J Math Chem (2009) 46:896–912
DOI 10.1007/s10910-009-9560-2

ORIGINAL PAPER

Symbolic derivation of Runge–Kutta–Nyström order conditions

Ch. Tsitouras · I. Th. Famelis

Received: 9 January 2007 / Accepted: 30 September 2008 / Published online: 19 July 2009

Abstract Runge–Kutta–Nyström methods possess a separate theory from the classical Runge–Kutta schemes for the derivation of their order conditions and principal truncation error terms. A new code using the Mathematica programming language properties and tensor products is proved very efficient in this task.

Keywords Rooted trees · Integer partitions · Truncation error · Mathematica

1 Introduction

Second order ordinary differential equations (ODEs) are widely used to model physical problems. Thus, methods for the numerical treatment of such ODEs are of great importance. There exist various classes of methods for the numerical solution of second order ODE problems and the class of Runge–Kutta–Nyström (RKN) methods are amongst the most popular ones. The construction of such methods require the derivation and the solution of equations called "order conditions". Such a procedure is a tedious task since the number of the nonlinear order conditions to be solved increases as the order of a method increases.

Thus, the use of a computer algebra system, such as Mathematica, for both the derivation and the solution of the order conditions is needed. Here, our concern is

Ch. Tsitouras (✉)
Department of Applied Sciences, TEI of Chalkis, 34400 Psahna, Greece
e-mail: tsitoura@teihal.gr
URL: http://users.ntua.gr/tsitoura/

I. Th. Famelis
Department of Mathematics, TEI of Athens, 12210 Egaleo, Greece
e-mail: ifamelis@teiath.gr
URL: http://math.teiath.gr/ifamelis/

to furnish a code for the construction of the order conditions. There have only been codes for the case of Runge–Kutta methods. Keiper [12] was probably the first who wrote a package for the symbolic manipulation program Mathematica. However that first package was limited in deriving low order conditions. Later, around 1993–1994 four researchers presented their proposal on this subject. In Hosea [11], a recurrence due to Albrecht [1] for generating order conditions is refined to produce truncation error coefficients. His code written in ANSI C, is called RKTEC and is available from `Netlib`. Harrison [10], and Papakostas [17] suggested the tensor notation deriving very interesting symbolic codes. That early package due to Papakostas helped a lot for the truncation error calculations in a series of papers of our group [20,22,23,19,24]. Later Sofroniou [21], gave an integrated package for deriving Runge–Kutta order conditions. Then Papakostas [18], proposed to avoid the derivation of trees in a such a package. Finally Famelis et al. [6], presented a very efficient code for the derivation of Runge–Kutta order conditions and our present work is an extension of that work. The only code for generating RKN trees that we are aware is due to Okunbor [15]. His program was build in the lines of Keiper code and does not compete the software presented here since it fails at high orders.

In the following section we outline the theory of RKN order conditions. Then we present the elements of Combinatorial Mathematics and Tree Theory have been used to approach the construction of a powerful and efficient symbolic package for the derivation of Runge–Kutta–Nyström order conditions and principal truncation error terms. In our approach the tree construction as matrix products produces a very fast and portable package which is cheap in memory usage too.

2 Runge–Kutta–Nyström order conditions

Runge–Kutta–Nyström methods are used to solve second order ODE problem

$$y'' = f(t, y(t)), \quad t \geq t_0,$$
$$y(t_0) = y_0, \quad y'(t_0) = y'_0.$$

These methods make no use of the past approximations and after getting the value y_n and y'_n as the numerical approximations of $y(t_n)$ and $y'(t_n)$ the methods proceed to the evaluation of y_{n+1} and y'_{n+1} as an estimation of $y(t_{n+1}) = y(t_n + h_n)$ and $y'(t_{n+1}) = y'(t_n + h_n)$ respectively, according to the following formulae:

$$y_{n+1} = y_n + h_n y'_n + h_n^2 \sum_{j=1}^{s} b_j f_j,$$
$$y'_{n+1} = y'_n + h_n \sum_{j=1}^{s} b'_j f_j,$$
$$f_j = f(t_n + c_j h_n, y_n + c_j h_n y'_n + h_n^2 \sum_{k=1}^{s} a_{ik} f_k).$$

This is the s-stage Runge–Kutta–Nyström method. The method's coefficients are usually represented by the Butcher tableau

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & & a_{1s} \\ c_2 & a_{21} & a_{22} & & a_{2s} \\ \vdots & & & & \\ c_s & a_{s1} & a_{s2} & & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \\ & b'_1 & b'_2 & \cdots & b'_s \end{array},$$

or in matrix form

$$\begin{array}{c|c} c & A \\ \hline & b^T \\ & b'^T \end{array}$$

with $c \in \Re^s$, $b, b' \in \Re^s$, and $A \in \Re^{s \times s}$.
As in Hairer et al. [8] we may restrict ourselves to systems of autonomous differential equations

$$(y^J)'' = f^J(y^1, \ldots, y^m),$$

In order to derive the Runge–Kutta–Nyström order conditions we must expand the theoretical solutions $y(t_n + h_n)$ and $y'(t_n + h_n)$ and the numerical solutions y_{n+1} and y'_{n+1}, as Taylor Series about the point (t_n, t_n). Considering

$$\epsilon = \begin{bmatrix} y(t_n + h_n) - y_{n+1} \\ y'(t_n + h_n) - y'_{n+1} \end{bmatrix}$$

we get expressions for the local truncation error of the method. For a method to be of order p, we must select the coefficients of the method so that

$$\epsilon = \bar{\Lambda} h^{p+1} + O(h^{p+2}),$$

where $\bar{\Lambda} = [\Lambda, \Lambda']^T$ is called principal local truncation error term. The equations that the coefficients of a method must fulfill so that a method attains a desired order are called order conditions.

