all the components of FINFO should be printed. The contents of FINFO are as follows:

FINFO(1): the current value of the 'miss-distance' or 'residual' function $f(\lambda)$ on which the shooting method is based. $f(\tilde{\lambda}) = 0$ in theory. This is set to zero if IFLAG < 0.

FINFO(2): an estimate of the quantity $\delta\lambda$ defined as follows. Consider the perturbation in the miss-distance $f(\lambda)$ that would result if the local error in the solution of the differential equation were always positive and equal to its maximum permitted value. Then $\delta\lambda$ is the perturbation in λ that would have the same effect on $f(\lambda)$. Thus, at the zero of $f(\lambda)$, $|\delta\lambda|$ is an approximate bound on the perturbation of the zero (that is the eigenvalue) caused by errors in numerical solution. If $\delta\lambda$ is very large then it is possible there has been a programming error in COEFFN such that q is independent of λ. If this is the case, an error exit with IFAIL = 5 should follow. FINFO(2) is set to zero if IFLAG < 0.

FINFO(3): the number of internal iterations, using the same value of λ and tighter accuracy tolerances, needed to bring the accuracy (that is, the value of $\delta\lambda$) to an acceptable value. Its value should normally be 1.0, and should almost never exceed 2.0.

FINFO(4): the number of calls to COEFFN at this iteration.

FINFO(5): the number of successful steps taken by the internal differential equation solver at this iteration.

FINFO(6): the number of unsuccessful steps used by the internal integrator at this iteration.

FINFO(7): the number of successful steps at the maximum step size taken by the internal integrator at this iteration.

FINFO(8): not used.

FINFO(9) to FINFO(15): set to zero, unless IFLAG < 0 in which case they hold the following values describing the point of failure:

FINFO(9): 1 or 2 depending on whether integration was in a forward or backward direction at the time of failure.

FINFO(10): the value of the independent variable, x, the point at which error occurred.

FINFO(11), FINFO(12), FINFO(13): the current values of the Pruefer dependent variables β, ϕ and ρ respectively. See Section 3 of routine document D02KEF for a description of these variables.

FINFO(14): the local-error tolerance being used by the internal integrator at the point of failure.

FINFO(15): the last integration meshpoint.

MONIT must be declared as EXTERNAL in the (sub)program from which D02KAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

10: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, K < 0,
or TOL ≤ 0.

IFAIL = 2

On entry, $a_1 = p(a)a_2 = 0$,
or $b_1 = p(b)b_2 = 0$,

(the array BCOND has been set up incorrectly).

IFAIL = 3

At some point between XL and XR the value of $p(x)$ computed by COEFFN became zero or changed sign. See the last call of MONIT for details.

IFAIL = 4

After 15 iterations the eigenvalue had not been found to the required accuracy.

IFAIL = 5

The bracketing phase (with parameter IFLAG of MONIT equal to 1) failed to bracket the eigenvalue within ten iterations. This is caused by an error in formulating the problem (for example, q is independent of λ), or by very poor initial estimates of ELAM, DELAM.

On exit ELAM and ELAM + DELAM give the endpoints of the interval within which no eigenvalue was located by the routine.

IFAIL = 6

To obtain the desired accuracy the local error tolerance was set so small at the start of some subinterval that the differential equation solver could not choose an initial stepsize large enough to make significant progress. See the last call of MONIT for diagnostics.

IFAIL = 7

At some point the stepsize in the differential equation solver was reduced to a value too small to make significant progress (for the same reasons as with IFAIL = 6). This could be due to pathological behaviour of $p(x)$ and $q(x;\lambda)$ or to an unreasonable accuracy requirement or to the current value of λ making the equation 'stiff'. See the last call of MONIT for details.

IFAIL = 8

TOL is too small for the problem being solved and the machine-precision being used. The local value of ELAM should be a very good approximation to the eigenvalue λ.

IFAIL = 9

C05AZF, called by D02KAF, has terminated with the error exit corresponding to a pole of the matching function. This error exit should not occur, but if it does, try solving the problem again with a smaller value for TOL.

Note: error exits with IFAIL = 2, 3, 6, 7 and 9 are caused by the inability to set up or solve the differential equation at some iteration and will be immediately preceded by a call of MONIT, giving diagnostic information.

IFAIL = 10 (D02KDY)
IFAIL = 11 (C05AZF)
IFAIL = 12 (D02KDF)

A serious error has occurred in the specified routine. Check all subroutine calls and array dimensions. Seek expert help.

7. Accuracy

The absolute accuracy of the computed eigenvalue is usually within a mixed absolute/relative bound defined by TOL (as defined above).

8. Further Comments

The time taken by the routine depends on the complexity of the coefficient functions, whether they or their derivatives are rapidly changing, the tolerance demanded, and how many iterations are needed to obtain convergence. The amount of work per iteration is roughly doubled when TOL is divided by 16. To make the most economical use of the routine, one should try to obtain good initial values for ELAM and DELAM.

See Section 8 of the specification of D02KDF for a discussion of the technique used.

9. Example

To find the fourth eigenvalue of Mathieu's equations

$$y'' + (\lambda-2q\cos 2x)y = 0$$

with boundary conditions

$$y'(0) = y'(\pi) = 0$$

and $q = 5$. We use a starting value ELAM = 15.0 and a step DELAM = 4.0. We illustrate the effect of varying TOL by choosing TOL = 1.0E−5 and 1.0E−6 (note the change in the output value of the error estimate DELAM). The value of π is calculated using X01AAF.

9.1. Program Text

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

```
*     D02KAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      INTEGER          QQ
*     .. Local Scalars ..
      real             DELAM, ELAM, PI, TOL, XL, XR
      INTEGER          I, IFAIL, K
*     .. Local Arrays ..
      real             BCOND(3,2)
*     .. External Functions ..
      real             X01AAF
      EXTERNAL         X01AAF
*     .. External Subroutines ..
      EXTERNAL         COEFFN, D02KAF, D02KAY
*     .. Common blocks ..
      COMMON           QQ
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02KAF Example Program Results'
      PI = X01AAF(DELAM)
      XL = 0
      XR = PI
      BCOND(1,1) = 1.0e0
      BCOND(2,1) = 0.0e0
      BCOND(1,2) = 1.0e0
      BCOND(2,2) = 0.0e0
      K = 4
      QQ = 5
      DO 20 I = 5, 6
         TOL = 10.0e0**(-I)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Calculation with TOL =', TOL
         ELAM = 15.0e0
         DELAM = 4.0e0
         IFAIL = 1
*
*        * To obtain monitoring information from the supplied
*        subroutine MONIT replace the name D02KAY by MONIT in
*        the next statement, and declare MONIT as external *
*
         CALL D02KAF(XL,XR,COEFFN,BCOND,K,TOL,ELAM,DELAM,D02KAY,IFAIL)
*
         WRITE (NOUT,*)
         IF (IFAIL.NE.0) THEN
            WRITE (NOUT,99996) ' D02KAF fails. IFAIL =', IFAIL
         ELSE
            WRITE (NOUT,*) ' Final results'
            WRITE (NOUT,*)
            WRITE (NOUT,99998) ' K =', K, '  QQ =', QQ, '  ELAM =',
     +        ELAM, '    DELAM =', DELAM
            WRITE (NOUT,99997) ' BCOND(3,1) =', BCOND(3,1),
     +        '    BCOND(3,2) =', BCOND(3,2)
            WRITE (NOUT,*)
         END IF
   20 CONTINUE
      STOP
*
```

```
99999 FORMAT (1X,A,e16.4)
99998 FORMAT (1X,A,I3,A,I3,A,F12.3,A,e12.2)
99997 FORMAT (1X,A,e12.4,A,e12.4)
99996 FORMAT (1X,A,I3)
      END
*
      SUBROUTINE COEFFN(P,Q,DQDL,X,ELAM,JINT)
*     .. Scalar Arguments ..
      real              DQDL, ELAM, P, Q, X
      INTEGER           JINT
*     .. Scalars in Common ..
      INTEGER           QQ
*     .. Intrinsic Functions ..
      INTRINSIC         COS, real
*     .. Common blocks ..
      COMMON            QQ
*     .. Executable Statements ..
      P = 1.0e0
      DQDL = 1.0e0
      Q = ELAM - 2.0e0*real(QQ)*COS(2.0e0*X)
      RETURN
      END
*
      SUBROUTINE MONIT(NIT,IFLAG,ELAM,FINFO)
*     .. Parameters ..
      INTEGER           NOUT
      PARAMETER         (NOUT=6)
*     .. Scalar Arguments ..
      real              ELAM
      INTEGER           IFLAG, NIT
*     .. Array Arguments ..
      real              FINFO(15)
*     .. Local Scalars ..
      INTEGER           I
*     .. Executable Statements ..
      IF (NIT.EQ.14) THEN
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Output from MONIT'
      END IF
      WRITE (NOUT,99999) NIT, IFLAG, ELAM, (FINFO(I),I=1,4)
      RETURN
*
99999 FORMAT (1X,2I4,F10.3,2e12.2,2F8.1)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D02KAF Example Program Results

Calculation with TOL =      0.1000E-04

 Final results

 K =  4  QQ =  5  ELAM =      17.097   DELAM =    0.11E-03
 BCOND(3,1) = -0.9064E+00    BCOND(3,2) =  0.9064E+00


Calculation with TOL =      0.1000E-05

 Final results

 K =  4  QQ =  5  ELAM =      17.097   DELAM =    0.11E-04
 BCOND(3,1) = -0.9075E+00    BCOND(3,2) =  0.9075E+00
```

With MONIT used instead of D02KAY as an argument of D02KAF in the example program, intermediate results similar to those below are obtained when TOL = 0.1E–4:

```
Output from MONIT
  14   1    15.000   -0.32E+00   -0.11E-03    1.0   206.0
  13   1    15.000   -0.32E+00   -0.57E-04    2.0   234.0
  12   1    19.000    0.26E+00   -0.67E-04    1.0   226.0
  11   2    17.225    0.18E-01   -0.68E-04    1.0   226.0
  10   2    17.089   -0.10E-02   -0.64E-04    1.0   226.0
   9   2    17.097    0.42E-05   -0.64E-04    1.0   226.0
   8   2    17.097   -0.19E-05   -0.64E-04    1.0   226.0
```

D02KDF – NAG Fortran Library Routine Document

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

1. Purpose

D02KDF finds a specified eigenvalue of a regular or singular second-order Sturm-Liouville system on a finite or infinite interval, using a Pruefer transformation and a shooting method. Provision is made for discontinuities in the coefficient functions or their derivatives.

2. Specification

```
      SUBROUTINE D02KDF (XPOINT, M, COEFFN, BDYVAL, K, TOL, ELAM, DELAM,
     1                   HMAX, MAXIT, MAXFUN, MONIT, IFAIL)
      INTEGER          M, K, MAXIT, MAXFUN, IFAIL
      real             XPOINT(M), TOL, ELAM, DELAM, HMAX(2,M)
      EXTERNAL         COEFFN, BDYVAL, MONIT
```

3. Description

D02KDF finds a specified eigenvalue $\tilde{\lambda}$ of a Sturm-Liouville system defined by a self-adjoint differential equation of the second-order

$$(p(x)y')' + q(x;\lambda)y = 0, \qquad a < x < b,$$

together with appropriate boundary conditions at the two, finite or infinite, end-points a and b. The functions p and q, which are real-valued, must be defined by a subroutine COEFFN. The boundary conditions must be defined by a subroutine BDYVAL, and in the case of a singularity at a or b take the form of an asymptotic formula for the solution near the relevant end-point.

For the theoretical basis of the numerical method to be valid, the following conditions should hold on the coefficient functions:

(a) $p(x)$ must be non-zero and of one sign throughout the interval (a,b); and

(b) $\frac{\partial q}{\partial \lambda}$ must be of one sign throughout (a,b) for all relevant values of λ, and must not be identically zero as x varies for any λ.

Points of discontinuity in the functions p and q or their derivatives are allowed, and should be included as 'break-points' in the array XPOINT.

The eigenvalue $\tilde{\lambda}$ is determined by a shooting method based on the Scaled Pruefer form of the differential equation as described in Pryce [5], with certain modifications. The Pruefer equations are integrated by a special internal routine using Merson's Runge-Kutta formula with automatic control of local error. Providing certain assumptions (see Section 8.1) are met, the computed value of $\tilde{\lambda}$ will have a mixed absolute/relative error, estimated by the user-supplied value TOL.

A good account of the theory of Sturm-Liouville systems, with some description of Pruefer transformations, is given in Birkhoff and Rota [4], Chapter X. An introduction to the user of Pruefer transformations for the numerical solution of eigenvalue problems arising from physics and chemistry is in Bailey [2].

The scaled Pruefer method is fairly recent, and is described in a short note in Pryce [6] and in some detail in the technical report (Pryce [5]).

4. References

[1] ABRAMOWITZ, M. and STEGUN, I.A.
Handbook of Mathematical Functions.
Dover Publications, Ch. 4.4, p. 79, 1968.

[2] BAILEY, P.B.
Sturm-Liouville Eigenvalues via a Phase Function.
SIAM J. Appl. Math. 4, pp. 242-249, 1966.

[3] BANKS, D.O. and KUROWSKI, I.
Computation of Eigenvalues of Singular Sturm-Liouville Systems.
Math. Comput. 22, pp. 304-310, 1968.

[4] BIRKHOFF, G. and ROTA, G.C.
Ordinary Differential Equations.
Ginn & Co., Boston and New York, 1962.

[5] PRYCE J.D.
Two codes for Sturm-Liouville problems.
Bristol University, Computer Science Technical Report, CS-81-01, 1981.

[6] PRYCE, J.D. and HARGRAVE, B.A.
The Scaled Prüfer Method for one-parameter and multi-parameter eigenvalue problems in ODEs.
Inst. Math. Appl., Numerical Analysis Newsletter, 1, No. 3, 1977.

5. Parameters

1: XPOINT(M) – ***real*** array. *Input*

On entry: the points where the boundary conditions computed by BDYVAL are to be imposed, and also any break-points, i.e. XPOINT(1) to XPOINT(m) must contain values $x_1,...,x_m$ such that

$$x_1 \leq x_2 < x_3 < ... < x_{m-1} \leq x_m$$

with the following meanings:

(a) x_1 and x_m are left and right end-points, a and b, of the domain of definition of the Sturm-Liouville system if these are finite. If either of a or b is infinite, the corresponding value x_1 or x_m may be a more-or-less arbitrary 'large' number of appropriate sign.

(b) x_2 and x_{m-1} are the Boundary Matching Points (BMP's), that is the points at which the left and right boundary conditions computed in BDYVAL are imposed.

If the left-hand end-point is a regular point then the user should set $x_2 = x_1$ $(= a)$, while if it is a singular point the user must set $x_2 > x_1$. Similarly $x_{m-1} = x_m$ $(= b)$ if the right-hand end-point is regular, and $x_{m-1} < x_m$ if it is singular.

(c) The remaining $m-4$ points $x_3,...,x_{m-2}$, if any, define 'break-points' which divide the interval $[x_2,x_{m-1}]$ into $m-3$ sub-intervals

$$i_1 = [x_2,x_3], \; ..., \; i_{m-3} = [x_{m-2},x_{m-1}].$$

Numerical integration of the differential equation is stopped and restarted at each break-point. In simple cases no break-points are needed. However if $p(x)$ or $q(x;\lambda)$ are given by different formulae in different parts of the interval, then integration is more efficient if the range is broken up by break-points in the appropriate way. Similarly points where any jumps occur in $p(x)$ or $q(x;\lambda)$, or in their derivatives up to the fifth order, should appear as break-points.

Examples are given in Sections 8 and 9. XPOINT determines the position of the Shooting Matching Point (SMP), as explained in Section 8.3.

Constraint: X(1) ≤ X(2) < ... < X(M−1) ≤ X(M).

2: M – INTEGER. *Input*

On entry: the number of points in the array XPOINT.

Constraint: M ≥ 4.

3: COEFFN – SUBROUTINE, supplied by the user. *External Procedure*

COEFFN must compute the values of the coefficient functions $p(x)$ and $q(x;\lambda)$ for given values of x and λ. Section 3 states conditions which p and q must satisfy.

Its specification is:

```
SUBROUTINE  COEFFN(P, Q, DQDL, X, ELAM, JINT)
real        P, Q, DQDL, X, ELAM
INTEGER     JINT
```

1: P – ***real***. *Output*

On exit: the value of $p(x)$ for the current value of x.

2: Q – ***real***. *Output*

On exit: the value of $q(x;\lambda)$ for the current value of x and the current trial value of λ.

3: DQDL – ***real***. *Output*

On exit: the value of $\dfrac{\partial q}{\partial \lambda}$ for the current value of x and the current trial value of λ. However DQDL is only used in error estimation and an approximation (say to within 20 per cent) will suffice.

4: X – ***real***. *Input*

On entry: the current value of x.

5: ELAM – ***real***. *Input*

On entry: the current trial value of the eigenvalue parameter λ.

6: JINT – INTEGER. *Input*

On entry: the index j of the sub-interval i_j (see specification of XPOINT) in which x lies.

See Sections 8.4 and 9 for examples. COEFFN must be declared as EXTERNAL in the (sub)program from which D02KDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

4: BDYVAL – SUBROUTINE, supplied by the user. *External Procedure*

BDYVAL must define the boundary conditions. For each end-point, BDYVAL must return (in YL or YR) values of $y(x)$ and $p(x)y'(x)$ which are consistent with the boundary conditions at the end-points; only the ratio of the values matters. Here x is a given point (XL or XR) equal to, or close to, the end-point.

For a **regular** end-point (a, say), $x = a$, a boundary condition of the form

$$c_1y(a) + c_2y'(a) = 0$$

can be handled by returning constant values in YL, e.g. $\mathrm{YL}(1) = c_2$ and $\mathrm{YL}(2) = -c_1p(a)$.

For a **singular** end-point however, YL(1) and YL(2) will in general be functions of XL and ELAM, and YR(1) and YR(2) functions of XR and ELAM, usually derived analytically from a power-series or asymptotic expansion. Examples are given in Sections 8.5 and 9.

Its specification is:

```
SUBROUTINE  BDYVAL(XL, XR, ELAM, YL, YR)
real        XL, XR, ELAM, YL(3), YR(3)
```

1: XL – ***real***. *Input*

On entry: if a is a regular end-point of the system (so that $a = x_1 = x_2$), then XL contains a. If a is a singular point (so that $a \le x_1 < x_2$), then XL contains a point x such that $x_1 < x \le x_2$.

2: XR – *real.* *Input*

On entry: if b is a regular end-point of the system (so that $x_{m-1} = x_m = b$), then XR contains b. If b is a singular point (so that $x_{m-1} < x_m \leq b$), then XR contains a point x such that $x_{m-1} \leq x < x_m$.

3: ELAM – *real.* *Input*

On entry: the current trial value of λ.

4: YL(3) – *real* array. *Output*

On exit: YL(1) and YL(2) should contain values of $y(x)$ and $p(x)y'(x)$ respectively (not both zero) which are consistent with the boundary condition at the left-hand end-point, given by x = XL. YL(3) should not be set.

5: YR(3) – *real* array. *Output*

On exit: YR(1) and YR(2) should contain values of $y(x)$ and $p(x)y'(x)$ respectively (not both zero) which are consistent with the boundary condition at the right-hand end-point, given by x = XR. YR(3) should not be set.

BDYVAL must be declared as EXTERNAL in the (sub)program from which D02KDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: K – INTEGER. *Input*

On entry: the index k of the required eigenvalue when the eigenvalues are ordered $\lambda_0 < \lambda_1 < \lambda_2 < ... < \lambda_k < ...$

Constraint: K $\geq$ 0.

6: TOL – *real.* *Input*

On entry: the tolerance parameter which determines the accuracy of the computed eigenvalue. The error estimate held in DELAM on exit satisfies the mixed absolute/relative error test

DELAM $\leq$ TOL×max(1.0,|ELAM|) (*)

where ELAM is the final estimate of the eigenvalue. DELAM is usually somewhat smaller than the right-hand side of (*) but not several orders of magnitude smaller.

Constraint: TOL > 0.0.

7: ELAM – *real.* *Input/Output*

On entry: an initial estimate of the eigenvalue $\tilde{\lambda}$.

On exit: the final computed estimate, whether or not an error occurred.

8: DELAM – *real.* *Input/Output*

On entry: an indication of the scale of the problem in the λ-direction. DELAM holds the initial 'search step' (positive or negative). Its value is not critical but the first two trial evaluations are made at ELAM and ELAM + DELAM, so the routine will work most efficiently if the eigenvalue lies between these values. A reasonable choice (if a closer bound is not known) is half the distance between adjacent eigenvalues in the neighbourhood of the one sought. In practice, there will often be a problem, similar to the one in hand but with known eigenvalues, which will help one to choose initial values for ELAM and DELAM.

If DELAM = 0.0 on entry, it is given the default value of 0.25×max(1.0,|ELAM|).

On exit: with IFAIL = 0, DELAM holds an estimate of the absolute error in the computed eigenvalue, that is $|\tilde{\lambda}$ – ELAM| ≈ DELAM (In Section 8.2 we discuss the assumptions under which this is true.) The true error is rarely more than twice, or less than a tenth, of the estimated error.

With IFAIL ≠ 0. DELAM may hold an estimate of the error, or its initial value, depending on the value of IFAIL. See Section 6 for further details.

9: HMAX(2,M) – ***real*** array. *Input/Output*

On entry: HMAX(1,j) should contain a maximum step size to be used by the differential equation code in the jth sub-interval i_j (as described in the specification of parameter XPOINT) for $j = 1,2,...,m-3$. If it is zero the routine generates a maximum step size internally.

It is recommended that HMAX(1,j) be set to zero unless the coefficient functions p and q have features (such as a narrow peak) within the jth sub-interval that could be 'missed' if a long step were taken. In such a case HMAX(1,j) should be set to about half the distance over which the feature should be observed. Too small a value will increase the computing time for the routine. See Section 8 for further suggestions.

The rest of the array is used as workspace.

On exit: HMAX(1,$m-1$) and HMAX(1,m) contain the sensitivity coefficients σ_l, σ_r, described in Section 8.6. Other entries also contain diagnostic output in case of an error exit (see Section 6 for details).

10: MAXIT – INTEGER. *Input/Output*

On entry: a bound on n_r, the number of rootfinding iterations allowed, that is the number of trial values of λ that are used; if MAXIT ≤ 0, no such bound is assumed.

Suggested value: MAXIT = 0. (See also under MAXFUN).

On exit: MAXIT will have been decreased by the number of iterations actually performed, whether or not it was positive on entry.

11: MAXFUN – INTEGER. *Input*

On entry: a bound on n_f, the number of calls to COEFFN made in any one rootfinding iteration. If MAXFUN ≤ 0, no such bound is assumed.

Suggested value: MAXFUN = 0.

MAXFUN and MAXIT may be used to limit the computational cost of a call to D02KDF, which is roughly proportional to $n_r \times n_f$.

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

MONIT is called by D02KDF at the end of each rootfinding iteration and allows the user to monitor the course of the computation by printing out the parameters (see Section 9 for an example).

If no monitoring is required, the dummy subroutine D02KAY may be used. (D02KAY is included in the NAG Fortran Library. In some implementations of the Library the name is changed to KAYD02: refer to the Users' Note for your implementation.)

Its specification is:

```
SUBROUTINE MONIT(MAXIT, IFLAG, ELAM, FINFO)
INTEGER    MAXIT, IFLAG
real       ELAM, FINFO(15)
```

1: MAXIT – INTEGER. *Input*

On entry: the current value of the parameter MAXIT of D02KDF, which is decreased by one at each iteration.

2: IFLAG – INTEGER. *Input*

On entry: IFLAG describes what phase the computation is in, as follows:

IFLAG < 0

an error occurred in the computation of the 'miss-distance' at this iteration; an error exit from D02KDF with IFAIL = –IFLAG will follow.

IFLAG = 1

the routine is trying to bracket the eigenvalue $\tilde{\lambda}$.

IFLAG = 2

the routine is converging to the eigenvalue $\tilde{\lambda}$ (having already bracketed it).

3: ELAM – ***real***. *Input*

On entry: the current trial value of λ.

4: FINFO(15) – ***real*** array. *Input*

On entry: information about the behaviour of the shooting method, and diagnostic information in the case of errors. It should not normally be printed in full if no error has occurred (that is, if IFLAG > 0), though the first few components may be of interest to the user. In case of an error (IFLAG < 0) all the components of FINFO should be printed. The contents of FINFO are as follows:

FINFO(1): the current value of the 'miss-distance' or 'residual' function $f(\lambda)$ on which the shooting method is based. FINFO(1) is set to zero if IFLAG < 0.

FINFO(2): an estimate of the quantity $\delta\lambda$ defined as follows. Consider the perturbation in the miss-distance $f(\lambda)$ that would result if the local error, in the solution of the differential equation, were always positive and equal to its maximum permitted value. Then $\delta\lambda$ is the perturbation in λ that would have the same effect on $f(\lambda)$. Thus, at the zero of $f(\lambda)$, $|\delta\lambda|$ is an approximate bound on the perturbation of the zero (that is the eigenvalue) caused by errors in numerical solution. If $\delta\lambda$ is very large then it is possible that there has been a programming error in COEFFN such that q is independent of λ. If this is the case, an error exit with IFAIL = 5 should follow. FINFO(2) is set to zero if IFLAG < 0.

FINFO(3): the number of internal iterations, using the same value of λ and tighter accuracy tolerances, needed to bring the accuracy (that is the value of $\delta\lambda$) to an acceptable value. Its value should normally be 1.0, and should almost never exceed 2.0.

FINFO(4): the number of calls to COEFFN at this iteration.

FINFO(5): the number of successful steps taken by the internal differential equation solver at this iteration.

FINFO(6): the number of unsuccessful steps used by the internal integrator at this iteration.

FINFO(7): the number of successful steps at the maximum step size taken by the internal integrator at this iteration.

FINFO(8): is not used.

FINFO(9) to FINFO(15): set to zero, unless IFLAG < 0 in which case they hold the following values describing the point of failure:

FINFO(9): the index of the sub-interval where failure occurred, in the range 1 to $m - 3$. In case of an error in BDYVAL, it is set to 0 or $m - 2$ depending on whether the left or right boundary condition caused the error.

FINFO(10): the value of the independent variable x, the point at which the error occurred. In case of an error in BDYVAL, it is set to the value of XL or XR as appropriate (see the specification of BDYVAL).

FINFO(11), FINFO(12), FINFO(13): the current value of the Pruefer dependent variables β, ϕ and ρ respectively. These are set to zero in case of an error in BDYVAL. (See Section 3 of routine document D02KEF for a description of these variables).

FINFO(14): the local-error tolerance being used by the internal integrator at the point of failure. This is set to zero in the case of an error in BDYVAL.

FINFO(15): the last integration mesh point. This is set to zero in the case of an error in BDYVAL.

MONIT must be declared as EXTERNAL in the (sub)program from which D02KDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

13: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

A parameter error. All parameters (except IFAIL) are left unchanged. The reason for the error is shown by the value of HMAX(2,1) as follows:

HMAX(2,1) = 1: M < 4;
HMAX(2,1) = 2: K < 0;
HMAX(2,1) = 3: TOL ≤ 0.0;
HMAX(2,1) = 4: XPOINT(1) to XPOINT(m) are not in ascending order. HMAX(2,2) gives the position i in XPOINT where this was detected.

IFAIL = 2

At some call to BDYVAL, invalid values were returned, that is, either YL(1) = YL(2) = 0.0, or YR(1) = YR(2) = 0.0 (a programming error in BDYVAL). See the last call of MONIT for details.

This error exit will also occur if $p(x)$ is zero at the point where the boundary condition is imposed. Probably BDYVAL was called with XL equal to a singular end-point a or with XR equal to a singular end-point b.

IFAIL = 3

At some point between XL and XR the value of $p(x)$ computed by COEFFN became zero or changed sign. See the last call of MONIT for details.

IFAIL = 4

MAXIT > 0 on entry, and after MAXIT iterations the eigenvalue had not been found to the required accuracy.

IFAIL = 5

The 'bracketing' phase (with parameter IFLAG of MONIT equal to 1) failed to bracket the eigenvalue within ten iterations. This is caused by an error in formulating the problem (for example, q is independent of λ), or by very poor initial estimates of ELAM, DELAM.

On exit ELAM and ELAM + DELAM give the end-points of the interval within which no eigenvalue was located by the routine.

IFAIL = 6

MAXFUN > 0 on entry, and the last iteration was terminated because more than MAXFUN calls to COEFFN were used. See the last call of MONIT for details.

IFAIL = 7

To obtain the desired accuracy the local error tolerance was set so small at the start of some sub-interval that the differential equation solver could not choose an initial step size large enough to make significant progress. See the last call of MONIT for diagnostics.

IFAIL = 8

At some point inside a sub-interval the step size in the diffential equation solver was reduced to a value too small to make significant progress (for the same reasons as with IFAIL = 7). This could be due to pathological behaviour of $p(x)$ and $q(x;\lambda)$ or to an unreasonable accuracy requirement or to the current value of λ making the equations 'stiff'. See the last call of MONIT for details.

IFAIL = 9

TOL is too small for the problem being solved and the ***machine precision*** being used. The final value of ELAM should be a very good approximation to the eigenvalue.

IFAIL = 10

C05AZF, called by D02KDF, has terminated with the error exit corresponding to a pole of the residual function $f(\lambda)$. This error exit should not occur, but if it does, try solving the problem again with a smaller TOL.

IFAIL = 11 (D02KDY)
IFAIL = 12 (C05AZF)

A serious error has occurred in the specified routine. Check all subroutine calls and array dimensions. Seek expert help.

HMAX(2,1) holds the failure exit number from the routine where the failure occurred. In the case of a failure in C05AZF, HMAX(2,2) holds the value of parameter IND of C05AZF.

Note: error exits with IFAIL = 2,3,6,7,8,11 are caused by being unable to set up or solve the differential equation at some iteration, and will be immediately preceded by a call of MONIT giving diagnostic information. For other errors, diagnostic information is contained in HMAX$(2,j)$, for $j = 1,2,...,m$, where appropriate.

7. Accuracy

See the discussion in Section 8.2.

8. Further Comments

8.1. Timing

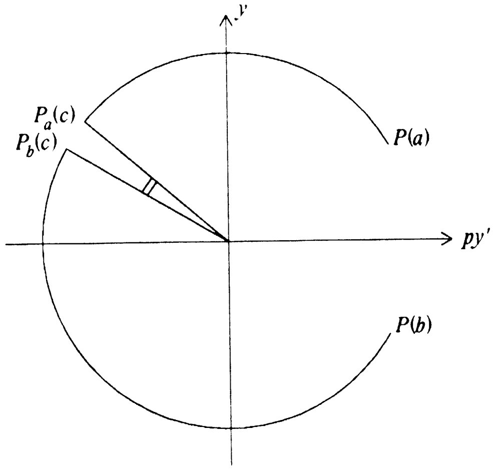
This depends on the complexity of the coefficient functions, whether they or their derivatives are rapidly changing, the tolerance demanded, and how many iterations are needed to obtain convergence. The amount of work per iteration is roughly doubled when TOL is divided by 16. To make economical use of the routine, one should try to obtain good initial values for ELAM and DELAM, and where appropriate good asymptotic formulae. Also the boundary matching points should not be set unnecessarily close to singular points.

8.2. General Description of the Algorithm

A shooting method, for differential equation problems containing unknown parameters, relies on the construction of a 'miss-distance function', which for given trial values of the parameters measures how far the conditions of the problem are from being met. The problem is then reduced to one of finding the values of the parameters for which the miss-distance function is zero, that is to a root-finding process. Shooting methods differ mainly in how the miss-distance is defined.

This routine defines a miss-distance $f(\lambda)$ based on the rotation round the origin of the point $P(x) = (p(x)y'(x),\ y(x))$ in the Phase Plane as the solution proceeds from a to b. The **boundary conditions** define the ray (i.e. two-sided line through the origin) on which $p(x)$

should start, and the ray on which it should finish. The **eigenvalue index** k defines the total number of half-turns it should make. Numerical solution is actually done by 'shooting forward' from $x = a$ and 'shooting backward' from $x = b$ to a matching point $x = c$. Then $f(\lambda)$ is taken as the angle between the rays to the two resulting points $P_a(c)$ and $P_b(c)$. A relative scaling of the py' and y axes, based on the behaviour of the coefficient functions p and q, is used to improve the numerical behaviour.



The resulting function $f(\lambda)$ is monotonic over $-\infty < \lambda < \infty$, increasing if $\frac{\partial q}{\partial \lambda} > 0$ and decreasing if $\frac{\partial q}{\partial \lambda} < 0$, with a unique zero at the desired eigenvalue $\tilde{\lambda}$. The routine measures $f(\lambda)$ in units of a half-turn. This means that as λ increases, $f(\lambda)$ varies by about 1 as each eigenvalue is passed. (This feature implies that the values of $f(\lambda)$ at successive iterations - especially in the early stages of the iterative process - can be used with suitable extrapolation or interpolation to help the choice of initial estimates for eigenvalues near to the one currently being found.)

The routine actually computes a value for $f(\lambda)$ with errors, arising from the local errors of the differential equation code and from the asymptotic formulae provided by the user if singular points are involved. However, the error estimate output in DELAM is usually fairly realistic, in that the actual error $|\tilde{\lambda}-\text{ELAM}|$ is within an order of magnitude of DELAM.

8.3. The Position of the Shooting Matching Point *c*

This point is always one of the values x_i in array XPOINT. It is chosen to be the value of that x_i, $2 \leq i \leq m-1$, that lies closest to the middle of the interval $[x_2, x_{m-1}]$. If there is a tie, the rightmost candidate is chosen. In particular if there are no break-points, then $c = x_{m-1} (= x_3)$ – that is the shooting is from left to right in this case. A break-point may be inserted purely to move c to an interior point of the interval, even though the form of the equations does not require it. This often speeds up convergence especially with singular problems.

8.4. Examples of Coding the COEFFN Routine

Coding COEFFN is straightforward except when break-points are needed. The examples below show:

(a) a simple case,

(b) a case where discontinuities in the coefficient functions or their derivatives necessitate break-points, and

(c) a case where break-points together with the HMAX parameter are an efficient way to deal with a coefficient function that is well-behaved except over one short interval.

(Some of these cases are among the examples in Section 9.)

Example A

The modified Bessel equation

$$x(xy')' + (\lambda x^2 - \nu^2)y = 0.$$

Assuming the interval of solution does not contain the origin and dividing through by x, we have $p(x) = x$, $q(x;\lambda) = \lambda x - \nu^2/x$. The code for COEFFN could be:

```
SUBROUTINE COEFFN (P, Q, DQDL, X, ELAM, JINT)
...
P = X
Q = ELAM*X - NU*NU/X
DQDL = X
RETURN
END
```

where NU (standing for ν) is a ***real*** variable that might be defined in a DATA statement, or might be in user-declared COMMON so that its value could be set in the main program.

Example B

A Schroedinger equation

$$y'' + (\lambda + q(x))y = 0$$

$$\text{where } q(x) = \begin{cases} x^2 - 10 & (|x| \leq 4) \\ \dfrac{6}{|x|} & (|x| > 4) \end{cases}$$

over some interval 'approximating to $(-\infty,\infty)$', say $[-20,20]$. Here we need break-points at ± 4, forming three sub-intervals $i_1 = [-20,-4]$, $i_2 = [-4,4]$, $i_3 = [4,20]$. The code for COEFFN could be:

```
SUBROUTINE COEFFN (P, Q, DQDL, X, ELAM, JINT)
...
IF (JINT.EQ.2) THEN
  Q = ELAM + X*X - 10.0E0
ELSE
  Q = ELAM + 6.0E0/ABS(X)
ENDIF
P = 1.0E0
DQDL = 1.0
RETURN
END
```

The array XPOINT would contain the values x_1, -20.0, -4.0, $+4.0$, $+20.0$, x_6 and m would be 6. The choice of appropriate values for x_1 and x_6 depends on the form of the asymptotic formula computed by BDYVAL and the technique is discussed in the next subsection.

Example C

$$y'' + \lambda(1 - 2e^{-100x^2})y = 0, \qquad -10 \leq x \leq 10.$$

Here $q(x;\lambda)$ is nearly constant over the range except for a sharp inverted spike over approximately $-0.1 \leq x \leq 0.1$. There is a danger that the routine will build up to a large step size and 'step over' the spike without noticing it. By using break-points – say ± 0.5 – one can restrict the step size near the spike without impairing the efficiency elsewhere.

The code for COEFFN could be:

```
SUBROUTINE COEFFN (P, Q, DQDL, X, ELAM, JINT)
...
P = 1.0
DQDL = 1.0 - 2.0 * EXP(-100.0*X*X)
Q = ELAM * DQDL
RETURN
END
```

XPOINT might contain -10.0, -10.0, -0.5, 0.5, 10.0, 10.0 (assuming ±10, are regular points) and m would be 6. HMAX$(1,j)$, $j = 1,2,3$ might contain 0.0, 0.1 and 0.0.

8.5. Examples of Boundary Conditions at Singular Points

Quoting from Bailey [2] page 243: 'Usually ... the differential equation has two essentially different types of solutions near a singular point, and the boundary condition there merely serves to distinguish one kind from the other. This is the case in all the standard examples of mathematical physics'.

In most cases the behaviour of the ratio $p(x)y'/y$ near the point is quite different for the two types of solution. Essentially what the user provides through the BDYVAL routine is an approximation to this ratio, valid as x tends to the singular point (SP).

The user must decide (a) how accurate to make this approximation or asymptotic formula, for example how many terms of a series to use, and (b) where to place the boundary matching point (BMP) at which the numerical solution of the differential equation takes over from the asymptotic formula. Taking the BMP closer to the SP will generally improve the accuracy of the asymptotic formula, but will make the computation more expensive as the Pruefer differential equations generally become progressively more ill-behaved as the SP is approached. The user is strongly recommended to experiment with placing the BMPs. In many singular problems quite crude asymptotic formulae will do. To help the user avoid needlessly accurate formulae, D02KDF outputs two 'sensitivity coefficients' σ_l,σ_r which estimate how much the errors at the BMPs affect the computed eigenvalue. They are described in detail below, see Section 8.6.

Example of coding BDYVAL:

The example below illustrates typical situations:

$$y'' + \left(\lambda - x - \frac{2}{x^2}\right)y = 0 \qquad \text{on } 0 < x < \infty$$

the boundary conditions being that y should remain bounded as x tends to 0 and x tends to ∞.

At the end $x = 0$ there is one solution that behaves like x^2 and another that behaves like x^{-1}. For the first of these solutions $p(x)y'/y$ is asymptotically $2/x$ while for the second it is asymptotically $-1/x$. Thus the desired ratio is specified by setting

$$\text{YL}(1) = x \text{ and } \text{YL}(2) = 2.0.$$

At the end $x = \infty$ the equation behaves like Airy's equation shifted through λ, i.e. like $y'' - ty = 0$ where $t = x - \lambda$, so again there are two types of solutions. The solution we require behaves as

$$\exp(-\tfrac{2}{3}t^{\frac{3}{2}})/\sqrt[4]{t}.$$

and the other as

$$\exp(+\tfrac{2}{3}t^{\frac{3}{2}})/\sqrt[4]{t}.$$

Hence, the desired solution has $p(x)y'/y \sim -\sqrt{t}$ so that we could set YL(1) = 1.0 and YL(2) = $-\sqrt{x-\lambda}$. The complete subroutine might thus be

```
SUBROUTINE BDYVAL (XL, XR, ELAM, YL, YR)
real XL, XR, ELAM, YL(3), YR(3)
YL(1) = XL
YL(2) = 2.0
YR(1) = 1.0
YR(2) = -SQRT(XR-ELAM)
RETURN
END
```

Clearly for this problem it is essential that any value given by D02KDF to XR is well to the right of the value of ELAM, so that the user must vary the right-hand BMP with the eigenvalue index k. One would expect λ_k to be near the kth zero of the Airy function Ai(x), so there is no problem estimating ELAM.

More accurate asymptotic formulae are easily found: near $x = 0$ by the standard Frobenius method, and near $x = \infty$ by using standard asymptotics for Ai(x), Ai$'(x)$, e.g. see Abramowitz and Stegun [1] page 448.

For example by the Frobenius method the solution near x = 0 has the expansion

$$y = x^2(c_0+c_1x+c_2x^2+...)$$

with

$$c_0 = 1,\ c_1 = 0,\ c_2 = \frac{-\lambda}{10},\ c_3 = \frac{1}{18},...,\ c_n = \frac{c_{n-3} - \lambda c_{n-2}}{n(n+3)}$$

This yields

$$\frac{p(x)y'}{y} = \frac{2 - \frac{2}{5}\lambda x^2 + ...}{x\left(1-\frac{\lambda}{10}x^2+...\right)}$$

8.6. The Sensitivity Parameters σ_l and σ_r

The sensitivity parameters σ_l, σ_r (held in HMAX(1,*m*−1) and HMAX(1,*m*) on output) estimate the effect of errors in the boundary conditions. For sufficiently small errors Δy, $\Delta py'$ in y and py' respectively, the relations

$$\Delta\lambda \simeq (y.\Delta py'-py'.\Delta y)_l\sigma_l$$

$$\Delta\lambda \simeq (y.\Delta py'-py'.\Delta y)_r\sigma_r$$

are satisfied, where the subscripts *l*, *r* denote errors committed at the left- and right-hand BMP's respectively, and $\Delta\lambda$ denotes the consequent error in the computed eigenvalue.

8.7. 'Missed Zeros'

This is a pitfall to beware of at a singular point. If the BMP is chosen so far from the SP that a zero of the desired eigenfunction lies in between them, then the routine will fail to 'notice' this zero. Since the index of *k* of an eigenvalue is the number of zeros of its eigenfunction, the result will be that

(a) The wrong eigenvalue will be computed for the given index k – in fact some $\lambda_{k+k'}$ will be found where $k' \geq 1$.

(b) The same index k can cause convergence to any of several eigenvalues depending on the initial values of ELAM and DELAM.

It is up to the user to take suitable precautions – for instance by varying the position of the BMP's in the light of knowledge of the asymptotic behaviour of the eigenfunction at different eigenvalues.

9. Example

We find the 11th eigenvalue of the example of Section 8.5, using the simple asymptotic formulae for the boundary conditions. The results exhibit slow convergence, mainly because XPOINT is set so that the shooting matching point c is at the right-hand end $x = 30.0$. The example results for D02KEF show that much faster convergence is obtained if XPOINT is set to contain an additional break-point at or near the maximum of the coefficient function $q(x;\lambda)$, which in this case is at $x = \sqrt[3]{4}$.

9.1. Program Text

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

```
*     D02KDF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          M
      PARAMETER        (M=4)
```

```
*     .. Local Scalars ..
      real             DELAM, ELAM, TOL
      INTEGER          IFAIL, IFLAG, K, MAXIT
*     .. Local Arrays ..
      real             HMAX(2,M), XPOINT(M)
*     .. External Subroutines ..
      EXTERNAL         BDYVL, COEFF, D02KAY, D02KDF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02KDF Example Program Results'
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'A singular problem'
      TOL = 1.0e-4
      XPOINT(1) = 0.0e0
      XPOINT(2) = 0.1e0
      XPOINT(3) = 30.0e0
      XPOINT(4) = 30.0e0
      HMAX(1,1) = 0.0e0
      MAXIT = 0
      K = 11
      ELAM = 14.0e0
      DELAM = 1.0e0
      IFLAG = 0
      IFAIL = 0
*
*     * To obtain monitoring information from the supplied
*     subroutine MONIT replace the name D02KAY by MONIT in
*     the next statement, and declare MONIT as external *
*
      CALL D02KDF(XPOINT,M,COEFF,BDYVL,K,TOL,ELAM,DELAM,HMAX,MAXIT,
     +            IFLAG,D02KAY,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Final results'
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'K =', K, '  ELAM =', ELAM, '  DELAM =', DELAM
      WRITE (NOUT,99998) 'HMAX(1,M-1) =', HMAX(1,M-1),
     + '    HMAX(1,M) =', HMAX(1,M)
      STOP
*
99999 FORMAT (1X,A,I3,A,F12.3,A,e12.2)
99998 FORMAT (1X,A,F10.3,A,F10.3)
      END
*
      SUBROUTINE COEFF(P,Q,DQDL,X,ELAM,JINT)
*     .. Scalar Arguments ..
      real             DQDL, ELAM, P, Q, X
      INTEGER          JINT
*     .. Executable Statements ..
      P = 1.0e0
      Q = ELAM - X - 2.0e0/(X*X)
      DQDL = 1.0e0
      RETURN
      END
*
      SUBROUTINE BDYVL(XL,XR,ELAM,YL,YR)
*     .. Scalar Arguments ..
      real             ELAM, XL, XR
*     .. Array Arguments ..
      real             YL(3), YR(3)
*     .. Intrinsic Functions ..
      INTRINSIC        SQRT
*     .. Executable Statements ..
      YL(1) = XL
      YL(2) = 2.0e0
      YR(1) = 1.0e0
      YR(2) = -SQRT(XR-ELAM)
      RETURN
      END
*
```

```
      SUBROUTINE MONIT(MAXIT,IFLAG,ELAM,FINFO)
*     .. Parameters ..
      INTEGER           NOUT
      PARAMETER         (NOUT=6)
*     .. Scalar Arguments ..
      real              ELAM
      INTEGER           IFLAG, MAXIT
*     .. Array Arguments ..
      real              FINFO(15)
*     .. Local Scalars ..
      INTEGER           I
*     .. Executable Statements ..
      IF (MAXIT.EQ.-1) THEN
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Output from MONIT'
      END IF
      WRITE (NOUT,99999) MAXIT, IFLAG, ELAM, (FINFO(I),I=1,4)
      RETURN
*
99999 FORMAT (1X,2I4,F10.3,2e12.2,2F8.1)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D02KDF Example Program Results

A singular problem

Final results

K = 11  ELAM =       14.947  DELAM =     0.86E-03
HMAX(1,M-1) =      0.000     HMAX(1,M) =      5.456
```

With MONIT used instead of D02KAY as an argument of D02KDF in the example program, intermediate results similar to these below are obtained:

```
Output from MONIT
   -1   1    14.000   -0.15E+01   -0.20E-03     1.0   679.0
   -2   1    15.000    0.50E+00   -0.36E-03     1.0   627.0
   -3   2    14.750   -0.50E+00   -0.49E-03     1.0   632.0
   -4   2    14.875   -0.50E+00   -0.24E-03     1.0   570.0
   -5   2    14.937   -0.50E+00   -0.66E-03     1.0   471.0
   -6   2    14.969    0.50E+00   -0.27E-03     1.0   441.0
   -7   2    14.953    0.50E+00   -0.41E-03     1.0   431.0
   -8   2    14.945   -0.50E+00   -0.41E-03     1.0   431.0
   -9   2    14.949    0.50E+00   -0.21E-03     1.0   421.0
  -10   2    14.947    0.50E+00   -0.41E-03     1.0   417.0
  -11   2    14.946   -0.50E+00   -0.67E-03     1.0   413.0
  -12   2    14.947   -0.50E+00   -0.37E-03     1.0   417.0
```

D02KEF – NAG Fortran Library Routine Document

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

1. Purpose

D02KEF finds a specified eigenvalue of a regular singular second-order Sturm-Liouville system on a finite or infinite range, using a Pruefer transformation and a shooting method. It also reports values of the eigenfunction and its derivatives. Provision is made for discontinuities in the coefficient functions or their derivatives.

2. Specification

```
      SUBROUTINE D02KEF (XPOINT, M, MATCH, COEFFN, BDYVAL, K, TOL, ELAM,
     1                   DELAM, HMAX, MAXIT, MAXFUN, MONIT, REPORT, IFAIL)
      INTEGER        M, MATCH, K, MAXIT, MAXFUN, IFAIL
      real           XPOINT(M), TOL, ELAM, DELAM, HMAX(2,M)
      EXTERNAL       COEFFN, BDYVAL, MONIT, REPORT
```

3. Description

D02KEF has essentially the same purpose as D02KDF with minor modifications to enable values of the eigenfunction to be obtained after convergence to the eigenvalue has been achieved.

It first finds a specified eigenvalue $\tilde{\lambda}$ of a Sturm-Liouville system defined by a self-adjoint differential equation of the second-order

$$(p(x)y')' + q(x;\lambda)y = 0, \qquad a < x < b$$

together with the appropriate boundary conditions at the two (finite or infinite) end-points a and b. The functions p and q, which are real-valued, must be defined by a subroutine COEFFN. The boundary conditions must be defined by a subroutine BDYVAL, and, in the case of a singularity at a or b, take the form of an asymptotic formula for the solution near the relevant end-point.

When the final estimate $\lambda = \tilde{\lambda}$ of the eigenvalue has been found, the routine integrates the differential equation once more with that value of λ, and with initial conditions chosen so that the integral

$$S = \int_a^b y(x)^2 \frac{\partial q}{\partial \lambda}(x;\lambda)\ dx$$

is approximately one. When $q(x;\lambda)$ is of the form $\lambda w(x) + q(x)$, which is the most common case, S represents the square of the norm of y induced by the inner product

$$<f,g> = \int_a^b f(x)g(x)w(x)dx,$$

with respect to which the eigenfunctions are mutually orthogonal. This normalisation of y is only approximate, but experience shows that S generally differs from unity by only one or two per cent.

During this final integration the REPORT routine supplied by the user is called at each integration mesh point x. Sufficient information is returned to permit the user to compute $y(x)$ and $y'(x)$ for printing or plotting. For reasons described in Section 8.2, D02KEF passes across to REPORT, not y and y', but the Pruefer variables β, ϕ and ρ on which the numerical method is based. Their relationship to y and y' is given by the equations

$$p(x)y' = \sqrt{\beta}\exp\left(\frac{\rho}{2}\right)\cos\left(\frac{\phi}{2}\right); \qquad y = \frac{1}{\sqrt{\beta}}\exp\left(\frac{\rho}{2}\right)\sin\left(\frac{\phi}{2}\right).$$

A specimen REPORT routine is given in Section 9 below.

For the theoretical basis of the numerical method to be valid, the following conditions should hold on the coefficient functions:

(a) $p(x)$ must be non-zero and of one sign throughout the interval (a,b); and,

(b) $\frac{\partial q}{\partial \lambda}$ must be of one sign throughout (a,b) for all relevant values of λ, and must not be identically zero as x varies, for any λ.

Points of discontinuity in the functions p and q or their derivatives are allowed, and should be included as 'break-points' in the array XPOINT.

A good account of the theory of Sturm-Liouville systems, with some description of Pruefer transformations, is given in Birkhoff and Rota [4], Chapter X. An introduction for the user of Pruefer transformations for the numerical solution of eigenvalue problems arising from physics and chemistry is Bailey [2].

The scaled Pruefer method is fairly recent, and is described in a short note by Pryce [6] and in some detail in the technical report [5].

4. References

[1] ABRAMOWITZ, M. and STEGUN, I.A.
Handbook of Mathematical Functions.
Dover Publications, Ch. 4.4, p. 79, 1968.

[2] BAILEY, P.B.
Sturm-Liouville Eigenvalues via a Phase Function.
SIAM J. Appl. Math. 14, pp. 242-249, 1966.

[3] BANKS, D.O. and KUROWSKI, I.
Computation of Eigenvalues of Singular Sturm-Liouville Systems.
Math. Comput., 22, pp. 304-310, 1968.

[4] BIRKHOFF, G. and ROTA, G.C.
Ordinary Differential Equations.
Ginn & Co., Boston and New York, 1962.

[5] PRYCE, J.D.
Two codes for Sturm-Liouville problems.
Bristol University, Computer Science Technical Report CS-81-01, 1981.

[6] PRYCE, J.D. and HARGRAVE, B.A.
The Scale Prüfer Method for one-parameter and multi-parameter eigenvalue problems in ODEs.
Inst. Math. Appl., Numerical Analysis Newsletter, 1, No. 3, 1977.

5. Parameters

1: XPOINT(M) – ***real*** array. *Input*

On entry: the points where the boundary conditions computed by BDYVAL are to be imposed, and also any break-points, i.e. XPOINT(1) to XPOINT(m) must contain values $x_1,...,x_m$ such that

$$x_1 \leq x_2 < x_3 < ... < x_{m-1} \leq x_m$$

with the following meanings:

(a) x_1 and x_m are the left and right end-points, a and b, of the domain of definition of the Sturm-Liouville system if these are finite. If either a or b is infinite, the corresponding value x_1 or x_m may be a more-or-less arbitrarily 'large' number of appropriate sign.

(b) x_2 and x_{m-1} are the Boundary Matching Points (BMP's), that is the points at which the left and right boundary conditions computed in BDYVAL are imposed.

If the left-hand end-point is a regular point then the user should set $x_2 = x_1$ $(= a)$, while if it is a singular point the user must set $x_2 > x_1$. Similarly $x_{m-1} = x_m$ $(= b)$ if the right-hand end-point is regular, and $x_{m-1} < x_m$ if it is singular.

(c) The remaining $m - 4$ points $x_3,...,x_{m-2}$, if any, define 'break-points' which divide the interval $[x_2,x_{m-1}]$ into $m - 3$ sub-intervals

$i_1 = [x_2,x_3]$, ..., $i_{m-3} = [x_{m-2},x_{m-1}]$

Numerical integration of the differential equation is stopped and restarted at each break-point. In simple cases no break-points are needed. However if $p(x)$ or $q(x;\lambda)$ are given by different formulae in different parts of the range, then integration is more efficient if the range is broken up by break-points in the appropriate way. Similarly points where any jumps occur in $p(x)$ or $q(x;\lambda)$, or in their derivatives up to the fifth order, should appear as break-points.

Constraint: X(1) ≤ X(2) < ... < X(M−1) ≤ X(M).

2: M – INTEGER. *Input*

On entry: the number of points in the array XPOINT.

Constraint: M ≥ 4.

3: MATCH – INTEGER. *Input/Output*

On entry: MATCH must be set to the index of the 'break-point' to be used as the matching point (see Section 8.3). If MATCH is set to a value outside the range $[2,m-1]$ then a default value is taken, corresponding to the break-point nearest the centre of the interval [XPOINT(2),XPOINT(m−1)].

On exit: the index of the break-point actually used as the matching point.

4: COEFFN – SUBROUTINE, supplied by the user. *External Procedure*

COEFFN must compute the values of the coefficient functions $p(x)$ and $q(x;\lambda)$ for given values of x and λ. Section 3 states conditions which p and q must satisfy.

Its specification is:

```
SUBROUTINE COEFFN(P, Q, DQDL, X, ELAM, JINT)
real       P, Q, DQDL, X, ELAM
INTEGER    JINT
```

1: P – ***real***. *Output*

On exit: the value of $p(x)$ for the current value of x.

2: Q – ***real***. *Output*

On exit: the value of $q(x;\lambda)$ for the current value of x and the current trial value of λ.

3: DQDL – ***real***. *Output*

On exit: the value of $\frac{\partial q}{\partial \lambda}(x;\lambda)$ for the current value of x and the current trial value of λ. However DQDL is only used in error estimation and an approximation (say to within 20%) will suffice.

4: X – ***real***. *Input*

On entry: the current value of x.

5: ELAM – ***real***. *Input*

On entry: the current trial value of the eigenvalue parameter λ.

6: JINT – INTEGER. *Input*

On entry: the index j of the sub-interval i_j (see specification of XPOINT) in which x lies.

See Sections 8.4 and 9 for examples. COEFFN must be declared as EXTERNAL in the (sub)program from which D02KEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: BDYVAL – SUBROUTINE, supplied by the user. *External Procedure*

BDYVAL must define the boundary conditions. For each end-point, BDYVAL must return (in YL or YR) values of $y(x)$ and $p(x)y'(x)$ which are consistent with the boundary conditions at the end-points; only the ratio of the values matters. Here x is a given point (XL or XR) equal to, or close to, the end-point.

For a **regular** end-point (a, say), $x = a$; and a boundary condition of the form

$$c_1 y(a) + c_2 y'(a) = 0$$

can be handled by returning constant values in YL, e.g. $\mathrm{YL}(1) = c_2$ and $\mathrm{YL}(2) = -c_1 p(a)$.

For a **singular** end-point however, YL(1) and YL(2) will in general be functions of XL and ELAM, and YR(1) and YR(2) functions of XR and ELAM, usually derived analytically from a power-series or asymptotic expansion. Examples are given in Sections 8.5 and 9.

Its specification is:

```
SUBROUTINE BDYVAL(XL, XR, ELAM, YL, YR)
real        XL, XR, ELAM, YL(3), YR(3)
```

1: XL – ***real***. *Input*

On entry: if a is a regular end-point of the system (so that $a = x_1 = x_2$), then XL contains a. If a is a singular point (so that $a \le x_1 < x_2$), then XL contains a point x such that $x_1 < x \le x_2$).

2: XR – ***real***. *Input*

On entry: if b is a regular end-point of the system (so that $x_{m-1} = x_m = b$), then XR contains b. If b is a singular point (so that $x_{m-1} < x_m \le b$), then XR contains a point x such that $x_{m-1} \le x < x_m$.

3: ELAM – ***real***. *Input*

On entry: the current trial value of λ.

4: YL(3) – ***real*** array. *Output*

On exit: YL(1) and YL(2) should contain values of $y(x)$ and $p(x)y'(x)$ respectively (not both zero) which are consistent with the boundary condition at the left-hand end-point, given by x = XL. YL(3) should not be set.

5: YR(3) – ***real*** array. *Output*

On exit: YR(1) and YR(2) should contain values of $y(x)$ and $p(x)y'(x)$ respectively (not both zero) which are consistent with the boundary condition at the right-hand end-point, given by x = XR. YR(3) should not be set.

