RKCHK user guide

P.W. Sharp*
Department of Mathematics, University of Auckland, P.B. 92019,
Auckland, NEW ZEALAND. Email: sharp@math.auckland.ac.nz

J.H. Verner†
Department of Mathematics and Statistics, Queen's University at Kingston,
Kingston, Ontario, CANADA, K7L 3N6

December 2, 1996

Abstract

RKCHK is an easy to use, dual precision Fortran 77 package for checking the accuracy of the coefficients of explicit Runge-Kutta methods. We present nine examples to illustrate the use of RKCHK. We also discuss the installation of RKCHK.

Subject Classifications: AMS: 65L06; CR: G.1.7
Keywords: Runge–Kutta, explicit, coefficients, dual precision

1 Introduction

RKCHK is a dual precision Fortran 77 package for checking the accuracy of the coefficients of explicit Runge-Kutta (ERK) methods. The package is intended for

*This work was partly supported by the University of Auckland Research Council.
†This work was supported by the Natural Sciences and Engineering Research Council of Canada and the Information Technology Research Centre of Ontario.

any ERK methods with a Butcher tableau of the form

$$\begin{array}{c|ccccc} 0 & & & & & \\ c_2 & a_{21} & & & & \\ c_3 & a_{31} & a_{32} & & & \\ \vdots & \vdots & \ddots & & & \\ c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & \\ \hline \text{order } p_1 & b_1^{(1)} & b_2^{(1)} & \cdots & b_{s-1}^{(1)} & b_s^{(1)} \\ \text{order } p_2 & b_1^{(2)} & b_2^{(2)} & \cdots & b_{s-1}^{(2)} & b_s^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \text{order } p_k & b_1^{(k)} & b_2^{(k)} & \cdots & b_{s-1}^{(k)} & b_s^{(k)} \end{array} \tag{1}$$

where $k \leq 12$.

We describe (§2) seven examples which illustrate the use of the double precision version of RKCHK, and two examples (§3) which illustrate the use of the extended precision version. The output from the nine examples was obtained on an Sparc 20 for which the unit round-off (see §4) is approximately 2×10^{-16}.

In presenting these examples, we assume RKCHK has been correctly installed on the user's computer. The installation is discussed in §4. The design of RKCHK is described in [3].

2 Double precision version

2.1 Example 1

Figure 1 is the tableau of a three-stage method consisting of two formulae. We use RKCHK to show numerically that the first formula of the method is order 3 and the second formula is order 2. (We have chosen a low order method for the first example to avoid large input files.)

$$\begin{array}{c|ccc} 0 & 0 & & \\ \frac{1}{2} & \frac{1}{2} & & \\ 1 & -1 & 2 & \\ \hline \text{order 3} & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \text{order 2} & 0 & 1 & 0 \end{array}$$

Figure 1: The Butcher tableau for the three-stage method of Example 1.

To run RKCHK, the user must supply a main program which calls the outer driver `drkchk` of RKCHK, a small input file containing the parameters which describe the structure of the ERK method, and the coefficients used by the method.

The coefficients can be supplied by either appending them to the input file or by providing a subroutine which returns the coefficients. In this example we will use an input file; the use of a subroutine is illustrated in Example 2.

2.1.1 Main program

Figure 2 gives a possible main program which calls `drkchk`. This program is very simple, generic in nature, and will be used for all the examples in this section.

```
program         rkchk

integer         in,out,ind
external        cfcn

in  = 5
out = 6

call drkchk (in,out,cfcn,ind)

if (ind .lt. 0) then
   call dwrerr (out,ind)
end if

stop

end
```

Figure 2: The main program for Example 1.

`drkchk` has four arguments, consisting of the three input arguments `in, out, cfcn` and the output argument `ind`. The first two arguments are the unit numbers for input and output respectively. The third argument is the name of the subroutine which returns the coefficients of the ERK method. If the coefficients are to be read from a file, `cfcn` is a dummy subroutine (one is supplied with RKCHK). The fourth argument is a status flag which specifies if the checking was completed, and if not, why not. On return from `drkchk`, if `ind` is negative, indicating an error condition, the subroutine `dwrerr` can be called to get a short explanation of the error. Error returns are illustrated in Examples 6 and 7.

