Article Submitted to Journal of Symbolic Computation

Symbolic Derivation of Runge-Kutta Order Conditions.

I.Th. Famelis *[1], S. N. Papakostas[2] and Ch. Tsitouras[3]

[1]*Department of Mathematics, Faculty of Applied Mathematics and Physical Sciences, National Technical University of Athens, Zografou Campus, GR15780 Athens, Greece*

[2]*Hegemo SA, 369 Sigrou Avenue, GR17654 Athens, Greece*

[3]*Department of Applied Sciences,TEI of Chalkis, GR 34400 Psahna, Greece*

Abstract

Tree theory, partitions of integer numbers, combinatorial mathematics and computer algebra are the basis for the construction of a powerful and efficient symbolic package for the derivation of Runge Kutta order conditions and principal truncation error terms.

1. Introduction

Ordinary Differential Equations (ODEs) are widely used to model physical problems. Thus, methods for the numerical treatment of ODEs are of great importance. There exist various classes of methods for the numerical solution of ODE problems and the class of Runge–Kutta (RK) methods are amongst the most popular ones. The construction of such methods needs the derivation and 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 computer algebra system, such as Mathematica, for both the derivation and the solution of the order conditions is needed. It is impossible to give a general algorithm for the solution of order conditions for all families and for all algebraic orders. Actually, such algorithms can be given for each family separately (Papakostas et. al., 1996),(Tsitouras, 1998),(Tsitouras, 2001). Here, our concern is to furnish a code for the construction of the order conditions. In the past there have been several attempts to meet that problem. As a first

*Corresponding author, Email: ifamelis@math.ntua.gr thought, a straight forward approach utilizing Taylor series expansions is very inefficient. Keiper (1990) 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 1993-94 four researchers presented their proposal about this subject.

In Hosea (1995), a recurrence due to Albrecht for generating order conditions is refined to produce truncation error coefficients. The code written in ANSI C, is called RKTEC and is available from `Netlib`. Then Harisson (1994), and Papakostas (1992-93) suggested the tensor notation deriving very interesting symbolic codes. That early package due to Papakostas helped a lot in the truncation error calculations in a some papers of our group (Papakostas et. al., 1996), (Tsitouras, 1998). Finally Sofroniou (1994), gave an integrated package for the derivation of Runge–Kutta order conditions.

Papakostas (1996), proposed to avoid the derivation of trees in such a package. So, in the following sections we present the theory of RK order conditions and the elements of Combinatorial Mathematics and Tree Theory we have used to approach the construction of a powerful and efficient symbolic package for the derivation of Runge–Kutta order conditions and principal truncation error terms. Approaching the tree construction as matrix products produces a very fast and portable package which is cheap in memory usage too.

2. Runge–Kutta Order Conditions

The most common ODE problem is the initial value problem

$$y' = f(t, y(t)), \quad y(x_0) = y_0. \tag{1}$$

Runge–Kutta type methods are the basic representatives of the class of single step numerical methods for the numerical solution of the above problem. Such methods make no use of the past approximations. When getting the value y_k as the numerical approximation of $y(t_k)$ the methods proceed to the evaluation of y_{k+1} as an estimation of $y(t_{k+1}) = y(t_k + h_k)$ according to the following formula:

$$y_{k+1} = y_k + h_k \sum_{i=1}^{s} b_i f_i \tag{2}$$

with

$$f_i = f\left(t_k + c_i h_k, y_k + h_k \sum_{j=1}^{s} a_{ij} f_j\right), \quad i = 1, 2, \cdots, s.$$

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

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

or in matrix form

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array},$$

with $c \in \Re^s$, $b \in \Re^s$, and $A \in \Re^{s\times s}$.

For RK methods we always assume that the simplifying assumption (the row-sum condition) holds:

$$c_i = \sum_{j=1}^{s} a_{ij}, \quad i = 1, 2, \cdots, s$$

or in matrix notation $c = Ae$, $e = [1, 1, \cdots, 1]^T \in \Re^s$. The above condition ensures that all points where f is evaluated are first order approximations.

Setting $t' = 1$, then (1) reduces, without loss of generality, to the most convenient autonomous system $y' = f(y)$ for which the row-sum condition is essential.

When advancing a Runge–Kutta method, applied to the above simplified problem, we actually try to approximate the corresponding Taylor series expansion:

$$y(t_{k+1}) = y(t_k) + h_k \cdot f(y_k) + \ldots + \frac{h_k^p}{p!} \cdot f^{(p)}(y_k) + O(h^{p+1}). \tag{3}$$

