Setup of Order Conditions for Splitting Methods*

Winfried Auzinger[1], Wolfgang Herfort[1], Harald Hofstätter[1], and Othmar Koch[2]

[1] Technische Universität Wien, Austria
w.auzinger@tuwien.ac.at, w.herfort@tuwien.ac.at,
hofi@harald-hofstaetter.at,
www.asc.tuwien.ac.at/~winfried, www.asc.tuwien.ac.at/~herfort,
www.harald-hofstaetter.at
[2] Universität Wien, Austria
othmar@othmar-koch.org,
www.othmar-koch.org

Abstract. For operator splitting methods, an approach based on Taylor expansion and the particular structure of its leading term as an element of a free Lie algebra is used for the setup of a system of order conditions. Along with a brief review of the underlying theoretical background, we discuss the implementation of the resulting algorithm in computer algebra, in particular using[3] Maple 18. A parallel version of such a code is described, and its performance on a computational node with 16 threads is documented.

Keywords: Evolution equations, splitting methods, order conditions, local error, Taylor expansion, parallel processing

1 Introduction

The construction of higher order discretization schemes of one-step type for the numerical solution of evolution equations is typically based on the setup and solution of a system of polynomial equations for a number of unknown coefficients. Classical examples are Runge-Kutta methods, and their various modifications, see e.g. [9].

To design particular schemes, we need to understand:

(i) how to generate a system of algebraic equations for the coefficients of the higher-order method sought,

(ii) how to solve the resulting system of polynomial equations.

Here we focus on (i) which depends on the particular class of methods one is interested in.[4] We consider *operator splitting methods,* which are based on the

* The final publication is available at Springer http://download.springer.com/

[3] Maple is a trademark of MapleSoft^{TM}.

[4] The aspect (ii) enters the discussion in [2]

idea of approximating the exact flow of an evolution equation by compositions of (usually two) separate subflows which are easier to evaluate. Splitting methods represent a very useful class of one-step methods for certain types of evolution equations, as for instance Schrödinger type equations, and if the operator splitting is done in an appropriate way, they have very good stability properties (possibly with complex instead of real coefficients for the case of parabolic equations). The more difficult problem is to find coefficients such that a higher order of accuracy is obtained, i.e., coping with (i) and (ii),

Computer algebra is an indispensable tool for dealing with (i). Typically there is a tradeoff between 'manual' a priori analysis and machine driven automatization. For operator-splitting methods, a well-known approach is based on a cumbersome recursive application of the Baker-Campbell-Hausdorff (BCH) formula, see [8]. Here we are advocating another approach, namely implementation of an algorithm for (i) which runs in a fully automatic mode. This approach is described in [1] and [2]; it is based on Taylor expansion and a theoretical result concerning the structure of the leading term in this expansion. This has the advantage that explicit knowledge of the BCH coefficients is not required. It may be called a generic, 'brute-force' approach. The efficiency of such a general algorithm cannot be optimal in an overall sense; on the other hand, it is easy to implement with optimal speedup on a parallel computer. Moreover, it can be easily adapted to special cases like coefficient symmetries, to operator splitting into more than two parts, and to pairs of splitting schemes akin to embedded Runge-Kutta methods.

In the present paper we focus on the implementation aspect, in particular in a parallel environment. The preparation of a parallel version was motivated by the computational complexity which strongly grows when the desired order is increased. Our parallel code scales in computation time at an (almost) optimal rate, and this speedup is of great practical advantage when trying out different variants, especially for more complex higher-order cases. This may also be viewed as a hardness test for parallelization in Maple.

Topic (ii) is not discussed in this paper. Details concerning the theoretical background and a discussion concerning concrete results and optimized schemes obtained are given in [2], and a collection of optimized schemes can be found at [3]. We note that a related approach has recently also been considered in [5].

In the rest of this introductory section we describe the mathematical background. In Section 2 we review our algorithm introduced in [1, 2] based on Taylor expansion of the local error. A parallel implementation is described in Section 3. Modifications and extensions are indicated in Section 4, and performance measures are documented in Section 5 by means of some examples.

1.1 Splitting Methods for the Integration of Evolution Equations

In many applications, the right hand side $F(u)$ of an evolution equation

$$\partial_t u(t) = F(u(t)) = A(u(t)) + B(u(t)), \quad t \geq 0, \quad u(0) \text{ given}, \tag{1}$$

splits up in a natural way into two terms $A(u)$ and $B(u)$, where the separate integration of the subproblems

$$\partial_t u(t) = A(u(t)), \qquad \partial_t u(t) = B(u(t))$$

is much easier to accomplish than for the original problem.

