Dual precision software for checking explicit Runge–Kutta methods

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. Email: jim@jhv.mast.queensu.ca

December 2, 1996

Abstract

Coefficients of an explicit Runge–Kutta method which might be used for initial-value ordinary differential equations, delay differential equations or Volterra integral equations, often require many digits for their representation. This can make manual checking of the coefficients unreliable.

We present a dual precision Fortran 77 package for checking the coefficients of an explicit Runge–Kutta method consisting of k formulae. In some instances when a coefficient is wrong, the output from the package can be used to deduce which coefficient is likely to be wrong.

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

1 Introduction

Over the last thirty years a lot of research has gone into deriving explicit Runge–Kutta (ERK) methods for initial value problems of the form

$$y' = f(x, y), \quad y(x_0) = y_0, \tag{1}$$

*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.

where $f : \mathbb{R} \times \mathbb{R}^n \mapsto \mathbb{R}^n$. One consequence of this research is that a large number of sets of coefficients for ERK methods have appeared with the more popular being made widely available through electronic distribution.

ERK methods are also used to solve delay differential equations of the form

$$y' = f(x, y(x), y(\theta(x, y(x)))), \quad x \geq x_0, \quad y(x_0) = y_0, \quad y(x) = \phi(x), x < x_0, \tag{2}$$

where $f : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}^n$ and $\theta(x, y(x)) \leq x$. In addition, ERK methods are used as explicit Pouzet Volterra Runge–Kutta methods for Volterra integral equations of the form

$$y(x) = g(x) + \int_0^x k(x, \tau, y(\tau)) d\tau, \quad x \in X := [0, x_f], \tag{3}$$

where $g : X \to \mathbb{R}^n$ and $k : D \times \mathbb{R}^n \to \mathbb{R}^n$, $D := \{(x, \tau) \mid 0 \leq \tau \leq x \leq x_f\}$.

A number of ERK methods have been implemented in integrators including DVERK [7], RKSUITE and DOPRI8 [6] for (1), RKLAG [9] and RETARD [6] for (2), and RKVIEP [3], VOLCON [5] and V2EXT [8] for (3).

The coefficients of ERK methods, whether they are used for (1), (2) or (3) often contain a large number of digits. This is particularly so of high order methods. For example, the coefficients for the widely used 13-stage, (7,8) pair of Prince and Dormand [10] are (approximately) represented by rational numbers which require 1317 digits. If any one of the more significant digits is wrong, either or both formulae in the pair will have a reduced order, and may not even have order one.

We present a Fortran 77 package called RKCHK which checks the accuracy of a set of coefficients. The idea is to compute residuals obtained on substituting the coefficients of a method into the order conditions. For high-order methods, often only approximations to an exact set of coefficients are available; for these, the residual of each order will be non-zero, but those up to order p will be at the level of unit round-off of the computer. Our package will indicate this by giving non-zero values of each residual for most or all orders: there will be a sudden increase in the residual following that of the design order which will indicate the order of a method.

RKCHK comes in two precisions: double precision and extended precision of approximately 38 digits. The latter is built around the multiprecision package MPFUN [1].

Since the extended precision version is more accurate than the double precision version, the latter may seem redundant. However, we believe it is not, because ERK methods are commonly used in double precision. In these cases, the double precision version of RKCHK will give users a good indication of the accuracy of the coefficients in the arithmetic they are using.

In section 2 we describe the checks performed by RKCHK on an ERK method. In section 3, we summarise the implementation of the double precision version of RKCHK and describe how it is used. Then in section 4 we discuss the extended precision version. We follow this in section 5 with two examples of using RKCHK, one for each of the double and extended precision versions. We also show how the output can be used in some instances to deduce which coefficient is likely to be wrong.