The Taylor expansion of the theoretical solution expressions involve derivatives of y^J. The six first derivatives are the following,

$$(y^J)^{(1)} = y'^J$$
$$(y^J)^{(2)} = f^J(y)$$

$$
\begin{aligned}
(y^J)^{(3)} &= \sum_K \frac{\partial f^J}{\partial y^K}(y)\ y'^K, \\
(y^J)^{(4)} &= \sum_{K,L} \frac{\partial^2 f^J}{\partial y^K\ \partial y^L}(y)\ y'^K\ y'^L + \sum_L \frac{\partial f^J}{\partial y^L}(y)\ f^L(y), \\
(y^J)^{(5)} &= \sum_{K,L} \frac{\partial^3 f^J}{\partial y^K\ \partial y^L\ \partial y}(y)\ y'^K\ y'^L\ y' \\
&\quad +3\sum_{M,L} \frac{\partial^2 f^J}{\partial y^M\ \partial y^L}(y)\ f^M(y)\ y'^L + \sum_{M,L} \frac{\partial f^J}{\partial y^L}(y)\ \frac{\partial f^L}{\partial y^M}(y)\ y'^M
\end{aligned}
$$

In a more concise notation [7], we can write the above as

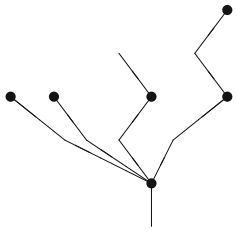
$$
\begin{aligned}
y^{(3)} &= f'(y)y',\ y^{(4)} = f''(y)(y', y') + f'(y)f(y), \\
y^{(5)} &= f'''(y', y', y') + 3f''(y)(f(y), y') + f'(y)f'(y)y', \text{etc.}
\end{aligned}
$$

When we try to expand the numerical solution we get even more complicated expressions.

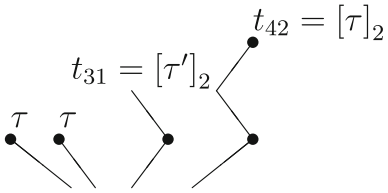
J. C. Butcher [2] established in the 1960 a theory based in tree theory for deriving the order conditions of a Runge–Kutta method. His book, is recommended for the interested reader. A simplified version of that theory can be found in Lambert [13]. The extension of the tree theory for the case of Runge–Kutta–Nyström methods can be found in [8] where the SN-trees (Special Nyström trees) are defined. Coleman [5], in the derivation of order conditions for two step hybrid methods, chooses a slightly different family of trees. T_2 trees are each of the SN-Trees grafted onto a meagre root. This approach is similar to the one given in [9]. In both cases the results are the same with slight modifications in the notation. For the presentation of our code we follow Coleman's approach.

The rooted trees of T_2 have two kind of vertices, meagre vertices which are represented by a point and fat vertices which are represented by a larger dot, •. Following Coleman [5], let $\emptyset$ be the empty tree, τ' the single meagre vertex tree and τ the following tree:

be members of set of trees T_2. We then define the rest of the trees recursively so that $t = [t_1, t_2, \ldots, t_m]_2 \in T_2$ if $t_1, t_2, \ldots, t_m \in T_2$. The root of such a tree is a meagre vertex and the tree is obtained by connecting the roots of the trees $t_1, t_2, \ldots, t_m$ to a new fat vertex, and then connecting that vertex to a new meagre root.



For example for the above tree we may write $t = [\tau, \tau, t_{31}, t_{42}]_2 = \left[\tau^2, t_{31}, t_{42}\right]_2$ where



Now, we can define some functions on T_2 rooted trees:

- Order $r(t)$:

$$r\left(\left[t_1^{n_1}, t_2^{n_2}, \ldots, t_k^{n_k}\right]_2\right) = 2 + n_1 r(t_1) + \cdots + n_k r(t_k)$$

with $r(\emptyset) = 0$, $r(\tau') = 1$ and $r(\tau) = 2$.

- Symmetry $\sigma(t)$:

$$\sigma\left(\left[t_1^{n_1}, t_2^{n_2}, \ldots, t_k^{n_k}\right]_2\right) = n_1! \cdots n_k! \sigma\left(t_1\right)^{n_1} \cdots \sigma\left(t_k\right)^{n_k}$$

with $\sigma(\emptyset) = 1$, $\sigma(\tau') = 1$ and $\sigma(\tau) = 1$.

- Density $\gamma(t)$:

$$\gamma\left(\left[t_1^{n_1}, t_2^{n_2}, \ldots, t_k^{n_k}\right]_2\right) = r\left(\left[t_1^{n_1}, t_2^{n_2}, \ldots, t_k^{n_k}\right]_2\right) \cdot \left(r\left(\left[t_1^{n_1}, t_2^{n_2}, \ldots, t_k^{n_k}\right]_2\right) - 1\right) \cdot \gamma\left(t_1\right)^{n_1} \cdots \cdot \gamma\left(t_k\right)^{n_k}$$

with $\gamma(\emptyset) = 1$, $\gamma(\tau') = 1$ and $\gamma(\tau) = 2$.