BDYVAL must be declared as EXTERNAL in the (sub)program from which D02KEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: K – INTEGER. *Input*

On entry: the index k of the required eigenvalue when the eigenvalues are ordered $\lambda_0 < \lambda_1 < \lambda_2 < \ldots < \lambda_k < \ldots$.

Constraint: K $\ge$ 0.

7: TOL – ***real***. *Input*

On entry: the tolerance parameter which determines the accuracy of the computed eigenvalue. The error estimate held in DELAM on exit satisfies the mixed absolute/relative error test

$$\mathrm{DELAM} \le \mathrm{TOL} \times \max(1.0, |\mathrm{ELAM}|) \qquad (*)$$

where ELAM is the final estimate of the eigenvalue. DELAM is usually somewhat smaller than the right-hand side of (*) but not several orders of magnitude smaller.

Constraint: TOL > 0.0.

8: ELAM – ***real***. *Input/Output*

On entry: an initial estimate of the eigenvalue $\tilde{\lambda}$.

On exit: the final computed estimate, whether or not an error occurred.

9: DELAM – ***real***. *Input/Output*

On entry: an indication of the scale of the problem in the λ-direction. DELAM holds the initial 'search step' (positive or negative). Its value is not critical but the first two trial evaluations are made at ELAM and ELAM + DELAM, so the routine will work most efficiently if the eigenvalue lies between these values. A reasonable choice (if a closer bound is not known) is half the distance between adjacent eigenvalues in the neighbourhood of the one sought. In practice, there will often be a problem, similar to the one in hand but with known eigenvalues, which will help one to choose initial values for ELAM and DELAM.

If DELAM = 0.0 on entry, it is given the default value of 0.25×max(1.0,|ELAM|).

On exit: with IFAIL = 0, DELAM holds an estimate of the absolute error in the computed eigenvalue, that is $|\tilde{\lambda}-\text{ELAM}| \simeq \text{DELAM}$. (In Section 8.2 we discuss the assumptions under which this is true.) The true error is rarely more than twice, or less than a tenth, of the estimated error.

With IFAIL ≠ 0, DELAM may hold an estimate of the error, or its initial value, depending on the value of IFAIL. See Section 6 for further details.

10: HMAX(2,M) – ***real*** array. *Input/Output*

On entry: HMAX(1,j) a maximum step size to be used by the differential equation code in the jth sub-interval i_j (as described in the specification of parameter XPOINT), for $j = 1,2,...,m-3$. If it is zero the routine generates a maximum step size internally.

It is recommended that HMAX(1,j) be set to zero unless the coefficient functions p and q have features (such as a narrow peak) within the jth sub-interval that could be 'missed' if a long step were taken. In such a case HMAX(1,j) should be set to about half the distance over which the feature should be observed. Too small a value will increase the computing time for the routine. See Section 8 for further suggestions.

The rest of the array is used as workspace.

On exit: HMAX(1,$m-1$) and HMAX(1,m) contain the sensitivity coefficients σ_l,σ_r, described in Section 8.6. Other entries contain diagnostic output in case of an error (see Section 6).

11: MAXIT – INTEGER. *Input/Output*

On entry: a bound on n_r, the number of root-finding iterations allowed, that is the number of trial values of λ that are used. If MAXIT ≤ 0, no such bound is assumed. (See also under MAXFUN.)

Suggested value: MAXIT = 0.

On exit: MAXIT will have been decreased by the number of iterations actually performed, whether or not it was positive on entry.

12: MAXFUN – INTEGER. *Input*

On entry: a bound on n_f, the number of calls to COEFFN made in any one root-finding iteration. If MAXFUN $\leq$ 0, no such bound is assumed.

Suggested value: MAXFUN = 0.

MAXFUN and MAXIT may be used to limit the computational cost of a call to D02KEF, which is roughly proportional to $n_r \times n_f$.

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

MONIT is called by D02KEF at the end of each root-finding iteration and allows the user to monitor the course of the computation by printing out the parameters (see Section 8 for an example).

If no monitoring is required, the dummy subroutine D02KAY may be used. (D02KAY is included in the NAG Fortran Library. In some implementations of the Library the name is changed to KAYD02: refer to the Users' Note for your implementation.)

Its specification is:

```
SUBROUTINE MONIT(MAXIT, IFLAG, ELAM, FINFO)
INTEGER    MAXIT, IFLAG
real       ELAM, FINFO(15)
```

1: MAXIT – INTEGER. *Input*

On entry: the current value of the parameter MAXIT of D02KEF; this is decreased by one at each iteration.

2: IFLAG – INTEGER. *Input*

On entry: IFLAG describes what phase the computation is in, as follows:

IFLAG < 0

an error occurred in the computation of the 'miss-distance' at this iteration; an error exit from D02KEF with IFAIL = –IFLAG will follow.

IFLAG = 1

the routine is trying to bracket the eigenvalue $\tilde{\lambda}$.

IFLAG = 2

the routine is converging to the eigenvalue $\tilde{\lambda}$ (having already bracketed it).

3: ELAM – ***real***. *Input*

On entry: the current trial value of λ.

4: FINFO(15) – ***real*** array. *Input*

On entry: information about the behaviour of the shooting method, and diagnostic information in the case of errors. It should **not** normally be printed in full if no error has occurred (that is, if IFLAG > 0), though the first few components may be of interest to the user. In case of an error (IFLAG < 0) all the components of FINFO should be printed. The contents of FINFO are as follows:

FINFO(1): the current value of the 'miss-distance' or 'residual' function $f(\lambda)$ on which the shooting method is based. (See Section 8.2 for further notes on it.) FINFO(1) is set to zero if IFLAG < 0.

FINFO(2): an estimate of the quantity $\partial\lambda$ defined as follows. Consider the perturbation in the miss-distance $f(\lambda)$ that would result if the local error, in the solution of the differential equation, were always positive and equal to its maximum permitted value. Then $\partial\lambda$ is the perturbation in λ that would have the same effect on $f(\lambda)$. Thus, at the zero of $f(\lambda)$,$|\partial\lambda|$ is an approximate bound on the perturbation of the zero (that is the eigenvalue) caused by errors in numerical solution. If $\partial\lambda$ is very large then it is possible that there has been a programming error in COEFFN such that q is independent of λ. If this is the case, an error exit with IFAIL = 5 should follow. FINFO(2) is set to zero if IFLAG < 0.

FINFO(3): the number of internal iterations, using the same value of λ and tighter accuracy tolerances, needed to bring the accuracy (that is the value of $\partial\lambda$) to an acceptable value. Its value should normally be 1.0, and should almost never exceed 2.0.

FINFO(4): the number of calls to COEFFN at this iteration.

FINFO(5): the number of successful steps taken by the internal differential equation solver at this iteration. A step is successful if it is used to advance the integration.

FINFO(6): the number of unsuccessful steps used by the internal integrator at this iteration.

FINFO(7): the number of successful steps at the maximum step size taken by the internal integrator at this iteration.

FINFO(8): is not used.

FINFO(9) to FINFO(15): set to zero, unless IFLAG $<$ 0 in which case they hold the following values describing the point of failure:

FINFO(9): contains the index of the sub-interval where failure occurred, in the range 1 to $m - 3$. In case of an error in BDYVAL, it is set to 0 or $m - 2$ depending on whether the left or right boundary condition caused the error.

FINFO(10): the value of the independent variable x, the point at which the error occurred. In case of an error in BDYVAL, it is set to the value of XL or XR as appropriate (see the specification of BDYVAL).

FINFO(11), FINFO(12), FINFO(13): the current values of the Pruefer dependent variables β, ϕ and ϱ respectively. These are set to zero in case of an error in BDYVAL.

FINFO(14): the local-error tolerance being used by the internal integrator at the point of failure. This is set to zero in the case of an error in BDYVAL.

FINFO(15): the last integration mesh point. This is set to zero in the case of an error in BDYVAL.

MONIT must be declared as EXTERNAL in the (sub)program from which D02KEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

This routine provides the means by which the user may compute the eigenfunction $y(x)$ and its derivative at each integration mesh point x. (See Section 8 for an example).

Its specification is:

```
SUBROUTINE REPORT (X, V, JINT)
INTEGER    JINT
real       X, V(3)
```

1: X – ***real***. *Input*

On entry: the current value of the independent variable x. See Section 8.3 for the order in which values of x are supplied.

2: V(3) – ***real*** array. *Input*

On entry: V(1), V(2), V(3) hold the current values of the Pruefer variables β, ϕ, ϱ respectively.

3: JINT – INTEGER. *Input*

On entry: JINT indicates the sub-interval between break-points in which X lies exactly as for the routine COEFFN, **except** that at the extreme left end-point (when x = XPOINT(2)) JINT is set to 0 and at the extreme right end-point (when $x = x_r$ = XPOINT(m−1)) JINT is set to $m - 2$.

REPORT must be declared as EXTERNAL in the (sub)program from which D02KEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

15: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

A parameter error. All parameters (except IFAIL) are left unchanged. The reason for the error is shown by the value of HMAX(2,1) as follows:

HMAX(2,1) = 1: M < 4;
HMAX(2,1) = 2: K < 0;
HMAX(2,1) = 3: TOL ≤ 0.0;
HMAX(2,1) = 4: XPOINT(1) to XPOINT(m) are not in ascending order.
HMAX(2,2) gives the position i in XPOINT where this was detected.

IFAIL = 2

At some call to BDYVAL, invalid values were returned, that is, either YL(1) = YL(2) = 0.0, or YR(1) = YR(2) = 0.0 (a programming error in BDYVAL). See the last call of MONIT for details.

This error exit will also occur if $p(x)$ is zero at the point where the boundary condition is imposed. Probably BDYVAL was called with XL equal to a singular end-point a or with XR equal to a singular end-point b.

IFAIL = 3

At some point between XL and XR the value of $p(x)$ computed by COEFFN became zero or changed sign. See the last call of MONIT for details.

IFAIL = 4

MAXIT > 0 on entry, and after MAXIT iterations the eigenvalue had not been found to the required accuracy.

IFAIL = 5

The 'bracketing' phase (with parameter IFLAG of MONIT equal to 1) failed to bracket the eigenvalue within ten iterations. This is caused by an error in formulating the problem (for example, q is independent of λ), or by very poor initial estimates of ELAM, DELAM.

On exit ELAM and ELAM + DELAM give the end-points of the interval within which no eigenvalue was located by the routine.

IFAIL = 6

MAXFUN > 0 on entry, and the last iteration was terminated because more than MAXFUN calls to COEFFN were used. See the last call of MONIT for details.

IFAIL = 7

To obtain the desired accuracy the local error tolerance was set so small at the start of some sub-interval that the differential equation solver could not choose an initial step size large enough to make significant progress. See the last call of MONIT for diagnostics.

IFAIL = 8

At some point inside a sub-interval the step size in the differential equation solver was reduced to a value too small to make significant progress (for the same reasons as with IFAIL = 7). This could be due to pathological behaviour of $p(x)$ and $q(x;\lambda)$ or to an unreasonable accuracy requirement or to the current value of λ making the equations 'stiff'. See the last call of MONIT for details.

IFAIL = 9

TOL is too small for the problem being solved and the ***machine precision*** is being used. The final value of ELAM should be a very good approximation to the eigenvalue.

IFAIL = 10

C05AZF, called by D02KEF, has terminated with the error exit corresponding to a pole of the residual function $f(\lambda)$. This error exit should not occur, but if it does, try solving the problem again with a smaller value for TOL.

IFAIL = 11 (D02KDY)
IFAIL = 12 (C05AZF)

A serious error has occurred in the specified routine. Check all subroutine calls and array dimensions. Seek expert help.

HMAX(2,1) holds the failure exit number from the routine where the failure occurred. In the case of a failure in C05AZF, HMAX(2,2) holds the value of parameter IND of C05AZF.

Note: error exits with IFAIL = 2, 3, 6, 7, 8, 11 are caused by being unable to set up or solve the differential equation at some iteration, and will be immediately preceded by a call of MONIT giving diagnostic information. For other errors, diagnostic information is contained in HMAX$(2,j)$, for $j = 1,2,...,m$, where appropriate.

7. Accuracy

See the discussion in Section 8.2.

8. Further Comments

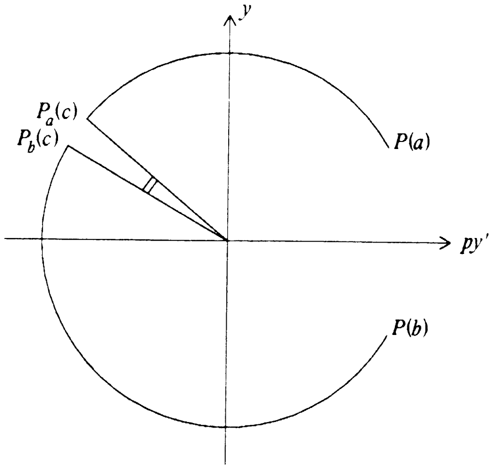
8.1. Timing

The time taken by the routine depends on the complexity of the coefficient functions, whether they or their derivatives are rapidly changing, the tolerance demanded, and how many iterations are needed to obtain convergence. The amount of work per iteration is roughly doubled when TOL is divided by 16. To make the most economical use of the routine, one should try to obtain good initial values for ELAM and DELAM, and, where appropriate, good asymptotic formulae. The boundary matching points should not be set unnecessarily close to singular points. The extra time needed to compute the eigenfunction is principally the cost of one additional integration once the eigenvalue has been found.

8.2. General Description of the Algorithm

A shooting method, for differential equation problems containing unknown parameters, relies on the construction of a 'miss-distance function', which for given trial values of the parameters measures how far the conditions of the problem are from being met. The problem is then reduced to one of finding the values of the parameters for which the miss-distance function is zero, that is to a root-finding process. Shooting methods differ mainly in how the miss-distance is defined.

This routine defines a miss-distance $f(\lambda)$ based on the rotation around the origin of the point $P(x) = (p(x)y'(x),\ y(x))$ in the Phase Plane as the solution proceeds from a to b. The **boundary-conditions** define the ray (i.e. two-sided line through the origin) on which $p(x)$ should start, and the ray on which it should finish. The **eigenvalue** index k defines the total number of half-turns it should make. Numerical solution is actually done by 'shooting forward' from $x = a$ and 'shooting backward' from $x = b$ to a matching point $x = c$. Then $f(\lambda)$ is taken as the angle between the rays to the two resulting points $P_a(c)$ and $P_b(c)$. A relative scaling of the py' and y axes, based on the behaviour of the coefficient functions p and q, is used to improve the numerical behaviour.



The resulting function $f(\lambda)$ is monotonic over $-\infty < \lambda < \infty$, increasing if $\frac{\partial q}{\partial \lambda} > 0$ and decreasing if $\frac{\partial q}{\partial \lambda} < 0$, with a unique zero at the desired eigenvalue $\tilde{\lambda}$. The routine measures $f(\lambda)$ in units of a half-turn. This means that as λ increases, $f(\lambda)$ varies by about 1 as each eigenvalue is passed. (This feature implies that the values of $f(\lambda)$ at successive iterations – especially in the early stages of the iterative process – can be used with suitable extrapolation or interpolation to help the choice of initial estimates for eigenvalues near to the one currently being found.)

The routine actually computes a value for $f(\lambda)$ with errors, arising from the local errors of the differential equation code and from the asymptotic formulae provided by the user if singular points are involved. However, the error estimate output in DELAM is usually fairly realistic, in that the actual error $|\tilde{\lambda}-\text{ELAM}|$ is within an order of magnitude of DELAM.

We pass the values of β, ϕ, ϱ across through REPORT rather than converting them to values of y, y' inside D02KEF, for the following reasons. First, there may be cases where auxiliary quantities can be more accurately computed from the Pruefer variables than from y and y'. Second, in singular problems on an infinite interval y and y' may underflow towards the end of the range, whereas the Pruefer variables remain well-behaved. Third, with high-order eigenvalues (and therefore highly oscillatory eigenfunctions) the eigenfunction may have a complete oscillation (or more than one oscillation) between two mesh points, so that values of y and y' at mesh points give a very poor representation of the curve. The probable behaviour of the Pruefer variables in this case is that β and ϱ vary slowly whilst ϕ increases quickly: for all three Pruefer variables linear interpolation between the values at adjacent mesh points is probably sufficiently accurate to yield acceptable intermediate values of β, ϕ, ϱ (and hence of y,y') for graphical purposes.

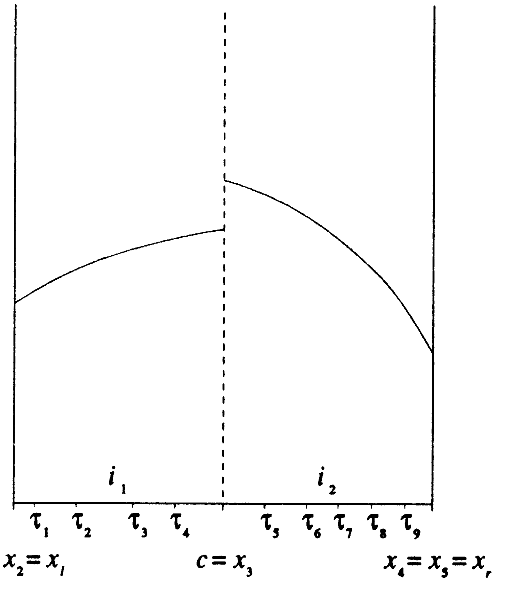
Similar considerations apply to the exponentially decaying 'tails' of the eigenfunctions that often occur in singular problems. Here ϕ has approximately constant value whilst ϱ increases rapidly in the direction of integration, though the step length is generally fairly small over such a range.

If the solution is output through REPORT at x-values which are too widely spaced, the step length can be controlled by choosing HMAX suitably, or, preferably, by reducing TOL. Both these choices will lead to more accurate eigenvalues and eigenfunctions but at some computational cost.

8.3. The Position of the Shooting Matching Point c

This point is always one of the values x_i in array XPOINT. It may be specified using the parameter MATCH. The default value is chosen to be the value of that x_i, $2 \le i \le m-1$, that lies closest to the middle of the interval $[x_2, x_{m-1}]$. If there is a tie, the rightmost candidate is chosen. In particular if there are no break-points then $c = x_{m-1}$ $(= x_3)$ – that is the shooting is from left to right in this case. A break-point may be inserted purely to move c to an interior point of the interval, even though the form of the equations does not require it. This often speeds up convergence especially with singular problems.

Note that the shooting method used by the code integrates first from the left-hand end x_l, then from the right-hand end x_r, to meet at the matching point c in the middle. This will of course be reflected in printed or graphical output. The diagram shows a possible sequence of nine mesh points τ_1 through τ_9 in the order in which they appear, assuming there are just two sub-intervals (so $m = 5$).



i_1
i_2
τ_1 τ_2 τ_3 τ_4 τ_5 τ_6 τ_7 τ_8 τ_9
$x_1 = x_2 = x_l$
$c = x_3$
$x_4 = x_5 = x_r$

Figure 1

Since the shooting method usually fails to match up the two 'legs' of the curve exactly, there is bound to be a jump in y, or in $p(x)y'$ or both, at the matching point c. The code in fact 'shares' the discrepancy out so that both y and $p(x)y'$ have a jump. A large jump does **not** imply an inaccurate eigenvalue, but implies either

(a) a badly chosen matching point: if $q(x;\lambda)$ has a 'humped' shape, c should be chosen near the maximum value of q, especially if q is negative at the ends of the interval.

(b) An inherently ill-conditioned problem, typically one where another eigenvalue is pathologically close to the one being sought. In this case it is extremely difficult to obtain an accurate eigenfunction.

In Section 9 below, we find the 11th eigenvalue and corresponding eigenfunction of the equation

$$y'' + (\lambda - x - 2/x^2)y = 0 \text{ on } 0 < x < \infty,$$

the boundary conditions being that y should remain bounded as x tends to 0 and x tends to ∞. The coding of this problem is discussed in detail in Section 8.5.

The choice of matching point c is open. If we choose $c = 30.0$ as in the D02KDF example program we find that the exponentially increasing component of the solution dominates and we get extremely inaccurate values for the eigenfunction (though the eigenvalue is determined accurately). The values of the eigenfunction calculated with $c = 30.0$ are given schematically in Figure 2.

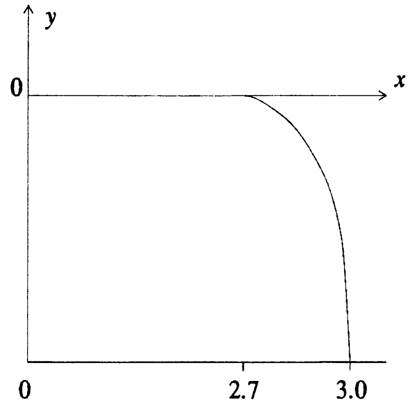


Figure 2

If we choose c as the maximum of the hump in $q(x;\lambda)$ (see (a) above) we instead obtain the accurate results given in Figure 3.

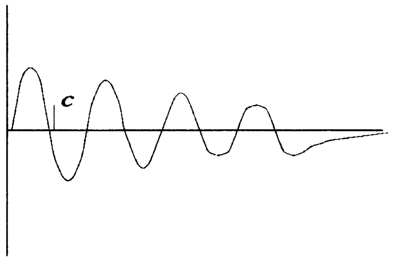


Figure 3

8.4. Examples of Coding the COEFFN Routine

Coding COEFFN is straightforward except when break-points are needed. The examples below show:

(a) a simple case,

(b) a case in which discontinuities in the coefficient functions or their derivatives necessitate break-points, and

(c) a case where break-points together with the HMAX parameter are an efficient way to deal with a coefficient function that is well-behaved except over one short interval.

(Some of these cases are among the examples in Section 9.)

Example A

The modified Bessel equation

$$x(xy')' + (\lambda x^2 - \nu^2)y = 0.$$

Assuming the interval of solution does not contain the origin, dividing through by x, we have $p(x) = x$, $q(x;\lambda) = \lambda x - \nu^2/x$. The code could be

```
SUBROUTINE COEFFN(P,Q,DQDL,X,ELAM,JINT)
. . .
P = X
Q = ELAM*X - NU*NU/X
DQDL = X
RETURN
END
```

where NU (standing for v) is a ***real*** variable that might be defined in a DATA statement, or might be in user-declared COMMON so that its value could be set in the main program.

Example B

The Schroedinger equation

$$y'' + (\lambda + q(x))y = 0$$

where $q(x) = \begin{cases} x^2 - 10 & (|x| \leq 4), \\ 6/|x| & (|x| > 4), \end{cases}$

over some interval 'approximating to $(-\infty,\infty)$', say [–20, 20]. Here we need break-points at ± 4, forming three sub-intervals $i_1 = [-20,-4]$, $i_2 = [-4,4]$, $i_3 = [4,20]$. The code could be

```
SUBROUTINE COEFFN(P,Q,DQDL,X,ELAM,JINT)
...
IF (JINT.EQ.2) THEN
  Q = ELAM + X*X - 10.0E0
ELSE
  Q = ELAM + 6.0E0/ABS(X)
ENDIF
P = 1.0E0
DQDL = 1.0E0
RETURN
END
```

The array XPOINT would contain the values x_1, –20.0, –4.0, +4.0, +20.0, x_6 and m would be 6. The choice of appropriate values for x_1 and x_6 depends on the form of the asymptotic formula computed by BDYVAL and the technique is discussed in the next subsection.

Example C

$$y'' + \lambda(1-2e^{-100x^2})y = 0, \qquad \text{over } -10 \leq x \leq 10.$$

Here $q(x;\lambda)$ is nearly constant over the range except for a sharp inverted spike over approximately $-0.1 \leq x \leq 0.1$. There is a danger that the routine will build up to a large step size and 'step over' the spike without noticing it. By using break-points – say at ± 0.5 – one can restrict the step size near the spike without impairing the efficiency elsewhere.

The code for COEFFN could be

```
SUBROUTINE COEFFN(P,Q,DQDL,X,ELAM,JINT)
...
P = 1.0E0
DQDL = 1.0E0 - 2.0E0*EXP(-100.0E0*X*X)
Q = ELAM*DQDL
RETURN
END
```

XPOINT might contain –0.0, –10.0, –0.5, 0.5, 10.0, 10.0 (assuming ± 10 are regular points) and m would be 6. HMAX$(1,j)$, j = 1,2,3 might contain 0.0, 0.1 and 0.0.

8.5. Examples of Boundary Conditions at Singular Points

Quoting from Bailey [2] page 243: 'Usually ... the differential equation has two essentially different types of solution near a singular point, and the boundary condition there merely serves to distinguish one kind from the other. This is the case in all the standard examples of mathematical physics.'

In most cases the behaviour of the ratio $p(x)y'/y$ near the point is quite different for the two types of solution. Essentially what the user provides through his BDYVAL routine is an approximation to this ratio, valid as x tends to the singular point (SP).

The user must decide (a) how accurate to make this approximation or asymptotic formula, for example how many terms of a series to use, and (b) where to place the boundary matching point (BMP) at which the numerical solution of the differential equation takes over from the asymptotic formula. Taking the BMP closer to the SP will generally improve the accuracy of the asymptotic formula, but will make the computation more expensive as the Pruefer differential equations generally become progressively more ill-behaved as the SP is approached. The user is strongly recommended to experiment with placing the BMPs. In many singular problems quite crude asymptotic formulae will do. To help the user avoid needlessly accurate formulae, D02KEF outputs two 'sensitivity coefficients' σ_l, σ_r which estimate how much the errors at the BMP's affect the computed eigenvalue. They are described in detail below, see Section 8.6.

Example of coding BDYVAL:

The example below illustrates typical situations:

$$y'' + \left(\lambda - x - \frac{2}{x^2}\right)y = 0, \qquad \text{for } 0 < x < \infty$$

the boundary conditions being that y should remain bounded as x tends to 0 and x tends to ∞.

At the end $x = 0$ there is one solution that behaves like x^2 and another that behaves like x^{-1}. For the first of these solutions $p(x)y'/y$ is asymptotically $2/x$ while for the second it is asymptotically $-1/x$. Thus the desired ratio is specified by setting

YL(1) = x and YL(2) = 2.0.

At the end $x = \infty$ the equation behaves like Airy's equation shifted through λ, i.e. like $y'' - ty = 0$ where $t = x - \lambda$, so again there are two types of solution. The solution we require behaves as

$$\exp\left(-\tfrac{2}{3}\ t^{\frac{3}{2}}\right)/\sqrt[4]{t}.$$

and the other as

$$\exp\left(+\tfrac{2}{3}\ t^{\frac{3}{2}}\right)/\sqrt[4]{t}.$$

ence, the desired solution has $p(x)y'/y \sim -\sqrt{t}$ so that we could set YR(1) = 1.0 and YR(2) = $-\sqrt{x-\lambda}$. The complete subroutine might thus be

```
SUBROUTINE BDYVAL (XL,XR,ELAM,YL,YR)
real               XL, XR, ELAM, YL(3), YR(3)
YL(1) = XL
YL(2) = 2.0E0
YR(1) = 1.0E0
YR(2) = -SQRT(XR - ELAM)
RETURN
END
```

Clearly for this problem it is essential that any value given by D02KEF to XR is well to the right of the value of ELAM, so that the user must vary the right-hand BMP with the eigenvalue index k. One would expect λ_k to be near the kth zero of the Airy function Ai(x), so there is no problem in estimating ELAM.

More accurate asymptotic formulae are easily found – near $x = 0$ by the standard Frobenius method, and near $x = \infty$ by using standard asymptotics for Ai(x), Ai′(x) (see [1], p. 448). For example, by the Frobenius method the solution near $x = 0$ has the expansion

$$y = x^2(c_0 + c_1x + c_2x^2 + ...)$$

with

$$c_0 = 1,\ c_1 = 0,\ c_2 = -\frac{\lambda}{10},\ c_3 = \frac{1}{18},\ ...,\ c_n = \frac{c_{n-3} - \lambda c_{n-2}}{n(n+3)}.$$

This yields

$$\frac{p(x)y'}{y} = \frac{2 - \frac{2}{5}\lambda x^2 + ...}{x\left(1 - \frac{\lambda}{10}x^2 + ...\right)}.$$

8.6. The Sensitivity Parameters σ_l and σ_r

The sensitivity parameters σ_l, σ_r (held in HMAX(1,m−1) and HMAX(1,m) on output) estimate the effect of errors in the boundary conditions. For sufficiently small errors Δy, $\Delta py'$ in y and py' respectively, the relations

$$\Delta\lambda \simeq (y.\Delta py' - py'.\Delta y)_l \sigma_l$$

$$\Delta\lambda \simeq (y.\Delta py' - py'.\Delta y)_r \sigma_r$$

are satisfied where the subscripts l, r denote errors committed at left- and right-hand BMP's respectively, and $\Delta\lambda$ denotes the consequent error in the computed eigenvalue.

8.7. 'Missed Zeros'

This is a pitfall to beware of at a singular point. If the BMP is chosen so far from the SP that a zero of the desired eigenfunction lies in between them, then the routine will fail to 'notice' this zero. Since the index of k of an eigenvalue is the number of zeros of its eigenfunction, the result will be that:

(a) The wrong eigenvalue will be computed for the given index k – in fact some $\lambda_{k+k'}$ will be found where $k' \geq 1$.

(b) The same index k can cause convergence to any of several eigenvalues depending on the initial values of ELAM and DELAM.

It is up to the user to take suitable precautions – for instance by varying the position of the BMP's in the light of his knowledge of the asymptotic behaviour of the eigenfunction at different eigenvalues.

9. Example

To find the 11th eigenvalue and eigenfunction of the example of Section 8.5, using the simple asymptotic formulae for the boundary conditions.

Comparison of the results from this example program with the corresponding results from D02KDF example program shows that similar output is produced from the routine MONIT, followed by the eigenfunction values from REPORT, and then a further line of information from MONIT (corresponding to the integration to find the eigenfunction). Final information is printed within the example program exactly as with D02KDF.

Note the discrepancy at the matching point c (= $\sqrt[3]{4}$, the maximum of $q(x;\lambda)$, in this case) between the solutions obtained by integrations from left and right end-points.

9.1. Program Text

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

```
*     D02KEF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          M
      PARAMETER        (M=5)
*     .. Local Scalars ..
      real             DELAM, ELAM, TOL
      INTEGER          IFAIL, IFLAG, K, MATCH, MAXIT
*     .. Local Arrays ..
      real             HMAX(2,M), XPOINT(M)
*     .. External Subroutines ..
      EXTERNAL         BDYVL, COEFF, D02KAY, D02KEF, REPORT
```

```
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02KEF Example Program Results'
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'A singular problem'
      TOL = 1.0e-4
      XPOINT(1) = 0.0e0
      XPOINT(2) = 0.1e0
      XPOINT(3) = 4.0e0**(1.0e0/3.0e0)
      XPOINT(4) = 30.0e0
      XPOINT(5) = 30.0e0
      HMAX(1,1) = 0.0e0
      HMAX(1,2) = 0.0e0
      MAXIT = 0
      K = 11
      ELAM = 14.0e0
      DELAM = 1.0e0
      MATCH = 0
      IFLAG = 0
      IFAIL = 0
*
*     To obtain monitoring information from the supplied
*     subroutine MONIT replace the name D02KAY by MONIT in
*     the next statement, and declare MONIT as external *
*
      CALL D02KEF(XPOINT,M,MATCH,COEFF,BDYVL,K,TOL,ELAM,DELAM,HMAX,
     +            MAXIT,IFLAG,D02KAY,REPORT,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Final results'
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'K =', K, '  ELAM =', ELAM, '  DELAM =', DELAM
      WRITE (NOUT,99998) 'HMAX(1,M-1) =', HMAX(1,M-1),
     +  '    HMAX(1,M) =', HMAX(1,M)
      STOP
*
99999 FORMAT (1X,A,I3,A,F12.3,A,e12.2)
99998 FORMAT (1X,A,F10.3,A,F10.3)
      END
*
      SUBROUTINE COEFF(P,Q,DQDL,X,ELAM,JINT)
*     .. Scalar Arguments ..
      real              DQDL, ELAM, P, Q, X
      INTEGER           JINT
*     .. Executable Statements ..
      P = 1.0e0
      Q = ELAM - X - 2.0e0/(X*X)
      DQDL = 1.0e0
      RETURN
      END
*
      SUBROUTINE BDYVL(XL,XR,ELAM,YL,YR)
*     .. Scalar Arguments ..
      real              ELAM, XL, XR
*     .. Array Arguments ..
      real              YL(3), YR(3)
*     .. Intrinsic Functions ..
      INTRINSIC         SQRT
*     .. Executable Statements ..
      YL(1) = XL
      YL(2) = 2.0e0
      YR(1) = 1.0e0
      YR(2) = -SQRT(XR-ELAM)
      RETURN
      END
*
```

```
      SUBROUTINE REPORT(X,V,JINT)
*     .. Parameters ..
      INTEGER           NOUT
      PARAMETER         (NOUT=6)
*     .. Scalar Arguments ..
      real              X
      INTEGER           JINT
*     .. Array Arguments ..
      real              V(3)
*     .. Local Scalars ..
      real              PYP, R, SQRTB, Y
*     .. External Functions ..
      real              X02AMF
      EXTERNAL          X02AMF
*     .. Intrinsic Functions ..
      INTRINSIC         COS, EXP, LOG, SIN, SQRT
*     .. Executable Statements ..
      IF (JINT.EQ.0) THEN
         WRITE (NOUT,*)
         WRITE (NOUT,*) ' Eigenfunction values'
         WRITE (NOUT,*) '       X          Y            PYP'
      END IF
      SQRTB = SQRT(V(1))
*     Avoid underflow in call of EXP
      IF (0.5e0*V(3).GE.LOG(X02AMF())) THEN
         R = EXP(0.5e0*V(3))
      ELSE
         R = 0.0e0
      END IF
      PYP = R*SQRTB*COS(0.5e0*V(2))
      Y = R/SQRTB*SIN(0.5e0*V(2))
      WRITE (NOUT,99999) X, Y, PYP
      RETURN
*
99999 FORMAT (1X,F10.3,1P,2F12.4)
      END
*
      SUBROUTINE MONIT(MAXIT,IFLAG,ELAM,FINFO)
*     .. Parameters ..
      INTEGER           NOUT
      PARAMETER         (NOUT=6)
*     .. Scalar Arguments ..
      real              ELAM
      INTEGER           IFLAG, MAXIT
*     .. Array Arguments ..
      real              FINFO(15)
*     .. Local Scalars ..
      INTEGER           I
*     .. Executable Statements ..
      IF (MAXIT.EQ.-1) THEN
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Output from MONIT'
      END IF
      WRITE (NOUT,99999) MAXIT, IFLAG, ELAM, (FINFO(I),I=1,4)
      RETURN
*
99999 FORMAT (1X,2I4,F10.3,2e12.2,2F8.1)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D02KEF Example Program Results

A singular problem

 Eigenfunction values
      X          Y           PYP
    0.100      0.1239      2.4777
    0.168      0.3434      3.9272
    0.216      0.5531      4.7669
    0.312      1.0703      5.8388
    0.407      1.6389      5.9480
    0.578      2.5048      3.7435
    0.724      2.7923     -0.0179
    0.908      2.2894     -5.3687
    1.136      0.5181     -9.3947
    1.451     -2.1535     -5.7951
    1.587     -2.6671     -1.5792
   30.000      0.0000      0.0000
   29.097      0.0000      0.0000
   28.753      0.0000      0.0000
   28.384      0.0000      0.0000
   28.114      0.0000      0.0000
   27.825      0.0000      0.0000
   27.518      0.0000      0.0000
   27.204      0.0000      0.0000
   26.834      0.0000      0.0000
   26.529      0.0000      0.0000
   26.215      0.0000      0.0000
   25.844      0.0000      0.0000
   25.546      0.0000      0.0000
   25.208      0.0000      0.0000
   24.892      0.0000      0.0000
   24.569      0.0000      0.0000
   24.175      0.0000      0.0000
   23.851      0.0000      0.0000
   23.482      0.0000      0.0000
   23.174      0.0000      0.0000
   22.806      0.0000      0.0000
   22.486      0.0000      0.0000
   22.098      0.0000      0.0000
   21.759      0.0000      0.0000
   21.351      0.0000      0.0001
   20.995     -0.0001      0.0002
   20.566     -0.0002      0.0005
   20.191     -0.0005      0.0013
   19.735     -0.0016      0.0035
   19.334     -0.0037      0.0081
   18.987     -0.0078      0.0161
   18.529     -0.0196      0.0384
   18.092     -0.0450      0.0832
   17.691     -0.0925      0.1609
   17.303     -0.1778      0.2898
   16.903     -0.3332      0.5027
   16.466     -0.6250      0.8540
   15.931     -1.2335      1.4473
   15.550     -1.8735      1.9097
   15.169     -2.6744      2.2636
   14.771     -3.5979      2.2923
   14.404     -4.3627      1.7713
   14.072     -4.7897      0.6994
   13.643     -4.6434     -1.5170
   13.175     -3.2554     -4.3946
   12.690     -0.5778     -6.2798
   12.262      2.0321     -5.4444
   11.871      3.5892     -2.1544
   11.489      3.5453      2.4652
   11.095      1.7261      6.4061
   10.691     -1.1184      6.8904
```

```
     10.299       -3.1686        2.9093
      9.926       -3.1465       -3.0988
      9.550       -1.0523       -7.4276
      9.042        2.5418       -4.9033
      8.538        2.7120        4.4439
      8.041       -0.8508        7.9253
      7.517       -3.0773       -0.9682
      7.031       -0.3714       -8.4971
      6.541        2.8684       -2.4575
      6.041        1.1567        8.1340
      5.521       -2.6813        3.4390
      5.024       -1.2139       -8.2270
      4.525        2.6005       -3.6522
      4.035        1.0883        8.5529
      3.536       -2.6508        2.8120
      3.035       -0.5047       -9.2862
      2.518        2.7330        0.2975
      2.016       -0.5905        9.3484
      1.587       -2.6715       -1.5826

Final results

K = 11  ELAM =        14.946  DELAM =     0.80E-03
HMAX(1,M-1) =     -0.015    HMAX(1,M) =      0.000
```

With MONIT used instead of D02KAY as an argument of D02KEF in the example program, intermediate results similar to those below are obtained:

```
Output  from  MONIT
  -1    1     14.000    -0.11E+01    -0.18E-03       1.0     644.0
  -2    1     15.000     0.66E-01    -0.21E-03       1.0     582.0
  -3    2     14.946    -0.91E-03    -0.25E-03       1.0     549.0
  -4    2     14.946    -0.14E-03    -0.47E-03       1.0     555.0
  -5    2     14.947     0.30E-03    -0.24E-03       1.0     536.0
```

D02LAF – NAG Fortran Library Routine Document

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

1. Purpose

D02LAF is a routine for integrating a non-stiff system of second order ordinary differential equations using Runge-Kutta-Nystrom techniques.

2. Specification

```
SUBROUTINE D02LAF (FCN, NEQ, T, TEND, Y, YP, YDP, RWORK, LRWORK,
1                          IFAIL)
INTEGER         NEQ, LRWORK, IFAIL
real            T, TEND, Y(NEQ), YP(NEQ), YDP(NEQ), RWORK(LRWORK)
EXTERNAL        FCN
```

3. Description

Given the initial values $x,y_1,y_2,...,y_{\mathrm{NEQ}}$, $y'_1,y'_2,...,y'_{\mathrm{NEQ}}$ the routine integrates a non-stiff system of second order differential equations of the type,

$$y''_i = f_i(x,y_1,y_2,...,y_{\mathrm{NEQ}}), \qquad i = 1,2,...,\mathrm{NEQ},$$

from x = T to x = TEND using a Runge-Kutta-Nystrom formula pair. The system is defined by a subroutine FCN supplied by the user, which evaluates f_i in terms of x and $y_1,y_2,...,y_{\mathrm{NEQ}}$, where $y_1,y_2,...,y_{\mathrm{NEQ}}$ are supplied at x.

There are two Runge-Kutta-Nystrom formula pairs implemented in this routine. The lower order method is intended for users with moderate accuracy requirements and may be used in conjunction with the interpolation routine D02LZF to produce solutions and derivatives at user-specified points. The higher order method is intended for users with high accuracy requirements.

In one-step mode the routine returns approximations to the solution, derivative and f_i at each integration point. In interval mode these values are returned at the end of the integration range. The user selects the order of the method, the mode of operation, the error control and various optional inputs by a prior call of D02LXF.

For a description of the Runge-Kutta-Nystrom formula pairs see Dormand *et al.* [2] and [3] and for a description of their practical implementation see Brankin *et al.* [1].

4. References

[1] BRANKIN, R.W., DORMAND, J.R., GLADWELL, I., PRINCE, P.J., and SEWARD, W.L.
A Runge-Kutta-Nystrom Code.
ACM Trans Math Softw. 15 pp. 31-40, 1989.

[2] DORMAND, J.R., EL-MIKKAWY, M.E.A. and PRINCE, P.J.
Families of Runge-Kutta-Nystrom Formulae.
Teeside Polytechnic Mathematical Report TPMR 86-1, 1986.

[3] DORMAND, J.R., EL-MIKKAWY, M.E.A. and PRINCE, P.J.
High Order Embedded Runge-Kutta-Nystrom Formulae.
Teeside Polytechnic Mathematical Report TPMR 86-2, 1986.

5. Parameters

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

FCN must evaluate the functions f_i (that is the second derivatives y_i'') for given values of its arguments $x, y_1, y_2, ..., y_{\mathrm{NEQ}}$.

Its specification is:

```
SUBROUTINE FCN(NEQ, T, Y, F)
INTEGER    NEQ
real       T,Y(NEQ),F(NEQ)
```

1: NEQ – INTEGER. *Input*

On entry: the number of differential equations.

2: T – ***real***. *Input*

On entry: the value of the argument x.

3: Y(NEQ) – ***real*** array. *Input*

On entry: the value of the argument y_i, for i = 1,2,...,NEQ.

4: F(NEQ) – ***real*** array. *Output*

On exit: the value of f_i, for i = 1,2,...,NEQ.

FCN must be declared as EXTERNAL in the (sub)program from which D02LAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: NEQ – INTEGER. *Input*

On entry: the number of second order ordinary differential equations to be solved by D02LAF. It must contain the same value as the parameter NEQ used in a prior call of D02LXF.

Constraint: NEQ ≥ 1.

3: T – ***real***. *Input/Output*

On entry: the initial value of the independent variable x.

On exit: the value of the independent variable, which is usually TEND, unless an error has occurred or the code is operating in one-step mode. If the integration is to be continued, possibly with a new value for TEND, T must not be changed.

Constraint: T ≠ TEND.

4: TEND – ***real***. *Input*

On entry: the end-point of the range of integration. If TEND < T on initial entry, integration will proceed in the negative direction. TEND may be reset, in the direction of integration, before any continuation call.

5: Y(NEQ) – ***real*** array. *Input/Output*

On entry: the initial values of the solution $y_1, y_2, ..., y_{\mathrm{NEQ}}$.

On exit: the computed values of the solution at the exit value of T. If the integration is to be continued, possibly with a new value for TEND, these values must not be changed.

6: YP(NEQ) – ***real*** array. *Input/Output*

On entry: the initial values of the derivatives $y'_1, y'_2, ..., y'_{\mathrm{NEQ}}$.

On exit: the computed values of the derivatives at the exit value of T. If the integration is to be continued, possibly with a new value for TEND, these values must not be changed.

7: YDP(NEQ) – ***real*** array. *Output*

On exit: the computed values of the second derivative at the exit value of T, unless illegal input is detected, in which case the elements of YDP may not have been initialised. If the integration is to be continued, possibly with a new value for TEND, these values must not be changed.

8: RWORK(LRWORK) – ***real*** array. *Workspace*

This **must** be the same parameter RWORK as supplied to D02LXF. It is used to pass information from D02LXF to D02LAF, and from D02LAF to both D02LYF and D02LZF. Therefore the contents of this array **must not** be changed before the call to D02LAF or calling either of the routines D02LYF and D02LZF.

9: LRWORK – INTEGER. *Input*

On entry: the dimension of the array RWORK as declared in the (sub)program from which D02LAF is called.

This must be the same parameter LRWORK as supplied to D02LXF.

10: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL = 1

Illegal input detected, i.e. one of the following conditions:

on any call, T = TEND, or the value of NEQ or LRWORK has been altered;

on a continuation call, the direction of integration has been changed;

D02LXF had not been called previously, or the previous call of D02LXF resulted in an error exit.

This error exit can be caused if elements of RWORK have been overwritten.

IFAIL = 2

The maximum number of steps has been attempted. (See parameter MAXSTP in D02LXF.) If integration is to be continued then the user need only reset IFAIL and call the routine again and a further MAXSTP steps will be attempted.

IFAIL = 3

In order to satisfy the error requirements, the stepsize needed is too small for the ***machine precision*** being used.

IFAIL = 4

The code has detected two successive error exits at the current value of x and cannot proceed. Check all input variables.

IFAIL = 5

The code has detected inefficient use of the integration method. The step size has been reduced by a significant amount too often in order to hit the output points specified by TEND. (Of the last 100 or more successful steps more than 10% are steps with sizes that have had to be reduced by a factor of greater than a half.)

7. Accuracy

The accuracy of integration is determined by the parameters TOL, THRES and THRESP in a prior call of D02LXF. Note that only the local error at each step is controlled by these parameters. The error estimates obtained are not strict bounds, but they are usually reliable over one step. Over a number of steps the overall error may accumulate in various ways, depending on the system. The code is designed so that a reduction in TOL should lead to an approximately proportional reduction in the error. The user is strongly recommended to call D02LAF with more than one value for TOL and compare the results obtained to estimate their accuracy. The accuracy obtained depends on the type of error test used. If the solution oscillates around zero a relative error test should be avoided, whereas if the solution is exponentially increasing an absolute error test should not be used. For a description of the error test see the specifications of the parameters TOL, THRES and THRESP in routine document D02LXF.

8. Further Comments

If the routine fails with IFAIL = 3 then the value of TOL may be so small that a solution cannot be obtained, in which case the routine should be called again with a larger value for TOL. If the accuracy requested is really needed then the user should consider whether there is a more fundamental difficulty. For example:

(a) in the region of a singularity the solution components will usually be of a large magnitude. D02LAF could be used in one-step mode to monitor the size of the solution with the aim of trapping the solution before the singularity. In any case numerical integration cannot be continued through a singularity, and analytical treatment may be necessary;

(b) if the solution contains fast oscillatory components, the routine will require a very small stepsize to preserve stability. This will usually be exhibited by excessive computing time and sometimes an error exit with IFAIL = 3. The Runge-Kutta-Nystrom methods are not efficient in such cases and the user should consider re-posing his problem as a system of first order ordinary differential equations and then using a routine from the D02M-D02N subchapter with the Blend formulae (see D02NWF).

D02LAF can be used for producing results at short intervals (for example, for tabulation), in two ways. By far the less efficient is to call D02LAF successively over short intervals, $t + (i-1)\times h$ to $t + i\times h$, although this is the only way if the higher order method has been selected and precisely **not** what it is intended for. A more efficient way, **only** for use when the lower order method has been selected, is to use D02LAF in one-step mode. The output values of parameters Y,YP,YDP,T and RWORK are set correctly for a call of D02LZF to compute the solution and derivative at the required points.

9. Example

We solve the following system (the two body problem)

$$y_1'' = -y_1/\left(y_1^2+y_2^2\right)^{3/2}$$

$$y_2'' = -y_2/\left(y_1^2+y_2^2\right)^{3/2}$$

over the range [0,20] with initial conditions $y_1 = 1.0 - \varepsilon$, $y_2 = 0.0$, $y_1' = 0.0$ and $y_2' = \sqrt{\left(\frac{1+\varepsilon}{1-\varepsilon}\right)}$ where ε, the eccentricity, is 0.5. The system is solved using the lower order method with relative local error tolerances 1.0E–4 and 1.0E–5 and default threshold tolerances. D02LAF is used in one-step mode (ONESTP = .TRUE.) and D02LZF provides solution values at intervals of 2.0.

9.1. Program Text

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

```
*     D02LAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER           NOUT
      PARAMETER         (NOUT=6)
      INTEGER           NEQ, LRWORK, NWANT
      PARAMETER         (NEQ=2,LRWORK=16+20*NEQ,NWANT=NEQ)
*     .. Local Scalars ..
      real              ECC, H, HNEXT, HSTART, HUSED, T, TEND, TINC,
     +                  TNEXT, TOL, TSTART, Y1, Y2, YP1, YP2
      INTEGER           IFAIL, ITOL, K, MAXSTP, NATT, NFAIL, NSUCC
      LOGICAL           HIGH, ONESTP, START
*     .. Local Arrays ..
      real              RWORK(LRWORK), THRES(NEQ), THRESP(NEQ), Y(NEQ),
     +                  YDP(NEQ), YP(NEQ), YPWANT(NWANT), YWANT(NWANT)
*     .. External Subroutines ..
      EXTERNAL          D02LAF, D02LXF, D02LYF, D02LZF, FCN2BD
*     .. Intrinsic Functions ..
      INTRINSIC         SQRT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02LAF Example Program Results'
      HIGH = .FALSE.
      ONESTP = .TRUE.
      TINC = 2.0e0
*
*     Initial conditions
*
      TSTART = 0.0e0
      ECC = 0.5e0
      Y1 = 1.0e0 - ECC
      Y2 = 0.0e0
      YP1 = 0.0e0
      YP2 = SQRT((1.0e0+ECC)/(1.0e0-ECC))
      TEND = 20.0e0
*
      DO 60 ITOL = 4, 5
         TOL = 10.0e0**(-ITOL)
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Calculation with TOL = ', TOL
         WRITE (NOUT,*)
         WRITE (NOUT,*) '   T      Y(1)       Y(2)'
*
*        Call D02LXF with default THRES,THRESP,MAXSTP and H
*
         THRES(1) = 0.0e0
         THRESP(1) = 0.0e0
         H = 0.0e0
         MAXSTP = 0
         START = .TRUE.
         IFAIL = 0
*
         CALL D02LXF(NEQ,H,TOL,THRES,THRESP,MAXSTP,START,ONESTP,HIGH,
     +               RWORK,LRWORK,IFAIL)
*
*        Set initial values
*
         Y(1) = Y1
         Y(2) = Y2
         YP(1) = YP1
         YP(2) = YP2
         T = TSTART
         TNEXT = T + TINC
         WRITE (NOUT,99998) T, (Y(K),K=1,NEQ)
*
```

```
*         Loop point for onestep mode
*
   20     IFAIL = -1
*
          CALL D02LAF(FCN2BD,NEQ,T,TEND,Y,YP,YDP,RWORK,LRWORK,IFAIL)
*
          IF (IFAIL.GT.0) THEN
             WRITE (NOUT,*)
             WRITE (NOUT,99997) 'D02LAF returned with IFAIL = ', IFAIL,
     +         ' at T = ', T
             STOP
          END IF
*
*         Loop point for interpolation
*
   40     IF (TNEXT.LE.T) THEN
             IFAIL = 0
*
             CALL D02LZF(NEQ,T,Y,YP,NEQ,TNEXT,YWANT,YPWANT,RWORK,LRWORK,
     +                   IFAIL)
*
             WRITE (NOUT,99998) TNEXT, (YWANT(K),K=1,NEQ)
             TNEXT = TNEXT + TINC
             GO TO 40
          END IF
*
          IF (T.LT.TEND) GO TO 20
*
          IFAIL = 0
*
          CALL D02LYF(NEQ,HNEXT,HUSED,HSTART,NSUCC,NFAIL,NATT,THRES,
     +                THRESP,RWORK,LRWORK,IFAIL)
*
          WRITE (NOUT,*)
          WRITE (NOUT,99996) ' Number of successful steps = ', NSUCC
          WRITE (NOUT,99996) ' Number of   failed   steps = ', NFAIL
   60 CONTINUE
      STOP
*
99999 FORMAT (1X,A,1P,e9.1)
99998 FORMAT (1X,F5.1,2(2X,F9.5))
99997 FORMAT (1X,A,I2,A,1P,e10.3)
99996 FORMAT (1X,A,I5)
      END
*
      SUBROUTINE FCN2BD(NEQ,T,Y,YDP)
*
*     Derivatives for two body problem in  y'' = f(t,y)  form
*
*     .. Scalar Arguments ..
      real              T
      INTEGER           NEQ
*     .. Array Arguments ..
      real              Y(NEQ), YDP(NEQ)
*     .. Local Scalars ..
      real              R
*     .. Intrinsic Functions ..
      INTRINSIC         SQRT
*     .. Executable Statements ..
      R = SQRT(Y(1)**2+Y(2)**2)**3
      YDP(1) = -Y(1)/R
      YDP(2) = -Y(2)/R
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D02LAF Example Program Results

Calculation with TOL =   1.0E-04

   T       Y(1)        Y(2)
  0.0    0.50000     0.00000
  2.0   -1.20573     0.61357
  4.0   -1.33476    -0.47685
  6.0    0.35748    -0.44558
  8.0   -1.03762     0.73022
 10.0   -1.42617    -0.32658
 12.0    0.05515    -0.72032
 14.0   -0.82880     0.81788
 16.0   -1.48103    -0.16788
 18.0   -0.26719    -0.84223
 20.0   -0.57803     0.86339

 Number of successful steps =   108
 Number of   failed   steps =    16

Calculation with TOL =   1.0E-05

   T       Y(1)        Y(2)
  0.0    0.50000     0.00000
  2.0   -1.20573     0.61357
  4.0   -1.33476    -0.47685
  6.0    0.35748    -0.44558
  8.0   -1.03762     0.73022
 10.0   -1.42617    -0.32658
 12.0    0.05516    -0.72031
 14.0   -0.82880     0.81787
 16.0   -1.48103    -0.16789
 18.0   -0.26718    -0.84223
 20.0   -0.57804     0.86338

 Number of successful steps =   169
 Number of   failed   steps =    15
```

D02LXF – NAG Fortran Library Routine Document

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

1. Purpose

D02LXF is a setup routine which must be called by the user prior to the first call of the integrator D02LAF and may be called prior to any further call to D02LAF.

2. Specification

```
      SUBROUTINE D02LXF (NEQ, H, TOL, THRES, THRESP, MAXSTP, START,
     1                   ONESTP, HIGH, RWORK, LRWORK, IFAIL)
      INTEGER          NEQ, MAXSTP, LRWORK, IFAIL
      real             H, TOL, THRES(NEQ), THRESP(NEQ), RWORK(LRWORK)
      LOGICAL          START, ONESTP, HIGH
```

3. Description

This routine permits the user to set optional inputs prior to any call of D02LAF. It must be called before the first call of routine D02LAF and it may be called before any continuation call of routine D02LAF.

4. References

None.

5. Parameters

1: NEQ – INTEGER. *Input*

On entry: the number of second order ordinary differential equations to be solved by D02LAF.

Constraint: NEQ ≥ 1.

2: H – ***real***. *Input*

On entry: if START = .TRUE., H may specify an initial step size to be attempted in D02LAF.

If START = .FALSE., then H may specify a step size to override the choice of next step attempted made internally to D02LAF.

The sign of H is not important, as the absolute value of H is chosen and the appropriate sign is selected by D02LAF.

If this option is not required then the user must set H = 0.0.

3: TOL – ***real***. *Input*

On entry: TOL must be set to a relative tolerance for controlling the error in the integration by D02LAF. D02LAF has been designed so that, for most problems, a reduction in TOL leads to an approximately proportional reduction in the error in the solution. However the actual relation between TOL and the accuracy of the solution cannot be guaranteed. The user is strongly recommended to repeat the integration with a smaller value of TOL and compare the results. See the description of THRES and THRESP for further details of how TOL is used.

Constraint: $10 \times \varepsilon \le$ TOL ≤ 1.0 (ε is the ***machine precision***, see X02AJF).

4: THRES(NEQ) – ***real*** array. *Input*

5: THRESP(NEQ) – ***real*** array. *Input*

On entry: THRES and THRESP may be set to thresholds for use in the error control of D02LAF. At each step in the numerical integration estimates of the local errors E1(i) and E2(i) in the solution, y_i, and its derivative, y_i', respectively are computed, for i = 1,2,...NEQ. For the step to be accepted conditions of the following type must be satisfied

$$\max_{1 \le i \le \mathrm{NEQ}} \left(\frac{\mathrm{E1}(i)}{\max(\mathrm{THRES}(i), |y_i|)} \right) \le \mathrm{TOL},$$

$$\max_{1 \le i \le \mathrm{NEQ}} \left(\frac{\mathrm{E2}(i)}{\max(\mathrm{THRESP}(i), |y_i'|)} \right) \le \mathrm{TOL}.$$

If one or both of these is not satisfied then the step size is reduced and the solution is recomputed.

If THRES(1) ≤ 0.0 on entry, then a value of 50.0×ε is used for THRES(i), for i = 1,2,...,NEQ, where ε is ***machine precision***. Similarly for THRESP.

Constraints: THRES(1) ≤ 0.0 or THRES(i) > 0.0, for i = 1,2,...,NEQ,
THRESP(1) ≤ 0.0 or THRESP(i) > 0.0, for i = 1,2,...,NEQ.

6: MAXSTP – INTEGER. *Input*

On entry: a bound on the number of steps attempted in any one call of D02LAF.

If MAXSTP ≤ 0 on entry, a value of 1000 is used.

7: START – LOGICAL. *Input/Output*

On entry: specifies whether or not the call of D02LAF is for a new problem. START = .TRUE. indicates that a new problem is to be solved. START = .FALSE. indicates the call of D02LXF is prior to a continuation call of D02LAF.

On exit: START = .FALSE..

8: ONESTP – LOGICAL. *Input*

On entry: the mode of operation for D02LAF. If ONESTP = .TRUE. D02LAF will operate in one-step mode, that is it will return after each succesful step. If ONESTP = .FALSE., D02LAF will operate in interval mode, that is it will return at the end of the integration interval.