2 Checks

We assume for $x_i = x_{i-1} + h$ the ERK method generates k approximations $y_i^{(l)}$ to $y(x_i)$, $i = 1, \ldots$, with the approximations defined as

$$\left.\begin{array}{rcl} y_i^{(1)} & = & u_{i-1} + h \sum_{j=1}^{s} b_j^{(1)} f_j \\ y_i^{(2)} & = & u_{i-1} + h \sum_{j=1}^{s} b_j^{(2)} f_j \\ \vdots & = & \vdots \\ y_i^{(k)} & = & u_{i-1} + h \sum_{j=1}^{s} b_j^{(k)} f_j \end{array}\right\} \tag{4}$$

where u_{i-1} is one of $y_{i-1}^{(1)}, \ldots, y_{i-1}^{(k)}$ and

$$\begin{array}{rcl} f_1 & = & f(x_{i-1}, u_{i-1}) \\ f_j & = & f(x_{i-1} + c_j h, u_{i-1} + h \displaystyle\sum_{l=1}^{j-1} a_{jl} f_l), \qquad j = 2, \ldots, s. \end{array} \tag{5}$$

We also assume the coefficients are selected so that approximation l is of order p_l, $l = 1, \ldots, k$; i.e.

$$y(x_i) - y_i^{(l)} = O(h^{p_l}).$$

We refer to each of the k approximations as a formula and the collection of approximations defined by (4) and (5) as an ERK method. We refer to c_i, $i = 2, \ldots, s$, as the abscissae, to a_{ij}, $j = 1, \ldots, i-1$, $i = 2, \ldots, s$, as the interior weights, and to $b_i^{(l)}$, $i = 1, \ldots, s$, $l = 1, \ldots, k$, as the exterior weights.

All existing ERK methods, except very low order ones, satisfy the row conditions

$$c_i = \sum_{j=1}^{i-1} a_{ij}, \quad i = 2, \ldots, s. \tag{6}$$

We use these conditions in the checking process.

Although most ERK methods consist of either one or two formulae, there do exist ERK methods which consist of up to 8 formulae (see Verner [11] for example). We assume $k \leq 12$.

For convenience we write the coefficients of the method as the Butcher tableau

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

2.1 Order conditions

Three sets of checks are performed by RKCHK. The first set is on the order conditions for each formula in the ERK method.

For the l^{th} formula in (4), $l = 1, \ldots, k$, the order conditions can be written as

$$v_{rm}^{(l)} := \frac{\alpha_{rm}}{r!}\left[1 - \gamma_{rm}\sum_{j=1}^{s}(b_j^{(l)}\phi_{rmj})\right] = 0, \quad m = 1, \ldots, N_r, \quad r = 1, \ldots, p_l, \tag{7}$$

where N_r is the number of order conditions of order r. The elementary weights ϕ_{rmj} are polynomials of degree $r-1$ in the interior weights and abscissae, while the parameters γ_{rm} and α_{rm} are integers. A comprehensive exposition on order conditions for Runge–Kutta methods is given in a number of monographs such as [2] (p79-104, 163-173) and [6] (p142-153).

As conditions (6) are assumed, each elementary weight ϕ_{rmj} can be expressed in terms of the interior weights alone. We elected to use this form to permit a more thorough checking of the interior weights. For each formula in a method, we first generate the left hand side of (7) in this form. We refer to these as the residuals (for the order conditions). For each order $q = 1, \ldots, p_k$ and each formula $l = 1, \ldots, k$, we then calculate the L_∞ norm of the vector $v_q^{(l)}$ of residuals. From this we calculate the scaled norm $||v_q^{(l)}||_\infty/u$ where u is the unit round-off or an estimate of it. We print the base-10 logarithm of the scaled norm. If the norm is zero, we set its logarithm to zero.

If the coefficients are exact and exact arithmetic is used when calculating the residuals and the scaled norms, the scaled norms would be zero. But in finite precision they will usually be non-zero. However our experience suggests the scaled norms will be $O(1)$ if the coefficients are accurate which means their logarithm should be small and positive or zero or negative.

