*IMA Journal of Numerical Analysis* (2005) Page 1 of 36
doi: 10.1093/imanum/dri017

# On the numerical solution of involutive ordinary differential systems: enhanced linear algebra

JUKKA TUOMELA, *Department of Mathematics, University of Joensuu, PL 111, 80101 Joensuu, Finland. email: jukka.tuomela@joensuu.fi.*

TEIJO ARPONEN, *Mathematics Institute, University of Warwick, Coventry CV4 7AL, U.K. email: arponen@maths.warwick.ac.uk.*

VILLESAMULI NORMI, *Department of Mathematics, University of Joensuu, PL 111, 80101 Joensuu, Finland. email: villesamuli.normi@joensuu.fi*

We analyse some Runge-Kutta type methods for computing one-dimensional integral manifolds, i.e. solutions to ODEs and DAEs. We show that we can reliably and reasonably fast compute the solutions which respect all the constraints of the problem. Moreover, we show that so called impasse points are regular points in our approach and hence require no special attention.

*Keywords*: differential algebraic equations, integral manifolds, Runge-Kutta methods, Symmetric LQ.
AMS subject classification: primary 34A26, 65L05, secondary 35N10, 58F40

## 1. Introduction

We continue the work started in Tuomela & Arponen (2000) and Tuomela & Arponen (2001) where we showed that using jet spaces to analyse ordinary differential equations (ODEs) is useful and interesting also from the numerical point of view. We recall that in this approach the term ODE covers also overdetermined systems, differential-algebraic equations (DAEs) and implicit differential equations (IDEs). So the advantage of our point of view is that we are able to treat a very large class of systems in a unified way. For example in a well-known collection of initial value problems "IVP test set" (2003) the problems are classified as ODEs, DAEs or IDEs. However, in our framework this distinction is not needed, and indeed one may argue that it is misleading conceptually.

In Tuomela & Arponen (2000) we discussed the theoretical background of our approach. Intuitively we may say that our method uses explicitly all the constraints of the problem, and in this way we avoid completely the drift off phenomenon which is usually a problem in DAE computations. We also compared our approach to other possible ways to treat these types of problems, and extensive references to relevant literature were given. So we do not repeat this discussion here, but simply recall that the standard way to view DAEs can be found in well-known books Hairer & Wanner (1991), Brenan *et al.* (1989).

In Tuomela & Arponen (2001) we analysed and implemented higher order methods and showed that explicit Runge-Kutta methods can be adapted to our context. Sometimes DAE systems are characterized as "infinitely stiff". Consequently in traditional approaches one is forced to use implicit methods. However, as in our approach there is really no distinction between "ODEs" and "DAEs" it is obvious that "DAEs" are not intrinsically stiff. Hence it is no surprise that our explicit schemes worked very well in standard "DAE" test problems.

Our initial code was very slow, however, since it was implemented in *Maple*. In the present article we introduce a new version of our method which has been implemented with C++. Although this new implementation speeds up the method, the main purpose of this paper is to study more carefully and enhance the related algorithms. The results show remarkable speedup due to the enhancements, even 90%. We did not compare implementation issues, i.e. there is no comparison to our old Maple code.

Hence our code is now reasonably fast. However, in the present article we will only consider standard examples found in the literature to demonstrate the efficiency and reliability of our code. In future articles we will take up cases found in actual simulations of "real life" problems.

The structure of the article is as follows. In section 2 we briefly recall some background material (more details can be found in Tuomela & Arponen (2000) and Tuomela & Arponen (2001)). In section 3 we formulate our computational problem and in particular indicate how Runge–Kutta schemes can be adapted to our context. In section 4 we discuss Newton type methods for solving nonlinear systems. These methods are needed when we have to project some computed point to the relevant manifold defined by the problem. It turns out that the Newton matrix has a particular block structure that we can take advantage of.

In section 5 the computation of the distribution is considered; i.e. given a point on the solution curve we need to compute the tangent direction of this curve. It is worth pointing out that complexity of this task is independent of $q$, the order of our ODE. In general this leads to computation of a 1-dimensional kernel of a matrix. However, in some important special cases the relevant direction can be readily computed by solving a standard square system of linear equations. Then in section 6 we present computational examples to demonstrate the efficiency and reliability of our code. Finally in section 7 we discuss some future perspectives and extensions of the present work.