2.1.2 Parameters of the method

In the input file containing the parameters of the ERK method, the user must supply the following values,

- `nfmla` - the number of formulae in the method; `nfmla` must be positive and less than 13. In this example, `nfmla = 2`.

- `s` - the number of stages in the method; `s` must be positive and less than 36. In this example, the order 3 formula uses 3 stages and the order 2 formula uses 2 of the same stages, which means the method uses three stages i.e. `s = 3`.

- `p` - an array of size at least `nfmla`, with the i^{th} element set to the order of the i^{th} formula in the method. The orders must be positive and less than 13. They can be specified in any order. In this example, `p(1) = 3` and `p(2) = 2`.

(iv) `file` - a boolean value which is true if the coefficients are to be read from a file, and false if the coefficients are to be obtained by calling the subroutine `cfcn`.

If `file = .true.`, the user must also supply the character string `reprcf` which specifies the way the coefficients are represented in the file. Three values are accepted for `reprcf`.

`ratint` The coefficients are rational numbers with both the numerator and denominator given as integers.

`ratfp` The coefficients are rational numbers with both the numerator and denominator given as decimal (floating point) numbers.

`fp` The coefficients are decimal (floating point) numbers.

2.1.3 Coefficients

If the coefficients are to be read from a file, they must be appended to the file containing the parameters of the method. The coefficients must be supplied one per line in the order c_2, c_3, a_{21}, a_{31}, a_{32}, $b_1^{(1)}, b_2^{(1)}, b_3^{(1)}$, $b_1^{(2)}, b_2^{(2)}, b_3^{(2)}$. It is necessary to include all of the zero coefficients.

When the coefficients are supplied as rational numbers:

- each coefficient must be given as n d, where n and d are the numerator and denominator of the coefficient;
- if a coefficient is a non-zero integer, the denominator must be given;
- if a coefficient is zero, the numerator and denominator must both be given, with the denominator non-zero.

Although the above description may seem a little complicated, the creation of an acceptable input file is straightforward. The input file for the method in Figure 1 is given in Figure 3, where we have added a description to the right of each entry in the file.

2	the number of formulae in the method
3	the number of stages in the method
3 2	the orders of the first and second formulae
.true.	the coefficients are to be read from a file
ratint	rational coefficients with the numerator and denominator represented as integers
1 2	c_2
1 1	c_3
1 2	a_{21}
-1 1	a_{31}
2 1	a_{32}
1 6	$b_1^{(1)}$
2 3	$b_2^{(1)}$
1 6	$b_3^{(1)}$
0 1	$b_1^{(2)}$
1 1	$b_2^{(2)}$
0 1	$b_3^{(2)}$

Figure 3: Input file for Example 1.

2.1.4 Output

Figures 4 and 5 contain the output for Example 1. Table 1 in Figure 4 is a summary of the error (residual) in satisfying the order conditions at each order up to the order of each formula in the method. If the coefficients are correct to machine precision, the logarithms in Table 1 should be small and positive, zero or negative. We observe this is the case which gives some numerical evidence the coefficients are correct (to machine precision).

The last line of Table 1 is an estimate, calculated from the residuals, of the number of significant digits the order conditions are satisfied to. The estimate is 15 which, since the unit round-off is approximately 2×10^{-16}, is probably the best we could expect.

Table 2 in Figure 5 is a summary of the error (residual) in satisfying the quadrature conditions (which are a subset of the order conditions). RKCHK has been written so that the order conditions are calculated using just b and a, while the quadrature conditions are calculated using just b and c. This difference helps the location of errors in the coefficients. If the coefficients are correct to machine precision, the logarithms in Table 2 should be negative, zero or small and positive. We observe this is the case here.

Table 3 in Figure 5 is a summary of the error (residual) in satisfying the condi-

```
In the summary below, the column heading

                Form  n
                 ord  k

is for the nth formula, specified as order k by the user.

TABLE 1

The summary below gives the base-10 logarithm of the
L-infinity norm of r/u where r is the vector of residuals

    (l)   alpha     |                s    (l)     |
   r   = ______qm_ | 1  - gamma    sum b   phi  | , m=1...N
    qm    (rho  )! |            qm i=1  i     i|          q
              qm

for the order conditions of a prescribed order q for formula
l, and u is the unit round-off, or an estimate of it, for
the computer. The norms are listed for each order and each
formula. The logs should be small and positive or zero or
negative. If a residual is zero its log is set to 0. The
last line of the summary gives an estimate of the number of
digits to which the order conditions are satisfied.


 Order     Form  1   Form  2
            ord  3    ord  2

   1       -0.26      0.00
   2        0.00      0.00
   3        0.00

No. dig.     15        15
```