Moreover, in the case of a system we have to consider

$$y'' = \frac{\partial f(y(t))}{\partial t} = \frac{\partial f}{\partial y} f = f' f,$$

$$y''' = \frac{\partial^2 f}{\partial y^2}(f, f) + \frac{\partial f}{\partial y} \cdot \frac{\partial f}{\partial y} f = f''(f, f) + f' f' f,$$

$$\begin{aligned} y^{(4)} &= \tfrac{\partial^3 f}{\partial y^3} \cdot (f, f, f) + \tfrac{\partial f}{\partial y} \cdot \tfrac{\partial f}{\partial y} \cdot \tfrac{\partial f}{\partial y} f + \\ &\quad \tfrac{\partial f}{\partial y} \cdot \tfrac{\partial^2 f}{\partial y^2} \cdot (f, f) + 3 \cdot \tfrac{\partial^2 f}{\partial y^2} \cdot \left(\tfrac{\partial f}{\partial y} \cdot f, f\right) \\ &= f''' \cdot (f, f, f) + f' f' f' f + f' f''(f, f) + 3 f''(f' f, f), \\ &\cdots \end{aligned}$$

and so on. The elementary differentials $f''(f, f)$, $f'''(f, f, f)$, $f'f''(f, f)$, $f''(f', f, f)$

are Frechet derivatives (Lambert, 1991, pp158). In case that equation (1) is a scalar autonomous problem we may use a simplified approach since for instance $f'f''(f,f)$ $=f''(f',f,f)$=$f'f''f^2 \in \Re$ (Papageorgiou and Tsitouras , 2002).

On the other hand we may expand f_i's in the numerical solution around the point (t_k, y_k) and derive the expression:

$$\begin{aligned} y_{k+1} = y_k + hq_{11}y_k' + h^2 q_{21} y_k'' + h^3 \left(q_{31} f''(f,f) + q_{32} f'f'f\right) + \\ \left(q_{41} f'''\left(f,f,f\right) + q_{42} f'f''\left(f,f\right) + q_{43} f''\left(f'f,f\right) + q_{44} f'f'f'f\right) + \cdots \end{aligned} \tag{4}$$

where q_{ij} depend exclusively on the coefficients A, b, c.

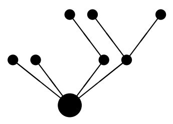
Subtracting (4) from (3) we get the local truncation error of the method.

$$\begin{aligned} y\left(t_{k+1}\right) - y_{k+1} = h\left(q_{11} - 1\right) f + h^2 \left(q_{21} - \tfrac{1}{2}\right) \cdot \tfrac{\partial f}{\partial y} f + \\ h^3 \left(\left(q_{31} - \tfrac{1}{6}\right) f''(f,f) + \left(q_{32} - \tfrac{1}{6}\right) f'f'f\right) + \cdots \end{aligned} \tag{5}$$

A RK method has order p if the local truncation error behaves like $O(h^{p+1})$. That means that in the above expression the coefficients of powers of h up to p are zero. The equations that must hold so that a RK method attains order p are called order conditions. So, requiring $t_{11} = q_{11} - 1 = 0$, $t_{21} = q_{21} - 1/2 = 0$, $t_{31} = q_{31} - 1/6 = 0$, $t_{32} = q_{32} - 1/6 = 0$, we get the order conditions for a third order method. The coefficient of h^{p+1} is called the principal local truncation error term. The minimization of this term is one of the consideration in the procedure of constructing RK methods.

J. C. Butcher established in the 60's a theory based in tree theory for deriving the order conditions of a Runge–Kutta method. His book (Butcher, 1987), is recommended for the interested reader. A simplified version of that theory can be found in (Lambert, 1991).

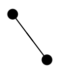
A rooted tree of $n-$th order is a set of n nodes joined by lines. One of the nodes is the root and its branches are not allowed to grow together again.



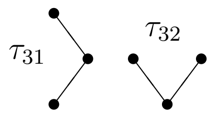
A rooted tree.

The unique matching between a rooted tree and an order condition becomes clear after splitting the tree, cutting the branches beginning from the root. In order to work with trees we consider the following notation. A tree will be named with a notation τ_{ij} where the first index states the number of nodes and the second index an internal enumeration in the class of trees with i nodes. So, the one node tree • will be τ_{11}. The two nodes rooted tree will be

τ_{21}

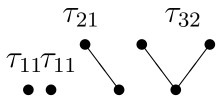


and the two rooted trees with three nodes:



Every tree with p nodes can be constructed by taking trees with cumulative order $p-1$ and graft them onto a new root. Using this point of view we can notate every tree as $\tau = [\tau_{ij}, \tau_{kl}, \ldots, \tau_{mn}]$ where $\tau_{ij}, \tau_{kl}, \ldots, \tau_{mn}$ are the trees which are grafted to a new root to form τ. If the same tree (e.g. τ_{kl}) in the grafting appears n times, we replace all τ_{kl} in this notation with one τ_{kl}^{n}.

So, for our example tree we may write $\tau = [\tau_{11}, \tau_{11}, \tau_{21}, \tau_{32}] = [\tau_{11}^{2}, \tau_{21}, \tau_{32}]$ where



are grafted to a new root.

We define the following functions on rooted trees, (Lambert, 1991, pp164):

- Order $r(\tau)$:

$$r\left(\left[\tau_{ij}^{n_1}, \tau_{kl}^{n_2}, \cdots, \tau_{mn}^{n_p}\right]\right) = 1 + n_1 r(\tau_{ij}) + \ldots + n_p r(\tau_{mn})$$

- Symmetry $\sigma(\tau)$:

$$\sigma\left(\left[\tau_{ij}^{n_1}, \tau_{kl}^{n_2}, \cdots, \tau_{mn}^{n_p}\right]\right) = n_1! \cdots n_k! \sigma\left(\tau_{ij}\right)^{n_1} \cdots \sigma\left(\tau_{mn}\right)^{n_p}$$