## 2. Basic tools

For more information on standard differential geometry we refer to Spivak (1979) and on jets to Saunders (1989). All maps and manifolds are assumed to be smooth, i.e. infinitely differentiable. All analysis is local, hence various maps and manifolds need to be smooth or defined only in some appropriate subsets. To simplify the notation these subsets are not indicated.

### 2.1 *Multilinear maps*

The $\ell$'th differential of a map $f : \mathbb{R}^m \mapsto \mathbb{R}^k$ is denoted by $d^\ell f$ and its value at $p$ by $d^\ell f_p$. Now $d^\ell f_p$ is a multilinear map $d^\ell f_p : \mathbb{R}^m \times \cdots \times \mathbb{R}^m \mapsto \mathbb{R}^k$ where $\mathbb{R}^m$ appears $\ell$ times. Note that $d^\ell f_p$ is symmetric with respect to its arguments and its values can be represented by an $m \times \cdots \times m \times k$ array. Contraction of this array with respect to last dimension is denoted by $d^\ell f_p(\cdot,\ldots,\cdot)w$. The subscript is sometimes omitted for simplicity, if the point $p$ is clear from the context.

For example the second differential of $f$ is an array of dimensions $m \times m \times k$, and if $u, v \in \mathbb{R}^m$ and $w \in \mathbb{R}^k$, then $d^2 f(u,v)w = d^2 f(v,u)w \in \mathbb{R}$. Also $d^2 f(\cdot,\cdot)w$ is a symmetric $m \times m$ matrix while $d^2 f(u,\cdot)$ we choose to be the $k \times m$ matrix (rather than its transpose). This choice will be convenient in later formulas. However, a word of caution is in order here: now $d^2 f(u,\cdot)w$ is *not* "the matrix $d^2 f(u,\cdot)$ times vector $w$" but instead "the matrix $d^2 f(u,\cdot)^t$ times $w$".

### 2.2 *Riemannian geometry*

Let $M$ be a manifold. The tangent bundle of $M$ is denoted by $TM$, and the tangent space at $p \in M$ by $T_pM$. The space of vector fields on $M$ (i.e the space of sections of $TM$) is denoted by $\mathfrak{X}(M)$ and the value of $Y \in \mathfrak{X}(M)$ at $p$ is $Y_p$. A distribution $\mathscr{D}$ is a map that associates to each point $p \in M$ a subspace $\mathscr{D}_p$ of $T_pM$.

Let $M$ be a submanifold of $\mathbb{R}^n$. The objects defined on $M$ can be taken to be defined on $\mathbb{R}^n$ without writing explicitly the inclusion map. The inner product in $\mathbb{R}^n$ is denoted by $\langle\cdot,\cdot\rangle$ and the same notation will be used also for inner products in $T_pM$ and $T_p\mathbb{R}^n$ as usual. We may regard $T_pM$ as a subspace of $T_p\mathbb{R}^n$. The orthogonal complement of $T_pM$, the normal space, is denoted by $N_pM$. This induces the orthogonal projections $\pi_t : T_p\mathbb{R}^n \mapsto T_pM$ and $\pi_n : T_p\mathbb{R}^n \mapsto N_pM$.

Let $X, Y \in \mathfrak{X}(M)$. We may (locally) extend these to vector fields in some neighbourhood of $M$. Denoting these extensions by the same letter we may define the covariant derivative of $Y$ with respect to $X$ as follows:

$$\nabla_X Y = \pi_t(dY\,X) \tag{2.1}$$

where $dYX$ is the usual directional derivative. The value of $\nabla_X Y$ at $p$ does not depend on the derivatives of $X$, so we can define a linear map $T_pM \to T_pM$ by

$$X_p \mapsto \left(\nabla_{X_p} Y\right)_p. \tag{2.2}$$

Let $\gamma : \mathbb{R} \to M$ be a curve and let $X \in \mathfrak{X}(M)$ be such that $X_{\gamma(t)} = \gamma'(t)$. The curve $\gamma$ is a geodesic if $\nabla_X X = 0$ along $\gamma$. Let $Y \in \mathfrak{X}(M)$; vectors $Y_{\gamma(t)}$ are said to be parallel along $\gamma$, if $\nabla_X Y = 0$. Let $\gamma(0) = p$ and $v \in T_pM$. Then there is a unique family of $Y(t) \in T_{\gamma(t)}M$ such that $Y(t)$ are parallel along $\gamma$ and $Y(0) = v$. Hence we have a linear map, parallel translation $\tau_t : T_pM \to T_{\gamma(t)}M$ defined by $\tau_t(Y(0)) = Y(t)$.

