14.2 Explicit Runge-Kutta Method for Ordinary Differential Equations

A. Purpose

These routines solve

$$\left.\begin{aligned} \frac{dy_i}{dt} &= f_i(t, y_1, y_2, ..., y_{NEQ}) \\ y_i(t_0) &= \eta_i \end{aligned}\right\}, \; i = 1, 2, ..., NEQ. \quad (1)$$

using an 8^{th} order explicit Runge-Kutta method. Other than lacking a built in facility for saving the solution, it has much the same functionality as the variable order Adams codes in Chapter 14.1. Except for crude tolerances, it requires significantly more function evaluations to get a given accuracy. It has the advantages of being slightly easier to use (?), requiring less internal overhead (and thus is faster for derivative functions that are very cheap to evaluate), and being able to generate slightly more accurate results.

B. Usage

Described below under B.1 through B.6 are:

B.1 Program Prototype, Double Precision

B.1.a The calling routine

The dimensioning parameters must satisfy the following constraints. We use i_k as an abbreviation for IDAT(k), which is defined further below. OPT has a dimension that depends on the options specified. The subroutine arguments should be declared as follows:

INTEGER IDAT(i_5)

DOUBLE PRECISION TS(≥ 3), **Y**($\geq i_4$), **OPT**(??), **DAT**(i_6), **WORK**(i_7)

To obtain results integrating from TS(1) to TS(3) with the default error control (see XRKRE1=3 in Section B.2), assign values to TS, Y, OPT(1) (=0.), IDAT(1) (=0), IDAT(4) (=NEQ), IDAT(5:7) (= i_5, i_6, i_7), and

```
100 CALL DXRK8(TS, Y, OPT, IDAT,
      DAT, WORK)
    if (IDAT(1) .ne. 3) go to 100
```

(Here the computation is complete.)

B.1.b Program Prototype, DXRK8F, Double Precision

The user supplied subroutine, DXRK8F, provides values of the derivative, given values of the independent and dependent variables.

```
SUBROUTINE DXRK8F(TS, Y, F, IDAT)
DOUBLE PRECISION TS(*), Y(*), F(*)
INTEGER IDAT(*)
Compute F(i) = f_i(TS(1), Y), i = 1, 2, ..., NEQ.
RETURN
END
```

The use of options 25 or 26 described in Section B.4. requires additional actions in DXRK8F.

B.1.c Program Prototype, DXRK8O, Double Precision

The user supplied subroutine, DXRK8O, provides output of results, and may set flags to inform the integrator of special situations.

```
SUBROUTINE DXRK8O(TS, Y, IDAT, DAT)
DOUBLE PRECISION TS(*), Y(*), DAT(*)
INTEGER IDAT(*)
Output results depending on the value of
  IDAT(1:3), see Section B.3 for details.
RETURN
END
```

B.1.d Argument Definitions

TS() [inout] Three locations in this array are used as follows. All of these must be initialized before the first call.

TS(1) = current value of t, the independent variable.

TS(2) = current value of h, the integration stepsize. If the initial value is 0, a starting stepsize is selected automatically. A new value for TS(2) is selected by the integrator at the conclusion of each step.

TS(3) = the value of t_f, the final output point. No derivatives are computed past t_f. The user output routine DXRK8O is called with IDAT(1) = 30 at $t = t_f$. If TS(3) is not changed at this time a return

is made with IDAT(1)=3, and IDAT(2) = 0. Otherwise the integration continues with a new value for t_f. Using options that give output at interpolated values is a better way to get output at multiple points since the code may be forced to take a small stepsize in order to integrate exactly up to TS(3).

Y() [inout] Vector of dependent variables, **y**, with initial values provided by the user, and updated by the integrator.

OPT() [inout] A vector providing access to optional features of the integrator. Defaults are such that you need not read about any options except those which you may have an interest in. Option 0 must be the last option specified and is the only one required. Many of the options take arguments which follow immediately after the option. The next option follows after the last argument for the previous option. See Section B.2 for details and the following table for a quick overview.

Options	Brief description
0	No more options.
1	Specify equation groups for options 2–10.
2–7	Various types of error control.
8	Control of diagnostic output.
9	Specify the **y**'s interpolated to.
10	Allows for very slightly better accuracy (?).
11	Not used.
12–14	Reverse communication options.
15–16	Set maximum and minimum stepsize.
17	Specify floating point precision.
18–19	Not used.
20	Max. steps between output points.
21	Give output at end of every step.
22	Specify interpolation point(s).
23	Specify end of step output past given point.
24	Specify $t + k \times \Delta t$ output points.
25–26	G-Stops (zeros of arbitrary functions).
27	Change criteria used for stepsize control.
28	Specify range allowed when interpolating.