Example 1. The solution of a linear ODE system with constant coefficients,

$$\partial_t u(t) = (A + B)\, u(t),$$

is given by

$$u(t) = e^{t(A+B)}\, u(0).$$

The simplest splitting approximation ('Lie-Trotter'), starting at some initial value u and applied with a time step of length $t = h$, is given by

$$\mathcal{S}(h, u) = e^{hB}\, e^{hA}\, u \approx e^{h(A+B)} u.$$

This is not exact (unless $AB = BA$), but it satisfies

$$\|(e^{hB}\, e^{hA} - e^{h(A+B)})u\| = \mathcal{O}(h^2) \quad \text{for} \quad h \to 0,$$

and the error of this approximation depends on behavior of the commutator $[A, B] = AB - BA$. □

A general splitting method takes steps of the form[5]

$$\mathcal{S}(h, u) = \mathcal{S}_s(h, \mathcal{S}_{s-1}(h, \ldots, \mathcal{S}_1(h, u))) \approx \Phi_F(h, u), \tag{2a}$$

with

$$\mathcal{S}_j(h, v) = \Phi_B(b_j\, h, \Phi_A(a_j\, h, v)), \tag{2b}$$

where the (real or complex) coefficients a_j, b_j have to be found such that a certain desired order of approximation for $h \to 0$ is obtained.

The local error of a splitting step is denoted by

$$\mathcal{S}(h, u) - \Phi_F(h, u) =: \mathcal{L}(h, u). \tag{3}$$

For our present purpose of finding asymptotic order conditions it is sufficient to consider the case of a linear system, $F(u) = F\, u = A\, u + B\, u$ with linear operators A and B. We denote

$$A_j = a_j\, A,\ B_j = b_j\, B, \quad j = 1 \ldots s.$$

Then,

$$\mathcal{S}(h, u) = \mathcal{S}(h)u, \quad \mathcal{S}(h) = \mathcal{S}_s(h)\, \mathcal{S}_{s-1}(h) \cdots \mathcal{S}_1(h) \approx e^{hF}, \tag{4a}$$

with

$$\mathcal{S}_j(h) = e^{hB_j}\, e^{hA_j}, \quad j = 1 \ldots s. \tag{4b}$$

For the linear case the local error (3) is of the form $\mathcal{L}(h)u$ with the linear operator $\mathcal{L}(h) = \mathcal{S}(h) - e^{hF}$.

[5] By Φ_F we denote the flow associated with the equation $\partial_t u = F(u)$, and Φ_A, Φ_B are defined analogously.

1.2 Commutators

Commutators of the involved operators play a central role. For formal consistency, we call A and B the 'commutators of degree 1'. There is (up to sign) one non-vanishing[6] commutator of degree 2,

$$[A, B] := A\,B - B\,A,$$

and there are two non-vanishing commutators of degree 3,

$$[A, [A, B]] = A\,[A, B] - [A, B]\,A, \quad [[A, B], B] = [A, B]\,B - B\,[A, B],$$

and so on; see Section 2.2 for commutators of higher degrees.

2 Taylor Expansion of the Local Error

2.1 Representation of Taylor Coefficients

Consider the Taylor expansion, about $h = 0$, of the local error operator $\mathcal{L}(h)$ of a consistent one-step method (satisfying the basic consistency condition $\mathcal{L}(0) = 0$),

$$\mathcal{L}(h) = \sum_{q=1}^{p} \frac{h^q}{q!}\,\frac{\mathrm{d}^q}{\mathrm{d}h^q}\,\mathcal{L}(h)\,\Big|_{h=0} + \mathcal{O}(h^{p+1}). \tag{5}$$

The method is of asymptotic order p iff $\mathcal{L}(h) = \mathcal{O}(h^{p+1})$ for $h \to 0$; thus the conditions for order $\geq p$ are given by

$$\frac{\mathrm{d}}{\mathrm{d}h}\,\mathcal{L}(h)\,\Big|_{h=0} = \ \ldots\ = \frac{\mathrm{d}^p}{\mathrm{d}h^p}\,\mathcal{L}(h)\,\Big|_{h=0} = 0. \tag{6}$$

The formulas in (6) need to be presented in a more explicit form, involving the operators A and B. For a splitting method (4), a calculation based on the Leibniz formula for higher derivatives shows[7] (see [2])

$$\frac{\mathrm{d}^q}{\mathrm{d}h^q}\,\mathcal{L}(h)\,\Big|_{h=0} = \sum_{|\boldsymbol{k}|=q} \binom{q}{\boldsymbol{k}} \prod_{j=s\ldots1} \sum_{\ell=0}^{k_j} \binom{k_j}{\ell} B_j^{\ell}\,A_j^{k_j-\ell} \ - \ (A+B)^q, \tag{7}$$

with $\boldsymbol{k} = (k_1, \ldots, k_s) \in \mathbb{N}_0^s$.