- Elementary weights $\Psi(t)$:

$$\Psi\left(\left[t_1^{n_1}, t_2^{n_2}, \ldots, t_k^{n_k}\right]\right) = \left(\Psi(t_1)^{n_1}\right) * \left(\Psi(t_2^{n_2})\right) * \cdots * \left(\Psi(t_k)^{n_k}\right)$$

with $\psi(\tau') = c$ and $\psi(\tau) = Ae$ where $e = [1, 1, \ldots, 1]^T \in \Re^s$.

- Elementary differentials $F(t)$:

$$F\left(\left[t_1^{n_1}, t_2^{n_2}, \ldots, t_k^{n_k}\right]\right) = f^{(n_1+\cdots+n_k)}\left(\underbrace{F\left(t_1\right), \ldots, F\left(t_1\right)}_{n_1 \text{ times}}, \ldots, \underbrace{F\left(t_k\right), \ldots, F\left(t_k\right)}_{n_k \text{ times}}\right)$$

with $F(\emptyset) = y$, $F(\tau') = y'$ and $F(\tau) = f(y)$, while "$*$" denotes the component-wise product between vectors:

$$[u_1 \;\; u_2 \cdots u_n]^T * [v_1 \;\; v_2 \cdots v_n]^T = [u_1 v_1 \;\; u_2 v_2 \cdots u_n v_n]^T.$$

This operation has the less priority. Parentheses, powers and dot products are always evaluated before "*". In the same sense powers follow the present definition, e.g., $c^2 = c * c$, $c^3 = c * c * c$, etc.

A Runge–Kutta–Nyström method is of order p if and only if

$$X(t) = \frac{1}{\sigma(t)} \left(b\Psi(t) - \frac{1}{\gamma(t)} \right) = 0, \tag{1}$$

for every $t \in T_2$ with $r(t) \leq p$ and

$$X'(t) = \frac{1}{\sigma(t)} \left(b'\Psi(t) - \frac{r(t)}{\gamma(t)} \right) = 0, \tag{2}$$

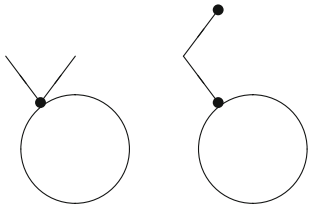
for every $t \in T_2$ with $r(t) \leq p + 1$. The above relations define the order conditions, which are linear in the components of b and b' and nonlinear in the components of A, c and relates them to set of the rooted trees T_2. Under the assumption

$$b^T = b'^T * (e - c),$$

condition (2) implies (1).

The number of order conditions increases rapidly as desired order increases. The formation of expressions of order conditions or principal local truncation error term is a tedious task when is done by hand, even if we follow the Butcher's theory. Simplifying conditions reduce the number of order conditions.

Let two trees t_1 and t_2 of T_2 which differ only in the part which is outside the circle.



t_1 t_2

Under the assumption that

$$Ae = \frac{c^2}{2}, \tag{3}$$

Table 1 Order conditions from 1 to 5

Order	Equation
1	$b'^T \cdot e = 1$
2	$b'^T \cdot c = \frac{1}{2}$,
3	$b'^T \cdot c^2 = \frac{1}{3}$,
4	$b'^T \cdot c^3 = \frac{1}{4}$,
4	$b'^T \cdot A \cdot c = \frac{1}{24}$,
5	$b'^T \cdot c^4 = \frac{1}{5}$,
5	$b'^T \cdot A \cdot c^2 = \frac{1}{60}$,
5	$b'^T \cdot (c * A \cdot c) = \frac{1}{30}$.

Table 2 Total number of conditions to achieve order p

Order p	1	2	3	4	5	6	7	8	9	10
Number of conditions for y	0	1	2	4	7	13	23	43	79	151
Number of conditions for y'	1	2	4	7	13	23	43	79	151	288
Number of conditions for y using (3)	0	1	2	3	5	8	13	22	37	64
Number of conditions for y' using (3)	1	2	3	5	8	13	22	37	64	112

holds then the order conditions coming from these two trees are the same. Using this assumption for a method of order 5 we have to solve only eight equations given in Table 1.

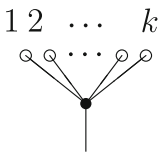
If we want to compute the coefficients for the local truncation error terms we have to consider all the order conditions for both b and b'.

The number of order conditions are given in Table 2. There are actually two series of numbers shown in the second and fourth rows of this table. Enumeration formulas for Nyström trees can be found in [3,4].

3 Tree theory and partitions

A tree is a mathematical object defined to be a connected linear graph which contains no cycles. A tree with one node, the root, distinguished from all other nodes is called a rooted tree. We have considered a specific class of rooted trees the T_2 trees. According to the existing theory, an one-to-one relation can be defined between the set of order p conditions and the T_2 rooted trees with p and $p+1$ nodes. So, the formation of the trees with $p+1$ nodes (and p nodes) can lead us to the corresponding order conditions of order p.

To understand the procedure of constructing all T_2 trees of order $p+1$ we will need elements from combinatorial mathematics and the fact that such a tree with $p+1$ nodes (of order $p+1$) can be constructed by taking trees with cumulative order $p-1$ obtained and connecting their roots to a new fat vertex, and then connecting that vertex to a new meagre root. In other words, the set of trees with $p+1$ nodes can be formed by taking combinations with repetition of k trees with cumulative order $p-1$.