IDAT() [inout] IDAT(1:3) are used to communicate information about the state of the integration to and from the integrator. The brief summary given here is adequate if you are not concerned with details of what is going on and are not doing anything fancy. See Section B.3 for more details. IDAT(1) gives a high level indication of what is going on as follows.

- 0 Starting an integration, must be set when starting a new integration.
- 1 Computing derivatives. Set on calls to DXRK8F.
- 2 Computing G's for G-Stops. Set on calls to DXRK8F.
- 3 Some kind of termination condition. If you aren't doing anything fancy the integration has reached the endpoint with no problems.
- 4 Some kind of error, see "Error Procedures and Restrictions" for details.
- ≥20 Some kind of output point has been reached. Set on calls to DXRK8O.

IDAT(4) must be set to NEQ, the number of differential equations in the system.

IDAT(5) the dimension of IDAT. If you have used no options then 49 is a big enough number. Else, let I8 = number of times option 8 (XRKDIG) has been turned "on", and I10 be defined similarly for option 10 (XRKXP). Let IE be the number of times options 2–7 (error control) have been specified (including the implied specification for the first group of equations if an explicit specification is not given) and IO the number of options for I/O (21 to 24) that have been specified. Then one should have IDAT(5) $\geq$ 54 + 2 $\times$ (I8+I10+IO) + 3 $\times$ IE. The actual space needed may be a bit smaller. If you choose a number too small you will get a diagnostic giving the size needed. If you use a negative value for IDAT, it will behave as if you had provided the negative of that value, except it will replace IDAT(5) with the smallest value that you could use.

IDAT(6) As for IDAT(5) (including negative values), except gives the dimension of DAT. IDAT(6) $\geq$ 30+2$\times$IDAT(4)+ space for all error tolerances + space for defining t output values and delta t for option 24 (XRKTDT) + if using extra precision, 1 for t and for each y that is kept in extended precision.

IDAT(7) As for IDAT(5), except for WORK. Define LWI = 0 if there is no interpolation being done to arbitrary points, and otherwise LWI = 6 + 7 $\times$ IDAT(4). Make IDAT(7) $\geq 10 \times$ IDAT(4) + LWI.

The rest of IDAT is used by the integrator for internal working space. Comments in the source code describe how this space is used.

DAT() [inout] Used by the integrator for internal working space. If there is nothing special, this must have a dimension of at least 30 + 2*NEQ. Some options require additional space in DAT. These options must start at a location > 30 + 2*NEQ, and any such space must start at a location > than the last space used by a previous option.

WORK() [inout] Data that need not be saved from one step to the next if the integration is interrupted for

another task. This data must be preserved if the integration is interrupted by some other means than by setting IDAT(1) = −2 when IDAT(1) ≥ 20.

F() [out] (Returned by users subprogram DXRK8F to DXRK8.) The user should compute $f_i(t, \mathbf{y})$ and store it in F(i), i = 1, 2,..., NEQ, whenever DXRK8F is called with IDAT(1) = 1. (F is part of the array WORK passed into DXRK8.)

B.2 Setting options using the array OPT()

All options are specified using the array OPT. Options are indicated with a parameter name, an "=", and a floating point integer followed by 0 or more arguments. The arguments are indicated in parentheses in the order in which they must be given. If an argument is followed by "=...", the "..." gives the default value for this option. Following the arguments for one option, one specifies the next option. Options are indicated with integers below, even though they must of course be supplied as floating point numbers. If an option is supposed to have an integer value, the nearest integer to the floating point value supplied is used. For clarity in specifying these options one may want to use the names for these parameters. The code gives declarations for defining these parameters. Other parameters used for accessing IDAT and DAT are defined there also. If one is accessing these arrays we recommend that it be done using these parameters.

XRKEND=0 End of options.

Begin Options For Groups of Equations

You need not use any groups, and options will just be treated as if there were one group that contains all the equations. Options discussed in this paragraph apply to a group of equations. If an option does not appear in the first group a default action is taken until that option is specified. One can imagine that there is an implicit group starting with equation index 0 prior to the first group, and an implicit group ending with equation index NEQ + 1, which effectively says that the previous group ends with equation index NEQ. A silly example can illustrate how this works. Suppose we have 6 equations and when we interpolate we only want to interplate for equations 1,2, and 5. The option index for setting a group is 1, that for interpolation is 9, and in this case a 0 says you don't want the action and a 1 that you do. the options vector would then look like (... 1 3 ... 9 0 ... 1 5 ... 9 1 ... 1 6 ... 9 0 ...). We could have started this with a (9 1), but that is not necessary as the default is to interpolate for everything. Of course other "1" options and other options may be specified interlaced with what is given. Error tolerances are always on (unless one says there is no error tolerance), and if one is not given prior to the first group a default error tolerance is used.

