UNIVERSITY OF CAMBRIDGE

Numerical Analysis Reports

Lie-group Methods for Isospectral Flows

Antonella Zanna

DAMTP 1997/NA02
March 1997

Department of Applied Mathematics and Theoretical Physics
Silver Street
Cambridge
England CB3 9EW

1 Introduction

We address ourself to the problem of evaluating the numerical solution of a class of ordinary differential equations on matrices, known under the name of *isospectral flows.* These flows arise in a variety of mathematical applications, ranging from linear algebra to solid state physics and biology.

Until few years ago, it was a computational observation that classical method for *Ordinary Differential Equations* (ODEs) produced a numerical output that was not isospectral, thus failing to retain the main qualitative feature of the assigned problem. However, in 1995, Calvo, Iserles and Zanna established a theory according to which classical methods for ODEs *could not* be isospectral, and they presented other methods *(modified Gauss–Legendre Runge–Kutta methods)* to construct isospectral solutions.

In this paper we propose to solve isospectral flows using an approach based on Lie-group numerical methods. Although this idea might appear contrived at a first glance, it is easy to convince ourselves that it is the natural way to analyse mathematically and solve numerically this class of problems.

The problem of integrating ODEs on Lie groups started as early as in the 1880's, with the work of Sophus Lie. The method of iterated commutators, that will be introduced in section § 3 is in effect intimately related to the *method of reduction* for linear ODEs on Lie group proposed by Lie [28].

As far as we are aware, the first *numerical* (opposed to *analytical)* methods to integrate ODEs on Lie groups appeared only in 1993 in a paper of Crouch and Grossman. Briefly later, Munthe-Kaas developed Lie-group methods of a Runge–Kutta type, while Owren and Marthinsen developed the theory of the methods proposed by Crouch and Grossman in a more complete manner. In 1996, Zanna proposed the method of iterated commutators, based on numerical schemes for ODEs introduced by Iserles in 1984 [12] and subsequently Iserles and Nørsett generalized and rendered numerically efficient a numerical approach *(the Magnus series)* introduced in 1954 by Magnus. At present, many issues related to this class of methods are being intensely analysed.

Some mathematicians have been skeptical with regards to the practical use of Lie-group methods. In a sense, it is counterintuitive to believe that Lie-group methods can compete with classical explicit schemes or with projection methods. However, numerical experiments indicate clearly that Lie-group methods can be cheaper than classical explicit schemes: Lie-group equations possess a lot of features and symmetries that, intelligently exploited, lead to superior numerical methods.

With regard to Lie-group methods for isospectral flows, it is fair to mention that in 1994 Helmke, Mahony and Moore introduced an isospectral scheme (which they called a *Lie method)* for the solution of *double-bracket* flows [17]. However, their method has order one only, which means that it is not very competitive if we want a good accuracy.

The approach that we present in this paper is much more general and allows us to introduce, analyse and implement Lie-group methods of arbitrarily high order.

2 From isospectral flows to Lie-group equations

2.1 Isospectrality and conserved integrals

Isospectral flows are characterized by the matrix differential equation

$$L' = [B, L], \qquad L(0) = L_0, \tag{2.1}$$

where $L, B \in \mathbb{R}^{d\times d}$ and L_0 is a given $d \times d$ initial matrix. The matrix function $B \equiv B(t, L)$ depends on L and, possibly, on the time t. The square brackets denote the commutator *Lie bracket* on matrices, $[B, L] = BL - LB$.

Each flow is characterized by the matrix B, which is usually skew-symmetric, while L is generally symmetric. For instance, if L_- and L_+ denote the lower and upper triangular part of the matrix L, when $B(L) = L_- - L_+$, then we have the *Toda flow,* associated with the Toda lattice equations governing the motion of the particles on a one-dimensional lattice under exponential nearest-neighbour interaction [24]. When $B(L) = f(L)_- - f(L)_+$, f being an analytic function of the spectrum of L, then we have the *QR flow* (see [6] or [27] for a list of references). When $B(L) = [N, L]$ for a fixed diagonal matrix N, then we have the *double-bracket flow* [2], and so on.

It is well known that the solution of (2.1) is *isospectral,* namely the the spectrum of the integral curve $L(t)$ of (2.1) does not change with time [24]. The d eigenvalues of L are related to the d *conserved integrals* that had been discovered by Hénon and Flaschka for the Toda lattice equations. The integrals are in involution, therefore Hénon and Flaschka concluded that the Toda lattice equations, and consequently also the other isospectral flows, are integrable systems [24].

The conserved integrals can be introduced by considering the d-degree characteristic polynomial associated with the matrix $L(t)$,

$$p(\lambda) = \det(\lambda I - L) = \lambda^d - p_{d-1}\lambda^{d-1} + \cdots + (-1)^d p_0,$$

whose zeros are the eigenvalues λ_i, $i = 1, \ldots, d$, of L. Since the eigenvalues do not depend on time, the coefficients

$$\begin{aligned} p_{d-1} &= \lambda_1 + \cdots + \lambda_d, \\ p_{d-2} &= \sum_{i=1}^{d} \sum_{j=i+1}^{d} \lambda_i \lambda_j, \\ &\vdots \\ p_0 &= \lambda_1 \lambda_2 \cdots \lambda_d. \end{aligned}$$

(symmetric polynomials) are constant and constitute the d integrals associated with the flow. Obviously, every function of the λ_is is an integral of the flow. However, in order to characterize univocally the eigenvalues, one needs to indicate d independent conditions, that can be chosen as

$$\mathrm{tr}(L^r) = \sum_{i=1}^{d} \lambda_i^r, \qquad r = 1, \ldots, d, \tag{2.2}$$

an alternative to the symmetric polynomials [22, 24]. Such conditions are at the basis of the Calvo–Iserles–Zanna analysis of numerical methods and their retention of isospectrality [6]. They proved that classical numerical methods, such as multistep and Runge–Kutta schemes cannot be isospectral for $d \geq 3$, in the sense that, given a classical numerical method, it is always possible to construct an isospectral flow for which it fails to retain isospectrality. In the 2×2 case, however, Gauss–Legendre Runge–Kutta methods can be isospectral, since in this case isospectrality is equivalent to conservation of quadratic invariants and is satisfied by all Runge–Kutta methods whose coefficients obey the condition [6, 9, 11]

$$M = O, \qquad m_{i,j} = b_i a_{i,j} + b_j a_{j,i} - b_i b_j,$$

where the $a_{i,j}$s and the b_is are the coefficients in the Butcher tableau

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

of the method [5].

2.2 From the Lax's formalism to equations of Lie type

An equation of evolution for an unknown function y is said to be in the *Lax form* if it is of the type

$$\frac{\mathrm{d}y}{\mathrm{d}t} = \mathcal{B}y - y\mathcal{B},$$

where $\mathcal{B}$ is in general an operator. Therefore isospectral flows as in (2.1) are in the Lax form, the operator being the typically skew-symmetric matrix B.

However, we have seen that classical methods for ODEs fail in retaining isospectrality for isospectral flows in the Lax form (2.1) hence, for isospectral numerical approximations to the problem, we shall follow a different approach.

We will follow the same construction as Flaschka to prove the integrability of the Toda flows. Flaschka associated to (2.1) the differential equation

$$u' = B(L)u, \qquad u(0) = I. \tag{2.3}$$

As a consequence of the skew-symmetry of B, Flaschka concluded that the solution of (2.3) is an *orthogonal flow* (i.e. $uu^T = u^T u = I$) and that the solution of (2.1) can be written as

$$L(t) = u(t)L_0 u^T(t), \tag{2.4}$$

as it can be verified by differentiating $u^T(t)L(t)u(t)$ with respect to the variable t and taking into account that $u(0) = I$. Hence $L(t)$ is orthogonally similar to the initial condition L_0.

It turns out that the procedure of introducing the orthogonal flow (2.4) is not just useful for a theoretical analysis of isospectrality, but it is also the right equation to solve in place of (2.1) to obtain isospectral numerical approximations. By introducing the orthogonal flow (2.3) together with the transformation (2.4) we reduce the problem of conserving d

polynomial integrals to the *retention of orthogonality,* consisting essentially of quadratic conservation laws in the entries of the matrix u.

As we have already mentioned, Gauss–Legendre Runge–Kutta methods (GLRK), and in general all Runge–Kutta methods with $M = O$, retain orthogonality. However, Higham observed that RK methods with $M = O$ are not orthogonal when the matrix B, evaluated at the intermediate stages, is not skew-symmetric [11]. Therefore, if we solve the differential equation (2.3)

$$u' = B(uL_0u^T)u, \qquad u(0) = I, \quad B(A) = A_- - A_+,$$

with a GLRK scheme, our numerical output for u will not be orthogonal since the internal stages $X_i = I + h\sum_{j=1}^{s} a_{i,j}B(X_jL_0X_j^T)X_j$ are not generally orthogonal, and hence the matrices $B(X_iL_0X_i^T)$ need not be skew-symmetric. Thus, extra attention should be paid to the numerical implementation of orthogonal methods (2.3) for isospectral flows.

Departing from these basic observations, Calvo, Iserles and Zanna [6] derived the isospectral *modified Gauss–Legendre Runge–Kutta methods* (MGLRK) for isospectral flows. Such schemes construct an *Hermite* interpolant $\tilde{B}$ to the function B by considering its values and the values of its derivatives at times t_n, t_{n+1}. The degree of the interpolant is chosen accordingly to the order of the method. See [6, 27] for more details. An example of a MGLRK of order four features in the appendix, in table 8.

The main disadvantage of the MGLRK schemes is that, being based on GLRK methods, they are implicit and require the solution of a nonlinear system of equations typically by some variant of fixed point iterations. Moreover, fixed point iterations might require some restriction on the stepsize for achieving convergence. Therefore, these methods can be costly, although the expense pays off since isospectrality guarantees that the error decays with time, while for classical methods, numerical error accumulated in each step and can not be eliminated in the course of the solution. This is displayed in the numerical experiments in section 4.

Consistently cheaper methods for isospectral flows can be obtained by taking into account that orthogonal matrices constitute a *Lie group* and using explicit *Lie-group methods* for the numerical solution of orthogonal flows.