A very important concept is generating functions [14]. Let $(\alpha_0, \alpha_1, \ldots, \alpha_r, \ldots)$ be a symbolic representation of a sequence of events (or in more simple situations a sequence of numbers). The function

$$F(x) = \alpha_0\mu_0(x) + \alpha_1\mu_1(x) + \cdots + \alpha_r\mu_r(x) + \cdots$$

is called the **ordinary generating function** of the sequence $(\alpha_0, \alpha_1, \ldots, \alpha_r, \ldots)$ where the $(\mu_0, \mu_1, \ldots, \mu_r, \ldots)$ is a sequence of functions of x that are used as indicators. The **indicator functions** are usually chosen in such way that no two distinct sequences will yield the same generating function. Generating functions are usually used to enumeration problems of combinatorial mathematics, such as in combinations of objects, but can be used to construct (generate) the elements of the sequence $(\alpha_0, \alpha_1, \ldots, \alpha_r, \ldots)$ as well.

If we set as $T^i = \{t_{i\#} | where\ t_{i\#}\ is\ a\ T_2\ rooted\ tree\ of\ order\ i\}$, and as $F_i(x)$ the generating function of combinations objects taken from T_i with repetition then

$$F_i(x) = \prod_{t_{i\#} \in T^i} \left(1 + t_{i\#}x + t_{i\#}^2x^2 + \cdots + t_{i\#}^n x^n + \cdots\right).$$

In this case the sequence $(\alpha_0, \alpha_1, \ldots, \alpha_r, \ldots)$ is the set T^i. Expanding this relation can be written as

$$F_i(x) = 1 + C(i,1)x + C(i,2)x^2 + \cdots + C(i,n)x^n + \cdots$$

where $C(i,k)$ is an expression for the combinations with repetition of order i objects (e.g., trees) in k positions. This is given as a sum of all possible combinations where each combination of objects is represented as a product of these objects, [14, p. 30]. For instance

$$\begin{aligned} F_3(x) &= \left(1 + t_{31}x + t_{31}^2x^2 + \cdots\right)\left(1 + t_{32}x + t_{32}^2x^2 + \cdots\right) \\ &= 1 + (t_{31} + t_{32})x + (t_{31}^2 + t_{32}^2 + t_{31}t_{32})x^2 + \cdots \\ &= 1 + C(3,1)x + C(3,2)x^2 + \cdots + C(3,n)x^n + \cdots . \end{aligned}$$

This approach can be used, as well, in the the case of rooted trees enumeration problems. For that purpose we set $t_{i\#} = 1$ in the above relations and so $C(i,n)$ is a number [14, pp. 31–32], [16, pp. 125–126].

In our case, using the above theory, we can form the generating function of the set of rooted trees. Taking into consideration that every tree can be formed by taking combinations with repetition of other trees and grafting them together and then adding the root, then the generating function of rooted trees is

$$
\begin{aligned}
F(x) &= x^2 \prod F_i(x) \\
&= x^2(1 + C(1,1)x + C(1,2)x^2 + \cdots + C(1,n)x^n + \cdots) \\
&\quad (1 + C(2,1)x + C(2,2)x^2 + \cdots + C(2,n)x^n + \cdots) \\
&\quad (1 + C(3,1)x + C(3,2)x^2 + \cdots + C(3,n)x^n + \cdots) \cdots .
\end{aligned}
$$

Expanding the above product and collecting the proper powers of x, all trees of order $p+1$, that are produced by the grafting of k other trees, can be determined by the term $x^2 \sum_{k=1}^{p-1} \tilde{C}_k x^k$ where the $\tilde{C}_k$ is the sum of products of k trees with cumulative order $p-1$ (or equivalently the combinations with repetition of k trees with cumulative order $p-1$). So, for our purpose, it is essential to form the products $t_{\pi_1\#}^{i_1} t_{\pi_2\#}^{i_2} \cdots t_{\pi_k\#}^{i_k}$ where $t_{\pi_j\#} \in T^{\pi_j}$ and $i_1\pi_1 + i_2\pi_2 + \cdots + i_k\pi_k = p-1, k = 1, 2, \ldots, p-1$. This connects our problem with the set of unrestricted partitions of an integer. The term x^2 that multiplies the sum represents the grafting of the trees that are combined in the k positions onto a new root.

An **unrestricted partition** of an integer p, is by definition, a collection of integers, without regard of order, whose sum is p. For example, an unrestricted partition of 5 is 1, 1, 1, 2. This is usually written as $1^3 2$. So, an unrestricted partition of p has the form $\pi_1^{i_1} \pi_2^{i_2} \cdots \pi_k^{i_k}$ where $i_1\pi_1 + i_2\pi_2 + \cdots + i_k\pi_k = p$, a notation similar to the one used for the trees.

In conclusion, in order to construct all the rooted trees of order $p+1$ we have to find all the unrestricted partitions of $p-1$ and for each of them to form all the corresponding combinations with repetition $t_{\pi_1\#}^{i_1} t_{\pi_2\#}^{i_2} \cdots t_{\pi_k\#}^{i_k}$ selecting $t_{\pi_j\#}$ from T^{π_j}.

