NAG Library Routine Document

D02PQF

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

D02PQF is a setup routine which must be called prior to the first call of either of the integration routines D02PEF, D02PFF and D02PGF.

2 Specification

```
SUBROUTINE D02PQF (N, TSTART, TEND, YINIT, TOL, THRESH, METHOD, HSTART,    &
                   IWSAV, RWSAV, IFAIL)

INTEGER            N, METHOD, IWSAV(130), IFAIL
REAL (KIND=nag_wp) TSTART, TEND, YINIT(N), TOL, THRESH(N), HSTART,         &
                   RWSAV(32*N+350)
```

3 Description

D02PQF and its associated routines (D02PEF, D02PFF, D02PGF, D02PHF, D02PJF, D02PRF, D02PSF, D02PTF and D02PUF) solve the initial value problem for a first-order system of ordinary differential equations. The routines, based on Runge–Kutta methods and derived from RKSUITE (see Brankin *et al.* (1991)), integrate

$$y' = f(t, y) \quad \text{given} \quad y(t_0) = y_0$$

where y is the vector of n solution components and t is the independent variable.

The integration proceeds by steps from the initial point t_0 towards the final point t_f. An approximate solution y is computed at each step. For each component y_i, for $i = 1, 2, \ldots, n$, the error made in the step, i.e., the local error, is estimated. The step size is chosen automatically so that the integration will proceed efficiently while keeping this local error estimate smaller than a tolerance that you specify by means of arguments TOL and THRESH.

D02PEF can be used to solve the 'usual task', namely integrating the system of differential equations to obtain answers at points you specify. D02PFF is used for more 'complicated' tasks where $f(t, y)$ can readily be coded within a routine argument and high-order interpolation is not required. D02PGF is used for the most 'complicated' tasks where $f(t, y)$ is best evaluated outside the integrator or where high-order interpolation is required.

You should consider carefully how you want the local error to be controlled. Essentially the code uses relative local error control, with TOL being the desired relative accuracy. For reliable computation, the code must work with approximate solutions that have some correct digits, so there is an upper bound on the value used for TOL. It is impossible to compute a numerical solution that is more accurate than the correctly rounded value of the true solution, so you are not allowed to specify TOL too small for the precision you are using. The magnitude of the local error in y_i on any step will not be greater than TOL $\times \max(\mu_i, \text{THRESH}(i))$ where μ_i is an average magnitude of y_i over the step. If THRESH(i) is smaller than the current value of μ_i, this is a relative error test and TOL indicates how many significant digits you want in y_i. If THRESH(i) is larger than the current value of μ_i, this is an absolute error test with tolerance TOL $\times$ THRESH(i). Relative error control is the recommended mode of operation, but pure relative error control, THRESH$(i) = 0.0$, is not permitted. See Section 9 for further information about error control.

D02PEF, D02PFF and D02PGF control local error rather than the true (global) error, the difference between the numerical and true solution. Control of the local error controls the true error indirectly. Roughly speaking, the code produces a solution that satisfies the differential equation with a discrepancy bounded in magnitude by the error tolerance. What this implies about how close the

numerical solution is to the true solution depends on the stability of the problem. Most practical problems are at least moderately stable, and the true error is then comparable to the error tolerance. To judge the accuracy of the numerical solution, you could reduce TOL substantially, e.g., use $0.1 \times$ TOL, and solve the problem again. This will usually result in a rather more accurate solution, and the true error of the first integration can be estimated by comparison. Alternatively, a global error assessment can be computed automatically using the argument METHOD. Because indirect control of the true error by controlling the local error is generally satisfactory and because both ways of assessing true errors cost twice, or more, the cost of the integration itself, such assessments are used mostly for spot checks, selecting appropriate tolerances for local error control, and exploratory computations.

D02PEF, D02PFF and D02PGF each implement three Runge–Kutta formula pairs, and you must select one for the integration. The best choice for METHOD depends on the problem. The order of accuracy is 3, 5 and 8 respectively. As a rule, the smaller TOL is, the larger you should take the order of the METHOD. If the components THRESH are small enough that you are effectively specifying relative error control, experience suggests

TOL	efficient METHOD
$10^{-2} - 10^{-4}$	order 2 and 3 pair
$10^{-3} - 10^{-6}$	order 4 and 5 pair
$10^{-5} -$	order 7 and 8 pair

The overlap in the ranges of tolerances appropriate for a given METHOD merely reflects the dependence of efficiency on the problem being solved. Making TOL smaller will normally make the integration more expensive. However, in the range of tolerances appropriate to a METHOD, the increase in cost is modest. There are situations for which one METHOD, or even this kind of code, is a poor choice. You should not specify a very small value for THRESH(i), when the ith solution component might vanish. In particular, you should not do this when $y_i = 0.0$. If you do, the code will have to work hard with any value for METHOD to compute significant digits, but the lowest order method is a particularly poor choice in this situation. All three methods are inefficient when the problem is 'stiff'. If it is only mildly stiff, you can solve it with acceptable efficiency with the order 2 and 3 pair, but if it is moderately or very stiff, a code designed specifically for such problems will be much more efficient. The higher the order the more smoothness is required of the solution in order for the method to be efficient.

When assessment of the true (global) error is requested, this error assessment is updated at each step. Its value can be obtained at any time by a call to D02PUF. The code monitors the computation of the global error assessment and reports any doubts it has about the reliability of the results. The assessment scheme requires some smoothness of $f(t, y)$, and it can be deceived if f is insufficiently smooth. At very crude tolerances the numerical solution can become so inaccurate that it is impossible to continue assessing the accuracy reliably. At very stringent tolerances the effects of finite precision arithmetic can make it impossible to assess the accuracy reliably. The cost of this is roughly twice the cost of the integration itself with the 5th and 8th order methods, and three times with the 3rd order method.