Figure 4: Table 1 of the output for Example 1.

```
TABLE 2

The table below gives the base-10 logarithm of the
relative residuals

                1     s      i
        r  = | --- - sum  b c  | / w ,    i=0...p-1
         i     i+1   j=1   j j      i

where   w  = u max(1,max (|b |)),
         i            j     j

p is the order of the formula and u is the unit round-off, or
an estimate of it, for the computer. The logs should be small
and positive or zero or negative. If a residual is zero, its
log is set to zero.

 Order     Form  1   Form  2
            ord  3    ord  2

   1       -0.38      0.00
   2       -0.86      0.00
   3        0.00

TABLE 3

The table below gives the base-10 logarithm of the
relative residuals

                  s
           r  = | c  - sum  a   | / w ,     i=1...s
            i      i   j=1   ij      i

where      w  = u max(1,max (|a  |))
            i            j     ij

and u is the unit round-off, or an estimate of it, for the
computer. The logs should be small and positive or zero or
negative for most methods. If a residual is zero, its log is
set to zero.

                    Stage i      r
                   --------------------
                       2        0.00
                       3        0.00
```

Figure 5: Tables 2 and 3 of the output for Example 1.

tions

$$c_i = \sum_{j=1}^{i-1} a_{ij}, \quad i = 2, \ldots, s. \tag{2}$$

Except possibly for very low order methods, if the coefficients are correct to machine precision, the logarithms in Table 3 should be negative, zero or small and positive, as is the case here.

The residuals are described in greater detail in [3].

2.2 Example 2

This example is the same as Example 1, except we use the subroutine `cfcn` to provide the coefficients of the ERK method. The input file is given in Figure 6. The

```
2
3
3 2
.false.
```

Figure 6: The input file for Example 2.

entry of `.false.` on the fourth line indicates the coefficients will be obtained from the subroutine `cfcn`. There is no need to specify `reprcf` because the subroutine will return the coefficients as floating point numbers.

Figure 7 contains a possible subroutine `cfcn`. The general form of `cfcn` is given in Figure 8. On return from `cfcn`

- the ith element of `c`, $i = 2, \ldots, s$ must be c_i in (1);
- the (i, j)th element of `a`, $j = 1, \ldots, i-1, i = 2, \ldots, s$ must be a_{ij} in (1);
- the (i, j)th element of `b`, $l = 1, ..., k$, $i = 1, ..., s$, is $b_i^{(l)}$ in (1).

The output from the program was the same as in Example 1.

2.3 Example 3

Hairer [2] presented an algorithm for calculating the coefficients of a 17-stage order 10 explicit Runge-Kutta formula. The coefficients are calculated by solving small systems of linear equations derived from the order conditions. An interesting question arises: how accurately are the order conditions satisfied?

To get some idea of the answer to the question, we implemented Hairer's algorithm in double precision on a Sparc 20, calculated a set of coefficients (using the same values of the free parameters given in [2]) and wrote them to a file as floating point numbers with 17 significant digits. We then checked the coefficients on the same computer and in double precision.

Figure 9 contains the first five lines of the input file. The rest of the input file

```
      subroutine cfcn (c,la,a,lb,b)
      integer          la,lb
      double precision c(*),a(la,*),b(lb,*)

c PURPOSE

c   CFCN returns the coefficients for Example 2.

      c(2)   =  1.d0/2.d0
      c(3)   =  1.d0
      a(2,1) =  1.d0/2.d0
      a(3,1) = -1.d0
      a(3,2) =  2.d0
      b(1,1) =  1.d0/6.d0
      b(2,1) =  2.d0/3.d0
      b(3,1) =  1.d0/6.d0
      b(1,2) =  0.d0
      b(2,2) =  1.d0
      b(3,2) =  0.d0

      return

      end
```

Figure 7: The subroutine `cfcn` for Example 2.

```
subroutine cfcn (c,la,a,lb,b)
integer          la,lb
double precision c(*),a(la,*),b(lb,*)
...
...
...
return
end
```

Figure 8: The general form of the subroutine `cfcn`.

```
1                One formula
17               17 stages
10               Order 10
.true.           The coefficients will be read from a file
fp               The coefficients are floating point numbers
```

Figure 9: The first five lines of the input file for Example 3. The descriptions on the right of each line are not a required part of the input.

consists of the coefficients of the order 10 method, one coefficient (as a floating point number) per line.