- Density $\gamma(\tau)$:

$$\gamma\left(\left[\tau_{ij}^{n_1}, \tau_{kl}^{n_2}, \cdots, \tau_{mn}^{n_p}\right]\right) = r\left(\left[\tau_{ij}^{n_1}, \tau_{kl}^{n_2}, \cdots, \tau_{mn}^{n_p}\right]\right) \gamma\left(\tau_{ij}\right)^{n_1} \cdots \gamma\left(\tau_{mn}\right)^{n_p}$$

- Elementary weights $\Psi(\tau)$:

$$\Psi\left(\left[\tau_{ij}^{n_1}, \tau_{kl}^{n_2}, \cdots, \tau_{mn}^{n_p}\right]\right) = \left(A\left(\Psi\left(\tau_{ij}\right)\right)^{n_1}\right) * \left(A\left(\Psi\left(\tau_{kl}\right)\right)^{n_2}\right) * \cdots * \left(A\left(\Psi\left(\tau_{mn}\right)\right)^{n_p}\right)$$

- Elementary differentials $F(\tau)$:

$$F\left(\left[\tau_{ij}^{n_1}, \tau_{kl}^{n_2}, \cdots, \tau_{mn}^{n_p}\right]\right) = f^{(n_1+\cdots+n_p)}\left(\underbrace{F\left(\tau_{ij}\right), \cdots, F\left(\tau_{ij}\right)}_{n_1 \quad times}, \cdots, \underbrace{F\left(\tau_{mn}\right), \cdots, F\left(\tau_{mn}\right)}_{n_p \quad times}\right)$$

with $r(\bullet) = \sigma(\bullet) = \gamma(\bullet) = 1$, $F(\bullet) = f$ and $\Psi(\bullet) = e$, while " $*$ " denotes the component–wise product between vectors[†].

According to Butcher's theory, equation (5) has the form

[†] $w = u * v$, with u, v, $w \in \Re^s$ means $w_i = u_i v_i$ for $i = 1, 2, \cdots, s$. In the same sense $w^k = w * w * \cdots * w$, k times.

$$y\left(t_{k+1}\right)-y_{k+1}=\sum_{i=1}^{\infty}\sum_{\tau\in T_i}h^i\frac{1}{\sigma\left(\tau\right)}\left(b\Psi\left(\tau\right)-\frac{1}{\gamma\left(\tau\right)}\right)F\left(\tau\right)$$

where T_i is the set of rooted trees of order i, σ and γ are integer–valued functions of τ, $\Psi\in\Re^s$, is a certain composition of A,b,c, with a form that depends only on τ and F is an elementary differential.

So, a Runge–Kutta method is of order p if and only if

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

for every $\tau\in T_i$, for $i=1,2,\cdots,p$. The above relation defines the order conditions, which are linear in the components of b and nonlinear in the components of A,c and relates them to the rooted trees.

Now, we can derive the order conditions.

1st order

$$X\left(\,\bullet\right)=\frac{1}{\sigma\left(\,\bullet\right)}\left(b\Psi\left(\,\bullet\right)-\frac{1}{\gamma\left(\,\bullet\right)}\right)=b\cdot e-1=0$$

2nd order $\tau_{21}=[\tau_{11}]$

$$X\left(\tau_{21}\right)=\frac{1}{\sigma\left(\tau_{21}\right)}\left(b\Psi\left(\tau_{21}\right)-\frac{1}{\gamma\left(\tau_{21}\right)}\right)=b\cdot c-\frac{1}{2}=0$$

3rd order $\tau_{31}=[\tau_{21}]$ and $\tau_{32}=[\tau_{11}^2]$ producing

$$X(\tau_{31})=bAc-1/6=0$$

and

$$X(\tau_{32})=1/2\cdot(bc^2-1/3)=0.$$

There are four rooted trees of fourth order and so we can continue to produce all the required order conditions for a RK method to have order p or to compute the principal local truncation error term. The order conditions are the same for all classes of RK methods (e.g. Explicit RK, Implicit RK) (Hairer et. al. , 1993, pg 207) .

The number of order conditions increases rapidly (Table 1) as desired order increases and moreover there is a barrier (Table 2) in the maximum attained order related to the method's number of stages (Riordan, 1958, pg 138), (Hairer et. al. , 1993). Whilst, 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.

Table 1: Total number of conditions to achieve order p.

Order p	1	2	3	4	5	6	7	8	9	10
No. of conditions	1	2	4	8	17	37	85	200	486	1205

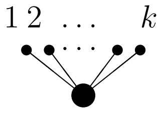
Table 2: Order barriers.

No. of RK stages	1	2	3	4	5	6	7	8	9
Max. attained order	1	2	3	4	4	5	6	6	7

3. Tree Theory and Partitions

As we have mentioned in previous sections, 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. According to Butcher's theory, an one-to-one relation can be defined between the set of order p conditions and the set of rooted trees with p nodes. So, the formation of the trees with p nodes can lead us to the corresponding order conditions of order p.

To understand the procedure of constructing all trees of order p we will need elements from combinatorial mathematics and the fact that a tree with p nodes (of order p) can be constructed by taking trees with cumulative order $p-1$ and graft them onto a new root. In other words, the set of trees with p nodes can be formed by taking combinations with repetition of k trees with cumulative order $p-1$.