XRKEQG=1 (N1=1) Index of first equation in a group of equations. This option is only necessary if one wants to treat groups of equations differently in some way. If this option is specified more than once, the values of N1 must be strictly increasing.

Begin Options for Error Control

See Section D for a discussion of how the error control works. In brief, for the i^{th} equation, an error tolerance, τ_i is defined by the options below, and third and fifth order error estimates, $e_{3,i}$ and $e_{5,i}$ are available from formulas use by the code. This data is used to generate an order 8 estimate of the current error.

XRKAE=2 (A2) For the group in which this error tolerance applies, τ_i (see just above) is A2.

XRKRE1=3 (A3=$\epsilon^{.75}$, B3=$\epsilon^{.75}$) τ_i^2 is A3^2 + B3^2 × (sum in group of y_i^2). Note that this is the default error test with A3 = B3 = $\epsilon^{.75}$, where ϵ is the smallest number > 0 such that the floating point addition $1 + \epsilon$ gives a result different from 1.

XRKRE2=4 (A4, B4) τ_i^2 is A4^2 + (B4^2 × y_i^2).

XRKAEV=5 (A5) Similar to option 2, except "A2" is now a vector, with values stored starting in DAT(A5), where A5 > DAT(LOCTY) + 2 × NEQ. (The parameter LOCTY has the value 30.) This has the same effect as option 2 followed by option 1 with N1 increased by 1 each time, for the number of times there are equations in this set. Space used in DAT for storing this kind of error control information should be contiguous and specified with increasing values of A5.

XRKREV=6 (A6) As for option 5, except applied similarly to option 4. Thus there are values ("$a-1$", "r_1", "a_2", ...) stored starting at DAT(A6). If both options 5 and 6 are used, the space used by the vectors for option 6 must not overlap with the space used by vectors for 5.

XRKNOE=7 Don't accumulate anything into the measure of the error for equations in this set.

End of Options for Error Control

XRKDIG=8 (N8=0.D0) N8 = 0.D0, turns off diagnostic output for an equation group, =1.D0 turns on output giving information used in computing the error measure for the groups.

XRK???=9 Reserved

XRKXP=10 (N10=0.D0) N10 = 1.D0 causes extra precision to be used in accumulating Y, =0.D0 uses standard floating point precision. This option implies that extra precision will also be used in accumulating TS(1). If one only wants the extra precision in

TS(1), follow this option with option 1, N1 = 1.D0, and then option 10 with N10=0.D0. (This option doesn't seem to help much.)

XRK???=11 Reserved

End of Options For Groups of Equations

Other Options

XRKFR1=12 Use reverse communication for computing $\mathbf{f}(t, \mathbf{y})$, rather than calling DXRK8F. When a return is made to the calling program requesting an evaluation of $\mathbf{f}(t, \mathbf{y})$, the user should store $\mathbf{f}(t, \mathbf{y})$ in F, and call DXRK8A. See Section B.5.

XRKFR2=13 Use reverse communication for computing $\mathbf{f}(t, \mathbf{y})$, rather than calling DXRK8F. Store $\mathbf{f}(t, \mathbf{y})$ in WORK(IDAT(2)). This saves copying F to WORK(IDAT(2)) inside DXRK8A. See Section B.5.

XRKOR=14 Use reverse communication for output rather than calling DXRK8O.

XRKMAX=15 (A15 = |TS(3) − current value in TS(1)|) A15 gives absolute value of the maximum stepsize allowed.

XRKMIN=16 (A16=0.D0) A16 gives the minimum absolute stepsize allowed.

XRKEPS=17 (A17=mach_eps) A17 gives the precision of the floating point arithmetic. If one knows that there is a loss of precision in computing the derivatives, one may want to set this to a value larger than the default value.

XRK???=18 Reserved

XRK???=19 Reserved

XRKMXS=20 (N20) N20 = maximum number of steps between output points. If this option is not set, 100,000 steps are taken before output with IDAT(1) = 20.

Begin Options for Getting Output

XRKEOS=21 Give output at end of all steps.

XRKTI=22 (A22) Interpolate the solution at t = A22. A new value may be set for the output point when output at this point is indicated. This and the following two options can be given multiple times.

XRKTS=23 (A23) As for option 22, except return value at the end of the first step at or past A23 point. (Avoids the expense of interpolation.)

XRKTDT=24 (A24, B24) As for option 22, except after giving the output, if one does not change the output point, the value of B24 is added to the current output point to get a new one.