2.3 Lie groups: A brief introduction

Lie groups are very special manifolds. In addition to a manifold structure, they are endowed with an internal binary operation, usually called *group multiplication,* which is compatible with the underlying topology. This means that if $a(t)$ and $b(t)$ are two smooth (infinitely many times differentiable) one-parameter families of elements of the Lie group G, then also $a(t)b(t)$ and $a(t)^{-1}$ are in G and they are smooth.

The Lie algebra $\mathfrak{g}$ is the tangent space to G at the identity. The tangent bundle of a Lie group is trivial, therefore the tangent space an every point of G can be identified with the set $\mathfrak{g}$.

Since in the present essay we refer exclusively to matrix Lie groups (for which the multiplication is the classical matrix multiplication), we adopt the following convention to distinguish between types (matrices in G and matrices in $\mathfrak{g}$) when talking about Lie-group equations: a lower letter $x, y, \ldots$, is used to denote a matrix in the Lie group, while an uppercase letter $A, B, \ldots$, is used to denote a matrix in the Lie algebra.

Ordinary differential equations on matrix Lie groups can be always written in the form

$$y' = A(t, y)y, \qquad y(0) = y_0 \in G, \tag{2.5}$$

where the function $A(t, y)$ lies in $\mathfrak{g}$. When A is independent of y then (2.5) is linear. Some recurring examples of Lie groups are:

- the *general linear group* $\mathrm{GL}(d, \mathbb{R})$ of invertible matrices,
- the *special orthogonal group* $\mathrm{SO}(d, \mathbb{R})$ of orthogonal matrices with the determinant equal to one,
- the *special linear group* $\mathrm{SL}(d, \mathbb{R})$ of invertible matrices with the determinant equal to one,

with the Lie algebras

- $\mathfrak{gl}(d, \mathbb{R})$, the set of $d \times d$ real matrices,
- $\mathfrak{so}(d, \mathbb{R})$, the set of $d \times d$ skew-symmetric matrices,
- $\mathfrak{sl}(d, \mathbb{R})$, the set of $d \times d$ matrices with zero trace,

respectively.

The algebra $\mathfrak{g}$ is endowed with an internal operation, called *commutator* operator or *Lie bracket*,

$$[\,\cdot\,, \cdot\,] : \mathfrak{g} \times \mathfrak{g} \to \mathfrak{g}. \tag{2.6}$$

Moreover, there exists a canonical map from the Lie algebra $\mathfrak{g}$ to the group G, the exponential,

$$\exp(\,\cdot\,) : \mathfrak{g} \to G, \qquad \exp A = \sum_{k=0}^{\infty} \frac{A^k}{k!}. \tag{2.7}$$

As we shall see in section 3, the exponential and the commutator are the basic tools to construct Lie-group numerical methods.

2.4 Isospectral flows as adjoint orbits of group actions

In this section we revisit the procedure followed by Flaschka, as presented in section 2.2, viewing it in a different light. To this end, we need to introduce first the concept of *group action*. For the material, we follow essentially definitions and notations in [4]. Likewise, we assume G to be connected, i.e. G is not union of two disjoint open sets.

Let G be a Lie group and M a manifold. A *left action* of G on M is an operation $\lambda : G \times M \to M$ such that

$$\begin{aligned} \lambda(I, m) &= m, \\ \lambda(g_1 g_2, m) &= \lambda(g_1, \lambda(g_2, m)), \qquad g_1, g_2 \in G, \quad m \in M, \end{aligned}$$

where I denotes the identity element in G. When there is no danger of confusion, we will denote the left action by means of a dot, that is

$$\lambda(g, m) = g \cdot m, \qquad g \in G, \quad m \in M.$$

Fixing an element $m_0 \in M$, we can define the *orbit* in M relative to m_0,

$$G_{m_0} = \{g \cdot m_0 \mid g \in G\}. \tag{2.8}$$

In particular, every orbit is sub-manifold of M. As an example, let G be any matrix sub-group of $\mathrm{GL}(d, \mathbb{R})$ and $M \equiv \mathbb{R}^d$. Given a matrix $g \in G$ and a vector $\boldsymbol{y} \in \mathbb{R}^d$, let

$$g \cdot \boldsymbol{y} = g\boldsymbol{y},$$

the standard matrix-vector product. It is easy to verify that this is a left action of G on $\mathbb{R}^d$. The orbit corresponding to a vector $\boldsymbol{y}_0 \in \mathbb{R}^d$ is simply

$$G_{y_0} = \{g\boldsymbol{y}_0 \mid g \in G\}.$$

We require the group action to be *transitive,* i.e. for every $m_1, m_2 \in M$, there exists a $g \in G$ such that $m_2 = g \cdot m_1$.

When a manifold is endowed by a transitive group action, then a fundamental theorem due to Sophus Lie (cf. [4], Prop. 3) states that differential equations on M can be determined entirely in terms of differential equations on the Lie group G and the left action.

Theorem 1 *Any ordinary differential equation on a manifold M endowed with a transitive left G-action, can be written in the form*

$$\gamma'(t) = \lambda_*(A(t))(\gamma(t)), \qquad \gamma(0) = m \in M, \tag{2.9}$$

where $A : \mathbb{R} \to \mathfrak{g}$ is a curve on the Lie algebra, $\mathfrak{X}(M)$ is the Lie algebra of vector fields on M, and $\lambda_* : \mathfrak{g} \to \mathfrak{X}(M)$ is the Lie algebra anti-homomorphism[1] *induced by the left action. Moreover, the solution of* (2.9) *is given by*

$$\gamma(t) = y(t) \cdot m,$$

where $y : \mathbb{R} \to G$ is the fundamental solution of the Lie-group equation

$$y'(t) = A(t)y(t), \quad y(0) = I. \tag{2.10}$$

Therefore, having integrators that solve differential equations on Lie groups means, at the same time, that we are able to construct integrators that automatically stay on M, for all those manifolds which are endowed with a transitive left G-action.

Going back to isospectral flows, let $G \equiv \mathrm{SO}(d)$, the special orthogonal group, and let M be the manifold $\mathbb{R}^{d \times d}$ of $d \times d$ real matrices. It is easy to verify that

$$u \cdot L = uLu^T \qquad u \in \mathrm{SO}(d), \quad L \in \mathbb{R}^{d \times d},$$

[1] I.e. for every $A, B \in \mathfrak{g}$, then $\lambda_*([A, B]) = -[\lambda_*(A), \lambda_*(B)]$, where the latter is a commutator of vector fields on M [1, 4]

is a left action, and the corresponding equation on the manifold M is the isospectral flow (2.1)

$$L' = [B(L), L], \qquad L(0) = L_0,$$

where $B(L)$ maps L into a skew-symmetric matrix. However, the action of G is not transitive on $\mathbb{R}^{d\times d}$, but it becomes transitive if we restrict to the orbit

$$G_{L_0} = \{uL_0u^T \mid u \in \mathrm{SO}(d)\}.$$

Thus, the knowledge of the fundamental solution $u(t)$ that solves the Lie-group equation corresponding to (2.10),

$$u' = B(L)u,$$

allows us to obtain the solution to (2.1).

For a more extensive survey on the relation between isospectral flows and adjoint orbits associated to Lie groups, we refer the reader to [3].

3 Numerical methods for Lie groups

3.1 The method of iterated commutators

The method of iterated commutators was introduced by Iserles in [12] to solve linear system with variable coefficients

$$\boldsymbol{y}' = A(t)\boldsymbol{y}, \qquad \boldsymbol{y}(0) = \boldsymbol{y}_0, \tag{3.1}$$

where $\boldsymbol{y}$ is a vector in $\mathbb{R}^d$ and $A(t)$ is an analytic $d \times d$ matrix function. Iserles showed that the solution $\boldsymbol{y}(t)$ of (3.1) can be written in the form

$$\boldsymbol{y}(t) = \lim_{m\to\infty} \mathrm{e}^{B_0(t)} \cdots \mathrm{e}^{B_m(t)}\boldsymbol{y}_0, \tag{3.2}$$

where the matrix functions $B_k(t)$ are obtained in an iterative manner. First of all, we let $A_0(t) = A(t)$ and evaluate

$$B_0(t) = \int_0^t A_0(\tau)\,\mathrm{d}\tau, \tag{3.3}$$

then, we construct the sequence of commutators

$$C_{0,0} = A_0(t), \qquad C_{0,i+1} = [C_{0,i}, B_0], \quad i \geq 0.$$

Next we form the matrix function

$$A_1(t) = \sum_{i=1}^{\infty} \frac{i}{(i+1)!} C_{0,i}(t)$$

and let

$$B_1(t) = \int_0^t A_1(\tau)\,\mathrm{d}\tau.$$

The procedure is hence repeated. Once that $A_k(t)$ and $B_k(t)$ are known, we let

$$A_{k+1}(t) = \sum_{i=1}^{\infty} \frac{i}{(i+1)!} C_{k,i}(t), \tag{3.4}$$

where

$$C_{k,0} = A_k(t), \qquad C_{k,i+1} = [C_{k,i}, B_k], \quad i \geq 0. \tag{3.5}$$

and finally

$$B_{k+1}(t) = \int_0^t A_{k+1}(\tau)\, \mathrm{d}\tau. \tag{3.6}$$

In passing, we mention that the *iterated commutators* in (3.5) can be conveniently written in a classical Lie-algebra notation as

$$C_{k,i} = \mathrm{ad}^i_{A_k}(B_k), \tag{3.7}$$

where $\mathrm{ad}_u(\cdot) : \mathfrak{g} \to \mathfrak{g}$, $u \in \mathfrak{g}$ is the *adjoint* operator, whose powers are evaluated as

$$\mathrm{ad}^0_u(v) = u, \qquad \mathrm{ad}^{i+1}_u(v) = [\mathrm{ad}^i_u(v), v], \quad i = 0, 1 \ldots, \tag{3.8}$$

where $u, v \in \mathfrak{g}$.

Iserles [12] proved that if we truncate (3.2) to an integer $M \geq 0$, then the order of the approximation is $p = 2^{M+2} - 2$ and will remain the same provided that we replace the integrals in (3.6) and the infinite series in (3.4) by finite sums and quadrature formulae of appropriate order.