Representation of (7) *in Maple.* The non-commuting operators A and B are represented by symbolic variables `A` and `B`, which can be declared to be non-commutative making use of the corresponding feature implemented in the package `Physics`. Now it is straightforward to generate the sum (7), with unspecified coefficients a_j, b_j, using standard combinatorial tools; for details see Section 3.

[6] 'Non-vanishing' means non-vanishing in general (generic case, with no special assumptions on A and B).

[7] If A and B commute, i.e., $AB = BA$, then all these expressions vanish.

2.2 The Leading Term of the Local Error Expansion

Formally, the multinomial sums in the expressions (7) are multivariate homogeneous polynomials of total degree q in the variables a_j, b_j, $j = 1 \dots s$, and the coefficients of these polynomials are power products of total degree q composed of powers of the non-commutative symbols A and B.

Example 2 ([2]*).* For $s = 2$ we obtain

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}h}\,\mathcal{L}(h)\,\Big|_{h=0} &= (a_1 + a_2)\,A + (b_1 + b_2)\,B \;-\; (A + B),\\
\frac{\mathrm{d}^2}{\mathrm{d}h^2}\,\mathcal{L}(h)\,\Big|_{h=0} &= ((a_1 + a_2)^2)\,A^2 + (2\,a_2\,b_1)\,A\,B\\
&\quad + (2\,a_1\,b_1 + 2\,a_1\,b_2 + 2\,a_2\,b_2)\,B\,A + ((b_1 + b_2)^2)\,B^2\\
&\quad - (A^2 + A\,B + B\,A + B^2).
\end{aligned}$$

The consistency condition for order $p \geq 1$ reads $\frac{\mathrm{d}}{\mathrm{d}h}\,\mathcal{L}(h)\,\big|_{h=0} = 0$, which is equivalent to $a_1 + a_2 = 1$ and $b_1 + b_2 = 1$.

At first sight, for order $p \geq 2$ we need 4, or (at second sight) 2 additional equations to be satisfied, such that $\frac{\mathrm{d}^2}{\mathrm{d}h^2}\,\mathcal{L}(h)\,\big|_{h=0} = 0$. However, assuming that the conditions for order $p \geq 1$ are satisfied, the second derivative $\frac{\mathrm{d}^2}{\mathrm{d}h^2}\,\mathcal{L}(h)\,\big|_{h=0}$ simplifies to the commutator expression

$$\frac{\mathrm{d}^2}{\mathrm{d}h^2}\,\mathcal{L}(h)\,\Big|_{h=0} = (2\,a_2\,b_1 - 1)\,[A, B],$$

giving the single additional condition $2\,a_2\,b_1 = 1$ for order $p \geq 2$. Assuming now that a_1, a_2 and b_1, b_2 are chosen such that all conditions for $p \geq 2$ are satisfied, the third derivative $\frac{\mathrm{d}^3}{\mathrm{d}h^3}\,\mathcal{L}(h)\,\big|_{h=0}$, which now represents the leading term of the local error, simplifies to a linear combination of the commutators $[A, [A, B]]$ and $[[A, B], B]$, of degree 3, namely

$$\frac{\mathrm{d}^3}{\mathrm{d}h^3}\,\mathcal{L}(h)\,\Big|_{h=0} = (3\,a_2^2\,b_1 - 1)\,[A, [A, B]] + (3\,a_2\,b_1^2 - 1)\,[[A, B], B]. \qquad \square$$

Remark 1. The classical second-order Strang splitting method corresponds to the choice $a_1 = \frac{1}{2}$, $b_1 = 1$, $a_2 = \frac{1}{2}$, $b_2 = 0$, or $a_1 = 0$, $b_1 = \frac{1}{2}$, $a_2 = 1$, $b_2 = \frac{1}{2}$.