## 2.3 *Some facts about block matrices*

Let us consider the $2 \times 2$ block matrix

$$C = \begin{pmatrix} A & B^t \\ B & 0 \end{pmatrix} \tag{2.3}$$

where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{\ell\times n}$, and we assume that $\ell \leqslant n$. In the case which interests us $A$ (and hence $C$) is symmetric, and $A$ and $B$ have full ranks. Let us further denote the Schur complement of $A$ by $S = -BA^{-1}B^t$ and recall that the inertia of the (symmetric) matrix is a triple $i(A) = \big(n_+(A), n_-(A), n_0(A)\big)$ where by $n_+(A)$ (resp. $n_-(A)$ and $n_0(A)$) is the number of positive (resp. negative and zero) eigenvalues of $A$. Now in (Horn & Johnson, 1994, p. 95) we find the following result:

LEMMA 2.1 Let $C$ be as in (2.3). Then $i(C) = i(A) + i(S)$ where $S$ is the Schur complement of $A$.

Suppose further that $A$ depends continuously on the parameter $h$ (the step size) and that $A = I$ when $h = 0$. It is easy to check that this, together with Lemma 2.1, implies

LEMMA 2.2 For sufficiently small $h$ we have the following properties:

- $A^{-1}$ exists and $A$ is positive definite
- $S$ is negative definite
- $n_+(C) = n$ and $n_-(C) = \ell$

Hence $A$ and $S$ tend to have eigenvalues of the same sign while typically $C$ has many eigenvalues of both signs. We have also the following result.

LEMMA 2.3 $n_-(S) \leqslant n_+(A)$ and $n_+(S) \leqslant n_-(A)$.

*Proof.* This is rather immediate consequence of the minimax principle for eigenvalues, see (Bhatia, 1997, p. 58). □

## 2.4 *Differential systems in jet spaces*

Here we simply give the basic definitions and refer to Tuomela & Arponen (2000) for a discussion and motivation of these concepts. Let $\pi : \mathscr{E} \to \mathscr{B}$ be a bundle and let $\pi^q : J_q(\mathscr{E}) \to \mathscr{B}$ be the bundle of $q$-jets of $\mathscr{E}$. Let us also introduce the standard projections

$$\pi_q^{q+r} : J_{q+r}(\mathscr{E}) \to J_q(\mathscr{E}).$$

In particular $\pi_0^q : J_q(\mathscr{E}) \to \mathscr{E}$.

DEFINITION 1 A (partial) differential system (or equation) of order $q$ on $\mathscr{E}$ is a submanifold $\mathscr{R}_q$ of $J_q(\mathscr{E})$.

In the sequel we will only consider the case where $\mathscr{E}$ is the trivial bundle $\mathscr{E} = \mathbb{R}\times\mathbb{R}^n$ with the projection $\pi : \mathbb{R}\times\mathbb{R}^n \to \mathbb{R}$. The coordinates of $J_q(\mathscr{E})$ (called jet coordinates) are denoted by $(x,y^1,\dots,y^n,y_1^1,\dots,y_q^n)$. The action of the projections in these coordinates is simply to "forget" the appropriate jets.

Above we defined what the differential equations are, but we have not yet introduced any equations. In jet coordinates the manifold $\mathscr{R}_q$ can be represented as a zero set of some map $f : J_q(\mathscr{E}) \simeq \mathbb{R}^{(q+1)n+1} \to \mathbb{R}^k$:

$$\mathscr{R}_q \quad : \quad f(x,y,y_1,\dots,y_q) = 0. \tag{2.4}$$

To define the notion of solution in our context we still need to introduce the following one forms

$$\alpha_j^i = dy_{j-1}^i - y_j^i dx \qquad i = 1,\dots,n \qquad j = 1,\dots,q. \tag{2.5}$$

Let $p \in J_q(\mathscr{E})$ and $v_p \in T_pJ_q(\mathscr{E})$ and let us further define distributions $\mathscr{C}$ and $\mathscr{D}$ by

$$\begin{aligned} \mathscr{C}_p &= \big\{ v_p \in T_pJ_q(\mathscr{E}) \,\big|\, \alpha_j^i(v_p) = 0 \big\}, \\ \mathscr{D}_p &= T_p\mathscr{R}_q \cap C_p, \end{aligned} \tag{2.6}$$

$\mathscr{C}$ is called *Cartan distribution*. Now we can define the solutions as follows.