A very important concept is generating functions (Liu, 1968). 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

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

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 $T_i = \{t | t \textit{ rooted tree of order } i\}$, and as $F_i(x)$ the generating function of combinations objects taken from T_i with repetition then

$$F_i(x) = \prod_{t \in T_i} (1 + tx + t^2x^2 + \cdots + t^nx^n + \cdots).$$

In this case the sequence $(\alpha_0, \alpha_1, \dots, \alpha_r, \dots)$ 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, (Liu, 1968, p 30). For instance

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

This approach can be used, as well, in the the case of rooted trees enumeration problems. For that purpose we set $t = 1$ in the above relations and so $C(i,n)$ is a number (Liu, 1968, pp 31-32), (Papaioanou, 2000, 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, then the generating function of rooted trees is

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

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 $x\sum_{k=1}^{p}\tilde{C}_k x^k$ where the $\tilde{C}_k$ is the sum of products of k trees with cumulative order p (or equivalently the combinations with repetition of k trees with cumulative order p). So, for our purpose, it is essential to form the products $\tau_{\pi_1\#}^{i_1}\tau_{\pi_2\#}^{i_2}\cdots\tau_{\pi_k\#}^{i_k}$ where $\tau_{\pi_j\#}\in T_{\pi_j}$ and $i_1\pi_1+i_2\pi_2+\cdots+i_k\pi_k = p$, $k = 1,2,\dots,p$. This connects our problem with the set of unrestricted partitions of an integer. The term x 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 and for each of them to form all the corresponding combinations with repetition $\tau_{\pi_1\#}^{i_1}\tau_{\pi_2\#}^{i_2}\cdots\tau_{\pi_k\#}^{i_k}$ selecting $\tau_{\pi_j\#}$ 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 τ_{ij} are not the trees but the corresponding matrix multiplication expressions $\Psi(\tau_{ij})$. Moreover the outer products is formed based on pointwise multiplication.

4. New Symbolic Code

Following (Papakostas, 1996), we have build a package with the name Trees16 for the symbolic environment of Mathematica. The backbone of the package are the modules **TT**, **S** and **G**.

The function **T** calls the module **TT** 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 **TT** applies exactly the ideas of tree construction from the previous section. **TT** needs function **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 section 2, module **S** builds a list with elements the values of **symmetry** of the rooted trees of a given order 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]:=≪ Trees16

Using the package functions we can either form the order conditions that should be fulfilled so that a Runge–Kutta 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 command should be typed in the Mathematica environment:

RKCond[a,b,c,e,order]

In the above command a, b, c, e can be Mathematica symbols and order a number for the desired order. The symbols a, b and c correspond to the method matrices according the Butcher Tableau notation 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 6 conditions.

In[2]:=RKCond[a, b, c, e, 6]
Out[2]:={

$$\{-1+b\cdot e\},\{-\left(\frac{1}{2}\right)+b\cdot c\},\{-\left(\frac{1}{6}\right)+b\cdot a\cdot c,-\left(\frac{1}{3}\right)+b\cdot c^2\},$$
$$\{-\left(\frac{1}{24}\right)+b\cdot a\cdot a\cdot c,-\left(\frac{1}{12}\right)+b\cdot a\cdot c^2,-\left(\frac{1}{8}\right)+b\cdot(c\,a\cdot c),-\left(\frac{1}{4}\right)+b\cdot c^3\},$$

$$\{-\left(\frac{1}{120}\right)+b\cdot a\cdot a\cdot a\cdot c,-\left(\frac{1}{60}\right)+b\cdot a\cdot a\cdot c^2,-\left(\frac{1}{40}\right)+b\cdot a\cdot(c\,a\cdot c),$$
$$-\left(\frac{1}{20}\right)+b\cdot a\cdot c^3,-\left(\frac{1}{30}\right)+b\cdot(c\,a\cdot a\cdot c),-\left(\frac{1}{15}\right)+b\cdot\left(c\,a\cdot c^2\right),$$
$$-\left(\frac{1}{20}\right)+b\cdot(a\cdot c)^2,-\left(\frac{1}{10}\right)+b\cdot\left(c^2\,a\cdot c\right),-\left(\frac{1}{5}\right)+b\cdot c^4\},$$

$$\{-\left(\frac{1}{720}\right)+b\cdot a\cdot a\cdot a\cdot a\cdot c,-\left(\frac{1}{360}\right)+b\cdot a\cdot a\cdot a\cdot c^2,-\left(\frac{1}{240}\right)+b\cdot a\cdot a\cdot(c\,a\cdot c),$$
$$-\left(\frac{1}{120}\right)+b\cdot a\cdot a\cdot c^3,-\left(\frac{1}{180}\right)+b\cdot a\cdot(c\,a\cdot a\cdot c),-\left(\frac{1}{90}\right)+b\cdot a\cdot\left(c\,a\cdot c^2\right),$$
$$-\left(\frac{1}{120}\right)+b\cdot a\cdot(a\cdot c)^2,-\left(\frac{1}{60}\right)+b\cdot a\cdot\left(c^2\,a\cdot c\right),-\left(\frac{1}{30}\right)+b\cdot a\cdot c^4,$$
$$-\left(\frac{1}{144}\right)+b\cdot(c\,a\cdot a\cdot a\cdot c),-\left(\frac{1}{72}\right)+b\cdot\left(c\,a\cdot a\cdot c^2\right),-\left(\frac{1}{48}\right)+b\cdot(c\,a\cdot(c\,a\cdot c)),$$

