NAG Library Routine Document

D02PVF

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

D02PVF is a setup routine which must be called prior to the first call of either of the integration routines D02PCF and D02PDF.

2 Specification

```
 SUBROUTINE D02PVF(NEQ, TSTART, YSTART, TEND, TOL, THRES, METHOD, TASK,
1                  ERRASS, HSTART, WORK, LENWRK, IFAIL)
 INTEGER           NEQ, METHOD, LENWRK, IFAIL
 double precision  TSTART, YSTART(NEQ), TEND, TOL, THRES(NEQ), HSTART,
1                  WORK(LENWRK)
 LOGICAL           ERRASS
 CHARACTER*1       TASK
```

3 Description

D02PVF and its associated routines (D02PCF, D02PDF, D02PWF, D02PXF, D02PYF and D02PZF) 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

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

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 parameters TOL and THRES.

D02PCF can be used to solve the 'usual task', namely integrating the system of differential equations to obtain answers at points you specify. D02PDF is used for all more 'complicated tasks'.

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 you can specify 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 $\times \max(\mu_i, \text{THRES}(i))$ where μ_i is an average magnitude of y_i over the step. If THRES(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 THRES(i) is larger than the current value of μ_i, this is an absolute error test with tolerance TOL $\times$ THRES(i). Relative error control is the recommended mode of operation, but pure relative error control, THRES$(i) = 0.0$, is not permitted. See Section 8 for further information about error control.

D02PCF and D02PDF 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 parameter ERRASS. 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.

D02PCF and D02PDF 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 value of METHOD. If the components THRES are small enough that you are effectively specifying relative error control, experience suggests

TOL	efficient METHOD
$10^{-2} - 10^{-4}$	1
$10^{-3} - 10^{-6}$	2
$10^{-5} -$	3

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 THRES(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 METHOD $= 1$ 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 METHOD $= 1$, but if it is moderately or very stiff, a code designed specifically for such problems will be much more efficient. The higher the order, i.e., the larger the value of METHOD, 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 D02PZF. 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 METHOD $= 2$ or 3, and three times with METHOD $= 1$.

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 parameter 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 parameter 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 Parameters

1: NEQ – INTEGER *Input*

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

Constraint: NEQ ≥ 1.

2: TSTART – ***double precision*** *Input*

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

3: YSTART(NEQ) – ***double precision*** array *Input*

On entry: y_0, the initial values of the solution, y_i, for $i = 1, 2, \ldots n$, at t_0.

4: TEND – ***double precision*** *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.

5: TOL – ***double precision*** *Input*

On entry: a relative error tolerance.

Constraint: $10.0 \times$ ***machine precision*** $\leq$ TOL ≤ 0.01.

6: THRES(NEQ) – ***double precision*** array *Input*

On entry: a vector of thresholds.

Constraint: THRES$(i) \geq \sqrt{\sigma}$, where σ is approximately the smallest possible machine number that can be reciprocated without overflow (see X02AMF).

7: METHOD – INTEGER *Input*

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

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

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

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

Constraint: METHOD $= 1$, 2 or 3.

8: TASK – CHARACTER*1 *Input*

On entry: determines whether the usual integration task is to be performed using D02PCF or a more complicated task is to be performed using D02PDF.

TASK = 'U'
D02PCF is to be used for the integration.

TASK = 'C'
D02PDF is to be used for the integration.

Constraint: TASK = 'U' or 'C'.

9: ERRASS – LOGICAL *Input*

On entry: specifies whether a global error assessment is to be computed with the main integration. ERRASS = .TRUE. specifies that it is.

10: HSTART – ***double precision*** *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.

11: WORK(LENWRK) – ***double precision*** array *Output*

On exit: contains information for use by D02PCF or D02PDF. This **must** be the same array as supplied to D02PCF or D02PDF. The contents of this array must remain unchanged between calls.

12: LENWRK – INTEGER *Input*

On entry: the dimension of the array WORK as declared in the (sub)program from which D02PVF is called. (LENWRK $\geq 32 \times$ NEQ is always sufficient.)

Constraints:

if TASK = 'U' and ERRASS = .FALSE.,
  if METHOD = 1, LENWRK $\geq 10 \times$ NEQ;
  if METHOD = 2, LENWRK $\geq 20 \times$ NEQ;
  if METHOD = 3, LENWRK $\geq 16 \times$ NEQ;

if TASK = 'U' and ERRASS = .TRUE.,
  if METHOD = 1, LENWRK $\geq 17 \times$ NEQ;
  if METHOD = 2, LENWRK $\geq 32 \times$ NEQ;
  if METHOD = 3, LENWRK $\geq 21 \times$ NEQ;

if TASK = 'C' and ERRASS = .FALSE.,
  if METHOD = 1, LENWRK $\geq 10 \times$ NEQ;
  if METHOD = 2, LENWRK $\geq 14 \times$ NEQ;
  if METHOD = 3, LENWRK $\geq 16 \times$ NEQ;

if TASK = 'C' and ERRASS = .TRUE.,
  if METHOD = 1, LENWRK $\geq 15 \times$ NEQ;
  if METHOD = 2, LENWRK $\geq 26 \times$ NEQ;
  if METHOD = 3, LENWRK $\geq 21 \times$ NEQ.

13: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this parameter you should refer to Section 3.3 in the Essential Introduction for details.

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

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

6 Error Indicators and Warnings

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

Errors or warnings detected by the routine:

IFAIL = 1

On entry, NEQ < 1,
or TEND is too close to TSTART,
or TOL > 0.01 or TOL $< 10.0 \times$ ***machine precision***,
or THRES$(i) < \sqrt{\sigma}$, where σ is approximately the smallest possible machine number that can be reciprocated without overflow (see X02AMF),
or METHOD $\neq$ 1, 2 or 3,
or TASK $\neq$ 'U' or 'C',
or LENWRK is too small.

7 Accuracy

Not applicable.

8 Further Comments

If TASK = 'C' then the value of the parameter TEND may be reset during the integration without the overhead associated with a complete restart; this can be achieved by a call to D02PWF.

It is often the case that a solution component y_i is of no interest when it is smaller in magnitude than a certain threshold. You can inform the code of this by setting THRES(i) to this threshold. In this way you avoid the cost of computing significant digits in y_i when only the fact that it is smaller than the threshold is of interest. This matter is important when y_i vanishes, and in particular, when the initial value YSTART(i) vanishes. An appropriate threshold depends on the general size of y_i 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 D02PCF with nominal values of THRES. As D02PCF steps from t_0 towards t_f for each $i = 1, 2, \ldots, n$ it forms YMAX(i), the largest magnitude of y_i computed at any step in the integration so far. Using this you can determine more appropriate values for THRES for an accurate integration. You might, for example, take THRES(i) to be $10.0 \times$ ***machine precision*** times the final value of YMAX(i).

9 Example

See Section 9 in D02PCF, D02PDF, D02PXF, D02PWF and D02PZF.