DEFINITION 2 Let $\mathscr{R}_q \subset J_q(\mathscr{E})$ be involutive and suppose that the distribution $\mathscr{D}$ defined in (2.6) is one-dimensional. A solution of $\mathscr{R}_q$ is an integral manifold of $\mathscr{D}$.

The important concept of involutivity is discussed in Tuomela & Arponen (2000). Intuitively a system is involutive if we cannot get new equations of order $q$ or less by differentiating the equations and eliminating the higher derivatives. Our notion of solution extends somewhat the classical notion of solution. The connection can be explained as follows. Suppose that $S$ is a solution in our sense and suppose that $y : \mathbb{R} \to \mathbb{R}^n$ is a solution in a classical sense. Then $\pi_0^q(S)$ coincides locally with the graph of $y$. Note that our solution is always smooth, but the classical solution (and its graph) may have singularities.

Let us assume that $\mathscr{R}_q$ is involutive and the corresponding distribution $\mathscr{D}$ one-dimensional. Since one-dimensional distributions always have integral manifolds, there always exists a solution to our problem in these circumstances.

In the following we will discuss how to compute efficiently these solutions.

## 3. Precise Formulation of the computational problem

We will use explicit Runge–Kutta type methods to compute the solution. Let us briefly outline how this is done. We will consider three separate cases:

- The *general case*: we use directly the formulation given in (2.4).
- *Systems with invariants*.
- *Holonomic systems*.

It is worthwhile to consider in detail the latter two cases because they occur very often in applications and it turns out that taking into account the special features of these systems yields a much more efficient implementation of the computational problem. Probably most problems in applications belong to these categories, so that the general case which is of course important for completeness, is perhaps not so interesting in practice.

In all three cases the computational problem will be as follows:

- Given a manifold $M$, a one dimensional distribution $\mathscr{D}$ on $M$ and a point $p \in M$ we want to compute (a part of) the integral manifold that passes through $p$

The information about $M$ and $\mathscr{D}$ is given in the following form:

- $M$ is a submanifold of $\mathbb{R}^m$ for some $m$, given as a zero set of some map $f$.
- The distribution $\mathscr{D}_p$ can be computed at each $p \in M$ either by computing the nullspace of some matrix or by solving a square linear system. In some cases we can even write down explicitly the vector $v_p$ which spans $\mathscr{D}_p$.

Let us denote by $V$ the vector field associated to $\mathscr{D}$; i.e. $|V_p| = 1$ and $V_p \in \mathscr{D}_p$.[1] We will use Runge–Kutta type methods as in Tuomela & Arponen (2000) and Tuomela & Arponen (2001). The general explicit $r$–stage Runga–Kutta method can be formulated in our context as follows. Let $\pi_M$ be the orthogonal projection map onto $M$. This map is well-defined in sufficiently small tubular neighbourhood of $M$ which is sufficient for our purposes. Let $\mathfrak{p} \in M$ be the current point, $h$ the current step size, $\mathsf{p}^i$ the intermediate points used by the method, and $\mathsf{p}$ the point produced by the scheme. The current point is also interpreted as the first intermediate point: $\mathfrak{p} = \mathsf{p}^1$. Then we can write

$$\begin{aligned}
\mathsf{p}^2 &= \pi_M\Big(\mathfrak{p} + h a_{21} V_{\mathfrak{p}}\Big)\\
\mathsf{p}^3 &= \pi_M\Big(\mathfrak{p} + h\big(a_{31} V_{\mathfrak{p}} + a_{32} V_{\mathsf{p}^2}\big)\Big)\\
&\ \vdots\\
\mathsf{p}^r &= \pi_M\Big(\mathfrak{p} + h\sum_{i=1}^{r-1} a_{ri} V_{\mathsf{p}^i}\Big)\\
\mathsf{p} &= \pi_M\Big(\mathfrak{p} + h\sum_{i=1}^{r} b_i V_{\mathsf{p}^i}\Big).
\end{aligned} \tag{3.1}$$

[1]The associated vector field exists locally, but not necessarily globally. However, in the present context we only need the local existence.