The observation from this simple example generalizes as follows:

Proposition 1. *The leading term* $\frac{\mathrm{d}^{p+1}}{\mathrm{d}h^{p+1}}\,\mathcal{L}(h)\,\big|_{h=0}$ *of the Taylor expansion of the local error* $\mathcal{L}(h)$ *of a splitting method of order* p *is a Lie element, i.e., it is a linear combination of commutators of degree* $p + 1$*.*

Proof. See [1, 8]. $\square$

Example 3. Assume that the coefficients a_j, b_j, $j = 1 \dots s$ have been found such that the associated splitting scheme is of order $p \geq 3$ (this necessitates $s \geq 3$). This means that

$$\frac{\mathrm{d}}{\mathrm{d}h}\,\mathcal{L}(h)\,\Big|_{h=0} = \frac{\mathrm{d}^2}{\mathrm{d}h^2}\,\mathcal{L}(h)\,\Big|_{h=0} = \frac{\mathrm{d}^3}{\mathrm{d}h^3}\,\mathcal{L}(h)\,\Big|_{h=0} = 0,$$

and from Proposition 1 we know that

$$\frac{\mathrm{d}^4}{\mathrm{d}h^4}\,\mathcal{L}(h)\,\Big|_{h=0} = \gamma_1\,[A,[A,[A,B]]] + \gamma_2\,[A,[[A,B],B]] + \gamma_3\,[[[[A,B],B],B]$$

holds, with certain coefficients γ_k depending on the a_j and b_j. Here we have made use of the fact that there are three independent commutators of degree 4 in A and B. □

Targeting for higher-order methods one needs to know a *basis of commutators* up to a certain degree. The answer to this question is known, and a full set of independent commutators of degree q can be represented by a set of words of length q over the alphabet $\{A, B\}$. A prominent example is the set of *Lyndon-Shirshov words* (see [6]) displayed in Table 1. A combinatorial algorithm due to Duval [7] can be used to generate this table.

Here, for instance, the word `AABBB` represents the commutator

$$[A,[[[A,B],B],B]] = \\ A^2B^3 - 3ABAB^2 + 3AB^2AB - 2AB^3A + 3BAB^2A - 3B^2ABA + B^3A^2,$$

with leading power product $A^2B^3 = AABBB$ (w.r.t. lexicographical order).

q	L_q	Lyndon-Shirshov words over the alphabet {`A`, `B`}
1	2	`A`, `B`
2	1	`AB`
3	2	`AAB`, `ABB`
4	3	`AAAB`, `AABB`, `ABBB`
5	6	`AAAAB`, `AAABB`, `AABAB`, `AABBB`, `ABABB`, `ABBBB`
6	9	`AAAAAB`, `AAAABB`, `AAABAB`, `AAABBB`, `AABABB`, `AABBAB`, `AABBBB`, `ABABBB`, `ABBBBB`
7	18	...
8	30	...
9	56	...
10	99	...
⋮	⋮	⋱

Table 1. L_q is the number of words of length q.

2.3 The Algorithm: Implicit Recursive Elimination

On the basis of Proposition 1, and with a table of Lyndon-Shirshov words available, we can build up a set of conditions for order $\geq p$ for a splitting method with s stages in the following way (recall the notation $A_j := a_j\, A$, $B_j = b_j\, B$). This procedure corresponds to [1, Algorithm 2]:

For $q = 1 \ldots p$:

- *Generate the symbolic expressions* (7) *in the indeterminate coefficients a_j, b_j and the non-commutative variables* `A` *and* `B`.
- *Extract the coefficients of the power products (of degree q) represented by all Lyndon-Shirshov words of length q, resulting in a set of L_q polynomials $P_{q,k}(a_j, b_j)$ of degree q in the coefficients a_j and b_j.*

The resulting set of $\sum_{q=1}^{p} L_q$ multivariate polynomial equations

$$P_{q,k}(a_j, b_j) = 0, \quad k = 1 \ldots L_q, \quad q = 1 \ldots p \tag{8}$$

represents the desired conditions for order p.