Although the method of iterated commutators might appear a strange and unconventional way to solve an arbitrary linear differential equation, Zanna [26] put in evidence that the algorithm proposed by Iserles is based on *elementary operations* on Lie groups and Lie algebras: the adjoint map, ad_u, maps $\mathfrak{g}$ onto $\mathfrak{g}$, integrals of matrices in $\mathfrak{g}$ are elements of $\mathfrak{g}$, the exponential is a map from the algebra $\mathfrak{g}$ to the group G and finally the product of exponentials and Lie-group elements is an element of G. In effect, the approach proposed by Iserles aimed at evaluating the *fundamental solution* of the system (3.1), which is a one-parameter subgroup of the Lie group $\mathrm{GL}(d)$, the Lie group of invertible matrices (linear transformations) whose Lie algebra corresponds to the set of all $d \times d$ matrices.

Various numerical methods of different orders can be obtained by truncating the infinite series (3.4) and considering a certain number M of exponentials. For each step with stepsize h, one has $A_k(h) = \mathcal{O}(h^s)$, $s = 2^{k+1} - 2$ and $B_k(h) = \mathcal{O}(h^{s+1})$, being the integral of A_k in the interval $[0, h]$. Therefore $C_{k,1}(h) = [A_k(h), B_k(h)]$ has order $2s + 2$ since the first terms in the Taylor expansions of A_k and B_k commute. In general it is possible to show (cf. [12]) that

$$C_{k,i}(h) = \mathcal{O}\left(h^{(i+1)(s+1)}\right), \quad s = 2^{k+1} - 2. \tag{3.9}$$

Hence, for a method of order p, we choose the index M such that $M = \left\lfloor \log_2\left(\frac{p+1}{2}\right)\right\rfloor$, and, for each $0 \leq k \leq M - 1$ we evaluate i_k commutators, $C_{k,j}$, $1 \leq j \leq i_k$, where i_k is the first integer such that

$$(i_k + 2)(2^{k+1} - 1) \geq p + 1.$$

Table 1: First functions and their orders for the method of iterated commutators

$A_0(h)=\mathcal{O}(1)$	$A_1(h)=\mathcal{O}(h^2)$	$A_2(h)=\mathcal{O}(h^6)$	$A_3(h)=\mathcal{O}(h^{14})$	$\ldots$
$B_0(h)=\mathcal{O}(h)$	$B_1(h)=\mathcal{O}(h^3)$	$B_2(h)=\mathcal{O}(h^7)$	$B_3(h)=\mathcal{O}(h^{15})$	
$C_{0,1}(h)=\mathcal{O}(h^2)$	$C_{1,1}(h)=\mathcal{O}(h^6)$	$C_{2,1}(h)=\mathcal{O}(h^{14})$	$\vdots$	
$C_{0,2}(h)=\mathcal{O}(h^3)$	$\underline{C_{1,2}(h)=\mathcal{O}(h^9)}$	$C_{2,2}(h)=\mathcal{O}(h^{21})$		
$C_{0,3}(h)=\mathcal{O}(h^4)$	$C_{1,3}(h)=\mathcal{O}(h^{12})$	$\vdots$		
$\vdots$	$\vdots$			
$\underline{C_{0,9}(h)=\mathcal{O}(h^{10})}$				
$\vdots$				

Some of the above functions and their orders for a step with stepsize h are tabulated in table 1. For instance, for order $p=10$, we have $M=2$ (three exponential factors). To construct $A_{k+1}(h), k=0,1$, we consider in the k-th column all the commutators up to the underlined one (inclusive). Thus i_0 and i_1 equal 9 and 2 respectively.

The definition of the numerical method is completed by specifying the quadrature formulae to integrate the functions $A_k(h)$. One might choose the same quadrature rule for all the $0 \le k \le M$ or different rules for each value of k. In this case, it might be convenient to consider *quadrature with moments,*[2] given the particular form of the matrices $A_k(h), k=1,2\ldots$. Popular choices (cf. [26]) are Gaussian points or Radau points. We will use the following notation for a numerical method:

$$\mathrm{IC}_p Q_{n_1} Q_{n_2} \cdots Q_{n_r}, \qquad n_1+n_2+\cdots n_r = M+1,$$

where IC stands for Iterated Commutators, p is the order of the method, Q is the initial letter of the quadrature formula used and its index denotes how many times in a row the same formula is used, like for atoms in molecules. The index 1 will be usually omitted. Thus $\mathrm{IC}_{14}G_2R$ means that we consider a method of order 14, with three exponentials, that Gaussian points are used for evaluating the first two integrals, while a Radau formula is used for the third integral.

Other variants [26] can be obtained by replacing the product of exponentials with a single exponential, by means of the Baker–Campbell–Hausdorff formula [25]. However, these are technicalities and an interested reader may find coefficients, more detailed algorithms and examples of implementation in the appendix.

To conclude, we mention that the method of iterated commutators is related to the *Lie method of reduction* that is used to find solutions of equations of Lie-type by successive analytical quadratures. More about this issue can be found in [28].

[2] A moment of order s corresponds to the weight function $w(x)=x^s$

3.2 The Magnus series

We consider once again the Lie-group differential equation

$$y' = A(t)y, \qquad y_0 \in G, \quad A(t) \in \mathfrak{g},$$

and we wish to represent its solution by means of a single exponential applied to the initial condition y_0. In other words, the problem reduces to finding (whenever it exists) a function $\sigma(t) \in \mathfrak{g}$ such that the solution of the Lie-group equation can be written in the form

$$y(t) = \mathrm{e}^{\sigma(t)} y_0, \quad \sigma \in \mathfrak{g}.$$

The issue was considered already by Hausdorff in 1906, who proved that $\sigma(t)$ satisfies the implicit differential equation

$$\sum_{k=0}^{\infty} \frac{(-1)^k}{(k+1)!} \mathrm{ad}_{\sigma'}^k(\sigma) = A, \quad \sigma(0) = O, \tag{3.10}$$

where the adjoint operator ad has been introduced in (3.8). Thus, since $\mathrm{ad}_u^k(v)$ is linear in v but not in u, the function σ obeys a differential equation by far much more complicated than original linear equation. However, in 1954, Magnus [14] was able to write (3.10) in the normal form

$$\sigma' = A + \tfrac{1}{2}\mathrm{ad}_A(\sigma) + \sum_{k=1}^{\infty} \frac{B_{2k}}{(2k)!} \mathrm{ad}_A^{2k}(\sigma), \tag{3.11}$$

where the B_ℓs are the *Bernoulli numbers.* Magnus derived also the first terms of the solution σ of (3.11),

$$\begin{aligned} \sigma(t) \;=\; & \int_0^t A(\tau_0)\,\mathrm{d}\tau_0 + \tfrac{1}{2}\int_0^t \left[A(\tau_0), \int_0^{\tau_0} A(\tau_1)\,\mathrm{d}\tau_1\right] \mathrm{d}\tau_0 \\ & + \tfrac{1}{4}\int_0^t \left[A(\tau_0), \int_0^{\tau_0} \left[A(\tau_1), \int_0^{\tau_2} A(\tau_2)\,\mathrm{d}\tau_2\right] \mathrm{d}\tau_1\right] \mathrm{d}\tau_0 \\ & + \tfrac{1}{12}\int_0^t \left[\left[A(\tau_0), \int_0^{\tau_0} A(\tau_1)\,\mathrm{d}\tau_1\right], \int_0^{\tau_2} A(\tau_2)\,\mathrm{d}\tau_2\right] \mathrm{d}\tau_0 + \cdots \end{aligned} \tag{3.12}$$

and he analysed some conditions under which the function σ exists in a neighbourhood of the matrix function A. The infinite series σ converges for $t > 0$ sufficiently small but might not converge in the large.

However, Magnus did not address the problem of determining the order of approximation of a truncated expression for σ neither he provided a constructive algorithm to derive other terms in the expansion. There two latter problem have been recently investigated and solved by Iserles and Nørsett [13]. Their analysis is based on *binary trees* representing integration and commutation. Furthermore, they reduce the quadrature of multiple integrals to univariate quadrature cutting drastically the number of function evaluations. The complexity of the methods takes into account also the cancellation that derive from the skew-symmetry of the commutator operator and the Jacobi identity,

$$[A,[B,C]] + [B,[C,A]] + [C,[A,B]] = O,$$

for all A, B and C in $\mathfrak{g}$. The methods proposed in [13] allow also an implementation *á la* iterated commutators, employing more than a single exponential. The coefficients for a method of this type are reported in table 6. The corresponding numerical scheme features in table 11, a Matlab code for Toda flows.

3.3 Methods of a Runge–Kutta type

3.3.1 The 'rigid-frame' approach of Crouch and Grossman

Motivated by problems arising in control theory, Crouch and Grossman published in 1993 an article about integration of ODEs on manifolds, introducing new numerical methods based on 'rigid frames' [7]. Given the manifold M and a point $p \in M$, we denote by TM_p the tangent space to M at the point p, while $TM = \bigcup_{p \in M} TM_p$ is the tangent bundle of M. A vector field F on M is a section of TM, i.e. to each point $p \in M$ it associates a vector $v(p)$ in the tangent space at p. For simplicity, when the vector field F is evaluated at a point p we will write $F(p) = F_p$. Next we assume that at the point p there exists a

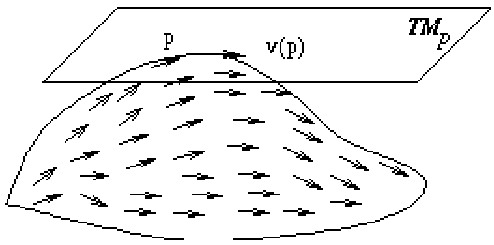

Figure 3.1: Vector field on a manifold

frame of vector fields, $E_i, i = 1 \ldots, s$, such that

$$F(p) = \sum_{i=1}^{s} f_i(p) E_i.$$

These vector fields generate a linear space, and should be chosen is such a way that their flow is easy to evaluate. Therefore, a differential equation on M,

$$y' = F(y), \qquad y \in M,$$

becomes

$$y' = \sum_{i=1}^{s} f_i(y) E_i. \tag{3.13}$$

Crouch and Grossman proposed to approximate the solution of (3.13) by a a scheme that is a generalization of classical explicit Runge–Kutta schemes: given that y_n is an approximation on M, one constructs the sequence of stages

$$
\begin{aligned}
p &= y_n, \\
F_p^1 &= F_p, \\
Y_r &= \mathrm{e}^{h a_{r,r-1} F_p^{r-1}} \circ \cdots \circ \mathrm{e}^{h a_{r,1} F_p^1} p, \qquad r = 1, \ldots, \nu \\
F_p^r &= \textstyle\sum_{i=1}^{r-1} f_i(Y_r) E_i, \qquad r = 1, \ldots, \nu \\
y_{n+1} &= \mathrm{e}^{h b_\nu F_p^\nu} \circ \cdots \circ \mathrm{e}^{h b_1 F_p^1} p,
\end{aligned}
\tag{3.14}
$$

where the exponential $\mathrm{e}^{hF_p} p$ denotes the flow relative to the vector field F at the point p and time $t = h$; the circles denote composition of maps and ν is the number of stages of the method.