$$-\left(\frac{1}{24}\right)+b\cdot\left(c\,a\cdot c^3\right),-\left(\frac{1}{72}\right)+b\cdot(a\cdot c\,a\cdot a\cdot c),-\left(\frac{1}{36}\right)+b\cdot\left(a\cdot c\,a\cdot c^2\right),$$
$$-\left(\frac{1}{36}\right)+b\cdot\left(c^2\,a\cdot a\cdot c\right),-\left(\frac{1}{18}\right)+b\cdot\left(c^2\,a\cdot c^2\right),-\left(\frac{1}{24}\right)+b\cdot\left(c\,(a\cdot c)^2\right),$$
$$-\left(\frac{1}{12}\right)+b\cdot\left(c^3\,a\cdot c\right),-\left(\frac{1}{6}\right)+b\cdot c^5\}\}$$

Moreover, A, b, c, e 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.

To get the list of the principal truncation error terms the following command should be typed in the Mathematica environment:
RKTrunc[a,b,c,e,order]
In the above command a, b, c, e 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 6.

In[3]:=RKTrunc[a, b, c, e, 6]

Out[3]:=

$$\{-\left(\frac{1}{5040}\right)+b\cdot a\cdot a\cdot a\cdot a\cdot a\cdot c, \frac{-\left(\frac{1}{2520}\right)+b\cdot a\cdot a\cdot a\cdot a\cdot c^2}{2},$$

$$-\left(\frac{1}{1680}\right)+b\cdot a\cdot a\cdot a\cdot (c\,a\cdot c)\,, \frac{-\left(\frac{1}{840}\right)+b\cdot a\cdot a\cdot a\cdot c^3}{6},$$

$$-\left(\frac{1}{1260}\right)+b\cdot a\cdot a\cdot (c\,a\cdot a\cdot c)\,, \frac{-\left(\frac{1}{630}\right)+b\cdot a\cdot a\cdot (c\,a\cdot c^2)}{2},$$

$$\frac{-\left(\frac{1}{840}\right)+b\cdot a\cdot a\cdot (a\cdot c)^2}{2}, \frac{-\left(\frac{1}{420}\right)+b\cdot a\cdot a\cdot (c^2\,a\cdot c)}{2},$$

$$\frac{-\left(\frac{1}{210}\right)+b\cdot a\cdot a\cdot c^4}{24}, -\left(\frac{1}{1008}\right)+b\cdot a\cdot (c\,a\cdot a\cdot a\cdot c)\,,$$

$$\frac{-\left(\frac{1}{504}\right)+b\cdot a\cdot (c\,a\cdot a\cdot c^2)}{2}, -\left(\frac{1}{336}\right)+b\cdot a\cdot (c\,a\cdot (c\,a\cdot c))\,,$$

$$\frac{-\left(\frac{1}{168}\right)+b\cdot a\cdot (c\,a\cdot c^3)}{6}, -\left(\frac{1}{504}\right)+b\cdot a\cdot (a\cdot c\,a\cdot a\cdot c)\,,$$

$$\frac{-\left(\frac{1}{252}\right)+b\cdot a\cdot (a\cdot c\,a\cdot c^2)}{2}, \frac{-\left(\frac{1}{252}\right)+b\cdot a\cdot (c^2\,a\cdot a\cdot c)}{2},$$

$$\frac{-\left(\frac{1}{126}\right)+b\cdot a\cdot (c^2\,a\cdot c^2)}{4}, \frac{-\left(\frac{1}{168}\right)+b\cdot a\cdot \left(c\,(a\cdot c)^2\right)}{2},$$

$$\frac{-\left(\frac{1}{84}\right)+b\cdot a\cdot (c^3\,a\cdot c)}{6}, \frac{-\left(\frac{1}{42}\right)+b\cdot a\cdot c^5}{120},$$

$$-\left(\frac{1}{840}\right)+b\cdot (c\,a\cdot a\cdot a\cdot a\cdot c)\,, \frac{-\left(\frac{1}{420}\right)+b\cdot (c\,a\cdot a\cdot a\cdot c^2)}{2},$$

$$-\left(\frac{1}{280}\right)+b\cdot (c\,a\cdot a\cdot (c\,a\cdot c))\,, \frac{-\left(\frac{1}{140}\right)+b\cdot (c\,a\cdot a\cdot c^3)}{6},$$

$$-\left(\frac{1}{210}\right)+b\cdot (c\,a\cdot (c\,a\cdot a\cdot c))\,, \frac{-\left(\frac{1}{105}\right)+b\cdot (c\,a\cdot (c\,a\cdot c^2))}{2},$$

$$\frac{-\left(\frac{1}{140}\right)+b\cdot \left(c\,a\cdot (a\cdot c)^2\right)}{2}, \frac{-\left(\frac{1}{70}\right)+b\cdot (c\,a\cdot (c^2\,a\cdot c))}{2},$$

$$\frac{-\left(\frac{1}{35}\right)+b\cdot (c\,a\cdot c^4)}{24}, -\left(\frac{1}{336}\right)+b\cdot (a\cdot c\,a\cdot a\cdot a\cdot c)\,,$$