We call this procedure *implicit recursive elimination,* because the equations generated in this way are correct in an 'a posteriori' sense (cf. Example 2):

- For $q = 1$, the basic consistency equations

$$\begin{aligned} P_{1,1}(a_j, b_j) &= a_1 + \ldots + a_s - 1 = 0, \\ P_{1,2}(a_j, b_j) &= b_1 + \ldots + b_s - 1 = 0, \end{aligned} \tag{9a}$$

  are obtained.
- *Assume* that (9a) is satisfied. Then, due to Proposition 1, the additional (quadratic) equation (note that $L_2 = 1$)

$$P_{2,1}(a_j, b_j) = 0, \tag{9b}$$

  represents one additional condition for a scheme of order $p = 2$.
- *Assume* that (9a) and (9b) are satisfied. Then, due to Proposition 1, the additional (cubic) equations (note that $L_3 = 2$)

$$P_{3,1}(a_j, b_j) = P_{3,2}(a_j, b_j) = 0, \tag{9c}$$

  represent two additional conditions for a scheme of order $p = 3$.
- The process is continued in the same manner.

If we (later) *have found* a solution $S = \{a_j, b_j,\, j = 1 \ldots s\}$ of the resulting system

$$(8) = \{(9a), (9b), (9c), \ldots\}$$

of multivariate polynomial equations, this means that

- S satisfies (9a) $\Rightarrow$
  S represents a solution of order $q = 1$ at least;

- S satisfies (9a) and (9b) $\Rightarrow$
  S represents a solution of order $q = 2$ at least;
- S satisfies (9a), (9b), and (9c) $\Rightarrow$
  S represents a solution of order $q = 3$ at least;

and so on. By induction we conclude that the whole procedure indeed results in a solution S representing a method of the desired order p. See [2] for a more detailed exposition of this argument.

Remark 2. In addition, it makes sense to generate the additional conditions for order $p + 1$. Even if we do not solve for these conditions, they represent the leading term of the local error, and this can be used to search for optimized solutions for order p, where the coefficients in $\frac{\mathrm{d}^{p+1}}{\mathrm{d}h^{p+1}} \mathcal{L}(h) \big|_{h=0}$ become minimal in size.

3 A Parallel Implementation

In our Maple code, a table of Lyndon-Shirshov words up to a fixed length (corresponding to the maximal order aimed for; see Table 1) is included as static data. The procedure `Order_conditions` displayed below generates a set of order conditions using the algorithm described in Section 2.3.

- First of all, we activate the package `Physics` and declare the symbols `A` and `B` as non-commutative.
- For organizing the multinomial expansion according to (7) we use standard functions from the packages `combinat` and `combstruct`.
- The number of terms during each stage rapidly increases as more stages are to be computed. Therefore we have implemented a parallel version based on the package `Grid`. Parallelization is taken into account as follows:
  - On a multi-core processor, all threads execute the same code. Each thread identifies itself via a call to `MyNode()`, and this is used to control execution. Communication between the threads is realized via message passing.
  - Thread 0 is the master thread controlling the overall execution.
  - For $q = 1 \dots p$:
    - Each of the working threads generates symbolic expressions of the form (recall $A_j = a_j\, A$, $B_j = b_j\, B$)

$$\Pi_{\boldsymbol{k}} := \binom{q}{\boldsymbol{k}} \prod_{j=s\dots 1} \sum_{\ell=0}^{k_j} \binom{k_j}{\ell} B_j^{\ell}\, A_j^{k_j-\ell}, \quad \boldsymbol{k} \in \mathbb{N}_0^s,$$

      appearing in the sum (7). Here the work is equidistributed over the threads, i.e., each of them generates a subset of $\{\Pi_{\boldsymbol{k}},\ \boldsymbol{k} \in \mathbb{N}_0^s\}$ in parallel.
    - For each of these expressions $\Pi_{\boldsymbol{k}}$, the coefficients of all Lyndon-Shirshov monomials of degree q are computed, and the according subsets of coefficients are summed up in parallel.

  - Finally, the master thread 0 sums up the results received from all the working threads. This results in the set of multivariate polynomials representing the order conditions at level q.