The order conditions are generated using a recursive tree algorithm. A number of recursive algorithms are possible. We use one based on some early work of Butcher (private communication).

Our scheme for generating the residual of the order conditions uses only the interior and exterior weights. We could have written the generation scheme so it used the abscissae as well and thereby reduced the amount of computation required. We elected not to do this in order to aid the identification of which coefficients were in error when one or more checks were not passed.

2.2 Quadrature conditions

Our second set of checks is of the quadrature conditions for each formula in the method. Since these conditions are a subset of the order conditions, they are already available from the first check. However, in order to check the abscissae, we recalculate the quadrature conditions using the abscissae and the exterior weights, and not using the interior and exterior weights as was done for the order conditions.

For the l^{th} formula, $l = 1, \ldots, k$, the quadrature conditions can be written as

$$r_1 := 1 - \sum_{j=1}^{s} b_j^{(l)} = 0, \quad r_q := \frac{1}{q} - \sum_{j=2}^{s} b_j^{(l)} c_j^{q-1} = 0, \quad q = 2, \ldots, p_l.$$

We form the residual of each quadrature condition and print the base-10 logarithm of

$$\frac{|r_i|}{u \max\{1, \max_j\{|b_j^{(l)}|\}\}}, \quad i = 1, \ldots, p_l.$$

As for the first set of checks, each logarithm should be a small positive number or zero or negative.

2.3 Row conditions

Our third and final set of checks is of the conditions

$$r_i = c_i - \sum_{j=1}^{i-1} a_{ij} = 0, \quad i = 2, \ldots, s. \tag{8}$$

(These conditions are often denoted by $A(1)$.) We form and print the base-10 logarithm of

$$\frac{|r_i|}{u \max\{1, \max_{1 \le j \le i-1}\{|a_{ij}|\}\}}, \quad i = 2, \ldots, s.$$

Each logarithm should be a small positive number or zero or negative, except possibly for very low order methods when conditions (6) are not satisfied.

3 Double precision version

3.1 Overview

The double precision version of RKCHK consists of 22 subroutines. These can be divided into six groups which for convenience we refer to as `driver`, `input`, `trees`, `residuals`, `output` and `machine`.

`driver` This group consists of five driver subroutines: `drkchk`, `drkcff`, `drktre`, `delm` and `doc`. The first driver is the one called by the user's program, the second controls the reading of the coefficients of the ERK method, the third controls the generation of the trees, the fourth controls the calculation of the elementary weights and the fifth controls the calculation of the residuals for the order conditions.

`input` This group consists of five subroutines: `drdval`, `dchkin`, `drdfp`, `drdrf` and `drdri`. The first subroutine is used to read information about the method being checked such as the number of stages and the number of formulae. The second subroutine performs some simple acceptance checks on the input values (for example, the number of stages must be positive), and returns with an error flag set if the values are unacceptable. The subroutines `drdfp`, `drdrf` and `drdri` read the coefficients from a file.

`trees` This group consists of two subroutines: `drktr1` and `drktrq`. These implement the recursive algorithm used to generate the Runge–Kutta trees.

`residuals` This group consists of four subroutines: `delmex`, `docin`, `dqres` and `da1res`. The subroutine `delmex` calculates the elementary weights and `docin` calculates the residuals for the order conditions. The remaining two subroutines calculate the residuals for the quadrature conditions and the row conditions respectively.

`output` This group consists of five subroutines: `dwrerr`, `dwroc`, `dwrqc`, `dwra1` and `dformt`. The first subroutine is called from the user's program after an error return from `drkchk`. It prints an explanation of the error return. The subroutines `dwroc`, `dwrqc` and `dwra1` print a table of the logarithms for the order conditions, the quadrature conditions and the row conditions respectively. The subroutine `dformt` returns the format statement needed by `dwroc` and `dwrqc` when printing a line of logarithms.