XRKXG=25 (N25). N25 gives the number of extrapolatory G-Stops.

XRKIG=26 (N26). N26 gives the number of interpolatory G-Stops.

End of Options for Getting Output

XRKEC=27 (A27=100.D0, B27=6.D0) Logic in the code attempts to select the stepsize so that A27×||Error Estimate||^2 is 1, and will reject a step if the square root of this is > B27. ||Error Estimate||^2 is computed by dividing error estimates by requested tolerances. These numbers will be adjusted upwards if the requested error can not be obtained.

XRKIFS=28 (A28=.1D0) Fraction of a step by which an interpolation point can be outside the current integration interval.

B.3 Additional detail for the argument IDAT

IDAT(1:3) are used for communication between the user's code and the integration package. The user should change values in these locations only in the rare cases indicated here. IDAT(1) is used as follows.

= $k < 0$ Such values must be set by the user when DXRK8O is called (would have been called if reverse communication is used). For $k < -1$ these return to the routine that called DXRK8 with IDAT(1) = 3, and IDAT(2) = $-k$.

- -1 Restart the integration after any updates necessary for the last interpolation point and after giving returns for any G-stops that occur at exactly this value of TS(1).
- -2 A return is requested for the purpose of saving the solution with the idea of possibly continuing this integration later. One can save the contents of TS, Y, IDAT, and DAT. Then later restore them and simply call DXRK8A to continue an integration. One need not save the contents of OPT or WORK.
- < -2 The integration is terminated and can not be continued. The value returned in IDAT(2) presumably has some meaning to the user. Such values also be set in DXRK8F.

= 0 Set by the user to indicate the integration is being started.

= 1 Compute derivatives. IDAT(2) gives the location in WORK where derivatives are stored. This is of interest to the user only if he has selected option 13.

= 2 Used for actions associated with G-Stops. See Section B.4.b for details.

= 3 Set when the integration is finished or being interrupted. IDAT(2) provides more detail (if needed) as follows.

= **0** TS(1) = TS(3), presumably the integration is finished. If the integrator is called with a new value for TS(3), the integration continues; if called with TS(3) unchanged, a diagnostic results. Continuing the integration by changing the value of TS(3) only makes sense if one does not require interpolation for any other purpose and is interested in saving space in WORK by avoiding one of the options that require interpolation.

= $k > 1$ Results from the user setting IDAT(1) < -1, see description of that case above.

= **4** An error has occurred. See Section E below for details.

= **20** Output from option 20.

> **20** Values of TS(1) and Y have been set for an output point, and IDAT(1) gives the integer index of the option that resulted in this output point. If 21 < IDAT(1) < 24, then IDAT(2) gives the order in which this option was declared among the options in this range of options, and DAT(IDAT(3)) contains the value of t associated with the output. Thus if one has a large number of different output points defined, one has a way to tell specifically which is responsible for the current output, and one can change the value in DAT(IDAT(3)) (and/or DAT(IDAT(3)+1) when IDAT(1) = 24), to set values for future output. Thus *e.g.* if one has an array of values t_i, where output is desired, one can start with A22 = t_1, and when getting output associated with this option, set DAT(IDAT(3)) = the next t_i in the list. In the case of G-Stops IDAT(2) gives the index in TS() of the g function which has a zero, and IDAT(3) gives the lowest index of the g which has a 0 at exactly the same value of TS(1). When these two indexes are not equal, successive calls will be made to dxrk8f, and dxrk8o until the user has been informed of all such stops. If the lowest indexed G-Stop is an extrapolatory stop, the integration will be restarted automatically after processing the last G-Stop reported for a given value of TS(1). If IDAT(1) is set to -1 or -2, all G-Stops at the current value of TS(1) will be processed prior to taking the action requested.

B.4 Usage of G-Stops

The "G-stop" feature provides a means for the integration package to monitor user defined functions, $g_i(t, \mathbf{y})$, and return control to the user's code for special actions whenever one of these functions changes sign. This feature is activated by Options 25 and 26, and is supported by the subroutine DXRK8G.