– The Maple code displayed below is, to some extent, to be read as pseudo-code. For simplicity of presentation we have ignored some technicalities, e.g., concerning the proper indexing of combinatorial tupels, etc. The original, working code is available from the authors.

```
> with(combinat)
> with(combstruct)
> with(Grid)
> with(Physics)
> Setup(noncommutativeprefix={A,B})

> Order_conditions := proc()
  global p,s,OC,    # I/O parameters via global variables
         Lyndon     # assume that table of Lyndon monomials is available
  this_thread := MyNode()   # each thread identifies itself
  max_threads := NumNodes() # number of available threads
  for j from 1 to s do
      A_j[j] := a[j]*A
      B_j[j] := b[j]*B
      term[-1][j] := 1
  end do
  OC=[0$p]
  for q from 1 to p do
    if this_thread>0 then  # working threads start computing
                           # master thread 0 is waiting
       Mn := allstructs(Composition(q+2),size=2)
       for j from 1 to s do
           term[q-1][j] := 0
           for mn from 1 to nops(Mn) do
               term[q-1][j] :=
                 term[q-1][j] +
                   multinomial(q,Mn[mn])*B_j[j]^Mn[mn][2]*A_j[j]^Mn[mn][1]
           end do
       end do
       k := iterstructs(Composition(q+s),size=s)
       OC_q_this_thread := [0$nops(Lyndon[q])]
       while not finished(k) do # generate expansion (7) term by term
         Ms := nextstruct(k)
         if get_active(this_thread) then # get_active:
                                         #   auxiliary Boolean function
                                         #   for equidistributing workload
            Pi_k := 1
            for j from s to 1 by -1 do
                Pi_k := Pi_k*term[Ms[j]-1][j]
            end do
            Pi_k := multinomial(q,Ms)*expand(Pi_k)
            OC_q_this_thread := # compare coefficients of Lyndon monomials
```

```
              OC_q_this_thread +
                [seq(coeff(Pi_k,Lyndon[q][l]),l=1..nops(Lyndon[q]))]
          end if
        end do
        Send(0,OC_q_this_thread) # send partial sum to master thread
    else # master thread 0 receives and sums up
         #   partial results from working threads
        OC[q] := [(-1)$nops(Lyndon[q])] # initialize sum
        for i_thread from 1 to max_threads-1 do
            OC[q] := OC[q] + Receive(i_thread)
        end do
    end if
end do
end proc

> # Example:
> p := 4
> s := 4
> Launch(Order_conditions,imports=["p","s"],exports=["OC"])  # run

> OC[1]
  [a[1]+a[2]+a[3]+a[4]-1,
   b[1]+b[2]+b[3]+b[4]-1]

> OC[2]
  [2*a[2]*b[1]+2*a[3]*b[1]+2*a[3]*b[2]
   +2*a[4]*b[1]+2*a[4]*b[2]+2*a[4]*b[3]-1]

> OC[3]
  [3*a[2]^2*b[1]+6*a[2]*a[3]*b[1]+6*a[2]*a[4]*b[1]
   +3*a[3]^2*b[1]+3*a[3]^2*b[2]+6*a[3]*a[4]*b[1]+6*a[3]*a[4]*b[2]
    +3*a[4]^2*b[1]+3*a[4]^2*b[2]+3*a[4]^2*b[3]-1,
   3*a[2]*b[1]^2+3*a[3]*b[1]^2+6*a[3]*b[1]*b[2]
    +3*a[3]*b[2]^2+3*a[4]*b[1]^2+6*a[4]*b[1]*b[2]+6*a[4]*b[1]*b[3]
     +3*a[4]*b[2]^2+6*a[4]*b[2]*b[3]+3*a[4]*b[3]^2-1]

> OC[4]
  [4*a[2]^3*b[1]+12*a[2]^2*a[3]*b[1]+12*a[2]^2*a[4]*b[1]
   +12*a[2]*a[3]^2*b[1]+24*a[2]*a[3]*a[4]*b[1]+12*a[2]*a[4]^2*b[1]
    +4*a[3]^3*b[1]+4*a[3]^3*b[2]+12*a[3]^2*a[4]*b[1]
     +12*a[3]^2*a[4]*b[2]+12*a[3]*a[4]^2*b[1]+12*a[3]*a[4]^2*b[2]
      +4*a[4]^3*b[1]+4*a[4]^3*b[2]+4*a[4]^3*b[3]-1,
   6*a[2]^2*b[1]^2+12*a[2]*a[3]*b[1]^2+12*a[2]*a[4]*b[1]^2
    +6*a[3]^2*b[1]^2+12*a[3]^2*b[1]*b[2]+6*a[3]^2*b[2]^2
     +12*a[3]*a[4]*b[1]^2+24*a[3]*a[4]*b[1]*b[2]+12*a[3]*a[4]*b[2]^2
      +6*a[4]^2*b[1]^2+12*a[4]^2*b[1]*b[2]+12*a[4]^2*b[1]*b[3]
       +6*a[4]^2*b[2]^2+12*a[4]^2*b[2]*b[3]+6*a[4]^2*b[3]^2-1,
   4*a[2]*b[1]^3+4*a[3]*b[1]^3+12*a[3]*b[1]^2*b[2]
    +12*a[3]*b[1]*b[2]^2+4*a[3]*b[2]^3+4*a[4]*b[1]^3
     +12*a[4]*b[1]^2*b[2]+12*a[4]*b[1]^2*b[3]+12*a[4]*b[1]*b[2]^2
```

```
+24*a[4]*b[1]*b[2]*b[3]+12*a[4]*b[1]*b[3]^2+4*a[4]*b[2]^3
 +12*a[4]*b[2]^2*b[3]+12*a[4]*b[2]*b[3]^2+4*a[4]*b[3]^3-1]
```