`machine` This group consists of just the subroutine `dmchne` which returns an estimate of the unit round-off (the user must explicitly set the value in `dmchne`).

3.2 Input values to `drkchk`

The input values can be divided into (i) the parameters of the method being checked and (ii) the coefficients.

For (i), the user must supply a file containing k, s, p_l, $l = 1, \ldots, k$, and a boolean value which is true if the coefficients of the method are to be read from a file. If they are, the user must also specify how the coefficients are represented. RKCHK provides for three representations: rational numbers with the numerator and denominator both integers, rational numbers with the numerator and denominator both floating point numbers, and floating point numbers. These three possibilities are specified in the input file by the strings `ratint`, `ratfp` and `fp` respectively and saved in the variable `reprcf`.

The values of k, s, p_l, $l = 1, \ldots, k$ must satisfy

$$1 \le k \le 12, \quad 1 \le s \le 35, \quad 1 \le p_l \le 12.$$

For (ii), the coefficients can be supplied by providing a file (which must be appended to the file for (i)), or by providing a subroutine which returns the coefficients. If the coefficients are supplied by file, there must be one coefficient per line with all zero coefficients included. For rational coefficients the numerator and denominator must both be entered even if the numerator is zero or the denominator is one.

3.3 Calling program

A sample calling program is given in Figure 1. `drkchk` has three input arguments

```
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 1: A sample calling program for `drkchk`.

(`in`, `out` and `cfcn`) and one output argument (`ind`). The arguments `in` and `out` are the unit numbers for input and output respectively, and `cfcn` is the name of the subroutine which returns the coefficients of the ERK method. If the coefficients are being read from a file, `cfcn` can be a dummy subroutine. The argument `ind` is a output status flag which indicates whether the checking was completed, and if not, why not.

3.4 Values of `ind`

There are three possible values of `ind` on exit from `drkchk`.

`ind = 1` RKCHK completed the checking. A summary is in the output file.

`ind = -1` One or more of the input values describing the method (excluding the coefficients) were unacceptable.

`ind = -2` Rational coefficients were being read from a file and the denominator of a coefficient was zero.

3.5 Machine dependence

The only machine dependent number used besides the unit numbers for input and output (both are input arguments to `drkchk`), is the unit round-off u. This value or an estimate of it must be explicitly set by the user in the subroutine `dmchne`.

The subroutine `dformt` does some simple character manipulation. We have written this in what we believe is a portable way.

3.6 Common blocks

RKCHK uses the common block `cchkin` to pass information about errors in the input values from `dchkin` to `dwrerr`.

4 Extended precision version

The extended precision version of RKCHK, which for convenience we denote by RKCHKep, uses the multiprecision package developed by Bailey [1]. Bailey's package consists of a suite of functions for performing arithmetic operations and transcendental functions on floating point numbers of arbitrarily high precision, together with 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 multiprecision functions 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).

(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 $10^{-4} \times 2.72$.

To avoid having to do this for the case `repref = fp`, we have written a subroutine called `mkcfcn` which reads the coefficients of the method as character strings from a file and prints the subroutine `cfcn`.

5 Examples

5.1 Example 1

To illustrate the use of the double precision version of RKCHK, we ran it on the 6-stage CSIRK method given by Verner [11]. This method consists of formulae of order 1 to 4 embedded in an order 5 formula. The coefficients are rational numbers and are exact.

We elected to read the coefficients from a file. Our input file is given in Figure 2, where to save space *in this article only*, we have put more than one coefficient per line. (For use with RKCHK, each coefficient must be entered on a single line.) The first line is the number of formulae, the second line is the number of stages and the third line gives the order of the formulae in the method. The value on the fourth line is true which means the coefficients are to be read from the file. The fifth line gives the type of representation of the coefficients: `ratint` signals the coefficients are rational numbers with both the denominator and numerator being integers.

The rest of the file gives the coefficients in the order $c_2, \ldots, c_6, a_{21}, a_{31}, a_{32}, a_{41}, \ldots, a_{s,s-1}$, followed by the exterior weights of the order 5 formula, the order 4 formula and so on down to the order 1 formula. It is important to note that although the lower order formulae do not use some of the six stages, the exterior weights for the stages not used must be entered (as `0 1`). (It is necessary to explicitly state which exterior weights are zero because the nonzero exterior weights can not be predicted from just s and the order of the formulae in the method).