The first step of the integration is critical because it sets the scale of the problem. The integrator will find a starting step size automatically if you set the argument HSTART to 0.0. Automatic selection of the first step is so effective that you should normally use it. Nevertheless, you might want to specify a trial value for the first step to be certain that the code recognizes the scale on which phenomena occur near the initial point. Also, automatic computation of the first step size involves some cost, so supplying a good value for this step size will result in a less expensive start. If you are confident that you have a good value, provide it via the argument HSTART.

4 References

Brankin R W, Gladwell I and Shampine L F (1991) RKSUITE: A suite of Runge–Kutta codes for the initial value problems for ODEs *SoftReport 91-S1* Southern Methodist University

5 Arguments

1: N – INTEGER *Input*

On entry: n, the number of ordinary differential equations in the system to be solved by the integration routine.

Constraint: N ≥ 1.

2: TSTART – REAL (KIND=nag_wp) *Input*

On entry: the initial value of the independent variable, t_0.

3: TEND – REAL (KIND=nag_wp) *Input*

On entry: the final value of the independent variable, t_f, at which the solution is required. TSTART and TEND together determine the direction of integration.

Constraint: TEND must be distinguishable from TSTART for the method and the precision of the machine being used.

4: YINIT(N) – REAL (KIND=nag_wp) array *Input*

On entry: y_0, the initial values of the solution, y_i, for $i = 1, 2, \ldots, n$.

5: TOL – REAL (KIND=nag_wp) *Input*

On entry: a relative error tolerance. The actual tolerance used is $\max(10 \times$ ***machine precision***$, \min(\text{TOL}, 0.01))$; that is, the minimum tolerance is set at 10 times ***machine precision*** and the maximum tolerance is set at 0.01.

6: THRESH(N) – REAL (KIND=nag_wp) array *Input*

On entry: a vector of thresholds. For the ith component, the actual threshold used is $\max\left(\sqrt{saferange}, \text{THRESH}(i)\right)$, where $saferange$ is the value returned by X02AMF.

7: METHOD – INTEGER *Input*

On entry: the Runge–Kutta method to be used.

METHOD $= 1$ or -1
A $2(3)$ pair is used.

METHOD $= 2$ or -2
A $4(5)$ pair is used.

METHOD $= 3$ or -3
A $7(8)$ pair is used.

Constraint: METHOD $= 1$, -1, 2, -2, 3 or -3. Note: if METHOD > 0 then global error assessment is to be computed with the main integration; if METHOD < 0 then global error assessment will not be computed.

8: HSTART – REAL (KIND=nag_wp) *Input*

On entry: a value for the size of the first step in the integration to be attempted. The absolute value of HSTART is used with the direction being determined by TSTART and TEND. The actual first step taken by the integrator may be different to HSTART if the underlying algorithm determines that HSTART is unsuitable. If HSTART $= 0.0$ then the size of the first step is computed automatically.

Suggested value: HSTART $= 0.0$.

9: IWSAV(130) – INTEGER array *Communication Array*

10: RWSAV($32 \times \mathrm{N} + 350$) – REAL (KIND=nag_wp) array *Communication Array*

On exit: the contents of the communication arrays must not be changed prior to calling one of the integration routines.

11: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation for details.

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

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

6 Error Indicators and Warnings

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

Errors or warnings detected by the routine:

IFAIL $= 1$

On entry, METHOD $= \langle value \rangle$.
Constraint: METHOD $= -3$, -2, -1, 1, 2 or 3.

On entry, N $= \langle value \rangle$.
Constraint: N ≥ 1.

On entry, too much workspace required.
Workspace provided was $\langle value \rangle$, workspace required is $\langle value \rangle$.

On entry, TSTART $= \langle value \rangle$.
Constraint: TSTART $\neq$ TEND.

On entry, TSTART is too close to TEND.
$|\mathrm{TSTART} - \mathrm{TEND}| = \langle value \rangle$, but this quantity should be at least $\langle value \rangle$.

IFAIL $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

See Section 3.9 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -399$

Your licence key may have expired or may not have been installed correctly.

See Section 3.8 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.7 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

Not applicable.

8 Parallelism and Performance

D02PQF is not threaded in any implementation.

9 Further Comments

If D02PFF and D02PGF is to be used for the integration then the value of the argument TEND may be reset during the integration without the overhead associated with a complete restart; this can be achieved by a call to D02PRF.

It is often the case that a solution component (the ith, say) is of no interest when it is smaller in magnitude than a certain threshold. You can inform the code of this by setting the ith component of THRESH to this threshold. In this way you avoid the cost of computing significant digits in the ith component of y when it is smaller than the threshold of interest. This matter is important when a component of y vanishes, and in particular, when the initial value is zero. An appropriate threshold depends on the general size of y in the course of the integration. Physical reasoning may help you select suitable threshold values. If you do not know what to expect of y, you can find out by a preliminary integration using D02PEF with nominal values of THRESH. As D02PEF integrates by steps in time, it stores, for each component, the largest magnitude of solution computed so far; these values are output in the array YMAX. This can help determine more appropriate values for THRESH for an accurate integration. For example, the values in THRESH could be set to $10 \times$ ***machine precision*** times the final value of YMAX.

10 Example

See Section 10 in D02PEF, D02PFF, D02PRF, D02PSF and D02PUF.