A distinction is made between *interpolating* and *extrapolating* G-stop functions. If the differential equations remain well behaved beyond a zero of a function $g(t, \mathbf{y})$ then g should be an interpolating G-stop function. Its zero can be detected by noting a sign change after completing a step of the solution algorithm and then using iteration and interpolation to locate the zero precisely. When $\mathbf{f}(t, \mathbf{y})$ has a discontinuity it is usually the case that there is a natural extension of $\mathbf{f}$ which is smooth and thus will allow the use of interpolatory G-stops. One can distinguish how $\mathbf{f}$ is to be computed in DXRK8F via a flag that is set to one value before the G-stop, and to another value after the stop. Thus if $y' = -|y| + \sin t,\ y_0 = 1$, one could integrate $y' = Y_{mul}\, y + \sin t$ starting with $Y_{mul} = -1$. The G-stop y detects a change in sign of y at the end of a step, iterates to find the 0, and informs the user of this, and provides the values of t and y at this point. The user then changes the sign of Y_{mul} and restarts the integration at the point of the sign change by setting IDAT(1) = -1. Note that unlike the first form for y', the second form has a discontinuity only upon having the G-stop flagged at which point the integration is restarted.

If there is no convenient way to compute $\mathbf{f}$ (or a smooth extension of $\mathbf{f}$) beyond the G-stop, the function $g(t, \mathbf{y})$ must be tested more frequently and its zero must be found by searching from one side using iteration and extrapolation. An example of such a problem is $y' = -\sqrt{|y|} + \sin t,\ y_0 = 1$. Since interpolating G-Stops are easier to use, require less computation, are more accurate and more reliable, they should be preferred when the problem allows their use.

Since the user is to store the values of the g's starting in TS(4), we index them in this way. Using Option 25 the user may specify that there are N25 extrapolating G-stop functions, say $g_i(t, \mathbf{y})$, $i = 4, ...,$ N25+3, and using Option 26 the user may specify that there are N26 interppolating G-stop functions, say $g_i(t, \mathbf{y})$, $i = N25 + 4, ...,$ N25+N26+3, where N25 is 0 if there are no extrapolating G-Stops. If one has a total of k g_i, then the dimension of TS must be at least $(3 + 2k + 1)$. Following the location where the computed g's are stored is the value of TS(1) for previously computed g's, and then the values of the g's that were computed at that previous time.

Whenever a sign change occurs in a g_i the code iterates to find the location of a zero of g_i and provides all output in increasing values of TS(1) (if integrating in the positive direction), whatever the order of the zeros of the g_i or that of other output points.

If there are multiple zeros of a g_i very close together, the code may (or may not) find one of them, but will never find both. (This is to insure that computational noise does not result in multiple stops.) It will purposely skip a sign change that occurs within a quarter of an integration step from one previously found. If it is necessary to

locate all sign changes the following procedure should be followed. Let d denote the lowest order total derivative of **g** with respect to t for which one knows that $(d^d/dt^d)\mathbf{g}$ does not change sign more than once in a given integration step. For some k, let $g_{k-j} = (d^j/dt^j)g$, $j = 0$, 1,..., d. For values of j between 0 and $d-1$ if you are told that g_{k-j} has a 0, then set TS(k-i+N25+N26+1) = TS(k-i) for $i = d,\ d-1,\ \ldots,\ j-1$. This saves the current value of these g's in the history so that a future sign change is not missed even if it is quite close to the previous one. And the fact that we see the sign change in a higher derivative insures that we don't miss sign changes in lower derivatives that precede it.

B.4.a Code for G-stops in subroutine DXRK8F

DXRK8F should be changed to look like the following.

```
      SUBROUTINE DXRK8F(TS, Y, F, IDAT)
      DOUBLE PRECISION TS(*), Y(*), F(*)
      INTEGER IDAT(*)
      IF (IDAT(1) .EQ. 1) THEN
        IF (N25 .NE. 0) THEN
          Compute TS(I)=G(I), I = 4, ..., N25+3
          CALL DXRK8G(TS, Y, F, IDAT)
            IF (IDAT(3) .NE. 0) RETURN
          END IF
        END IF
        Compute F(i) = f_i(T(1), Y), i = 1, 2, ..., NEQ.
        RETURN
      END IF
  100 Compute TS(I)=G(I), I = 4, ..., N25+N26+3
      CALL DXRK8G(TS, Y, F, IDAT)
      IF (IDAT(2) .EQ. -1) GO TO 100
      IF (IDAT(2) .GT. 0) THEN
        You may want to adjust flags for
        a sign change in TS(IDAT(2))
      END IF
      RETURN
      END
```

Although "F" is used in the call to DXRK8G, when using forward communication this "F" is the same as WORK. If using reverse communication one **must** use WORK for this argument.

In most cases N25 is 0, and the block headed by the check for this is not necessary. It may happen that you would like a given g to sometimes be extrapolating and sometimes interpolating[1]. If you make such a g the last of the extrapolating g's (starting in extrapolating mode), or the first of the interpolating g's (starting in interpolating mode), you can change how it is treated by changing the index of the last extrapolating g. This index is held in IDAT(NXGS), where NXGS=16. Thus if it was extrapolating and is now interpolating decrease IDAT(16), and if going the other way increase it. If IDAT(NXGS) is ≤ 3 there are no extrapolating G-Stops.