$$\frac{-\left(\frac{1}{168}\right)+b\cdot(a\cdot c\,a\cdot a\cdot c^2)}{2},-\left(\frac{1}{112}\right)+b\cdot(a\cdot c\,a\cdot(c\,a\cdot c)),$$

$$\frac{-\left(\frac{1}{56}\right)+b\cdot(a\cdot c\,a\cdot c^3)}{6},\frac{-\left(\frac{1}{168}\right)+b\cdot(c^2\,a\cdot a\cdot a\cdot c)}{2},$$
$$\frac{-\left(\frac{1}{84}\right)+b\cdot(c^2\,a\cdot a\cdot c^2)}{4},\frac{-\left(\frac{1}{56}\right)+b\cdot(c^2\,a\cdot(c\,a\cdot c))}{2},$$
$$\frac{-\left(\frac{1}{28}\right)+b\cdot(c^2\,a\cdot c^3)}{12},\frac{-\left(\frac{1}{252}\right)+b\cdot(a\cdot a\cdot c)^2}{2},$$

$$\frac{-\left(\frac{1}{126}\right)+b\cdot(a\cdot c^2\,a\cdot a\cdot c)}{2},\frac{-\left(\frac{1}{63}\right)+b\cdot(a\cdot c^2)^2}{8},$$
$$-\left(\frac{1}{84}\right)+b\cdot(c\,a\cdot c\,a\cdot a\cdot c),\frac{-\left(\frac{1}{42}\right)+b\cdot(c\,a\cdot c\,a\cdot c^2)}{2},$$

$$\frac{-\left(\frac{1}{42}\right)+b\cdot(c^3\,a\cdot a\cdot c)}{6},\frac{-\left(\frac{1}{21}\right)+b\cdot(c^3\,a\cdot c^2)}{12},$$
$$\frac{-\left(\frac{1}{56}\right)+b\cdot(a\cdot c)^3}{6},\frac{-\left(\frac{1}{28}\right)+b\cdot\left(c^2\,(a\cdot c)^2\right)}{4},$$
$$\frac{-\left(\frac{1}{14}\right)+b\cdot(c^4\,a\cdot c)}{24},\frac{-\left(\frac{1}{7}\right)+b\cdot c^6}{720}\}.$$

5. Comparisons Conclusions

Nowadays, Sofroniou code is the one that is usually used to produce the Runge–Kutta methods order conditions and the principal local truncation error terms. The code is provided as a standard Mathematica package with the name **NumericalMath'Butcher**. The two functions of Butcher package are the **RungeKuttaOrderConditions[order]** and **ButcherPrincipalError[order]** .

We have compared the Sofroniou versus our New Symbolic Code both in time in seconds that is required and the memory in bytes needed by the system to perform the symbolic computation. For that purpose we have used the Mathematica build in functions **Timing** and **MemoryInUse**. The comparisons were performed in the Mathematica 4.0 environment on a Pentium III 600 Mhz system having 384 Mbytes RAM memory which was running Windows 2000 Server Operating System. The results are presented in Tables 3 - 6 and Figures 1-4 for various orders.

It is obvious that the New code is much faster and needs less memory compared to Sofroniou code. For high order methods the Sofroniou code time measurements are expanding making the usage of the functions practically difficult to use. On the other hand the New code time measurements remain low and make them handy to use. More over Sofroniou code has failed to work for order 17 methods as the system run out of memory resources and the Mathematica kernel was halted.

Hosea's algorithm is strictly numerical (using floating point numbers) and evaluates a somewhat different truncation error coefficients following Albrecht approach, e.g. $t_{32} = \frac{1}{2}bc^2 - bAc$ in this case. The results of the codes due to Harrison and Papakostas (Papakostas, 1992-93) are more or less similar to Sofroniou's results.

Finally, another remarkable fact is that the source code of the new package covers less than two journal pages and this helps in the direction of better and easier understanding.

Table 3: New code vs Sofroniou code order conditions time in secs.

order	New	Sofroniou	% Improvement
7	0.04	0.05	20
8	0.08	0.11	27
9	0.15	0.26	42
10	0.271	0.781	65
11	0.571	2.564	78
12	1.272	8.192	85
13	3.155	26.498	88
14	7.992	91.882	91
15	21.301	354.099	94
16	57.353	1550.97	96

Table 4: New code vs Sofroniou code order conditions Memory requirement in bytes.

order	New	Sofroniou	% Improvement
7	1759344	1373776	-28
8	1780744	1435760	-24
9	1834752	1592216	-15
10	1968608	1991960	1
11	2311640	3033264	24
12	3192328	5750344	45
13	5493136	12629496	57
14	11560104	30676680	62
15	27644224	78364656	65
16	70712824	205115872	66

6. Appendix

The Mathematica package implementing the new method.

```
BeginPackage[ "Trees16‘", {"DiscreteMath‘Combinatorica‘"}];
Clear["Trees16‘*" ]
```

Table 5: New code vs Sofroniou principal local truncation error terms time in secs.

order	New	Sofroniou	% Improvement
8	0.11	0.12	8
9	0.21	0.29	28
10	0.371	0.901	59
11	0.731	2.984	76
12	1.552	9.844	84
13	3.665	32.016	89
14	9.163	111.901	92
15	23.614	453.296	95
16	62.3	1909.82	97
17	170.575	∞	