A summary of the residuals is given in Figure 10. The estimate of the number of significant digits (the last line of Table 1 of the output from RKCHK) was 15. This, together with the residuals in Figure 10, suggests the coefficients are correct to at least 15 digits which, as in Examples 1 and 2, is about the best we could expect for a unit round-off of approximately 2×10^{-16}.

Table 1		Table 2		Table 3	
Order	Residual	Order	Residual	Stage	Residual
1	0.05	1	-0.31	1	0.00
2	0.00	2	-0.98	2	0.00
3	-0.73	3	0.00	3	0.00
4	-0.86	4	-0.76	4	0.00
5	-0.95	5	-1.16	5	-0.86
6	-1.33	6	-0.46	6	0.00
7	0.20	7	-0.86	7	0.00
8	0.11	8	-1.16	8	0.00
9	-0.05	9	-0.61	9	0.00
10	-0.23	10	-0.68	10	0.00
				11	-1.16
				12	-1.28
				13	0.20
				14	-1.46
				15	0.00
				16	0.00
				17	0.03

Figure 10: The three sets of residuals for Example 3.

2.4 Example 4

Verner [4] gave examples of complete sets of imbedded ERK methods. One example was a 13-stage method consisting of formulae of orders one to eight.

Since the coefficients of this method are rational numbers with just a few digits in the numerator and denominator, we appended the coefficients to the input file, instead of calculating them using the algorithm described in [4]. Figure 11 gives the first five lines of the input file. The rest of the input file has the coefficients, one coefficient per line with zero coefficients included.

Figure 12 contains the three tables of residuals produced by RKCHK (to save space we have omitted the pre-ambles for the tables). We observe the logarithms

```
8                 Eight formulae
13                13 stages
8 7 6 5 4 3 2 1   Orders 8 to 1
.true.            The coefficients will be read from a file
ratint            Rational coefficients with integer numerator and
                  denominator
```

Figure 11: The first five lines of the input file for Example 4.

are either negative, zero or small and positive which suggests the coefficients are correct to machine precision.

2.5 Example 5

This example illustrates how, when a coefficient is wrong, the output from RKCHK can be used to deduce which coefficient is likely to be wrong.

Figure 13 below gives an input file to RKCHK for the six-stage CSIRK method given by Verner [4]. This method consists of formulae of order 1 to 4 embedded in an order 5 formula. The coefficients are rational numbers and are exact. To save space we have put more than one coefficient per line. When using RKCHK there must be exactly one coefficient per line. Figure 14 gives the residuals produced by RKCHK.

The logarithms of the residuals in Table 1 of the output in Figure 14 are negative, zero or small and positive which is a good indication the order conditions are satisfied to machine precision. This implies, since the order conditions are calculated in RKCHK using the interior and exterior weights, that both sets of weights are probably correct.

The logarithms of the residuals in Table 2 for the formulae of orders 1 to 4 are zero or negative, which is a good indication the quadrature conditions for the four formulae as calculated by RKCHK are satisfied. However the logarithms for the order 5 formula, except for the first one, are large.

Since the quadrature conditions are calculated using only the exterior weights and the abscissae we can conclude using just Table 2 that a) the exterior weights for the formulae of orders 1 to 4 are probably correct, and b) the exterior weights for the formula of order 5 or the abscissae are probably wrong.

If we assume the exterior weights are correct (as suggested by Table 1), the error must be in the abscissae. If one or more of c_2, c_3, c_4 or c_5 is wrong, then since $b_6^{(l)} = 0, l = 2, 3, 4, 5$, the quadrature conditions for the formulae of orders 1 to 4 would probably not be satisfied. Hence we conclude c_6 is probably wrong.