It is possible to generalize such schemes to be implicit, and in that case one needs a procedure for solving nonlinear systems on manifolds. Owren and Welfert [23] have proposed a nonlinear solver that is a version of the Newton iteration when the manifold M is a Lie group. If M is instead a Grassman or a Stiefel manifold, then one might wish to use a variant of the conjugate gradients method, as proposed by Edelman, Arias and Smith [10].

In particular, the scheme (3.14) can be applied to Lie groups, whereby the frame $\{E_1, \ldots, E_s\}$ exists globally since the tangent bundle is trivial. In this case it is not necessary to change the frame during the time-stepping.

The order conditions for (3.14) have been systematically analysed by Owren and Marthinsen [21]. The stages (3.14) are expanded and can be represented with trees, with a formalism very similar to classical analysis of Runge–Kutta methods. However, for a given order, there are more conditions than for classical RK schemes: whereby in the classical theory trees corresponding to elementary differentials are non-planar, it is not so for the Crouch–Grossman (CG) methods, whose trees are instead planar. Furthermore, the order conditions are not all linearly independent, and the reduction of the condition is related to the theory of top random shuffles [8].

At the moment, the methods of this form are available just for order $1 \le p \le 4$.

3.3.2 The methods of Munthe-Kaas

The methods of Munthe-Kaas exploit the Runge–Kutta formalism in a very different manner from the CG schemes. We may say that the CG schemes combine a number of elements in the group and that the coefficients of the scheme are chosen so that such combination approximates the true solution of the Lie-group equation to order p.

The idea at the basis of the Munthe-Kaas methods is instead that the approximations are performed in the algebra, and then the result is mapped back to the group by means of the exponential. The main advantage in following this procedure is that the Lie algebra $\mathfrak{g}$ is a linear space, hence we can perform all the classical operation that we usually perform in $\mathbb{R}^d$, like addition and multiplication by scalar, without major complications. A Lie group has instead a nonlinear structure. Thus, basic tools, like for instance Taylor expansions, become very difficult to use.

Also the order theory of the methods of Munthe-Kaas formulates nicely in terms of elementary differentials represented by trees [18].

Given the Lie-group differential equation

$$y' = A(t,y)y, \qquad y \in G, \quad A(t,y) \in \mathfrak{g},$$

the methods of Munthe-Kaas are easily constructed. First of all we choose the coefficients $(A, \boldsymbol{b}, \boldsymbol{c})$ of our favourite classical RK method. Assuming that $y_n \in G$ is known, we construct the stages and the approximation y_{n+1} of an explicit method in the following fashion

$$\begin{aligned}
\Psi_1 &= O \\
p_1 &= y_n \\
\tilde{K}_1 &= A(t_n, p_1) \\
K_1 &= \mathrm{dexp}^{-1}{}_{\Psi_1}(\tilde{K}_1) \\
&\vdots \\
\Psi_i &= h \textstyle\sum_{j=1}^{i-1} a_{i,j} K_j \qquad\qquad i = 2, \ldots, \nu, \\
p_i &= \exp(\Psi_i) y_n \\
\tilde{K}_i &= A(t_n + c_i h, p_i) \\
K_i &= \mathrm{dexp}^{-1}{}_{\Psi_i}(\tilde{K}_i) \\
&\vdots \\
\Psi &= h \textstyle\sum_{i=1}^{\nu} b_i K_i \\
y_{n+1} &= \exp(\Psi) y_n.
\end{aligned} \tag{3.15}$$

The scheme (3.15) can be easily generalized to be implicit.

The function dexp^{-1} is a ‘correction function’ since it corrects some non-commutative effects due to the non-commutative group product. For $A, B \in \mathfrak{g}$, we have

$$\mathrm{dexp}^{-1}{}_A(B) = B + \tfrac{1}{2}[A,B] + \sum_{k=1}^{\infty} \frac{B_{2k}}{(2k)!} \mathrm{ad}_A^{2k}(B),$$

which is the same as (3.11). The function dexp^{-1} is the inverse[3] of the function dexp,

$$\mathrm{dexp}_A(B) = \sum_{k=0}^{\infty} \frac{(-1)^k}{(k+1)!} \mathrm{ad}_A^k(B), \tag{3.16}$$

which features in the derivation of the exponential function,

$$\frac{\mathrm{d}\mathrm{e}^{A(t)}}{\mathrm{d}t} = \mathrm{dexp}_{A(t)}(A'(t))\mathrm{e}^{A(t)}$$

[25]. A numerical method of this class is reported in table 12, adapted to the solution of Toda flows.

[3]The function dexp is a diffeomorphism in a sufficiently small neighbourhood of the zero of $\mathfrak{g}$

Note that, if we choose $(G, \cdot) \equiv (R^d, +)$, then the exponential and the function dexp^{-1} reduce to the identity, therefore the scheme (3.15) of Munthe-Kaas reduce to the classical RK one. For more details, we refer the reader to [19, 18, 20].

3.4 Some numerical examples

The Lie-group methods introduced in this section are based on exponentials and commutators. Although one might think that evaluating exponentials and commutators is expensive, it is easy to convince ourself that in many situations this is not the case.

As a first case of study we consider the linear differential equation

$$y' = A(t)y, \qquad t \in [0, 5], \quad y \in \mathbb{R}^{3\times 3}, \tag{3.17}$$

where the matrix $A(t)$ is chosen to be

$$A(t) = \begin{pmatrix} 0 & -\sin(t) & \cos(t) \\ \sin(t) & 0 & -\sin(t) \\ -\cos(t) & \sin(t) & 0 \end{pmatrix}. \tag{3.18}$$

Since the matrix $A(t)$ is skew-symmetric, then (3.17)-(3.18) will evolve in the Lie group SO(3), provided that the initial condition is and orthogonal matrix. The initial condition is chosen to be the Q factor in the QR decomposition of the matrix

$$\begin{pmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{pmatrix}, \tag{3.19}$$

obtained by the `Matlab` command `[Q, R] = qr(magic(3))`.

In figures 3.2-3.3 we compare the accuracy versus floating point operations (flops) of various fourth-order methods. The error is measured as the departure (in 2-norm) of the numerical solution from the exact solution at $t = 5$. The marked points are the measurement for stepsizes $h = \frac{1}{2}, \frac{1}{4}, \ldots, \frac{1}{64}$ accordingly to the legend below.

- Blue circles joined by solid line: An explicit Runge–Kutta method (for the coefficients, cf. table 2);
- green circles joined by solid line: The same explicit Runge–Kutta method as above, whereby at each step we map back to the manifold SO(3) by means of a QR factorization, retaining the factor Q;
- magenta plus joined by dotted line: The implicit Gauss–Legendre Runge–Kutta method of order four (cf. table 3);
- red circles joined by solid line: The iterated commutators with Gaussian coefficients $\mathrm{IC}_4 G_2$ (cf. table 4);
- cyan circles joined by solid line: The iterated commutators with Gaussian coefficients $\mathrm{IC}_4 G_2$ with BCH formula (cf. table 4);

- magenta circles joined by solid line: The iterated commutators with Gauss–Radau points IC_4GR (cf. table 5);

- blue plus joined by dotted line: The iterated commutators with Gauss–Radau points IC_4GR with BCH formula (cf. table 5);

- green plus joined by dotted line: A Magnus series with two exponentials of Iserles and Nørsett (cf. table 5);

- red plus joined by dotted line: A Lie-group method of Munthe-Kaas with coefficients as in table 2.

In figure 3.2 we employ a naive approach to solve the Lie-group equation: matrix exponentials are evaluated using the `Matlab` built-in function `expm`, while matrix commutators are evaluated using the formula $[A, B] = AB - BA$. The IC_4GR without BCH formula

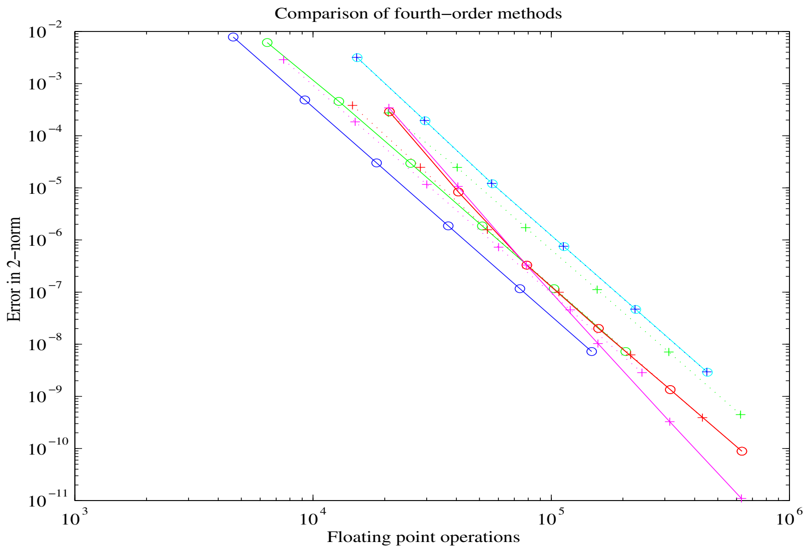

Figure 3.2: Error versus flops, for $A(t)$ chosen as (3.18)

display order five for this particular example, as it can be deduced from the slope of its corresponding line. Note that the Lie-group methods display generally a smaller error constant. In particular, observe that both the IC_4G_2 and the RK scheme of Munthe-Kaas overlay almost exactly the explicit method with QR projection: although for a fixed stepsize they are more expensive, because of the smaller error constant, they reach the same precision at the same cost with a stepsize roughly speaking four times as large!