Table 6: New code vs Sofroniou code principal local truncation error terms memory requirement in bytes.

order	New	Sofroniou	% Improvement
7	1780120	1452352	-23
8	1832712	1632760	-12
9	1961680	2091800	6
10	2297360	3293352	30
11	3160992	6437112	51
12	5439512	14438888	62
13	11453928	35634168	68
14	27523104	91734176	70
15	70747200	240942752	71
16	188029616	∞	

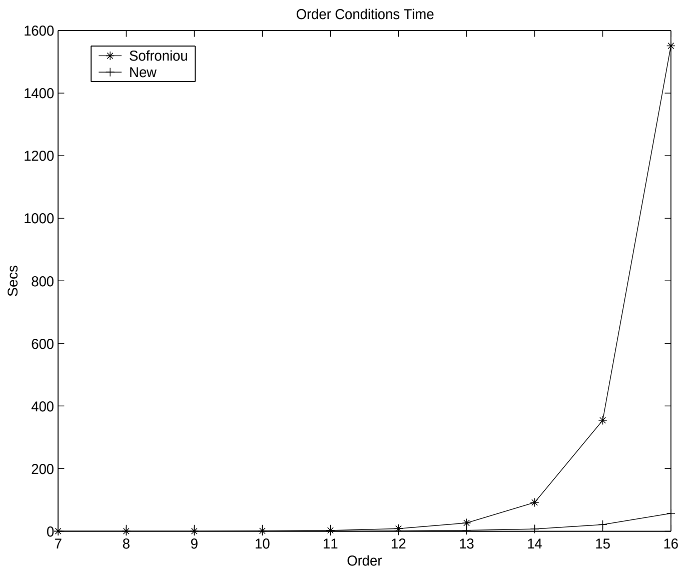


Figure 1: The results presented in plots

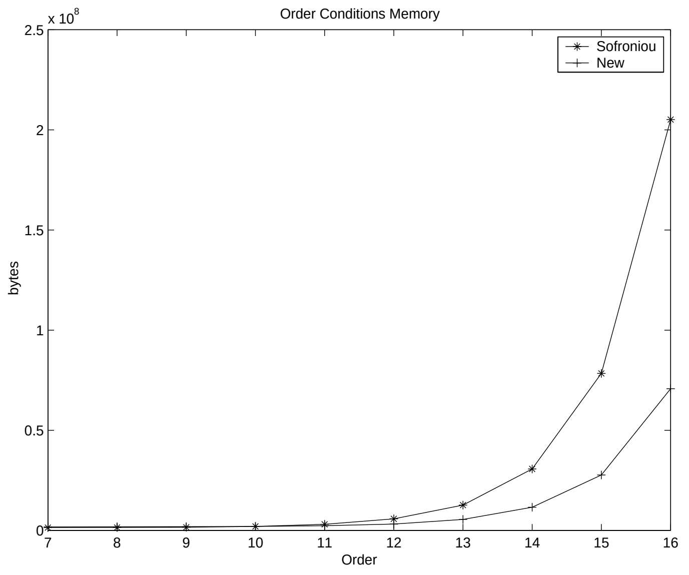


Figure 2: The results presented in plots

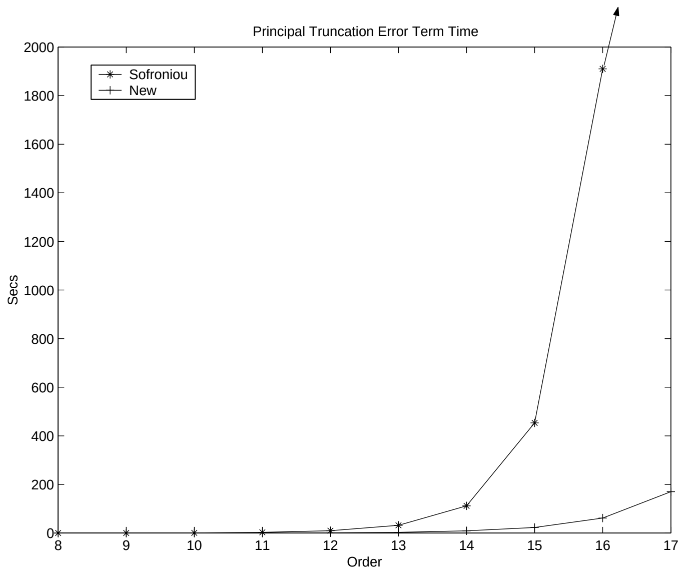


Figure 3: The results presented in plots

```
  RKTrunc::usage = " RKTrunc[a,b,c,e,order] finds RK principal truncation
error of order order+1. "
  RKCond::usage = " RKCond[a,b,c,e,order] finds RK order conditions
of orders 1 to order. "
  Begin["‘Private‘"];
  Clear["Trees16‘Private‘*"];
  RKTrunc[aa_,bb_,cc_,ee_,orderr_]:=
1/S[orderr+1]*(T[aa,bb,cc,ee,orderr+1]-G[orderr+1]);
```