In order to program the procedure mentioned above in a symbolic computation environment, such as Mathematica, the tree oriented notation is not the best choice. In a programming point of view the best way is to work by forming the matrix notation products of the expressions involving the method coefficients following the lines of the previous section. This simplifies the whole procedure and produces a faster code. Moreover, the main concern is neither the derivation of the trees themselves nor the order condition expressions with the elementary differentials. The main consideration is to produce the order conditions or the principal error terms. So, in the code, that will be presented in the next section, the t_{ij} are not the trees but the corresponding matrix multiplication expressions $\Psi(t_{ij})$. Moreover the outer products is formed based on pointwise multiplication.

4 The symbolic code

Following [6], we have build a package with the name RKN for the symbolic environment of Mathematica. The backbone of the package are the modules **T1**, **T0**, **G1**, **G0** and **S**.

The functions **T1** and **T0** call the module **T** to produce a list of the method coefficient matrix notation products corresponding to the rooted trees of a given order. This is done recursively. To achieve that **T** applies exactly the ideas of tree construction from the previous section. The functions need **Partition** of Mathematica package

Combinatorica to build the unrestricted partitions of an integer and the modules **Combinations** and **Combinations2** to form combinations without repetition.

Using the same ideas and the formulae given in Sect. 2, module **S** builds a list with elements the values of symmetry of the rooted trees of a given order and **G0**, **G1**, and **G** a list of the corresponding density values.

After setting as working directory the directory which the package file is stored the package can be loaded by giving the following input:

```
In[1] :=<<RKN
```

Using the package functions we can either form the order conditions that should be fulfilled so that a Runge–Kutta–Nyström method attains a given order or the principal truncation error terms of a method of a given order.

To get the list of the order conditions the following commands should be typed in the Mathematica environment:
`BOC[a,b,c,e,order]` and `DBOC[a,db,c,e,order]`

In the above commands `a, b, c, e, db` can be Mathematica symbols and `order` a number for the desired order. The symbols `a, b, db` and `c` correspond to the method matrices according the Butcher Tableau notation, where `db` corresponds to b' and `e` to an array of ones with dimension the number of stages of the method.

In the following example we get as an outcome a list with elements lists of order 1 to 5 conditions.

```
In[2] :=BOC[a, b, c, e, 5]
Out[2] :={{}, {-1/2 + b.e}, {-1/6 + b.c}, {-1/12 +
b.c^2, -1/24 + b.a.e}, {-1/20 + b.c^3, -1/40 +
b.(c*a.e), -1/120 + b.a.c}}

In[3] :=DBOC[a, db, c, e, 5]
Out[3] :={{-1 + db.e}, {-1/2 + db.c},
{-1/3 + db.c^2, -1/6 + db.a.e}, {-1/4 + db.c^3, -1/8 +
db.c*a.e, -1/24 + db.a.c}, {-1/5 + db.c^4, -1/30 +
db.c*a.c, -1/10 + db.c^2*a.e, -1/20 + db.a.e^2, -1/60 +
db.a.c^2, -1/120 + db.a.a.e}}
```

Moreover, `a, b, c, e, db` can be matrices in the Mathematica notation of lists. These matrices may have either symbolic or numeric entries. In the former case the outcome is going to be the analytic expressions of the order conditions that should become zero to attain the desired order. In the latter case the output is a list of the quantities that the method fail to fulfill the order conditions.

Changing in the lines of the code the value of the variable `s` from 0 to 1 we can get the order conditions when we assume that the simplifying assumption (3) holds. If we do so the results are the following:

```
In[4] :=BOC[a, b, c, e, 5]
Out[4] :={{}, {-1/2 + b.e}, {-1/6 + b.c},
{-1/12 + b.c^2}, {-1/20 + b.c^3, -1/120 + b.a.c}}
```

```
In[5] :=DBOC[a, db, c, e, 5]
Out[5] :={{-1 + db.e}, {-1/2 + db.c}, {-(1/3) +
db.c^2}, {-1/4 + db.c^3, -1/24 + db.a.c}, {-1/5 +
db.c^4, -1/30 + db.(c*a.c), -1/60 + db.a.c^2}}
```

To get the list of the principal truncation error terms the following command should be typed in the Mathematica environment:
`BTR[a,b,c,e,order]` and `DBTR[a,db,c,e,order]`

In the above command `a, b, c, e, db` can be Mathematica symbols and order a number for the desired order or matrices with symbolic or numeric entries as mentioned above.

In the following example we get as an outcome a list with elements of the principal truncation error terms for a method of order 5.

```
In[6] :=BTR[a, b, c, e, 5]
Out[6] :={-1/720 + b.a.a.e, 1/2*(-1/360 +
b.a.c^2), -1/180 + b.(c*a.c), 1/2*(-1/120 + b.(a.e)^2),
1/2*(-1/60 + b.(c^2*a.e)), 1/24*(-1/30 + b.c^4)}
```

```
In[7] :=DBTR[a, db, c, e, 5]
Out[7] :={-1/720 + db.a.a.c, -1/240 +
db.a.(c*a.e), 1/6*(-1/120 + db.a.c^3), -1/144
+ db.(c*a.a.e), 1/2*(-1/72 + db.(c*a.c^2)), -1/72 +
db.(a.c*a.e), 1/2*(-1/36 + db.(c^2*a.c)), 1/2*(-1/24
+ db.(c*(a.e)^2)), 1/6*(-1/12 + db.(c^3*a.e)),
1/120*(-1/6 + db.c^5)}
```

Moreover, `a, b, c, e, db` can be matrices in the Mathematica notation of lists. These matrices may have either symbolic or numeric entries. In the former case the outcome is going to be the analytic expressions of the order conditions that should become zero to attain the desired order. In the latter case a list of the quantities that the method fail to fulfill the order conditions.