9: HIGH – LOGICAL. *Input*

On entry: if HIGH = .TRUE., a high order method will be used, whereas if HIGH = .FALSE., a low order method will be used. (See the specification of D02LAF for further details.)

10: RWORK(LRWORK) – ***real*** array. *Workspace*

This **must** be the same parameter RWORK supplied to D02LAF. It is used to pass information to D02LAF and therefore the contents of this array **must not** be changed before calling D02LAF.

11: LRWORK – INTEGER. *Input*

On entry: the dimension of the array RWORK as declared in the (sub)program from which D02LXF is called.

Constraints: LRWORK ≥ 16 + 20×NEQ if HIGH = .TRUE.,
LRWORK ≥ 16 + 11×NEQ if HIGH = .FALSE..

12: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

THRES(1) > 0.0 and for some i THRES(i) ≤ 0.0, 1 ≤ i ≤ NEQ,
and/or,
THRESP(1) > 0.0 and for some i THRESP(i) ≤ 0.0, 1 ≤ i ≤ NEQ.

IFAIL = 2

LRWORK is too small.

IFAIL = 3

TOL does not satisfy $10\times\varepsilon \leq \text{TOL} \leq 1.0$

7. Accuracy

Not applicable.

8. Further Comments

Prior to a continuation call of D02LAF, the user may reset any of the optional parameters by calling D02LXF with START = .FALSE..

The user may reset:

(a) H	– to override the internal step size selection;
(b) TOL,THRES,THRESP	– to change the error requirements;
(c) MAXSTP	– to increase or decrease the number of steps attempted before an error exit is returned;
(d) ONESTP	– to change the mode of operation of D02LAF;
(e) HIGH	– to change the order of the method being used.

9. Example

See example program for D02LAF.

D02LYF – NAG Fortran Library Routine Document

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

1. Purpose

D02LYF is a diagnostic routine which may be called after a call of the integrator D02LAF.

2. Specification

```
      SUBROUTINE D02LYF (NEQ, HNEXT, HUSED, HSTART, NSUCC, NFAIL, NATT,
     1                   THRES, THRESP, RWORK, LRWORK, IFAIL)
      INTEGER          NEQ, NSUCC, NFAIL, NATT, LRWORK, IFAIL
      real             HNEXT, HUSED, HSTART, THRES(NEQ), THRESP(NEQ),
     1                 RWORK(LRWORK)
```

3. Description

This routine permits the user to extract information about the performance of D02LAF and the setting of some optional parameters. It may be called only after a call of D02LAF.

4. References

None.

5. Parameters

1: NEQ – INTEGER. *Input*

On entry: the number of second order ordinary differential equations solved by D02LAF. It must be the same as the parameter NEQ supplied to D02LXF and D02LAF.

2: HNEXT – ***real***. *Output*

On exit: the next step size which D02LAF, if called, would attempt.

3: HUSED – ***real***. *Output*

On exit: the last successful step size used by D02LAF.

4: HSTART – ***real***. *Output*

On exit: the initial step size used on the current integration problem by D02LAF.

5: NSUCC – INTEGER. *Output*

On exit: the number of steps attempted by D02LAF that have been successful since the start of the current problem.

6: NFAIL – INTEGER. *Output*

On exit: the number of steps attempted by D02LAF that have failed since the start of the current problem.

7: NATT – INTEGER. *Output*

On exit: the number of steps attempted before the initial step was successful. Over a large number of problems the cost of an attempted step of this type is approximately half that of a normal attempted step.

8: THRES(NEQ) – ***real*** array. *Output*

On exit: the ith solution threshold value used in the error control strategy. (See D02LXF.)

9: THRESP(NEQ) – ***real*** array. *Output*

On exit: the *i*th derivative threshold value used in the error control strategy. (See D02LXF.)

10: RWORK(LRWORK) – ***real*** array. *Workspace*

This **must** be the same parameter RWORK as supplied to D02LAF. It is used to pass information from D02LAF to D02LYF and therefore the contents of this array **must not** be changed before calling D02LYF.

11: LRWORK – INTEGER. *Input*

On entry: the dimension of the array RWORK as declared in the (sub)program from which D02LYF is called.

This must be the same parameter LRWORK as supplied to D02LXF.

12: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

D02LAF has not been called, or one or both of the parameters NEQ and LRWORK does not match the corresponding parameter supplied to D02LXF.

This error exit can be caused if elements of RWORK have been overwritten.

7. Accuracy

Not applicable.

8. Further Comments

None.

9. Example

See the example for D02LAF.

D02LZF – NAG Fortran Library Routine Document

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

1. Purpose

D02LZF interpolates components of the solution of a non-stiff system of second order differential equations from information provided by the integrator D02LAF, when the low order method has been used.

2. Specification

```
      SUBROUTINE D02LZF (NEQ, T, Y, YP, NWANT, TWANT, YWANT, YPWANT,
     1                   RWORK, LRWORK, IFAIL)
      INTEGER          NEQ, NWANT, LRWORK, IFAIL
      real             T, Y(NEQ), YP(NEQ), TWANT, YWANT(NWANT),
     1                 YPWANT(NWANT), RWORK(LRWORK)
```

3. Description

D02LZF evaluates the first NWANT (≤ NEQ) components of the solution of a non-stiff system of second order ordinary differential equations at any point using a special Runge-Kutta-Nystrom formula (Dormand and Prince [1]) and information generated by D02LAF when the low order method has been used. This information must be presented unchanged to D02LZF. D02LZF should not normally be used to extrapolate outside the range of the values from D02LAF.

4. References

[1] DORMAND, J.R. and PRINCE, P.J.
Runge-Kutta-Nystrom Triples.
Teeside Polytechnic Computer Science Report TP-CS-86-05, 1986.

5. Parameters

1: NEQ – INTEGER. *Input*

On entry: the number of second order ordinary differential equations being solved by D02LAF. It must contain the same value as the parameter NEQ in a prior call of D02LAF.

2: T – ***real***. *Input*

On entry: the current value, t, at which the solution and its derivative have been computed (as returned in parameter T on output from D02LAF).

3: Y(NEQ) – ***real*** array. *Input*

On entry: the ith component of the solution at t, for i = 1,2,...,NEQ, as returned from D02LAF.

4: YP(NEQ) – ***real*** array. *Input*

On entry: the ith component of the derivative at t, for i = 1,2,...,NEQ, as returned from D02LAF.

5: NWANT – INTEGER. *Input*

On entry: the number of components of the solution and derivative whose values at TWANT are required. The first NWANT components are evaluated.

Constraint: 1 ≤ NWANT ≤ NEQ.

6: TWANT – ***real***. *Input*

On entry: the point at which components of the solution and derivative are to be evaluated. TWANT should not normally be an extrapolation point, that is TWANT should satisfy

TOLD ≤ TWANT ≤ T,

or if integration is proceeding in the negative direction

TOLD ≥ TWANT ≥ T,

where TOLD is the previous integration point which is held in an element of the array RWORK and is, to within rounding, T – HUSED. (HUSED is given by D02LYF.) Extrapolation is permitted but not recommended, and an IFAIL value of 2 is returned whenever extrapolation is attempted.

7: YWANT(NWANT) – ***real*** array. *Output*

On exit: the calculated value of the ith component of the solution at t = TWANT, for i = 1,2,...,NWANT.

8: YPWANT(NWANT) – ***real*** array. *Output*

On exit: the calculated value of the ith component of the derivative at t = TWANT, for i = 1,2,...,NWANT.

9: RWORK(LRWORK) – ***real*** array. *Workspace*

This **must** be the same parameter RWORK as supplied to D02LAF. It is used to pass information from D02LAF to D02LZF and therefore the contents of this array **must not** be changed before calling D02LZF.

10: LRWORK – INTEGER. *Input*

On entry: the dimension of the array RWORK as declared in the (sub)program from which D02LZF is called.

This must be the same parameter LRWORK as supplied to the setup routine D02LXF.

11: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

If D02LZF is to be used for extrapolation at TWANT, IFAIL should be set to 1 before entry. It is then essential to test the value of IFAIL on exit.

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

Illegal input detected, i.e. one of the following conditions:

D02LAF has not been called;

one or both of the parameters NEQ and LRWORK does not match the corresponding parameter supplied to the setup routine D02LXF;

no integration steps have been taken since the last call to D02LXF with START = .TRUE.;

NWANT < 1, or NWANT > NEQ.

This error exit can be caused if elements of RWORK have been overwritten.

IFAIL = 2

D02LZF has been called for extrapolation. The values of the solution and its derivative at TWANT have been calculated and placed in YWANT and YPWANT before returning with this error number (see Section 7).

IFAIL = 3

D02LAF last used the high order method to integrate the system of differential equations. Interpolation is not permitted with this method.

7. Accuracy

The error in interpolation is of a similar order to the error arising from the integration using D02LAF with the lower order method.

The same order of accuracy can be expected when extrapolating using D02LZF. However, the actual error in extrapolation will, in general, be much larger than for interpolation.

8. Further Comments

When interpolation for only a few components is required then it is more efficient to order the components of interest so that they are numbered first.

9. Example

See example program for D02LAF.

Subchapter D02M-D02N

Integrators for Stiff Ordinary Differential Equations

Chapter D02M/N

Integrators for Stiff Ordinary Differential Systems

1 Introduction

This subchapter contains the specifications of the integrators, the setup routines and diagnostic routines which have been developed from the SPRINT package, Berzins and Furzeland [1].

The integrators D02NBF, D02NCF and D02NDF are designed for solving stiff systems of explicitly defined ordinary differential equations,

$$y' = g(t,y).$$

The integrators D02NGF, D02NHF and D02NJF are designed for solving stiff systems of implicitly defined ordinary differential equations,

$$A(t,y)y' = g(t,y).$$

This formulation permits solution of differential/algebraic systems (DAEs). The facilities provided are essentially those of the explicit solvers.

The integrator routines have almost identical calling sequences but each is designed to solve a problem where the Jacobian is of a particular structure: full matrix (D02NBF and D02NGF), banded matrix (D02NCF and D02NHF) or sparse matrix (D02NDF and D02NJF). Each of these structures has associated with it a linear algebra setup routine: D02NSF, D02NTF and D02NUF respectively. A linear algebra setup routine must be called before the first call to the appropriate integrator. These linear algebra setup routines check various parameters of the corresponding integrator routine and set certain parameters for the linear algebra computations. A routine, D02NXF, is supplied which permits extraction of diagnostic information after a call to either of the sparse linear algebra solvers D02NDF and D02NJF.

With the integrators are also associated three integrator setup routines D02NVF, D02NWF and D02MVF, one of which must be called before the first call to any integrator routine. They provide input to the Backward Differentiation Formulae (BDF), the Blend Formulae and the special fixed leading coefficient BDF codes respectively. On return from an integrator, if it is feasible to continue the integration, D02NZF may be called to reset various integration parameters. It is often of considerable interest to determine statistics concerning the integration process. D02NYF is provided with this aim in mind. It should prove especially useful to those who wish to integrate many similar problems as it provides suitable values for many of the input parameters and indications of the difficulties encountered when solving the problem.

Hence, the general form of a program calling one of the integrator routines D02NBF, D02NCF, D02NDF, D02NGF, D02NHF or D02NJF will be

```
declarations
    .
    .
call linear algebra setup routine
call integrator setup routine
call integrator
call integrator diagnostic routine (if required)
call linear algebra diagnostic routine (if appropriate and if required)
    .
    .
STOP
END
```

The required calling sequence for different Jacobian structures and system types is represented diagramatically in Figure 1.

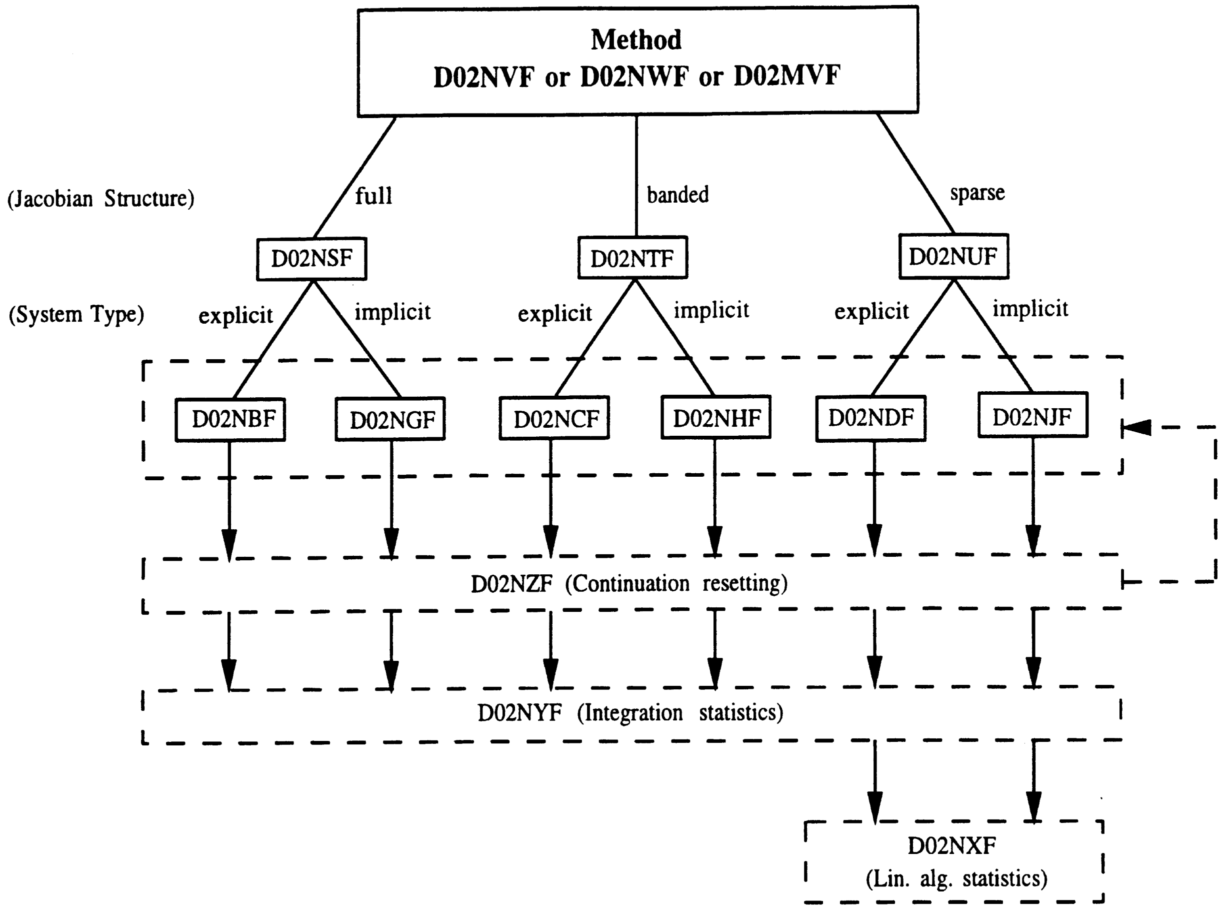


Figure 1
Schema for forward communication routine calling sequences

The integrators D02NMF and D02NNF are reverse communication routines designed for solving explicit and implicit stiff ordinary differential systems respectively. Users are warned that they should use these routines only when the integrators mentioned above are inadequate for their application. For example, if it is difficult to write one or more of the subroutines FCN (RESID) or JAC (or MONITR) or if the integrators are to be embedded in a package, it may be advisable to consider these routines.

Since these routines use reverse communication the user need define no EXTERNAL parameters. This makes them especially suitable for large scale computations where encapsulation of the definition of the differential system or its Jacobian matrix in subroutine form may be particularly difficult to achieve.

D02NMF is the reverse communication counterpart of the forward communication routines D02NBF, D02NCF and D02NDF whereas D02NNF is the reverse communication counterpart of the forward communication routines D02NGF, D02NHF and D02NJF. When using these reverse communication routines it is necessary to call the same linear algebra and integrator setup routines as for the forward communication counterpart. All the other continuation and interrogation routines available for use with the forward communication routines are also available to the user when calling the reverse communication routines.

There is also a routine, D02NRF, to inform the user how to supply the Jacobian when the sparse linear algebra option is being employed with either of D02NMF and D02NNF.

Hence, the general form of a program calling one of the integrator routines D02NMF or D02NNF will be

```
      declarations
      call linear algebra setup routine
      call integrator setup routine
      IREVCM = 0
 1000 call integrator( ..., IREVCM, ...)
      IF (IREVCM.GT.0) THEN

      evaluate residual and Jacobian (including a call to D02NRF if
      sparse linear algebra is being used), call the MONITR routine etc.

      GO TO 1000
      ENDIF

      call integrator diagnostic routine (if required)
      call linear algebra diagnostic routine (if appropriate and if required)
      STOP
      END
```

The required calling sequence in the case of reverse communication, is represented diagramatically in Figure 2.

In the example programs for the eight integrators D02NBF, D02NCF, D02NDF, D02NGF, D02NHF, D02NJF, D02NMF and D02NNF we attempt to illustrate the various options available. Many of these options are available in all the routines and the user is invited to scan all the example programs for illustrations of their use. In each case we use as an example the stiff Robertson problem

$$\begin{aligned} a' &= -0.04a + 10^4bc \\ b' &= 0.04a - 10^4bc - 3\times 10^7b^2 \\ c' &= 3\times 10^7b^2 \end{aligned}$$

despite the fact that it is not a sensible choice to use either the banded or the sparse linear algebra for this problem. Their use here serves for illustration of the techniques involved. For the implicit integrators D02NGF, D02NHF and D02NJF we write the Robertson problem in residual form, as an implicit differential system and as a differential/algebraic system respectively. Here we are exploiting the fact that $a+b+c$ is constant and hence one of the equations may be replaced by $(a+b+c)'=0.0$ or $a+b+c=1.0$ (for our particular choice of initial conditions). For the reverse communication routines D02NMF and D02NNF our examples are intended only to illustrate the reverse communication technique.

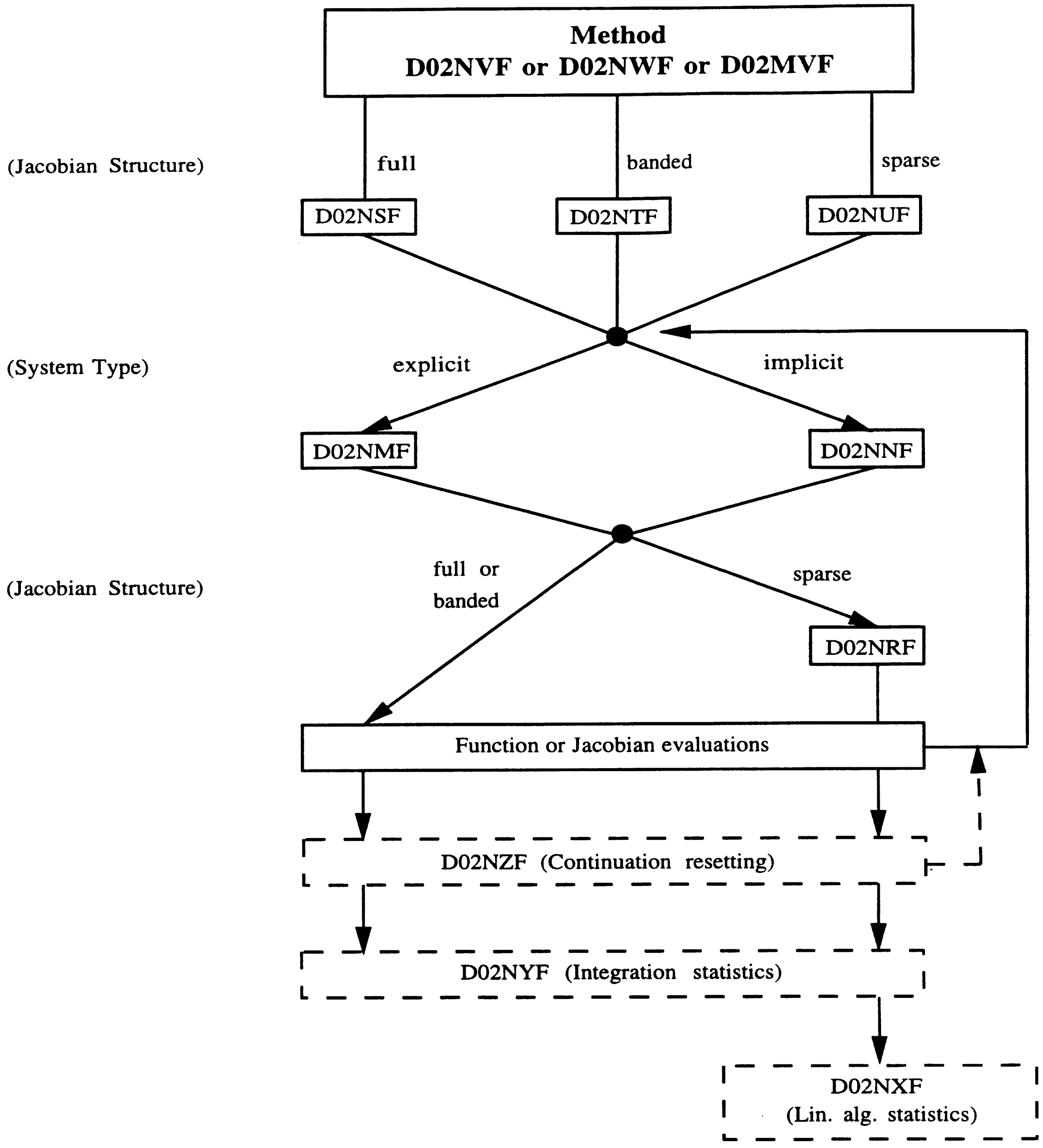


Figure 2
Schema for reverse communication routine calling sequences

2 References

[1] Berzins M and Furzeland R M (1985) A user's manual for SPRINT – A versatile software package for solving systems of algebraic, ordinary and partial differential equations: Part 1 – Algebraic and ordinary differential equations *Report TNER.85.085* Shell Research Limited

D02MVF – NAG Fortran Library Routine Document

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

1. Purpose

D02MVF is a setup routine which must be called by the user, prior to an integrator in the subchapter D02M-D02N, if the DASSL implementation of Backward Differentiation Formulae (BDF) is to be used.

2. Specification

```
      SUBROUTINE D02MVF (NEQMAX, NY2DIM, MAXORD, CONST, TCRIT, HMIN,
     1                   HMAX, H0, MAXSTP, MXHNIL, NORM, RWORK,
     2                   IFAIL)
      INTEGER          NEQMAX, NY2DIM, MAXORD, MAXSTP, MXHNIL, IFAIL
      real             CONST(3), TCRIT, HMIN, HMAX, H0,
     1                 RWORK(50+4*NEQMAX)
      CHARACTER*1      NORM
```

3. Description

An integrator setup routine must be called before the call to any integrator in this subchapter. The setup routine D02MVF makes the choice of the DASSL integrator and permits the user to define options appropriate to this choice.

4. References

None.

5. Parameters

1: NEQMAX – INTEGER. *Input*

On entry: a bound on the maximum number of differential equations to be solved.

Constraint: NEQMAX ≥ 1.

2: NY2DIM – INTEGER. *Input*

On entry: the second dimension of the array YSAVE that will be supplied to the integrator, as declared in the (sub)program from which the integrator is called.

Constraint: NY2DIM ≥ MAXORD + 3.

3: MAXORD – INTEGER. *Input*

On entry: the maximum order to be used for the BDF method. If MAXORD = 0 then MAXORD = 5 is assumed.

Constraint: 0 ≤ MAXORD ≤ 5.

4: CONST(3) – ***real*** array. *Input/Output*

On entry: values to be used to control step size choice during integration. If any CONST(i) = 0.0 on entry, it is replaced by its default value described below. In most cases this is the recommended setting.

CONST(1), CONST(2), and CONST(3) are factors used to bound step size changes. If the current step size h fails, then the modulus of the next step size is bounded by CONST(1)$*|h|$. The default value of CONST(1) is 2.0. Note that the new step size may be used with a method of different order to the failed step. If the initial step size is h, then the modulus of the step size on the second step is bounded by CONST(3)$*|h|$. At any other stage in the integration, if the current step size is h, then the modulus of the next step size is bounded by CONST(2)$*|h|$. The default values are 10.0 for CONST(2) and 1000.0 for CONST(3).

Constraints: the following constraints must be satisfied after any zero values have been replaced by default values:

0.0 < CONST(1) < CONST(2) < CONST(3),
CONST(2) > 1.0,
CONST(3) > 1.0.

On exit: the values actually used by the routine.

5: TCRIT – ***real***. *Input*

On entry: a point beyond which integration must not be attempted. The use of TCRIT is described under the parameter ITASK in the specification for the integrator. A value, 0.0 say, must be specified even if ITASK subsequently specifies that TCRIT will not be used.

6: HMIN – ***real***. *Input*

On entry: the minimum absolute step size to be allowed. Set HMIN = 0.0 if this option is not required.

7: HMAX – ***real***. *Input*

On entry: the maximum absolute step size to be allowed. Set HMAX = 0.0 if this option is not required.

8: H0 – ***real***. *Input*

On entry: the step size to be attempted on the first step. Set H0 = 0.0 if the initial step size is to be calculated internally.

9: MAXSTP – INTEGER. *Input*

On entry: the maximum number of steps to be attempted during one call to the integrator after which it will return with IFAIL = 2. Set MAXSTP = 0 if no limit is to be imposed.

10: MXHNIL – INTEGER. *Input*

On entry: the maximum number of warnings printed (if ITRACE ≥ 0) per problem when $t + h = t$ on a step (h = current step size). If MXHNIL ≤ 0, a default value of 10 is assumed.

11: NORM – CHARACTER*1. *Input*

On entry: indicates the type of norm to be used. Three options are available:

'M' maximum norm,
'A' averaged L2 norm,
'D' is the same as 'A'.

If VNORM denotes the norm of the vector v of length n, then for the averaged L2 norm

$$\text{VNORM} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{v_i}{w_i}\right)^2},$$

while for the maximum norm

$$\text{VNORM} = \max_{1\le i\le n}\left|\frac{v_i}{w_i}\right|.$$

If the user wishes to weight the maximum norm or the L2 norm, then RTOL and ATOL should be scaled appropriately on input to the integrator (see under ITOL in the specification of the integrator for the formulation of the weight vector w_i from RTOL and ATOL).

Only the first character to the actual argument NORM is passed to D02MVF; hence it is permissible for the actual argument to be more descriptive, e.g. 'Maximum', 'Average L2' or 'Default' in a call to D02MVF.

Constraint: NORM must be one of 'M', 'A' or 'D'.

12: RWORK(50+4*NEQMAX) – ***real*** array. *Workspace*

This must be the same workspace array as the array RWORK supplied to the integrator. It is used to pass information from the setup routine to the integrator and therefore the contents of this array must not be changed before calling the integrator.

13: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, NEQMAX < 1,
or NY2DIM < MAXORD + 3,
or MAXORD < 0,
or MAXORD > 5,
or invalid value for element of the array CONST,
or NORM ≠ 'M', 'A' or 'D'.

7. Accuracy

Not applicable.

8. Further Comments

None.

9. Example

We solve the plane pendulum problem defined by the equations:

$$\begin{aligned} x' &= u \\ y' &= v \\ u' &= -\lambda x \\ v' &= -\lambda y - 1 \\ x^2+y^2 &= 1. \end{aligned}$$

The additional algebraic contraint $xu + yv = 0$ can be derived, and after appropriate substitution and manipulation to avoid a singular Jacobian we solve the equations:

$$\begin{aligned} y_1{}' &= y_3 - y_6 y_1 \\ y_2{}' &= y_4 - y_6 y_2 \\ y_3{}' &= -y_5 y_1 \\ y_4{}' &= -y_5 y_2 - 1 \\ 0 &= y_1 y_3 + y_2 y_4 \\ 0 &= y_1^2 + y_2^2 - 1 \end{aligned}$$

with given initial conditions and derivatives.

9.1. Program Text

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

```
*     D02MVF Example Program Text
*     Mark 14 Release.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, NEQMAX, NRW, NINF, NWKJAC, MAXORD, NY2DIM,
     +                 MAXSTP, MXHNIL
      PARAMETER        (NEQ=6,NEQMAX=NEQ,NRW=50+4*NEQMAX,NINF=23,
     +                 NWKJAC=NEQMAX*(NEQMAX+1),MAXORD=5,
     +                 NY2DIM=MAXORD+3,MAXSTP=5000,MXHNIL=5)
      real             H0, HMAX, HMIN
      PARAMETER        (H0=0.0e0,HMAX=0.0e0,HMIN=1.0e-10)
*     .. Local Scalars ..
      real             DUM, PI, T, TCRIT, TOUT
      INTEGER          I, IFAIL, ITASK, ITOL, ITRACE, MAXOD1
*     .. Local Arrays ..
      real             ATOL(NEQMAX), CONST(3), RTOL(NEQMAX), RWORK(NRW),
     +                 WKJAC(NWKJAC), Y(NEQMAX), YDOT(NEQMAX),
     +                 YSAVE(NEQMAX,NY2DIM)
      INTEGER          INFORM(NINF)
      LOGICAL          LDERIV(2)
*     .. External Functions ..
      real             X01AAF
      EXTERNAL         X01AAF
*     .. External Subroutines ..
      EXTERNAL         D02MVF, D02NBY, D02NGF, D02NSF, DAEJAC, DAERES,
     +                 X04ABF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02MVF Example Program Results'
      CALL X04ABF(1,NOUT)
      DO 20 I = 1, 3
         CONST(I) = 0.0e0
   20 CONTINUE
      ITRACE = 0
      PI = X01AAF(DUM)
      RTOL(1) = 1.0e-3
      ATOL(1) = 1.0e-6
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'Pendulum problem with relative tolerance = ',
     +  RTOL(1)
      WRITE (NOUT,99999) '                  and absolute tolerance = ',
     +  ATOL(1)
      ITOL = 1
      T = 0.0e0
      TOUT = PI
*     Set initial values and derivatives
      Y(1) = 1.e0
      Y(2) = 0.0e0
      Y(3) = 0.e0
      Y(4) = 0.e0
      Y(5) = 0.0e0
      Y(6) = 0.0e0
      YDOT(1) = Y(3) - Y(6)*Y(1)
      YDOT(2) = Y(4) - Y(6)*Y(2)
      YDOT(3) = -Y(5)*Y(1)
      YDOT(4) = -Y(5)*Y(2) - 1.e0
      YDOT(5) = -3.e0*Y(4)
      YDOT(6) = 0.0e0
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     + '   t          y1      y2      y3      y4      y5      y6'
      WRITE (NOUT,99998) T, (Y(I),I=1,6)
      ITASK = 4
      TCRIT = TOUT
      IFAIL = 0
      MAXOD1 = 0
*
      CALL D02MVF(NEQMAX,NY2DIM,MAXOD1,CONST,TCRIT,HMIN,HMAX,H0,MAXSTP,
     +            MXHNIL,'AVERAGE-L2',RWORK,IFAIL)
*
      CALL D02NSF(NEQ,NEQMAX,'ANALYTIC',NWKJAC,RWORK,IFAIL)
*
      IFAIL = -1
      LDERIV(1) = .TRUE.
      LDERIV(2) = .TRUE.
*
      CALL D02NGF(NEQ,NEQMAX,T,TOUT,Y,YDOT,RWORK,RTOL,ATOL,ITOL,INFORM,
     +            DAERES,YSAVE,NY2DIM,DAEJAC,WKJAC,NWKJAC,D02NBY,LDERIV,
     +            ITASK,ITRACE,IFAIL)
*
      IF (IFAIL.NE.0) THEN
         WRITE (NOUT,99997) 'Integration terminated with IFAIL = ',
     +     IFAIL, ' at T = ', T
      ELSE
         WRITE (NOUT,99998) T, (Y(I),I=1,NEQ)
      END IF
      STOP
*
99999 FORMAT (1X,A,1P,e7.1)
99998 FORMAT (1X,F6.4,3X,6(F8.4))
99997 FORMAT (1X,A,I2,A,F6.4)
      END
      SUBROUTINE DAERES(NEQ,T,Y,YDOT,R,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IRES, NEQ
*     .. Array Arguments ..
      real              R(NEQ), Y(NEQ), YDOT(NEQ)
*     .. Executable Statements ..
      IF (IRES.EQ.-1) THEN
         R(1) = -YDOT(1)
         R(2) = -YDOT(2)
         R(3) = -YDOT(3)
         R(4) = -YDOT(4)
         R(5) = 0.0e0
         R(6) = 0.0e0
      ELSE
         R(1) = Y(3) - Y(6)*Y(1) - YDOT(1)
         R(2) = Y(4) - Y(6)*Y(2) - YDOT(2)
         R(3) = -Y(5)*Y(1) - YDOT(3)
         R(4) = -Y(5)*Y(2) - 1.e0 - YDOT(4)
         R(5) = Y(1)*Y(3) + Y(2)*Y(4)
         R(6) = Y(1)**2 + Y(2)**2 - 1.0e0
      END IF
      RETURN
      END
      SUBROUTINE DAEJAC(NEQ,T,Y,YDOT,H,D,P)
*     .. Parameters ..
      real              ONE, TWO
      PARAMETER         (ONE=1.0e0,TWO=2.0e0)
*     .. Scalar Arguments ..
      real              D, H, T
      INTEGER           NEQ
*     .. Array Arguments ..
      real              P(NEQ,NEQ), Y(NEQ), YDOT(NEQ)
*     .. Local Scalars ..
      real              HXD
```

```
*     .. Executable Statements ..
      HXD = H*D
      P(1,1) = (ONE+HXD*Y(6))
      P(1,3) = -HXD
      P(1,6) = HXD*Y(1)
      P(2,2) = (ONE+HXD*Y(6))
      P(2,4) = -HXD
      P(2,6) = HXD*Y(2)
      P(3,1) = HXD*Y(5)
      P(3,3) = ONE
      P(3,5) = HXD*Y(1)
      P(4,2) = HXD*Y(5)
      P(4,4) = ONE
      P(4,5) = HXD*Y(2)
      P(5,1) = -HXD*Y(3)
      P(5,2) = -HXD*Y(4)
      P(5,3) = -HXD*Y(1)
      P(5,4) = -HXD*Y(2)
      P(6,1) = -TWO*HXD*Y(1)
      P(6,2) = -TWO*HXD*Y(2)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D02MVF Example Program Results

Pendulum problem with relative tolerance = 1.0E-03
                  and absolute tolerance = 1.0E-06

   t          y1      y2      y3      y4      y5      y6
0.0000     1.0000  0.0000  0.0000  0.0000  0.0000  0.0000
3.1416    -0.9872 -0.1597 -0.0902  0.5579  0.4790  0.0000
```

D02MZF – NAG Fortran Library Routine Document

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

1. Purpose

D02MZF interpolates components of the solution of a system of first order differential equations from information provided by the integrators in the D02M-D02N Subchapter.

2. Specification

```
      SUBROUTINE D02MZF (TSOL, SOL, M, NEQMAX, NEQ, YSAVE,
     1                   NY2DIM, RWORK, IFAIL)
      INTEGER          M, NEQMAX, NEQ, NY2DIM, IFAIL
      real             TSOL, SOL(M), YSAVE(NEQMAX,NY2DIM),
     1                 RWORK(50+4*NEQMAX)
```

3. Description

D02MZF evaluates the first M components of the solution of a system of ordinary differential equations at any point using natural polynomial interpolation based on information generated by the integrator. This information must be passed unchanged to D02MZF. D02MZF should not normally be used to extrapolate outside the range of values obtained from the above routine.

4. References

See the Chapter Introduction.

5. Parameters

1: TSOL – ***real***. *Input*

On entry: the point at which the first M components of the solution are to be evaluated. TSOL should not normally be an extrapolation point. Extrapolation is permitted but not recommended.

2: SOL(M) – ***real*** array. *Output*

On exit: the calculated value of the solution at TSOL.

3: M – INTEGER. *Input*

On entry: the number of components of the solution whose values are required.

Constraint: $1 \le M \le NEQ$.

4: NEQMAX – INTEGER. *Input*

On entry: the value used for the parameter NEQMAX when calling the integrator.

Constraint: $NEQMAX \ge 1$.

5: NEQ – INTEGER. *Input*

On entry: the value used for the parameter NEQ when calling the integrator.

Constraint: $1 \le NEQ \le NEQMAX$.

6: YSAVE(NEQMAX,NY2DIM) – ***real*** array. *Input*

On entry: the values provided in the array YSAVE on return from the integrator.

7: NY2DIM – INTEGER. *Input*

On entry: the value used for the parameter NY2DIM when calling the integrator.

8: RWORK(50+4*NEQMAX) – ***real*** array. *Input*

On entry: the values provided in the array RWORK on return from the integrator.

9: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL = 1

On entry, M < 1,
or NEQMAX < 1,
or NEQ < 1,
or M > NEQ,
or NEQ > NEQMAX.

IFAIL = 2

On entry, when accessing an element of the array RWORK an unexpected quantity was found. The user has not passed the correct array to D02MZF or has overwritten elements of this array.

IFAIL = 3

On entry, D02MZF has been called for extrapolation. Before returning with this error exit, the value of the solution at TSOL is calculated and placed in SOL.

7. Accuracy

The solution values returned will be of a similar accuracy to those computed by the integrator.

8. Further Comments

Users are recommended to employ the interpolant provided by D02XKF if using the backward differentiation integrator specified by calling setup D02NVF with the parameter PETZLD set to .FALSE. .

9. Example

See D02NGF example program.

D02NBF – NAG Fortran Library Routine Document

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

1. Purpose

D02NBF is a forward communication routine for integrating stiff systems of explicit ordinary differential equations when the Jacobian is a full matrix.

2. Specification

```
 SUBROUTINE D02NBF (NEQ, NEQMAX, T, TOUT, Y, YDOT, RWORK, RTOL,
1                    ATOL, ITOL, INFORM, FCN, YSAVE, NY2DIM, JAC,
2                    WKJAC, NWKJAC, MONITR, ITASK, ITRACE, IFAIL)
 INTEGER        NEQ, NEQMAX, ITOL, INFORM(23), NY2DIM, NWKJAC,
1               ITASK, ITRACE, IFAIL
 real           T, TOUT, Y(NEQMAX), YDOT(NEQMAX),
1               RWORK(50+4*NEQMAX), RTOL(*), ATOL(*),
2               YSAVE(NEQMAX,NY2DIM), WKJAC(NWKJAC)
 EXTERNAL       FCN, JAC, MONITR
```

3. Description

D02NBF is a general purpose routine for integrating the initial value problem for a stiff system of explicit ordinary differential equations,

$$y' = g(t,y)$$

It is designed specifically for the case where the Jacobian $\frac{\partial g}{\partial y}$ is a full matrix.

Both interval and step oriented modes of operation are available and also modes designed to permit intermediate output within an interval oriented mode.

An outline of a typical program calling D02NBF is given below. It calls the full matrix linear algebra setup routine D02NSF, and the Backward Differentiation Formula (BDF) integrator setup routine D02NVF, and its diagnostic counterpart D02NYF.

```
C
C     declarations
C
      EXTERNAL FCN, JAC, MONITR
      .
      .
      .
      IFAIL = 0
      CALL D02NVF(...,IFAIL)
      CALL D02NSF(NEQ, NEQMAX, JCEVAL, NWKJAC, RWORK, IFAIL)
      IFAIL = - 1
      CALL D02NBF(NEQ, NEQMAX, T, TOUT, Y, YDOT, RWORK, RTOL,
     +   ATOL, ITOL, INFORM, FCN, YSAVE, NY2DIM, JAC, WKJAC, NWKJAC,
     +   MONITR, ITASK, ITRACE, IFAIL)
      IF (IFAIL.EQ.1. OR. IFAIL.GE.14) STOP
      IFAIL = 0
      CALL D02NYF(...)
      .
      .
      .
      STOP
      END
```

The linear algebra setup routine D02NSF and one of the integrator setup routines, D02NVF or D02NWF, must be called prior to the call of D02NBF. The integrator diagnostic routine D02NYF may be called after the call to D02NBF. There is also a routine, D02NZF, designed to permit the user to change step size on a continuation call to D02NBF without restarting the integration process.

4. References

See Subchapter Introduction.

5. Parameters

1: NEQ – INTEGER. *Input*

On entry: the number of differential equations to be solved.

Constraint: NEQ ≥ 1.

2: NEQMAX – INTEGER. *Input*

On entry: a bound on the maximum number of differential equations to be solved during the integration.

Constraint: NEQMAX ≥ NEQ.

3: T – ***real***. *Input/Output*

On entry: the value of the independent variable, t. The input value of T is used only on the first call as the initial point of the integration.

On exit: the value at which the computed solution y is returned (usually at TOUT).

4: TOUT – ***real***. *Input*

On entry: the next value of t at which a computed solution is desired. For the initial t, the input value of TOUT is used to determine the direction of integration. Integration is permitted in either direction (see also ITASK).

Constraint: TOUT ≠ T.

5: Y(NEQMAX) – ***real*** array. *Input/Output*

On entry: the values of the dependent variables (solution). On the first call the first NEQ elements of Y must contain the vector of initial values.

On exit: the computed solution vector, evaluated at T (usually T = TOUT).

6: YDOT(NEQMAX) – ***real*** array. *Output*

On exit: the time derivatives y' of the vector y at the last integration point.

7: RWORK(50+4*NEQMAX) – ***real*** array. *Workspace*

8: RTOL(*) – ***real*** array. *Input*

Note: the dimension of RTOL must be at least 1 or NEQ (see ITOL).

On entry: the relative local error tolerance.

Constraint: RTOL(i) ≥ 0.0 for all relevant i (see ITOL).

9: ATOL(*) – ***real*** array. *Input*

Note: the dimension of ATOL must be at least 1 or NEQ (see ITOL).

On entry: the absolute local error tolerance.

Constraint: ATOL(i) ≥ 0.0 for all relevant i (see ITOL).

10: ITOL – INTEGER. *Input*

On entry: a value to indicate the form of the local error test. ITOL indicates to D02NBF whether to interpret either or both of RTOL or ATOL as a vector or a scalar. The error test to be satisfied is $\|e_i/w_i\| < 1.0$, where w_i is defined as follows:

ITOL	RTOL	ATOL	w_i
1	scalar	scalar	$\mathrm{RTOL}(1)\times\lvert y_i\rvert + \mathrm{ATOL}(1)$
2	scalar	vector	$\mathrm{RTOL}(1)\times\lvert y_i\rvert + \mathrm{ATOL}(i)$
3	vector	scalar	$\mathrm{RTOL}(i)\times\lvert y_i\rvert + \mathrm{ATOL}(1)$
4	vector	vector	$\mathrm{RTOL}(i)\times\lvert y_i\rvert + \mathrm{ATOL}(i)$

e_i is an estimate of the local error in y_i, computed internally, and the choice of norm to be used is defined by a previous call to an integrator setup routine.

Constraint: $1 \le \mathrm{ITOL} \le 4$.

11: INFORM(23) – INTEGER array. *Workspace*

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

FCN must evaluate the derivative vector for the explicit ordinary differential equation system, defined by $y' = g(t,y)$.

Its specification is:

```
SUBROUTINE FCN(NEQ, T, Y, F, IRES)
INTEGER    NEQ, IRES
real       T, Y(NEQ), F(NEQ)
```

1: NEQ – INTEGER. *Input*

On entry: the number of differential equations being solved.

2: T – ***real***. *Input*

On entry: the current value of the independent variable t.

3: Y(NEQ) – ***real*** array. *Input*

On entry: the value of y_i, for $i = 1,2,\ldots,\mathrm{NEQ}$.

4: F(NEQ) – ***real*** array. *Output*

On exit: the value y'_i, given by $y'_i = g_i(t,y)$, for $i = 1,2,\ldots,\mathrm{NEQ}$.

5: IRES – INTEGER. *Input/Output*

On entry: IRES = 1.

On exit: the user may set IRES as follows to indicate certain conditions in FCN to the integrator:

IRES = 1

indicates a normal return from FCN, that is IRES is not altered by the user and integration continues.

IRES = 2

indicates to the integrator that control should be passed back immediately to the calling (sub)program with the error indicator set to IFAIL = 11.

IRES = 3

indicates to the integrator that an error condition has occurred in the solution vector, its time derivative or in the value of t. The integrator will use a smaller time step to try to avoid this condition. If this is not possible the integrator returns to the calling (sub)program with the error indicator set to IFAIL = 7.

IRES = 4

indicates to the integrator to stop its current operation and to enter the MONITR routine immediately with parameter IMON = −2.

FCN must be declared as EXTERNAL in the (sub)program from which D02NBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

13: YSAVE(NEQMAX,NY2DIM) – ***real*** array. *Workspace*

14: NY2DIM – INTEGER. *Input*

On entry: the second dimension of the array YSAVE as declared in the (sub)program from which D02NBF is called. An appropriate value for NY2DIM is described in the specification of the integrator setup routines D02NVF and D02NWF. This value must be the same as that supplied to the integrator setup routine.

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

JAC must evaluate the Jacobian of the system. If this option is not required, the actual argument for JAC must be the dummy routine D02NBZ. (D02NBZ is included in the NAG Fortran Library and so need not be supplied by the user. Its name may be implementation dependent: see the Users' Note for your implementation for details.) The user indicates to the integrator whether this option is to be used by setting the parameter JCEVAL appropriately in a call to the linear algebra setup routine D02NSF.

First we must define the system of nonlinear equations which is solved internally by the integrator. The time derivative, y', generated internally has the form

$$y' = (y-z)/(hd),$$

where h is the current stepsize and d is a parameter that depends on the integration method in use. The vector y is the current solution and the vector z depends on information from previous time steps. This means that $\frac{d}{dy'}(\ ) = \frac{1}{(hd)}\frac{d}{dy}(\ )$. The system of nonlinear equations that is solved has the form

$$y' - g(t,y) = 0$$

but it is solved in the form

$$r(t,y) = 0,$$

where the function r is defined by

$$r(t,y) = hd((y-z)/(hd)-g(t,y)).$$

It is the Jacobian matrix $\frac{\partial r}{\partial y}$ that the user must supply in the routine JAC as follows:

$$\frac{\partial r_i}{\partial y_j} = 1 - (hd)\frac{\partial g_i}{\partial y_j}, \qquad \text{if } i = j,$$

$$\frac{\partial r_i}{\partial y_j} = -(hd)\frac{\partial g_i}{\partial y_j}, \qquad \text{otherwise.}$$

Its specification is:

```
SUBROUTINE JAC(NEQ, T, Y, H, D, P)
INTEGER    NEQ
real       T, Y(NEQ), H, D, P(NEQ,NEQ)
```

1: NEQ – INTEGER. *Input*

On entry: the number of differential equations being solved.

2: T – ***real.*** *Input*

On entry: the current value of the independent variable t.

3: Y(NEQ) – ***real*** array. *Input*

On entry: the current solution component y_i, for $i = 1,2,...,\text{NEQ}$.

4: H – ***real.*** *Input*

On entry: the current stepsize.

5: D – ***real.*** *Input*

On entry: the parameter d which depends on the integration method.

6: P(NEQ,NEQ) – ***real*** array. *Output*

On exit: P(i,j) must contain $\frac{\partial r_i}{\partial y_j}$, for $i,j = 1,2,...,\text{NEQ}$.

Only the non-zero elements of this array need be set, since it is preset to zero before the call to JAC.

JAC must be declared as EXTERNAL in the (sub)program from which D02NBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

16: WKJAC(NWKJAC) – ***real*** array. *Workspace*

17: NWKJAC – INTEGER. *Input*

On entry: the dimension of the array WKJAC as declared in the (sub)program from which D02NBF is called. This value must be the same as that supplied to the linear algebra setup routine D02NSF.

Constraint: NWKJAC ≥ NEQMAX×(NEQMAX+1).

18: MONITR – SUBROUTINE, supplied by the user. *External Procedure*

MONITR performs tasks requested by the user. If this option is not required, the actual argument for MONITR must be the dummy routine D02NBY. (D02NBY is included in the NAG Fortran Library and so need not be supplied by the user. Its name may be implementation dependent: see the Users' Note for your implementation for details.)

Its specification is:

```
SUBROUTINE MONITR(NEQ, NEQMAX, T, HLAST, HNEXT, Y, YDOT, YSAVE, R,
                  ACOR, IMON, INLN, HMIN, HMAX, NQU)
INTEGER           NEQ, NEQMAX, IMON, INLN, NQU
real              T, HLAST, HNEXT, Y(NEQMAX), YDOT(NEQMAX), YSAVE(NEQMAX,*),
                  R(NEQMAX), ACOR(NEQMAX,2), HMIN, HMAX
```

1: NEQ – INTEGER. *Input*

On entry: the number of differential equations being solved.

2: NEQMAX – INTEGER. *Input*

On entry: an upper bound on the number of differential equations to be solved.

3: T – ***real.*** *Input*

On entry: the current value of the independent variable.

4: HLAST – ***real.*** *Input*

On entry: the last stepsize successfully used by the integrator.

5: HNEXT – ***real.*** *Input/Output*

On entry: the stepsize that the integrator proposes to take on the next step.

On exit: the next stepsize to be used. If this is different from the input value, then IMON must be set to 4.

6: Y(NEQMAX) – ***real*** array. *Input/Output*

On entry: the values of the dependent variables, y, evaluated at t.

On exit: these values must not be changed unless IMON is set to 2.

7: YDOT(NEQMAX) – ***real*** array. *Input*

On entry: the time derivatives y' of the vector y.

8: YSAVE(NEQMAX,*) – ***real*** array. *Input*

On entry: workspace to enable the user to carry out interpolation using either of the routines D02XJF or D02XKF.

9: R(NEQMAX) – ***real*** array. *Input*

On entry: if IMON = 0 and INLN = 3, the first NEQ elements contain the residual vector, $y' - g(t,y)$.

10: ACOR(NEQMAX,2) – ***real*** array. *Input*

On entry: with IMON = 1, ACOR(i,1) contains the weight used for the ith equation when the norm is evaluated, and ACOR(i,2) contains the estimated local error for the ith equation. The scaled local error at the end of a timestep may be obtained by calling the ***real*** function D02ZAF as follows

```
      IFAIL = 1
      ERRLOC = D02ZAF(NEQ, ACOR(1,2), ACOR(1,1), IFAIL)
C     CHECK IFAIL BEFORE PROCEEDING
```

11: IMON – INTEGER. *Input/Output*

On entry: a flag indicating under what circumstances MONITR was called:

IMON = –2

entry from the integrator after IRES = 4 (set in FCN) caused an early termination (this facility could be used to locate discontinuities).

IMON = –1

the current step failed repeatedly.

IMON = 0

entry after a call to the internal nonlinear equation solver (see below).

IMON = 1

the current step was successful.

On exit: IMON may be re-set to determine subsequent action in D02NBF:

IMON = –2

integration is to be halted. A return will be made from the integrator to the calling (sub)program with IFAIL = 12.

IMON = –1

allow the integrator to continue with its own internal strategy. The integrator will try up to 3 restarts unless IMON is set ≠ –1 on exit.

IMON = 0

return to the internal nonlinear equation solver, where the action taken is determined by the value of INLN (see below).

IMON = 1

normal exit to the integrator to continue integration.

IMON = 2

restart the integration at the current time point. The integrator will restart from order 1 when this option is used. The MONITR provided solution Y will be used for the initial conditions.

IMON = 3

try to continue with the same stepsize and order as was to be used before the call to MONITR. HMIN and HMAX may be altered if desired.

IMON = 4

continue the integration but using a new value HNEXT and possibly new values of HMIN and HMAX.

12: INLN – INTEGER. *Output*

On exit: the action to be taken by the internal nonlinear equation solver when MONITR is exited with IMON = 0. By setting INLN = 3 and returning to the integrator, the residual vector is evaluated and placed in the array R, and then MONITR is called again. At present this is the only option available: INLN must not be set to any other value.

13: HMIN – ***real.*** *Input/Output*

On entry: the minimum stepsize to be taken on the next step.

On exit: the minimum stepsize to be used. If this is different from the input value, then IMON must be set to 3 or 4.

14: HMAX – ***real.*** *Input/Output*

On entry: the maximum stepsize to be taken on the next step.

On exit: the maximum stepsize to be used. If this is different from the input value, then IMON must be set to 3 or 4. If HMAX is set to zero, no limit is assumed.

15: NQU – INTEGER. *Input*

On entry: the order of the integrator used on the last step. This is supplied to enable the user to carry out interpolation using either of the routines D02XJF or D02XKF.

MONITR must be declared as EXTERNAL in the (sub)program from which D02NBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

19: ITASK – INTEGER. *Input*

On entry: the task to be performed by the integrator. The permitted values for ITASK and their meanings are detailed below:

ITASK = 1

normal computation of output values of $y(t)$ at t = TOUT (by overshooting and interpolating).

ITASK = 2

take one step only and return.

ITASK = 3

stop at the first internal integration point at or beyond t = TOUT and return.

ITASK = 4

normal computation of output values of $y(t)$ at t = TOUT but without overshooting t = TCRIT. TCRIT must be specified as an option in one of the integrator setup routines prior to the first call to the integrator, or specified in the optional input routine prior to a continuation call. TCRIT may be equal to or beyond TOUT, but not before it, in the direction of integration.

ITASK = 5

take one step only and return, without passing TCRIT. TCRIT must be specified as under ITASK = 4.

Constraint: 1 ≤ ITASK ≤ 5.

20: ITRACE – INTEGER. *Input*

On entry: the level of output that is printed by the integrator. ITRACE may take the value -1, 0, 1, 2 or 3. If ITRACE < -1, then -1 is assumed and similarly if ITRACE > 3, then 3 is assumed. If ITRACE $= -1$, no output is generated. If ITRACE $= 0$, only warning messages are printed on the current error message unit (see X04AAF). If ITRACE > 0, then warning messages are printed as above, and on the current advisory message unit (see X04ABF) output is generated which details Jacobian entries, the nonlinear iteration and the time integration. The advisory messages are given in greater detail the larger the value of ITRACE.

21: IFAIL – INTEGER. *Input/Output*

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

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

For this routine, because the values of output parameters may be useful even if IFAIL $\neq 0$ on exit, users are recommended to set IFAIL to -1 before entry. **It is then essential to test the value of IFAIL on exit.** To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL = 1

An illegal input was detected on entry, or after an internal call to MONITR. If ITRACE > -1, then the form of the error will be detailed on the current error message unit (see X04AAF).

IFAIL = 2

The maximum number of steps specified has been taken (see the description of optional inputs in the integrator setup routines and the optional input continuation routine, D02NZF).

IFAIL = 3

With the given values of RTOL and ATOL no further progress can be made across the integration range from the current point T. The components Y(1),Y(2),...,Y(NEQ) contain the computed values of the solution at the current point T.

IFAIL = 4

There were repeated error test failures on an attempted step, before completing the requested task, but the integration was successful as far as T. The problem may have a singularity, or the local error requirements may be inappropriate.

IFAIL = 5

There were repeated convergence test failures on an attempted step, before completing the requested task, but the integration was successful as far as T. This may be caused by an inaccurate Jacobian matrix or one which is incorrectly computed.

IFAIL = 6

Some error weight w_i became zero during the integration (see description of ITOL). Pure relative error control (ATOL(i) = 0.0) was requested on a variable (the ith) which has now vanished. The integration was successful as far as T.

IFAIL = 7

The user-supplied subroutine FCN set its error flag (IRES = 3) continually despite repeated attempts by the integrator to avoid this.

IFAIL = 8

Not used for this integrator.

IFAIL = 9

A singular Jacobian $\frac{\partial r}{\partial y}$ has been encountered. This error exit is unlikely to be taken when solving explicit ordinary differential equations. The user should check his problem formulation and Jacobian calculation.

IFAIL = 10

An error occurred during Jacobian formulation or backsubstitution (a more detailed error description may be directed to the current error message unit, see X04AAF).

IFAIL = 11

The user-supplied subroutine FCN signalled the integrator to halt the integration and return (IRES = 2). Integration was successful as far as T.

IFAIL = 12

The user-supplied subroutine MONITR set IMON = –2 and so forced a return but the integration was successful as far as T.

IFAIL = 13

The requested task has been completed, but it is estimated that a small change in RTOL and ATOL is unlikely to produce any change in the computed solution. (Only applies when the user is not operating in one step mode, that is when ITASK ≠ 2 or 5).

IFAIL = 14

The values of RTOL and ATOL are so small that the routine is unable to start the integration.

IFAIL = 15

The linear algebra setup routine D02NSF was not called prior to calling D02NBF.

7. Accuracy

The accuracy of the numerical solution may be controlled by a careful choice of the parameters RTOL and ATOL, and to a much lesser extent by the choice of norm. Users are advised to use scalar error control unless the components of the solution are expected to be poorly scaled. For the type of decaying solution typical of many stiff problems, relative error control with a small absolute error threshold will be most appropriate (that is the user is advised to choose ITOL = 1 with ATOL(1) small but positive).

8. Further Comments

The cost of computing a solution depends critically on the size of the differential system and to a lesser extent on the degree of stiffness of the problem. For D02NBF the cost is proportional to NEQ^3, though for problems which are only mildly nonlinear the cost may be dominated by factors proportional to NEQ^2 except for very large problems.

In general the user is advised to choose the backward differentiation formula option (setup routine D02NVF) but if efficiency is of great importance and especially if it is suspected that $\frac{\partial g}{\partial y}$ has complex eigenvalues near the imaginary axis for some part of the integration, the user should try the BLEND option (setup routine D02NWF).