In figure 3.3 the Lie-group methods are optimized by using the Euler–Rodrigues formula for evaluating the exponential (cf. [15, 16]) and the vector cross-product to evaluate the commutator [15]. This reduces the number of flops for the exponential from about 750 to 41 and the number of flops for the commutator from about 65 to 9. There is no doubt that Lie-group methods are by far more advantageous than classical methods. At the same cost (number of floating point operations) all Lie-group methods display smaller error, gaining up to two more significant digits (IC_4G_2).

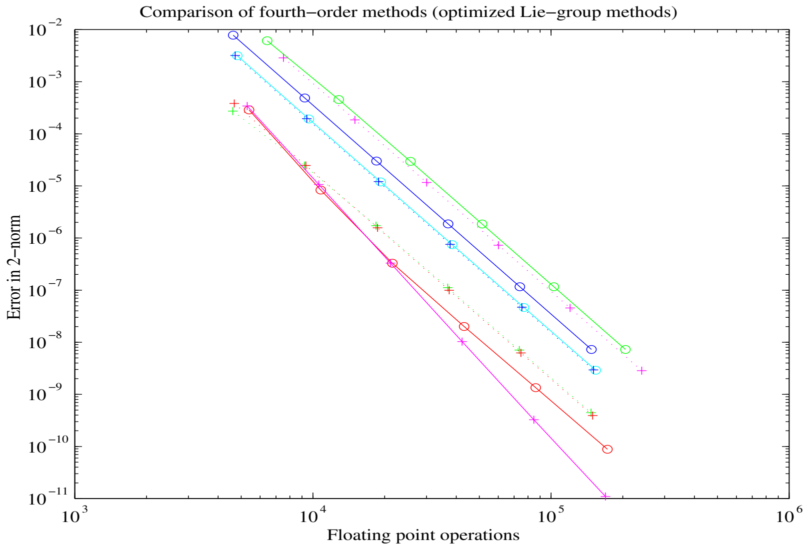

Figure 3.3: Error versus flops, for $A(t)$ chosen as (3.18), and optimized Lie-group methods

In the next example (figure 3.4), we change the matrix A to be

$$A(t) = \begin{pmatrix} 0 & -t & \cos(t) \\ t & 0 & \log(1+t) \\ -\cos(t) & -\log(1+t) & 0 \end{pmatrix}, \tag{3.20}$$

and we compare the same methods as before. The results are very similar. The IC$_4G_2$ and the Magnus series display a very small error constant but at last their lines become parallel to the other fourth order methods. The IC$_4GR$ coefficients produces numerical approximations which again is consistent with order five.

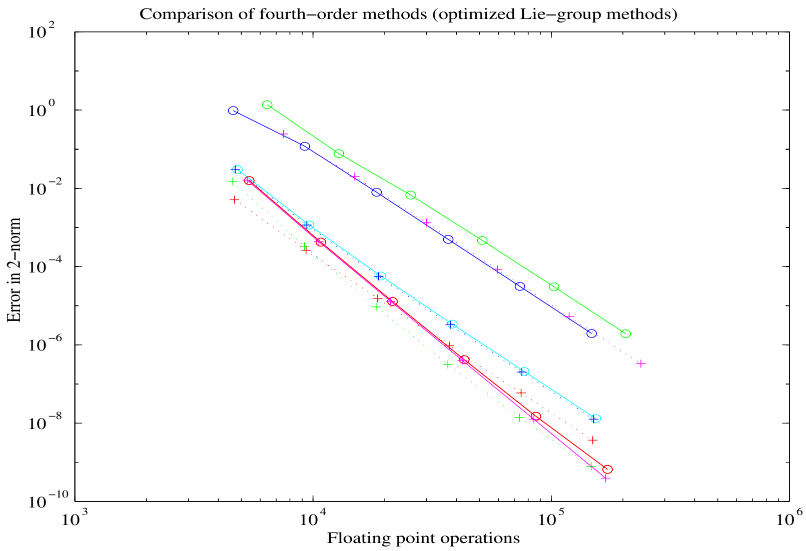

Figure 3.4: Error versus flops, for $A(t)$ as in (3.20), with optimized Lie-group methods

As a last example, we consider a problem for the Lie group SL(3). We choose the matrix $A(t)$ in (3.17) as

$$A(t) = \begin{pmatrix} t & \cos(t) & t^2 \\ -\sin(t) & 0 & t(t-2) \\ -t & 0 & -t \end{pmatrix}. \tag{3.21}$$

In this case both the explicit RK method and the Gauss–Legendre Runge–Kutta method do not preserve the determinant of the initial condition, therefore if we wish to stay on the manifold, we need to map the numerical approximation back on SL(3). At each step we normalize the numerical approximation by dividing the matrix by the third root of its determinant. The numerical results are displayed in figure 3.5, where the error is evaluated at $t = 5$ for stepsize $h = \frac{1}{2}, \frac{1}{4}, \ldots, \frac{1}{512}$. The projection of the Gauss–Legendre

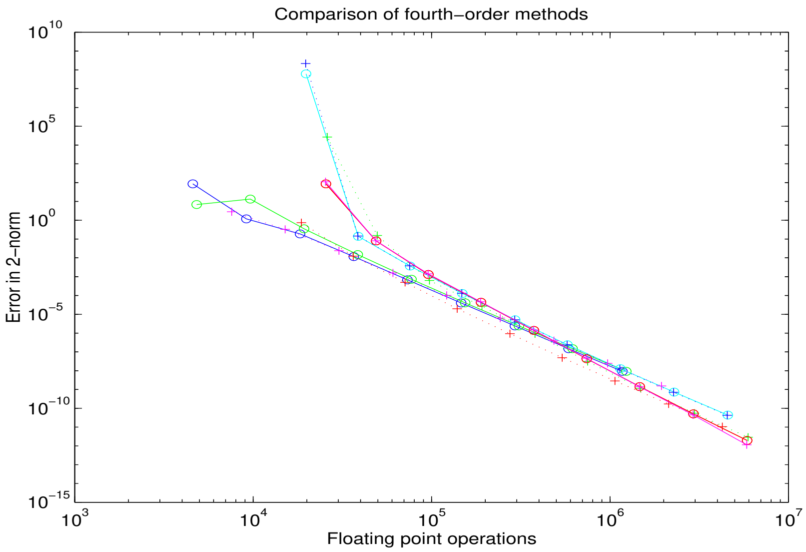

Figure 3.5: Error versus flops, for $A(t)$ as in (3.21), with non-optimized Lie-group methods

RK is not plotted since it is not significatively different from the GLRK itself. Note that *the Lie group methods are not optimized,* since both the Rodrigues formula and the vector cross-product formula for the commutator cannot be used in this case — nonetheless, Lie-group methods are cheaper than the classical methods when an accuracy higher than five significant digits is required. Note that for large stepsizes ($h > \frac{1}{4}$) the methods that employ the BCH formula, as well as the Magnus series, produce results quite far from the exact solution. This is not surprising since we may expect that the BCH formula and Magnus expansion do not converge for large stepsize.

3.5 Conclusions

The numerical experiments presented in §3.4 display features of numerical methods which, at a first glance, are completely counterintuitive. Even if the Lie-group methods that we have introduced in this section might appear very complicated and expensive, numerical results show that this is not necessarily the case and *it is possible* to derive cheap methods.

The result obtained in $\mathbb{R}^{3\times 3}$ encourage the conjecture that, for a general dimension d, it is possible to obtain numerical methods at least comparable with classical explicit schemes, if not cheaper. Lie groups possess a lot of structure and this should be exploited also in the numerical implementation of the methods.

With regard to nonlinear problems, at this stage it is not clear in which way to generalize the method of iterated commutators or the Magnus series. Several approaches exist. In [26] some recursive implementation of the Lie-group methods and some waveform relaxation techniques are proposed. In the numerical experiments for isospectral flows, presented in §4.1, nonlinear equations are linearized at the point t_n by means of a Taylor expansion of order sufficently high.

To conclude this section, we extrapolate from the examples presented in §3.4 that Lie-group methods are advantageous when one needs to find the solution of a small-dimensional problem to machine accuracy and as fast as possible. This constitutes one possible 'environmental niche' for such methods, besides various other applications.

4 Lie-group methods for isospectral flows

4.1 Iterated commutators and Magnus series for isospectral flows

The method of iterated commutators and Magnus series that we have presented in section 3 are conveniently implemented for a linear problem. We need, however, to find a way to apply these methods to the nonlinear differential equation

$$u' = B(uL_0u^T)u, \qquad u_0 = I.$$

Suppose that an (isospectral) approximation $L_n \approx L(t_n)$ is given. Then, in the interval $[t_n, t_n + h]$ we solve instead the following linear problem

$$u' = \tilde{B}(t)u, \qquad u(t_n) = I, \tag{4.1}$$

where the matrix B is a Taylor expansion at $t = t_n$ for the function $B(uL_nu^T)$ in the interval $[t_n, t_n + h]$. Obviously, the expansion has to be of order sufficiently high not to reduce the order of the numerical method that we are employing in the numerical solution of (4.1). Assume that the order of the method is p: then it is sufficient to consider the Taylor expansion to order $p - 1$. At this point, the equation (4.1) is of a linear type and we can employ the method of iterated commutators or Magnus series.