```
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 2: Input file for Example 1.

When we ran RKCHK using the main program of the previous section, we obtained the output in Figure 3 and 4.

The logarithms of the residuals in Table 1 of the output are small positive numbers or zero or negative, 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.

Because 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.

```
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 for
the order conditions of a prescribed order 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   Form  3   Form  4   Form  5
            ord  5    ord  4    ord  3    ord  2    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
```

Figure 3: 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   Form  3   Form  4   Form  5
            ord  5    ord  4    ord  3    ord  2    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

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       -0.56
                        4        0.00
                        5       -0.22
                        6       12.48
```

Figure 4: Tables 2 and 3 of the output for Example 1.

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 that 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 row conditions are satisfied for stages 2 to 5. Since the row conditions depend only on the abscissae and the interior weights, we conclude that 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 (6) 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. (We intentionally introduced this error in the coefficients from [11] to illustrate the capacity of RKCHK to identify errors in the coefficients.)

5.2 Example 2

To illustrate the extended precision version of RKCHK, we generated the coefficients of an explicit Runge–Kutta formula from the family of order 10 formulae derived by Hairer [4] using double precision arithmetic, and then checked the accuracy of the coefficients. The coefficients were generated using the algorithm described in [4].

The output from RKCHKep is given in Figures 5. To save space we have omitted the pre-amble to the tables and placed them side by side. We observe from the output, since u is set to 10^{-38}, that the coefficients are accurate to about 15 digits.

References

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

[2] J.C. Butcher, *The numerical analysis of ordinary differential equations*, John Wiley & Sons Ltd, 1987.

[3] R.P. Duncan, *A Runge–Kutta method using variable stepsizes for Volterra integral equations of the 2nd kind*, TR 157/82, Dept. of Computer Science, University of Toronto, 1982.

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

[5] E. Hairer, Ch. Lubich, M. Schlichte, *Fast numerical solution of nonlinear Volterra convolution equations*, SIAM J. Sci. Stat. Comp., **6**, (1985) 532-541,

[6] E. Hairer, S.P. Nørsett, G. Wanner, *Solving ordinary differential equations I*, Springer series in computational mathematics, **8**, Springer Verlag.

TABLE 1		TABLE 2		TABLE 3	
Order	Form 1 ord 10	Order	Form 1 ord 10	Stage i	r
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 5: Tables 1, 2 and 3 for Example 2.

[7] T.E. Hull, W.H. Enright, K.J. Jackson, *User's guide for DVERK - a subroutine for solving non-stiff ODE's*, Tech. Rep. 100, Department of Computer Science, Univ. of Toronto, 1976.

[8] A.N. Lomakovich, V.A. Ischuk, *The approximation solution of certain nonlinear Volterra type integral equations by a bilateral method of Runge–Kutta–Fehlberg form* (Russian), Vychisl, Prikl, Math (Kiev), 23, 2904. A translation by C.T.H. Baker, M.S. Keech and P. Sermer appeared as Numer. Anal. Report No. 26, Dept. of Mathematics, University of Manchester.

[9] K.W. Neves, S. Thompson, *Software for the Numerical Solution of Systems of Functional Differential Equations With State Dependent Delays*, Applied Numerical Mathematics, **9**, (1992), 385-401.

[10] Prince P.J., Dormand J.R., *High order embedded Runge-Kutta formulae*, J. Comp. Appl. Math., **7** (1981), 67-76.

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