9. Example

We solve the well-known stiff Robertson problem

$$\begin{aligned} a' &= -0.04a + 1.0\mathrm{E}4bc \\ b' &= 0.04a - 1.0\mathrm{E}4bc - 3.0\mathrm{E}7b^2 \\ c' &= \phantom{-0.04a - 1.0\mathrm{E}4bc - {}} 3.0\mathrm{E}7b^2 \end{aligned}$$

over the range [0,10] with initial conditions $a = 1.0$, and $b = c = 0.0$ using scalar error contol (ITOL = 1) and computation of the solution at TOUT = 10.0 with TCRIT set to 10.0 (ITASK = 4). D02NBY is used for MONITR, we use a BDF integrator (setup routine D02NVF) and we select a modified Newton method. We illustrate the use of both a numerical and an analytical Jacobian.

9.1. Program Text

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

```
*     D02NBF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, NEQMAX, NRW, NINF, NWKJAC, MAXORD, NY2DIM,
     +                 MAXSTP, MXHNIL
      PARAMETER        (NEQ=3,NEQMAX=NEQ,NRW=50+4*NEQMAX,NINF=23,
     +                 NWKJAC=NEQMAX*(NEQMAX+1),MAXORD=5,
     +                 NY2DIM=MAXORD+1,MAXSTP=200,MXHNIL=5)
      real             H0, HMAX, HMIN
      PARAMETER        (H0=0.0e0,HMAX=10.0e0,HMIN=1.0e-10)
      LOGICAL          PETZLD
      PARAMETER        (PETZLD=.FALSE.)
*     .. Local Scalars ..
      real             H, HU, T, TCRIT, TCUR, TOLSF, TOUT
      INTEGER          I, IFAIL, IMXER, ITASK, ITOL, ITRACE, NITER, NJE,
     +                 NQ, NQU, NRE, NST
*     .. Local Arrays ..
      real             ATOL(NEQMAX), CONST(6), RTOL(NEQMAX), RWORK(NRW),
     +                 WKJAC(NWKJAC), Y(NEQMAX), YDOT(NEQMAX),
     +                 YSAVE(NEQMAX,NY2DIM)
      INTEGER          INFORM(NINF)
      LOGICAL          ALGEQU(NEQMAX)
*     .. External Subroutines ..
      EXTERNAL         D02NBF, D02NBY, D02NBZ, D02NSF, D02NVF, D02NYF,
     +                 FCN, JAC, X04ABF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02NBF Example Program Results'
      CALL X04ABF(1,NOUT)
*
*     First case. Integrate to TOUT without passing TOUT (set TCRIT to
*     TOUT and ITASK=4) using B.D.F formulae with a Newton method.
*     Default values for the array CONST are used. Employ scalar
*     tolerances and the Jacobian is evaluated internally.
*     MONITR subroutine replaced by NAG dummy routine D02NBY.
*
      T = 0.0e0
      TOUT = 10.0e0
      ITASK = 4
      Y(1) = 1.0e0
      Y(2) = 0.0e0
      Y(3) = 0.0e0
      ITOL = 1
      RTOL(1) = 1.0e-4
      ATOL(1) = 1.0e-7
      DO 20 I = 1, 6
         CONST(I) = 0.0e0
   20 CONTINUE
```

```
      TCRIT = TOUT
      IFAIL = 0
*
      CALL D02NVF(NEQMAX,NY2DIM,MAXORD,'Newton',PETZLD,CONST,TCRIT,HMIN,
     +            HMAX,H0,MAXSTP,MXHNIL,'Average-L2',RWORK,IFAIL)
      CALL D02NSF(NEQ,NEQMAX,'Numerical',NWKJAC,RWORK,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) '  Numerical Jacobian'
      WRITE (NOUT,*)
      WRITE (NOUT,*) '    X         Y(1)            Y(2)           Y(3)'
      WRITE (NOUT,99999) T, (Y(I),I=1,NEQ)
*
*     Soft fail and error messages only
      ITRACE = 0
      IFAIL = 1
*
      CALL D02NBF(NEQ,NEQMAX,T,TOUT,Y,YDOT,RWORK,RTOL,ATOL,ITOL,INFORM,
     +            FCN,YSAVE,NY2DIM,D02NBZ,WKJAC,NWKJAC,D02NBY,ITASK,
     +            ITRACE,IFAIL)
*
      IF (IFAIL.EQ.0) THEN
         WRITE (NOUT,99999) T, (Y(I),I=1,NEQ)
*
         CALL D02NYF(NEQ,NEQMAX,HU,H,TCUR,TOLSF,RWORK,NST,NRE,NJE,NQU,
     +               NQ,NITER,IMXER,ALGEQU,INFORM,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99997) ' HUSED = ', HU, '  HNEXT = ', H,
     +     '  TCUR = ', TCUR
         WRITE (NOUT,99996) ' NST = ', NST, '    NRE = ', NRE,
     +     '    NJE = ', NJE
         WRITE (NOUT,99996) ' NQU = ', NQU, '    NQ  = ', NQ,
     +     '  NITER = ', NITER
         WRITE (NOUT,99995) ' Max Err Comp = ', IMXER
         WRITE (NOUT,*)
      ELSE
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'Exit D02NBF with IFAIL = ', IFAIL,
     +     '  and T = ', T
      END IF
*
*     Second case. Integrate to TOUT without passing TOUT (set TCRIT to
*     TOUT and ITASK=4) using B.D.F formulae with a Newton method.
*     Default values for the array CONST are used. Employ scalar
*     tolerances and the Jacobian is evaluated by JAC.
*     MONITR subroutine replaced by NAG dummy routine D02NBY.
*
      T = 0.0e0
      Y(1) = 1.0e0
      Y(2) = 0.0e0
      Y(3) = 0.0e0
      IFAIL = 0
*
      CALL D02NVF(NEQMAX,NY2DIM,MAXORD,'Newton',PETZLD,CONST,TCRIT,HMIN,
     +            HMAX,H0,MAXSTP,MXHNIL,'Average-L2',RWORK,IFAIL)
*
      CALL D02NSF(NEQ,NEQMAX,'Analytical',NWKJAC,RWORK,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) '  Analytic Jacobian'
      WRITE (NOUT,*)
      WRITE (NOUT,*) '    X         Y(1)            Y(2)           Y(3)'
      WRITE (NOUT,99999) T, (Y(I),I=1,NEQ)
      IFAIL = 1
*
```

```
      CALL D02NBF(NEQ,NEQMAX,T,TOUT,Y,YDOT,RWORK,RTOL,ATOL,ITOL,INFORM,
     +            FCN,YSAVE,NY2DIM,JAC,WKJAC,NWKJAC,D02NBY,ITASK,ITRACE,
     +            IFAIL)
*
      IF (IFAIL.EQ.0) THEN
         WRITE (NOUT,99999) T, (Y(I),I=1,NEQ)
*
         CALL D02NYF(NEQ,NEQMAX,HU,H,TCUR,TOLSF,RWORK,NST,NRE,NJE,NQU,
     +               NQ,NITER,IMXER,ALGEQU,INFORM,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99997) ' HUSED = ', HU, '  HNEXT = ', H,
     +     '  TCUR = ', TCUR
         WRITE (NOUT,99996) ' NST = ', NST, '    NRE = ', NRE,
     +     '    NJE = ', NJE
         WRITE (NOUT,99996) ' NQU = ', NQU, '    NQ  = ', NQ,
     +     '  NITER = ', NITER
         WRITE (NOUT,99995) ' Max Err Comp = ', IMXER
         WRITE (NOUT,*)
      ELSE
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'Exit D02NBF with IFAIL = ', IFAIL,
     +     '  and T = ', T
      END IF
      STOP
*
99999 FORMAT (1X,F8.3,3(F13.5,2X))
99998 FORMAT (1X,A,I2,A,e12.5)
99997 FORMAT (1X,A,e12.5,A,e12.5,A,e12.5)
99996 FORMAT (1X,A,I6,A,I6,A,I6)
99995 FORMAT (1X,A,I4)
      END
*
      SUBROUTINE FCN(NEQ,T,Y,R,IRES)
*     .. Scalar Arguments ..
      real             T
      INTEGER          IRES, NEQ
*     .. Array Arguments ..
      real             R(NEQ), Y(NEQ)
*     .. Executable Statements ..
      R(1) = -0.04e0*Y(1) + 1.0e4*Y(2)*Y(3)
      R(2) = 0.04e0*Y(1) - 1.0e4*Y(2)*Y(3) - 3.0e7*Y(2)*Y(2)
      R(3) = 3.0e7*Y(2)*Y(2)
      RETURN
      END
*
      SUBROUTINE JAC(NEQ,T,Y,H,D,P)
*     .. Scalar Arguments ..
      real             D, H, T
      INTEGER          NEQ
*     .. Array Arguments ..
      real             P(NEQ,NEQ), Y(NEQ)
*     .. Local Scalars ..
      real             HXD
*     .. Executable Statements ..
      HXD = H*D
      P(1,1) = 1.0e0 - HXD*(-0.04e0)
      P(1,2) = -HXD*(1.0e4*Y(3))
      P(1,3) = -HXD*(1.0e4*Y(2))
      P(2,1) = -HXD*(0.04e0)
      P(2,2) = 1.0e0 - HXD*(-1.0e4*Y(3)-6.0e7*Y(2))
      P(2,3) = -HXD*(-1.0e4*Y(2))
*     Do not need to set P(3,1) since Jacobian preset to zero
*     P(3,1) =        - HXD*(0.0E0)
      P(3,2) = -HXD*(6.0e7*Y(2))
      P(3,3) = 1.0e0 - HXD*(0.0e0)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D02NBF Example Program Results

  Numerical Jacobian

    X          Y(1)           Y(2)           Y(3)
   0.000     1.00000        0.00000        0.00000
  10.000     0.84136        0.00002        0.15863

 HUSED =  0.51867E+00  HNEXT =  0.51867E+00  TCUR =  0.10000E+02
 NST =     55    NRE =    132    NJE =      17
 NQU =      3    NQ  =      3  NITER =      79
 Max Err Comp =    3


  Analytic Jacobian

    X          Y(1)           Y(2)           Y(3)
   0.000     1.00000        0.00000        0.00000
  10.000     0.84136        0.00002        0.15863

 HUSED =  0.51867E+00  HNEXT =  0.51867E+00  TCUR =  0.10000E+02
 NST =     55    NRE =     81    NJE =      17
 NQU =      3    NQ  =      3  NITER =      79
 Max Err Comp =    3
```

D02NCF – NAG Fortran Library Routine Document

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

1. Purpose

D02NCF is a forward communication routine for integrating stiff systems of explicit ordinary differential equations when the Jacobian is a banded matrix.

2. Specification

```
      SUBROUTINE D02NCF (NEQ, NEQMAX, T, TOUT, Y, YDOT, RWORK, RTOL,
     1                   ATOL, ITOL, INFORM, FCN, YSAVE, NY2DIM, JAC,
     2                   WKJAC, NWKJAC, JACPVT, NJCPVT, MONITR, ITASK,
     3                   ITRACE, IFAIL)
      INTEGER            NEQ, NEQMAX, ITOL, INFORM(23), NY2DIM, NWKJAC,
     1                   JACPVT(NJCPVT), NJCPVT, ITASK, ITRACE, IFAIL
      real               T, TOUT, Y(NEQMAX), YDOT(NEQMAX),
     1                   RWORK(50+4*NEQMAX), RTOL(*), ATOL(*),
     2                   YSAVE(NEQMAX,NY2DIM), WKJAC(NWKJAC)
      EXTERNAL           FCN, JAC, MONITR
```

3. Description

D02NCF is a general purpose routine for integrating the initial value problem for a stiff system of explicit ordinary differential equations,

$$y' = g(t,y).$$

It is designed specifically for the case where the Jacobian $\frac{\partial g}{\partial y}$ is a banded matrix.

Both interval and step oriented modes of operation are available and also modes designed to permit intermediate output within an interval oriented mode.

An outline of a typical calling program for D02NCF is given below. It calls the banded matrix linear algebra setup routine D02NTF, the Backward Differentiation Formula (BDF) integrator setup routine D02NVF, and its diagnostic counterpart D02NYF.

```
C
C     declarations
C
      EXTERNAL FCN, JAC, MONITR
           .
           .
           .
      IFAIL = 0
      CALL D02NVF(...,IFAIL)
      CALL D02NTF(NEQ, NEQMAX, JCEVAL, ML, MU, NWKJAC, NJCPVT,
     +   RWORK, IFAIL)
      IFAIL = -1
      CALL D02NCF(NEQ, NEQMAX, T, TOUT, Y, YDOT, RWORK, RTOL,
     +   ATOL, ITOL, INFORM, FCN, YSAVE, NY2DIM, JAC, WKJAC, NWKJAC,
     +   JACPVT, NJCPVT, MONITR, ITASK, ITRACE, IFAIL)
      IF (IFAIL.EQ.1 .OR. IFAIL.GE.14) STOP
      IFAIL = 0
      CALL D02NYF(...)
           .
           .
           .
      STOP
      END
```

The linear algebra setup routine D02NTF and one of the integrator setup routines, D02NVF or D02NWF, must be called prior to the call of D02NCF. The integrator diagnostic routine D02NYF, may be called after the call to D02NCF. There is also a routine, D02NZF, designed to permit the user to change step size on a continuation call to D02NCF without restarting the integration process.

4. References

See Subchapter Introduction.

5. Parameters

1: NEQ – INTEGER. *Input*

On entry: the number of differential equations to be solved.

Constraint: NEQ $\geq$ 1.

2: NEQMAX – INTEGER. *Input*

On entry: a bound on the maximum number of differential equations to be solved during the integration.

Constraint: NEQMAX $\geq$ NEQ.

3: T – ***real.*** *Input/Output*

On entry: the value of the independent variable, t. The input value of T is used only on the first call as the initial point of the integration.

On exit: the value at which the computed solution y is returned (usually at TOUT).

4: TOUT – ***real.*** *Input*

On entry: the next value of t at which a computed solution is desired. For the initial t, the input value of TOUT is used to determine the direction of integration. Integration is permitted in either direction (see also ITASK).

Constraint: TOUT $\neq$ T.

5: Y(NEQMAX) – ***real*** array. *Input/Output*

On entry: the values of the dependent variables (solution). On the first call the first NEQ elements of y must contain the vector of initial values.

On exit: the computed solution vector, evaluated at T (usually T = TOUT).

6: YDOT(NEQMAX) – ***real*** array. *Output*

On exit: the time derivatives y' of the vector y at the last integration point.

7: RWORK(50+4*NEQMAX) – ***real*** array. *Workspace*

8: RTOL(*) – ***real*** array. *Input*

Note: the dimension of RTOL must be at least 1 or NEQ (see ITOL).

On entry: the relative local error tolerance.

Constraint: RTOL(i) $\geq$ 0.0 for all relevant i (see ITOL).

9: ATOL(*) – ***real*** array. *Input*

Note: the dimension of ATOL must be at least 1 or NEQ (see ITOL).

On entry: the absolute local error tolerance.

Constraint: ATOL(i) $\geq$ 0.0 for all relevant i (see ITOL).

10: ITOL – INTEGER. *Input*

On entry: a value to indicate the form of the local error test. ITOL indicates to D02NCF whether to interpret either or both of RTOL or ATOL as a vector or a scalar. The error test to be satisfied is $\|e_i/w_i\| < 1.0$, where w_i is defined as follows:

ITOL	RTOL	ATOL	w_i
1	scalar	scalar	$\mathrm{RTOL}(1)\times\lvert y_i\rvert + \mathrm{ATOL}(1)$
2	scalar	vector	$\mathrm{RTOL}(1)\times\lvert y_i\rvert + \mathrm{ATOL}(i)$
3	vector	scalar	$\mathrm{RTOL}(i)\times\lvert y_i\rvert + \mathrm{ATOL}(1)$
4	vector	vector	$\mathrm{RTOL}(i)\times\lvert y_i\rvert + \mathrm{ATOL}(i)$

e_i is an estimate of the local error in y_i, computed internally, and the choice of norm to be used is defined by a previous call to an integrator setup routine.

Constraint: $1 \le \mathrm{ITOL} \le 4$.

11: INFORM(23) – INTEGER array. *Workspace*

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

FCN must evaluate the derivative vector for the explicit ordinary differential equation system, defined by $y' = g(t,y)$.

Its specification is:

```
SUBROUTINE FCN(NEQ, T, Y, F, IRES)
INTEGER    NEQ, IRES
real       T, Y(NEQ), F(NEQ)
```

1: NEQ – INTEGER. *Input*

On entry: the number of differential equations being solved.

2: T – ***real***. *Input*

On entry: the current value of the independent variable t.

3: Y(NEQ) – ***real*** array. *Input*

On entry: the value of y_i for $i = 1,2,\ldots,\mathrm{NEQ}$.

4: F(NEQ) – ***real*** array. *Output*

On exit: the value y_i', given by $y_i' = g_i(t,y)$, for $i = 1,2,\ldots,\mathrm{NEQ}$.

5: IRES – INTEGER. *Input/Output*

On entry: IRES = 1.

On exit: the user may set IRES as follows to indicate certain conditions in FCN to the integrator:

IRES = 1

indicates a normal return from FCN, that is IRES is not altered by the user and integration continues.

IRES = 2

indicates to the integrator that control should be passed back immediately to the calling (sub)program with the error indicator set to IFAIL = 11.

IRES = 3

indicates to the integrator that an error condition has occurred in the solution vector, its time derivative or in the value of t. The integrator will use a smaller time step to try to avoid this condition. If this is not possible the integrator returns to the calling (sub)program with the error indicator set to IFAIL = 7.

IRES = 4

indicates to the integrator to stop its current operation and to enter the MONITR routine immediately with parameter IMON = –2.

FCN must be declared as EXTERNAL in the (sub)program from which D02NCF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

13: YSAVE(NEQMAX, NY2DIM) – ***real*** array. *Workspace*

14: NY2DIM – INTEGER. *Input*

On entry: the second dimension of the array YSAVE as declared in the (sub)program from which D02NCF is called. An appropriate value for NY2DIM is described in the specifications of the integrator setup routines D02NVF and D02NWF. This value must be the same as that supplied to the integrator setup routine.

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

JAC must evaluate the Jacobian of the system. If this option is not required, the actual argument for JAC must be the dummy routine D02NCZ. (D02NCZ is included in the NAG Fortran Library and so need not be supplied by the user. Its name may be implementation dependent: see the Users' Note for your implementation for details.) The user indicates to the integrator whether this option is to be used by setting the parameter JCEVAL appropriately in a call to the linear algebra setup routine D02NTF.

First we must define the system of nonlinear equations which is solved internally by the integrator. The time derivative, y', generated internally has the form

$$y' = (y-z)/(hd),$$

where h is the current stepsize and d is a parameter that depends on the integration method in use. The vector y is the current solution and the vector z depends on information from previous time steps. This means that $\frac{d}{dy'}(\;) = \frac{1}{(hd)}\frac{d}{dy}(\;)$. The system of nonlinear equations that is solved has the form

$$y' - g(t,y) = 0$$

but it is solved in the form

$$r(t,y) = 0,$$

where the function r is defined by

$$r(t,y) = hd((y-z)/(hd)-g(t,y)).$$

It is the Jacobian matrix $\frac{\partial r}{\partial y}$ that the user must supply in the routine JAC as follows:

$$\frac{\partial r_i}{\partial y_j} = 1 - (hd)\frac{\partial g_i}{\partial y_j}, \qquad \text{if } i = j,$$

$$\frac{\partial r_i}{\partial y_j} = -(hd)\frac{\partial g_i}{\partial y_j}, \qquad \text{otherwise.}$$

Its specification is:

```
SUBROUTINE JAC(NEQ, T, Y, H, D, ML, MU, P)
INTEGER    NEQ, ML, MU
real       T, Y(NEQ), H, D, P(ML+MU+1,NEQ)
```

1: NEQ – INTEGER. *Input*

On entry: the number of differential equations being solved.

2: T – ***real***. *Input*

On entry: the current value of the independent variable, t.

3: Y(NEQ) – ***real*** array. *Input*

On entry: the current solution component y_i, for $i = 1,2,...,$NEQ.

4: H – ***real.*** *Input*

On entry: the current stepsize.

5: D – ***real.*** *Input*

On entry: the parameter d which depends on the integration method.

6: ML – INTEGER. *Input*
7: MU – INTEGER. *Input*

On entry: the number of subdiagonals and superdiagonals respectively in the band.

8: P(ML+MU+1,NEQ) – ***real*** array. *Output*

On exit: elements of the Jacobian matrix $\frac{\partial r}{\partial y}$ stored as specified by the following pseudo-code:

```
      DO 20 I = 1, NEQ
         J1 = MAX(I-ML,1)
         J2 = MIN(I+MU,NEQ)
         DO 10 J = J1, J2
            K = MIN(ML+1-I,0)+J
            P(K,I) = ∂R/∂Y(I,J)
   10    CONTINUE
   20 CONTINUE
```

See also the routine document for F01LBF.

Only non-zero elements of this array need be set, since it is preset to zero before the call to JAC.

JAC must be declared as EXTERNAL in the (sub)program from which D02NCF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

16: WKJAC(NWKJAC) – ***real*** array. *Workspace*
17: NWKJAC – INTEGER. *Input*

On entry: the dimension of the array WKJAC as declared in the (sub)program from which D02NCF is called. This value must be the same as that supplied to the linear algebra setup routine D02NTF.

Constraint: NWKJAC ≥ $(2m_L+m_U+1)\times$NEQMAX where m_L and m_U are the number of subdiagonals and superdiagonals respectively in the band, defined by a call to D02NTF.

18: JACPVT(NJCPVT) – INTEGER array. *Workspace*
19: NJCPVT – INTEGER. *Input*

On entry: the dimension of the array JACPVT as declared in the (sub)program from which D02NCF is called. This value must be the same as that supplied to the linear algebra setup routine D02NTF.

Constraint: NJCPVT ≥ NEQMAX.

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

MONITR performs tasks requested by the user. If this option is not required, the actual argument for MONITR must be the dummy routine D02NBY. (D02NBY is included in the NAG Fortran Library and so need not be supplied by the user. Its name may be implementation dependent: see the Users' Note for your implementation for details).

Its specification is:

```
 SUBROUTINE MONITR(NEQ, NEQMAX, T, HLAST, HNEXT, Y, YDOT, YSAVE, R,
1                  ACOR, IMON, INLN, HMIN, HMAX, NQU)
 INTEGER           NEQ, NEQMAX, IMON, INLN, NQU
 real              T, HLAST, HNEXT, Y(NEQMAX), YDOT(NEQMAX), YSAVE(NEQMAX,*),
1                  R(NEQMAX), ACOR(NEQMAX,2), HMIN, HMAX
```

1: NEQ – INTEGER. *Input*

On entry: the number of differential equations being solved.

2: NEQMAX – INTEGER. *Input*

On entry: an upper bound on the number of differential equations to be solved.

3: T – ***real***. *Input*

On entry: the current value of the independent variable.

4: HLAST – ***real***. *Input*

On entry: the last stepsize successfully used by the integrator.

5: HNEXT – ***real***. *Input/Output*

On entry: the stepsize that the integrator proposes to take on the next step.

On exit: the next stepsize to be used. If this is different from the input value, then IMON must be set to 4.

6: Y(NEQMAX) – ***real*** array. *Input/Output*

On entry: the values of the dependent variables, y, evaluated at t.

On exit: these values must not be changed unless IMON is set to 2.

7: YDOT(NEQMAX) – ***real*** array. *Input*

On entry: the time derivatives y' of the vector y.

8: YSAVE(NEQMAX,*) – ***real*** array. *Input*

On entry: workspace to enable the user to carry out interpolation using either of the routines D02XJF or D02XKF.

9: R(NEQMAX) – ***real*** array. *Input*

On entry: if IMON = 0 and INLN = 3, the first NEQ elements contain the residual vector $y' - g(t,y)$.

10: ACOR(NEQMAX,2) – ***real*** array. *Input*

On entry: with IMON = 1, ACOR(i,1) contains the weight used for the ith equation when the norm is evaluated, and ACOR(i,2) contains the estimated local error for the ith equation. The scaled local error at the end of a timestep may be obtained by calling the ***real*** function D02ZAF as follows

```
      IFAIL = 1
      ERRLOC = D02ZAF(NEQ, ACOR(1,2), ACOR(1,1), IFAIL)
C     CHECK IFAIL BEFORE PROCEEDING
```

11: IMON – INTEGER. *Input/Output*

On entry: a flag indicating under what circumstances MONITR was called:

IMON = −2

entry from the integrator after IRES = 4 (set in FCN) caused an early termination (this facility could be used to locate discontinuities).

IMON = −1

the current step failed repeatedly.

IMON = 0

entry after a call to the internal nonlinear equation solver (see below).

IMON = 1

the current step was successful.

On exit: IMON may be re-set to determine subsequent action in D02NCF:

IMON = –2

integration is to be halted. A return will be made from the integrator to the calling (sub)program with IFAIL = 12.

IMON = –1

allow the integrator to continue with its own internal strategy. The integrator will try up to 3 restarts unless IMON is set ≠ –1 on exit.

IMON = 0

return to the internal nonlinear equation solver, where the action taken is determined by the value of INLN (see below).

IMON = 1

normal exit to the integrator to continue integration.

IMON = 2

restart the integration at the current time point. The integrator will restart from order 1 when this option is used. The MONITR provided solution Y will be used for the initial conditions.

IMON = 3

try to continue with the same stepsize and order as was to be used before the call to MONITR. HMIN and HMAX may be altered if desired.

IMON = 4

continue the integration but using a new value HNEXT and possibly new values of HMIN and HMAX.

12: INLN – INTEGER. *Output*

On exit: the action to be taken by the internal nonlinear equation solver when MONITR is exited with IMON = 0. By setting INLN = 3 and returning to the integrator, the residual vector is evaluated and placed in the array R, and then MONITR is called again. At present this is the only option available: INLN must not be set to any other value.

13: HMIN – ***real.*** *Input/Output*

On entry: the minimum stepsize to be taken on the next step.

On exit: the minimum stepsize to be used. If this is different from the input value, then IMON must be set to 3 or 4.

14: HMAX – ***real.*** *Input/Output*

On entry: the maximum stepsize to be taken on the next step.

On exit: the maximum stepsize to be used. If this is different from the input value, then IMON must be set to 3 or 4. If HMAX is set to zero, no limit is assumed.

15: NQU – INTEGER. *Input*

On entry: the order of the integrator used on the last step. This is supplied to enable the user to carry out interpolation using either of the routines D02XJF or D02XKF.

MONITR must be declared as EXTERNAL in the (sub)program from which D02NCF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

21: ITASK – INTEGER. *Input*

On entry: the task to be performed by the integrator. The permitted values for ITASK and their meanings are detailed below:

ITASK = 1

normal computation of output values of $y(t)$ at t = TOUT (by overshooting and interpolating).

ITASK = 2

take one step only and return.

ITASK = 3

stop at the first internal integration point at or beyond t = TOUT and return.

ITASK = 4

normal computation of output values of $y(t)$ at t = TOUT but without overshooting t = TCRIT. TCRIT must be specified as an option in one of the integrator setup routines prior to the first call to the integrator, or specified in the optional input routine prior to a continuation call. TCRIT may be equal to or beyond TOUT, but not before it, in the direction of integration.

ITASK = 5

take one step only and return, without passing TCRIT. TCRIT must be specified as under ITASK = 4.

Constraint: 1 ≤ ITASK ≤ 5.

22: ITRACE – INTEGER. *Input*

On entry: the level of output that is printed by the integrator. ITRACE may take the value –1, 0, 1, 2 or 3. If ITRACE < –1, then –1 is assumed and similarly if ITRACE > 3, then 3 is assumed. If ITRACE = –1, no output is generated. If ITRACE = 0, only warning messages are printed on the current error message unit (see X04AAF). If ITRACE > 0, then warning messages are printed as above, and on the current advisory message unit (see X04ABF) output is generated which details Jacobian entries, the nonlinear iteration and the time integration. The advisory messages are given in greater detail the larger the value of ITRACE.

23: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.** To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL = 1

An illegal input was detected on entry, or after an internal call to MONITR. If ITRACE > –1, then the form of the error will be detailed on the current error message unit (see X04AAF).

IFAIL = 2

The maximum number of steps specified has been taken (see the description of optional inputs in the integrator setup routines and the optional input continuation routine, D02NZF).

IFAIL = 3

With the given values of RTOL and ATOL no further progress can be made across the integration range from the current point T. The components Y(1),Y(2),...,Y(NEQ) contain the computed values of the solution at the current point T.

IFAIL = 4

There were repeated error test failures on an attempted step, before completing the requested task, but the integration was successful as far as T. The problem may have a singularity, or the local error requirements may be inappropriate.

IFAIL = 5

There were repeated convergence test failures on an attempted step, before completing the requested task, but the integration was successful as far as T. This may be caused by an inaccurate Jacobian matrix or one which is incorrectly computed.

IFAIL = 6

Some error weight w_i became zero during the integration (see description of ITOL). Pure relative error control (ATOL(i) = 0.0) was requested on a variable (the ith) which has now vanished. The integration was successful as far as T.

IFAIL = 7

The user-supplied subroutine FCN set its error flag (IRES = 3) continually despite repeated attempts by the integrator to avoid this.

IFAIL = 8

Not used for this integrator.

IFAIL = 9

A singular Jacobian $\frac{\partial r}{\partial y}$ has been encountered. This error exit is unlikely to be taken when solving explicit ordinary differential equations. The user should check his problem formulation and Jacobian calculation.

IFAIL = 10

An error occurred during Jacobian formulation or backsubstitution (a more detailed error description may be directed to the current error message unit, see X04AAF).

IFAIL = 11

The user-supplied subroutine FCN signalled the integrator to halt the integration and return (IRES = 2). Integration was successful as far as T.

IFAIL = 12

The user-supplied subroutine MONITR set IMON = –2 and so forced a return but the integration was successful as far as T.

IFAIL = 13

The requested task has been completed, but it is estimated that a small change in RTOL and ATOL is unlikely to produce any change in the computed solution. (Only applies when the user is not operating in one step mode, that is when ITASK ≠ 2 or 5).

IFAIL = 14

The values of RTOL and ATOL are so small that the routine is unable to start the integration.

IFAIL = 15

The linear algebra setup routine D02NTF was not called prior to calling D02NCF.

7. Accuracy

The accuracy of the numerical solution may be controlled by a careful choice of the parameters RTOL and ATOL, and to a much lesser extent by the choice of norm. Users are advised to use scalar error control unless the components of the solution are expected to be poorly scaled. For the type of decaying solution typical of many stiff problems, relative error control with a small absolute error threshold will be most appropriate (that is the user is advised to choose ITOL = 1 with ATOL(1) small but positive).

8. Further Comments

The cost of computing a solution depends critically on the size of the differential system and to a lesser extent on the degree of stiffness of the problem. For D02NCF the cost is proportional to NEQ×(ML+MU+1)^2 though for problems which are only mildly nonlinear the cost may be dominated by factors proportional to NEQ×(ML+MU+1) except for very large problems.

In general the user is advised to choose the Backward Differentiation Formula option (setup routine D02NVF) but if efficiency is of great importance and especially if it is suspected that $\frac{\partial g}{\partial y}$ has complex eigenvalues near the imaginary axis for some part of the integration, the user should try the BLEND option (setup routine D02NWF).

9. Example

We solve the well-known stiff Robertson problem

$$\begin{aligned} a' &= -0.04a + 1.0\text{E}4bc \\ b' &= 0.04a - 1.0\text{E}4bc - 3.0\text{E}7b^2 \\ c' &= \phantom{-0.04a - 1.0\text{E}4bc - {}} 3.0\text{E}7b^2 \end{aligned}$$

over the range [0,10] with initial conditions $a = 1.0$ and $b = c = 0.0$ using scalar relative error control and vector absolute error control (ITOL = 2). We obtain the solution at TOUT = 5.0 and TOUT = 10.0 by overshooting and internal C^0 interpolation (ITASK = 1). D02NBY is used for MONITR, we use the BLEND integrator (setup routine D02NWF) and we choose the option of an analytical Jacobian.

9.1. Program Text

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

```
*     D02NCF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, NEQMAX, NRW, NINF, ML, MU, NJCPVT, NWKJAC,
     +                 MAXORD, NY2DIM, MAXSTP, MXHNIL
      PARAMETER        (NEQ=3,NEQMAX=NEQ,NRW=50+4*NEQMAX,NINF=23,ML=1,
     +                 MU=2,NJCPVT=NEQMAX,NWKJAC=NEQMAX*(2*ML+MU+1),
     +                 MAXORD=11,NY2DIM=MAXORD+3,MAXSTP=200,MXHNIL=5)
      real             H0, HMAX, HMIN, TCRIT
      PARAMETER        (H0=0.0e0,HMAX=10.0e0,HMIN=1.0e-10,TCRIT=0.0e0)
*     .. Local Scalars ..
      real             H, HU, T, TCUR, TOLSF, TOUT
      INTEGER          I, IFAIL, IMXER, ITASK, ITOL, ITRACE, NITER, NJE,
     +                 NQ, NQU, NRE, NST
```

```
*     .. Local Arrays ..
      real             ATOL(NEQMAX), CONST(6), RTOL(NEQMAX), RWORK(NRW),
     +                 WKJAC(NWKJAC), Y(NEQMAX), YDOT(NEQMAX),
     +                 YSAVE(NEQMAX,NY2DIM)
      INTEGER          INFORM(NINF), JACPVT(NJCPVT)
      LOGICAL          ALGEQU(NEQMAX)
*     .. External Subroutines ..
      EXTERNAL         D02NBY, D02NCF, D02NTF, D02NWF, D02NYF, D02NZF,
     +                 FCN, JAC, X04ABF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02NCF Example Program Results'
      CALL X04ABF(1,NOUT)
*
*     Integrate to TOUT (ITASK=1 i.e. overshooting and internal
*     interpolation) using the blend method. Default values for the
*     array CONST are used. Employ scalar relative tolerance and vector
*     absolute tolerance. The Jacobian is evaluated by JAC.
*     MONITR subroutine replaced by NAG dummy routine D02NBY.
*
      T = 0.0e0
      TOUT = 5.0e0
      ITASK = 1
      Y(1) = 1.0e0
      Y(2) = 0.0e0
      Y(3) = 0.0e0
      ITOL = 2
      RTOL(1) = 1.0e-4
      ATOL(1) = 1.0e-7
      ATOL(2) = 1.0e-8
      ATOL(3) = 1.0e-7
      DO 20 I = 1, 6
         CONST(I) = 0.0e0
   20 CONTINUE
      IFAIL = 0
*
      CALL D02NWF(NEQMAX,NY2DIM,MAXORD,CONST,TCRIT,HMIN,HMAX,H0,MAXSTP,
     +            MXHNIL,'Average-L2',RWORK,IFAIL)
      CALL D02NTF(NEQ,NEQMAX,'Analytical',ML,MU,NWKJAC,NJCPVT,RWORK,
     +            IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y(1)           Y(2)           Y(3)'
      WRITE (NOUT,99999) T, (Y(I),I=1,NEQ)
*
*     Soft fail and error messages only
      ITRACE = 0
      IFAIL = 1
*
      CALL D02NCF(NEQ,NEQMAX,T,TOUT,Y,YDOT,RWORK,RTOL,ATOL,ITOL,INFORM,
     +            FCN,YSAVE,NY2DIM,JAC,WKJAC,NWKJAC,JACPVT,NJCPVT,
     +            D02NBY,ITASK,ITRACE,IFAIL)
*
      IF (IFAIL.EQ.0) THEN
         WRITE (NOUT,99999) T, (Y(I),I=1,NEQ)
      ELSE
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'Exit D02NCF with IFAIL = ', IFAIL,
     +     '  and T = ', T
         STOP
      END IF
*     Reset TOUT and call D02NZF to override internal choice for
*     stepsize. No changes to other parameters.
      H = 0.7e0
      IFAIL = 0
*
```

```
      CALL D02NZF(NEQMAX,TCRIT,H,HMIN,HMAX,MAXSTP,MXHNIL,RWORK,IFAIL)
*
      TOUT = 10.0e0
      IFAIL = 1
*
      CALL D02NCF(NEQ,NEQMAX,T,TOUT,Y,YDOT,RWORK,RTOL,ATOL,ITOL,INFORM,
     +            FCN,YSAVE,NY2DIM,JAC,WKJAC,NWKJAC,JACPVT,NJCPVT,
     +            D02NBY,ITASK,ITRACE,IFAIL)
*
      IF (IFAIL.EQ.0) THEN
         WRITE (NOUT,99999) T, (Y(I),I=1,NEQ)
*
         CALL D02NYF(NEQ,NEQMAX,HU,H,TCUR,TOLSF,RWORK,NST,NRE,NJE,NQU,
     +               NQ,NITER,IMXER,ALGEQU,INFORM,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99997) ' HUSED = ', HU, '  HNEXT = ', H,
     +     '  TCUR = ', TCUR
         WRITE (NOUT,99996) ' NST = ', NST, '    NRE = ', NRE,
     +     '    NJE = ', NJE
         WRITE (NOUT,99996) ' NQU = ', NQU, '    NQ  = ', NQ,
     +     '  NITER = ', NITER
         WRITE (NOUT,99995) ' Max Err Comp = ', IMXER
         WRITE (NOUT,*)
      ELSE
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'Exit D02NCF with IFAIL = ', IFAIL,
     +     '  and T = ', T
      END IF
      STOP
*
99999 FORMAT (1X,F8.3,3(F13.5,2X))
99998 FORMAT (1X,A,I2,A,e12.5)
99997 FORMAT (1X,A,e12.5,A,e12.5,A,e12.5)
99996 FORMAT (1X,A,I6,A,I6,A,I6)
99995 FORMAT (1X,A,I4)
      END
*
      SUBROUTINE FCN(NEQ,T,Y,R,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IRES, NEQ
*     .. Array Arguments ..
      real              R(NEQ), Y(NEQ)
*     .. Executable Statements ..
      R(1) = -0.04e0*Y(1) + 1.0e4*Y(2)*Y(3)
      R(2) = 0.04e0*Y(1) - 1.0e4*Y(2)*Y(3) - 3.0e7*Y(2)*Y(2)
      R(3) = 3.0e7*Y(2)*Y(2)
      RETURN
      END
*
      SUBROUTINE JAC(NEQ,T,Y,H,D,ML,MU,P)
*     .. Scalar Arguments ..
      real              D, H, T
      INTEGER           ML, MU, NEQ
*     .. Array Arguments ..
      real              P(ML+MU+1,NEQ), Y(NEQ)
*     .. Local Scalars ..
      real              HXD
```

```
*     .. Executable Statements ..
      HXD = H*D
*
      P(1,1) = 1.0e0 - HXD*(-0.04e0)
      P(2,1) = -HXD*(1.0e4*Y(3))
      P(3,1) = -HXD*(1.0e4*Y(2))
*
      P(1,2) = -HXD*(0.04e0)
      P(2,2) = 1.0e0 - HXD*(-1.0e4*Y(3)-6.0e7*Y(2))
      P(3,2) = -HXD*(-1.0e4*Y(2))
*
      P(1,3) = -HXD*(6.0e7*Y(2))
      P(2,3) = 1.0e0 - HXD*(0.0e0)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D02NCF Example Program Results

     X           Y(1)           Y(2)           Y(3)
   0.000        1.00000        0.00000        0.00000
   5.000        0.89152        0.00002        0.10846
  10.000        0.84137        0.00002        0.15861

  HUSED =  0.10390E+01  HNEXT =  0.10390E+01  TCUR =  0.10102E+02
  NST =     80    NRE =    346    NJE =     18
  NQU =      4    NQ  =      4  NITER =    344
  Max Err Comp =    3
```

D02NDF – NAG Fortran Library Routine Document

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

1. Purpose

D02NDF is a forward communication routine for integrating stiff systems of explicit ordinary differential equations when the Jacobian is a sparse matrix.

2. Specification

```
 SUBROUTINE D02NDF (NEQ, NEQMAX, T, TOUT, Y, YDOT, RWORK, RTOL,
1                   ATOL, ITOL, INFORM, FCN, YSAVE, NY2DIM, JAC,
2                   WKJAC, NWKJAC, JACPVT, NJCPVT, MONITR, ITASK,
3                   ITRACE, IFAIL)
 INTEGER        NEQ, NEQMAX, ITOL, INFORM(23), NY2DIM, NWKJAC,
1               JACPVT(NJCPVT), NJCPVT, ITASK, ITRACE, IFAIL
 real           T, TOUT, Y(NEQMAX), YDOT(NEQMAX),
1               RWORK(50+4*NEQMAX), RTOL(*), ATOL(*),
2               YSAVE(NEQMAX,NY2DIM), WKJAC(NWKJAC)
 EXTERNAL       FCN, JAC, MONITR
```

3. Description

D02NDF is a general purpose routine for integrating the initial value problem for a stiff system of explicit ordinary differential equations,

$$y' = g(t,y)$$

It is designed specifically for the case where the Jacobian $\frac{\partial g}{\partial y}$ is a sparse matrix.

Both interval and step oriented modes of operation are available and also modes designed to permit intermediate output within an interval oriented mode.

An outline of a typical program calling D02NDF is given below. It calls the sparse matrix linear algebra setup routine D02NUF, and the Backward Differentiation Formula (BDF) integrator setup routine D02NVF, its diagnostic counterpart D02NYF, and the sparse linear algebra diagnositc routine D02NXF.

```
C
C     declarations
C
      EXTERNAL FCN, JAC, MONITR
            .
            .
            .
      IFAIL = 0
      CALL D02NVF(...,IFAIL)
      CALL D02NUF(NEQ, NEQMAX, JCEVAL, NWKJAC, IA, NIA, JA, NJA,
     + JACPVT, NJCPVT, SENS, U, ETA, LBLOCK, ISPLIT, RWORK,
     + IFAIL)
      IFAIL = -1
      CALL D02NDF(NEQ, NEQMAX, T, TOUT, Y, YDOT, RWORK, RTOL,
     + ATOL, ITOL, INFORM, FCN, YSAVE, NY2DIM, JAC, WKJAC,
     + NWKJAC, JACPVT, NJCPVT, MONITR, ITASK, ITRACE, IFAIL)
      IF(IFAIL.EQ.1 .OR .IFAIL.GE.14) STOP
      IFAIL = 0
      CALL D02NXF(...)
      CALL D02NYF(...)
            .
            .
            .
      STOP
      END
```

The linear algebra setup routine D02NUF and one of the integrator setup routines, D02NVF or D02NWF, must be called prior to the call of D02NDF. Either or both of the integrator diagnostic routine D02NYF, or the sparse matrix linear algebra diagnostic routine D02NXF, may be called after the call to D02NDF. There is also a routine, D02NZF, designed to permit the user to change step size on a continuation call to D02NDF without restarting the integration process.

4. References

See Subchapter Introduction.

5. Parameters

1: NEQ – INTEGER. *Input*

On entry: the number of differential equations to be solved.

Constraint: NEQ $\geq$ 1.

2: NEQMAX – INTEGER. *Input*

On entry: a bound on the maximum number of differential equations to be solved during the integration.

Constraint: NEQMAX $\geq$ NEQ.

3: T – ***real***. *Input/Output*

On entry: the value of the independent variable t. The input value of T is used only on the first call as the initial point of the integration.

On exit: the value at which the computed solution y is returned (usually at TOUT).

4: TOUT – ***real***. *Input*

On entry: the next value of t at which a computed solution is desired. For the initial t, the input value of TOUT is used to determine the direction of integration. Integration is permitted in either direction (see also ITASK).

Constraint: TOUT $\neq$ T.

5: Y(NEQMAX) – ***real*** array. *Input/Output*

On entry: the values of the dependent variables (solution). On the first call the first NEQ elements of Y must contain the vector of initial values.

On exit: the computed solution vector, evaluated at t (usually t = TOUT).

6: YDOT(NEQMAX) – ***real*** array. *Output*

On exit: the time derivatives y' of the vector y at the last integration point.

7: RWORK(50+4*NEQMAX) – ***real*** array. *Workspace*

8: RTOL(*) – ***real*** array. *Input*

Note: the dimension of RTOL must be at least 1 or NEQ (see ITOL).

On entry: the relative local error tolerance.

Constraint: RTOL(i) $\geq$ 0.0 for all relevant i (see ITOL).

9: ATOL(*) – ***real*** array. *Input*

Note: the dimension of ATOL must be at least 1 or NEQ (see ITOL).

On entry: the absolute local error tolerance.

Constraint: ATOL(i) $\geq$ 0.0 for all relevant i (see ITOL).

10: ITOL – INTEGER. *Input*

On entry: a value to indicate the form of the local error test. ITOL indicates to D02NDF whether to interpret either or both of RTOL or ATOL as a vector or a scalar. The error test to be satisfied is $\|e_i/w_i\| < 1.0$ where w_i is defined as follows:

ITOL	RTOL	ATOL	w_i
1	scalar	scalar	$\mathrm{RTOL}(1)\times\lvert y_i\rvert + \mathrm{ATOL}(1)$
2	scalar	vector	$\mathrm{RTOL}(1)\times\lvert y_i\rvert + \mathrm{ATOL}(i)$
3	vector	scalar	$\mathrm{RTOL}(i)\times\lvert y_i\rvert + \mathrm{ATOL}(1)$
4	vector	vector	$\mathrm{RTOL}(i)\times\lvert y_i\rvert + \mathrm{ATOL}(i)$

e_i is an estimate of the local error in y_i, computed internally, and the choice of norm to be used is defined by a previous call to an integrator setup routine.

Constraint: $1 \le \mathrm{ITOL} \le 4$.

11: INFORM(23) – INTEGER array. *Workspace*

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

FCN must evaluate the derivative vector for the explicit ordinary differential equation system, defined by $y' = g(t,y)$.

Its specification is:

```
SUBROUTINE FCN(NEQ, T, Y, F, IRES)
INTEGER    NEQ, IRES
real       T, Y(NEQ), F(NEQ)
```

1: NEQ – INTEGER. *Input*

On entry: the number of differential equations being solved.

2: T – ***real***. *Input*

On entry: the current value of the independent variable t.

3: Y(NEQ) – ***real*** array. *Input*

On entry: the value of y_i, for $i = 1,2,\ldots,\mathrm{NEQ}$.

4: F(NEQ) – ***real*** array. *Output*

On exit: the value y_i', given by $y_i' = g_i(t,y)$, for $i = 1,2,\ldots,\mathrm{NEQ}$.

5: IRES – INTEGER. *Input/Output*

On entry: IRES = 1.

On exit: the user may set IRES as follows to indicate certain conditions in FCN to the integrator:

IRES = 1

indicates a normal return from FCN, that is IRES is not altered by the user and integration continues.

IRES = 2

indicates to the integrator that control should be passed back immediately to the calling (sub)program with the error indicator set to IFAIL = 11.

IRES = 3

indicates to the integrator that an error condition has occurred in the solution vector, its time derivative or in the value of t. The integrator will use a smaller time step to try to avoid this condition. If this is not possible, the integrator returns to the calling (sub)program with the error indicator set to IFAIL = 7.

IRES = 4

indicates to the integrator to stop its current operation and to enter the MONITR routine immediately with parameter IMON = –2.

FCN must be declared as EXTERNAL in the (sub)program from which D02NDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

13: YSAVE(NEQMAX, NY2DIM) – ***real*** array. *Workspace*

14: NY2DIM – INTEGER. *Input*

On entry: the second dimension of the array YSAVE as declared in the (sub)program from which D02NDF is called. An appropriate value for NY2DIM is described in the specifications of the integrator setup routines D02NVF and D02NWF. This value must be the same as that supplied to the integrator setup routine.

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

JAC must evaluate the Jacobian of the system. If this option is not required, the actual argument for JAC must be the dummy routine D02NDZ. (D02NDZ is included in the NAG Fortran Library and so need not be supplied by the user. Its name may be implementation dependent: see the Users' Note for your implementation for details.) The user indicates to the integrator whether this option is to be used by setting the parameter JCEVAL appropriately in a call to the sparse matrix linear algebra setup routine D02NUF.

First we must define the system of nonlinear equations which is solved internally by the integrator. The time derivative, y', generated internally has the form

$$y' = (y-z)/(hd),$$

where h is the current stepsize and d is a parameter that depends on the integration method in use. The vector y is the current solution and the vector z depends on information from previous time steps. This means that $\frac{d}{dy'}(\ ) = \frac{1}{(hd)}\frac{d}{dy}(\ )$. The system of nonlinear equations that is solved has the form

$$y' - g(t,y) = 0$$

but is solved in the form

$$r(t,y) = 0,$$

where the function r is defined by

$$r(t,y) = hd((y-z)/(hd)-g(t,y)).$$

It is the Jacobian matrix $\frac{\partial r}{\partial y}$ that the user must supply in the routine JAC as follows:

$$\frac{\partial r_i}{\partial y_j} = 1 - (hd)\frac{\partial g_i}{\partial y_j} \qquad \text{if } i = j,$$

$$\frac{\partial r_i}{\partial y_j} = -(hd)\frac{\partial g_i}{\partial y_j} \qquad \text{otherwise.}$$

Its specification is:

```
SUBROUTINE JAC(NEQ, T, Y, H, D, J, PDJ)
INTEGER    NEQ, J
real       T, Y(NEQ), H, D, PDJ(NEQ)
```

1: NEQ – INTEGER. *Input*

On entry: the number of differential equations being solved.

2: T – ***real***. *Input*

On entry: the current value of the independent variable t.

3: Y(NEQ) – ***real*** array. *Input*

On entry: the current solution component y_i, for $i = 1,2,...,$NEQ.

4: H – ***real.*** *Input*

On entry: the current stepsize.

5: D – ***real.*** *Input*

On entry: the parameter d which depends upon the integration method.

6: J – INTEGER. *Input*

On entry: the column of the Jacobian that JAC must return in the array PDJ.

7: PDJ(NEQ) – ***real*** array. *Output*

On exit: PJD(i) should be set to the (i,j)th element of the Jacobian, where j is given by J above. Only non-zero elements of this array need be set, since it is preset to zero before the call to JAC.

JAC must be declared as EXTERNAL in the (sub)program from which D02NDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

16: WKJAC(NWKJAC) – ***real*** array. *Workspace*

17: NWKJAC – INTEGER. *Input*

On entry: the dimension of the array WKJAC as declared in the (sub)program from which D02NDF is called. The actual size depends on whether the sparsity structure is supplied or whether it is to be estimated. An appropriate value for NWKJAC is described in the specification of the linear algebra setup routine D02NUF. This value must be the same as that supplied to D02NUF.

18: JACPVT(NJCPVT) – INTEGER array. *Workspace*

19: NJCPVT – INTEGER. *Input*

On entry: the dimension of the array JACPVT as declared in the (sub)program from which D02NDF is called. The actual size depends on whether the sparsity structure is supplied or whether it is to be estimated. An appropriate value for NJCPVT is described in the specification of the linear algebra setup routine D02NUF. This value must be the same as that supplied to D02NUF.

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

MONITR performs tasks requested by the user. If this option is not required, the actual argument for MONITR must be the dummy routine D02NBY. (D02NBY is included in the NAG Fortran Library and so need not be supplied by the user. Its name may be implementation dependent: see the Users' Note for your implementation for details.)

Its specification is:

```
 SUBROUTINE MONITR(NEQ, NEQMAX, T, HLAST, HNEXT, Y, YDOT, YSAVE, R,
1           ACOR, IMON, INLN, HMIN, HMAX, NQU)
 INTEGER    NEQ, NEQMAX, IMON, INLN, NQU
 real       T, HLAST, HNEXT, Y(NEQMAX), YDOT(NEQMAX), YSAVE(NEQMAX,*),
1           R(NEQMAX), ACOR(NEQMAX,2), HMIN, HMAX
```

1: NEQ – INTEGER. *Input*

On entry: the number of differential equations being solved.

2: NEQMAX – INTEGER. *Input*

On entry: an upper bound on the number of differential equations to be solved.

3: T – ***real.*** *Input*

On entry: the current value of the independent variable.

4: HLAST – ***real***. *Input*

On entry: the last stepsize successfully used by the integrator.

5: HNEXT – ***real***. *Input/Output*

On entry: the stepsize that the integrator proposes to take on the next step.

On exit: the next stepsize to be used. If this is different from the input value, then IMON must be set to 4.

6: Y(NEQMAX) – ***real*** array. *Input/Output*

On entry: the values of the dependent variables, y, evaluated at t.

On exit: these values must not be changed unless IMON is set to 2.

7: YDOT(NEQMAX) – ***real*** array. *Input*

On entry: the time derivatives y' of the vector y.

8: YSAVE(NEQMAX,*) – ***real*** array. *Input*

On entry: workspace to enable the user to carry out interpolation using either of the routines D02XJF or D02XKF.

9: R(NEQMAX) – ***real*** array. *Input*

On entry: if IMON = 0 and INLN = 3, the first NEQ elements contain the residual vector $y' - g(t,y)$.

10: ACOR(NEQMAX,2) – ***real*** array. *Input*

On entry: with IMON = 1, ACOR(i,1) contains the weight used for the ith equation when the norm is evaluated, and ACOR(i,2) contains the estimated local error for the ith equation. The scaled local error at the end of a timestep may be obtained by calling the ***real*** function D02ZAF as follows

```
      IFAIL = 1
      ERRLOC = D02ZAF(NEQ, ACOR(1,2), ACOR(1,1), IFAIL)
C     CHECK IFAIL BEFORE PROCEEDING
```

11: IMON – INTEGER. *Input/Output*

On entry: a flag indicating under what circumstances MONITR was called:

IMON = −2

entry from the integrator after IRES = 4 (set in FCN) caused an early termination (this facility could be used to locate discontinuities).

IMON = −1

the current step failed repeatedly.

IMON = 0

entry after a call to the internal nonlinear equation solver (see below).

IMON = 1

the current step was successful.

On exit: IMON may be re-set to determine subsequent action in D02NDF:

IMON = −2

integration is to be halted. A return will be made from the integrator to the calling (sub)program with IFAIL = 12.

IMON = −1

allow the integrator to continue with its own internal strategy. The integrator will try up to 3 restarts ur'ess IMON is set ≠ −1 on exit.

IMON = 0

return to the internal nonlinear equation solver, where the action taken is determined by the value of INLN (see below).

IMON = 1

normal exit to the integrator to continue integration.

IMON = 2

restart the integration at the current time point. The integrator will restart from order 1 when this option is used. The MONITR provided solution Y will be used for the initial conditions.

IMON = 3

try to continue with the same stepsize and order as was to be used before the call to MONITR. HMIN and HMAX may be altered if desired.

IMON = 4

continue the integration but using a new value HNEXT and possibly new values of HMIN and HMAX.

12: INLN – INTEGER. *Output*

On exit: the action to be taken by the internal nonlinear equation solver when MONITR is exited with IMON = 0. By setting INLN = 3 and returning to the integrator, the residual vector is evaluated and placed in the array R, and then MONITR is called again. At present this is the only option available: INLN must not be set to any other value.

13: HMIN – ***real.*** *Input/Output*

On entry: the minimum stepsize to be taken on the next step.

On exit: the minimum stepsize to be used. If this is different from the input value, then IMON must be set to 3 or 4.

14: HMAX – ***real.*** *Input/Output*

On entry: the maximum stepsize to be taken on the next step.

On exit: the maximum stepsize to be used. If this is different from the input value, then IMON must be set to 3 or 4. If HMAX is set to zero, no limit is assumed.

15: NQU – INTEGER. *Input*

On entry: the order of the integrator used on the last step. This is supplied to enable the user to carry out interpolation using either of the routines D02XJF or D02XKF.

MONITR must be declared as EXTERNAL in the (sub)program from which D02NDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

21: ITASK – INTEGER. *Input*

On entry: the task to be performed by the integrator. The permitted values for ITASK and their meanings are detailed below:

ITASK = 1

normal computation of output values of $y(t)$ at t = TOUT (by overshooting and interpolating).

ITASK = 2

take one step only and return.

ITASK = 3

stop at the first internal integration point at or beyond t = TOUT and return.

ITASK = 4

normal computation of output values of $y(t)$ at t = TOUT but without overshooting t = TCRIT. TCRIT must be specified as an option in one of the integrator setup routines prior to the first call to the integrator, or specified in the optional input routine prior to a continuation call. TCRIT may be equal to or beyond TOUT, but not before it, in the direction of integration.

ITASK = 5

take one step only and return, without passing TCRIT. TCRIT must be specified as under ITASK = 4.

Constraint: $1 \leq$ ITASK ≤ 5.

22: ITRACE – INTEGER. *Input*

On entry: the level of output that is printed by the integrator. ITRACE may take the value -1, 0, 1, 2 or 3. If ITRACE < -1, then -1 is assumed and similarly if ITRACE > 3, then 3 is assumed. If ITRACE $= -1$, no output is generated. If ITRACE = 0, only warning messages are printed on the current error message unit (see X04AAF). If ITRACE > 0, then warning messages are printed as above, and on the current advisory message unit (see X04ABF) output is generated which details Jacobian entries, the nonlinear iteration and the time integration. The advisory messages are given in greater detail the larger the value of ITRACE.

23: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq 0$ on exit, users are recommended to set IFAIL to -1 before entry. **It is then essential to test the value of IFAIL on exit.** To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL = 1

An illegal input was detected on entry, or after an internal call to MONITR. If ITRACE > -1, then the form of the error will be detailed on the current error message unit (see X04AAF).

IFAIL = 2

The maximum number of steps specified has been taken (see the description of optional inputs in the integrator setup routines and the optional input continuation routine, D02NZF).

IFAIL = 3

With the given values of RTOL and ATOL no further progress can be made across the integration range from the current point T. The components Y(1),Y(2),...,Y(NEQ) contain the computed values of the solution at the current point T.

IFAIL = 4

There were repeated error test failures on an attempted step, before completing the requested task, but the integration was successful as far as T. The problem may have a singularity, or the local error requirements may be inappropriate.