We evaluate the expansion for methods of order four, assuming (like in most examples of isospectral flows, for instance Toda flows, double-bracket flows and so on) that the matrix B is linear in its arguments, i.e.

$$B(\alpha L_1 + \beta L_2) = \alpha B(L_1) + \beta B(L_2).$$

For simplicity sake, whenever there is no danger of confusion, we denote L_n by L and $B(L)$ by B. Taking into account that

$$L' = [B, L], \qquad B'(L) = B(L') = B([B, L]),$$

we have the following expressions for the other derivatives of L

$$\begin{aligned} L'' &= [B([B,L]),L] + [B,[B,L]], \\ L''' &= [B([B([B,L]),L] + [B,[B,L]]),L] + 2[B([B,L]),[B,L]] \\ &\qquad + [B,[B([B,L]),L]] + [B,[B,[B,L]]]. \end{aligned} \tag{4.2}$$

Hence

$$\tilde{B}(t) = B(L + hL' + \tfrac{1}{2}h^2L'' + \tfrac{1}{6}h^3L''') = B(uL_nu^T) + \mathcal{O}\left(h^4\right). \tag{4.3}$$

There are many advantages in following this procedure, in comparison with the *modified Gauss–Legendre Runge–Kutta* methods for isospectral flows introduced in [6]. First of all, the schemes obtained in this way are explicit, therefore there is no need for time-consuming and expensive nonlinear iterations at each step, as for the MGLRK methods, and secondly, the methods produce automatically orthogonal matrices.

4.2 Lie-group Runge–Kutta-type methods for isospectral flows

Since the associated orthogonal flow is nonlinear in u, then we might also decide to employ instead Lie-group methods of a Runge–Kutta type, as proposed by Munthe-Kaas [19] and Owren and Marthinsen [21]. In such case, we exploit the associated nonlinear orthogonal flow

$$u' = B(t,u)u, \qquad u_n = I, \qquad t \in [t_n, t_n + h],$$

where

$$B(t,u) = B(u(t)L_nu(t)^T).$$

Once that the approximation u_{n+1} for $u(t_{n+1})$ has been evaluated, we compute $L_{n+1} = u_{n+1}L_nu_{n+1}^T$.

4.3 Comparison of numerical methods for isospectral flows

We present some numerical experiments to compare the numerical methods and their performances when applied to isospectral flows. As a case of study we consider a Toda flow

$$L' = [B(L), L] \qquad t \in [0, 40],$$

with the initial condition

$$L_0 = \begin{pmatrix} -1 & 1 & 0 \\ 1 & 0.5 & 1 \\ 0 & 1 & 0.5 \end{pmatrix}. \tag{4.4}$$

As already mentioned in [6, 27], these parameters correspond to displacement $\boldsymbol{q}_0 = (0,0,0)$ and momentum $\boldsymbol{p}_0 = (1, -0.5, -0.5)$ for a lattice of three particles. Recall that three is the minimal number of particles we need to consider to distinguish between the isospectral and non-isospectral behaviour of some classical methods like for instance Gauss–Legendre RK schemes.

We compare the following methods of order four:

- Blue circles joined by solid line: An explicit Runge–Kutta method (with coefficients in table 2);
- magenta plus joined by dotted line: The implicit Gauss–Legendre Runge–Kutta method of order four (cf. table 3);

which, accordingly to the theory developed in [6], are not isospectral, and the following isospectral methods:

- magenta circles joined by solid line: Modified Gauss–Legendre RK method as proposed in [6] (cf. table 8);
- red circles joined by solid line: IC_4G_2 (cf. table 9);
- cyan circles joined by solid line: IC_4G_2 (cf. table 9) with BCH formula;
- blue plus joined by dotted line: IC_4GR (cf. table 10) with BCH formula;
- green plus joined by dotted line: Magnus series with two exponentials of Iserles and Nørsett (cf. table 11);
- red plus joined by dotted line: A Lie-group method of Munthe-Kaas (cf. table 12).

The first isospectral method is implicit and the implicit iterations are solved by fixed point iteration; the remaining isospectral methods solve (4.1) with Lie-group explicit methods.

The numerical experiment consists of sampling the error and the number of floating point operations of each numerical method at different times, $t = 5, 10, \ldots, 40$, and for different values of the stepsize of integration, $h = \frac{1}{2}, \frac{1}{4}, \ldots, \frac{1}{64}$. For this example, the MGLRK of order four fails to converge with fixed point iterations when $h = \frac{1}{2}$. The corresponding data are hence not reported.

In figure 4.1 we plot the error of the numerical methods against the number of floating point operations when halving the stepsize. Each line correspond to a value of time $t \in \{5, \ldots, 40\}$. For a fixed value of time, the cheaper a method is, the nearest is its line to the bottom left corner. Note also that, for a fixed stepsize, the explicit method is only slightly cheaper than the isospectral methods that employ Lie-group integrators but the error of the latter methods is by far much smaller.

Figure 4.2 is the same as figure 4.1, except that in this case, the lines correspond to individual trajectories for different values of the stepsize. Note that the error of the non-isospectral method is constant over the whole interval of integration, as also emphasized in figure 4.3, where the error is plotted against time for stepsize $h = \frac{1}{4}$. While for non-isospectral methods numerical error in the first steps if fatal, for isospectral methods the error decays and becomes of the order of machine accuracy when $t \to \infty$.

Keeping in mind figure 4.2, we see that the behaviour of all the isospectral methods that employ Lie-group schemes is very much the same. Except for the method of Munthe-Kaas, both the iterated commutators and the Magnus series have the same error and tiny difference in the number of floating point operations. The numerical error here is essentially due to the approximation of the function B accordingly to the formula (4.3). The lines corresponding to these methods overlay almost exactly in figures 4.3-4.4.

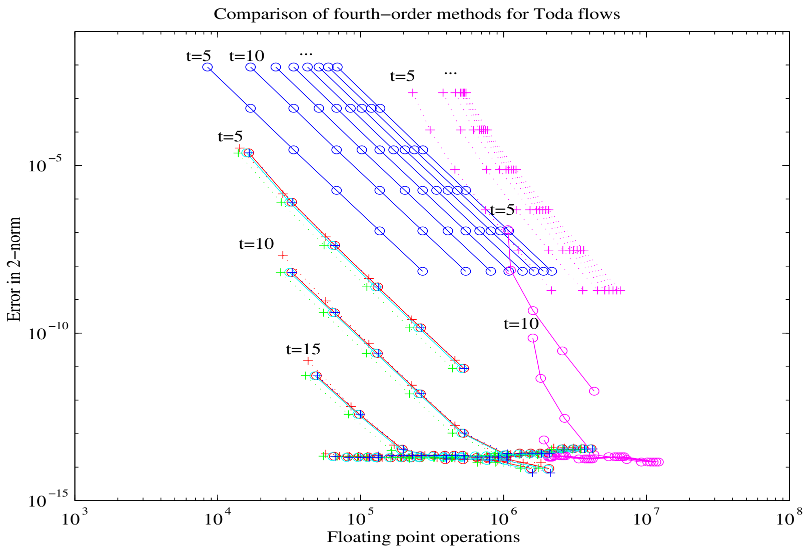

Figure 4.1: Error versus floating point operations at fixed values of time while halving the stepsize

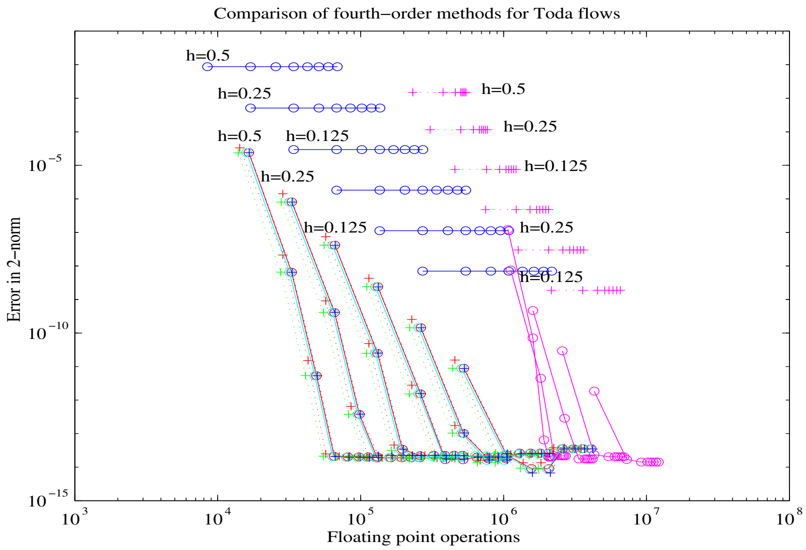

Figure 4.2: Error versus floating point operations for trajectories corresponding to the same stepsize, for various values of h

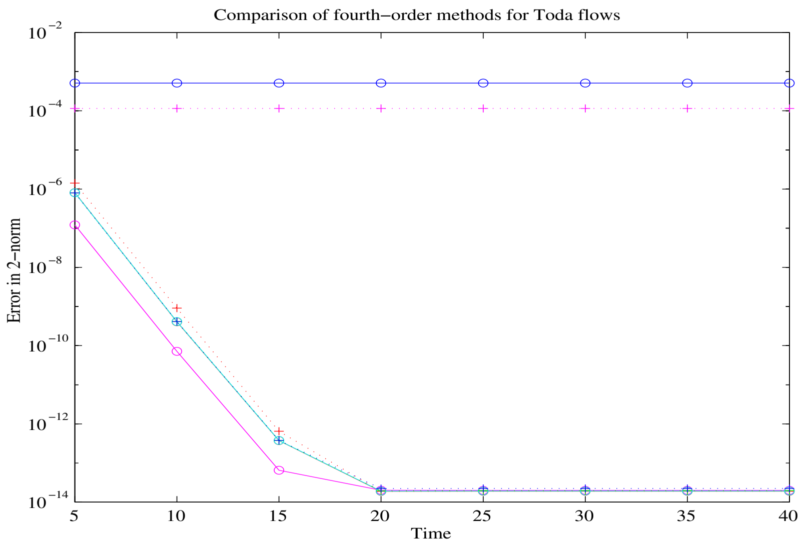

Figure 4.3: Error versus time for stepsize of integration equal to $h = \frac{1}{4}$