The logarithms of the residuals in Table 3 for stages 2 to 5 are zero or negative which is a good indication the conditions (2) are satisfied for stages 2 to 5. Since the row conditions depend only on the abscissae and the interior weights, we conclude

Order	Form 1 (ord 8)	Form 2 (ord 7)	Form 3 (ord 6)	Form 4 (ord 5)	Form 5 (ord 4)	Form 6 (ord 3)	Form 7 (ord 2)	Form 8 (ord 1)
1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.00	-0.56	0.00	0.35	0.35	-0.26	0.00	
3	-0.43	-0.43	-0.26	0.22	0.47	-0.33		
4	-1.33	-0.56	-0.38	0.35	-0.08			
5	-1.43	-0.78	-0.71	-0.11				
6	-1.41	-0.94	-0.90					
7	-1.62	-1.18						
8	-1.94							
No. dig.	15	15	15	15	15	15	15	15

Order	Form 1 (ord 8)	Form 2 (ord 7)	Form 3 (ord 6)	Form 4 (ord 5)	Form 5 (ord 4)	Form 6 (ord 3)	Form 7 (ord 2)	Form 8 (ord 1)
1	-0.61	-1.46	-0.53	0.00	0.00	0.00	0.00	0.00
2	-0.56	0.00	-0.95	-0.58	-0.86	0.00	0.00	
3	-1.16	-0.68	-0.91	-0.58	-1.16	0.00		
4	-0.86	-0.68	-0.74	-0.64	-1.10			
5	-0.86	-0.76	-0.93	-0.49				
6	-1.16	-0.98	-0.96					
7	-0.68	-0.61						
8	-0.86							

Stage i	|r|
1	0.00
2	0.00
3	0.00
4	0.00
5	0.00
6	0.00
7	0.00
8	-0.65
9	-0.69
10	-0.46
11	-0.51
12	-0.49
13	-0.44

Figure 12: The residuals for Example 4.

```
5
6
5 4 3 2 1
.true.
ratint
3 10 2 5 1 1 39 40 1 41
3 10
2 15 4 15
7 12            -10 3        15 4
12207 25600   -1677 640    15847 5120    299 12800
887 15360    -433 1920     213 1024     -39 2560       0 1
1 12             0 1       1405 2484    -322 351      1600 1311  320 6669
59 468           0 1        475 828     -8 9          3200 2691   0 1
1 12             0 1         25 36       2 9            0 1       0 1
-1 4             0 1          5 4         0 1           0 1       0 1
1 1              0 1          0 1         0 1           0 1       0 1
```

Figure 13: Input file for Example 5.

the abscissae and the interior weights for the first 5 stages are probably correct. However the logarithm for the sixth stage is large. If the row condition is supposed to be satisfied by the sixth stage (which is highly likely for an order 5 formula), we have further evidence that c_6 is wrong.

In fact c_6 should be 1/40 and not 1/41.

2.6 Example 6

This example illustrates the output from RKCHK when one or more parameters of the ERK method have values unacceptable to RKCHK.

Figure 15 contains the input file. It is the same as in Example 1 except the number of stages has been change to -3 and the order of the formulae to 43 and 0. These values are unacceptable since the number of stages must be positive and less than 36, and the order of the formulae must be positive and less than 13.

RKCHK detected the unacceptable values and exited with `ind = -1`. The subroutine `dwrerr` was then called (see the main program in Example 1) which produced the output in Figure 16.

2.7 Example 7

This example illustrates the output from RKCHK when the coefficients are specified as rational numbers and one of the denominators is zero. We used the same file as in Example 1, except the entry for c_3 was changed from `1 1` to `1 0`. The output is given in Figure 17.

TABLE 1

Order	Form 1 ord 5	Form 2 ord 4	Form 3 ord 3	Form 4 ord 2	Form 5 ord 1
1	0.00	0.00	0.00	0.00	0.00
2	-0.26	-0.26	0.00	0.00	
3	0.31	0.31	0.00		
4	0.26	0.35			
5	-0.05				
No. dig.	15	15	15	15	15

TABLE 2

Order	Form 1 ord 5	Form 2 ord 4	Form 3 ord 3	Form 4 ord 2	Form 5 ord 1
1	-0.25	-0.03	0.00	0.00	0.00
2	11.08	-0.03	0.00	0.00	
3	9.77	0.00	-0.86		
4	8.34	0.00			
5	6.86				