IFAIL = 5

There were repeated convergence test failures on an attempted step, before completing the requested task, but the integration was successful as far as T. This may be caused by an inaccurate Jacobian matrix or one which is incorrectly computed.

IFAIL = 6

Some error weight w_i became zero during the integration (see description of ITOL). Pure relative error control (ATOL(i) = 0.0) was requested on a variable (the ith) which has now vanished. The integration was successful as far as T.

IFAIL = 7

The user-supplied subroutine FCN set its error flag (IRES = 3) continually despite repeated attempts by the integrator to avoid this.

IFAIL = 8

Not used for the integrator.

IFAIL = 9

A singular Jacobian $\frac{\partial r}{\partial y}$ has been encountered. This error exit is unlikely to be taken when solving explicit ordinary differential equations. The user should check his problem formulation and Jacobian calculation.

IFAIL = 10

An error occurred during Jacobian formulation or backsubstitution (a more detailed error description may be directed to the current error message unit, see X04AAF).

IFAIL = 11

The user-supplied subroutine FCN signalled the integrator to halt the integration and return (IRES = 2). Integration was successful as far as T.

IFAIL = 12

The user-supplied subroutine MONITR set IMON = –2 and so forced a return but the integration was successful as far as T.

IFAIL = 13

The requested task has been completed, but it is estimated that a small change in RTOL and ATOL is unlikely to produce any change in the computed solution. (Only applies when the user is not operating in one step mode, that is when ITASK $\neq$ 2 or 5).

IFAIL = 14

The values of RTOL and ATOL are so small that the routine is unable to start the integration.

IFAIL = 15

The linear algebra setup routine D02NUF was not called prior to calling D02NDF.

7. Accuracy

The accuracy of the numerical solution may be controlled by a careful choice of the parameters RTOL and ATOL, and to a much lesser extent by the choice of norm. Users are advised to use scalar error control unless the components of the solution are expected to be poorly scaled. For the type of decaying solution typical of many stiff problems, relative error control with a small absolute error threshold will be most appropriate (that is the user is advised to choose ITOL = 1 with ATOL(1) small but positive).

8. Further Comments

Since numerical stability and memory are often conflicting requirements when solving ordinary differential systems where the Jacobian matrix is sparse, we provide a diagnostic routine, D02NXF, whose aim is to inform the user how much memory is required to solve his problem and to give the user some indicators of numerical stability.

In general the user is advised to choose the backward differentiation formula option (setup routine D02NVF) but if efficiency is of great importance and especially if it is suspected that $\frac{\partial g}{\partial y}$ has complex eigenvalues near the imaginary axis for some part of the integration, the user should try the BLEND option (setup routine D02NWF).

9. Example

We solve the well-known stiff Robertson problem

$$\begin{aligned} a' &= -0.04a + 1.0\text{E}4bc \\ b' &= 0.04a - 1.0\text{E}4bc - 3.0\text{E}7b^2 \\ c' &= \phantom{-0.04a - 1.0\text{E}4bc -{}} 3.0\text{E}7b^2 \end{aligned}$$

over the range [0,10.0] with initial conditions $a = 1.0$ and $b = c = 0.0$ using scalar error control (ITOL = 1). We compute the solution up to 10.0 by overshooting and interpolating (ITASK = 1) and we compute the intermediate solution on an equispaced mesh through a user supplied MONITR routine. The integration algorithm used is the BDF method (setup routine D02NVF) and we use a modified Newton method. We illustrate the use of the 'N' (Numerical) and 'S' (Structural) options in turn for calculating the Jacobian.

9.1. Program Text

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

```
*     D02NDF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, NEQMAX, NRW, NINF, NELTS, NJCPVT, NWKJAC,
     +                 NIA, NJA, MAXORD, NY2DIM, MAXSTP, MXHNIL
      PARAMETER        (NEQ=3,NEQMAX=NEQ,NRW=50+4*NEQMAX,NINF=23,
     +                 NELTS=8,NJCPVT=150,NWKJAC=100,NIA=NEQMAX+1,
     +                 NJA=NELTS,MAXORD=5,NY2DIM=MAXORD+1,MAXSTP=200,
     +                 MXHNIL=5)
      real             H0, HMAX, HMIN, TCRIT
      PARAMETER        (H0=0.0e0,HMAX=10.0e0,HMIN=1.0e-10,TCRIT=0.0e0)
      LOGICAL          PETZLD
      PARAMETER        (PETZLD=.FALSE.)
      real             ETA, U, SENS
      PARAMETER        (ETA=1.0e-4,U=0.1e0,SENS=0.0e0)
      LOGICAL          LBLOCK
      PARAMETER        (LBLOCK=.TRUE.)
*     .. Scalars in Common ..
      real             XOUT
*     .. Local Scalars ..
      real             H, HU, T, TCUR, TOLSF, TOUT
      INTEGER          I, ICALL, IFAIL, IGROW, IMXER, ISPLIT, ITASK,
     +                 ITOL, ITRACE, LIWREQ, LIWUSD, LRWREQ, LRWUSD,
     +                 NBLOCK, NGP, NITER, NJE, NLU, NNZ, NQ, NQU, NRE,
     +                 NST
*     .. Local Arrays ..
      real             ATOL(NEQMAX), CONST(6), RTOL(NEQMAX), RWORK(NRW),
     +                 WKJAC(NWKJAC), Y(NEQMAX), YDOT(NEQMAX),
     +                 YSAVE(NEQMAX,NY2DIM)
      INTEGER          IA(NIA), INFORM(NINF), JA(NJA), JACPVT(NJCPVT)
      LOGICAL          ALGEQU(NEQMAX)
*     .. External Subroutines ..
      EXTERNAL         D02NDF, D02NDZ, D02NUF, D02NVF, D02NXF, D02NYF,
     +                 FCN, MONITR, X04ABF
*     .. Common blocks ..
      COMMON           XOUT
```

```
*     .. Data statements ..
      DATA             IA/1, 3, 6, 9/, JA/1, 2, 1, 2, 3, 1, 2, 3/
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02NDF Example Program Results'
      CALL X04ABF(1,NOUT)
*
*     First case. Integrate to TOUT by overshooting (ITASK=1) using
*     using B.D.F formulae with a Newton method. Default values for the
*     array CONST are used. Employ scalar relative tolerance and scalar
*     absolute tolerance. The Jacobian and its structure are evaluated
*     internally. Carry out interpolation in the MONITR routine using
*     the NAG routine D02XKF.
*
      T = 0.0e0
      TOUT = 10.0e0
      ITASK = 1
      Y(1) = 1.0e0
      Y(2) = 0.0e0
      Y(3) = 0.0e0
      ITOL = 1
      RTOL(1) = 1.0e-4
      ATOL(1) = 1.0e-7
      DO 20 I = 1, 6
         CONST(I) = 0.0e0
   20 CONTINUE
      ISPLIT = 0
      IFAIL = 0
*
      CALL D02NVF(NEQMAX,NY2DIM,MAXORD,'Newton',PETZLD,CONST,TCRIT,HMIN,
     +            HMAX,H0,MAXSTP,MXHNIL,'Average-L2',RWORK,IFAIL)
*
      CALL D02NUF(NEQ,NEQMAX,'Numerical',NWKJAC,IA,NIA,JA,NJA,JACPVT,
     +            NJCPVT,SENS,U,ETA,LBLOCK,ISPLIT,RWORK,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) '    Numerical Jacobian, structure not supplied'
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y(1)           Y(2)           Y(3)'
      WRITE (NOUT,99999) T, (Y(I),I=1,NEQ)
      XOUT = 2.0e0
*
*     Soft fail and error messages only
      ITRACE = 0
      IFAIL = 1
*
      CALL D02NDF(NEQ,NEQMAX,T,TOUT,Y,YDOT,RWORK,RTOL,ATOL,ITOL,INFORM,
     +            FCN,YSAVE,NY2DIM,D02NDZ,WKJAC,NWKJAC,JACPVT,NJCPVT,
     +            MONITR,ITASK,ITRACE,IFAIL)
*
      IF (IFAIL.EQ.0) THEN
*
         CALL D02NYF(NEQ,NEQMAX,HU,H,TCUR,TOLSF,RWORK,NST,NRE,NJE,NQU,
     +               NQ,NITER,IMXER,ALGEQU,INFORM,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99997) ' HUSED = ', HU, '  HNEXT = ', H,
     +     '  TCUR = ', TCUR
         WRITE (NOUT,99996) ' NST = ', NST, '    NRE = ', NRE,
     +     '    NJE = ', NJE
         WRITE (NOUT,99996) ' NQU = ', NQU, '    NQ  = ', NQ,
     +     '  NITER = ', NITER
         WRITE (NOUT,99995) ' Max err comp = ', IMXER
         WRITE (NOUT,*)
         ICALL = 0
*
```

```
         CALL D02NXF(ICALL,LIWREQ,LIWUSD,LRWREQ,LRWUSD,NLU,NNZ,NGP,
     +               ISPLIT,IGROW,LBLOCK,NBLOCK,INFORM)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99994) ' NJCPVT (required ', LIWREQ, '  used ',
     +     LIWUSD, ')'
         WRITE (NOUT,99994) ' NWKJAC (required ', LRWREQ, '  used ',
     +     LRWUSD, ')'
         WRITE (NOUT,99993) ' No. of LU-decomps ', NLU,
     +     '  No. of nonzeros ', NNZ
         WRITE (NOUT,99995) ' No. of FCN calls to form Jacobian ', NGP,
     +     '  Try ISPLIT ', ISPLIT
         WRITE (NOUT,99992) ' Growth est ', IGROW,
     +     '  No. of blocks on diagonal ', NBLOCK
      ELSE IF (IFAIL.EQ.10) THEN
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'Exit D02NDF with IFAIL = ', IFAIL,
     +     '  and T = ', T
         ICALL = 1
*
         CALL D02NXF(ICALL,LIWREQ,LIWUSD,LRWREQ,LRWUSD,NLU,NNZ,NGP,
     +               ISPLIT,IGROW,LBLOCK,NBLOCK,INFORM)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99994) ' NJCPVT (required ', LIWREQ, '  used ',
     +     LIWUSD, ')'
         WRITE (NOUT,99994) ' NWKJAC (required ', LRWREQ, '  used ',
     +     LRWUSD, ')'
      ELSE
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'Exit D02NDF with IFAIL = ', IFAIL,
     +     '  and T = ', T
      END IF
*
*     Second case. Integrate to TOUT by overshooting (ITASK=1) using
*     B.D.F formulae with a Newton method. Default values for the
*     array CONST are used. Employ scalar relative tolerance and scalar
*     absolute tolerance. The Jacobian is evaluated internally but its
*     structure is supplied. Carry out interpolation in the
*     MONITR routine using the NAG routine D02XKF.
*
      T = 0.0e0
      Y(1) = 1.0e0
      Y(2) = 0.0e0
      Y(3) = 0.0e0
      ISPLIT = 0
      IFAIL = 0
*
      CALL D02NVF(NEQMAX,NY2DIM,MAXORD,'Newton',PETZLD,CONST,TCRIT,HMIN,
     +            HMAX,H0,MAXSTP,MXHNIL,'Average-L2',RWORK,IFAIL)
      CALL D02NUF(NEQ,NEQMAX,'Structural',NWKJAC,IA,NIA,JA,NJA,JACPVT,
     +            NJCPVT,SENS,U,ETA,LBLOCK,ISPLIT,RWORK,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) '  Numerical Jacobian, structure supplied'
      WRITE (NOUT,*)
      WRITE (NOUT,*) '    X          Y(1)           Y(2)           Y(3)'
      WRITE (NOUT,99999) T, (Y(I),I=1,NEQ)
      XOUT = 2.0e0
*
*     Soft fail and error messages only
      ITRACE = 0
      IFAIL = 1
*
```

```
      CALL D02NDF(NEQ,NEQMAX,T,TOUT,Y,YDOT,RWORK,RTOL,ATOL,ITOL,INFORM,
     +            FCN,YSAVE,NY2DIM,D02NDZ,WKJAC,NWKJAC,JACPVT,NJCPVT,
     +            MONITR,ITASK,ITRACE,IFAIL)
*
      IF (IFAIL.EQ.0) THEN
*
         CALL D02NYF(NEQ,NEQMAX,HU,H,TCUR,TOLSF,RWORK,NST,NRE,NJE,NQU,
     +               NQ,NITER,IMXER,ALGEQU,INFORM,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99997) ' HUSED = ', HU, '  HNEXT = ', H,
     +     '  TCUR = ', TCUR
         WRITE (NOUT,99996) ' NST = ', NST, '    NRE = ', NRE,
     +     '    NJE = ', NJE
         WRITE (NOUT,99996) ' NQU = ', NQU, '    NQ  = ', NQ,
     +     '  NITER = ', NITER
         WRITE (NOUT,99995) ' Max err comp = ', IMXER
         WRITE (NOUT,*)
         ICALL = 0
*
         CALL D02NXF(ICALL,LIWREQ,LIWUSD,LRWREQ,LRWUSD,NLU,NNZ,NGP,
     +               ISPLIT,IGROW,LBLOCK,NBLOCK,INFORM)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99994) ' NJCPVT (required ', LIWREQ, '  used ',
     +     LIWUSD, ')'
         WRITE (NOUT,99994) ' NWKJAC (required ', LRWREQ, '  used ',
     +     LRWUSD, ')'
         WRITE (NOUT,99993) ' No. of LU-decomps ', NLU,
     +     '  No. of nonzeros ', NNZ
         WRITE (NOUT,99995) ' No. of FCN calls to form Jacobian ', NGP,
     +     '  Try ISPLIT ', ISPLIT
         WRITE (NOUT,99992) ' Growth est ', IGROW,
     +     '  No. of blocks on diagonal ', NBLOCK
      ELSE IF (IFAIL.EQ.10) THEN
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'Exit D02NDF with IFAIL = ', IFAIL,
     +     '  and T = ', T
         ICALL = 1
*
         CALL D02NXF(ICALL,LIWREQ,LIWUSD,LRWREQ,LRWUSD,NLU,NNZ,NGP,
     +               ISPLIT,IGROW,LBLOCK,NBLOCK,INFORM)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99994) ' NJCPVT (required ', LIWREQ, '  used ',
     +     LIWUSD, ')'
         WRITE (NOUT,99994) ' NWKJAC (required ', LRWREQ, '  used ',
     +     LRWUSD, ')'
      ELSE
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'Exit D02NDF with IFAIL = ', IFAIL,
     +     '  and T = ', T
      END IF
      STOP
*
99999 FORMAT (1X,F8.3,3(F13.5,2X))
99998 FORMAT (1X,A,I2,A,e12.5)
99997 FORMAT (1X,A,e12.5,A,e12.5,A,e12.5)
99996 FORMAT (1X,A,I6,A,I6,A,I6)
99995 FORMAT (1X,A,I4,A,I4)
99994 FORMAT (1X,A,I8,A,I8,A)
99993 FORMAT (1X,A,I4,A,I8)
99992 FORMAT (1X,A,I8,A,I4)
      END
*
```

```
      SUBROUTINE FCN(NEQ,T,Y,R,IRES)
*     .. Scalar Arguments ..
      real            T
      INTEGER         IRES, NEQ
*     .. Array Arguments ..
      real            R(NEQ), Y(NEQ)
*     .. Executable Statements ..
      R(1) = -0.04e0*Y(1) + 1.0e4*Y(2)*Y(3)
      R(2) = 0.04e0*Y(1) - 1.0e4*Y(2)*Y(3) - 3.0e7*Y(2)*Y(2)
      R(3) = 3.0e7*Y(2)*Y(2)
      RETURN
      END
*
      SUBROUTINE MONITR(N,NMAX,T,HLAST,H,Y,YDOT,YSAVE,R,ACOR,IMON,INLN,
     +                  HMIN,HMXI,NQU)
*     .. Parameters ..
      INTEGER           NOUT
      PARAMETER         (NOUT=6)
      INTEGER           NY2DIM
      PARAMETER         (NY2DIM=6)
*     .. Scalar Arguments ..
      real              H, HLAST, HMIN, HMXI, T
      INTEGER           IMON, INLN, N, NMAX, NQU
*     .. Array Arguments ..
      real              ACOR(NMAX,2), R(N), Y(N), YDOT(N), YSAVE(NMAX,*)
*     .. Scalars in Common ..
      real              XOUT
*     .. Local Scalars ..
      INTEGER           I, IFAIL
*     .. External Subroutines ..
      EXTERNAL          D02XKF
*     .. Common blocks ..
      COMMON            XOUT
*     .. Executable Statements ..
      IF (IMON.NE.1) RETURN
   20 IF ( .NOT. (T-HLAST.LT.XOUT .AND. XOUT.LE.T)) RETURN
      IFAIL = 1
*     C1 interpolation
      CALL D02XKF(XOUT,R,N,YSAVE,NMAX,NY2DIM,ACOR(1,2),N,T,NQU,HLAST,H,
     +            IFAIL)
*
      IF (IFAIL.NE.0) THEN
         IMON = -2
      ELSE
         WRITE (NOUT,99999) XOUT, (R(I),I=1,N)
         XOUT = XOUT + 2.0e0
         IF (XOUT.LT.10.0e0) GO TO 20
      END IF
      RETURN
*
99999 FORMAT (1X,F8.3,3(F13.5,2X))
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D02NDF Example Program Results

 Numerical Jacobian, structure not supplied

    X           Y(1)           Y(2)           Y(3)
  0.000       1.00000        0.00000        0.00000
  2.000       0.94161        0.00003        0.05836
  4.000       0.90552        0.00002        0.09446
  6.000       0.87927        0.00002        0.12072
  8.000       0.85855        0.00002        0.14144
 10.000       0.84137        0.00002        0.15863

HUSED =  0.90236E+00  HNEXT =  0.90236E+00  TCUR =  0.10769E+02
NST =      55    NRE =    134    NJE =      16
NQU =       4    NQ  =      4  NITER =      78
Max err comp =    3


NJCPVT (required       100  used       150)
NWKJAC (required        29  used        71)
No. of LU-decomps   16  No. of nonzeros        7
No. of FCN calls to form Jacobian    3  Try ISPLIT   73
Growth est      1108  No. of blocks on diagonal    1

 Numerical Jacobian, structure supplied

    X           Y(1)           Y(2)           Y(3)
  0.000       1.00000        0.00000        0.00000
  2.000       0.94161        0.00003        0.05836
  4.000       0.90551        0.00002        0.09446
  6.000       0.87926        0.00002        0.12072
  8.000       0.85854        0.00002        0.14144
 10.000       0.84136        0.00002        0.15863

HUSED =  0.90178E+00  HNEXT =  0.90178E+00  TCUR =  0.10766E+02
NST =      55    NRE =    128    NJE =      16
NQU =       4    NQ  =      4  NITER =      78
Max err comp =    3


NJCPVT (required       106  used       150)
NWKJAC (required        31  used        70)
No. of LU-decomps   16  No. of nonzeros        8
No. of FCN calls to form Jacobian    3  Try ISPLIT   73
Growth est    277504  No. of blocks on diagonal    1
```

D02NGF – NAG Fortran Library Routine Document

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

1. Purpose

D02NGF is a forward communication routine for integrating stiff systems of implicit ordinary differential equations coupled with algebraic equations when the Jacobian is a full matrix.

2. Specification

```
 SUBROUTINE D02NGF (NEQ, NEQMAX, T, TOUT, Y, YDOT, RWORK, RTOL,
1                   ATOL, ITOL, INFORM, RESID, YSAVE, NY2DIM, JAC,
2                   WKJAC, NWKJAC, MONITR, LDERIV, ITASK, ITRACE,
3                   IFAIL)
 INTEGER          NEQ, NEQMAX, ITOL, INFORM(23), NY2DIM, NWKJAC,
1                 ITASK, ITRACE, IFAIL
 real             T, TOUT, Y(NEQMAX), YDOT(NEQMAX),
1                 RWORK(50+4*NEQMAX), RTOL(*), ATOL(*),
2                 YSAVE(NEQMAX,NY2DIM), WKJAC(NWKJAC)
 LOGICAL          LDERIV(2)
 EXTERNAL         RESID, JAC, MONITR
```

3. Description

D02NGF is a general purpose routine for integrating the initial value problem for a stiff system of implicit ordinary differential equations with coupled algebraic equations written in the form,

$$A(t,y)y' = g(t,y)$$

It is designed specifically for the case where the resulting Jacobian is a full matrix (see description of argument JAC in Section 5).

Both interval and step oriented modes of operation are available and also modes designed to permit intermediate output within an interval oriented mode.

An outline of a typical calling program for D02NGF is given below. It calls the full matrix linear algebra setup routine D02NSF, and the Backward Differentiation Formula (BDF) integrator setup routine D02NVF, and its diagnostic counterpart D02NYF.

```
C
C     declarations
C
      EXTERNAL RESID, JAC, MONITR
              .
              .
              .
      IFAIL = 0
      CALL D02NVF(...,IFAIL)
      CALL D02NSF(NEQ, NEQMAX, JCEVAL, NWKJAC, RWORK, IFAIL)
      IFAIL = -1
      CALL D02NGF(NEQ, NEQMAX, T, TOUT, Y, YDOT, RWORK, RTOL,
     + ATOL, ITOL, INFORM, RESID, YSAVE, NY2DIM, JAC, WKJAC,
     + NWKJAC, MONITR, LDERIV, ITASK, ITRACE, IFAIL)
      IF (IFAIL.EQ.1 .OR. IFAIL.GE.14) STOP
      IFAIL = 0
      CALL D02NYF(...)
              .
              .
              .
      STOP
      END
```

The linear algebra setup routine, D02NSF, and one of the integrator setup routines, D02MVF, D02NVF or D02NWF must be called prior to the call of D02NGF. The integrator diagnostic routine D02NYF may be called after the call to D02NGF. There is also a routine, D02NZF, designed to permit the user to change step size on a continuation call to D02NGF without restarting the integration process.

4. References

See Subchapter Introduction.

5. Parameters

1: NEQ – INTEGER. *Input*

On entry: the number of equations to be solved.

Constraint: NEQ ≥ 1.

2: NEQMAX – INTEGER. *Input*

On entry: a bound on the maximum number of equations to be solved during the integration.

Constraint: NEQMAX ≥ NEQ.

3: T – ***real***. *Input/Output*

On entry: the value of the independent variable t. The input value of T is used only on the first call as the initial point of the integration.

On exit: the value at which the computed solution y is returned (usually at TOUT).

4: TOUT – ***real***. *Input/Output*

On entry: the next value of t at which a computed solution is desired. For the initial t, the input value of TOUT is used to determine the direction of integration. Integration is permitted in either direction (see also ITASK).

Constraint: TOUT ≠ T.

On exit: normally unchanged. However when ITASK = 6, then TOUT contains the value of T at which initial values have been computed without performing any integration. See descriptions of ITASK and LDERIV below.

5: Y(NEQMAX) – ***real*** array. *Input/Output*

On entry: the values of the dependent variables (solution). On the first call the first NEQ elements of y must contain the vector of initial values.

On exit: the computed solution vector, evaluated at t (usually t = TOUT).

6: YDOT(NEQMAX) – ***real*** array. *Input/Output*

On entry: if LDERIV(1) = .TRUE., YDOT must contain approximations to the time derivatives y' of the vector y. If LDERIV(1) = .FALSE., then YDOT need not be set on entry.

On exit: the time derivatives y' of the vector y at the last integration point.

7: RWORK(50+4*NEQMAX) – ***real*** array. *Workspace*

8: RTOL(*) – ***real*** array. *Input*

Note: the dimension of RTOL must be at least 1 or NEQ (see ITOL).

On entry: the relative local error tolerance.

Constraint: RTOL(i) ≥ 0.0 for all relevant i (see ITOL).

9: ATOL(*) – ***real*** array. *Input*

Note: the dimension of ATOL must be at least 1 or NEQ (see ITOL).

On entry: the absolute local error tolerance.

Constraint: ATOL(i) $\geq$ 0.0 for all relevant i (see ITOL).

10: ITOL – INTEGER. *Input*

On entry: a value to indicate the form of the local error test. ITOL indicates to D02NGF whether to interpret either or both of RTOL or ATOL as a vector or a scalar. The error test to be satisfied is $\|e_i/w_i\| < 1.0$, where w_i is defined as follows:

ITOL	RTOL	ATOL	w_i
1	scalar	scalar	RTOL(1)$\times|y_i|$ + ATOL(1)
2	scalar	vector	RTOL(1)$\times|y_i|$ + ATOL(i)
3	vector	scalar	RTOL(i)$\times|y_i|$ + ATOL(1)
4	vector	vector	RTOL(i)$\times|y_i|$ + ATOL(i)

e_i is an estimate of the local error in y_i computed internally, and the choice of norm to be used is defined by a previous call to an integrator setup routine.

Constraint: 1 $\leq$ ITOL $\leq$ 4.

11: INFORM(23) – INTEGER array. *Workspace*

12: RESID – SUBROUTINE, supplied by the user. *External Procedure*

RESID must evaluate the residual

$$r = g(t,y) - A(t,y)y'$$

in one case and

$$r = -A(t,y)y'$$

in another.

Its specification is:

```
SUBROUTINE RESID(NEQ, T, Y, YDOT, R, IRES)
INTEGER    NEQ, IRES
real       T, Y(NEQ), YDOT(NEQ), R(NEQ)
```

1: NEQ – INTEGER. *Input*

On entry: the number of equations being solved.

2: T – ***real***. *Input*

On entry: the current value of the independent variable t.

3: Y(NEQ) – ***real*** array. *Input*

On entry: the value of y_i, for i = 1,2,...,NEQ.

4: YDOT(NEQ) – ***real*** array. *Input*

On entry: the value of y_i' at t, for i = 1,2,...,NEQ.

5: R(NEQ) – ***real*** array. *Output*

On exit: R(i) must contain the ith component of r, for i = 1,2,...,NEQ where

$$r = g(t,y) - A(t,y)y' \qquad (1)$$

or

$$r = -A(t,y)y' \qquad (2)$$

and where the definition of r is determined by the input value of IRES.

6: IRES – INTEGER. *Input/Output*

On entry: the form of the residual that must be returned in array R. If IRES = –1, then the residual defined in equation (2) above must be returned. If IRES = 1, then the residual defined in equation (1) above must be returned.

On exit: IRES should be unchanged unless one of the following actions is required of the integrator, in which case IRES should be set accordingly.

IRES = 2

indicates to the integrator that control should be passed back immediately to the calling (sub)program with the error indicator set to IFAIL = 11.

IRES = 3

indicates to the integrator that an error condition has occurred in the solution vector, its time derivative or in the value of t. The integrator will use a smaller time step to try to avoid this condition. If this is not possible, the integrator returns to the calling (sub)program with the error indicator set to IFAIL = 7.

IRES = 4

indicates to the integrator to stop its current operation and to enter the MONITR routine immediately with parameter IMON = –2.

RESID must be declared as EXTERNAL in the (sub)program from which D02NGF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

13: YSAVE(NEQMAX, NY2DIM) – ***real*** array. *Workspace*

14: NY2DIM – INTEGER. *Input*

On entry: the second dimension of the array YSAVE as declared in the (sub)program from which D02NGF is called. An appropriate value for NY2DIM is described in the specifications of the integrator setup routines D02MVF, D02NVF and D02NWF. This value must be the same as that supplied to the integrator setup routine.

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

JAC must evaluate the Jacobian of the system. If this option is not required, the actual argument for JAC must be the dummy routine D02NGZ. (D02NGZ is included in the NAG Fortran Library and so need not be supplied by the user. Its name may be implementation dependent: see the Users' Note for your implementation for details.) The user indicates to the integrator whether this option is to be used by setting the parameter JCEVAL appropriately in a call to the linear algebra setup routine D02NSF.

First we must define the system of nonlinear equations which is solved internally by the integrator. The time derivative, y', generated internally has the form

$$y' = (y-z)/(hd)$$

where h is the current stepsize and d is a parameter that depends on the integration method in use. The vector y is the current solution and the vector z depends on information from previous time steps. This means that $\frac{d}{dy'}(\ ) = \frac{1}{(hd)}\frac{d}{dy}(\ )$. The system of nonlinear equations that is solved has the form

$$A(t,y)y' - g(t,y) = 0$$

but is solved in the form

$$r(t,y) = 0,$$

where r is the function defined by

$$r(t,y) = (hd)(A(t,y)(y-z)/(hd)-g(t,y)).$$

It is the Jacobian matrix $\frac{\partial r}{\partial y}$ that the user must supply in the routine JAC as follows:

$$\frac{\partial r_i}{\partial y_j} = a_{ij}(t,y) + (hd)\frac{\partial}{\partial y_j}\left(\sum_{k=1}^{\mathrm{NEQ}} a_{ik}(t,y)y'_k - g_i(t,y)\right)$$

Its specification is:

```
SUBROUTINE JAC(NEQ, T, Y, YDOT, H, D, P)
INTEGER    NEQ
real       T, Y(NEQ), YDOT(NEQ), H, D, P(NEQ, NEQ)
```

1: NEQ – INTEGER. *Input*

On entry: the number of equations being solved.

2: T – ***real***. *Input*

On entry: the current value of the independent variable t.

3: Y(NEQ) – ***real*** array. *Input*

On entry: the current solution component y_i, for i = 1,2,...,NEQ.

4: YDOT(NEQ) – ***real*** array. *Input*

On entry: the derivative of the solution at the current point t.

5: H – ***real***. *Input*

On entry: the current stepsize.

6: D – ***real***. *Input*

On entry: the parameter d which depends on the integration method.

7: P(NEQ,NEQ) – ***real*** array. *Output*

On exit: P(i,j) must contain $\frac{\partial r_i}{\partial y_j}$, for i,j = 1,2,...,NEQ.

Only non-zero elements of this array need be set, since it is preset to zero before the call to JAC.

JAC must be declared as EXTERNAL in the (sub)program from which D02NGF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

16: WKJAC(NWKJAC) – ***real*** array. *Workspace*

17: NWKJAC – INTEGER. *Input*

On entry: the dimension of the array WKJAC as declared in the (sub)program from which D02NGF is called. This value must be the same as that supplied to the linear algebra setup routine D02NSF.

Constraint: NWKJAC ≥ NEQMAX×(NEQMAX+1).

18: MONITR – SUBROUTINE, supplied by the user. *External Procedure*

MONITR performs tasks requested by the user. If this option is not required the actual argument for MONITR must be the dummy routine D02NBY. (D02NBY is includd in the NAG Fortran Library and so need not be supplied by the user. Its name may be implementation dependent: see the Users' Note for your implementation for details.)

Its specification is:

```
 SUBROUTINE MONITR(NEQ, NEQMAX, T, HLAST, HNEXT, Y, YDOT, YSAVE, R,
1           ACOR, IMON, INLN, HMIN, HMAX, NQU)
 INTEGER    NEQ, NEQMAX, IMON, INLN, NQU
 real       T, HLAST, HNEXT, Y(NEQMAX), YDOT(NEQMAX), YSAVE(NEQMAX,*),
1           R(NEQMAX), ACOR(NEQMAX,2), HMIN, HMAX
```

1: NEQ – INTEGER. *Input*

On entry: the number of equations being solved.

2: NEQMAX – INTEGER. *Input*

On entry: an upper bound on the number of equations to be solved.

3: T – ***real.*** *Input*

On entry: the current value of the independent variable.

4: HLAST – ***real.*** *Input*

On entry: the last stepsize successfully used by the integrator.

5: HNEXT – ***real.*** *Input/Output*

On entry: the stepsize that the integrator proposes to take on the next step.

On exit: the next stepsize to be used. If this is different from the input value, then IMON must be set to 4.

6: Y(NEQMAX) – ***real*** array. *Input/Output*

On entry: the values of the dependent variables, y, evaluated at t.

On exit: these values must not be changed unless IMON is set to 2.

7: YDOT(NEQMAX) – ***real*** array. *Input*

On entry: the time derivatives y' of the vector y.

8: YSAVE(NEQMAX,*) – ***real*** array. *Input*

On entry: workspace to enable the user to carry out interpolation using either of the routines D02XJF or D02XKF.

9: R(NEQMAX) – ***real*** array. *Input*

On entry: if IMON = 0 and INLN = 3, then the first NEQ elements contain the residual vector $A(t,y)y' - g(t,y)$.

10: ACOR(NEQMAX,2) – ***real*** array. *Input*

On entry: with IMON = 1, ACOR(i,1) contains the weight used for the ith equation when the norm is evaluated and ACOR(i,2) contains the estimated local error for the ith equation. The scaled local error at the end of a timestep may be obtained by calling the ***real*** function D02ZAF as follows

```
      IFAIL = 1
      ERRLOC = D02ZAF(NEQ, ACOR(1,2), ACOR(1,1), IFAIL)
C     CHECK IFAIL BEFORE PROCEEDING
```

11: IMON – INTEGER. *Input/Output*

On entry: a flag indicating under what circumstances MONITR was called:

IMON = –2

entry from the integrator after IRES = 4 (set in RESID) caused an early termination (this facility could be used to locate discontinuities).

IMON = –1

the current step failed repeatedly.

IMON = 0

entry after a call to the internal nonlinear equation solver (see below).

IMON = 1

the current step was successful.

On exit: IMON may be re-set to determine subsequent action in D02NGF:

IMON = –2

integration is to be halted. A return will be made from the integrator to the calling (sub)program with IFAIL = 12.

IMON = –1

allow the integrator to continue with its own internal strategy. The integrator will try up to 3 restarts unless IMON is set ≠ –1 on exit.

IMON = 0

return to the internal nonlinear equation solver, where the action taken is determined by the value of INLN (see below).

IMON = 1

normal exit to the integrator to continue integration.

IMON = 2

restart the integration at the current time point. The integrator will restart from order 1 when this option is used. The MONITR provided solution Y will be used for the initial conditions.

IMON = 3

try to continue with the same stepsize and order as was to be used before the call to MONITR. HMIN and HMAX may be altered if desired.

IMON = 4

continue the integration but using a new value HNEXT and possibly new values of HMIN and HMAX.

12: INLN – INTEGER. *Output*

On exit: the action to be taken by the internal nonlinear equation solver when MONITR is exited with IMON = 0. By setting INLN = 3 and returning to the integrator, the residual vector is evaluated and placed in the array R, and then MONITR is called again. At present this is the only option available: INLN must not be set to any other value.

13: HMIN – ***real.*** *Input/Output*

On entry: the minimum stepsize to be taken on the next step.

On exit: the minimum stepsize to be used. If this is different from the input value, then IMON must be set to 3 or 4.

14: HMAX – ***real.*** *Input/Output*

On entry: the maximum stepsize to be taken on the next step.

On exit: the maximum stepsize to be used. If this is different from the input value, then IMON must be set to 3 or 4. If HMAX is set to zero, no limit is assumed.

15: NQU – INTEGER. *Input*

On entry: the order of the integrator used on the last step. This is supplied to enable the user to carry out interpolation using either of the routines D02XJF or D02XKF.

MONITR must be declared as EXTERNAL in the (sub)program from which D02NGF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

19: LDERIV(2) – LOGICAL array. *Input/Output*

On entry: LDERIV(1) must be set to .TRUE., if the user has supplied both an initial y and an initial y'. LDERIV(1) must be set to .FALSE., if only the initial y has been supplied.

LDERIV(2) must be set to .TRUE., if the integrator is to use a modified Newton method to evaluate the initial y and y'. Note that y and y', if supplied, are used as initial estimates. This method involves taking a small step at the start of the integration, and if ITASK = 6 on entry, T and TOUT will be set to the result of taking this small step. LDERIV(2) must be set to .FALSE., if the integrator is to use functional iteration to evaluate the initial y and y', and if this fails a modified Newton method will then be attempted. LDERIV(2) = .TRUE. is recommended if there are implicit equations or the initial y and y' are zero.

On exit: LDERIV(1) is normally unchanged. However if ITASK = 6 and internal initialisation was successful then LDERIV(1) = .TRUE..

LDERIV(2) = .TRUE., if implicit equations were detected. Otherwise LDERIV(2) = .FALSE..

20: ITASK – INTEGER. *Input*

On entry: the task to be performed by the integrator. The permitted values for ITASK and their meanings are detailed below:

ITASK = 1

normal computation of output values of $y(t)$ at t = TOUT (by overshooting and interpolating).

ITASK = 2

take one step only and return.

ITASK = 3

stop at the first internal integration point at or beyond t = TOUT and return.

ITASK = 4

normal computation of output values of $y(t)$ at t = TOUT but without overshooting t = TCRIT. TCRIT must be specified as an option in one of the integrator setup routines prior to the first call to the integrator, or specified in the optional input routine prior to a continuation call. TCRIT may be equal to or beyond TOUT, but not before it, in the direction of integration.

ITASK = 5

take one step only and return, without passing TCRIT. TCRIT must be specified as under ITASK = 4.

ITASK = 6

the integrator will solve for the initial values of y and y' only and then return to the calling (sub)program without doing the integration. This option can be used to check the initial values of y and y'. Functional iteration or a 'small' backward Euler method used in conjunction with a damped Newton iteration is used to calculate these values (see LDERIV above). Note that if a backward Euler step is used then the value of t will have been advanced a short distance from the initial point.

Note: if D02NGF is recalled with a different value of ITASK (and TOUT altered), then the initialisation procedure is repeated, possibly leading to different initial conditions.

Constraint: 1 ≤ ITASK ≤ 6.

21: ITRACE – INTEGER. *Input*

On entry: the level of output that is printed by the integrator. ITRACE may take the value −1, 0, 1, 2 or 3. If ITRACE < −1, then −1 is assumed and similarly if ITRACE > 3, then 3 is assumed. If ITRACE = −1, no output is generated. If ITRACE = 0, only warning messages are printed on the current error message unit (see X04AAF). If ITRACE > 0 then warning messages are printed as above, and on the current advisory message unit (see X04ABF) output is generated which details Jacobian entries, the nonlinear iteration and the time integration. The advisory messages are given in greater detail the larger the value of ITRACE.

22: IFAIL – INTEGER. *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to −1 before entry. **It is then essential to test**

the value of IFAIL on exit. To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL = 1

An illegal input was detected on entry, or after an internal call to MONITR. If ITRACE > −1, then the form of the error will be detailed on the current error message unit (see X04AAF).

IFAIL = 2

The maximum number of steps specified has been taken (see the description of optional inputs in the integrator setup routines and the optional input continuation routine, D02NZF).

IFAIL = 3

With the given values of RTOL and ATOL no further progress can be made across the integration range from the current point T. The components Y(1),Y(2),...,Y(NEQ) contain the computed values of the solution at the current point T.

IFAIL = 4

There were repeated error test failures on an attempted step, before completing the requested task, but the integration was successful as far as T. The problem may have a singularity, or the local error requirements may be inappropriate.

IFAIL = 5

There were repeated convergence test failures on an attempted step, before completing the requested task, but the integration was successful as far as T. This may be caused by an inaccurate Jacobian matrix or one which is incorrectly computed.

IFAIL = 6

Some error weight w_i became zero during the integration (see description of ITOL). Pure relative error control (ATOL(i) = 0.0) was requested on a variable (the ith) which has now vanished. The integration was successful as far as T.

IFAIL = 7

The user-supplied subroutine RESID set its error flag (IRES = 3) continually despite repeated attempts by the integrator to avoid this.

IFAIL = 8

LDERIV(1) = .FALSE. on entry but the internal initialisation routine was unable to initialise y' (more detailed information may be directed to the current error message unit, see X04AAF).

IFAIL = 9

A singular Jacobian $\frac{\partial r}{\partial y}$ has been encountered. The user should check his problem formulation and Jacobian calculation.

IFAIL = 10

An error occurred during Jacobian formulation or backsubstitution (a more detailed error description may be directed to the current error message unit see X04AAF).

IFAIL = 11

The user-supplied subroutine RESID signalled the integrator to halt the integration and return (IRES = 2). Integration was successful as far as T.

IFAIL = 12

The user-supplied subroutine MONITR set IMON = –2 and so forced a return but the integration was successful as far as T.

IFAIL = 13

The requested task has been completed, but it is estimated that a small change in RTOL and ATOL is unlikely to produce any change in the computed solution. (Only applies when the user is not operating in one step mode, that is when ITASK ≠ 2 or 5).

IFAIL = 14

The values of RTOL and ATOL are so small that the routine is unable to start the integration.

IFAIL = 15

The linear algebra setup routine D02NSF was not called before the call to D02NGF.

7. Accuracy

The accuracy of the numerical solution may be controlled by a careful choice of the parameters RTOL and ATOL, and to a much lesser extent by the choice of norm. Users are advised to use scalar error control unless the components of the solution are expected to be poorly scaled. For the type of decaying solution typical of many stiff problems, relative error control with a small absolute error.threshold will be most appropriate (that is the user is advised to choose ITOL = 1 with ATOL(1) small but positive).

8. Further Comments

The cost of computing a solution depends critically on the size of the differential system and to a lesser extent on the degree of stiffness of the problem. For D02NGF the cost is proportional to NEQ^3, though for problems which are only mildly nonlinear the cost may be dominated by factors proportional to NEQ^2 except for very large problems.

In general the user is advised ,to choose the BDF option (setup routine D02NVF) but if efficiency is of great importance and especially if it is suspected that $\frac{\partial}{\partial y}(A^{-1}g)$ has complex eigenvalues near the imaginary axis for some part of the integration, the user should try the BLEND option (setup routine D02NWF).

9. Example

We solve the well-known stiff Robertson problem written in implicit form

$$\begin{aligned} r_1 &= -0.04a + 1.0\mathrm{E}4bc && - a' \\ r_2 &= \;\;\,0.04a - 1.0\mathrm{E}4bc - 3.0\mathrm{E}7b^2 && - b' \\ r_3 &= \qquad\qquad\qquad\quad\;\; 3.0\mathrm{E}7b^2 && - c' \end{aligned}$$

with initial conditions $a = 1.0$ and $b = c = 0.0$ over the range [0,0.1] with vector error control (ITOL = 4), the BDF method (setup routine D02NVF) and functional iteration. The Jacobian is calculated numerically if the functional iteration encounters difficulty and the integration is in one-step mode (ITASK = 2), with C^0 interpolation to calculate the solution at intervals of 0.02 using D02XJF externally. D02NBY is used for MONITR.

9.1. Program Text

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

```
*     D02NGF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, NEQMAX, NRW, NINF, NWKJAC, MAXORD, NY2DIM,
     +                 MAXSTP, MXHNIL
      PARAMETER        (NEQ=3,NEQMAX=NEQ,NRW=50+4*NEQMAX,NINF=23,
     +                 NWKJAC=NEQMAX*(NEQMAX+1),MAXORD=5,
     +                 NY2DIM=MAXORD+1,MAXSTP=200,MXHNIL=5)
      real             H0, HMAX, HMIN, TCRIT
      PARAMETER        (H0=0.0e0,HMAX=10.0e0,HMIN=1.0e-10,TCRIT=0.0e0)
      LOGICAL          PETZLD
      PARAMETER        (PETZLD=.FALSE.)
*     .. Local Scalars ..
      real             H, HU, T, TCUR, TOLSF, TOUT, XOUT
      INTEGER          I, IFAIL, IMXER, IOUT, ITASK, ITOL, ITRACE,
     +                 NITER, NJE, NQ, NQU, NRE, NST
*     .. Local Arrays ..
      real             ATOL(NEQMAX), CONST(6), RTOL(NEQMAX), RWORK(NRW),
     +                 SOL(NEQMAX), WKJAC(NWKJAC), Y(NEQMAX),
     +                 YDOT(NEQMAX), YSAVE(NEQMAX,NY2DIM)
      INTEGER          INFORM(NINF)
      LOGICAL          ALGEQU(NEQMAX), LDERIV(2)
*     .. External Subroutines ..
      EXTERNAL         D02NBY, D02NGF, D02NGZ, D02NSF, D02NVF, D02NYF,
     +                 D02XJF, RESID, X04ABF
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02NGF Example Program Results'
      CALL X04ABF(1,NOUT)
*
*     Integrate to TOUT by overshooting TOUT in one step mode (ITASK=2)
*     using B.D.F formulae with a functional iteration method.
*     Default values for the array CONST are used. Employ vector
*     tolerances and the Jacobian is evaluated internally, if necessary.
*     MONITR subroutine replaced by NAG dummy routine D02NBY.
*     Interpolation outside D02NGF using D02XJF.
*
      T = 0.0e0
      TOUT = 0.1e0
      ITASK = 2
      Y(1) = 1.0e0
      Y(2) = 0.0e0
      Y(3) = 0.0e0
      LDERIV(1) = .FALSE.
      LDERIV(2) = .FALSE.
      ITOL = 4
      RTOL(1) = 1.0e-4
      RTOL(2) = 1.0e-3
      RTOL(3) = 1.0e-4
      ATOL(1) = 1.0e-7
      ATOL(2) = 1.0e-8
      ATOL(3) = 1.0e-7
      DO 20 I = 1, 6
         CONST(I) = 0.0e0
   20 CONTINUE
      IFAIL = 0
*
```

```
      CALL D02NVF(NEQMAX,NY2DIM,MAXORD,'Functional-iteration',PETZLD,
     +             CONST,TCRIT,HMIN,HMAX,H0,MAXSTP,MXHNIL,'Average-L2',
     +             RWORK,IFAIL)
*     Linear algebra setup required (in case functional iteration
*     encounters any difficulty).
      CALL D02NSF(NEQ,NEQMAX,'Numerical',NWKJAC,RWORK,IFAIL)
*
      XOUT = 0.02e0
      IOUT = 1
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) '   X          Y(1)           Y(2)           Y(3)'
      WRITE (NOUT,99999) T, (Y(I),I=1,NEQ)
*
*     Soft fail and error messages only
      ITRACE = 0
   40 IFAIL = 1
*
      CALL D02NGF(NEQ,NEQMAX,T,TOUT,Y,YDOT,RWORK,RTOL,ATOL,ITOL,INFORM,
     +            RESID,YSAVE,NY2DIM,D02NGZ,WKJAC,NWKJAC,D02NBY,LDERIV,
     +            ITASK,ITRACE,IFAIL)
*
      IF (IFAIL.EQ.0) THEN
*
         CALL D02NYF(NEQ,NEQMAX,HU,H,TCUR,TOLSF,RWORK,NST,NRE,NJE,NQU,
     +               NQ,NITER,IMXER,ALGEQU,INFORM,IFAIL)
*
   60    CONTINUE
         IF (TCUR-HU.LT.XOUT .AND. XOUT.LE.TCUR) THEN
            IFAIL = 0
*           C0 interpolation
            CALL D02XJF(XOUT,SOL,NEQ,YSAVE,NEQMAX,NY2DIM,NEQ,TCUR,NQU,
     +                  HU,H,IFAIL)
*
            WRITE (NOUT,99999) XOUT, (SOL(I),I=1,NEQ)
            IOUT = IOUT + 1
            XOUT = real(IOUT)*0.02e0
            IF (IOUT.LT.6) THEN
               GO TO 60
            ELSE
               WRITE (NOUT,*)
               WRITE (NOUT,99997) ' HUSED = ', HU, '  HNEXT = ', H,
     +           '  TCUR = ', TCUR
               WRITE (NOUT,99996) ' NST = ', NST, '    NRE = ', NRE,
     +           '    NJE = ', NJE
               WRITE (NOUT,99996) ' NQU = ', NQU, '    NQ  = ', NQ,
     +           '  NITER = ', NITER
               WRITE (NOUT,99995) ' Max err comp = ', IMXER
            END IF
         ELSE
            GO TO 40
         END IF
      ELSE
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'Exit D02NGF with IFAIL = ', IFAIL,
     +     '  and T = ', T
      END IF
      STOP
*
99999 FORMAT (1X,F8.3,3(F13.5,2X))
99998 FORMAT (1X,A,I2,A,e12.5)
99997 FORMAT (1X,A,e12.5,A,e12.5,A,e12.5)
99996 FORMAT (1X,A,I6,A,I6,A,I6)
99995 FORMAT (1X,A,I4)
      END
*
```

```
      SUBROUTINE RESID(NEQ,T,Y,YDOT,R,IRES)
*     .. Scalar Arguments ..
      real             T
      INTEGER          IRES, NEQ
*     .. Array Arguments ..
      real             R(NEQ), Y(NEQ), YDOT(NEQ)
*     .. Executable Statements ..
      R(1) = -YDOT(1)
      R(2) = -YDOT(2)
      R(3) = -YDOT(3)
*
      IF (IRES.EQ.1) THEN
         R(1) = -0.04e0*Y(1) + 1.0e4*Y(2)*Y(3) + R(1)
         R(2) = 0.04e0*Y(1) - 1.0e4*Y(2)*Y(3) - 3.0e7*Y(2)*Y(2) + R(2)
         R(3) = 3.0e7*Y(2)*Y(2) + R(3)
      END IF
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D02NGF Example Program Results

    X          Y(1)           Y(2)           Y(3)
   0.000      1.00000        0.00000        0.00000
   0.020      0.99920        0.00004        0.00076
   0.040      0.99841        0.00004        0.00155
   0.060      0.99763        0.00004        0.00234
   0.080      0.99685        0.00004        0.00311
   0.100      0.99608        0.00004        0.00389

 HUSED =  0.35237E-03  HNEXT =  0.35237E-03  TCUR =   0.10016E+00
 NST =    229    NRE =    691   NJE =       0
 NQU =      2    NQ  =      2  NITER =       0
 Max err comp =    3
```

D02NHF – NAG Fortran Library Routine Document

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

1 Purpose

D02NHF is a forward communication routine for integrating stiff systems of implicit ordinary differential equations coupled with algebraic equations when the Jacobian is a banded matrix.

2 Specification

```
 SUBROUTINE D02NHF(NEQ, NEQMAX, T, TOUT, Y, YDOT, RWORK, RTOL,
1                  ATOL, ITOL, INFORM, RESID, YSAVE, NY2DIM, JAC,
2                  WKJAC, NWKJAC, JACPVT, NJCPVT, MONITR, LDERIV,
3                  ITASK, ITRACE, IFAIL)
 INTEGER           NEQ, NEQMAX, ITOL, INFORM(23), NY2DIM, NWKJAC,
1                  JACPVT(NJCPVT), NJCPVT, ITASK, ITRACE, IFAIL
 real              T, TOUT, Y(NEQMAX), YDOT(NEQMAX),
1                  RWORK(50+4*NEQMAX), RTOL(*), ATOL(*),
2                  YSAVE(NEQMAX,NY2DIM), WKJAC(NWKJAC)
 LOGICAL           LDERIV(2)
 EXTERNAL          RESID, JAC, MONITR
```

3 Description

D02NHF is a general purpose routine for integrating the initial value problem for a stiff system of implicit ordinary differential equations coupled with algebraic equations, written in the form,

$$A(t,y)y' = g(t,y).$$

It is designed specifically for the case where the resulting Jacobian is a banded matrix (see description of the argument JAC in Section 5).

Both interval and step oriented modes of operation are available and also modes designed to permit intermediate output within an interval oriented mode.

An outline of a typical calling program for D02NHF is given below. It calls the banded matrix linear algebra setup routine D02NTF, and the Backward Differentiation Formula (BDF) integrator setup routine D02NVF, and its diagnostic counterpart D02NYF.

```
C
C     declarations
C
      EXTERNAL RESID, JAC, MONITR
           .
           .
           .
      IFAIL = 0
      CALL D02NVF(...,IFAIL)
      CALL D02NTF(NEQ, NEQMAX, JCEVAL, ML, MU, NWKJAC, NJCPVT,
     + RWORK, IFAIL)
      IFAIL = -1
      CALL D02NHF(NEQ, NEQMAX, T, TOUT, Y, YDOT, RWORK, RTOL,
     + ATOL, ITOL, INFORM, RESID, YSAVE, NY2DIM, JAC, WKJAC,
     + NWKJAC, JACPVT, NJCPVT, MONITR, LDERIV, ITASK, ITRACE,
     + IFAIL)
      IF (IFAIL.EQ.1 .OR. IFAIL.GE.14) STOP
      IFAIL = 0
      CALL D02NYF(...)
              .
              .
              .
      STOP
      END
```

The linear algebra setup routine D02NTF and one of the integrator setup routines, D02MVF, D02NVF or D02NWF, must be called prior to the call of D02NHF. The integrator diagnostic routine D02NYF may be called after the call to D02NHF. There is also a routine, D02NZF, designed to permit the user to change step size on a continuation call to D02NHF without restarting the integration process.

4 References

None.

5 Parameters

1: NEQ — INTEGER *Input*

On entry: the current trial value of the eigenvalue parameter λ.

2: NEQMAX — INTEGER *Input*

On entry: a bound on the maximum number of equations to be solved during the integration.

Constraint: NEQMAX $\geq$ NEQ.

3: T — ***real*** *Input/Output*

On entry: the value of the independent variable, t. The input value of T is used only on the first call as the initial point of the integration.

On exit: the value at which the computed solution y is returned (usually at TOUT).

4: TOUT — ***real*** *Input/Output*

On entry: the next value of t at which a computed solution is desired. For the initial t, the input value of TOUT is used to determine the direction of integration. Integration is permitted in either direction (see also ITASK).

Constraint: TOUT $\neq$ T.

On exit: normally unchanged. However when ITASK = 6, then TOUT contains the value of T at which initial values have been computed without performing any integration. See descriptions of ITASK and LDERIV below.

5: Y(NEQMAX) — ***real*** array *Input/Output*

On entry: the values of the dependent variables (solution). On the first call the first NEQ elements of y must contain the vector of initial values.

On exit: the computed solution vector, evaluated at t (usually t = TOUT).

6: YDOT(NEQMAX) — ***real*** array *Input/Output*

On entry: if LDERIV(1) = .TRUE., YDOT must contain approximations to the time derivatives y' of the vector y. If LDERIV(1) = .FALSE., then YDOT need not be set on entry.

On exit: the time derivatives y' of the vector y at the last integration point.

7: RWORK(50+4*NEQMAX) — ***real*** array *Workspace*

8: RTOL(∗) — ***real*** array *Input*

Note: the dimension of the array RTOL must be at least 1 or NEQ (see ITOL).

On entry: the relative local error tolerance.

Constraint: RTOL(i) $\geq$ 0.0 for all relevant i (see ITOL).

9: ATOL(∗) — ***real*** array *Input*

Note: the dimension of the array ATOL must be at least 1 or NEQ (see ITOL).

On entry: the absolute local error tolerance.

Constraint: ATOL(i) $\geq$ 0.0 for all relevant i (see ITOL).

10: ITOL — INTEGER *Input*

On entry: a value to indicate the form of the local error test. ITOL indicates to D02NHF whether to interpret either or both of RTOL or ATOL as a vector or a scalar. The error test to be satisfied is $\|e_i/w_i\| < 1.0$, where w_i is defined as follows:

ITOL	RTOL	ATOL	w_i
1	scalar	scalar	RTOL(1) × $\lvert y_i\rvert$ + ATOL(1)
2	scalar	vector	RTOL(1) × $\lvert y_i\rvert$ + ATOL(i)
3	vector	scalar	RTOL(i) × $\lvert y_i\rvert$ + ATOL(1)
4	vector	vector	RTOL(i) × $\lvert y_i\rvert$ + ATOL(i)

e_i is an estimate of the local error in y_i, computed internally, and the choice of norm to be used is defined by a previous call to an integrator setup routine.

Constraint: 1 $\leq$ ITOL $\leq$ 4.

11: INFORM(23) — INTEGER array *Workspace*

12: RESID — SUBROUTINE, supplied by the user. *External Procedure*

RESID must evaluate the residual

$$r = g(t,y) - A(t,y)y'$$

in one case and

$$r = -A(t,y)y'$$

in another.

Its specification is:

```
SUBROUTINE RESID(NEQ, T, Y, YDOT, R, IRES)
INTEGER        NEQ, IRES
real           T, Y(NEQ), YDOT(NEQ), R(NEQ)
```

1: NEQ — INTEGER *Input*

On entry: the number of equations being solved.

2: T — ***real*** *Input*

On entry: the current value of the independent variable, t.

3: Y(NEQ) — ***real*** array *Input*

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

4: YDOT(NEQ) — ***real*** array *Input*

On entry: the value of y_i' at t, for $i = 1,2,\ldots,$NEQ.

5: R(NEQ) — ***real*** array *Output*

On exit: R(i) must contain the ith component of r_i, for $i = 1, 2, \ldots$,NEQ, where,

$$r = g(t,y) - A(t,y)y' \qquad (1)$$

or

$$r = -A(t,y)y' \qquad (2)$$

and where the definition of r is determined by the input value of IRES.

6: IRES — INTEGER *Input/Output*

On entry: the form of the residual that must be returned in the array R. If IRES = −1, then the residual defined by equation (2) above must be returned; if IRES = 1, then the residual defined by equation (1) above must be returned.

On exit: IRES should be unchanged unless one of the following actions is required of the integrator, in which case IRES should be set accordingly.

IRES = 2

indicates to the integrator that control should be passed back immediately to the calling (sub)program with the error indicator set to IFAIL = 11.

IRES = 3

indicates to the integrator that an error condition has occurred in the solution vector, its time derivative or in the value of t. The integrator will use a smaller time step to try to avoid this condition. If this is not possible the integrator returns to the calling (sub)program with the error indicator set to IFAIL = 7.

IRES = 4

indicates to the integrator to stop its current operation and to enter the MONITR routine immediately with parameter IMON = −2.

RESID must be declared as EXTERNAL in the (sub)program from which D02NHF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

13: YSAVE(NEQMAX,NY2DIM) — ***real*** array *Workspace*

14: NY2DIM — INTEGER *Input*

On entry: the second dimension of the array YSAVE as declared in the (sub)program from which D02NHF is called.An appropriate value for NY2DIM is described in the specifications of the integrator setup routines D02MVF, D02NVF and D02NWF. This value must be the same as that supplied to the integrator setup routine.