The additions in the right hand side are defined by the fact that $M$ is a submanifold of $\mathbb{R}^m$. The coefficients $a_{ij}$ and $b_i$ define the particular method. Note that in standard Runge–Kutta schemes there is also a third set of coefficients (usually denoted by $c_i$) which are needed if the relevant vector field depends explicitly on time. However, although our formulation of course covers also the time dependent case, we do not need this set of coefficients in our scheme.

So to implement the scheme we have to solve two basic subproblems:

- given a point $a \in \mathbb{R}^m$, project it orthogonally to $M$, i.e. compute $\pi_M(a)$.
- given $p \in M$, compute $\mathscr{D}_p$.

We will consider these problems in each of the three cases discussed above.

## 3.1 *General systems*

Let us suppose that $k < m = (q+1)n+1$ and consider the map $f : J_q(\mathscr{E}) \simeq \mathbb{R}^m \mapsto \mathbb{R}^k$ as in (2.4). We assume that zero is a regular value of $f$ and that $f$ is involutive. Then we set

$$M = \mathscr{R}_q = f^{-1}(0) \subset \mathbb{R}^m$$

This implies that $M$ is a smooth $m-k$ – dimensional manifold, and that the columns of $(df_p)^t$ span $N_pM$.

Now given some $a \in \mathbb{R}^m$ and denoting $p = \pi_M(a)$, we can try to compute $p$ by solving the following problem:

$$\begin{cases} p + (df_p)^t \mu - a = 0 \\ f(p) = 0. \end{cases} \tag{3.2}$$

If $a$ is sufficiently close to $M$ this problem has a unique solution. In our application this can always be achieved by taking the step-size $h$ sufficiently small, because by (3.1) we always have $a = \mathfrak{p} + O(h)$.

Next we have to compute the distribution defined in (2.6), i.e. we have to find $v \in \mathscr{D}_p$ such that $|v| = 1$. Recall that the coordinates of $\mathbb{R}^m$ are denoted $(x, y, y_1, \dots, y_q)$. Given $p \in M$ the distribution $\mathscr{D}_p$ can in these coordinates be represented as a nullspace of the following matrix:

$$\mathsf{D} = \begin{pmatrix} -\tilde{y} & I_{nq} & 0_{nq\times n} \\ f_x & D_1 & D_2 \end{pmatrix} \tag{3.3}$$

where $f_x = \partial f/\partial x$, $D_2 = \partial f/\partial y_q$, $D_1$ contains the partial derivatives of $f$ with respect to $y, y_1, \dots, y_{q-1}$ and finally $\tilde{y} \in \mathbb{R}^{nq}$ contains the vectors $y_1, y_2, \dots, y_q$. Note that the first row of $\mathsf{D}$ comes from Cartan distribution (jet coordinates correspond to derivatives or in other words $\mathscr{D}_p \subset \mathscr{C}_p$) and the second row contains the tangent conditions ($\mathscr{D}_p \subset T_pM$).

The computation of the nullspace of $\mathsf{D}$ can further be reduced to the computation of the nullspace of the following $k \times (n+1)$-matrix:

$$D = \begin{pmatrix} f_x + D_1\tilde{y} & D_2 \end{pmatrix}. \tag{3.4}$$

This can be seen as follows: let $D\tilde{v} = 0$ where $\tilde{v} = (v_1, \hat{v})$. Let us set $v = (v_1,\ v_1\tilde{y},\ \hat{v})$. Then it is easy to check that $\mathsf{D}v = 0$. Note further that the dimensions of $D$ are independent of $q$.

## 3.2 *Systems with invariants*

Consider the system (2.4). In the present context we mean by an invariant an equation of (2.4) which does not depend on $y_q$. For simplicity we now restrict our attention to quasilinear systems; i.e. we will consider systems of the form

$$\mathscr{R}_q \quad : \quad \begin{cases} Ay_q + \tilde{f}(x,y,\ldots,y_{q-1}) = 0 \\ f(x,y,\ldots,y_{q-1}) = 0 \end{cases} \tag{3.5}$$

where $A$ is some nonsingular matrix which does not depend on $y_q$. In the examples below the systems with invariants are of this form. In fact in all but one case $A$ is the identity matrix.