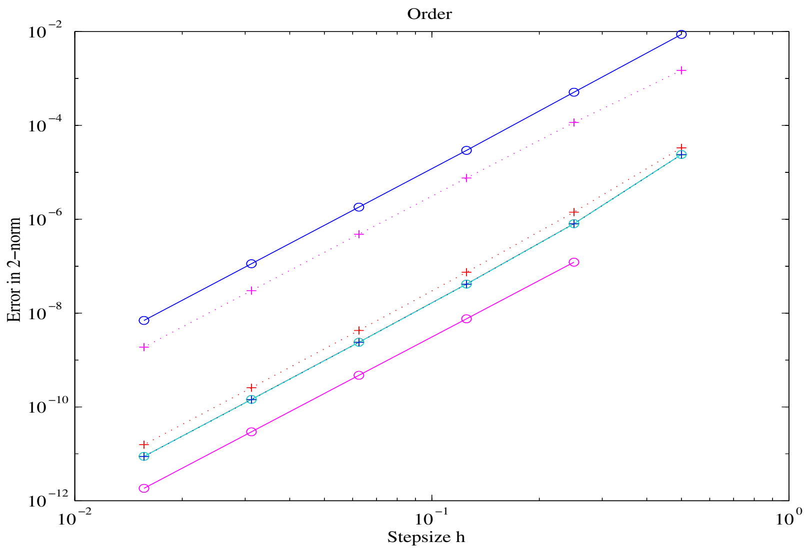

Figure 4.4: Error versus stepsize evaluated at time $t = 5$

Since the behaviour of isospectral and non-isospectral methods is so different (clearly in favour of explicit isospectral methods that use Lie-group numerical methods), in figure 4.4 we plot the error at $t = 5$ versus the stepsize of integration h. Both the axis are in logarithmic scale, hence we can evaluate the order of the method from the slope of the error line corresponding to the method: for each power of ten in the abscissae we count the powers of ten in the ordinate for each line. All the lines are parallel, which means that the methods are all of the same order, and in effect, all the methods compared in figures 4.1-4.3 have order four. Note that all the isospectral methods have a smaller error than non-isospectral schemes.

Acknowledgements

I would like to express my gratitude to Arieh Iserles and Hans Munthe-Kaas for having carefully read the manuscript and providing helpful comments and suggestions.

References

[1] R. Abraham, J.E. Marsden and T. Ratiu, *Manifolds, Tensor Analysis, and Applications,* 2nd edition, Applied Mathematical Sciences, **75,** Springer-Verlag, 1988.

[2] R.W. Brockett, "Dynamical systems that sort lists, diagonalize matrices, and solve linear programming problems", *Lin. Alg. Appl.*, **146**, 79–91, 1991.

[3] R.W. Brockett, "Differential geometry and the design of gradient algorithm", *Proceedings of Symposia in Pure Mathematics,* AMS, **54**, part 1, 69–92, 1993.

[4] R.L. Bryant, "An introduction to Lie groups and symplectic geometry", in *Geometry and Quantum Field Theory,* (D.S. Freed and K.K. Uhlenbeck eds) AMS IAS/Park City Math. Series, vol. 1, 1995.

[5] J.C. Butcher, "A-stability property of implicit Runge–Kutta methods", *BIT,* **45**, 358–361, 1975.

[6] M.P. Calvo, A. Iserles and A. Zanna, "Numerical solution of isospectral flows", DAMTP Techn. Rep. NA03, University of Cambridge, 1995, to appear in *Math. of Comp.*

[7] P.E. Crouch and R. Grossman, "Numerical integration of Ordinary Differential Equations on manifolds", *J. Nonlinear Sci.*, **3**, 1–33, 1993.

[8] P. Diaconis, J.A. Fill and J. Pitman, "Analysis of top random shuffles", *Combinatorics, Probability and Computing,* **1,** 1992.

[9] L. Dieci, R.D. Russell and E.S. van Vleck, "Unitary integrators and applications to continuous orthonormalization techniques", *SIAM J. Numer. Anal.*, **31**, 261–281, 1994.

[10] A. Edelman, T. Arias and S.T. Smith, "Conjugate gradient and Newton's method on the Grassman and Stiefel manifolds", submitted to *SIAM J. Matrix Anal. Appl.*

[11] D.J. Higham, "Runge–Kutta type methods for orthogonal integration", Techn. Rep. NA/168, Department of Mathematics and Computer Science, University of Dundee, 1996.

[12] A. Iserles, "Solving linear ordinary differential equations by exponentials of iterated commutators", *Numer. Math.,* **45**, 183–199, 1984.

[13] A. Iserles and S.P. Nørsett, "On the solution of differential equations in Lie groups", in preparation.

[14] W. Magnus, "On the exponential solution of Differential Equations for a linear operator", *Comm. Pure and Appl. Math.*, **7**, 649–673, 1954.

[15] J.E. Marsden and T.S. Ratiu, *Introduction to Mechanics and Symmetry,* Springer-Verlag, New York, 1994.

[16] A. Marthinsen, H. Munthe-Kaas and B. Owren, "Simulation of ordinary differential equations on manifolds — Some numerical experiments and verifications", in *Applied Modelling and Simulation - Proceedings of the 38th SIMS Simulation Conference,* T. Iversen (editor), Tapir trykkeri, Norway, 231–242, 1996.

[17] J.B. Moore, R.E. Mahony and U. Helmke, "Numerical gradient algorithms for eigenvalue and singular value calculations", *SIAM J. Matrix Anal. Appl.*, **15**, 881–902, 1994.

[18] H. Munthe-Kaas, "Lie–Butcher theory for Runge–Kutta methods", *BIT,* **35,** 572–587, 1995.

[19] H. Munthe-Kaas, "Runge–Kutta methods on Lie groups", submitted to *BIT.*

[20] H. Munthe-Kaas and A. Zanna, "Numerical integration of differential equations on homogeneous manifolds", in *Foundations of Computational Mathematics* (F. Cucker, ed.), Springer-Verlag, New York, 305–315, 1997.

[21] B. Owren and A. Marthinsen, "Integration methods based on rigid frames", NTNU Techn. Rep. 1/1997, 1997.

[22] A.M. Perelomov, *Integrable Systems of Classical Mechanics and Lie Algebras,* vol. 1, Birkhäuser Verlag, Basel, 1990.

[23] B. Owren and B. Welfert, "The Newton iteration on Lie groups", NTNU Techn. Rep. 3/1996, 1996.

[24] M. Toda, *Theory of Nonlinear Lattices,* Springer-Verlag, Berlin, 1978.

[25] V.S. Varadarajan, *Lie Groups, Lie Algebras, and Their Representations,* Springer-Verlag, New York, 1984.

[26] A. Zanna, "The method of iterated commutators for ordinary differential equations on Lie groups", DAMTP Techn. Rep. NA12, University of Cambridge, 1996.

[27] A. Zanna, *Numerical Analysis of Isospectral Flows,* essay, 1996.

[28] A. Zanna and H. Munthe-Kaas, "Iterated commutators, Lie's reduction method and ordinary differential equations on matrix Lie groups", in *Foundations of Computational Mathematics* (F. Cucker, ed.), Springer-Verlag, New York, 434–441, 1997.

Appendix and tables

Table 2: The Butcher tableau of an explicit fourth-order Runge–Kutta method.

$$\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2} & & & \\
\frac{1}{2} & 0 & \frac{1}{2} & & \\
1 & 0 & 0 & 1 & \\
\hline
 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
\end{array}$$

Table 3: Coefficients of the fourth-order Gauss–Legendre RK method

$$\begin{array}{c|cc}
\frac{1}{2}-\frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{1}{4}-\frac{\sqrt{3}}{6} \\
\frac{1}{2}+\frac{\sqrt{3}}{6} & \frac{1}{4}+\frac{\sqrt{3}}{6} & \frac{1}{4} \\
\hline
 & \frac{1}{2} & \frac{1}{2}
\end{array}$$

Table 4: Coefficients the iterated commutators of order 4 with Gaussian points (IC$_4G_2$)

B_0	nodes	$\frac{1}{2}-\frac{\sqrt{3}}{6}$	$\frac{1}{2}+\frac{\sqrt{3}}{6}$
	weights	$\frac{1}{2}$	$\frac{1}{2}$
B_1	nodes	$\frac{1}{2}-\frac{\sqrt{3}}{6}$	$\frac{1}{2}+\frac{\sqrt{3}}{6}$
	weights	$\frac{1}{2}$	$\frac{1}{2}$

Table 5: Coefficients for iterated commutators of order 4 with Gauss–Radau points (IC$_4GR$)

B_0	nodes	$\frac{1}{2}-\frac{\sqrt{3}}{6}$	$\frac{1}{2}+\frac{\sqrt{3}}{6}$
	weights	$\frac{1}{2}$	$\frac{1}{2}$
B_1	nodes	$\frac{3}{5}$	1
	weights	$\frac{125}{216}$	$\frac{1}{8}$

Table 6: Coefficients for the Magnus series of order 4 with $r = 2$ (cf. [13])

σ_0	nodes	$\frac{1}{2}-\frac{\sqrt{3}}{6}$	$\frac{1}{2}+\frac{\sqrt{3}}{6}$
	weights	$\frac{1}{2}$	$\frac{1}{2}$
σ_1	nodes	1	
	weights	$\frac{\sqrt{3}}{6}$	