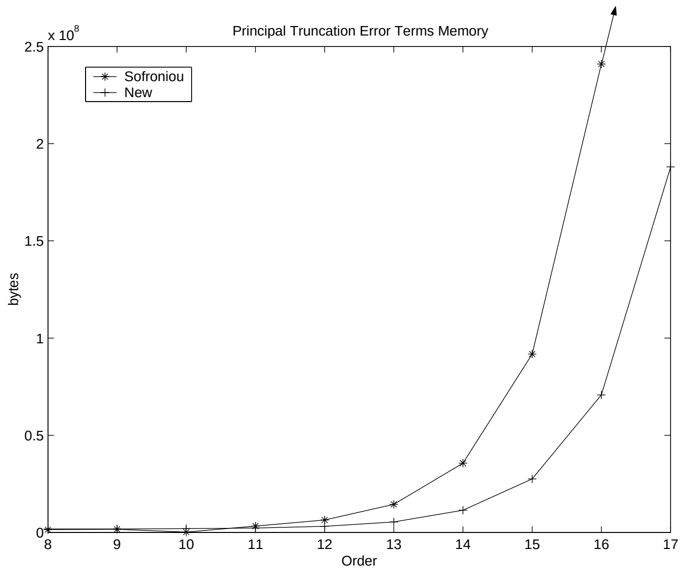


Figure 4: The results presented in plots

```
  RKCond[aa_,bb_,cc_,ee_,orderr_]:=
Table[T[aa,bb,cc,ee,i]-G[i],{i,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] := Compinations[list, 1] = Map[{{#}}&, list];
  Combinations2[list_, num_] :=
  Apply[Times, Flatten[Combinations[list, num], 1], {1}] /; (num > 1)
  Combinations2[list_, 1] := list;
  (*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)
  TT[a_,c_,e_,1] = {c}; G[1] = {1};
  G[order_] := G[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
   ];
```

```
TT[a_,c_,e_,order_] := TT[a,c,e,order] =
 Module[{temp},
 temp = Map[Combinations2[TT[a,c,e,#[[1]]]/.at->a, #[[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[(at . #)&, temp, {1}]  ];
T[a_,b_,c_,e_,1]= {b.e};
T[a_,b_,c_,e_,order_]:= TT[a,c,e,order] /. at -> b;
S[1] = {1};
S[order_] := S[order] =
 Module[{temp},
 temp = Map[Combinations2[MapIndexed[ff, S[#[[1]]]], #[[2]]] &,
Map[RunLengthEncode[#] &, Partitions[order-1], {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]
 ];
MyOuter[lists__] := Flatten[Outer[Times, lists],
Length[{lists}] - 1];
End[]
EndPackage[]
```

References

Butcher, J. C., The Numerical Analysis of Ordinary Differential Equations, (1987) *Wiley, Chichester*

Hairer, E., Nørsett, S.P., Wanner, G., Solving Ordinary Differential Equations I, second edition, Springer,(1993) *Heidelberg*

Harrison, A. J., Runge-Kutta Order Conditions Package *http://www.mathsource.com/Content/Enhancements/Numerical/0206-457*

Hosea, M. E., A new recurrence for computing Runge-Kutta truncation error coefficients (1997) *SIAM J. Numer. Anal.* **32** pp. 1989-2001

Keiper, J. NumericalMath'Butcher'.m *Wolfram Research Inc* (1990)

Lambert, J. D., Numerical methods for ordinary differential systems (1991) *Wiley, Chichester*

Liu, C. L., Introduction to Combinatorial Theory (1968) *Mac Grow–Hill*

Papageorgiou, G. and Tsitouras, Ch., Runge–Kutta pairs for scalar autonomous Initial Value Problems *Int. J. Comput. Math.* (2002) **80** pp. 201-209

Papaioannou, A., Enumeration of Graphs (in Greek) (2000) *NTUA, Athens*

Papakostas, S. N. Unpublished software (1992-1993)

Papakostas, S. N. Algebraic analysis and the development of numerical ODE solvers of the Runge–Kutta type, Ph.D. Thesis (in Greek) (1996) *Athens*

Papakostas, S. N. and Tsitouras, Ch., High algebraic order, high phase-lag order Runge-Kutta and Nyström pairs (1999) *SIAM J Sci. Comput.* **21** pp. 747-763

Papakostas, S. N., Tsitouras Ch., and Papageorgiou G., A general family of explicit Runge-Kutta pairs of orders 6(5) (1996) *SIAM J Numer. Anal.* **33** pp. 917-936

Riordan, J., An Introduction to Combinatorial Analysis (1958) *Wiley, N. York*

Sofroniou, M., Symbolic Derivation of Runge–Kutta methods (1994) *J. Symbol. Comput.* **18** pp. 265-296

Tsitouras, Ch., A parameter study of a Runge–Kutta pair of orders 6(5) (1998) *Appl. Math. Lett.* **11** pp 65-69

Tsitouras, Ch. and Papakostas, S. N., Cheap error estimation for Runge–Kutta pairs(1999) *SIAM J Sci. Comput.* **20** pp. 2067-2088

Tsitouras, Ch., Optimal Runge–Kutta pairs of orders 9(8) (2001) *Appl. Numer. Math.,* **38** pp. 123-134

Wolfram, S., Mathematica. Mathematica book v4.0 (1999) *Wolfram med. inc & Campridge Univ. Pr.*