Once again, changing the value of the variable `s` from 0 to 1 in the code we can get the coefficients when we assume that the simplifying assumption (3) holds. As we have mentioned in such a case all the trees should be considered. Now the output is the following:

```
In[8] :=BTR[a, b, c, e, 5]
Out[8] :={1/2*(-1/360 + b.a.c^2),
1/2*(-1/360 + b.a.c^2), -1/180 + b.(c*a.c), 1/8*(-1/30
+ b.c^4), 1/4*(-1/30 + b.c^4), 1/24*(-1/30 + b.c^4)}
```

```
In[9] :=DBTR[a, db, c, e, 5]
Out[9] :={-1/720 + db.a.a.c,
1/2*(-1/120 + db.a.c^3), 1/6*(-1/120 + db.a.c^3),
1/2*(-1/72 + db.(c*a.c^2)), 1/2*(-1/72 + db.(c*a.c^2)),
1/2*(-1/36 + db.(c^2*a.c)), 1/2*(-1/36 + db.(c^2*a.c)),
1/8*(-1/6 + db.c^5), 1/12*(-1/6 + db.c^5), 1/120*(-1/6 +
 db.c^5)}
```

Table 3 Number of trees and computation times (in seconds) for RK and RKN symbolic derivation algorithms

Runge–Kutta		Order	Runge–Kutta–Nyström	
Time	Trees		Trees	Time
0	20	6	10	0
0.02	48	7	20	0.01
0.02	115	8	36	0.02
0.03	286	9	72	0.02
0.09	719	10	137	0.05
0.13	1842	11	275	0.08
0.28	4766	12	541	0.11
0.61	12486	13	1098	0.17
1.45	32973	14	2208	0.28
4.00	87811	15	4521	0.41
10.47	235381	16	9240	0.72
29.50	634847	17	19084	1.24
80.44	1721159	18	39451	2.19
–	–	19	82113	4.13
–	–	20	171240	8.02
–	–	21	358794	16.47
–	–	22	753460	33.19

The algorithm presented here is competitive to the one given in [6] for RK methods. In Table 3 we present computation times for our algorithms for RK and RKN methods and the corresponding number of trees for various orders. Actually we used the RKTrunc function implemented in [6] and DBTR function of our new package. The RK algorithm was incapable for deriving 19-th order conditions due to memory limitations. The comparison with the method given in [15] is not presented since it fails for orders greater than 12.

The comparisons were performed in the Mathematica 5.1 environment [25] on a Pentium 2.4 MHz system having 512 Mbytes RAM memory which was running Windows XP–SP2 Operating System.

5 Conclusions

In this paper we have presented, for the first time, a set of very efficient routines for the symbolic derivation of Runge–Kutta–Nyström order conditions and principal local truncation error coefficients. The code is fast and economical in computer memory. Finally, another remarkable fact is that the source code of the new package covers a little more than three journal pages and this helps in the direction of better and easier understanding.