If one is adjusting flags so that **f** will be computed with a different definition, then when IDAT(1) is 26 in DXRK8O (see Section B.3), one must set IDAT(2) $= -1$ to restart the integration. In the case of IDAT(1) = 25, DXRK8 restarts the integration with TS(2) unchanged if the user has made no changes in IDAT.

B.4.b Argument Definitions, G-stops

TS(), Y(), F() [inout] Same as in B.1 above, except that when IDAT(1) = 2, F() really corresponds to WORK in the call to DXRK8, and contains the information necessary to do interpolations.

IDAT() [inout] When there are no G-Stops, one need not examine IDAT. But otherwise one must distinguish between IDAT(1) = 1, which means it is time to compute the derivatives, and IDAT(1) = 2, which means something connected with G-Stops is happening. When IDAT(1) is 1, one need only know that if there are extrapolating G-Stops then the g's must be computed and DXRK8G called. Following the call one returns if IDAT(3) is not 0. Otherwise **f** is computed as if there were no extrapolating G-Stops.

After a call to DXRK8G, with IDAT(1) = 2, IDAT(2) and IDAT(3) define additional state as follows.

IDAT(2) $= -2$ Return to DXRK8, a sign change has occurred, but the code needs more derivative values in order to interpolate.

IDAT(2) $= -1$ We are iterating to find a 0, compute all of the g's and call DXRK8G.

IDAT(2) $= 0$ No sign change was seen.

IDAT(2) $= k > 0$ We have found a zero for the g stored in TS(k). The index in IDAT(3) gives the index of the lowest indexed g which has a 0 at exactly this time. (When there are multiple G-Stops at the same time, those with higher indexes are given first.) You will be told in both DXRK8F and DXRK8O of all G-Stops that occur at a given time even if you request a restart of the integration.

[1] An example is where a satellite may be in full sunlight, in the full earth shadow or only partially shadowed. The transition from full sunlight or full shadow can be interpolating G-Stops while those leaving partial shadow would use extrapolating stops. Most of the time one would be using the interpolating stops in this example. Of course in both cases one would be restarting the integration after a G-Stop.

B.5 Program Prototype, Reverse Communication, Double Precision

Start as described in Section B.1.a except:

(a) To use reverse communication for derivative computation set Option 12 or 13. With 12, derivative values are stored in an array F. The latter option requires saving the derivative values in WORK(IDAT(2)) which saves doing a copy inside of DXRK8A. In this latter case, F need not be declared, and F in the call to DXRK8A can be replaced by OPT (it won't be referenced).

(b) To use reverse communication for output set Option 14.

(c) The array OPT() must be dimensioned sufficiently large to handle the selected options.

(e) Where code given earlier had a "RETURN", use "GO TO 100".

```
      CALL DXRK8(TS, Y, OPT, IDAT,
        DAT, WORK)
100 CONTINUE
      CALL DXRK8A(TS, Y, F, IDAT,
        DAT, WORK)
```

```
IF(IDAT(1) .LE. 2) THEN
   {This point can only be reached if Option 12
   or 13 has been selected. Do what was specified
   in Section B.1.b for subroutine DXRK8F (or in
   Section B.4.a if G-Stops are being computed).}
   GO TO 100
ELSE IF(IDAT(1) .GE. 20) THEN
   {This point can only be reached if Option 14
   has been selected. Do what was specified in
   Section B.1.c for subroutine DXRKK8O.}
   GO TO 100
ELSE IF (IDAT(1) .NE. 3) THEN
   GO TO 100
END IF
(Here the computation is complete.)
```

B.6 Modifications for Single Precision Usage

Change names starting with DXRK8 to start with SXRK8. Change all DOUBLE PRECISION type statements to REAL.

C. Examples and Remarks

The program DRDXRK8 with output ODDXRK8, illustrates integrating the equations of motion for a simple two body problem.

We wish to emphasize that if you have equations with different characteristics, magnitudes, etc., you should group the equations using option 1. Without such grouping, the error control is liable to give unsatisfactory results.

D. Functional Description

The Initial Value problem stated in Section A is solved using an order 8 explicit Runge-Kutta method with built in error estimators due to Dormand and Prince. The code was derived from DOP853 of [1]. The code here differs in providing many more options, by allowing interpolation to be done in such a way that there is no cost on steps which don't require interpolation, and in different algorithms for starting, selecting the stepsize, detecting stiffness, etc. More details, including comparisons with other Runge-Kutta methods and with DIVA of Chapter 14.1 can be found in [2]. Results in [2] indicate that high order Runge-Kutta methods perform better at high accuracies, while doing as well at low precision. Changes in the code for stepsize selection is documented in [3].