15: JAC — SUBROUTINE, supplied by the user. *External Procedure*

JAC must evaluate the Jacobian of the system. If this option is not required, the actual argument for JAC must be the dummy routine D02NHZ. (D02NHZ is included in the NAG Fortran Library and so need not be supplied by the user. Its name may be implementation dependent: see the Users' Note for your implementation for details.) The user indicates whether this option is to be used by setting the parameter JCEVAL appropriately in a call to the linear algebra setup routine D02NTF.

First we must define the system of nonlinear equations which is solved internally by the integrator. The time derivative, y', generated internally, has the form

$$y' = (y - z)/(hd)$$

where h is the current step size and d is a parameter that depends on the integration method in use. The vector y is the current solution and the vector z depends on information from the previous time steps. This means that $\frac{d}{dy'}() = \frac{1}{(hd)}\frac{d}{dy}()$. The system of nonlinear equations that is solved has the form

$$A(t,y)y' - g(t,y) = 0$$

but is solved in the form

$$r(t,y) = 0$$

where r is the function defined by

$$r(t,y) = (hd)(A(t,y)(y-z)/(hd) - g(t,y)).$$

It is the Jacobian matrix $\frac{\partial r}{\partial y}$ that the user must supply in the routine JAC as follows:

$$\frac{\partial r_i}{\partial y_j} = a_{ij}(t,y) + (hd)\frac{\partial}{\partial y_j}\left(\sum_{k=1}^{\mathrm{NEQ}} a_{ik}(t,y)y'_k - g_i(t,y)\right)$$

Its specification is:

```
SUBROUTINE JAC(NEQ, T, Y, YDOT, H, D, ML, MU, P)
INTEGER         NEQ, ML, MU
real            T, Y(NEQ), YDOT(NEQ), H, D, P(ML+MU+1,NEQ)
```

1: NEQ — INTEGER *Input*

On entry: the number of equations being solved.

2: T — ***real*** *Input*

On entry: the current value of the independent variable, t.

3: Y(NEQ) — ***real*** array *Input*

On entry: the current solution component y_i, for $i = 1,2,\ldots,$NEQ.

4: YDOT(NEQ) — ***real*** array *Input*

On entry: the derivative of the solution at the current point t.

5: H — ***real*** *Input*

On entry: the current step size.

6: D — ***real*** *Input*

On entry: the parameter d which depends on the integration method.

7: ML — INTEGER *Input*

8: MU — INTEGER *Input*

On entry: the number of sub-diagonals and super-diagonals respectively in the band.

9: P(ML+MU+1,NEQ) — ***real*** array *Output*

On exit: elements of the Jacobian matrix $\frac{\partial r}{\partial y}$ stored as specified by the following pseudo code

```
      DO 20 I = 1, NEQ
         J1 = MAX(I-ML,1)
         J2 = MIN(I+MU,NEQ)
         DO 10 J = J1, J2
            K = MIN(ML+1-I,0)+J
            P(K,I) = $\partial R/\partial Y(I,J)$
   10    CONTINUE
   20 CONTINUE
```

See also the routine document for F07BDF.

Only non-zero elements of this array need be set, since it is preset to zero before the call to JAC.

JAC must be declared as EXTERNAL in the (sub)program from which D02NHF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

16: WKJAC(NWKJAC) — ***real*** array *Workspace*

17: NWKJAC — INTEGER *Input*

On entry: the dimension of the array WKJAC as declared in the (sub)program from which D02NHF is called. This value must be the same as that supplied to the linear algebra setup routine D02NTF.

Constraint: NWKJAC $\geq (2m_L + m_U + 1) \times$ NEQMAX, where m_L and m_U are the number of sub-diagonals and super-diagonals respectively in the band defined by a call to D02NTF.

18: JACPVT(NJCPVT) — INTEGER array *Workspace*

19: NJCPVT — INTEGER *Input*

On entry: the size of the array JACPVT. This value must be the same as that supplied to the linear algebra setup routine D02NTF.

Constraint: NJCPVT $\geq$ NEQMAX.

20: MONITR — SUBROUTINE, supplied by the user. *External Procedure*

MONITR performs tasks requested by the user. If this option is not required, then the actual argument for MONITR must be the dummy routine D02NBY. (D02NBY is included in the NAG Fortran Library and so need not be supplied by the user. Its name may be implementation dependent: see the Users' Note for your implementation for details.)

Its specification is:

```
      SUBROUTINE MONITR(NEQ, NEQMAX, T, HLAST, HNEXT, Y, YDOT, YSAVE, R,
     1                  ACOR, IMON, INLN, HMIN, HMAX, NQU)
      INTEGER           NEQ, NEQMAX, IMON, INLN, NQU
      real              T, HLAST, HNEXT, Y(NEQMAX), YDOT(NEQMAX),
     1                  YSAVE(NEQMAX,*), R(NEQMAX), ACOR(NEQMAX,2),
     2                  HMIN, HMAX
```

1: NEQ — INTEGER *Input*

On entry: the number of equations being solved.

2: NEQMAX — INTEGER *Input*

On entry: an upper bound on the number of equations to be solved.

3: T — ***real*** *Input*

On entry: the current value of the independent variable.

4: HLAST — ***real*** *Input*

On entry: the last step size successfully used by the integrator.

5: HNEXT — ***real*** *Input/Output*

On entry: the step size that the integrator proposes to take on the next step.

On exit: the next step size to be used. If this is different from the input value, then IMON must be set to 4.

6: Y(NEQMAX) — ***real*** array *Input/Output*

On entry: the values of the dependent variables, y, evaluated at t.

On exit: these values must not be changed unless IMON is set to 2.

7: YDOT(NEQMAX) — ***real*** array *Input*

On entry: the time derivatives y' of the vector y.

8: YSAVE(NEQMAX,*) — ***real*** array *Input*

On entry: workspace to enable the user to carry out interpolation using either of the routines D02XJF or D02XKF.

9: R(NEQMAX) — ***real*** array *Input*

On entry: if IMON = 0 and INLN = 3, the first NEQ elements contain the residual vector $A(t,y)y' - g(t,y)$.

10: ACOR(NEQMAX,2) — ***real*** array *Input*

On entry: with IMON = 1, ACOR(i, 1) contains the weight used for the ith equation when the norm is evaluated, and ACOR(i, 2) contains the estimated local error for the ith equation. The scaled local error at the end of a timestep may be obtained by calling the ***real*** function D02ZAF as follows

```
          IFAIL = 1
          ERRLOC = D02ZAF(NEQ, ACOR(1,2), ACOR(1,1), IFAIL)
    C     CHECK IFAIL BEFORE PROCEEDING
```

11: IMON — INTEGER *Input/Output*

On entry: a flag indicating under what circumstances MONITR was called:

IMON = −2

entry from the integrator after IRES = 4 (set in RESID) caused an early termination (this facility could be used to locate discontinuities).

IMON = −1

the current step failed repeatedly.

IMON = 0

entry after a call to the internal nonlinear equation solver (see below).

IMON = 1

the current step was successful.

On exit: IMON may be reset to determine subsequent action in D02NHF:

IMON = −2

integration is to be halted. A return will be made from the integrator to the calling (sub)program with IFAIL = 12.

IMON = −1

allow the integrator to continue with its own internal strategy. The integrator will try up to 3 restarts unless IMON is set $\neq -1$ on exit.

IMON = 0

return to the internal nonlinear equation solver, where the action taken is determined by the value of INLN (see below).

IMON = 1

normal exit to the integrator to continue integration.

IMON = 2

restart the integration at the current time point. The integrator will restart from order 1 when this option is used. The MONITR provided solution Y will be used for the initial conditions.

IMON = 3

try to continue with the same step size and order as was to be used before the call to MONITR. HMIN and HMAX may be altered if desired.

IMON = 4

continue the integration but using a new value HNEXT and possibly new values of HMIN and HMAX.

12: INLN — INTEGER *Output*

On exit: the action to be taken by the internal nonlinear equation solver when MONITR is exited with IMON = 0. By setting INLN = 3 and returning to the integrator, the residual vector is evaluated and placed in the array R, and then MONITR is called again. At present this is the only option available: INLN must not be set to any other value.

13: HMIN — ***real*** *Input/Output*

On entry: the minimum step size to be taken on the next step.

On exit: the minimum step size to be used. If this is different from the input value, then IMON must be set to 3 or 4.

14: HMAX — ***real*** *Input/Output*

On entry: the maximum step size to be taken on the next step.

On exit: the maximum step size to be used. If this is different from the input value, then IMON must be set to 3 or 4. If HMAX is set to zero, no limit is assumed.

15: NQU — INTEGER *Input*

On entry: the order of the integrator used on the last step. This is supplied to enable the user to carry out interpolation using either of the routines D02XJF or D02XKF.

MONITR must be declared as EXTERNAL in the (sub)program from which D02NHF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

21: LDERIV(2) — LOGICAL array *Input/Output*

On entry: LDERIV(1) must be set to .TRUE., if the user has supplied both an initial y and an initial y'. LDERIV(1) must be set to .FALSE., if only the initial y has been supplied.

LDERIV(2) must be set to .TRUE. if the integrator is to use a modified Newton method to evaluate the initial y and y'. Note that y and y', if supplied, are used as initial estimates. This method involves taking a small step at the start of the integration, and if ITASK = 6 on entry, T and TOUT will be set to the result of taking this small step. LDERIV(2) must be set to .FALSE. if the integrator is to use functional iteration to evaluate the initial y and y', and if this fails a modified Newton method will then be attempted. LDERIV(2) =.TRUE. is recommended if there are implicit equations or the initial y and y' are zero.

On exit: LDERIV(1) is normally unchanged. However if ITASK = 6 and internal initialisation was successful then LDERIV(1) = .TRUE..

LDERIV(2) = .TRUE., if implicit equations were detected. Otherwise LDERIV(2) = .FALSE..

22: ITASK — INTEGER *Input*

On entry: the task to be performed by the integrator. The permitted values for ITASK and their meanings are detailed below:

ITASK = 1

normal computation of output values of $y(t)$ at t = TOUT (by overshooting and interpolating).

ITASK = 2

take one step only and return.

ITASK = 3

stop at the first internal integration point at or beyond t = TOUT and return.

ITASK = 4

normal computation of output values of $y(t)$ at t = TOUT but without overshooting t = TCRIT. TCRIT must be specified as an option in one of the integrator setup routines prior to the first call to the integrator, or specified in the optional input routine prior to a continuation call. TCRIT may be equal to or beyond TOUT, but not before it, in the direction of integration.

ITASK = 5

take one step only and return, without passing TCRIT. TCRIT must be specified as under ITASK = 4.

ITASK = 6

the integrator will solve for the initial values of y and y' only and then return to the calling (sub)program without doing the integration. This option can be used to check the initial values of y and y'. Functional iteration or a 'small' backward Euler method used in conjunction with a damped Newton iteration is used to calculate these values (see LDERIV above). Note that if a backward Euler step is used then the value of t will have been advanced a short distance from the initial point.

Note. If D02NHF is recalled with a different value of ITASK (and TOUT altered), then the initialisation procedure is repeated, possibly leading to different initial conditions.

Constraint: $1 \leq \text{ITASK} \leq 6$.

23: ITRACE — INTEGER *Input*

On entry: the level of output that is printed by the integrator. ITRACE may take the value -1, 0, 1, 2 or 3. If ITRACE < -1, then -1 is assumed and similarly if ITRACE > 3, then 3 is assumed. If ITRACE $= -1$, no output is generated. If ITRACE $= 0$, only warning messages are printed on the current error message unit (see X04AAF). If ITRACE > 0 then warning messages are printed as above, and on the current advisory message unit (see X04ABF) output is generated which details Jacobian entries, the nonlinear iteration and the time integration. The advisory messages are given in greater detail the larger the value of ITRACE.

24: IFAIL — INTEGER *Input/Output*

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

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq 0$ on exit, users are recommended to set IFAIL to -1 before entry. **It is then essential to test the value of IFAIL on exit**. To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6 Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL = 1

An illegal input was detected on entry, or after an internal call to MONITR. If ITRACE > -1, then the form of the error will be detailed on the current error message unit (see X04AAF).

IFAIL = 2

The maximum number of steps specified has been taken (see the description of optional inputs in the integrator setup routines and the optional input continuation D02NZF).

IFAIL = 3

With the given values of RTOL and ATOL no further progress can be made across the integration range from the current point T. The components Y(1),Y(2),...,Y(NEQ) contain the computed values of the solution at the current point T.

IFAIL = 4

There were repeated error test failures on an attempted step, before completing the requested task, but the integration was successful as far as T. The problem may have a singularity, or the local error requirements may be inappropriate.

IFAIL = 5

There were repeated convergence test failures on an attempted step, before completing the requested task, but the integration was successful as far as T. This may be caused by an inaccurate Jacobian matrix or one which is incorrectly computed.

IFAIL = 6

Some error weight w_i became zero during the integration (see description of ITOL). Pure relative error control (ATOL(i) = 0.0) was requested on a variable (the ith) which has now vanished. The integration was successful as far as T.

IFAIL = 7

The user-supplied subroutine RESID set its error flag (IRES = 3) continually despite repeated attempts by the integrator to avoid this.

IFAIL = 8

LDERIV(1) = .FALSE. on entry but the internal initialisation routine was unable to initialise y' (more detailed information may be directed to the current error message unit, see X04AAF).

IFAIL = 9

A singular Jacobian $\frac{\partial r}{\partial y}$ has been encountered. The user should check his problem formulation and Jacobian calculation.

IFAIL = 10

An error occurred during Jacobian formulation or back-substitution (a more detailed error description may be directed to the current error message unit, see X04AAF).

IFAIL = 11

The user-supplied subroutine RESID signalled the integrator to halt the integration and return (IRES = 2). Integration was successful as far as T.

IFAIL = 12

The user-supplied subroutine MONITR set IMON = -2 and so forced a return but the integration was successful as far as T.

IFAIL = 13

The requested task has been completed, but it is estimated that a small change in RTOL and ATOL is unlikely to produce any change in the computed solution. (Only applies when the user is not operating in one step mode, that is when ITASK $\neq$ 2 or 5).

IFAIL = 14

The values of RTOL and ATOL are so small that the routine is unable to start the integration.

IFAIL = 15

The linear algebra setup routine D02NTF was not called before the call to D02NHF.

7 Accuracy

The accuracy of the numerical solution may be controlled by a careful choice of the parameters RTOL and ATOL, and to a much lesser extent by the choice of norm. Users are advised to use scalar error control unless the components of the solution are expected to be poorly scaled. For the type of decaying solution typical of many stiff problems, relative error control with a small absolute error threshold will be most appropriate (that is the user is advised to choose ITOL = 1 with ATOL(1) small but positive).

8 Further Comments

The cost of computing a solution depends critically on the size of the differential system and to a lesser extent on the degree of stiffness of the problem. For D02NHF the cost is proportional to NEQ × $(\mathrm{ML}+\mathrm{MU}+1)^2$ though for problems which are only mildly nonlinear the cost may be dominated by factors proportional to NEQ × (ML+MU+1) except for very large problems.

In general the user is advised to choose the BDF option (setup routine D02NVF) but if efficiency is of great importance and especially if it is suspected that $\frac{\partial}{\partial y}(A^{-1}g)$ has complex eigenvalues near the imaginary axis for some part of the integration, the user should try the BLEND option (setup routine D02NWF).

9 Example

We solve the well-known stiff Robertson problem written as an implicit differential system and in implicit form

$$\begin{aligned} r_1 &= a'+b'+c' \\ r_2 &= 0.04a-1.0\mathrm{E}4bc-3.0\mathrm{E}7b^2- \quad b' \\ r_3 &= 3.0\mathrm{E}7\mathrm{b}^2- \quad c' \end{aligned}$$

exploiting the fact that we can show that $(a+b+c)'=0$ for all time. Integration is over the range [0,10.0] with initial conditions $a = 1.0$ and $b = c = 0.0$ using scalar relative error control and vector absolute error control (ITOL = 2). We integrate using a BDF method (setup routine D02NVF) and a modified Newton method. The Jacobian is calculated numerically and we employ a default monitor, dummy routine D02NBY. We perform a normal integration (ITASK = 1) to obtain the value at TOUT = 10.0 by integrating past TOUT and interpolating. We also illustrate the use of ITASK = 6 to calculate initial values of y and y' and then return without integrating further.

9.1 Program Text

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

```
*     D02NHF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, NEQMAX, NRW, NINF, ML, MU, NJCPVT, NWKJAC,
     +                 MAXORD, NY2DIM, MAXSTP, MXHNIL
      PARAMETER        (NEQ=3,NEQMAX=NEQ,NRW=50+4*NEQMAX,NINF=23,ML=1,
     +                 MU=2,NJCPVT=NEQMAX,NWKJAC=NEQMAX*(2*ML+MU+1),
     +                 MAXORD=5,NY2DIM=MAXORD+1,MAXSTP=200,MXHNIL=5)
      real             H0, HMAX, HMIN, TCRIT
      PARAMETER        (H0=1.0e-4,HMAX=10.0e0,HMIN=1.0e-10,TCRIT=0.0e0)
      LOGICAL          PETZLD
      PARAMETER        (PETZLD=.FALSE.)
*     .. Local Scalars ..
      real             H, HU, T, TCUR, TOLSF, TOUT
      INTEGER          I, IFAIL, IMXER, ITASK, ITOL, ITRACE, NITER, NJE,
     +                 NQ, NQU, NRE, NST
*     .. Local Arrays ..
      real             ATOL(NEQMAX), CONST(6), RTOL(NEQMAX), RWORK(NRW),
     +                 WKJAC(NWKJAC), Y(NEQMAX), YDOT(NEQMAX),
     +                 YSAVE(NEQMAX,NY2DIM)
      INTEGER          INFORM(NINF), JACPVT(NJCPVT)
      LOGICAL          ALGEQU(NEQMAX), LDERIV(2)
*     .. External Subroutines ..
      EXTERNAL         D02NBY, D02NHF, D02NHZ, D02NTF, D02NVF, D02NYF,
     +                 RESID, X04ABF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02NHF Example Program Results'
      CALL X04ABF(1,NOUT)
*
*     Set ITASK=6 to provide initial estimates of solution and its
*     time derivative. Default values for the array CONST are used.
*     Use the B.D.F. formulae with a Newton method.
*     Employ scalar relative tolerance and vector absolute tolerance.
*     The Jacobian is evaluated internally.
*     MONITR subroutine replaced by NAG dummy routine D02NBY.
*
      T = 0.0e0
      TOUT = 10.0e0
      ITASK = 6
      Y(1) = 1.0e0
      Y(2) = 0.0e0
      Y(3) = 0.0e0
      LDERIV(1) = .FALSE.
      LDERIV(2) = .FALSE.
      ITOL = 2
      RTOL(1) = 1.0e-4
      ATOL(1) = 1.0e-6
      ATOL(2) = 1.0e-7
      ATOL(3) = 1.0e-6
      DO 20 I = 1, 6
         CONST(I) = 0.0e0
   20 CONTINUE
      IFAIL = 0
*
      CALL D02NVF(NEQMAX,NY2DIM,MAXORD,'Newton',PETZLD,CONST,TCRIT,HMIN,
     +            HMAX,H0,MAXSTP,MXHNIL,'Average-L2',RWORK,IFAIL)
*
      CALL D02NTF(NEQ,NEQMAX,'Numerical',ML,MU,NWKJAC,NJCPVT,RWORK,
     +            IFAIL)
*
*     Soft fail and error messages only
      ITRACE = 0
      IFAIL = 1
*
      CALL D02NHF(NEQ,NEQMAX,T,TOUT,Y,YDOT,RWORK,RTOL,ATOL,ITOL,INFORM,
     +            RESID,YSAVE,NY2DIM,D02NHZ,WKJAC,NWKJAC,JACPVT,NJCPVT,
     +            D02NBY,LDERIV,ITASK,ITRACE,IFAIL)
*
      WRITE (NOUT,*)
      IF (IFAIL.EQ.0) THEN
         WRITE (NOUT,99999) ' Initial    Y : ', (Y(I),I=1,NEQ)
         WRITE (NOUT,99999) ' Initial YDOT : ', (YDOT(I),I=1,NEQ)
      ELSE
         WRITE (NOUT,99997) 'Exit D02NHF with IFAIL = ', IFAIL,
     +     '  and T = ', T
         STOP
      END IF
*     Use these initial estimates and integrate to TOUT (overshoot and
*     interpolate)
      ITASK = 1
      IFAIL = 1
      TOUT = 10.0e0
```

```
*
      CALL D02NHF(NEQ,NEQMAX,T,TOUT,Y,YDOT,RWORK,RTOL,ATOL,ITOL,INFORM,
     +            RESID,YSAVE,NY2DIM,D02NHZ,WKJAC,NWKJAC,JACPVT,NJCPVT,
     +            D02NBY,LDERIV,ITASK,ITRACE,IFAIL)
*
      IF (IFAIL.EQ.0) THEN
         WRITE (NOUT,*)
         WRITE (NOUT,*)
     +     '    X           Y(1)           Y(2)           Y(3)'
         WRITE (NOUT,99998) TOUT, (Y(I),I=1,NEQ)
*
         CALL D02NYF(NEQ,NEQMAX,HU,H,TCUR,TOLSF,RWORK,NST,NRE,NJE,NQU,
     +               NQ,NITER,IMXER,ALGEQU,INFORM,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99996) ' HUSED = ', HU, '  HNEXT = ', H,
     +     '  TCUR = ', TCUR
         WRITE (NOUT,99995) ' NST = ', NST, '    NRE = ', NRE,
     +     '    NJE = ', NJE
         WRITE (NOUT,99995) ' NQU = ', NQU, '    NQ  = ', NQ,
     +     '  NITER = ', NITER
         WRITE (NOUT,99994) ' Max err comp = ', IMXER
      ELSE
         WRITE (NOUT,*)
         WRITE (NOUT,99997) 'Exit D02NHF with IFAIL = ', IFAIL,
     +     '  and T = ', T
      END IF
      STOP
*
99999 FORMAT (1X,A,3(F11.4,2X))
99998 FORMAT (1X,F8.3,3(F13.5,2X))
99997 FORMAT (1X,A,I2,A,e12.5)
99996 FORMAT (1X,A,e12.5,A,e12.5,A,e12.5)
99995 FORMAT (1X,A,I6,A,I6,A,I6)
99994 FORMAT (1X,A,I4)
      END
*
      SUBROUTINE RESID(NEQ,T,Y,YDOT,R,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IRES, NEQ
*     .. Array Arguments ..
      real              R(NEQ), Y(NEQ), YDOT(NEQ)
*     .. Executable Statements ..
      R(1) = -YDOT(1) - YDOT(2) - YDOT(3)
      R(2) = -YDOT(2)
      R(3) = -YDOT(3)
      IF (IRES.EQ.1) THEN
         R(1) = 0.0e0 + R(1)
         R(2) = 0.04e0*Y(1) - 1.0e4*Y(2)*Y(3) - 3.0e7*Y(2)*Y(2) + R(2)
         R(3) = 3.0e7*Y(2)*Y(2) + R(3)
      END IF
      RETURN
      END
```

9.2 Program Data

None.

9.3 Prograrr Results

```
D02NHF Example Program Results
WARNING... EQUATION(=I1) AND POSSIBLY OTHER EQUATIONS ARE
IMPLICIT AND IN CALCULATING THE INITIAL VALUES THE EQNS
WILL BE TREATED AS IMPLICIT.
IN ABOVE MESSAGE I1 =          1

 Initial    Y :       1.0000       0.0000       0.0000
 Initial YDOT :      -0.0400       0.0400       0.0000
WARNING... EQUATION(=I1) AND POSSIBLY OTHER EQUATIONS ARE
IMPLICIT AND IN CALCULATING THE INITIAL VALUES THE EQNS
WILL BE TREATED AS IMPLICIT.
IN ABOVE MESSAGE I1 =          1

    X          Y(1)           Y(2)           Y(3)
  10.000      0.84135        0.00002        0.15863

 HUSED =  0.91752D+00  HNEXT =  0.91752D+00  TCUR =  0.10885D+02
 NST =      51    NRE =    118    NJE =     14
 NQU =       4    NQ  =      4  NITER =     68
 Max err comp =    3
```

D02NJF – NAG Fortran Library Routine Document

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

1. Purpose

D02NJF is a forward communication routine for integrating stiff systems of implicit ordinary differential equations coupled with algebraic equations when the Jacobian is a sparse matrix.

2. Specification

```
 SUBROUTINE D02NJF (NEQ, NEQMAX, T, TOUT, Y, YDOT, RWORK, RTOL,
1                   ATOL, ITOL, INFORM, RESID, YSAVE, NY2DIM, JAC,
2                   WKJAC, NWKJAC, JACPVT, NJCPVT, MONITR, LDERIV,
3                   ITASK, ITRACE, IFAIL)
 INTEGER        NEQ, NEQMAX, ITOL, INFORM(23), NY2DIM, NWKJAC,
1               JACPVT(NJCPVT), NJCPVT, ITASK, ITRACE, IFAIL
 real           T, TOUT, Y(NEQMAX), YDOT(NEQMAX),
1               RWORK(50+4*NEQMAX), RTOL(*), ATOL(*),
2               YSAVE(NEQMAX,NY2DIM), WKJAC(NWKJAC)
 LOGICAL        LDERIV(2)
 EXTERNAL       RESID, JAC, MONITR
```

3. Description

D02NJF is a general purpose routine for integrating the initial value problem for a stiff system of implicit ordinary differential equations coupled with algebraic equations written in the form,

$$A(t,y)y' = g(t,y).$$

It is designed specifically for the case where the resulting Jacobian is a sparse matrix (see description of argument JAC in Section 5).

Both interval and step oriented modes of operation are available and also modes designed to permit intermediate output within an interval oriented mode.

An outline of a typical calling program for D02NJF is given below. It calls the sparse matrix linear algebra setup routine D02NUF, the Backward Differentiation Formula (BDF) integrator setup routine D02NVF, its diagnostic counterpart D02NYF, and the sparse matrix linear algebra diagnostic routine D02NXF.

```
C
C     declarations
C
      EXTERNAL RESID, JAC, MONITR
             .
             .
             .
      IFAIL = 0
      CALL D02NVF(...,IFAIL)
      CALL D02NUF(NEQ, NEQMAX, JCEVAL, NWKJAC, IA, NIA, JA, NJA,
     + JACPVT, NJCPVT, SENS, U, ETA, LBLOCK, ISPLIT, RWORK,
     + IFAIL)
      IFAIL = -1
      CALL D02NJF(NEQ, NEQMAX, T, TOUT, Y, YDOT, RWORK, RTOL,
     + ATOL, ITOL, INFORM, RESID, YSAVE, NY2DIM, JAC, WKJAC,
     + NWKJAC, JACPVT, NJCPVT, MONITR, LDERIV, ITASK, ITRACE,
     + IFAIL)
      IF(IFAIL.EQ.1.OR.IFIAL.GE.14)STOP
      IFAIL = 0
      CALL D02NXF(...)
      CALL D02NYF(...)
            .
            .
            .
      STOP
      END
```

The linear algebra setup routine D02NUF and one of the integrator setup routines, D02MVF, D02NVF or D02NWF, must be called prior to the call of D02NJF. Either or both of the integrator diagnostic routine D02NYF, or the sparse matrix linear algebra diagnostic routine D02NXF, may be called after the call to D02NJF. There is also a routine, D02NZF, designed to permit the user to change step size on a continuation call to D02NJF without restarting the integration process.

4. References

See Subchapter Introduction.

5. Parameters

1: NEQ – INTEGER. *Input*

On entry: the number of equations to be solved.

Constraint: NEQ ≥ 1.

2: NEQMAX – INTEGER. *Input*

On entry: a bound on the maximum number of equations to be solved during the integration.

Constraint: NEQMAX ≥ NEQ.

3: T – ***real.*** *Input/Output*

On entry: the value of the independent variable t. The input value of T is used only on the first call as the initial point of the integration.

On exit: the value at which the computed solution y is returned (usually at TOUT).

4: TOUT – ***real.*** *Input/Output*

On entry: the next value of t at which a computed solution is desired. For the initial t, the input value of TOUT is used to determine the direction of integration. Integration is permitted in either direction (see also ITASK).

On exit: normally unchanged. However, when ITASK = 6, then TOUT contains the value of T at which initial values have been computed without performing any integration. See descriptions of ITASK and LDERIV below.

5: Y(NEQMAX) – ***real*** array. *Input/Output*

On entry: the values of the dependent variables (solution). On the first call the first NEQ elements of y must contain the vector of initial values.

On exit: the computed solution vector, evaluated at t (usually t = TOUT).

6: YDOT(NEQMAX) – ***real*** array. *Input/Output*

On entry: if LDERIV(1) = .TRUE., YDOT must contain approximations to the time derivatives y' of the vector y. If LDERIV(1) = .FALSE., then YDOT need not be set on entry.

On exit: the time derivatives y' of the vector y at the last integration point.

7: RWORK(50+4*NEQMAX) – ***real*** array. *Workspace*

8: RTOL(*) – ***real*** array. *Input*

Note: the dimension of the array RTOL must be at least 1 or NEQ (see ITOL).

On entry: the relative local error tolerance.

Constraint: RTOL(i) ≥ 0.0 for all relevant i (see ITOL).

9: ATOL(*) – ***real*** array. *Input*

Note: the dimension of the array ATOL must be at least 1 or NEQ (see ITOL).

On entry: the absolute local error tolerance.

Constraint: ATOL(i) ≥ 0.0 for all relevant i (see ITOL).

10: ITOL – INTEGER. *Input*

On entry: a value to indicate the form of the local error test. ITOL indicates to D02NJF whether to interpret either or both of RTOL or ATOL as a vector or a scalar. The error test to be satisfied is $\|e_i/w_i\| < 1.0$, where w_i is defined as follows

ITOL	RTOL	ATOL	w_i
1	scalar	scalar	$\mathrm{RTOL}(1)\times\lvert y_i\rvert + \mathrm{ATOL}(1)$
2	scalar	vector	$\mathrm{RTOL}(1)\times\lvert y_i\rvert + \mathrm{ATOL}(i)$
3	vector	scalar	$\mathrm{RTOL}(i)\times\lvert y_i\rvert + \mathrm{ATOL}(1)$
4	vector	vector	$\mathrm{RTOL}(i)\times\lvert y_i\rvert + \mathrm{ATOL}(i)$

e_i is an estimate of the local error in y_i, computed internally, and the choice of norm to be used is defined by a previous call to an integrator setup routine.

Constraint: 1 ≤ ITOL ≤ 4.

11: INFORM(23) – INTEGER array. *Workspace*

12: RESID – SUBROUTINE, supplied by the user. *External Procedure*

RESID must evaluate the residual

$$r = g(t,y) - A(t,y)y'$$

in one case and

$$r = -A(t,y)y'$$

in another.

Its specification is:

```
SUBROUTINE RESID(NEQ, T, Y, YDOT, R, IRES)
INTEGER    NEQ, IRES
real       T, Y(NEQ), YDOT(NEQ), R(NEQ)
```

1: NEQ – INTEGER. *Input*

On entry: the number of equations being solved.

2: T – ***real***. *Input*

On entry: the current value of the independent variable t.

3: Y(NEQ) – ***real*** array. *Input*

On entry: the value of y_i, for i = 1,2,...,NEQ.

4: YDOT(NEQ) – ***real*** array. *Input*

On entry: the value of y_i' at t, for i = 1,2,...,NEQ.

5: R(NEQ) – ***real*** array. *Output*

On exit: R(i) must contain the ith component of r, for i = 1,2,...,NEQ where

$$r = g(t,y) - A(t,y)y' \qquad (1)$$

or

$$r = -A(t,y)y' \qquad (2)$$

and where the definition of r is determined by the input value of IRES.

6: IRES – INTEGER. *Input/Output*

On entry: the form of the residual that must be returned in array R. If IRES = –1, then the residual defined in equation (2) above must be returned. If IRES = 1, then the residual defined in equation (1) above must be returned.

On exit: IRES should be unchanged unless one of the following actions is required of the integrator, in which case IRES should be set accordingly.

IRES = 2

indicates to the integrator that control should be passed back immediately to the calling (sub)program with the error indicator set to IFAIL = 11.

IRES = 3

indicates to the integrator that an error condition has occurred in the solution vector, its time derivative or in the value of t. The integrator will use a smaller time step to try to avoid this condition. If this is not possible the integrator returns to the calling (sub)program with the error indicator set to IFAIL = 7.

IRES = 4

indicates to the integrator to stop its current operation and to enter the MONITR routine immediately with parameter IMON = –2.

RESID must be declared as EXTERNAL in the (sub)program from which D02NJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

13: YSAVE(NEQMAX, NY2DIM) – ***real*** array. *Workspace*

14: NY2DIM – INTEGER. *Input*

On entry: the second dimension of the array YSAVE as declared in the (sub)program from which D02NJF is called. An appropriate value for NY2DIM is described in the specifications of the integrator setup routines D02MVF, D02NVF and D02NWF. This value must be the same as that supplied to the integrator setup routine.

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

JAC must evaluate the Jacobian of the system. If this option is not required, JAC must be the dummy routine D02NJZ. (D02NJZ is included in the NAG Fortran Library and so need not be supplied by the user. Its name may be implementation dependent: see the Users' Note for your implementation for details.) The user indicates to the integrator whether this option is to be used by setting the parameter JCEVAL appropriately in a call to the linear algebra setup routine D02NUF.

First we must define the system of nonlinear equations which is solved internally by the integrator. The time derivative, y', generated internally, has the form

$$y' = (y-z)/(hd)$$

where h is the current step size and d is a parameter that depends on the integration method in use. The vector y is the current solution and the vector z depends on information from previous time steps. This means that $\frac{d}{dy'}(\ ) = \frac{1}{(hd)}\frac{d}{dy}(\ )$. The system of nonlinear equations that is solved has the form

$$A(t,y)y' - g(t,y) = 0$$

but is solved in the form

$$r(t,y) = 0$$

where r is the function defined by

$$r(t,y) = (hd)(A(t,y)(y-z)/(hd)-g(t,y)).$$

It is the Jacobian matrix $\frac{\partial r}{\partial y}$ that the user must supply in the routine JAC as follows

$$\frac{\partial r_i}{\partial y_j} = a_{ij}(t,y) + (hd)\frac{\partial}{\partial y_j}\left(\sum_{k=1}^{\mathrm{NEQ}} a_{ik}(t,y)y'_k - g_i(t,y)\right)$$

Its specification is:

```
SUBROUTINE JAC(NEQ, T, Y, YDOT, H, D, J, PDJ)
INTEGER    NEQ, J
real       T, Y(NEQ), YDOT(NEQ), H, D, PDJ(NEQ)
```

1: NEQ – INTEGER. *Input*

On entry: the number of equations being solved.

2: T – ***real***. *Input*

On entry: the current value of the independent variable t.

3: Y(NEQ) – ***real*** array. *Input*

On entry: the current solution component y_i, $i = 1,2,\ldots,\mathrm{NEQ}$.

4: YDOT(NEQ) – ***real*** array. *Input*

On entry: the derivative of the solution at the current point t.

5: H – ***real***. *Input*

On entry: the current step size.

6: D – ***real***. *Input*

On entry: the parameter d which depends on the integration method.

7: J – INTEGER. *Input*

On entry: the column of the Jacobian that JAC must return in the array PDJ.

8: PDJ(NEQ) – ***real*** array. *Output*

On exit: PDJ(i) should be set to the (i,j)th element of the Jacobian, where j is given by J above. Only non-zero elements of this array need be set, since it is preset to zero before the call to JAC.

JAC must be declared as EXTERNAL in the (sub)program from which D02NJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

16: WKJAC(NWKJAC) – ***real*** array. *Workspace*

17: NWKJAC – INTEGER. *Input*

On entry: the dimension of the array WKJAC as declared in the (sub)program from which D02NJF is called. The actual size depends on whether the sparsity structure is supplied or whether it is to be estimated. An appropriate value for NWKJAC is described in the specification for the linear algebra setup routine D02NUF. This value must be the same as that supplied to D02NUF.

18: JACPVT(NJCPVT) – INTEGER array. *Workspace*

19: NJCPVT – INTEGER. *Input*

On entry: the dimension of the array JACPVT as declared in the (sub)program from which D02NJF is called. The actual size depends on whether the sparsity structure is supplied or whether it is to be estimated. An appropriate value for NJCPVT is described in the specification for the linear algebra setup routine D02NUF. This value must be same as that supplied to D02NUF.

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

MONITR performs tasks requested by the user. If this option is not required the actual argument for MONITR must be the dummy routine D02NBY. (D02NBY is included in the NAG Fortran Library and so need not be supplied by the user. Its name may be implementation dependent: see the Users' Note for your implementation for details.)

Its specification is:

```
SUBROUTINE MONITR(NEQ, NEQMAX, T, HLAST, HNEXT, Y, YDOT, YSAVE, R,
1             ACOR, IMON, INLN, HMIN, HMAX, NQU)
INTEGER       NEQ, NEQMAX, IMON, INLN, NQU
real          T, HLAST, HNEXT, Y(NEQMAX), YDOT(NEQMAX), YSAVE(NEQMAX,*),
1             R(NEQMAX), ACOR(NEQMAX,2), HMIN, HMAX
```

1: NEQ – INTEGER. *Input*

On entry: the number of equations being solved.

2: NEQMAX – INTEGER. *Input*

On entry: an upper bound on the number of equations to be solved.

3: T – ***real***. *Input*

On entry: the current value of the independent variable.

4: HLAST – ***real***. *Input*

On entry: the last step size successfully used by the integrator.

5: HNEXT – ***real***. *Input/Output*

On entry: the step size that the integrator proposes to take on the next step.

On exit: the next step size to be used. If this is different from the input value, then IMON must be set to 4.

6: Y(NEQMAX) – ***real*** array. *Input/Output*

On entry: the values of the dependent variables, y, evaluated at t.

On exit: these values must not be changed unless IMON is set to 2.

7: YDOT(NEQMAX) – ***real*** array. *Input*

On entry: the time derivatives y' of the vector y.

8: YSAVE(NEQMAX,*) – ***real*** array. *Input*

On entry: workspace to enable the user to carry out interpolation using either of the routines D02XJF or D02XKF.

9: R(NEQMAX) – ***real*** array. *Input*

On entry: if IMON = 0 and INLN = 3, the first NEQ elements contain the residual vector $A(t,y)y' - g(t,y)$.

10: ACOR(NEQMAX,2) – ***real*** array. *Input*

On entry: with IMON = 1, ACOR(i,1) contains the weight used for the ith equation when the norm is evaluated and ACOR(i,2) contains the estimated local error for the ith equation. The scaled local error at the end of a timestep may be obtained by calling the ***real*** function D02ZAF as follows

```
      IFAIL = 1
      ERRLOC = D02ZAF(NEQ, ACOR(1,2), ACOR(1,1), IFAIL)
C     CHECK IFAIL BEFORE PROCEEDING
```

11: IMON – INTEGER. *Input/Output*

On entry: a flag indicating under what circumstances MONITR was called:

IMON = –2

entry from the integrator after IRES = 4 (set in RESID) caused an early termination (this facility could be used to locate discontinuities).

IMON = –1

the current step failed repeatedly.

IMON = 0

entry after a call to the internal nonlinear equation solver (see below).

IMON = 1

the current step was successful.

On exit: IMON may be reset to determine subsequent action in D02NJF:

IMON = –2

integration is to be halted. A return will be made from the integrator to the calling (sub)program with IFAIL = 12.

IMON = –1

allow the integrator to continue with its own internal strategy. The integrator will try up to 3 restarts unless IMON is set ≠ –1 on exit.

IMON = 0

return to the internal nonlinear equation solver, where the action taken is determined by the value of INLN (see below).

IMON = 1

normal exit to the integrator to continue integration.

IMON = 2

restart the integration at the current time point. The integrator will restart from order 1 when this option is used. The MONITR provided solution Y will be used for the initial conditions.

IMON = 3

try to continue with the same step size and order as was to be used before the call to MONITR. HMIN and HMAX may be altered if desired.

IMON = 4

continue the integration but using a new value HNEXT and possibly new values of HMIN and HMAX.

12: INLN – INTEGER. *Output*

On exit: the action to be taken by the internal nonlinear equation solver when MONITR is exited with IMON = 0. By setting INLN = 3 and returning to the integrator, the residual vector is evaluated and placed in the array R, and then MONITR is called again. At present this is the only option available: INLN must not be set to any other value.

13: HMIN – ***real***. *Input/Output*

On entry: the minimum step size to be taken on the next step.

On exit: the minimum step size to be used. If this is different from the input value, then IMON must be set to 3 or 4.

14: HMAX – *real*. *Input/Output*

On entry: the maximum step size to be taken on the next step.

On exit: the maximum step size to be used. If this is different from the input value, then IMON must be set to 3 or 4. If HMAX is set to zero, no limit is assumed.

15: NQU – INTEGER. *Input*

On entry: the order of the integrator used on the last step. This is supplied to enable the user to carry out interpolation using either of the routines D02XJF or D02XKF.

MONITR must be declared as EXTERNAL in the (sub)program from which D02NJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

21: LDERIV(2) – LOGICAL array. *Input/Output*

On entry: LDERIV(1) must be set to .TRUE. if the user has supplied both an initial y and an initial y'. LDERIV(1) must be set to .FALSE., if only the initial y has been supplied.

LDERIV(2) must be set to .TRUE., if the integrator is to use a modified Newton method to evaluate the initial y and y'. Note that y and y', if supplied, are used as initial estimates. This method involves taking a small step at the start of the integration, and if ITASK = 6 on entry, T and TOUT will be set to the result of taking this small step. LDERIV(2) must be set to .FALSE. if the integrator is to use functional iteration to evaluate the initial y and y', and if this fails a modified Newton method will then be attempted. LDERIV(2) = .TRUE. is recommended if there are implicit equations or the initial y and y' are zero.

On exit: LDERIV(1) is normally unchanged. However if ITASK = 6 and internal initialisation was successful then LDERIV(1) = .TRUE..

LDERIV(2) = .TRUE., if implicit equations were detected. Otherwise
LDERIV(2) = .FALSE..

22: ITASK – INTEGER. *Input*

On entry: the task to be performed by the integrator. The permitted values for ITASK and their meanings are detailed below:

ITASK = 1

normal computation of output values of $y(t)$ at t = TOUT (by overshooting and interpolating).

ITASK = 2

take one step only and return.

ITASK = 3

stop at the first internal integration point at or beyond t = TOUT and return.

ITASK = 4

normal computation of output values of $y(t)$ at t = TOUT but without overshooting t = TCRIT. TCRIT must be specified as an option in one of the integrator setup routines prior to the first call to the integrator, or specified in the optional input routine prior to a continuation call. TCRIT may be equal to or beyond TOUT, but not before it, in the direction of integration.

ITASK = 5

take one step only and return, without passing TCRIT. TCRIT must be specified as under ITASK = 4.

ITASK = 6

the integrator will solve for the initial values of y and y' only and then return to the calling (sub)program without doing the integration. This option can be used to check the initial values of y and y'. Functional iteration or a 'small' backward Euler method used in conjunction with a damped Newton iteration is used to calculate these values (see LDERIV above). Note that if a backward Euler step is used then the value of t will have been advanced a short distance from the initial point.

Note: if D02NJF is recalled with a different value of ITASK (and TOUT altered), then the initialisation procedure is repeated, possibly leading to different initial conditions.

Constraint: $1 \le$ ITASK ≤ 6.

23: ITRACE – INTEGER. *Input*

On entry: the level of output that is printed by the integrator. ITRACE may take the value -1, 0, 1, 2 or 3. If ITRACE < -1, then -1 is assumed and similarly if ITRACE > 3, then 3 is assumed. If ITRACE $= -1$, no output is generated. If ITRACE $= 0$, only warning messages are printed on the current error message unit (see X04AAF). If ITRACE > 0 then warning messages are printed as above, and on the current advisory message unit (see X04ABF) output is generated which details Jacobian entries, the nonlinear iteration and the time integration. The advisory messages are given in greater detail the larger the value of ITRACE.

24: IFAIL – INTEGER. *Input/Output*

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

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

For this routine, because the values of output parameters may be useful even if IFAIL $\neq 0$ on exit, users are recommended to set IFAIL to -1 before entry. **It is then essential to test the value of IFAIL on exit.** To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL = 1

An illegal input was detected on entry, or after an internal call to MONITR. If ITRACE > -1, then the form of the error will be detailed on the current error message unit (see X04AAF).

IFAIL = 2

The maximum number of steps specified has been taken (see the description of optional inputs in the integrator setup routines and the optional input continuation routine, D02NZF).

IFAIL = 3

With the given values of RTOL and ATOL no further progress can be made across the integration range from the current point T. The components Y(1),Y(2),...,Y(NEQ) contain the computed values of the solution at the current point T.

IFAIL = 4

There were repeated error test failures on an attempted step, before completing the requested task, but the integration was successful as far as T. The problem may have a singularity, or the local error requirements may be inappropriate.

IFAIL = 5

There were repeated convergence test failures on an attempted step, before completing the requested task, but the integration was successful as far as T. This may be caused by an inaccurate Jacobian matrix or one which is incorrectly computed.

IFAIL = 6

Some error weight w_i became zero during the integration (see description of ITOL). Pure relative error control (ATOL(i) = 0.0) was requested on a variable (the ith) which has now vanished. The integration was successful as far as T.

IFAIL = 7

The user-supplied subroutine RESID set its error flag (IRES = 3) continually despite repeated attempts by the integrator to avoid this.

IFAIL = 8

LDERIV(1) = .FALSE. on entry but the internal initialisation routine was unable to initialise y' (more detailed information may be directed to the current error message unit, see X04AAF).

IFAIL = 9

A singular Jacobian $\frac{\partial r}{\partial y}$ has been encountered. The user should check his problem formulation and Jacobian calculation.

IFAIL = 10

An error occurred during Jacobian formulation or back-substitution (a more detailed error description may be directed to the current error message unit, see X04AAF).

IFAIL = 11

The user-supplied subroutine RESID signalled the integrator to halt the integration and return (IRES = 2). Integration was successful as far as T.

IFAIL = 12

The user-supplied subroutine MONITR set IMON = –2 and so forced a return but the integration was successful as far as T.

IFAIL = 13

The requested task has been completed, but it is estimated that a small change in RTOL and ATOL is unlikely to produce any change in the computed solution. (Only applies when the user is not operating in one step mode, that is when ITASK ≠ 2 or 5).

IFAIL = 14

The values of RTOL and ATOL are so small that the routine is unable to start the integration.

IFAIL = 15

The linear algebra setup routine D02NUF was not called before the call to D02NJF.

7. Accuracy

The accuracy of the numerical solution may be controlled by a careful choice of the parameters RTOL and ATOL, and to a much lesser extent by the choice of norm. Users are advised to use scalar error control unless the components of the solution are expected to be poorly scaled. For the type of decaying solution typical of many stiff problems, relative error control with a small absolute error threshold will be most appropriate (that is the user is advised to choose ITOL = 1 with ATOL(1) small but positive).

8. Further Comments

Since numerical stability and memory are often conflicting requirements when solving ordinary differential systems where the Jacobian matrix is sparse we provide a diagnostic routine, D02NXF, whose aim is to inform the user how much memory is required to solve his problem and to give the user some indicators of numerical stability.

In general the user is advised to choose the backward differentiation formula option (setup D02NVF) but if efficiency is of great importance and especially if it is suspected that $\frac{\partial}{\partial y}(A^{-1}g)$ has complex eigenvalues near the imaginary axis for some part of the integration, the user should try the BLEND option (setup routine D02NWF).

9. Example

We solve the well-known stiff Robertson problem written as a mixed differential/algebraic system in implicit form

$$\begin{aligned} r_1 &= a + b + c - 1.0 \\ r_2 &= 0.04a - 1.0\mathrm{E}4bc - 3.0\mathrm{E}7b^2 - b' \\ r_3 &= 3.0\mathrm{E}7b^2 - c' \end{aligned}$$

exploiting the fact that, from the initial conditions $a = 1.0$ and $b = c = 0.0$, we know that $a + b + c = 1$ for all time. We integrate over the range [0,10.0] with vector relative error control and scalar absolute error control (ITOL = 3) and using the BDF integrator (setup routine D02NVF) and a modified Newton method. The Jacobian is evaluated, in turn, using the 'A' (Analytical) and 'F' (Full information) options. We provide a monitor routine to terminate the integration when the value of the component a falls below 0.9.

9.1. Program Text

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

```
*     D02NJF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, NEQMAX, NRW, NINF, NELTS, NJCPVT, NWKJAC,
     +                 NIA, NJA, MAXORD, NY2DIM, MAXSTP, MXHNIL
      PARAMETER        (NEQ=3,NEQMAX=NEQ,NRW=50+4*NEQMAX,NINF=23,
     +                 NELTS=8,NJCPVT=150,NWKJAC=100,NIA=NEQMAX+1,
     +                 NJA=NELTS,MAXORD=5,NY2DIM=MAXORD+1,MAXSTP=200,
     +                 MXHNIL=5)
      real             H0, HMAX, HMIN, TCRIT
      PARAMETER        (H0=0.0e0,HMAX=10.0e0,HMIN=1.0e-10,TCRIT=0.0e0)
      LOGICAL          PETZLD
      PARAMETER        (PETZLD=.TRUE.)
      real             ETA, U, SENS
      PARAMETER        (ETA=1.0e-4,U=0.1e0,SENS=1.0e-6)
      LOGICAL          LBLOCK
      PARAMETER        (LBLOCK=.TRUE.)
*     .. Local Scalars ..
      real             H, HU, T, TCUR, TOLSF, TOUT
      INTEGER          I, ICALL, IFAIL, IGROW, IMXER, ISPLIT, ITASK,
     +                 ITOL, ITRACE, LIWREQ, LIWUSD, LRWREQ, LRWUSD,
     +                 NBLOCK, NGP, NITER, NJE, NLU, NNZ, NQ, NQU, NRE,
     +                 NST
*     .. Local Arrays ..
      real             ATOL(NEQMAX), CONST(6), RTOL(NEQMAX), RWORK(NRW),
     +                 WKJAC(NWKJAC), Y(NEQMAX), YDOT(NEQMAX),
     +                 YSAVE(NEQMAX,NY2DIM)
      INTEGER          IA(NIA), INFORM(NINF), JA(NJA), JACPVT(NJCPVT)
      LOGICAL          ALGEQU(NEQMAX), LDERIV(2)
```

```
*     .. External Subroutines ..
      EXTERNAL         D02NJF, D02NUF, D02NVF, D02NXF, D02NYF, JAC,
     +                 MONITR, RESID, X04ABF
*     .. Data statements ..
      DATA             IA/1, 3, 6, 9/, JA/1, 2, 1, 2, 3, 1, 2, 3/
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02NJF Example Program Results'
      CALL X04ABF(1,NOUT)
*
*     First case. Integrate to TOUT by overshooting (ITASK=1) using
*     B.D.F formulae with a Newton method. Also set PETZLD to
*     .TRUE. so that the Petzold error test is used (since an algebraic
*     equation is defined in the system). Default values for the
*     array CONST are used. Employ vector relative tolerance and scalar
*     absolute tolerance. The Jacobian is supplied by JAC and its
*     structure is determined internally by calls to JAC.
*     The MONITR routine is used to force a return when the first
*     component of the system falls below the value 0.9.
*
      T = 0.0e0
      TOUT = 10.0e0
      ITASK = 1
      Y(1) = 1.0e0
      Y(2) = 0.0e0
      Y(3) = 0.0e0
      LDERIV(1) = .FALSE.
      LDERIV(2) = .FALSE.
      ITOL = 3
      RTOL(1) = 1.0e-4
      RTOL(2) = 1.0e-3
      RTOL(3) = 1.0e-4
      ATOL(1) = 1.0e-7
      DO 20 I = 1, 6
         CONST(I) = 0.0e0
   20 CONTINUE
      ISPLIT = 0
      IFAIL = 0
*
      CALL D02NVF(NEQMAX,NY2DIM,MAXORD,'Newton',PETZLD,CONST,TCRIT,HMIN,
     +            HMAX,H0,MAXSTP,MXHNIL,'Average-L2',RWORK,IFAIL)
      CALL D02NUF(NEQ,NEQMAX,'Analytical',NWKJAC,IA,NIA,JA,NJA,JACPVT,
     +            NJCPVT,SENS,U,ETA,LBLOCK,ISPLIT,RWORK,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) '  Analytic Jacobian, structure not supplied'
      WRITE (NOUT,*)
      WRITE (NOUT,*) '    X          Y(1)           Y(2)           Y(3)'
      WRITE (NOUT,99999) T, (Y(I),I=1,NEQ)
*
*     Soft fail and error messages only
      ITRACE = 0
      IFAIL = 1
*
      CALL D02NJF(NEQ,NEQMAX,T,TOUT,Y,YDOT,RWORK,RTOL,ATOL,ITOL,INFORM,
     +            RESID,YSAVE,NY2DIM,JAC,WKJAC,NWKJAC,JACPVT,NJCPVT,
     +            MONITR,LDERIV,ITASK,ITRACE,IFAIL)
*
      IF (IFAIL.EQ.0 .OR. IFAIL.EQ.12) THEN
         WRITE (NOUT,99999) T, (Y(I),I=1,NEQ)
         IFAIL = 0
*
```

```
         CALL D02NYF(NEQ,NEQMAX,HU,H,TCUR,TOLSF,RWORK,NST,NRE,NJE,NQU,
     +               NQ,NITER,IMXER,ALGEQU,INFORM,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99997) ' HUSED = ', HU, '  HNEXT = ', H,
     +     '  TCUR = ', TCUR
         WRITE (NOUT,99996) ' NST = ', NST, '    NRE = ', NRE,
     +     '    NJE = ', NJE
         WRITE (NOUT,99996) ' NQU = ', NQU, '    NQ  = ', NQ,
     +     '  NITER = ', NITER
         WRITE (NOUT,99995) ' Max err comp = ', IMXER
         ICALL = 0
*
         CALL D02NXF(ICALL,LIWREQ,LIWUSD,LRWREQ,LRWUSD,NLU,NNZ,NGP,
     +               ISPLIT,IGROW,LBLOCK,NBLOCK,INFORM)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99994) ' NJCPVT (required ', LIWREQ, '  used ',
     +     LIWUSD, ')'
         WRITE (NOUT,99994) ' NWKJAC (required ', LRWREQ, '  used ',
     +     LRWUSD, ')'
         WRITE (NOUT,99993) ' No. of LU-decomps ', NLU,
     +     '  No. of nonzeros ', NNZ
         WRITE (NOUT,99992) ' No. of FCN calls to form Jacobian ', NGP,
     +     '  Try ISPLIT ', ISPLIT
         WRITE (NOUT,99991) ' Growth est ', IGROW,
     +     '  No. of blocks on diagonal ', NBLOCK
      ELSE IF (IFAIL.EQ.10) THEN
         ICALL = 1
*
         CALL D02NXF(ICALL,LIWREQ,LIWUSD,LRWREQ,LRWUSD,NLU,NNZ,NGP,
     +               ISPLIT,IGROW,LBLOCK,NBLOCK,INFORM)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99994) ' NJCPVT (required ', LIWREQ, '  used ',
     +     LIWUSD, ')'
         WRITE (NOUT,99994) ' NWKJAC (required ', LRWREQ, '  used ',
     +     LRWUSD, ')'
      ELSE
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'Exit D02NJF with IFAIL = ', IFAIL,
     +     '  and T = ', T
      END IF
*
*     Second case. Integrate to TOUT by overshooting (ITASK=1) using
*     B.D.F formulae with a Newton method. Also set PETZLD to
*     .TRUE. so that the Petzold error test is used (since an algebraic
*     equation is defined in the system). Default values for the
*     array CONST are used. Employ vector relative tolerance and scalar
*     absolute tolerance. The Jacobian is supplied by JAC and its
*     structure is also supplied.
*     The MONITR routine is used to force a return when the first
*     component of the system falls below the value 0.9.
*
      T = 0.0e0
      Y(1) = 1.0e0
      Y(2) = 0.0e0
      Y(3) = 0.0e0
*
      ISPLIT = 0
      IFAIL = 0
*
```

```
      CALL D02NVF(NEQMAX,NY2DIM,MAXORD,'Newton',PETZLD,CONST,TCRIT,HMIN,
     +            HMAX,H0,MAXSTP,MXHNIL,'Average-L2',RWORK,IFAIL)
*
      CALL D02NUF(NEQ,NEQMAX,'Full information',NWKJAC,IA,NIA,JA,NJA,
     +            JACPVT,NJCPVT,SENS,U,ETA,LBLOCK,ISPLIT,RWORK,IFAIL)
*
      LDERIV(1) = .FALSE.
      LDERIV(2) = .FALSE.
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) '  Analytic Jacobian, structure supplied'
      WRITE (NOUT,*)
      WRITE (NOUT,*) '    X          Y(1)           Y(2)           Y(3)'
      WRITE (NOUT,99999) T, (Y(I),I=1,NEQ)
      IFAIL = 1
*
      CALL D02NJF(NEQ,NEQMAX,T,TOUT,Y,YDOT,RWORK,RTOL,ATOL,ITOL,INFORM,
     +            RESID,YSAVE,NY2DIM,JAC,WKJAC,NWKJAC,JACPVT,NJCPVT,
     +            MONITR,LDERIV,ITASK,ITRACE,IFAIL)
*
      IF (IFAIL.EQ.0 .OR. IFAIL.EQ.12) THEN
         WRITE (NOUT,99999) T, (Y(I),I=1,NEQ)
         IFAIL = 0
*
         CALL D02NYF(NEQ,NEQMAX,HU,H,TCUR,TOLSF,RWORK,NST,NRE,NJE,NQU,
     +               NQ,NITER,IMXER,ALGEQU,INFORM,IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99997) ' HUSED = ', HU, '  HNEXT = ', H,
     +     '  TCUR = ', TCUR
         WRITE (NOUT,99996) ' NST = ', NST, '    NRE = ', NRE,
     +     '    NJE = ', NJE
         WRITE (NOUT,99996) ' NQU = ', NQU, '    NQ  = ', NQ,
     +     '  NITER = ', NITER
         WRITE (NOUT,99995) ' Max err comp = ', IMXER
         WRITE (NOUT,*)
         ICALL = 0
*
         CALL D02NXF(ICALL,LIWREQ,LIWUSD,LRWREQ,LRWUSD,NLU,NNZ,NGP,
     +               ISPLIT,IGROW,LBLOCK,NBLOCK,INFORM)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99994) ' NJCPVT (required ', LIWREQ, '  used ',
     +     LIWUSD, ')'
         WRITE (NOUT,99994) ' NWKJAC (required ', LRWREQ, '  used ',
     +     LRWUSD, ')'
         WRITE (NOUT,99993) ' No. of LU-decomps ', NLU,
     +     '  No. of nonzeros ', NNZ
         WRITE (NOUT,99992) ' No. of FCN calls to form Jacobian ', NGP,
     +     '  Try ISPLIT ', ISPLIT
         WRITE (NOUT,99991) ' Growth est ', IGROW,
     +     '  No. of blocks on diagonal ', NBLOCK
      ELSE IF (IFAIL.EQ.10) THEN
         ICALL = 1
*
         CALL D02NXF(ICALL,LIWREQ,LIWUSD,LRWREQ,LRWUSD,NLU,NNZ,NGP,
     +               ISPLIT,IGROW,LBLOCK,NBLOCK,INFORM)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99994) ' NJCPVT (required ', LIWREQ, '  used ',
     +     LIWUSD, ')'
         WRITE (NOUT,99994) ' NWKJAC (required ', LRWREQ, '  used ',
     +     LRWUSD, ')'
      ELSE
         WRITE (NOUT,*)
         WRITE (NOUT,99998) 'Exit D02NJF with IFAIL = ', IFAIL,
     +     '  and T = ', T
      END IF
      STOP
*
```

```
99999 FORMAT (1X,F8.3,3(F13.5,2X))
99998 FORMAT (1X,A,I2,A,e12.5)
99997 FORMAT (1X,A,e12.5,A,e12.5,A,e12.5)
99996 FORMAT (1X,A,I6,A,I6,A,I6)
99995 FORMAT (1X,A,I4)
99994 FORMAT (1X,A,I8,A,I8,A)
99993 FORMAT (1X,A,I4,A,I8)
99992 FORMAT (1X,A,I4,A,I4)
99991 FORMAT (1X,A,I8,A,I4)
      END
*
      SUBROUTINE RESID(NEQ,T,Y,YDOT,R,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IRES, NEQ
*     .. Array Arguments ..
      real              R(NEQ), Y(NEQ), YDOT(NEQ)
*     .. Executable Statements ..
      R(1) = 0.0e0
      R(2) = -YDOT(2)
      R(3) = -YDOT(3)
      IF (IRES.EQ.1) THEN
         R(1) = Y(1) + Y(2) + Y(3) - 1.0e0 + R(1)
         R(2) = 0.04e0*Y(1) - 1.0e4*Y(2)*Y(3) - 3.0e7*Y(2)*Y(2) + R(2)
         R(3) = 3.0e7*Y(2)*Y(2) + R(3)
      END IF
      RETURN
      END
*
      SUBROUTINE JAC(NEQ,T,Y,YDOT,H,D,J,PDJ)
*     .. Scalar Arguments ..
      real           D, H, T
      INTEGER        J, NEQ
*     .. Array Arguments ..
      real           PDJ(NEQ), Y(NEQ), YDOT(NEQ)
*     .. Local Scalars ..
      real           HXD
*     .. Executable Statements ..
      HXD = H*D
      IF (J.EQ.1) THEN
         PDJ(1) = 0.0e0 - HXD*(1.0e0)
         PDJ(2) = 0.0e0 - HXD*(0.04e0)
*        PDJ(3) = 0.0 - HXD*(0.)
      ELSE IF (J.EQ.2) THEN
         PDJ(1) = 0.0e0 - HXD*(1.0e0)
         PDJ(2) = 1.0e0 - HXD*(-1.0e4*Y(3)-6.0e7*Y(2))
         PDJ(3) = 0.0e0 - HXD*(6.0e7*Y(2))
      ELSE IF (J.EQ.3) THEN
         PDJ(1) = 0.0e0 - HXD*(1.0e0)
         PDJ(2) = 0.0e0 - HXD*(-1.0e4*Y(2))
         PDJ(3) = 1.0e0 - HXD*(0.0e0)
      END IF
      RETURN
      END
*
      SUBROUTINE MONITR(N,NMAX,T,HLAST,H,Y,YDOT,YSAVE,R,ACOR,IMON,INLN,
     +                  HMIN,HMXI,NQU)
*     .. Scalar Arguments ..
      real              H, HLAST, HMIN, HMXI, T
      INTEGER           IMON, INLN, N, NMAX, NQU
*     .. Array Arguments ..
      real              ACOR(NMAX,2), R(N), Y(N), YDOT(N), YSAVE(NMAX,*)
*     .. Executable Statements ..
      IF (Y(1).LE.0.9e0) IMON = -2
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D02NJF Example Program Results

   Analytic Jacobian, structure not supplied

     X          Y(1)            Y(2)            Y(3)
   0.000       1.00000         0.00000         0.00000
   4.993       0.89160         0.00002         0.10838

  HUSED =  0.62790E+00  HNEXT =  0.62790E+00  TCUR =  0.49932E+01
  NST =      50    NRE =    146    NJE =      19
  NQU =       4    NQ  =      4  NITER =     131
  Max err comp =    3

  NJCPVT (required       93  used       150)
  NWKJAC (required       29  used        76)
  No. of LU-decomps   19  No. of nonzeros        7
  No. of FCN calls to form Jacobian    0  Try ISPLIT    73
  Growth est   143336  No. of blocks on diagonal    1

   Analytic Jacobian, structure supplied

     X          Y(1)            Y(2)            Y(3)
   0.000       1.00000         0.00000         0.00000
   4.904       0.89278         0.00002         0.10720

  HUSED =  0.59495E+00  HNEXT =  0.59495E+00  TCUR =  0.49038E+01
  NST =      47    NRE =    131    NJE =      15
  NQU =       4    NQ  =      4  NITER =     118
  Max err comp =    3


  NJCPVT (required       99  used       150)
  NWKJAC (required       31  used        75)
  No. of LU-decomps   15  No. of nonzeros        8
  No. of FCN calls to form Jacobian    0  Try ISPLIT    73
  Growth est     1026  No. of blocks on diagonal    1
```

D02NMF – NAG Fortran Library Routine Document

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

1. Purpose

D02NMF is a reverse communication routine for integrating stiff systems of explicit ordinary differential equations.