TABLE 3

Stage i	r
2	0.00
3	-0.56
4	0.00
5	-0.22
6	12.48

Figure 14: The residuals for Example 5.

```
2
-3
43 0
.true.
ratint
1 2 1 1
1 2
-1 1 2 1
1 6 2 3 1 6
0 1 1 1 0 1
```

Figure 15: Input file for Example 6.

```
*** RKCHK: ERROR ***

Input values (other than the coefficients) were unacceptable
for the following reasons:

      - s was less than 1 or greater than 35.
      - p for at least one of the formulae was less than 1 or
        greater than 12.
```

Figure 16: The output for Example 6.

```
*** RKCHK: ERROR ***

  Rational coefficients were being checked and the
  denominator of a coefficient was zero.
```

Figure 17: The output for Example 7.

3 Extended precision version

The extended precision version of RKCHK, which for convenience we denote by RKCHKep, uses the multi-precision package MPFUN developed by Bailey [1]. Central to MPFUN is a suite of functions for performing arithmetic operations and transcendental functions on floating point numbers of arbitrarily high precision. There is also a translator which by means of special comments in the user's program, translates the program to the required precision. This translated version along with the suite of multi-precision functions and any user files (such as the subroutine `cfcn`, see Example 9) is then compiled and linked to form the executable module.

We provide an untranslated and a translated version of RKCHKep, with the subroutines and a sample main program for each version in a single file. The untranslated version is essentially the double precision version of RKCHK with some changes to the comments and the special translator comments added. The subroutine names are the same as in the double precision version. The translated version is for a precision level of 45 which typically gives 38 digits of accuracy. This amount of accuracy should be sufficient for most purposes.

If greater accuracy is required, the untranslated version can be converted to the required precision using the following steps.

(i) Change the `45` in the special comment `cmp+ precision level 45` at the start of the untranslated file to `cmp+ precision level n` where `n` should be a least 7 greater than the number of digits of accuracy required. For example, if 60 digits are required, `n` should be at least 67.

(ii) Change `urnd` in the subroutine `dmchne` to 10^{7-n} where n is from step (i).

```
          TABLE 1                          TABLE 2

 Order     Form  1   Form  2    Order     Form  1   Form  2
            ord  3    ord  2               ord  3    ord  2

   1        0.00      0.00        1      -12.57      0.00
   2        0.00      0.00        2      -12.57      0.00
   3        0.00                  3      -12.57

No. Dig      38        38

                        TABLE 3

                     Stage i      r
                    --------------------
                        2        0.00
                        3        0.00
```

Figure 18: The residuals for Example 8.

(iii) Translate the untranslated version using the translator of [1].

Once an executable module has been obtained, RKCHKep is used in the same way as the double precision version of RKCHK, except when floating point numbers are being read from a file (i.e. when `reprcf` is either `fp` or `ratfp`). In these two cases, the exponent (if there is one) of each floating point number must appear first. For example, 2.72×10^{-4} must be entered as $10^{-4} \times 2.72$. This modified representation of real numbers is required by MPFUN.

3.1 Example 8

This example is Example 1 re-run with RKCHKep. We used the main program supplied with RKCHKep and the same input files as in Example 1. The output is given in Figure 18. We observe from the residuals that the coefficients are correct to at least 38 digits. The values of -12.57 in Table 2 of Figure 18 may seem too negative. However, it must be remembered that although the unit round-off is set at 10^{-38}, the calculations are being done in 45 digits.

3.2 Example 9

In this example, we re-did the checking of Example 3 (a formula from the family of order 10, 17-stage ERK formulae derived by Hairer).

We decided to supply the coefficients of the ERK method by using the subroutine `cfcn`. We could have appended them to the input file, but this would have meant

changing the floating point representation of the coefficients to the form required by MPFUN (exponent first, then the mantissa).

Instead of writing the subroutine `cfcn` by hand we generated it automatically using the subroutine `mkcfcn` which, along with a main program, is supplied with RKCHK. The subroutine `mkcfcn` first reads the floating point coefficients of an ERK method as character strings from a file. It then prints `cfcn` with the coefficients specified as floating point assignment statements. The user can provide a description at the beginning of the input file to `mkcfcn`. This is printed in the prologue of `cfcn`. The subroutine must then be translated to the required precision.