Since this code uses the highest order formulas that are likely practical is probably about as efficient as one can do with existing Runge-Kutta formulas, while providing a range of functionality not usually a part of a Runge-Kutta code.

Following the practice in [1] when referring to orders, order refers to global order. The local order is 1 greater than the global order. For the i^{th} equation there are third and fifth order error estimates, $e_{3,i}$, and $e_{5,i}$, and a desired absolute tolerance τ_i. The code forms

$$\mathrm{E}_3^2 = \sum_{i=1}^{NEQ} (e_{3,i}/\tau_i)^2, \tag{2}$$

and E_5^2 defined similarly using $e_{5,i}$. Since the code generates a result of order 8, we would like an error estimate of order 8. (Such an estimate if obtained directly would require doing extra function evaluations.)

An error estimate of order 8 is obtained from $h \times \mathrm{E}_5 \times (\mathrm{E}_5/\mathrm{E}_3)$, where h is the stepsize. (Actually we use the squares of (these estimates / accuracy requested), and E_3^2 is (.01 $\times$ the average of recent values obtained from Eq. (2).) It is this order 8 estimator that is used in controlling the stepsize. When E_5 / E_3 is > 1 the code uses $h \times \mathrm{E}_5$ as the error estimate, and if this occurs too frequently it may give a diagnostic that the system appears stiff. Except for extremely long integrations, this diagnostic will only be given once. The code attempts to keep the square of the order 8 error estimate $<$ DAT(LERTRY+1) (nominal value is .16), and attempts to keep the log of this estimate close to $-$DAT(LERTRY) (nominal value 4.6, which is close to $-\ln(.01)$). If using an absolute error tolerance or a relative error tolerance when the solution may get very close to zero, one should

insure that the absolute error part of the tolerance does not underflow or overflow, when it is squared. No checks are made to insure that this is the case.

References

1. E. Hairer, S. P. Nørsett, and G. Wanner, **Solving Ordinary Differential Equations I**, Springer Verlag, Berlin, second revised edition (1993).

2. Fred T. Krogh, **An Adams Guy Does the Runge-Kutta**. Technical Report 554, Jet Propulsion Laboratory (March 1997). Available from `http://mathalacarte.com/fkrogh`.

3. Fred T. Krogh, *Stepsize selection for ordinary differential equations*, **ACM Transactions on Mathematical Software 37**, 2 (April 2010) 15:1–15:11.

E. Error Procedures and Restrictions

The following values for IDAT(2) are possible on a return with IDAT(1) = 4. For values of IDAT(1) < 10, the integration can be continued, for larger values it will be stopped. All of these errors print an error message.

2 Error estimate too large, and at min. stepsize. Integration will continue at the minimum stepsize if nothing is done.

3 User did not call DXRK8G when it was requested.

4 Looks like problem is getting stiff. This diagnostic will occur at most one time, and if it is ignored, the stepsize is restricted.

5 Looks like too much precision has been requested. On this return IDAT(3) is 0. If it is not changed, the code will change the accuracy requested to what it regards as a more reasonable value. (It does this by changing the locations in DAT that are set by option 27.)

6 Not used.

7 DXRK8G called and IDAT(1) is not 1 or 2.

8 An Extrapolating G-Stop discovered at end of step is ignored.

9 Calling DXRK8G when IDAT(2) < 0.

10 Trying to continue an integration that finished.

11 Equation indexes in options are badly ordered.

12 Equation index for group is too large (> IDAT(4))

13 A repeated spec. for the same group of equations.

14 An undefined option.

15 Entered with an invalid value for IDAT(1).

16 Trying to continue after a fatal error reported.

17 Absolute/Relative error tolerance vectors overlap.

18 Need to set aside more space in IDAT.

19 Need to set aside more space in DAT.

20 Need to set aside more space in WORK.

21 Option required inside equation group follows the end of the last group.

22 Bad value for IDAT(1) on entry to DXRK8A.

23 Initial TS(2) has the wrong sign.

24 DXRK8G was not called when requested.

25 There was a sign change in an extrapolated g while computing the derivatives necessary to do interpolation. This should only happen if there is significant rounding errors (or bugs) in the computation of the g.

26 Only 0 and 1 can be used to turn options off and on.

27 The parameters specified in option 27 are too close together.

All error diagnostics and messages are handled by calls to the library subroutines MESS and DMESS or SMESS of Chapter 19.3. The user can change the printing and/or stopping actions taken by the error message program or change the file to which such messages are sent by calling MESS.