2. Specification

```
 SUBROUTINE D02NMF (NEQ, NEQMAX, T, TOUT, Y, YDOT, RWORK, RTOL,
1                   ATOL, ITOL, INFORM, YSAVE, NY2DIM, WKJAC, NWKJAC,
2                   JACPVT, NJCPVT, IMON, INLN, IRES, IREVCM, ITASK,
3                   ITRACE, IFAIL)
 INTEGER        NEQ, NEQMAX, ITOL, INFORM(23), NY2DIM, NWKJAC,
1               JACPVT(NJCPVT), NJCPVT, IMON, INLN, IRES, IREVCM,
2               ITASK, ITRACE, IFAIL
 real           T, TOUT, Y(NEQMAX), YDOT(NEQMAX),
1               RWORK(50+4*NEQMAX), RTOL(*), ATOL(*),
2               YSAVE(NEQMAX,NY2DIM), WKJAC(NWKJAC)
```

3. Description

D02NMF is a general purpose routine for integrating the initial value problem for a stiff system of explicit ordinary differential equations,

$$y' = g(t,y).$$

An outline of a typical calling program is given below:

```
C
C     declarations
C
      call linear algebra setup routine
      call integrator setup routine
      IREVCM=0
 1000 CALL D02NMF(NEQ, NEQMAX, T, TOUT, Y, YDOT, RWORK, RTOL,
     +  ATOL, ITOL, INFORM, YSAVE, NY2DIM, WKJAC, NWKJAC, JACPVT,
     +  NJCPVT, IMON, INLN, IRES, IREVCM, ITASK, ITRACE, IFAIL)
      IF (IREVCM.GT.0) THEN
        IF (IREVCM. EQ. 8) THEN
          supply the Jacobian matrix                             (i)
        ELSE IF(IREVCM.EQ.9) THEN
          perform monitoring tasks requested by the user         (ii)
        ELSE IF(IRECVM.EQ.1.OR.IREVCM.GE.3.AND.IREVCM.LE.5)THEN
          evaluate the derivative                               (iii)
        ELSE IF(IREVCM.EQ.10)THEN
          indicates an unsuccessful step
        ENDIF
        GO TO 1000
      ENDIF
C
C     post processing (optional linear algebra diagnostic call
C     (sparse case only), optional integrator diagnostic call)
C
      STOP
      END
```

There are three major operations that may be required of the (sub)program from which D02NMF is called on an intermeditate return (IREVCM ≠ 0) from D02NMF; these are denoted (i), (ii) and (iii) above.

The following sections describe in greater detail exactly what is required of each of these operations.

(i) Supply the Jacobian Matrix.

The user need only provide this facility if the parameter JCEVAL = 'A' (or 'F' if using sparse matrix linear algebra) in a call to the linear algebra setup routine. If the Jacobian matrix is to be evaluated numerically by the integrator, then the remainder of section (i) can be ignored.

We must define the system of nonlinear equations which is solved internally by the integrator. The time derivative, y', has the form

$$y' = (y-z)/(hd)$$

where h is the current step size and d is a parameter that depends on the integration method in use. The vector y is the current solution and the vector z depends on information from previous time steps. This means that $\frac{d}{dy'}(\ ) = \frac{1}{(hd)}\frac{d}{dy}(\ )$. The system of nonlinear equations that is solved has the form

$$y' - g(t,y) = 0$$

but is solved in the form

$$r(t,y) = 0,$$

where the function r is defined by

$$r(t,y) = (hd)((y-z)/(hd)-g(t,y)).$$

It is the Jacobian matrix $\frac{\partial r}{\partial y}$ that the user must supply as follows:

$$\frac{\partial r_i}{\partial y_j} = 1 - (hd)\frac{\partial g_i}{\partial y_j} \quad \text{if } i = j,$$
$$\frac{\partial r_i}{\partial y_j} = \quad - (hd)\frac{\partial g_i}{\partial y_j} \quad \text{otherwise,}$$

where t, h and d are located in RWORK(19), RWORK(16) and RWORK(20) respectively and the array Y contains the current values of the dependent variables. Only the non-zero elements of the Jacobian need be set, since the locations where it is to be stored are preset to zero.

Hereafter in this document this operation will be referred to as JAC.

(ii) Perform tasks requested by the user.

This operation is essentially a monitoring function and additionally provides the opportunity of changing the current values of Y, HNEXT (the step size that the integrator proposes to take on the next step), HMIN (the minimum step size to be taken on the next step), and HMAX (the maximum step size to be taken on the next step). The scaled local error at the end of a timestep may be obtained by calling ***real*** function D02ZAF as follows:

```
      IFAIL = 1
      ERRLOC = D02ZAF(NEQ,RWORK(51+NEQMAX),RWORK(51),IFAIL)
C     CHECK IFAIL BEFORE PROCEEDING
```

The following gives details of the location within the array RWORK of variables that may be of interest to the user:

Variable	Specification	Location
TCURR	the current value of the independent variable	RWORK(19)
HLAST	last step size successfully used by the integrator	RWORK(15)
HNEXT	step size that the integrator proposes to take on the next step	RWORK(16)
HMIN	minimum step size to be taken on the next step	RWORK(17)
HMAX	maximum step size to be taken on the next step	RWORK(18)
NQU	the order of the integrator used on the last step	RWORK(10)

Users are advised to consult the description of MONITR in routine document D02NBF for details on what optional input can be made.

If Y is changed, then IMON must be set to 2 before return to D02NMF. If either of the values of HMIN or HMAX are changed, then IMON must be set ≥ 3 before return to D02NMF. If HNEXT is changed, then IMON must be set to 4 before return to D02NMF.

In addition the user can force D02NMF to evaluate the residual vector

$$y' - g(t,y)$$

be setting IMON = 0 and INLN = 3 and then returning to D02NMF; on return to this monitoring operation the residual vector will be stored in RWORK(50+2×NEQMAX+i), for i = 1,2...,NEQ.

Hereafter in this document this operation will be referred to as MONITR.

(iii) Evaluate the derivative.

This operation must evaluate the derivative vector for the explicit ordinary differential equation system defined by

$$y' = g(t,y)$$

where t is located in RWORK(19).

Hereafter in this document this operation will be referred to as FCN.

4. References

See Subchapter Introduction.

5. Parameters

Note: this routine uses **reverse communication.** Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the **parameter IREVCM**. Between intermediate exits and re-entries, **all parameters other than IMON, INLN, IRES, RWORK, YDOT and WKJAC must remain unchanged.**

1: NEQ – INTEGER. *Input*

On initial entry: the number of differential equations to be solved.

Constraint: NEQ ≥ 1.

2: NEQMAX – INTEGER. *Input*

On initial entry: an upper bound on the maximum number of differential equations to be solved during the integration.

Constraint: NEQMAX ≥ NEQ.

3: T – ***real.*** *Input/Output*

On initial entry: the value of the independent variable t. The input value of T is used only on the first call as the initial point of the integration.

On final exit: the value at which the computed solution y is returned (usually at TOUT).

4: TOUT – ***real.*** *Input*

On initial entry: the next value of t at which a computed solution is desired. For the initial t, an input value of TOUT is used to determine the direction of integration. Integration is permitted in either direction (see also ITASK).

Constraint: TOUT ≠ T.

5: Y(NEQMAX) – ***real*** array. *Input/Output*

On initial entry: the values of dependent variables (solution). On the first call the first NEQ elements of y must contain the vector of initial values.

On final exit: the computed solution vector, evaluated at T (usually T = TOUT).

6: YDOT(NEQMAX) – ***real*** array. *Input/Output*

On intermediate re-entry: YDOT must be set to the derivatives as defined under the description of IREVCM.

On final exit: the time derivatives y' of the vector y at the last integration point.

7: RWORK(50+4*NEQMAX) – ***real*** array. *Input/Output*

On intermediate re-entry: elements of RWORK must be set to quantities as defined under the description of IREVCM.

On intermediate exit: contains information for JAC, FCN and MONITR operations as described in Section 3 and the parameter IREVCM.

8: RTOL(*) – ***real*** array. *Input*

Note: the dimension of the array RTOL must be at least 1 or NEQ (see ITOL).

On initial entry: the relative local error tolerance.

Constraint: RTOL(i) $\geq$ 0.0 for all relevant i (see ITOL).

9: ATOL(*) – ***real*** array. *Input*

Note: the dimension of the array ATOL must be at least 1 or NEQ (see ITOL).

On initial entry: the absolute local error tolerance.

Constraint: ATOL(i) $\geq$ 0.0 for all relevant i (see ITOL).

10: ITOL – INTEGER. *Input*

On initial entry: a value to indicate the form of the local error test. ITOL indicates to D02NMF whether to interpret either or both of RTOL or ATOL as a vector or a scalar. The error test to be satisfied is $\|e_i/w_i\| < 1.0$, where w_i is defined as follows:

ITOL	RTOL	ATOL	w_i
1	scalar	scalar	$\mathrm{RTOL}(1)\times\lvert y_i\rvert + \mathrm{ATOL}(1)$
2	scalar	vector	$\mathrm{RTOL}(1)\times\lvert y_i\rvert + \mathrm{ATOL}(i)$
3	vector	scalar	$\mathrm{RTOL}(i)\times\lvert y_i\rvert + \mathrm{ATOL}(1)$
4	vector	vector	$\mathrm{RTOL}(i)\times\lvert y_i\rvert + \mathrm{ATOL}(i)$

e_i is an estimate of the local error in y_i, computed internally, and the choice of norm to be used is defined by a previous call to an integrator setup routine.

Constraint: $1 \leq$ ITOL ≤ 4.

11: INFORM(23) – INTEGER array. *Workspace*

12: YSAVE(NEQMAX,NY2DIM) – ***real*** array. *Workspace*

13: NY2DIM – INTEGER. *Input*

On initial entry: the second dimension of the array YSAVE as declared in the (sub)program from which D02NMF is called. An appropriate value for NY2DIM is described in the specifications of the integrator setup routines D02NVF and D02NWF. This value must be the same as that supplied to the integrator setup routine.

14: WKJAC(NWKJAC) – ***real*** array. *Input/Output*

On intermediate re-entry: elements of the Jacobian as defined under the description of IREVCM. If a numerical Jacobian was requested then WKJAC is used for workspace.

On intermediate exit: the Jacobian is overwritten.

15: NWKJAC – INTEGER. *Input*

On initial entry: the dimension of the array WKJAC as declared in the (sub)program from which D02NMF is called. The actual size depends on the linear algebra method used. An appropriate value for NWKJAC is described in the specifications of the linear algebra setup routines D02NSF, D02NTF and D02NUF for full, banded and sparse matrix linear algebra respectively. This value must be the same as that supplied to the linear algebra setup routine.

16: JACPVT(NJCPVT) – INTEGER array. *Workspace*

17: NJCPVT – INTEGER. *Input*

On initial entry: the dimension of the array JACPVT as declared in the (sub)program from which D02NMF is called. The actual size depends on the linear algebra method used. An appropriate value for NJCPVT is described in the specifications of the linear algebra setup routines D02NTF and D02NUF for banded and sparse matrix linear algebra respectively. This value must be the same as that supplied to the linear algebra setup routine. When full matrix linear algebra is chosen, the array JACPVT is not used and hence NJCPVT should be set to 1.

18: IMON – INTEGER. *Input/Output*

On intermediate exit: used to pass information between D02NMF and the MONITR operation (see Section 3). With IREVCM = 9, IMON contains a flag indicating under what circumstances the return from D02NMF occurred.

IMON = –2

Exit from D02NMF after IRES = 4 (set in the FCN operation (see Section 3)) caused an early termination (this facility could be used to locate discontinuities).

IMON = –1

The current step failed repeatedly.

IMON = 0

Exit from D02NMF after a call to the internal nonlinear equation solver.

IMON = 1

The current step was successful.

On intermediate re-entry: IMON may be reset to determine subsequent action in D02NMF.

IMON = –2

Integration is to be halted. A return will be made from D02NMF to the (sub)program from which D02NMF is called with IFAIL = 12.

IMON = –1

Allow D02NMF to continue with its own internal strategy. The integrator will try up to 3 restarts unless IMON is set ≠ –1.

IMON = 0

Return to the internal nonlinear equation solver, where the action taken is determined by the value of INLN (see below).

IMON = 1

Normal return to D02NMF to continue integration.

IMON = 2

Restart the integration at the current time point. The integrator will restart from order 1 when this option is used. The solution Y, provided by the MONITR operation (see Section 3), will be used for the initial conditions.

IMON = 3

Try to continue with the same step size and order as was to be used before entering the MONITR operation (see Section 3). HMIN and HMAX may be altered if desired.

IMON = 4

Continue the integration but using a new value HNEXT and possibly new values of HMIN and HMAX.

19: INLN – INTEGER. *Input*

On intermediate re-entry: with IMON = 0 and IREVCM = 9, INLN specifies the action to be taken by the internal nonlinear equation solver. By setting INLN = 3 and returning to D02NMF, the residual vector is evaluated and placed in RWORK(50+2×NEQMAX+i), for i = 1,2,...,NEQ, and then the MONITR operation (see Section 3) is invoked again. At present this is the only option available: INLN must not be set to any other value.

20: IRES – INTEGER. *Input/Output*

On intermediate exit: with IREVCM = 1, 2, 3, 4 or 5, IRES contains the value 1.

On intermediate re-entry: IRES should be unchanged unless one of the following actions is required of D02NMF in which case IRES should be set accordingly.

IRES = 2

indicates to D02NMF that control should be passed back immediately to the (sub)program from which D02NMF is called with the error indicator set to IFAIL = 11.

IRES = 3

indicates to D02NMF that an error condition has occurred in the solution vector, its time derivative or in the value of t. The integrator will use a smaller time step to try to avoid this condition. If this is not possible D02NMF returns to the (sub)program from which D02NMF is called with the error indicator set to IFAIL = 7.

IRES = 4

indicates to D02NMF to stop its current operation and to enter the MONITR operation (see Section 3) immediately.

21: IREVCM – INTEGER. *Input/Output*

On initial entry: IREVCM must contain 0.

On intermediate re-entry: should remain unchanged.

On intermediate exit: indicates what action the user must take before re-entering. The possible exit values of IREVCM are 1, 3, 4, 5, 8, 9, 10, which should be interpreted as follows:

IREVCM = 1, 3, 4 and 5

indicates that an FCN operation (see Section 3) is required: $y' = g(t,y)$ must be supplied, where Y(i) is located in y_i, for i = 1,2,...,NEQ.

For IREVCM = 1 or 3, y_i' should be placed in location RWORK(50+2×NEQMAX+i), for i = 1,2,...,NEQ.

For IREVCM = 4, y_i' should be placed in location RWORK(50+NEQMAX+i), for i = 1,2,...,NEQ.

For IREVCM = 5, y_i' should be placed in location YDOT(i), for i = 1,2,...,NEQ.

IREVCM = 8

indicates that a JAC operation (see Section 3) is required: the Jacobian matrix must be supplied.

If full matrix linear algebra is being used, then the (i,j)th element of the Jacobian must be stored in WKJAC($(j-1)\times$NEQ$+i$).

If banded matrix linear algebra is being used then the (i,j)th element of the Jacobian must be stored in WKJAC($(i-1)\times m_B+k$), where $m_B = m_L+m_U+1$ and $k = \min(m_L-i+1,0) + j$; here m_L and m_U are the number of sub-diagonals and super-diagonals, respectively, in the band.

If sparse matrix linear algebra is being used then D02NRF must be called to determine which column of the Jacobian is required and where it should be stored.

```
CALL D02NRF(J,IPLACE,INFORM)
```

will return in J the number of the column of the Jacobian that is required and will set IPLACE = 1 or 2. If IPLACE = 1, then the (i,j)th element of the Jacobian must be stored in RWORK(50+2×NEQMAX$+i$); otherwise it must be stored in RWORK(50+NEQMAX$+i$).

IREVCM = 9

indicates that a MONITR operation (see Section 3) can be performed.

IREVCM = 10

indicates that the current step was not successful, due to error test failure or convergence test failure. The only information supplied to the user on this return is the current value of the independent variable t, located in RWORK(19). No values must be changed before re-entering D02NMF; this facility enables the user to determine the number of unsuccessful steps.

On final exit: IREVCM = 0 indicated the user-specified task has been completed or an error has been encountered (see descriptions for ITASK and IFAIL).

Constraint: IREVCM = 0, 1, 3, 4, 5, 8, 9, 10.

22: ITASK – INTEGER. *Input*

On initial entry: the task to be performed by the integrator. The permitted values for ITASK and their meanings are detailed below:

ITASK = 1

normal computation of output values of $y(t)$ at t = TOUT (by overshooting and interpolating).

ITASK = 2

take one step only and return.

ITASK = 3

stop at the first internal integration point at or beyond t = TOUT and return.

ITASK = 4

normal computation of output values of $y(t)$ at t = TOUT but without overshooting t = TCRIT. TCRIT must be specified as an option in one of the integrator setup routines prior to the first call to the integrator, or specified in the optional input routine prior to a continuation call. TCRIT may be equal to or beyond TOUT, but not before it in the direction of integration.

ITASK = 5

take one step only and return, without passing TCRIT. TCRIT must be specified as under ITASK = 4.

Constraint: 1 ≤ ITASK ≤ 5.

23: ITRACE – INTEGER. *Input*

On initial entry: the level of output that is printed by the integrator. ITRACE may take the value –1, 0, 1, 2 or 3. If ITRACE < –1, then –1 is assumed and similarly if ITRACE > 3, then 3 is assumed. If ITRACE = –1, no output is generated. If ITRACE = 0, only warning messages are printed on the current error message unit (see X04AAF). If ITRACE > 0, then warning messages are printed as above, and on the current advisory message unit (see X04ABF) output is generated which details Jacobian entries, the nonlinear iteration and the time integration. The advisory messages are given in greater detail the larger the value of ITRACE.

24: IFAIL – INTEGER. *Input/Output*

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

On final exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.** To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL = 1

On entry, the integrator detected an illegal input, or that a linear algebra and/or integrator setup routine has not been called prior to the call to the integrator. If ITRACE ≥ 0, the form of the error will be detailed on the current error message unit (see X04AAF).

IFAIL = 2

The maximum number of steps specified has been taken (see the description of optional inputs in the integrator setup routines and the optional input continuation routine, D02NZF).

IFAIL = 3

With the given values of RTOL and ATOL no further progress can be made across the integration range from the current point T. The components Y(1),Y(2),...,Y(NEQ) contain the computed values of the solution at the current point T.

IFAIL = 4

There were repeated error test failures on an attempted step, before completing the requested task, but the integration was successful as far as T. The problem may have a singularity, or the local error requirements may be inappropriate.

IFAIL = 5

There were repeated convergence test failures on an attempted step, before completing the requested task, but the integration was successful as far as T. This may be caused by an inaccurate Jacobian matrix or one which is incorrectly computed.

IFAIL = 6

Some error weight w_i became zero during the integration (see description of ITOL). Pure relative error control (ATOL(i) = 0.0) was requested on a variable (the i(th)) which has now vanished. The integration was successful as far as T.

IFAIL = 7

The FCN operation, (see Section 3), set the error flag IRES = 3 continually despite repeated attempts by the integrator to avoid this.

IFAIL = 8

Not used for this integrator.

IFAIL = 9

A singular Jacobian $\frac{\partial r}{\partial y}$ has been encountered. This error exit is unlikely to be taken when solving explicit ordinary differential equations. The user should check his problem formulation and Jacobian calculation.

IFAIL = 10

An error occurred during Jacobian formulation or back-substitution (a more detailed error description may be directed to the current error message unit, see X04AAF).

IFAIL = 11

The FCN operation, (see Section 3), signalled the integrator to halt the integration and return by setting IRES = 2. Integration was successful as far as T.

IFAIL = 12

The MONITR operation, (see Section 3), set IMON = –2 and so forced a return but the integration was successful as far as T.

IFAIL = 13

The requested task has been completed, but it is estimated that a small change in RTOL and ATOL is unlikely to produce any change in the computed solution. (Only applies when the user is not operating in one step mode, that is when ITASK ≠ 2 or 5).

IFAIL = 14

The values of RTOL and ATOL are so small that the routine is unable to start the integration.

7. Accuracy

The accuracy of the numerical solution may be controlled by a careful choice of the parameters RTOL and ATOL, and to a much lesser extent by the choice of norm. Users are advised to use scalar error control unless the components of the solution are expected to be poorly scaled. For the type of decaying solution typical of many stiff problems, relative error control with a small absolute error threshold will be most appropriate (that is the user is advised to choose ITOL = 1 with ATOL(1) small but positive).

8. Further Comments

The cost of computing a solution depends critically on the size of the differential system and to a lesser extent on the degree of stiffness of the problem; also on the type of linear algebra being used. For further details see Section 8 of the documents for D02NBF (full matrix), D02NCF (banded matrix) or D02NDF (sparse matrix).

In general the user is advised to choose the backward differentiation formula option (setup routine D02NVF) but if efficiency is of great importance and especially if it is suspected that $\frac{\partial g}{\partial y}$ has complex eigenvalues near the imaginary axis for some part of the integration, the user should try the BLEND option (setup routine D02NWF).

9. Example

We solve the well-known stiff Robertson problem

$$\begin{aligned} a' &= -0.04a + 1.0\mathrm{E}4bc \\ b' &= 0.04a - 1.0\mathrm{E}4bc - 3.0\mathrm{E}7b^2 \\ c' &= \phantom{-0.04a - 1.0\mathrm{E}4bc - {}} 3.0\mathrm{E}7b^2 \end{aligned}$$

over the range [0,10] with initial conditions $a = 1.0$ and $b = c = 0.0$ and with scalar error control (ITOL = 1). We integrate until we pass TOUT = 10.0 providing C^1 interpolation at intervals of 2.0 through a MONITR operation. The integration method used is the BDF method (setup routine D02NVF) with a modified Newton method. We specify that the Jacobian is a full matrix (setup routine D02NSF) and is to be calculated numerically.

9.1. Program Text

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

```
*     D02NMF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, NEQMAX, NRW, NINF, NWKJAC, NJCPVT, MAXORD,
     +                 NY2DIM, MAXSTP, MXHNIL
      PARAMETER        (NEQ=3,NEQMAX=NEQ,NRW=50+4*NEQMAX,NINF=23,
     +                 NWKJAC=NEQMAX*(NEQMAX+1),NJCPVT=1,MAXORD=5,
     +                 NY2DIM=MAXORD+1,MAXSTP=200,MXHNIL=5)
      INTEGER          LACORB, LSAVRB
      PARAMETER        (LACORB=50+NEQMAX,LSAVRB=LACORB+NEQMAX)
      real             H0, HMAX, HMIN, TCRIT
      PARAMETER        (H0=0.0e0,HMAX=10.0e0,HMIN=1.0e-10,TCRIT=0.0e0)
      LOGICAL          PETZLD
      PARAMETER        (PETZLD=.FALSE.)
*     .. Local Scalars ..
      real             H, HLAST, HNEXT, HU, T, TC, TCUR, TOLSF, TOUT,
     +                 XOUT
      INTEGER          I, IFAIL, IFLAG, IMON, IMXER, INLN, IOUT, IRES,
     +                 IREVCM, ITASK, ITOL, ITRACE, LACOR1, LACOR2,
     +                 LACOR3, LSAVR1, LSAVR2, LSAVR3, NITER, NJE, NQ,
     +                 NQU, NRE, NST
*     .. Local Arrays ..
      real             ATOL(NEQMAX), CONST(6), RTOL(NEQMAX), RWORK(NRW),
     +                 WKJAC(NWKJAC), Y(NEQMAX), YDOT(NEQMAX),
     +                 YSAVE(NEQMAX,NY2DIM).
      INTEGER          INFORM(NINF), JACPVT(NJCPVT)
      LOGICAL          ALGEQU(NEQMAX)
*     .. External Subroutines ..
      EXTERNAL         D02NMF, D02NSF, D02NVF, D02NYF, D02XKF, X04ABF
*     .. Intrinsic Functions ..
      INTRINSIC        INT, real
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02NMF Example Program Results'
      WRITE (NOUT,*)
      CALL X04ABF(1,NOUT)
*
```

```
*     Integrate to TOUT by overshooting TOUT (ITASK=1) using B.D.F.
*     formulae with a Newton method. Default values for the array CONST
*     are used. Employ scalar tolerances and the Jacobian is evaluated
*     internally. On the reverse communication call equivalent to the
*     MONITR call in forward communication routines carry out
*     interpolation using D02XKF.
*
      T = 0.0e0
      TOUT = 10.0e0
      ITASK = 1
      IOUT = 1
      XOUT = 2.0e0
      Y(1) = 1.0e0
      Y(2) = 0.0e0
      Y(3) = 0.0e0
      ITOL = 1
      RTOL(1) = 1.0e-4
      ATOL(1) = 1.0e-7
      DO 20 I = 1, 6
         CONST(I) = 0.0e0
   20 CONTINUE
      IFAIL = 0
*
      CALL D02NVF(NEQMAX,NY2DIM,MAXORD,'Newton',PETZLD,CONST,TCRIT,HMIN,
     +            HMAX,H0,MAXSTP,MXHNIL,'Average-L2',RWORK,IFAIL)
      CALL D02NSF(NEQ,NEQMAX,'Numerical',NWKJAC,RWORK,IFAIL)
*
      LACOR1 = LACORB + 1
      LACOR2 = LACORB + 2
      LACOR3 = LACORB + 3
      LSAVR1 = LSAVRB + 1
      LSAVR2 = LSAVRB + 2
      LSAVR3 = LSAVRB + 3
      WRITE (NOUT,*) '     X          Y(1)           Y(2)           Y(3)'
      WRITE (NOUT,99999) T, (Y(I),I=1,NEQ)
*
*     Soft fail and error messages only
      IREVCM = 0
      ITRACE = 0
   40 IFAIL = 1
*
      CALL D02NMF(NEQ,NEQMAX,T,TOUT,Y,YDOT,RWORK,RTOL,ATOL,ITOL,INFORM,
     +            YSAVE,NY2DIM,WKJAC,NWKJAC,JACPVT,NJCPVT,IMON,INLN,
     +            IRES,IREVCM,ITASK,ITRACE,IFAIL)
*
      IF (IREVCM.NE.0) THEN
         IF (IREVCM.EQ.1 .OR. IREVCM.EQ.3) THEN
*           Equivalent to FCN evaluation in forward communication
*           routines
            RWORK(LSAVR1) = -0.04e0*Y(1) + 1.0e4*Y(2)*Y(3)
            RWORK(LSAVR2) = 0.04e0*Y(1) - 1.0e4*Y(2)*Y(3) - 3.0e7*Y(2)
     +                      *Y(2)
            RWORK(LSAVR3) = 3.0e7*Y(2)*Y(2)
         ELSE IF (IREVCM.EQ.4) THEN
*           Equivalent to FCN evaluation in forward communication
*           routines
            RWORK(LACOR1) = -0.04e0*Y(1) + 1.0e4*Y(2)*Y(3)
            RWORK(LACOR2) = 0.04e0*Y(1) - 1.0e4*Y(2)*Y(3) - 3.0e7*Y(2)
     +                      *Y(2)
            RWORK(LACOR3) = 3.0e7*Y(2)*Y(2)
         ELSE IF (IREVCM.EQ.5) THEN
*           Equivalent to FCN evaluation in forward communication
*           routines
            YDOT(1) = -0.04e0*Y(1) + 1.0e4*Y(2)*Y(3)
            YDOT(2) = 0.04e0*Y(1) - 1.0e4*Y(2)*Y(3) - 3.0e7*Y(2)*Y(2)
            YDOT(3) = 3.0e7*Y(2)*Y(2)
         ELSE IF (IREVCM.EQ.9) THEN
```

```
*           Equivalent to MONITR call in forward communication routines
            IF (IMON.EQ.1) THEN
               TC = RWORK(19)
               HLAST = RWORK(15)
               HNEXT = RWORK(16)
               NQU = INT(RWORK(10))
   60          CONTINUE
               IF (TC-HLAST.LT.XOUT .AND. XOUT.LE.TC) THEN
                  IFLAG = 1
*
                  CALL D02XKF(XOUT,RWORK(LSAVR1),NEQ,YSAVE,NEQMAX,
     +                        NY2DIM,RWORK(LACOR1),NEQ,TC,NQU,HLAST,
     +                        HNEXT,IFLAG)
*
                  IF (IFLAG.NE.0) THEN
                     IMON = -2
                  ELSE
                     WRITE (NOUT,99999) XOUT, (RWORK(LSAVRB+I),I=1,NEQ)
                     IOUT = IOUT + 1
                     XOUT = real(IOUT)*2.0e0
                     IF (IOUT.LT.6) GO TO 60
                  END IF
               END IF
            END IF
         ELSE IF (IREVCM.EQ.2 .OR. IREVCM.EQ.6 .OR. IREVCM.EQ.7 .OR.
     +            IREVCM.EQ.8) THEN
            WRITE (NOUT,*)
            WRITE (NOUT,99995) 'Illegal value of IREVCM = ', IREVCM
            STOP
         END IF
         GO TO 40
      ELSE
         IF (IFAIL.EQ.0) THEN
*
            CALL D02NYF(NEQ,NEQMAX,HU,H,TCUR,TOLSF,RWORK,NST,NRE,NJE,
     +                  NQU,NQ,NITER,IMXER,ALGEQU,INFORM,IFAIL)
*
            WRITE (NOUT,*)
            WRITE (NOUT,99997) ' HUSED = ', HU, '  HNEXT = ', H,
     +        '  TCUR = ', TCUR
            WRITE (NOUT,99996) ' NST = ', NST, '    NRE = ', NRE,
     +        '    NJE = ', NJE
            WRITE (NOUT,99996) ' NQU = ', NQU, '    NQ  = ', NQ,
     +        '  NITER = ', NITER
            WRITE (NOUT,99995) ' Max err comp = ', IMXER
            WRITE (NOUT,*)
         ELSE
            WRITE (NOUT,*)
            WRITE (NOUT,99998) 'Exit D02NMF with IFAIL = ', IFAIL,
     +        '  and T = ', T
         END IF
      END IF
      STOP
*
99999 FORMAT (1X,F8.3,3(F13.5,2X))
99998 FORMAT (1X,A,I2,A,e12.5)
99997 FORMAT (1X,A,e12.5,A,e12.5,A,e12.5)
99996 FORMAT (1X,A,I6,A,I6,A,I6)
99995 FORMAT (1X,A,I4)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D02NMF Example Program Results

     X           Y(1)          Y(2)          Y(3)
    0.000       1.00000       0.00000       0.00000
    2.000       0.94161       0.00003       0.05836
    4.000       0.90551       0.00002       0.09446
    6.000       0.87926       0.00002       0.12072
    8.000       0.85854       0.00002       0.14144
   10.000       0.84136       0.00002       0.15863

  HUSED =  0.90178E+00  HNEXT =  0.90178E+00  TCUR =  0.10766E+02
  NST =      55    NRE =    128    NJE =     16
  NQU =       4    NQ  =      4  NITER =     78
  Max err comp =    3
```

D02NNF – NAG Fortran Library Routine Document

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

1 Purpose

D02NNF is a reverse communication routine for integrating stiff systems of implicit ordinary differential equations coupled with algebraic equations.

2 Specification

```
 SUBROUTINE D02NNF(NEQ, NEQMAX, T, TOUT, Y, YDOT, RWORK, RTOL,
1                  ATOL, ITOL, INFORM, YSAVE, NY2DIM, WKJAC,
2                  NWKJAC, JACPVT, NJCPVT, IMON, INLN, IRES,
3                  IREVCM, LDERIV, ITASK, ITRACE, IFAIL)
 INTEGER           NEQ, NEQMAX, ITOL, INFORM(23), NY2DIM, NWKJAC,
1                  JACPVT(NJCPVT), NJCPVT, IMON, INLN, IRES,
2                  IREVCM, ITASK, ITRACE, IFAIL
 real              T, TOUT, Y(NEQMAX), YDOT(NEQMAX),
1                  RWORK(50+4*NEQMAX), RTOL(*), ATOL(*),
2                  YSAVE(NEQMAX,NY2DIM), WKJAC(NWKJAC)
 LOGICAL           LDERIV(2)
```

3 Description

D02NNF is a general purpose routine for integrating the initial value problem for a stiff system of implicit ordinary differential equations coupled with algebraic equations, written in the form,

$$A(t,y)y' = g(t,y)$$

An outline of a typical calling program is given below:

```
C
C     declarations
C
      call linear algebra setup routine
      call integrator setup routine
      IREVCM=0
 1000 CALL D02NNF(NEQ, NEQMAX, T, TOUT, Y, YDOT, RWORK, RTOL,
       ATOL, ITOL, INFORM, YSAVE, NY2DIM, WKJAC, NWKJAC, JACPVT,
       NJCPVT, IMON, INLN, IRES, IREVCM, LDERIV,
       ITASK, ITRACE, IFAIL)

      IF (IREVCM.GT.0) THEN
        IF (IREVCM.GT.7 .AND. IREVCM.LT.11) THEN
          IF (IREVCM.EQ.8) THEN
            supply the Jacobian matrix                      (i)
          ELSE IF (IREVCM.EQ.9) THEN
            perform monitoring tasks requested by the user   (ii)
          ELSE IF (IREVCM.EQ.10) THEN
            indicates an unsuccessful step
          END IF
        ELSE
          evaluate the residual                             (iii)
        ENDIF
        GO TO 1000
      END IF
C
C     post processing (optional linear algebra diagnostic call
C     (sparse case only), optional integrator diagnostic call)
C
      STOP
      END
```

There are three major operations that may be required of the calling (sub)program on an intermediate return (IREVCM $\neq 0$) from D02NNF; these are denoted (i), (ii) and (iii) above.

The following sections describe in greater detail exactly what is required of each of these operations.

(i) Supply the Jacobian matrix.

The user need only provide this facility if the parameter JCEVAL = 'A' (or 'F' if using sparse matrix linear algebra) in a call to the linear algebra setup routine. If the Jacobian matrix is to be evaluated numerically by the integrator, then the remainder of section (i) can be ignored.

We must define the system of nonlinear equations which is solved internally by the integrator, the time derivative, y', has the form

$$y' = (y - z)/(hd)$$

where h is the current step size and d is a parameter that depends on the integration method in use. The vector y is the current solution and the vector z depends on information from previous time steps. This means that $\frac{d}{dy'}() = \frac{1}{(hd)}\frac{d}{dy}()$. The system of nonlinear equations that is solved has the form

$$A(t,y)y' - g(t,y) = 0$$

but is solved in the form

$$r(t,y) = 0$$

where r is the function defined by

$$r(t,y) = (hd)(A(t,y)(y-z)/(hd) - g(t,y)).$$

It is the Jacobian matrix $\frac{\partial r}{\partial y}$ that the user must supply as follows:

$$\frac{\partial r_i}{\partial y_j} = a_{ij}(t,y) + hd\frac{\partial}{\partial y_j}\left(\sum_{k=1}^{\mathrm{NEQ}} a_{ik}(t,y)y_k' - g_i(t,y)\right)$$

where t, h and d are located in RWORK(19), RWORK(16) and RWORK(20) respectively and the arrays Y and YDOT contain the current solution and time derivatives respectively. Only the non-zero elements of the Jacobian need be set, since the locations where it is to be stored are preset to zero.

Hereafter in this document this operation will be referred to as JAC.

(ii) Perform tasks requested by the user.

This operation is essentially a monitoring function and additionally provides the opportunity of changing the current values of Y, YDOT, HNEXT (the step size that the integrator proposes to take on the next step), HMIN (the mimimum step size to be taken on the next step), and HMAX (the maximum step size to be taken on the next step). The scaled local error at the end of a time step may be obtained by calling the ***real*** function D02ZAF as follows:

```
      IFAIL = 1
      ERRLOC = D02ZAF(NEQ,RWORK(51+NEQMAX),RWORK(51),IFAIL)
C     CHECK IFAIL BEFORE PROCEEDING
```

The following gives details of the location within the array RWORK of variables that may be of interest to the user:

Variable	**Specification**	**Location**
TCURR	the current value of the independent variable	RWORK(19)
HLAST	last step size successfully used by the integrator	RWORK(15)
HNEXT	step size that the integrator proposes to take on the next step	RWORK(16)
HMIN	minimum step size to be taken on the next step	RWORK(17)
HMAX	maximum step size to be taken on the next step	RWORK(18)
NQU	the order of the integrator used on the last step	RWORK(10)

Users are advised to consult the description of MONITR in D02NGF for details on what optional input can be made.

If either Y or YDOT are changed, then IMON must be set to 2 before return to D02NNF. If either of the values HMIN or HMAX are changed, then IMON must be set ≥ 3 before return to D02NNF. If HNEXT is changed, then IMON must be set to 4 before return to D02NNF.

In addition the user can force D02NNF to evaluate the residual vector

$$A(t,y)y' - g(t,y)$$

by setting IMON $= 0$ and INLN $= 3$ and then returning to D02NNF; on return to this monitoring operation the residual vector will be stored in RWORK(50+2×NEQMAX+i), for $i = 1,2,\ldots$,NEQ.

Hereafter in this document this operation will be referred to as MONITR.

(iii) Evaluate the residual.

This operation must evaluate the residual

$$r = g(t,y) - A(t,y)y' \quad (1)$$

in one case and

$$r = -A(t,y)y' \quad (2)$$

in another, where t is located in RWORK(19). The form of the residual that is returned is determined by the value of IRES returned by D02NNF. If IRES $= -1$, then the residual defined by equation (2) above must be returned; if IRES $= 1$, then the residual returned by equation (1) above must be returned.

Hereafter in this document this operation will be referred to as RESID.

4 References

None.

5 Parameters

Note: this routine uses **reverse communication.** Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the **parameter IREVCM**. Between intermediate exits and re-entries, **all parameters other than YDOT, RWORK, WKJAC, IMON, INLN and IRES must remain unchanged.**

1: NEQ — INTEGER *Input*

On initial entry: the number of equations to be solved.

Constraint: NEQ ≥ 1.

2: NEQMAX — INTEGER *Input*

On initial entry: a bound on the maximum number of equations to be solved during the integration.

Constraint: NEQMAX $\geq$ NEQ.

3: T — ***real*** *Input/Output*

On initial entry: the value of the independent variable t. The input value of T is used only on the first call as the initial point of the integration.

On final exit: the value at which the computed solution y is returned (usually at TOUT).

4: TOUT — ***real*** *Input*

On initial entry: the next value of t at which a computed solution is desired. For the initial t, the input value of TOUT is used to determine the direction of integration. Integration is permitted in either direction (see also ITASK).

Constraint: TOUT $\neq$ T.

5: Y(NEQMAX) — ***real*** array *Input/Output*

On initial entry: the values of the dependent variables (solution). On the first call the first NEQ elements of y must contain the vector of initial values.

On final exit: the computed solution vector evaluated at T (usually t = TOUT).

6: YDOT(NEQMAX) — ***real*** array *Input/Output*

On initial entry: if LDERIV(1) = .TRUE., YDOT must contain approximations to the time derivatives y' of the vector y. If LDERIV(1) = .FALSE., then YDOT need not be set on entry.

On final exit: contains the time derivatives y' of the vector y at the last integration point.

7: RWORK(50+4*NEQMAX) — ***real*** array *Input/Output*

On intermediate re-entry: must contain residual evaluations as described under the parameter IREVCM.

On intermediate exit: contains information for JAC, RESID and MONITR operations as described under Section 3 and the parameter IREVCM.

8: RTOL(*) — ***real*** array *Input*

Note: the dimension of the array RTOL must be at least 1 or NEQ (see ITOL).

On initial entry: the relative local error tolerance.

Constraint: RTOL(i) $\geq$ 0.0 for all relevant i (see ITOL).

9: ATOL(*) — ***real*** array *Input*

Note: the dimension of the array ATOL must be at least 1 or NEQ (see ITOL).

On initial entry: the absolute local error tolerance.

Constraint: ATOL(i) $\geq$ 0.0 for all relevant i (see ITOL).

10: ITOL — INTEGER *Input*

On initial entry: a value to indicate the form of the local error test. ITOL indicates to D02NNF whether to interpret either or both of RTOL or ATOL as a vector or a scalar. The error test to be satisfied is $\|e_i/w_i\| < 1.0$, where w_i is defined as follows:

ITOL	RTOL	ATOL	w_i
1	scalar	scalar	RTOL(1) × $\lvert y_i\rvert$ + ATOL(1)
2	scalar	vector	RTOL(1) × $\lvert y_i\rvert$ + ATOL(i)
3	vector	scalar	RTOL(i) × $\lvert y_i\rvert$ + ATOL(1)
4	vector	vector	RTOL(i) × $\lvert y_i\rvert$ + ATOL(i)

e_i is an estimate of the local error in y_i, computed internally, and the choice of norm to be used is defined by a previous call to an integrator setup routine.

Constraint: 1 $\leq$ ITOL $\leq$ 4.

11: INFORM(23) — INTEGER array *Workspace*

12: YSAVE(NEQMAX,NY2DIM) — ***real*** array *Workspace*

13: NY2DIM — INTEGER *Input*

On initial entry: the second dimension of the array YSAVE as declared in the (sub)program from which D02NNF is called. An appropriate value for NY2DIM is described in the specifications of the integrator setup routines D02MVF, D02NVF and D02NWF. This value must be the same as that supplied to the integrator setup routine.

14: WKJAC(NWKJAC) — ***real*** array *Input/Output*

On intermediate re-entry: elements of the Jacobian as defined under the description of IREVCM. If a numerical Jacobian was requested then WKJAC is used for workspace.

On intermediate exit: the Jacobian is overwritten.

15: NWKJAC — INTEGER *Input*

On initial entry: the dimension of the array WKJAC as declared in the (sub)program from which D02NNF is called. The actual size depends on the linear algebra method used. An appropriate value for NWKJAC is described in the specifications of the linear algebra setup routines D02NSF, D02NTF and D02NUF for full, banded and sparse matrix linear algebra respectively. This value must be the same as that supplied to the linear algebra setup routine.

16: JACPVT(NJCPVT) — INTEGER array *Workspace*

17: NJCPVT — INTEGER *Input*

On initial entry: the dimension of the array JACPVT as declared in the (sub)program from which D02NNF is called. The actual size depends on the linear algebra method used. An appropriate value for NJCPVT is described in the specifications of the linear algebra setup routines D02NTF and D02NUF for banded and sparse matrix linear algebra respectively. This value must be the same as that supplied to the linear algebra setup routine. When full matrix linear algebra is chosen, the array JACPVT is not used and hence NJCPVT should be set to 1.

18: IMON — INTEGER *Input/Output*

On intermediate exit: used to pass information between D02NNF and the MONITR operation (see Section 3). With IREVCM = 9, IMON contains a flag indicating under what circumstances the return from D02NNF occurred:

IMON = -2

Exit from D02NNF after IRES= 4 (set in RESID operation (see Section 3) caused an early termination (this facility could be used to locate discontinuities).

IMON = -1

The current step failed repeatedly.

IMON = 0

Exit from D02NNF after a call to the internal nonlinear equation solver.

IMON = 1

The current step was successful.

On intermediate re-entry: IMON may be reset to determine subsequent action in D02NNF.

IMON = -2

Integration is to be halted. A return will be made from D02NNF to the calling (sub)program with IFAIL = 12.

IMON = -1

Allow D02NNF to continue with its own internal strategy. The integrator will try up to 3 restarts unless IMON is set $\neq -1$.

IMON = 0

Return to the internal nonlinear equation solver, where the action taken is determined by the value of INLN (see below).

IMON = 1

Normal exit to D02NNF to continue integration.

IMON = 2

Restart the integration at the current time point. The integrator will restart from order 1 when this option is used. The internal initialisation module solves for new values of y and y' by using the values supplied in Y and YDOT by the MONITR operation (see Section 3) as initial estimates.

IMON = 3

Try to continue with the same step size and order as was to be used before entering the MONITR operation (see Section 3). HMIN and HMAX may be altered if desired.

IMON = 4

Continue the integration but using a new value HNEXT and possibly new values of HMIN and HMAX.

19: INLN — INTEGER *Input*

On intermediate re-entry: with IMON = 0 and IREVCM = 9, INLN specifies the action to be taken by the internal nonlinear equation solver. By setting INLN = 3 and returning to D02NNF, the residual vector is evaluated and placed in RWORK(50+2×NEQMAX+i), for $i = 1, 2, \ldots$,NEQ and then the MONITR operation (see Section 3) is invoked again. At present this is the only option available: INLN must not be set to any other value.

20: IRES — INTEGER *Input/Output*

On intermediate exit: with IREVCM = 1, 2, 3, 4, 5, 6, 7 or 11 IRES specifies the form of the residual to be returned by the RESID operation (see Section 3).

If IRES = 1 then

$$r = g(t, y) - A(t, y)y'$$

must be returned.

If IRES = −1 then

$$r = -A(t, y)y'$$

must be returned.

On intermediate re-entry: IRES should be unchanged unless one of the following actions is required of D02NNF in which case IRES should be set accordingly.

IRES = 2

indicates to D02NNF that control should be passed back immediately to the calling (sub)program with the error indicator set to IFAIL = 11.

IRES = 3

indicates to D02NNF that an error condition has occurred in the solution vector, its time derivative or in the value of t. The integrator will use a smaller time step to try to avoid this condition. If this is not possible D02NNF returns to the calling (sub)program with the error indicator set to IFAIL = 7.

IRES = 4

indicates to D02NNF to stop its current operation and to enter the MONITR operation (see Section 3) immediately.

21: IREVCM — INTEGER *Input/Output*

On initial entry: IREVCM must contain 0.

On intermediate re-entry: should remain unchanged.

On intermediate exit: indicates what action the user must take before re-entering D02NNF. The possible exit values of IREVCM are 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 which should be interpreted as follows:

IREVCM = 1, 2, 3, 4, 5, 6, 7, 11

indicates that a RESID operation (see Section 3) is required: the user must supply the residual of the system. For each of these values of IREVCM y_i is located in Y(i), $i = 1, 2, \ldots$,NEQ.

For IREVCM = 1, 3, 6 or 11, y_i' is located in YDOT(i) and r_i should be stored in RWORK(50+2×NEQMAX+i), for $i = 1, 2, \ldots$,NEQ.

For IREVCM = 2, y_i' is located in RWORK(50+NEQMAX+i) and r_i should be stored in RWORK(50+2×NEQMAX+i), for $i = 1, 2, \ldots$,NEQ.

For IREVCM = 4 or 7, y_i' is located in YDOT(i) and r_i should be stored in RWORK(50+NEQMAX+i), for $i = 1, 2, \ldots$,NEQ.

For IREVCM = 5, y_i' is located in RWORK(50+2×NEQMAX+i) and r_i should be stored in YDOT(i), for $i = 1, 2, \ldots$,NEQ.

IREVCM = 8

indicates that a JAC operation (see Section 3) is required: the user must supply the Jacobian matrix.

If full matrix linear algebra is being used, then the (i, j)th element of the Jacobian must be stored in WKJAC(($j - 1$)×NEQ+i).

If banded matrix linear algebra is being used, then the (i, j)th element of the Jacobian must be stored in WKJAC($(i-1) \times m_B + k$), where $m_B = m_L + m_U + 1$ and $k = \min(m_L - i + 1, 0) + j$; here m_L and m_U are the number of sub-diagonals and super-diagonals, respectively, in the band.

If sparse matrix linear algebra is being used, then D02NRF must be called to determine which column of the Jacobian is required and where it should be stored.

```
CALL D02NRF(J, IPLACE, INFORM)
```

will return in J the number of the column of the Jacobian that is required and will set IPLACE = 1 or 2. If IPLACE = 1, then the (i, j)th element of the Jacobian must be stored in RWORK(50+2×NEQMAX+i); otherwise it must be stored in RWORK(50+NEQMAX+i).

IREVCM = 9

indicates that a MONITR operation (see Section 3) can be performed.

IREVCM = 10

indicates that the current step was not successful, due to error test failure or convergence test failure. The only information supplied to the user on this return is the current value of the variable t, located in RWORK(19). No values must be changed before re-entering D02NNF; this facility enables the user to determine the number of unsuccessful steps.

On final exit: IREVCM = 0 indicating that the user-specified task has been completed or an error has been encountered (see descriptions for ITASK and IFAIL.

Constraint: $0 \leq$ IREVCM ≤ 11.

22: LDERIV(2) — LOGICAL array *Input*

On initial entry: LDERIV(1) must be set to .TRUE. if the user has supplied both an initial y and an initial y'. LDERIV(1) must be set to .FALSE. if only the initial y has been supplied.

LDERIV(2) must be set to .TRUE. if the integrator is to use a modified Newton method to evaluate the initial y and y'. Note that y and y', if supplied, are used as initial estimates. This method involves taking a small step at the start of the integration, and if ITASK = 6 on entry, T and TOUT will be set to the result of taking this small step. LDERIV(2) must be set to .FALSE. if the integrator is to use functional iteration to evaluate the initial y and y', and if this fails a modified Newton method will then be attempted. LDERIV(2) = .TRUE. is recommended if there are implicit equations or the initial y and y' are zero.

On final exit: LDERIV(1) is normally unchanged. However if ITASK = 6 and internal initialisation was successful then LDERIV(1) = .TRUE..

LDERIV(2) = .TRUE., if implicit equations were detected. Otherwise LDERIV(2) = .FALSE..

23: ITASK — INTEGER *Input*

On initial entry: the task to be performed by the integrator. The permitted values for ITASK and their meanings are detailed below:

ITASK = 1

normal computation of output values of $y(t)$ at t = TOUT (by overshooting and interpolating).

ITASK = 2

take one step only and return.

ITASK = 3

stop at the first internal integration point at or beyond t = TOUT and return.

ITASK = 4

normal computation of output values of $y(t)$ at t = TOUT but without overshooting t = TCRIT. TCRIT must be specified as an option in one of the integrator setup routines prior to the first call to the integrator, or specified in the optional input routine prior to a continuation call. TCRIT may be equal to or beyond TOUT, but not before it in the direction of integration.

ITASK = 5

take one step only and return, without passing TCRIT. TCRIT must be specified under ITASK = 4.

ITASK = 6

the integrator will solve for the initial values of y and y' only and then return to the calling (sub)program without doing the integration. This option can be used to check the initial values of y and y'. Functional iteration or a 'small' backward Euler method used in conjunction with a damped Newton iteration is used to calculate these values (see LDERIV above). Note that if a backward Euler step is used then the value of t will have been advanced a short distance from the initial point.

Note. If D02NNF is recalled with a different value of ITASK (and TOUT altered) then the initialisation procedure is repeated, possibly leading to different initial conditions.

Constraint: $1 \leq$ ITASK ≤ 6.