For practical use some further tools have been developed, e.g. for generating tables of polynomial coefficients for further use, e.g., by numerical software other than Maple. This latter job can also be parallelized.

3.1 Special Cases

Some special cases are of interest:

- *Symmetric schemes* are characterized by the property $\mathcal{S}(-h, \mathcal{S}(h, u)) = u$. Here, either $a_1 = 0$ or $b_s = 0$, and the remaining coefficient sets (a_j) and (b_j) are palindromic. Symmetric schemes have an even order p, and the order conditions for even orders need not be included; see [8]. Thus, we use a special ansatz and generate a reduced set of equations.
- *Palindromic schemes* were introduced in [2] and characterized by the property $\mathcal{S}(-h, \check{\mathcal{S}}(h, u)) = u$, where $\check{\mathcal{S}}$ denotes the scheme $\mathcal{S}$ with the role of A and B interchanged. In this case, the full coefficient set

  $$(a_1, b_1, \ldots, a_s, b_s)$$

  is palindromic. As for symmetric schemes, this means that a special ansatz is used, and again it is sufficient to generate a reduced set of equations, see [2].

Apart from these modifications, the basic algorithm remains unchanged.

4 Modifications and Extensions

4.1 Splitting into more than two Operators

Our algorithm directly generalizes to the case of splitting into more than two operators. Consider evolution equations where the right-hand side splits into three parts,

$$\partial_t u(t) = F(u(t)) = A(u(t)) + B(u(t)) + C(u(t)), \tag{10}$$

and associated splitting schemes,

$$\mathcal{S}(h, u) = \mathcal{S}_s(h, \mathcal{S}_{s-1}(h, \ldots, \mathcal{S}_1(h, u))) \approx \Phi_F(h, u), \tag{11a}$$

with

$$\mathcal{S}_j(h, v) = \Phi_C(c_j\, h, \Phi_B(b_j\, h, \Phi_A(a_j\, h, v))), \tag{11b}$$

see [4]. Here the linear representation (7) generalizes as follows, with $A_j = a_j\, A$, $B_j = b_j\, B$, $C_j = c_j\, C$, and $\boldsymbol{k} = (k_1, \ldots, k_s) \in \mathbb{N}_0^s$, $\boldsymbol{\ell} = (\ell_A, \ell_B, \ell_C) \in \mathbb{N}_0^3$:

$$\frac{\mathrm{d}^q}{\mathrm{d}h^q}\, \mathcal{L}(h)\,\Big|_{h=0} = \sum_{|\boldsymbol{k}|=q} \binom{q}{\boldsymbol{k}} \prod_{j=s\ldots1} \sum_{|\boldsymbol{\ell}|=k_j} \binom{k_j}{\boldsymbol{\ell}} C_j^{\ell_C}\, B_j^{\ell_B}\, A_j^{\ell_A} \;-\; (A+B+C)^q. \tag{12}$$

On the basis of these identities, the algorithm from Section 2.3 generalizes in a straightforward way. The Lyndon basis representing independent commutators now corresponds to Lyndon words over the alphabet $\{\mathtt{A}, \mathtt{B}, \mathtt{C}\}$, see [2]. Concerning special cases (symmetries etc.) and parallelization, similar considerations as before apply.

4.2 Pairs of Splitting Schemes

For the purpose of adaptive time-splitting algorithms, the construction of (optimized) pairs of schemes of orders $(p, p+1)$ is favorable. Generating a respective set of order conditions can also be accomplished by a modification of our code; the difference lies in the fact that some coefficients are chosen a priori (corresponding to a given method of order $p+1$), but apart from this the generation of order conditions for an associated scheme of order p works analogously as before. Finding optimal schemes is then accomplished by tracing a large set of possible solutions; see [2].