F. Supporting Information

The source language is ANSI Fortran 77.

Subroutine designed and written by: Fred T. Krogh, JPL, February 1997. Extensively modified by Krogh, February 2008.

Entry	Required Files
DXRK8	AMACH, DMESS, DXRK8, MESS
DXRK8A	AMACH, DMESS, DXRK8, MESS
DXRK8G	AMACH, DMESS, DXRK8, DXRK8G, DZERO, MESS
SXRK8	AMACH, MESS, SMESS, SXRK8
SXRK8A	AMACH, MESS, SMESS, SXRK8
SXRK8G	AMACH, DMESS, MESS, SXRK8, SXRK8G, SZERO

DRDXRK8

```
c       program DRDXRK8
c>> 2008-02-24 DRDXRK8  Krogh -- Change dimenstions for new usage.
c>> 2001-05-25 DRDXRK8  Krogh -- Added comma to format.
c>> 1997-12-18 DRDXRK8  Krogh    Initial code
c--D replaces "?": DR?XRK8, ?XRK8, ?XRK8F, ?XRK8O
c Sample driver for DXRK8 --  Set up to solve simple two body problem
c with circular motion.
c
      integer NEQ, LIDAT, LDAT, LWORK
      parameter (NEQ=4, LIDAT=49+2, LDAT=32+2*NEQ+2+2, LWORK=9*4+6+8*4)
      integer IDAT(LIDAT)
      double precision DAT(LDAT), TS(3), TOL, Y(NEQ), WORK(LWORK)
      double precision OPT(6)
c++S Default SETTOL = 'TOL = 1.E-4'
c++  Default SETTOL = 'TOL = 1.D-10'
c++  Replace 'TOL = 1.D-10' = SETTOL
      parameter (TOL = 1.D-10)
      double precision XRKEND, XRKAE, XRKTDT
      parameter (XRKEND=0.D0, XRKAE=2.D0, XRKTDT=24.D0)
c             Abs. Err. tol.   Output at intervals of 2 Pi       End opt.
      data  OPT /XRKAE,TOL, XRKTDT,0.D0,6.283185307179586477D0, XRKEND/
      data   IDAT(4), IDAT(5), IDAT(6), IDAT(7) /
     1            NEQ,    LIDAT,    LDAT,    LWORK /
c       Starting:  T     H;  Final T.
      data  TS / 0.D0, 0.D0, 20.D0 /
c       Initial Y
      data Y / 1.D0, 0.D0, 0.D0, 1.D0 /
c
c Do the integration
      IDAT(1) = 0
  100 continue
         call DXRK8(TS, Y, OPT, IDAT, DAT, WORK)
         if (IDAT(1) .NE. 3) go to 100
      stop
      end

      subroutine DXRK8F(T, Y, F, IDAT)
c Sample derivative subroutine for use with DXRK8
c This evaluates derivatives for a simple two body problem.
c
      integer          IDAT(*)
      double precision T, Y(*), F(*), RQBI
c
c Evaluate the derivatives
      RQBI = 1.D0 / ((sqrt(Y(1)**2+Y(3)**2))**3)
      F(1) = Y(2)
      F(2) = -Y(1)*RQBI
      F(3) = Y(4)
      F(4) = -Y(3)*RQBI
      return
      end

      subroutine DXRK8O(TS, Y, IDAT, DAT)
c Sample output subroutine for use with DXRK8.
c This subroutine gives output for a simple two body problem.
c
      integer          IDAT(*)
```

```
      double precision TS(*), Y(*), DAT(*)
 1000 format (6X, 'RESULTS FOR A SIMPLE 2-BODY PROBLEM'//
     1    8X, 'T', 13X, 'U/V', 11X, 'UP/VP'/' ')
 1001 format (1P,3E15.6 / 15X, 2E15.6/' ')
c
c Do the output
      if (TS(1) .eq. 0.D0) then
        write (*, 1000)
      end if
      write (*, 1001) TS(1), Y(1), Y(2), Y(3), Y(4)
      return
      end
```

ODDXRK8

RESULTS FOR A SIMPLE 2–BODY PROBLEM

T	U/V	UP/VP
0.000000E+00	1.000000E+00	0.000000E+00
	0.000000E+00	1.000000E+00
6.283185E+00	1.000000E+00	5.645950E−11
	−5.873593E−11	1.000000E+00
1.256637E+01	1.000000E+00	1.088883E−10
	−1.094066E−10	1.000000E+00
1.884956E+01	1.000000E+00	1.514382E−10
	−1.523538E−10	1.000000E+00
2.000000E+01	4.080821E−01	−9.129453E−01
	9.129453E−01	4.080821E−01