24: ITRACE — INTEGER *Input*

On initial entry: the level of output that is printed by the integrator. ITRACE may take the value -1, 0, 1, 2 or 3. If ITRACE < -1, then -1 is assumed and similarly if ITRACE > 3, then 3 is assumed. If ITRACE $= -1$, no output is generated. If ITRACE = 0, only warning messages are printed on the current error message unit (see X04AAF). If ITRACE > 0, then warning messages are printed as above, and on the current advisory message unit (see X04ABF) output is generated which details Jacobian entries, the nonlinear iteration and the time integration. The advisory messages are given in greater detail the larger the value of ITRACE.

25: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, the integrator detected an illegal input or that a linear algebra and/or integrator setup routine has not been called prior to the call to the integrator. If ITRACE $\geq$ 0, the form of the error will be detailed on the current error message unit (see X04AAF).

IFAIL = 2

The maximum number of steps specified has been taken (see the description of optional inputs in the integrator setup routines and the optional input continuation routine, D02NZF).

IFAIL = 3

With the given values of RTOL and ATOL no further progress can be made across the integration range from the current point T. The components Y(1),Y(2),...,Y(NEQ) contain the computed values of the solution at the current point T.

IFAIL = 4

There were repeated error test failures on an attempted step, before completing the requested task, but the integration was successful as far as T. The problem may have a singularity, or the local error requirements may be inappropriate.

IFAIL = 5

There were repeated convergence test failures on an attempted step, before completing the requested task, but the integration was successful as far as T. This may be caused by an inaccurate Jacobian matrix or one which is incorrectly computed.

IFAIL = 6

Some error weight w_i became zero during the integration (see description of ITOL). Pure relative error control (ATOL(i) = 0.0) was requested on a variable (the ith) which has now vanished. The integration was successful as far as T.

IFAIL = 7

The RESID operation (see Section 3) set the error flag IRES = 3 continually despite repeated attempts by the integrator to avoid this.

IFAIL = 8

LDERIV(1) = .FALSE. on entry but the internal initialisation routine was unable to initialise y' (more detailed information may be directed to the current error message unit, see X04AAF).

IFAIL = 9

A singular Jacobain $\frac{\partial r}{\partial y}$ has been encountered. The user should check his problem formulation and Jacobian calculation.

IFAIL = 10

An error occurred during Jacobian formulation or back-substitution (a more detailed error description may be directed to the current error message unit, see X04AAF).

IFAIL = 11

The RESID operation (see Section 3) signalled the integrator to halt the integration and return by setting IRES = 2. Integration was successful as far as T.

IFAIL = 12

The MONITR operation (see Section 3) set IMON = −2 and so forced a return but the integration was successful as far as T.

IFAIL = 13

The requested task has been completed, but it is estimated that a small change in RTOL and ATOL is unlikely to produce any change in the computed solution. (Only applies when the user is not operating in one step mode, that is when ITASK ≠ 2 or 5).

IFAIL = 14

The values of RTOL and ATOL are so small that the routine is unable to start the integration.

7 Accuracy

The accuracy of the numerical solution may be controlled by a careful choice of the parameters RTOL and ATOL, and to a much lesser extent by the choice of norm. Users are advised to use scalar error control unless the components of the solution are expected to be poorly scaled. For the type of decaying solution typical of many stiff problems, relative error control with a small absolute error threshold will be most appropriate (that is the user is advised to choose ITOL = 1 with ATOL(1) small but positive).

8 Further Comments

The cost of computing a solution depends critically on the size of the differential system and to a lesser extent on the degree of stiffness of the problem; also on the type of linear algebra being used. For further details see Section 8 of the documents for D02NGF (full matrix), D02NHF (banded matrix) or D02NJF (sparse matrix).

In general the user is advised to choose the Backward Differentiation Formula option (setup routine D02NVF) but if efficiency is of great importance and especially if it is suspected that $\frac{\partial}{\partial y}(A^{-1}g)$ has complex eigenvalues near the imaginary axis for some part of the integration, the user should try the BLEND option (setup routine D02NWF).

9 Example

We solve the well-known stiff Robertson problem written as a differential system in implicit form

$$\begin{array}{rcl} r_1 & = & (a' + \quad b' + c') \\ r_2 & = & 0.04a - 1.0\mathrm{E}4bc - 3.0\mathrm{E}7b^2 \quad - b' \\ r_3 & = & 3.0\mathrm{E}7b^2 \quad - c' \end{array}$$

over the range [0,10] with initial conditions $a = 1.0$ and $b = c = 0.0$ and with scalar error control (ITOL = 1). We integrate to the first internal integration point past TOUT = 10.0 (ITASK = 3), using a BDF method (setup routine D02NVF) and a modified Newton method. We treat the Jacobian as sparse (setup routine D02NUF) and we calculate it analytically. In this program we also illustrate the monitoring of step failures (IREVCM = 10) and forcing of a return when the component a falls below 0.9 in the evaluation of the residual by setting IRES = 2.

9.1 Program Text

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

```
*     D02NNF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, NEQMAX, NRW, NINF, NJCPVT, NWKJAC, NIA, NJA,
     +                 MAXORD, NY2DIM, MAXSTP, MXHNIL
      PARAMETER        (NEQ=3,NEQMAX=NEQ,NRW=50+4*NEQMAX,NINF=23,
     +                 NJCPVT=150,NWKJAC=100,NIA=1,NJA=1,MAXORD=5,
     +                 NY2DIM=MAXORD+1,MAXSTP=200,MXHNIL=5)
      INTEGER          LACORB, LSAVRB
      PARAMETER        (LACORB=50+NEQMAX,LSAVRB=LACORB+NEQMAX)
      real             H0, HMAX, HMIN, TCRIT
      PARAMETER        (H0=1.0e-4,HMAX=10.0e0,HMIN=1.0e-10,TCRIT=0.0e0)
      LOGICAL          PETZLD
      PARAMETER        (PETZLD=.TRUE.)
      real             ETA, U, SENS
      PARAMETER        (ETA=1.0e-4,U=0.1e0,SENS=1.0e-6)
      LOGICAL          LBLOCK
      PARAMETER        (LBLOCK=.TRUE.)
*     .. Local Scalars ..
      real             H, HU, HXD, T, TCUR, TOLSF, TOUT
      INTEGER          I, ICALL, IFAIL, IGROW, IMON, IMXER, INLN,
     +                 IPLACE, IRES, IREVCM, ISPLIT, ITASK, ITOL,
     +                 ITRACE, J, LACOR1, LACOR2, LACOR3, LIWREQ,
     +                 LIWUSD, LRWREQ, LRWUSD, LSAVR1, LSAVR2, LSAVR3,
     +                 NBLOCK, NFAILS, NGP, NITER, NJE, NLU, NNZ, NQ,
     +                 NQU, NRE, NST
*     .. Local Arrays ..
      real             ATOL(NEQMAX), CONST(6), RTOL(NEQMAX), RWORK(NRW),
     +                 WKJAC(NWKJAC), Y(NEQMAX), YDOT(NEQMAX),
     +                 YSAVE(NEQMAX,NY2DIM)
      INTEGER          IA(NIA), INFORM(NINF), JA(NJA), JACPVT(NJCPVT)
      LOGICAL          ALGEQU(NEQMAX), LDERIV(2)
*     .. External Subroutines ..
      EXTERNAL         D02NNF, D02NRF, D02NUF, D02NVF, D02NXF, D02NYF,
     +                 X04ABF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02NNF Example Program Results'
      WRITE (NOUT,*)
      CALL X04ABF(1,NOUT)
*
*     Integrate towards TOUT stopping at the first mesh point beyond
*     TOUT (ITASK=3) using the B.D.F. formulae with a Newton method.
*     Employ scalar tolerances and the Jacobian is supplied, but its
*     structure is evaluated internally by calls to the Jacobian
*     forming part of the program (IREVCM=8). Default values for the
*     array CONST are used. Also count the number of step failures
*     (IREVCM=10).
*
      T = 0.0e0
      TOUT = 10.0e0
      ITASK = 3
      Y(1) = 1.0e0
      Y(2) = 0.0e0
      Y(3) = 0.0e0
      LDERIV(1) = .FALSE.
      LDERIV(2) = .FALSE.
      ITOL = 1
      RTOL(1) = 1.0e-4
      ATOL(1) = 1.0e-7
      DO 20 I = 1, 6
         CONST(I) = 0.0e0
```

```
   20 CONTINUE
      ISPLIT = 0
      NFAILS = 0
      IFAIL = 0
*
      CALL D02NVF(NEQMAX,NY2DIM,MAXORD,'Newton',PETZLD,CONST,TCRIT,HMIN,
     +            HMAX,H0,MAXSTP,MXHNIL,'Average-l2',RWORK,IFAIL)
      CALL D02NUF(NEQ,NEQMAX,'Analytical',NWKJAC,IA,NIA,JA,NJA,JACPVT,
     +            NJCPVT,SENS,U,ETA,LBLOCK,ISPLIT,RWORK,IFAIL)
*
*     Soft fail and error messages only
      IREVCM = 0
      IFAIL = 1
      ITRACE = 0
*
      LACOR1 = LACORB + 1
      LACOR2 = LACORB + 2
      LACOR3 = LACORB + 3
      LSAVR1 = LSAVRB + 1
      LSAVR2 = LSAVRB + 2
      LSAVR3 = LSAVRB + 3
      WRITE (NOUT,*) '      X          Y(1)           Y(2)           Y(3)'
      WRITE (NOUT,99999) T, (Y(I),I=1,NEQ)
*
   40 CONTINUE
*
      CALL D02NNF(NEQ,NEQMAX,T,TOUT,Y,YDOT,RWORK,RTOL,ATOL,ITOL,INFORM,
     +            YSAVE,NY2DIM,WKJAC,NWKJAC,JACPVT,NJCPVT,IMON,INLN,
     +            IRES,IREVCM,LDERIV,ITASK,ITRACE,IFAIL)
*
      IF (IREVCM.GT.0) THEN
         IF (IREVCM.EQ.1 .OR. IREVCM.EQ.3 .OR. IREVCM.EQ.6 .OR.
     +       IREVCM.EQ.11) THEN
*           Equivalent to RESID evaluation in forward communication
*           routines
            RWORK(LSAVR1) = -YDOT(1) - YDOT(2) - YDOT(3)
            RWORK(LSAVR2) = -YDOT(2)
            RWORK(LSAVR3) = -YDOT(3)
            IF (IRES.EQ.1) THEN
               RWORK(LSAVR1) = 0.0e0 + RWORK(LSAVR1)
               RWORK(LSAVR2) = 0.04e0*Y(1) - 1.0e4*Y(2)*Y(3) -
     +                         3.0e7*Y(2)*Y(2) + RWORK(LSAVR2)
               RWORK(LSAVR3) = 3.0e7*Y(2)*Y(2) + RWORK(LSAVR3)
            END IF
         ELSE IF (IREVCM.EQ.2) THEN
*           Equivalent to RESID evaluation in forward communication
*           routines
            RWORK(LSAVR1) = -RWORK(LACOR1) - RWORK(LACOR2) -
     +                      RWORK(LACOR3)
            RWORK(LSAVR2) = -RWORK(LACOR2)
            RWORK(LSAVR3) = -RWORK(LACOR3)
         ELSE IF (IREVCM.EQ.4 .OR. IREVCM.EQ.7) THEN
*           Equivalent to RESID evaluation in forward communication
*           routines
            RWORK(LACOR1) = -YDOT(1) - YDOT(2) - YDOT(3)
            RWORK(LACOR2) = -YDOT(2)
            RWORK(LACOR3) = -YDOT(3)
            IF (IRES.EQ.1) THEN
```

```
               RWORK(LACOR1) = 0.0e0 + RWORK(LACOR1)
               RWORK(LACOR2) = 0.04e0*Y(1) - 1.0e4*Y(2)*Y(3) -
     +                         3.0e7*Y(2)*Y(2) + RWORK(LACOR2)
               RWORK(LACOR3) = 3.0e7*Y(2)*Y(2) + RWORK(LACOR3)
            END IF
         ELSE IF (IREVCM.EQ.5) THEN
*           Equivalent to RESID evaluation in forward communication
*           routines
            YDOT(1) = 0.0e0 - RWORK(LSAVR1) - RWORK(LSAVR2) -
     +                RWORK(LSAVR3)
            YDOT(2) = 0.04e0*Y(1) - 1.0e4*Y(2)*Y(3) - 3.0e7*Y(2)*(2) -
     +                RWORK(LSAVR2)
            YDOT(3) = 3.0e7*Y(2)*Y(2) - RWORK(LSAVR3)
         ELSE IF (IREVCM.EQ.8) THEN
*           Equivalent to JAC evaluation in forward communication
*           routines
            CALL D02NRF(J,IPLACE,INFORM)
*
            HXD = RWORK(16)*RWORK(20)
*
            IF (IPLACE.LT.2) THEN
               IF (J.LT.2) THEN
                  RWORK(LSAVR1) = 1.0e0 - HXD*(0.0e0)
                  RWORK(LSAVR2) = 0.0e0 - HXD*(0.04e0)
*                 RWORK(LSAVR3) = 0.0 - HXD*(0.0)
               ELSE IF (J.EQ.2) THEN
                  RWORK(LSAVR1) = 1.0e0 - HXD*(0.0e0)
                  RWORK(LSAVR2) = 1.0e0 - HXD*(-1.0e4*Y(3)-6.0e7*Y(2))
                  RWORK(LSAVR3) = 0.0e0 - HXD*(6.0e7*Y(2))
               ELSE IF (J.GT.2) THEN
                  RWORK(LSAVR1) = 1.0e0 - HXD*(0.0e0)
                  RWORK(LSAVR2) = 0.0e0 - HXD*(-1.0e4*Y(2))
                  RWORK(LSAVR3) = 1.0e0 - HXD*(0.0e0)
               END IF
            ELSE
               IF (J.LT.2) THEN
                  RWORK(LACOR1) = 1.0e0 - HXD*(0.0e0)
                  RWORK(LACOR2) = 0.0e0 - HXD*(0.04e0)
*                 RWORK(LACOR3) = 0.0 - HXD*(0.0)
               ELSE IF (J.EQ.2) THEN
                  RWORK(LACOR1) = 1.0e0 - HXD*(0.0e0)
                  RWORK(LACOR2) = 1.0e0 - HXD*(-1.0e4*Y(3)-6.0e7*Y(2))
                  RWORK(LACOR3) = 0.0e0 - HXD*(6.0e7*Y(2))
               ELSE IF (J.GT.2) THEN
                  RWORK(LACOR1) = 1.0e0 - HXD*(0.0e0)
                  RWORK(LACOR2) = 0.0e0 - HXD*(-1.0e4*Y(2))
                  RWORK(LACOR3) = 1.0e0 - HXD*(0.0e0)
               END IF
            END IF
*           Step failure
         ELSE IF (IREVCM.EQ.10) THEN
            NFAILS = NFAILS + 1
         END IF
         GO TO 40
      ELSE
         IF (IFAIL.EQ.0) THEN
            WRITE (NOUT,99999) T, (Y(I),I=1,NEQ)
*
```

```
            CALL D02NYF(NEQ,NEQMAX,HU,H,TCUR,TOLSF,RWORK,NST,NRE,NJE,
     +                  NQU,NQ,NITER,IMXER,ALGEQU,INFORM,IFAIL)
*
            WRITE (NOUT,*)
            WRITE (NOUT,99997) ' HUSED = ', HU, '  HNEXT = ', H,
     +        '  TCUR = ', TCUR
            WRITE (NOUT,99996) ' NST = ', NST, '    NRE = ', NRE,
     +        '    NJE = ', NJE
            WRITE (NOUT,99996) ' NQU = ', NQU, '    NQ  = ', NQ,
     +        '  NITER = ', NITER
            WRITE (NOUT,99995) ' Max err comp = ', IMXER,
     +        '   No. of failed steps = ', NFAILS
            ICALL = 0
*
            CALL D02NXF(ICALL,LIWREQ,LIWUSD,LRWREQ,LRWUSD,NLU,NNZ,NGP,
     +                  ISPLIT,IGROW,LBLOCK,NBLOCK,INFORM)
*
            WRITE (NOUT,*)
            WRITE (NOUT,99994) ' NJCPVT (required ', LIWREQ, '  used ',
     +        LIWUSD, ')'
            WRITE (NOUT,99994) ' NWKJAC (required ', LRWREQ, '  used ',
     +        LRWUSD, ')'
            WRITE (NOUT,99993) ' No. of LU-decomps ', NLU,
     +        '  No. of nonzeros ', NNZ
            WRITE (NOUT,99995) ' No. of FCN calls to form Jacobian ',
     +        NGP, '  Try ISPLIT ', ISPLIT
            WRITE (NOUT,99992) ' Growth est ', IGROW,
     +        '  No. of blocks on diagonal ', NBLOCK
         ELSE IF (IFAIL.EQ.10) THEN
            ICALL = 1
*
            CALL D02NXF(ICALL,LIWREQ,LIWUSD,LRWREQ,LRWUSD,NLU,NNZ,NGP,
     +                  ISPLIT,IGROW,LBLOCK,NBLOCK,INFORM)
*
            WRITE (NOUT,*)
            WRITE (NOUT,99994) ' NJCPVT (required ', LIWREQ, '  used ',
     +        LIWUSD, ')'
            WRITE (NOUT,99994) ' NWKJAC (required ', LRWREQ, '  used ',
     +        LRWUSD, ')'
         ELSE
            WRITE (NOUT,*)
            WRITE (NOUT,99998) 'Exit D02NNF with IFAIL = ', IFAIL,
     +        '  and T = ', T
         END IF
      END IF
      STOP
*
99999 FORMAT (1X,F8.3,3(F13.5,2X))
99998 FORMAT (1X,A,I2,A,e12.5)
99997 FORMAT (1X,A,e12.5,A,e12.5,A,e12.5)
99996 FORMAT (1X,A,I6,A,I6,A,I6)
99995 FORMAT (1X,A,I4,A,I4)
99994 FORMAT (1X,A,I8,A,I8,A)
99993 FORMAT (1X,A,I4,A,I8)
99992 FORMAT (1X,A,I8,A,I4)
      END
```

9.2 Program Data

None.

9.3 Program Results

```
D02NNF Example Program Results

    X          Y(1)           Y(2)           Y(3)
   0.000      1.00000        0.00000        0.00000
WARNING... EQUATION(=I1) AND POSSIBLY OTHER EQUATIONS ARE
IMPLICIT AND IN CALCULATING THE INITIAL VALUES THE EQNS
WILL BE TREATED AS IMPLICIT.
IN ABOVE MESSAGE I1 =         1
  10.488      0.83759        0.00002        0.16239

 HUSED =  0.60471D+00  HNEXT =  0.60471D+00  TCUR =  0.10488D+02
 NST =     65    NRE =    163    NJE =     14
 NQU =      3    NQ  =      3  NITER =    154
 Max err comp =    3   No. of failed steps =    0

 NJCPVT (required       74  used      150)
 NWKJAC (required       16  used       77)
 No. of LU-decomps   14  No. of nonzeros       5
 No. of FCN calls to form Jacobian    0  Try ISPLIT   73
 Growth est       862  No. of blocks on diagonal    3
```

D02NRF – NAG Fortran Library Routine Document

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

1. Purpose

D02NRF is an enquiry routine for communicating with D02NMF or D02NNF when supplying columns of a sparse Jacobian matrix.

2. Specification

```
SUBROUTINE D02NRF (J, IPLACE, INFORM)
INTEGER         J, IPLACE, INFORM(23)
```

3. Description

D02NRF is required when D02NMF or D02NNF is being used with sparse matrix linear algebra. After an exit from D02NMF or D02NNF with IREVCM = 8, D02NRF must be called to determine which column of the Jacobian is required and where it is to be placed in the array RWORK (a parameter of D02NMF or D02NNF).

4. References

None.

5. Parameters

1: J – INTEGER. *Output*

On exit: the index j of the column of the Jacobian which is required.

2: IPLACE – INTEGER. *Output*

On exit: indicates which locations in the array RWORK to fill with the jth column. If IPLACE = 1 the (i,j)th element of the Jacobian must be placed in RWORK(50+2×NEQMAX+i), otherwise the (i,j)th element must be placed in RWORK(50+NEQMAX+i). If JCEVAL = 'F', in the previous call to D02NUF, then IPLACE = 2 always, hence the jth column of the Jacobian must be placed in RWORK(50+NEQMAX+i), for i = 1,2,...,NEQ.

RWORK, NEQ and NEQMAX are parameters of D02NMF and D02NNF.

3: INFORM(23) – INTEGER array. *Workspace*

This must be the same array as the array INFORM supplied to D02NMF or D02NNF. Its contents must not be changed between calls of D02NMF or D02NNF and calls of D02NRF.

6. Error Indicators and Warnings

None.

7. Accuracy

Not applicable.

8. Further Comments

None.

9. Example

See the example for D02NNF.

D02NSF – NAG Fortran Library Routine Document

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

1. Purpose

D02NSF is a setup routine which must be called by the user, prior to an integrator in the subchapter D02M-D02N, if full matrix linear algebra is required.

2. Specification

```
SUBROUTINE D02NSF (NEQ, NEQMAX, JCEVAL, NWKJAC, RWORK, IFAIL)
INTEGER        NEQ, NEQMAX, NWKJAC, IFAIL
real           RWORK(50+4*NEQMAX)
CHARACTER*1    JCEVAL
```

3. Description

This routine defines the linear algebra to be used as full matrix linear algebra, permits the user to specify the method for calculating the Jacobian and checks the validity of certain input values.

4. References

None.

5. Parameters

1: NEQ – INTEGER. *Input*

On entry: the number of differential equations.

Constraint: $1 \leq \text{NEQ} \leq \text{NEQMAX}$.

2: NEQMAX – INTEGER. *Input*

On entry: a bound on the maximum number of differential equations to be solved during the integration.

Constraint: $\text{NEQMAX} \geq \text{NEQ}$.

3: JCEVAL – CHARACTER*1. *Input*

On entry: specifies the technique to be used to compute the Jacobian, as follows:

JCEVAL = 'N'

the Jacobian is to be evaluated numerically by the integrator. If this option is used, then the actual argument corresponding to JAC in the call to D02NBF or D02NGF must be either D02NBZ or D02NGZ respectively.

JCEVAL = 'A'

the user will supply a subroutine to evaluate the Jacobian on a call to the integrator.

JCEVAL = 'D'

the default choice is to be made. In this case 'D' is interpreted as 'N'.

Only the first character of the actual argument JCEVAL is passed to D02NSF; hence it is permissible for the actual argument to be more descriptive e.g. 'Numerical', 'Analytical' or 'Default' on a call to D02NSF.

Constraint: JCEVAL = 'N', 'A' or 'D'.

4: NWKJAC – INTEGER. *Input*

On entry: the size of the workspace array WKJAC, which the user is supplying to the integrator, as declared in the (sub)program from which D02NSF is called.

Constraint: NWKJAC ≥ NEQMAX×(NEQMAX+1).

5: RWORK(50+4*NEQMAX) – ***real*** array. *Workspace*

This must be the same workspace array as the array RWORK supplied to the integrator. It is used to pass information from the setup routine to the integrator and therefore the contents of this array must not be changed before calling the integrator.

6: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, NEQ < 1,
or NEQ > NEQMAX,
or NWKJAC < NEQMAX×(NEQMAX+1),
or JCEVAL ≠ 'N', 'A' or 'D'.

7. Accuracy

Not applicable.

8. Further Comments

This routine must be called as a setup routine before a call to either D02NBF or D02NGF and may be called as the linear algebra setup routine before a call to either D02NMF or D02NNF.

9. Example

See the examples for D02NBF, D02NGF and D02NMF.

D02NTF – NAG Fortran Library Routine Document

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

1. Purpose

D02NTF is a setup routine which must be called by the user, prior to an integrator in the subchapter D02M-D02N, if banded matrix linear algebra is required.

2. Specification

```
 SUBROUTINE D02NTF (NEQ, NEQMAX, JCEVAL, ML, MU, NWKJAC, NJCPVT,
1                    RWORK, IFAIL)
 INTEGER         NEQ, NEQMAX, ML, MU, NWKJAC, NJCPVT, IFAIL
 real            RWORK(50+4*NEQMAX)
 CHARACTER*1     JCEVAL
```

3. Description

This routine defines the linear algebra to be used as banded matrix linear algebra, permits the user to specify the method for calculating the Jacobian and checks the validity of certain input values.

4. References

None.

5. Parameters

1: NEQ – INTEGER. *Input*

On entry: the number of differential equations.

Constraint: 1 ≤ NEQ ≤ NEQMAX.

2: NEQMAX – INTEGER. *Input*

On entry: a bound on the maximum number of differential equations to be solved during the integration.

Constraint: NEQMAX ≥ NEQ.

3: JCEVAL – CHARACTER*1. *Input*

On entry: specifies the technique to be used to compute the Jacobian as follows:

JCEVAL = 'N'

the Jacobian is to be evaluated numerically by the integrator. If this option is used, then the actual argument corresponding to JAC in the call to D02NCF or D02NHF must be either D02NCZ or D02NHZ respectively.

JCEVAL = 'A'

the user will supply a subroutine to evaluate the Jacobian on a call to the integrator.

JCEVAL = 'D'

the default choice is to be made. In this case 'D' is interpreted as 'N'.

Only the first character of the actual argument JCEVAL is passed to D02NTF; hence it is permissible for the actual argument to be more descriptive e.g. 'Numerical', 'Analytical' or 'Default' on a call to D02NTF.

Constraint: JCEVAL = 'N', 'A' or 'D'.

4: ML – INTEGER. *Input*

On entry: the number of sub-diagonals in the band, m_L.

Constraint: $0 \le$ ML $\le$ NEQ–1.

5: MU – INTEGER. *Input*

On entry: the number of super-diagonals in the band, m_U.

Constraint: $0 \le$ MU $\le$ NEQ–1.

6: NWKJAC – INTEGER. *Input*

On entry: the size of the workspace array WKJAC, which the user is supplying to the integrator, as declared in the (sub)program from which D02NTF is called.

Constraint: NWKJAC $\ge$ (2×ML+MU+1)×NEQMAX.

7: NJCPVT – INTEGER. *Input*

On entry: the size of the workspace array JACPVT, which the user is supplying to the integrator, as declared in the (sub)program from which D02NTF is called.

Constraint: NJCPVT $\ge$ NEQMAX.

8: RWORK(50+4*NEQMAX) – ***real*** array. *Workspace*

This must be the same workspace array as the array RWORK supplied to the integrator. It is used to pass information from the setup routine to the integrator and therefore the contents of this array must not be changed before calling the integrator.

9: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, JCEVAL ≠ 'N' or 'A' or 'D',
or NEQ < 1,
or ML < 0 or ML > NEQ–1,
or MU < 0 or MU > NEQ–1,
or NEQ > NEQMAX,
or NJCPVT < NEQMAX,
or NWKJAC < (2×ML+MU+1)×NEQMAX.

7. Accuracy

Not applicable.

8. Further Comments

This routine must be called as a setup routine before a call to either D02NCF or D02NHF and may be called as the linear algebra setup routine before a call to either D02NMF or D02NNF.

9. Example

See the examples for D02NCF and D02NHF.

D02NUF – NAG Fortran Library Routine Document

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

1. Purpose

D02NUF is a setup routine which must be called by the user, prior to an integrator in the subchapter D02M-D02N, if sparse matrix linear algebra is required.

2. Specification

```
 SUBROUTINE D02NUF (NEQ, NEQMAX, JCEVAL, NWKJAC, IA, NIA, JA, NJA,
1                   JACPVT, NJCPVT, SENS, U, ETA, LBLOCK, ISPLIT,
2                   RWORK, IFAIL)
 INTEGER         NEQ, NEQMAX, NWKJAC, IA(NIA), NIA, JA(NJA), NJA,
1                JACPVT(NJCPVT), NJCPVT, ISPLIT, IFAIL
 real            SENS, U, ETA, RWORK(50+4*NEQMAX)
 LOGICAL         LBLOCK
 CHARACTER*1     JCEVAL
```

3. Description

This routine defines the linear algebra to be used as sparse matrix linear algebra, permits the user to specify the method for calculating the Jacobian and its structure, and checks the validity of certain input values.

4. References

None.

5. Parameters

1: NEQ – INTEGER. *Input*

On entry: the number of differential equations.

Constraint: 1 ≤ NEQ ≤ NEQMAX.

2: NEQMAX – INTEGER. *Input*

On entry: a bound on the maximum number of differential equations to be solved during the integration.

Constraint: NEQ ≤ NEQMAX.

3: JCEVAL – CHARACTER*1. *Input*

On entry: specifies the technique to be used to compute the Jacobian, as follows:

JCEVAL = 'N'

the sparsity structure and the value of the Jacobian are to be determined numerically by the integrator.

JCEVAL = 'S'

the sparsity structure of the Jacobian is supplied in the arrays IA and JA but its value is to be determined numerically. This is the recommended mode of operation unless it is a simple matter to supply the Jacobian.

JCEVAL = 'A'

the Jacobian will be evaluated by calls to a subroutine JAC supplied by the user. The sparsity structure will be estimated by calls to JAC; that is, no explicit sparsity structure need be supplied in the arrays IA and JA.

JCEVAL = 'F'

the sparsity structure of the Jacobian is supplied in IA and JA, and its value will be determined by calls to a subroutine JAC supplied by the user. This is the recommended mode of operation if the subroutine JAC is simple to form.

JCEVAL = 'D'

the default choice is to be made. In this case 'D' is interpreted as 'S'.

If the sparsity structure is supplied in arrays IA and JA, then any evidence from the numerical or analytical formation of the Jacobian that this structure is not correct, is ignored.

Only the first character of the actual argument JCEVAL is passed to D02NUF; hence it is permissible for the actual argument to be more descriptive e.g. 'Numerical', 'Structural', 'Analytical', 'Full information' or 'Default' in a call to D02NUF.

If the option JCEVAL = 'N', 'S' or 'D' is used then the actual argument corresponding to JAC in the call to D02NDF or D02NJF must be either D02NDZ or D02NJZ respectively.

Constraint: JCEVAL = 'N', 'S', 'A', 'F' or 'D'.

4: NWKJAC – INTEGER. *Input*

On entry: the size of the array WKJAC, that the user is supplying to the integrator, as declared in the (sub)program from which D02NUF is called.

Suggested value: NWKJAC = 4×NEQMAX if JCEVAL = 'N' or 'A'. If NWKJAC is less than this estimate, then a message is printed on the current advisory message unit (see X04ABF), and execution continues.

Constraints: NWKJAC ≥ NELEMENT + 2×NEQ if JCEVAL = 'S', 'F' or 'D', where NELEMENT is the total number of non-zeros.

5: IA(NIA) – INTEGER array. *Input*

On entry: if JCEVAL = 'S', 'F' or 'D', IA must contain details of the sparsity pattern to be used for the Jacobian. See JA below.

IA is not used if JCEVAL = 'N' or 'A'.

6: NIA – INTEGER. *Input*

On entry: the dimension of the array IA as declared in the (sub)program from which D02NUF is called.

Constraints: NIA ≥ NEQ + 1 if JCEVAL = 'S', 'F' or 'D',
NIA ≥ 1 otherwise.

7: JA(NJA) – INTEGER array. *Input*

On entry: if JCEVAL = 'S' or 'F' or 'D', JA must contain details of the sparsity pattern to be used for the Jacobian. JA contains the row indices where non-zero elements occur, reading in columnwise order, and IA contains the starting locations in JA of the descriptions of columns 1,2,...,NEQ in that order, with IA(1) = 1. Thus for each column index j = 1,2,...,NEQ, the values of the row index i in column j where a non-zero element may occur are given by

$$i = \mathrm{JA}(k) \text{ where } \mathrm{IA}(j) \le k < \mathrm{IA}(j{+}1).$$

Thus the total number of non-zeros, NELEMENT, must be IA(NEQ+1)−1. For example, for the following matrix

$$\begin{pmatrix} x & 0 & x & 0 & 0 \\ 0 & x & x & x & 0 \\ x & x & x & 0 & 0 \\ x & 0 & 0 & x & x \\ 0 & 0 & 0 & x & x \end{pmatrix}$$

where x represents non-zero elements (13 in all) the arrays IA and JA should be

IA(k)	1	4	6	9	12	14							
JA(k)	1	3	4	2	3	1	2	3	2	4	5	4	5

JA is not used if JCEVAL = 'N' or 'A'.

8: NJA – INTEGER. *Input*

On entry: the dimension of the array JA as declared in the (sub)program from which D02NUF is called.

Constraints: NJA ≥ IA(NEQ+1) – 1 if JCEVAL = 'S', 'F' or 'D',
NJA ≥ 1 otherwise.

9: JACPVT(NJCPVT) – INTEGER array. *Workspace*

This must be the same array JACPVT as supplied to the integrator. It is used to pass information about the supplied sparsity structure to the integrator and therefore the contents of this array must not be changed before calling the integrator.

10: NJCPVT – INTEGER. *Input*

On entry: the length of the array JACPVT, which the user is supplying to the integrator, as dimensioned in the sub(program) from which D02NUF is called.

Suggested value: NJCPVT = 20×NEQMAX if JCEVAL = 'N' or 'A'. If NJCPVT is less than this estimate, then a message is printed on the current advisory message unit (see X04ABF), and execution continues.

Constraints: NJCPVT ≥ 3×NELEMENT + 14×NEQ if JCEVAL = 'S', 'F' or 'D',
where NELEMENT is the total number of non-zeros.

11: SENS – ***real.*** *Input*

On entry: a threshold parameter used to determine whether or not a matrix element is zero; when SENS is set to 0.0 on entry, the routine will use SENS = 100.0×***machine precision***. Otherwise the absolute value of SENS is used.

12: U – ***real.*** *Input*

On entry: U should have a value between 0.0 and 0.9999. Otherwise a default value of 0.1 is used. When the sparsity pattern has been evaluated, the first Jacobian computed is decomposed with U governing the choice of pivots; subsequent Jacobian decompositions use the same pattern of decomposition until the sparsity pattern is re-evaluated. When searching a row for a pivot, any element is excluded from the search which is less than U times the largest of those elements in the row available as pivots. Thus decreasing U biases the algorithm towards maintaining sparsity at the expense of numerical stability.

13: ETA – ***real.*** *Input*

On entry: a relative pivot threshold, below which on subsequent decompositions (as described under U above), an internal error is provoked. If ETA > 1.0 then no check on pivot size is made. If ETA ≤ 0.0 then the default value ETA = 1.0E–4 is used.

14: LBLOCK – LOGICAL. *Input*

On entry: indicates if preordering is used before decomposition.

If LBLOCK = .TRUE., on entry, the Jacobian matrix is preordered to block lower triangular form before a decomposition is performed (this is the recommended mode). If the user knows the structure of the Jacobian to be irreducible, that is not permutable to block lower triangular form, then the user should set LBLOCK = .FALSE.. For example, a Jacobian arising from using the method of lines for parabolic partial differential equations would normally be irreducible. (See the specification of D02NXF for optional output concerning LBLOCK.)

15: ISPLIT – INTEGER. *Input*

On entry: this parameter is used for splitting the integer workspace JACPVT to effect an efficient decomposition. It must satisfy 1 ≤ ISPLIT ≤ 99. If ISPLIT lies outside this range on entry, a default value of 73 is used. An appropriate value for ISPLIT for subsequent runs on similar problems is available via the optional output D02NXF.

Suggested value: ISPLIT = 73, unless the user has information from a previous run of a similar problem.

16: RWORK(50+4*NEQMAX) – ***real*** array. *Workspace*

This must be the same workspace array as the array RWORK supplied to the integrator. It is used to pass information from the setup routine to the integrator and therefore the contents of this array must not be changed before calling the integrator.

17: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, an illegal input was detected.

7. Accuracy

Not applicable.

8. Further Comments

This routine must be called as a setup routine before a call to either D02NDF or D02NJF and may be called as the linear algebra setup routine before a call to D02NMF or D02NNF.

9. Example

See the examples for D02NDF, D02NJF and D02NNF.

D02NVF – NAG Fortran Library Routine Document

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

1. Purpose

D02NVF is a setup routine which must be called by the user, prior to an integrator in the subchapter D02M-D02N, if Backward Differentiation Formulae (BDF) are to be used.

2. Specification

```
 SUBROUTINE D02NVF (NEQMAX, NY2DIM, MAXORD, METHOD, PETZLD, CONST,
1                   TCRIT, HMIN, HMAX, H0, MAXSTP, MXHNIL, NORM,
2                   RWORK, IFAIL)
 INTEGER        NEQMAX, NY2DIM, MAXORD, MAXSTP, MXHNIL, IFAIL
 real           CONST(6), TCRIT, HMIN, HMAX, H0,
1               RWORK(50+4*NEQMAX)
 LOGICAL        PETZLD
 CHARACTER*1    METHOD, NORM
```

3. Description

An integrator setup routine must be called before the call to any integrator in this subchapter. The setup routine D02NVF makes the choice of the BDF integrator and permits the user to define options appropriate to this choice.

4. References

See Subchapter Introduction.

5. Parameters

1: NEQMAX – INTEGER. *Input*

On entry: a bound on the maximum number of differential equations to be solved.

Constraint: NEQMAX $\geq$ 1.

2: NY2DIM – INTEGER. *Input*

On entry: the second dimension of the array YSAVE that will be supplied to the integrator, as declared in the (sub)program from which the integrator is called.

Constraint: NY2DIM $\geq$ MAXORD + 1.

3: MAXORD – INTEGER. *Input*

On entry: the maximum order to be used for the BDF method.

Constraint: 0 < MAXORD $\leq$ 5.

4: METHOD – CHARACTER*1. *Input*

On entry: specifies the method to be used to solve the system of nonlinear equations arising on each step of the BDF code. If METHOD = 'N', a modified Newton iteration is used. If METHOD = 'F', functional iteration is used. If METHOD = 'D', the modified Newton iteration is used.

Note: a linear algebra setup routine must be called even when using functional iteration, since if difficulty is encountered a switch is made to a modified Newton method.

Only the first character of the actual argument METHOD is passed to D02NVF; hence it is permissible for the actual argument to be more descriptive e.g. 'Newton', 'Functional iteration' or 'Default' in a call to D02NVF.

Constraint: METHOD = 'N', 'F' or 'D'.

5: PETZLD – LOGICAL. *Input*

On entry: specifies whether the Petzold local error test is to be used. If PETZLD is set to .TRUE. on entry, then the Petzold local error test is used, otherwise a conventional test is used. The Petzold test results in extra overhead cost but is more stable and reliable for differential/algebraic equations.

6: CONST(6) – ***real*** array. *Input/Output*

On entry: values to be used to control step size choice during integration. If any CONST(i) = 0.0 on entry, it is replaced by its default value described below. In most cases this is the recommended setting.

CONST(1), CONST(2), and CONST(3) are factors used to bound step size changes. If the current step size h fails, then the modulus of the next step size is bounded by CONST(1)×$|h|$. The default value of CONST(1) is 2.0. Note that the new step size may be used with a method of different order to the failed step. If the initial step size is h, then the modulus of the step size on the second step is bounded by CONST(3)×$|h|$. At any other stage in the integration, if the current step size is h, then the modulus of the next step size is bounded by CONST(2)×$|h|$. The default values are 10.0 for CONST(2) and 1000.0 for CONST(3).

CONST(4), CONST(5) and CONST(6) are 'tuning' constants used in determining the next order and step size. They are used to scale the error estimates used in determining whether to keep the same order of the BDF method, decrease the order or increase the order respectively. The larger the value of CONST(i), i = 4, 5, 6 the less likely the choice of the corresponding order. The default values are: CONST(4) = 1.2, CONST(5) = 1.3, CONST(6) = 1.4.

Constraints: the following contraints must be satisfied after any zero values have been replaced by their default values:

0.0 < CONST(1) < CONST(2) < CONST(3);
CONST(i) > 1.0, for i = 2,3,...,6.

On exit: the values actually used by the routine.

7: TCRIT – ***real***. *Input*

On entry: a point beyond which integration must not be attempted. The use of TCRIT is described under the parameter ITASK in the specification for the integrator. A value, 0.0 say, must be specified even if ITASK subsequently specifies that TCRIT will not be used.

8: HMIN – ***real***. *Input*

On entry: the minimum absolute step size to be allowed. Set HMIN = 0.0 if this option is not required.

9: HMAX – ***real***. *Input*

On entry: the maximum absolute step size to be allowed. Set HMAX = 0.0 if this option is not required.

10: H0 – ***real***. *Input*

On entry: the step size to be attempted on the first step. Set H0 = 0.0 if the initial step size is to be calculated internally.

11: MAXSTP – INTEGER. *Input*

On entry: the maximum number of steps to be attempted during one call to the integrator after which it will return with IFAIL = 2. Set MAXSTP = 0 if no limit is to be imposed.

12: MXHNIL – INTEGER. *Input*

On entry: the maximum number of warnings printed (if ITRACE ≥ 0) per problem when $t + h = t$ on a step (h = current step size). If MXHNIL ≤ 0, a default value of 10 is assumed.

13: NORM – CHARACTER*1. *Input*

On entry: indicates the type of norm to be used. Three options are available:

'M' maximum norm
'A' averaged L2 norm.
'D' is the same as 'A'

If VNORM denotes the norm of the vector v of length n, then for the averaged L2 norm

$$\text{VNORM} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(v_i/w_i)^2},$$

while for the maximum norm

$$\text{VNORM} = \max_i |v_i/w_i|.$$

If the user wishes to weight the maximum norm or the L2 norm, then RTOL and ATOL should be scaled appropriately on input to the integrator (see under ITOL in the specification of the integrator for the formulation of the weight vector w_i from RTOL and ATOL).

Only the first character to the actual argument NORM is passed to D02NVF; hence it is permissible for the actual argument to be more descriptive e.g. 'Maximum', 'Average L2' or 'Default' in a call to D02NVF.

Constraint: NORM = 'M', 'A' or 'D'.

14: RWORK(50+4*NEQMAX) – ***real*** array. *Workspace*

This must be the same workspace array as the array RWORK supplied to the integrator. It is used to pass information from the setup routine to the integrator and therefore the contents of this array must not be changed before calling the integrator.

15: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, an illegal input was detected.

7. Accuracy

Not applicable.

8. Further Comments

None.

9. Example

See the example for D02NBF.

D02NWF – NAG Fortran Library Routine Document

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

1. Purpose

D02NWF is a setup routine which must be called by the user, prior to an integrator in the D02M-D02N subchapter, if the BLEND formulae are to be used.

2. Specification

```
     SUBROUTINE D02NWF (NEQMAX, NY2DIM, MAXORD, CONST, TCRIT, HMIN, HMAX,
    1                   H0, MAXSTP, MXHNIL, NORM, RWORK, IFAIL)
     INTEGER          NEQMAX, NY2DIM, MAXORD, MAXSTP, MXHNIL, IFAIL
     real             CONST(6), TCRIT, HMIN, HMAX, H0,
    1                 RWORK(50+4*NEQMAX)
     CHARACTER*1      NORM
```

3. Description

An integrator setup routine (D02NVF, D02NWF or D02NVF are those available currently) must be called before the call to any integrator in this subchapter. The setup D02NWF makes the choice of the BLEND integrator and permits the user to define options appropriate to this choice.

4. References

See Subchapter Introduction.

5. Parameters

1: NEQMAX – INTEGER. *Input*

On entry: a bound on the maximum number of differential equations to be solved.

Constraint: NEQMAX $\geq$ 1.

2: NY2DIM – INTEGER. *Input*

On entry: the second dimension of the array YSAVE that will be supplied to the integrator, as declared in the (sub)program from which the integrator is called.

Constraint: NY2DIM $\geq$ MAXORD + 3.

3: MAXORD – INTEGER. *Input*

On entry: the maximum order to be used for the BLEND method.

Constraint: 0 < MAXORD $\leq$ 11.

4: CONST(6) – ***real*** array. *Input/Output*

On entry: values to be used to control stepsize choice during integration. If any CONST(i) = 0.0 on entry, it is replaced by its default value described below. In most cases this is the recommended setting.

CONST(1), CONST(2), and CONST(3) are factors used to bound stepsize changes. If the current stepsize h fails, then the modulus of the next stepsize is bounded by CONST(1)$\times|h|$. The default value of CONST(1) is 2.0. Note that the new stepsize may be used with a method of different order to the failed step. If the initial stepsize is h, then the modulus of the stepsize on the second step is bounded by CONST(3)$\times|h|$. At any other stage in the integration, if the current stepsize is h then the modulus of the next stepsize is bounded by CONST(2)$\times|h|$. The default values are 10.0 for CONST(2) and 1000.0 for CONST(3).

CONST(4), CONST(5) and CONST(6) are 'tuning' constants used in determining the next order and stepsize. They are used to scale the error estimates used in determining whether to keep the same order of the BLEND method, decrease the order or increase the order respectively. The larger the value of CONST(i), i = 4, 5, 6 the less likely the choice of the corresponding order. The default values are: CONST(4) = 1.2, CONST(5) = 1.3, CONST(6) = 1.4.

Constraints: the following contraints must be satisfied after any zero values have been replaced by their default values:
0.0 < CONST(1) < CONST(2) < CONST(3);
CONST(i) > 1.0 for i = 2,3,...,6.

On exit: the values actually used by the routine.

5: TCRIT – ***real***. *Input*

On entry: a point beyond which integration must not be attempted. The use of TCRIT is described under the parameter ITASK in the specification for the integrator. A value, 0.0 say, must be specified even if ITASK subsequently specifies that TCRIT will not be used.

6: HMIN – ***real***. *Input*

On entry: the minimum absolute stepsize to be allowed. Set HMIN = 0.0 if this option is not required.

7: HMAX – ***real***. *Input*

On entry: the maximum absolute stepsize to be allowed. Set HMAX = 0.0 if this option is not required.

8: H0 – ***real***. *Input*

On entry: the stepsize to be attempted on the first step. Set H0 = 0.0 if the initial stepsize is to be calculated internally.

9: MAXSTP – INTEGER. *Input*

On entry: the maximum number of steps to be attempted during one call to the integrator after which it will return with IFAIL = 2. Set MAXSTP = 0 if no limit is to be imposed.

10: MXHNIL – INTEGER. *Input*

On entry: the maximum number of warnings printed (if ITRACE ≥ 0) per problem when $t + h = t$ on a step (h = current stepsize). If MXHNIL ≤ 0, a default value of 10 is assumed.

11: NORM – CHARACTER*1. *Input*

On entry: indicates the type of norm to be used. Three options are available:

'M' maximum norm
'A' averaged L2 norm.
'D' is the same as 'A'

If VNORM denotes the norm of the vector v of length n, then for the averaged L2 norm

$$\text{VNORM} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(v_i/w_i)^2},$$

while for the maximum norm

$$\text{VNORM} = \max_i |v_i/w_i|.$$

If the user wishes to weight the maximum norm or the L2 norm, then RTOL and ATOL should be scaled appropriately on input to the integrator (see under ITOL in the specification of the integrator for the formulation of the weight vector w_i from RTOL and ATOL).

Only the first character of the actual argument NORM is passed to D02NWF; hence it is permissible for the actual argument to be more descriptive e.g. 'Maximum', 'Average L2' or 'Default' in a call to D02NWF.

Constraint: NORM = 'M', 'A' or 'D'.

12: RWORK(50+4*NEQMAX) – ***real*** array. *Workspace*

This must be the same workspace array as the array RWORK supplied to the integrator. It is used to pass information from the setup routine to the integrator and therefore the contents of this array must not be changed before calling the integrator.

13: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, an illegal input was detected.

7. Accuracy

Not applicable.

8. Further Comments

None.

9. Example

See the example for D02NCF.

D02NXF – NAG Fortran Library Routine Document

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

1. Purpose

D02NXF is an optional output routine which the user may call, on exit from an integrator in the D02M-D02N subchapter, if sparse matrix linear algebra has been selected.

2. Specification

```
      SUBROUTINE D02NXF (ICALL, LIWREQ, LIWUSD, LRWREQ, LRWUSD, NLU, NNZ,
     1                   NGP, ISPLIT, IGROW, LBLOCK, NBLOCK, INFORM)
      INTEGER         ICALL, LIWREQ, LIWUSD, LRWREQ, LRWUSD, NLU, NNZ,
     1                NGP, ISPLIT, IGROW, NBLOCK, INFORM(23)
      LOGICAL         LBLOCK
```

3. Description

This routine permits the user to examine the various outputs from the sparse linear algebra routines called by the integrator.

4. References

See Subchapter Introduction.

5. Parameters

1: ICALL – INTEGER. *Input*

On entry: indicates whether or not all output parameters have been set during the call to the integrator. If so, that is, if the integrator returned with IFAIL = 0 or 12, then ICALL must be set to 0. Otherwise ICALL must be set to 1, indicating that integration did not take place due to lack of space in arrays WKJAC and JACPVT, and only LIWREQ, LIWUSD, LRWREQ, LRWUSD have been set.

2: LIWREQ – INTEGER. *Output*

On exit: the length of the INTEGER workspace JACPVT reserved for the sparse matrix routines.

3: LIWUSD – INTEGER. *Output*

On exit: the length of the INTEGER workspace JACPVT actually used by the sparse matrix routines.

4: LRWREQ – INTEGER. *Output*

On exit: the length of the ***real*** workspace WKJAC reserved for the sparse matrix routines.

5: LRWUSD – INTEGER. *Output*

On exit: the length of the ***real*** workspace WKJAC actually used by the sparse matrix routines.

6: NLU – INTEGER. *Output*

On exit: the number of *LU* decompositions done during the integration.

7: NNZ – INTEGER. *Output*

On exit: the number of non-zeros in the Jacobian.

8: NGP – INTEGER. *Output*

On exit: the number of FCN or RESID calls needed to form the Jacobian.

9: ISPLIT – INTEGER. *Output*

On exit: an appropriate value for the parameter ISPLIT when calling D02NUF for subsequent runs of similar problems.

10: IGROW – INTEGER. *Output*

On exit: an estimate of the growth of the elements encountered during the last *LU* decomposition performed. If the actual estimate exceeds the largest possible integer value for the machine being used (see X02BBF) the IGROW is set to the value returned by X02BBF.

11: LBLOCK – LOGICAL. *Input*

On entry: the value used for the parameter LBLOCK when calling D02NUF.

12: NBLOCK – INTEGER. *Output*

On exit: if LBLOCK = .TRUE., NBLOCK contains the number of diagonal blocks in the Jacobian matrix permuted to block lower triangular form. If NBLOCK = 1 then on subsequent runs of a similar problem LBLOCK should be set to .FALSE. in the call to D02NUF. If LBLOCK = .FALSE., NBLOCK = 1.

13: INFORM(23) – INTEGER array. *Workspace*

This must be the same array as the array INFORM supplied to the integrator. It is used to pass information from the integrator to D02NXF and therefore its contents must not be changed before calling D02NXF.

6. Error Indicators and Warnings

None.

7. Accuracy

Not applicable.

8. Further Comments

The output from this routine, in particular the values of LIWREQ, LIWUSD, LRWREQ, LRWUSD, ISPLIT and IGROW, should be used to determine appropriate values for the parameters of the setup routine D02NUF on further calls to the integrator for the same or similar problems.

9. Example

See the examples for D02NDF, D02NJF and D02NNF.

D02NYF – NAG Fortran Library Routine Document

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

1. Purpose

D02NYF is an diagnostic routine which the user may call either after any user-specified exit or after a mid-integration error exit from any of the integrators in the D02M-D02N subchapter.

2. Specification

```
   SUBROUTINE D02NYF (NEQ, NEQMAX, HU, H, TCUR, TOLSF, RWORK, NST, NRE,
  1                   NJE, NQU, NQ, NITER, IMXER, ALGEQU, INFORM,
  2                   IFAIL)
   INTEGER          NEQ, NEQMAX, NST, NRE, NJE, NQU, NQ, NITER, IMXER,
  1                 INFORM(23), IFAIL
   real             HU, H, TCUR, TOLSF, RWORK(50+4*NEQMAX)
   LOGICAL          ALGEQU(NEQ)
```

3. Description

This routine permits the user to inspect statistics produced by any integrator in this subchapter. These statistics concern the integration only.

4. References

See Subchapter Introduction.

5. Parameters

1: NEQ – INTEGER. *Input*

On entry: the value used for the parameter NEQ when calling the integrator.

Constraint: NEQ ≥ 1.

2: NEQMAX – INTEGER. *Input*

On entry: the value used for the parameter NEQMAX when calling the integrator.

Constraint: NEQMAX ≥ NEQ.

3: HU – ***real.*** *Output*

On exit: the last successful stepsize.

4: H – ***real.*** *Output*

On exit: the proposed next stepsize for continuing the integration.

5: TCUR – ***real.*** *Output*

On exit: the value of the independent variable, t, which the integrator has actually reached. TCUR will always be at least as far as the output value of the argument t in the direction of integration, but may be further (if overshooting and interpolation at TOUT was specified).

6: TOLSF – ***real.*** *Output*

On exit: a tolerance scale factor, TOLSF ≥ 1.0, which is computed when a request for too much accuracy is detected by the integrator (indicated by a return with IFAIL = 3 or IFAIL = 14). If ITOL is left unaltered but RTOL and ATOL are uniformly scaled up by a factor of TOLSF the next call to the integrator is deemed likely to succeed.

7: RWORK(50+4*NEQMAX) – ***real*** array. *Workspace*

This must be the same workspace array as the array RWORK supplied to the integrator. It is used to pass information from the integrator to D02NYF and therefore the contents of this array must not be changed before calling D02NYF.

8: NST – INTEGER. *Output*

On exit: the number of steps taken in the integration so far.

9: NRE – INTEGER. *Output*

On exit: the number of function or residual evaluations (FCN or RESID calls) used in the integration so far.

10: NJE – INTEGER. *Output*

On exit: the number of Jacobian evaluations used in the integration so far. This equals the number of matrix *LU* decompositions.

11: NQU – INTEGER. *Output*

On exit: the order of the method last used (successfully) in the integration.

12: NQ – INTEGER. *Output*

On exit: the proposed order of the method for continuing the integration.

13: NITER – INTEGER. *Output*

On exit: the number of iterations performed in the integration so far by the nonlinear equation solver.

14: IMXER – INTEGER. *Output*

On exit: the index of the component of largest magnitude in the weighted local error vector (e_i/w_i), for i = 1,2,...,NEQ.

15: ALGEQU(NEQ) – LOGICAL array. *Output*

On exit: ALGEQU(i) = .TRUE. if the ith equation integrated was detected to be algebraic, otherwise ALGEQU(i) = .FALSE.. Note that when the integrators for explicit equations are being used, then ALGEQU(i) = .FALSE., for i = 1,2,...,NEQ.

16: INFORM(23) – INTEGER array. *Workspace*

This must be the same array as the array INFORM supplied to the integrator. It is used to pass information from the integrator to D02NYF and therefore its contents must not be changed before calling D02NYF.

17: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, NEQ < 1,
or NEQMAX < 1,
or NEQ > NEQMAX.

7. Accuracy

Not applicable.

8. Further Comments

Statistics for sparse matrix linear algebra calls (if appropriate) may be determined by a call to D02NXF.

9. Example

See the example for D02NBF.

D02NZF – NAG Fortran Library Routine Document

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

1. Purpose

D02NZF is a setup routine which must be called, if optional inputs need resetting, prior to a continuation call to any of the integrators in the D02M-D02N subchapter.

2. Specification

```
      SUBROUTINE D02NZF (NEQMAX, TCRIT, H, HMIN, HMAX, MAXSTP, MXHNIL,
     1                   RWORK, IFAIL)
      INTEGER            NEQMAX, MAXSTP, MXHNIL, IFAIL
      real               TCRIT, H, HMIN, HMAX, RWORK(50+4*NEQMAX)
```

3. Description

This routine is provided to permit the user to reset many of the parameters which control the integration 'on the fly', that is in conjunction with the interrupt facility permitted through the parameter ITASK of the integrator. In addition to a number of parameters which the user can set initially through one of the integrator setup routines, the stepsize to be attempted on the next step may be changed.

4. References

See Subchapter Introduction.

5. Parameters

1: NEQMAX – INTEGER. *Input*

On entry: the value used for the parameter NEQMAX when calling the integrator.

Constraint: NEQMAX $\geq$ 1.

2: TCRIT – ***real.*** *Input*

On entry: a point beyond which integration must not be attempted. The use of TCRIT is described under the parameter ITASK in the specification for the integrator. A value, 0.0 say, must be specified even if ITASK subsequently specifies that TCRIT will not be used.

3: H – ***real.*** *Input*

On entry: the next stepsize to be attempted. Set H = 0.0 if the current value of H is not to be changed.

4: HMIN – ***real.*** *Input*

On entry: the minimum absolute stepsize to be allowed. Set HMIN = 0.0 if this option is not required. Set HMIN < 0.0 if the current value of HMIN is not to be changed.

5: HMAX – ***real.*** *Input*

On entry: the maximum absolute stepsize to be allowed. Set HMAX = 0.0 if this option is not required. Set HMAX < 0.0 if the current value of HMAX is not to be changed.

6: MAXSTP – INTEGER. *Input*

On entry: the maximum number of steps to be attempted during one call to the integrator after which it will return with IFAIL = 2. Set MAXSTP = 0 if this option is not required. Set MAXSTP < 0 if the current value of MAXSTP is not to be changed.

7: MXHNIL – INTEGER. *Input*

On entry: the maximum number of warnings printed (if ITRACE ≥ 0) per problem when $t + h = t$ on a step (h = current stepsize). If MXHNIL ≤ 0, a default value of 10 is assumed.

8: RWORK(50+4*NEQMAX) – ***real*** array. *Workspace*

This must be the same workspace array as the array RWORK supplied to the integrator. It is used to pass information from the integrator to D02NZF and therefore its contents must not be changed before calling D02NZF.

9: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

NEQMAX < 1.

7. Accuracy

Not applicable.

8. Further Comments

None.

9. Example

See the example for D02NCF.

End of Subchapter D02M-D02N