Appendix

The Mathematica package implementing the code:

```
BeginPackage[ "RKN`",{"DiscreteMath `Combinatorica`"}];

Clear["RKN`*" ]
```

```
     DBTR::usage = " DBTR[a,db,c,e,order,s]  finds RKN
                    principal truncation error of order
                    order+1 for DB. "
     BTR::usage = " BTR[a,b,c,e,order,s]  finds RKN
                   principal truncation error of
                   order order+1 for B. "
     DBOC::usage = " DBOC[a,db,c,e,order,s] finds RKN
                    order conditions of orders 1
                    to order. "
     BOC::usage = " DBOC[a,b,c,e,order,s] finds RKN
                    order conditions of orders 1
                    to order. "

Begin["`Private`"]; Clear[ "RKN`Private`*" ];

     DBTR[aa_,dbb_,cc_,ee_,orderr_] :=
1/S1[orderr+1]*(T1[aa,dbb,cc,ee,orderr+1]- G1[orderr+1]);
     BTR[aa_,bb_,cc_,ee_,orderr_] :=
1/S0[orderr+1]*(T0[aa,bb,cc,ee,orderr+1]- G0[orderr+1]);
     DBOC[aa_,dbb_,cc_,ee_,orderr_] :=
Table[Map[First,Split[Sort[(T1[aa,dbb,cc,ee,j]-G1[j])]]],
    {j,1,orderr}];
    BOC[aa_,bb_,cc_,ee_,orderr_] :=
Table[Map[First,Split[Sort[(T0[aa,bb,cc,ee,j]-G0[j])]]],
    {j,1,orderr}];

RunLengthEncode[x_List] := (Through[{First,Length}[#1]]&)
    /@ Split[x];
Combinations[list_, num_] :=
 Module[{i},
        Table[Map[Prepend[#, list[[i]]]&,
                 Flatten[Combinations[list, num - 1]
                        [[Array[Identity, Length[list]-
                        i +1, i]]], 1
                        ], {1}],
              {i, 1, Length[list]} ]]/;  (num > 1) ;
Combinations[list_, 1] := Combinations[list, 1]
    = Map[{{#}}&, list];
Combinations2[list_, num_] :=
 Apply[Times, Flatten[Combinations[list, num], 1],
    {1} ]/;  (num > 1);
Combinations2[list_, 1] := list;
(*-------------------------------------------------*)

s=0; T[a_,c_,e_,1] = {c}; T[a_,c_,e_,2] = {a.e};
    G[1] = {1};
```

```
G[2] = {1/2}; S[1] = {1}; S[2]={1}; Switch[s,1,
    T[a_,c_,e_,2]={c^2};
G[2] = {1}; S[2]={2};  ];
G[order_] := G[order] =
 Module[{temp},
        temp = Map[Combinations2[G[#[[1]]], #[[2]]]&,
            Map[RunLengthEncode[#]&, Partitions[order-2],
                {1}], {2}];
        temp = Apply[Times, temp, {3}];
        temp = Map[Prepend[#, Times]&, temp, 1];
        temp = Apply[Outer, temp, {1}];
        temp = Flatten[temp];
        temp = (1/((order-1)*order)) * temp
       ];

T[a_,c_,e_,order_] := T[a,c,e,order] =
 Module[{temp},
        temp = Map[Combinations2[T[a,c,e,#[[1]] ],
            #[[2]]]&, Map[RunLengthEncode[#]&,
            Partitions[order-2], {1}], {2}];
        temp = Map[CoverList[#]&, temp, {3}];
        temp = Apply[MyOuter, temp, {1}];
        temp = Flatten[temp, 1];
        temp = temp /. CoverList[every_] -> every;
        temp = Map[(a . #)&, temp, {1}]
       ];

MyOuter[lists__] := Flatten[Outer[Times, lists],
    Length[{lists}] - 1];

S[order_] := S[order] =
 Module[{temp},
    temp = Map[Combinations2[MapIndexed[ff, S[#[[1]]]],
    #[[2]]] &, Map[RunLengthEncode[#] &,
    Partitions[order-2], {1}], {2}];
    temp=temp /. {ff[a_, b_]^p_ -> Factorial[p]*a^p,
    ff[a_, b_] -> a};
    temp = Apply[MyOuter, temp, {1}];
    temp = Flatten[temp, 1]
               ];

T1[a_,db_,c_,e_,1] ={db.e};  G1[1] = {1};
T1[a_,db_,c_,e_,order_] :=T1[a,db,c,e,order] =
 Module[{temp},
        temp = Map[Combinations2[T[a,c,e,#[[1]] ],
        #[[2]]]&, Map[RunLengthEncode[#] &,
```

```
        Partitions[order-1], {1}], {2}];
        temp = Map[CoverList[#]&, temp, {3}];
        temp = Apply[MyOuter, temp, {1}];
        temp = Flatten[temp, 1];
        temp = temp /. CoverList[every_] -> every;
        temp = Map[(db . #)&, temp, {1}]
       ]

G1[order_] := G1[order] =
 Module[{temp},
        temp = Map[Combinations2[G[#[[1]]], #[[2]]]&,
            Map[RunLengthEncode[#]&, Partitions[order-1],
            {1}], {2}];
        temp = Apply[Times, temp, {3}];
        temp = Map[Prepend[#, Times]&, temp, 1];
        temp = Apply[Outer, temp, {1}];
        temp = Flatten[temp];
        temp = (1/order) * temp
       ];

S1[order_] :=S[order+1];
T0[a_,b_,c_,e_,1] = {};  G0[1] = {};
T0[a_,b_,c_,e_,2] = {b.e};
G0[2] = {1/2};
T0[a_,b_,c_,e_,order_] :=
T0[a,b,c,e,order] =
 Module[{temp},
        temp = Map[Combinations2[T[a,c,e,#[[1]] ],
        #[[2]]]&, Map[RunLengthEncode[#] &,
        Partitions[order-2], {1}], {2}];
        temp = Map[CoverList[#]&, temp, {3}];
        temp = Apply[MyOuter, temp, {1}];
        temp = Flatten[temp, 1];
        temp = temp /. CoverList[every_] -> every;
        temp = Map[(b . #)&, temp, {1}]
       ]

G0[order_] := G0[order] =
 Module[{temp},
        temp = Map[Combinations2[G[#[[1]]], #[[2]]]&,
            Map[RunLengthEncode[#]&, Partitions[order-2],
            {1}], {2}];
        temp = Apply[Times, temp, {3}];
        temp = Map[Prepend[#, Times]&, temp, 1];
        temp = Apply[Outer, temp, {1}];
        temp = Flatten[temp];
```

```
        temp = (1/((order-1)*order)) * temp
       ];

       S0[order_] :=S[order];
End[]; EndPackage[];
```

The code can be retrieved from authors web pages. A brief description follows:

`MyOuter`: Performs outer products of elements of lists.

`Combinations`: Produces the non ordered combinations without repetition of n objects taken form the elements of a list.

`Combinations2`: Returns the products of the elements taken from Combinations.

`RunLengthEncode`: Gives a list of pairs (x, y) which correspond to of element x of length y in a list.

`T`: This function applies the main ideas of our approach. Using Combinations2 and recursion produces a list with all the possible matrix expressions which correspond to trees with cumulative order $p-1$. The function CoverList is a dummy function which is used to protect from the outer product, the elements of the lists (in levelspec 3) which are produced by the recursion. These elements which are expressions that correspond to the branches of each tree are multiplied using MyOuter and the list is flattened to produce the full list needed. Then the CoverList protection is taken out and the expressions are multiplied by a to meet the fact that a new node is added to each tree.