Now if the system (3.5) is involutive, then the solutions are well defined as curves in $\mathscr{R}_q \subset J_q(\mathscr{E})$. However, we can simplify the computations considerably by projecting these solutions to $J_{q-1}(\mathscr{E})$. So here let us denote

$$\pi^q_{q-1}(\mathscr{R}_q) = M = f^{-1}(0) \subset J_{q-1}(\mathscr{E})$$

and suppose that zero is a regular value of $f$. But now we see that the projection step is in this case formally precisely the same as in the general case: we have to solve a system of the form (3.2). However, the meaning of $f$ and $M$ is different in the two cases.

Of course here $f$ by itself does not define solutions uniquely; we need also to project the distribution. Geometrically this is given by $d\pi^q_{q-1}$, but in practice it can be computed as follows:

- given $p \in M \subset J_{q-1}(\mathscr{E})$, solve $y_q$ using the equation

$$Ay_q + \tilde{f}(x,y,\ldots,y_{q-1}) = 0 \tag{3.6}$$

- set $\mathscr{D}_p = \operatorname{span}\{(1,y_1,\ldots,y_q)\}$.

If $A$ is in fact the identity matrix the distribution is defined by

$$\mathscr{D}_p = \operatorname{span}\{(1,y_1,\ldots,y_{q-1},-\tilde{f})\}.$$

Note that in the standard literature the first row of (3.5) is usually called *system* or *equations* and the second row is *constraints*. In our formulation we may say that “constraints” determine where the solutions “really” are while the “equations” give “only” the distribution.

As a special case of this procedure we get the standard way of viewing ODEs; namely given a system

$$y_1 + f(x,y) = 0$$

we usually do not consider it in $J_1(\mathscr{E})$, but instead consider it as a vector field in $\mathbb{R}^n$ (if $f$ does not depend on $x$) or as a distribution in $\mathscr{E}$ ($\simeq$ time dependent vector field in $\mathbb{R}^n$).

Of course it may be possible in some cases to reduce the problem which is initially given in $J_q(\mathscr{E})$ to a problem in $J_{q-\iota}(\mathscr{E})$ for some $\iota > 1$. We will not consider explicitly this case because we will not need it in the examples and because it is quite analogous to the case where $\iota = 1$.

## 3.3 *Holonomic systems*

The mechanical systems with holonomic constraints can be written in the following form:

$$\begin{cases} E(y,y_1)y_2 + K(y,y_1) + (dg)^t\lambda = 0 \\ g(y) = 0 \\ dg\,y_1 = 0. \end{cases} \tag{3.7}$$

Here $E$ is a positive definite matrix (mass matrix) which is often constant and diagonal, $K$ models the forces acting on the system, $g$ is some smooth map $g : \mathbb{R}^n \to \mathbb{R}^l$, and $\lambda : \mathbb{R} \to \mathbb{R}^l$ is the Lagrange multiplier. Now it turns out that these problems can be formulated in such a way that only the last two equations in (3.7) are needed to define the relevant manifold, and the first equation is only needed to compute the distribution. This is explained in detail in Tuomela & Arponen (2000) so we do not repeat the discussion here.

However, in many cases in addition to "constraints", given by the map $g$ there are additional invariants or constants of motions in the system which are not integrability conditions and consequently cannot be found by passing to the involutive form. Note also that there is no Lagrange multiplier associated to these invariants. A typical invariant of this type is the conservation of energy. These invariants can be expressed by a map $\tilde{g} : \mathbb{R}^{2n} \to \mathbb{R}^r$ and hence in presence of invariants the map $f$ in (3.2) is given by

$$f(y,y_1) = \begin{pmatrix} dg_y y_1 \\ g(y) \\ \tilde{g}(y,y_1) \end{pmatrix} \qquad \text{where} \quad f : \mathbb{R}^{2n} \to \mathbb{R}^{2l+r} \quad \text{and} \quad M = f^{-1}(0) \subset \mathbb{R}^{2n}. \tag{3.8}$$

We will suppose that zero is a regular value of $g$ and $f$; hence $N = g^{-1}(0) \subset \mathbb{R}^n$ is a smooth manifold and we may say that the first two equations of the above system define the tangent bundle $TN$. In this way $M$ is a submanifold (but in general not a subbundle) of $TN$. Then taking into account the time variable we have

$$\mathbb{R} \times M \subset \mathbb{R} \times TN \subset J_1(\mathscr{E})$$

So as in the case of systems with invariants, the relevant manifold is defined by "constraints", and now we have to use the "equations", i.e. the first row of (3.7) to compute the distribution. As explained in Tuomela & Arponen (2000) this is done as follows:

- given $p = (x,y,y_1) \in \mathbb{R} \times M$, solve the linear system

$$\begin{pmatrix} E & (dg)^t \\ dg & 0 \end{pmatrix} \begin{pmatrix} y_2 \\ \lambda \end{pmatrix} = -\begin{pmatrix} K \\ d^2g\,(y_1,y_1) \end{pmatrix} \tag{3.9}$$

- set $\mathscr{D}_p = \operatorname{span}(1,y_1,y_2)$.

Note that the addition of invariants (the map $\tilde{g}$ in (3.8)) has no effect on the computation of the distribution. Note also that in case of systems with invariants as well as in the holonomic case the overdetermined nature of the problem has completely disappeared, because the computation of the distribution leads to a square system.

So all in all there are following subproblems we have to treat:

## Projection, i.e. solution of the equation (3.2).

- In the general case $f$ is given by (2.4).
- In case of system with invariants $f$ is given by (3.5).
- In the holonomic case $f$ is given by (3.8).

Note that in first two cases $f$ is more or less arbitrary, i.e. we do not assume any special property of $f$. However, geometrically $f$ represents two very different things. In the third case we can exploit the special structure of $f$ to speed up the computations.

## Distribution.

- Compute the nullspace of (3.4) in the general case.
- Solve the system (3.6) in case of system with invariants.
- Solve the system (3.9) in the holonomic case.

We will consider all these cases in detail below.

## 4. Projection

### 4.1 *Newton type methods*

In the projection step we need to find a solution to some nonlinear system of equations. Let $F : \mathbb{R}^{\nu} \to \mathbb{R}^{\nu}$ be some map and suppose that we want to solve $F(z) = 0$. Let us further suppose that our system has a solution which is denoted by $z_*$ and that $dF_* = dF(z_*)$ has full rank. Then choosing the initial guess $z^0$ sufficiently close to $z_*$ we can compute the solution by Newton's method:

Algorithm 4.1

```
• choose $z^0$
• for $k = 0,\ 1,\ \ldots$ do until convergence
    – solve $dF_k s^k = -F_k$ where $dF_k = dF_{z^k}$ and $F_k = F(z^k)$
    – set $z^{k+1} = z^k + s^k$
```

Now in case of big systems the computation of the jacobian of $F$ may not be easy to implement. However, in the present paper we will suppose that various differentials which are needed are computed symbolically. There are some alternative strategies which may be more efficient in some cases, but we leave the consideration of these methodes to future papers. However, we will make some remarks in conclusion.

The standard way to solve the linear system in Newton iteration is by LU–decomposition. This is always a reasonable choice, if $\nu$ is not too big and $dF$ is dense. However, typically in large systems the jacobian is increasingly sparse and in that case iterative methods become competitive. Moreover one may speculate that since $s^k$'s are "only" search directions, perhaps there is no need to solve them very accurately, provided that the overall iteration converges. This leads to the *inexact Newton method* Dembo *et al.* (1982). This can be formulated as follows:

Algorithm 4.2

```
• choose $z^0$
• for $k = 0,\ 1,\ \ldots$ do until convergence
    – choose $\varepsilon_k \in (0,1)$
    – find $s^k$ such that $dF_k s^k = -F_k + r^k$ with $|r^k| < \varepsilon_k |F_k|$
    – set $z^{k+1} = z^k + s^k$
```

Here $|\cdot|$ can be any convenient norm on $\mathbb{R}^\nu$. Let us further define an associated norm $|s|_* = |dF_* s|$.[2] The following Theorem guarantees that the above algorithm indeed converges Dembo *et al.* (1982).

THEOREM 4.1 Let us assume that $\varepsilon_k \leqslant \varepsilon < c < 1$ for all $k$. If $z^0$ is sufficiently close to $z_*$, then $z^k \to z_*$ as $k \to \infty$ and we have the estimate

$$|z^{k+1} - z_*|_* \leqslant c|z^k - z_*|_*$$

Moreover if $\varepsilon_k \to 0$, then the convergence is superlinear.