Table 7: A Matlab program for the classical Gauss–Legendre of order 4 for Toda flows

```
function L1 = TodaGLRK4(L0, h, t0, tf);

% function L1 = TodaGLRK4(L0, h, t0, tf);
%
% This function solves the Toda flow with the classical
% Gauss--Legendre RK method of order 4 (not isospectral).
%
% L0 = initial condition           [t0, tf] = interval of integration
% h = stepsize of integration      L1 = numerical solution at t = tf

I = eye(size(L0));
c1 = 0.5 - sqrt(3)/6;
c2 = 0.5 + sqrt(3)/6;
a11 = 0.25;
a12 = 0.25 - sqrt(3)/6;
a21 = 0.25 + sqrt(3)/6;
a22 = 0.25;
b1 = 0.5;
b2 = 0.5;
epsilon = 1e-14;                % tolerance on fixed point iterations
for tn = t0 : h : tf-eps
  L1 = L0;
  Csi = L0;
  Eta = L0;
  err = 1;
  while err > epsilon
    Bcsi = QRflow(Csi);         % QRflow(L) = L_{-} - L_{+}
    Beta = QRflow(Eta);
    Xcsi = comm(Bcsi, Csi);     % comm(A,B) = AB - BA
    Xeta = comm(Beta, Eta);
    Csi1 = L0 + h * (a11 * Xcsi + a12 * Xeta);
    Eta1 = L0 + h * (a21 * Xcsi + a22 * Xeta);
    err = norm([Csi, Eta] - [Csi1, Eta1]);
    Csi = Csi1;
    Eta = Eta1;
  end
  K1 = comm(QRflow(Csi), Csi);
  K2 = comm(QRflow(Eta), Eta);
  L1 = L0 + h * (b1 * K1 + b2 * K2);
  L0 = L1;
end
```

Table 8: A Matlab program for the Modified Gauss–Legendre of order 4 for Toda flows.

```
function L1 = TodaMGLRK4(L0, h, t0, tf);

% function L1 = TodaMGLRK4(L0, h, t0, tf)
%
% this function evaluates the Toda flow with of the Modified Implicit
% Gauss--Legendre RK method of order 4 (isospectral).
%
% L0 = initial condition           [t0, tf] = interval of integration
% h = stepsize of integration      L1 = numerical solution at t = tf

n = size(L0);
I = eye(n);
c1 = 0.5 - sqrt(3)/6;
c2 = 0.5 + sqrt(3)/6;
a11 = 0.25;
a12 = 0.25 - sqrt(3)/6;
a21 = 0.25 + sqrt(3)/6;
a22 = 0.25;
b1 = 0.5;
b2 = 0.5;
epsilon = 1e-14;           % tolerance on fixed point iterations
for tn = t0 : h : tf-eps
  L1 = L0;
  err = 1;
  while err > epsilon
    [B1, B2] = Btilde([c1,c2], h, L0, L1);    % Btilde evaluates the
                                              % Hermite interpolant at
                                              % t = c1 and t = c2
    G = [ I-a11*h*B1   -h*a12*B2;
           -a21*h*B1  I-h*a22*B2];
    V = G \ [I; I];
    X1 = V(1:n, :);
    X2 = V(n+1 : 2*n, :);
    K1 = B1 * X1;
    K2 = B2 * X2;
    U1 = I + h * ( b1 * K1 + b2 * K2);
    LL = U1 * L0 * U1';
    err = norm(L1 - LL);
    L1 = LL;
  end
  L0 = L1;
end
```

Table 9: A Matlab program for the $\mathrm{IC}_4 G_2$ method for Toda flows

```
function L1 = TodaIC4_Gauss(L0, h, t0, tf);

%  function L1 = TodaIC4_Gauss(L0, h, t0, tf);
%
% Solves the Toda flow with the Lie group invariant IC4G_2
%

c1 = 1/2 - sqrt(3)/6;
c2 = 1/2 + sqrt(3)/6;
w1 = 1/2;
w2 = 1/2;
c11 = c1^2 * h;
c12 = c1*c2 * h;
c22 = c2^2 * h;
for tn = t0 : h : tf-eps
  %
  % integrate the orthogonal flow
  %
  [X11, X12, X22, X1, X2] = B_IC4(L0, c11, c12, c22, c1*h, c2*h);
  sigma0_1 = c1 * h * (w1 * X11 + w2 * X12);
  sigma0_2 = c2 * h * (w1 * X12 + w2 * X22);
  sigma0_h = h * (w1 * X1 + w2 * X2);

  C01 = comm(X1, sigma0_1);
  C02 = comm(C01, sigma0_1);
  C03 = comm(C02, sigma0_1);
  A1_1 = 1/2 * C01 + 1/3 * C02 + 1/8 * C03;

  C01 = comm(X2, sigma0_2);
  C02 = comm(C01, sigma0_2);
  C03 = comm(C02, sigma0_2);
  A1_2 = 1/2 * C01 + 1/3 * C02 + 1/8 * C03;

  sigma1_h = h * (w1 * A1_1 + w2 * A1_2);

  U1 = ER_expm(sigma0_h) * ER_expm(sigma1_h);
          % for the corresponding methods with BCH formula, replace
          % the above U1 with
          % U1 = ER_expm(sigma0_h + sigma1_h+ 0.5*comm(sigma0_h, sigma1_h);
  L1 = U1 * L0 * U1';         % update the numerical solution
  L0 = L1;
end
```

Table 10: A Matlab program for the IC_4GR for Toda flows

```
function L1 = TodaIC4_Radau(L0, h, t0, tf);
%  function L1 = TodaIC4_Radau(L0, h, t0, tf);
%
% Solves the Toda flow with the Lie group invariant IC4GR
% B_IC4R(L0, t1,t2,t3,t4,t5,t6) evaluates the 3-rd order interpolant
% to B at t=t1,...t6.

c11 = 1/2 - sqrt(3)/6;
c12 = 1/2 + sqrt(3)/6;
w11 = 1/2;
w12 = 1/2;
c21 = 3/5;
c22 = 1;
w21 = 125/216;
w22 = 1/8;
t1 = c21*c11*h;
t2 = c21*c12*h;
t3 = (c21+(c22-c21)*c11)*h;
t4 = (c21+(c22-c21)*c12)*h;
t5 = c21*h;
t6 = h;
for tn = t0 : h : tf-eps
  % integrate the orthogonal flow
  [X1, X2, X3, X4, X5, X6] = B_IC4R(L0, t1,t2,t3,t4,t5,t6);
  sigma0_21 = c21 * h * (w11 * X1 + w12 * X2);
  sigma0_h = sigma0_21 + h*(1-c21)*(w11 * X3 + w12 * X4);
  C01 = comm(X5, sigma0_21);
  C02 = comm(C01, sigma0_21);
  C03 = comm(C02, sigma0_21);
  sigma1_1 = 1/2 * C01 + 1/3 * C02 + 1/8 * C03;
  C01 = comm(X6, sigma0_h);
  C02 = comm(C01, sigma0_h);
  C03 = comm(C02, sigma0_h);
  sigma1_2 = 1/2 * C01 + 1/3 * C02 + 1/8 * C03;
  sigma1_h = h * (w21 * sigma1_1 + w22 * sigma1_2);

  U1 = ER_expm(sigma0_h) * ER_expm(sigma1_h);
            % for the version with the BCH formula, replace
            % the above line with
            % U1 = ER_expm(sigma0_h+sigma1_h+0.5*comm(sigma0_h, sigma1_h));
  L1 = U1 * L0 * U1';            % update the numerical solution
  L0 = L1;
end
```

Table 11: A Matlab program for the Magnus method with $r = 2$ and order 4 for Toda flows

```
function L1 = TodaMagnus4(L0, h, t0, tf);

%  function L1 = TodaMagnus4(L0, h, t0, tf);
%
% Solves the Toda flow with the Lie group invariant method Magnus4
% (cf. Iserles and Norsett)
%

c1 = 1/2 - sqrt(3)/6;
c2 = 1/2 + sqrt(3)/6;
w11 = 1/2;
w12 = 1/2;
w21 = sqrt(3)/6;
for tn = t0 : h : tf-eps
  %
  % integrate the orthogonal flow
  %
  [X1, X2, Xh] = B_M4(L0, c1*h, c2*h, h);
  Psi_h =  h * (w11*X1 + w12*X2);
  sigma0_h = Psi_h + 0.5 * h^2 * w21 * comm(X2, X1);

  AD_1 = comm(Xh, sigma0_h);
  AD_2 = comm(AD_1, sigma0_h);
  AD_3 = comm(AD_2, sigma0_h);
  ADPsi_0 = comm(Xh, Psi_h);
  ADPsi_1 = comm(ADPsi_0, sigma0_h);
  ADPsi_2 = comm(ADPsi_1, sigma0_h);
  A1_h = 1/2 * AD_1 + 1/3 * AD_2 + 1/8 * AD_3 ...
      - (ADPsi_0 + 1/2 * ADPsi_1 + 1/6 * ADPsi_2)/2;

  sigma1_h = 1/4 * h * A1_h;
  U1 = ER_expm(sigma0_h) * ER_expm(sigma1_h);

  L1 = U1 * L0 * U1';           % update the numerical solution
  L0 = L1;
end
```

Table 12: A Matlab program for a 4-th order Lie-group method of Munthe-Kaas for Toda flows

```
function L1 = TodaMK4(L0, h, t0, tf);

%  function L1 = TodaMK4(L0, h, t0, tf)
%
% Solves the Toda flow with the Lie group invariant RK method RK4r.m

c = [0 1/2 1/2 1];
a = [0          0          0          0
    1/2         0          0          0
    0           1/2        0          0
    0           0          1          0   ];
b = [1/6 1/3 1/3 1/6];
ord = 4;
[n, m] = size(L0);
I = eye(n);
for tn = t0 : h : tf-eps
  %
  % integrate the orthogonal flow
  %
  P1 = I;
  K1 = QRflow(L0);

  U2 = h * a(2,1) * K1;
  P2 = ER_expm(U2)*P1;
  K2 = B_RK4(P2, L0);             % evaluates B(P2*L0*P2^T)
  K2 = dexpinvr(U2,K2,ord);

  U3 = h * (a(3,1) * K1 + a(3,2) * K2);
  P3 = ER_expm(U3)*P1;
  K3 = B_RK4(P3, L0);
  K3 = dexpinvr(U3,K3,ord);

  U4 = h * (a(4,1) * K1 + a(4,2) * K2 + a(4,3)*K3);
  P4 = ER_expm(U4)*P1;
  K4 = B_RK4(P4, L0);
  K4 = dexpinvr(U4,K4,ord);

  V = h * (b(1) * K1 + b(2) * K2 + b(3) * K3 + b(4)*K4);
  Y1 = ER_expm(V)*P1;

  L1 = Y1 * L0 * Y1';         % update the numerical solution
  L0 = L1;
end
```