`s`: $s = 0$ no simplifying assumptions, $s = 1$ for $Ae = \frac{c^2}{2}$

`T1`: Taking the results of T gives a list with the expressions for y' which corresponds to the grafting of the trees with cumulative order p into a new fat vertex and connecting that vertex to a new meagre root.

`T0`: Taking the results of T gives a list with the expressions for y which corresponds to the grafting of the trees with cumulative order $p-1$ into a new fat vertex and connecting that vertex to a new meagre root.

`G`: Using Combinations2 and recursion produces a list with all the values of density function $\gamma(t)$ which correspond to all possible trees with cumulative order $p-1$.

`G0`: Taking the results of G gives a list with the density values for y' which corresponds to the grafting of the trees with cumulative order $p-1$ into a new fat vertex and connecting that vertex to a new meagre root.

`G1`: Taking the results of G gives list with the density values for y which corresponds to the grafting of the trees with

cumulative order p into a new fat vertex and connecting that vertex to a new meagre root.

`S:` Using Combinations2 and recursion produces a list with elements all the values of symmetry function $\sigma(t)$ which correspond to all possible trees with cumulative order $p-1$.

`S0:` The list of the symmetries for the trees corresponding to y' after the grafting is the same as the list of symmetries of all possible trees with cumulative order p

`S1:` The list of the symmetries for the trees corresponding to y after the grafting is the same as the list of symmetries of all possible trees with cumulative order $p+1$.

References

1. P. Albrecht, Numerical treatment of O.D.E.s: the theory of A-methods. Numer. Math. **47**, 59–87 (1985)
2. J.C. Butcher, *The Numerical Analysis of Ordinary Differential Equations* (Wiley, Chichester, 1987)
3. M.P. Calvo, J.M. Sanz-Serna, Order conditions for canonical Runge-Kutta-Nyström methods. BIT **32**, 131–142 (1992)
4. M.P. Calvo, J.M. Sanz-Serna, High order symplectic Runge-Kutta-Nyström methods. SIAM J. Sci. Comput. **14**, 1237–1252 (1993)
5. J.P. Coleman, Order conditions for a class of two-step methods for $y'' = f(x, y)$. IMA J. Numer. Anal. **23**, 197–220 (2003)
6. I.Th. Famelis, S.N. Papakostas, Ch. Tsitouras, Symbolic derivation of Runge-Kutta order conditions. J. Symb. Comput. **37**, 311–327 (2004)
7. E. Hairer, C. Lubich, G. Wanner, *Geometric Numerical Integration* (Springer, Berlin, 2002)
8. E. Hairer, S.P. Nørsett, G. Wanner, *Solving Ordinary Differential Equations I* , 2nd edn. (Springer, Heidelberg, 1993)
9. E. Hairer, G. Wanner, A theory of Nyström methods. Numer. Math. **25**, 383–400 (1976)
10. A.J. Harrison, Runge-Kutta order conditions package, http://library.wolfram.com/infocenter/MathSource/1524/
11. M.E. Hosea, A new recurrence for computing Runge-Kutta truncation error coefficients. SIAM J. Numer. Anal. **32**, 1989–2001 (1997)
12. J. Keiper, NumericalMath'Butcher'.m, Ver. 1.2, Wolfram Research Inc., 1989
13. J.D. Lambert, *Numerical Methods for Ordinary Differential Systems* (Wiley, Chichester, 1991)
14. C.L. Liu, *Introduction to Combinatorial Mathematics* (McGraw–Hill, New York, 1968)
15. D.I. Okunbor, Canonical integration methods for Hamiltonian dynamical systems, Report UIUCDS-R-92-1885, Univ. Illinois, Urbana, 1992
16. A. Papaioannou, *Enumeration of Graphs (in Greek)* (NTUA, Athens, 2000)
17. Papakostas S.N., Unpublished software, 1992–1993
18. Papakostas S.N., Algebraic analysis and the development of numerical ODE solvers of the Runge–Kutta type (in Greek), Ph.D. Dissertation, Athens, 1996
19. S.N. Papakostas, Ch. Tsitouras, High algebraic order, high phase-lag order Runge-Kutta and Nyström pairs. SIAM J. Sci. Comput. **21**, 747–763 (1999)
20. S.N. Papakostas, Ch. Tsitouras, G. Papageorgiou, A general family of explicit Runge-Kutta pairs of orders 6(5). SIAM J. Numer. Anal. **33**, 917–936 (1996)
21. M. Sofroniou, Symbolic Derivation of Runge–Kutta methods. J. Symb. Comput. **18**, 265–296 (1994)
22. Ch. Tsitouras, A parameter study of a Runge–Kutta pair of orders 6(5). Appl. Math. Lett. **11**, 65–69 (1998)
23. Ch. Tsitouras, S.N. Papakostas, Cheap error estimation for Runge–Kutta pairs. SIAM J. Sci. Comput. **20**, 2067–2088 (1999)
24. Ch. Tsitouras, Optimal Runge–Kutta pairs of orders 9(8). Appl. Numer. Math. **38**, 123–134 (2001)
25. Wolfram S., *The Mathematica Book*, 5th edn. (Wolfram Med., Champaign, 2003)