5 Computational Performance; Conclusions

The following computations were performed on a node consisting of two processors of type AMD Opteron 6132 HR (2.2 GHz) with 8 cores. This means that together with a master thread up to 15 working threads can be used. An ample memory of 32 GB is available.

Beginning with order $p = 6$, the computational effort becomes significant (and strongly increases with higher orders). We consider two different parameter settings (without assuming any symmetries for the setup of order conditions):

(i) AB – splitting, 10-stage scheme ($s = 10$), desired order $p = 6$,
(ii) ABC – splitting, 15-stage scheme ($s = 15$), desired order $p = 6$.

Timing data are specified in the format [d]:[hh]:mm:ss.

For case (i) we compare the performance for the fully parallelized version including 15 working threads with a restricted, essentially sequential version where only 1 working thread is used.

(i)
- 15 active working threads:
  wall clock time = 00:45,
  total CPU time = 09:48.
  This amounts to an overall processor utilization of about 85 %.
- 1 active working thread:
  wall clock time = 07:46,
  total CPU time = 07:58.

We approximately observe the expected linear speedup of running (wall clock) time with the number of threads used. The slightly increased cost (in terms of total CPU time) of the fully parallel version is to be attributed to communication between the threads.

(ii) • 15 active working threads:
wall clock time = 01:46:03,
total CPU time = 1:01:29:47.
This amounts to an efficient overall processor utilization of about 90 %.
For this case we have not performed a run with a single working thread.

For the ABC case the absolute timing data are significantly larger due to the fact that the number of terms in the Taylor expansion of the local error grows much faster.[8]

Especially in this latter case, the poor computational performance of a general-purpose symbolic system like Maple (including the `Physics` package) becomes evident. Here, parallelization is essential to reduce wall-clock times as much as possible. The algorithm presented here could also be implemented in a 'slimmer' language as for instance C or Julia, but of course at the expense of implementing many auxiliary components like various combinatorial functions and, in particular, handling of expressions involving non-commutative variables. In this sense, our implementation is a pragmatic one: Make use of a readily available software package and gain performance via parallelization, a strategy which may be relevant also for other kinds of symbolic codes.

Acknowledgements. This work was supported by the Austrian Science Fund (FWF) under grant P24157-N13, and by the Vienna Science and Technology Fund (WWTF) under grant MA-14-002. Computational results based on the ideas in this work have been achieved in part using the Vienna Scientific Cluster (VSC).

References

1. Auzinger, W., Herfort, W.: Local error structures and order conditions in terms of Lie elements for exponential splitting schemes. Opuscula Math. 34, 243–255 (2014)
2. Auzinger, W., Hofstätter, H., Ketcheson, D., Koch, O.: Practical splitting methods for the adaptive integration of nonlinear evolution equations. Part I: Construction of optimized schemes and pairs of schemes. to appear in BIT Numer. Math.
3. Auzinger, W., Koch, O.: Coefficients of various splitting methods. At `www.asc.tuwien.ac.at/~winfried/splitting/`.
4. Auzinger, W., Koch, O., Thalhammer, M.: Defect-based local error estimators for high-order splitting methods involving three linear operators. Numer. Algorithms 70, 61–91 (2015)
5. Blanes S., Casas, F., Farrés, A., Laskar, J., Makazaga, J., A. Murua, A.: New families of symplectic splitting methods for numerical integration in dynamical astronomy. Appl. Numer. Math. 68, 58–72 (2013)
6. Bokut, L., Sbitneva, L., Shestakov, I.: Lyndon-Shirshov words, Gröbner-Shirshov bases, and free Lie algebras. In 'Non-Associative Algebra and Its Applications', Chapter 3. Chapman & Hall / CRC, Boca Raton, Fl. (2006)

[8] There are special cases of practical interest where this growth is much more moderate; we do not discuss such details here.

7. Duval, J.P.: Géneration d'une section des classes de conjugaison et arbre des mots de Lyndon de longueur bornée. Theoret. Comput. Sci. 60, 255–283 (1988)
8. Hairer, E., Lubich, C., Wanner, G.: *Geometric Numerical Integration*. 2nd ed., Springer-Verlag, Berlin–Heidelberg–New York, 2006.
9. Ketcheson, D., MacDonald, C., Ruuth, S.: Spatially partitioned embedded Runge-Kutta methods. SIAM J. Numer. Anal. 51, 2887–2910 (2013)