Figure 19 is the input file for Example 9. The fourth entry has been set `.false`

```
1                 One formula
17                17 stages
10                Order 10
.false.           The coefficients will be obtained by calling cfcn
```

Figure 19: The input file for Example 9.

to indicate the coefficients will be provided by a subroutine.

The residuals are given in Figures 20. We observe, since u is set to 10^{-38}, the coefficients are accurate to about 15 digits.

```
     TABLE 1              TABLE 2                TABLE 3

 Order     Form  1     Order     Form 1       Stage i     r
            ord 10               ord 10      ------------------

   1       20.78         1       20.78           2       0.00
   2       21.19         2       21.10           3       0.00
   3       21.33         3       21.51           4      21.00
   4       21.31         4       21.65           5      21.46
   5       21.17         5       21.70           6       0.00
   6       20.96         6       21.73           7      21.48
   7       22.50         7       21.73           8      20.52
   8       22.41         8       21.73           9      20.31
   9       22.25         9       21.72          10      20.78
  10       22.07        10       21.71          11      21.23
                                                12      20.95
No. dig.     15                                 13      22.23
                                                14      20.34
                                                15       0.00
                                                16       0.00
                                                17      22.28
```

Figure 20: The residuals for Example 9.

4 Installation

RKCHK was designed to be portable and has been tested on a number of platforms. Despite this, it is important to be thorough when installing RKCHK on a new platform.

4.1 Obtaining the files

The files for RKCHK are available at

`http://www.math.auckland.ac.nz/~sharp/software/rkchk/html_files/rkchk.html`

They are grouped into the following seven sub-directories.

- *rkchk_dp* - the double precision version of RKCHK, including the sample main program used in this guide. The subroutines and main program are available as individual files, and as one file with the sample main program at the start of the file.
- *rkchk_ep* - the extended precision version of RKCHK, including the sample main program used in this guide. The sub-directory contains two files: one is the untranslated version and the other is the translated version for a precision level of 45. In both files, the main program is at the start.
- *mpfun* - the multi-precision package MPFUN. The sub-directory contains two files. One is the suite of multi-precision functions, the other is the translator.
- *input_files* - the input files for the examples in this guide, including the input file for the creation of the subroutine `cfcn` in Example 9.
- *output_files* - the output files for the examples in this guide.
- *cfcn* - the subroutine `mkcfcn`, a main program for running it, and the examples of `cfcn` used in this guide.
- *make_files* - a makefile for each of the double precision version of RKCHK, the extended precision version of RKCHK and for the subroutine `mkcfcn`.

There is also an eighth sub-directory called *html_files* which contains *.html* files.

4.2 Setting the unit round-off

Once the files have been obtained, the first step of the installation is to set the unit round-off to a sensible value for the computer being used.

The unit round-off is defined as the smallest positive machine number u such that

$$1 + u > 1$$

on the computer.

The value of u for RKCHK is set in the subroutine `dmchne` as the variable `urnd`. It is not necessary that `urnd` be the true value of u, but the value used should err on the large side.

On many machines in double precision, u is in the range $[10^{-16}, 10^{-15}]$. The double precision version of RKCHK is supplied with `urnd` set to 2×10^{-16}.

In RKCHKep, if the precision level is n, `urnd` should be set to 10^{7-n}. Since RKCHKep is supplied with $n = 45$, `urnd` is set to 10^{-38}.

4.3 Using the sample input files

One way to check the installation of RKCHK is to run it on the sample input files and compare the output with the sample output. Because of differences in arithmetic between computers, the output files may differ. However, the differences should not be large, and conclusions which can be drawn from the output should be similar.

References

[1] D. H. Bailey, *Multi-precision Translation and Execution of Fortran Programs*, ACM Transactions on Mathematical Software, **19**, no. 3, Sept. 1993, p. 288-319.

[2] E. Hairer, *A Runge-Kutta method of order 10*, J. Inst. Maths Applics, **21**, (1978), 47-59.

[3] P.W. Sharp, J.H. Verner, *Dual precision software for checking explicit Runge-Kutta methods*, Report Series No. 363, Department of Mathematics, University of Auckland, November, 1996.

[4] Verner J.H., *Families of imbedded Runge-Kutta methods*, SIAM J. Num. Anal., **16**, No. 5, (1979), 857-875.