Note that the conditions for the convergence are not very strict. To apply the above idea we must now discuss how to choose an appropriate iterative method.

### 4.2 *Iterative methods for linear systems*

Consider the system (3.2). It will be convenient to set $z = (p, \mu)$ and write our system as

$$F(z) = \begin{pmatrix} p + (df_p)^t \mu - a \\ f(p) \end{pmatrix} = 0. \tag{4.1}$$

Recalling that $a = \mathfrak{p} + O(h)$ it is straightforward to verify that for sufficiently small step size we have:

(i) there is a solution to our problem, which will be denoted by $z_* = (\mathsf{p}, \mu_*)$.

(ii) $dF_*$ cannot be singular.

Properties (i) and (ii), and Theorem 4.1 imply that if $z^0$ is sufficiently close to $z_*$, the iteration will converge to $z_*$. The successive iterates are denoted by $z^k = (p^k, \mu^k)$.

In order to choose an iterative method let us analyse the structure of $dF$ more closely. By simple computation we obtain

$$dF = \begin{pmatrix} I + d^2 f(\cdot,\cdot)\mu & (df)^t \\ df & 0 \end{pmatrix}. \tag{4.2}$$

Now obviously $dF$ is symmetric, hence of the form (2.3). By Lemma 2.2 we conclude that $dF$ has typically many positive and negative eigenvalues. This means that conjugate gradient method cannot be used and moreover this implies slow convergence even for such methods which are suitable for indefinite problems like GMRES Trefethen & Bau (1997). To get a faster scheme we will exploit the block structure of $dF$. Let us denote

$$dF = C = \begin{pmatrix} I + d^2 f(\cdot,\cdot)\mu & (df)^t \\ df & 0 \end{pmatrix} = \begin{pmatrix} A & B^t \\ B & 0 \end{pmatrix}.$$

We would like to solve

$$Cs = b \quad \Leftrightarrow \quad \begin{pmatrix} A & B^t \\ B & 0 \end{pmatrix} \begin{pmatrix} s^1 \\ s^2 \end{pmatrix} = \begin{pmatrix} b^1 \\ b^2 \end{pmatrix}. \tag{4.3}$$

It is straightforward to verify the following formula

$$C^{-1} = \begin{pmatrix} A^{-1} & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} A^{-1}B^t \\ -I \end{pmatrix} S^{-1} \begin{pmatrix} BA^{-1} & -I \end{pmatrix}$$

[2]This defines a norm because by hypothesis $dF_*$ is of full rank.

where $S = -BA^{-1}B^t$ is the Schur complement of $A$. In Miao (1991) more general formulas for (pseudo)inverses of block matrices are given. We can now write an algorithm for the solution of $Cs = b$ based on the above formula.

Algorithm 4.3
INPUT: matrix $C$, vector $b$ partitioned as in (4.3)
OUTPUT: vector $s$ partitioned as in (4.3)

- solve $Au^1 = b^1$
- compute $v^1 = Bu^1 - b^2$
- solve $Sv^2 = v^1$.
- compute $u^2 = B^t v^2$
- solve $Au^3 = u^2$
- set $s = \left(u^1 + u^3, -v^2\right)$

In the process we have to solve systems $Au^1 = b^1$, $Sv^2 = v^1$ and $Au^3 = u^2$ in some way. Note that $A$ (resp. $S$) is symmetric but not positive (resp. negative) definite in general. However, Lemmas 2.1 and 2.3 imply that $A$ (resp. $S$) has only few negative (resp. positive) eigenvalues. Moreover eigenvalues of $A$ are very clustered, because $A$ approaches the identity matrix when the step size goes to zero. This situation is very favorable for iterative methods. Hence iterating $A$ systems to convergence is cheap. However, the convergence of $S$ systems is not usually very fast. So we will allow a relatively large error in the solution of $S$ systems.

Using the notation $r_s = v^1 - Sv^2$ and considering the residuals in $A$ systems as negligible a simple calculation shows that

$$r = b - Cs = \begin{pmatrix} 0 \\ -r_s \end{pmatrix}.$$

Thus in particular $|r| = |r_s|$. This leads to

Algorithm 4.4
INPUT: matrix $C$, vector $b$ partitioned as in (4.3); tolerance $\varepsilon \in (0,1)$
OUTPUT: vector $s$ partitioned as in (4.3)

- solve $Au^1 = b^1$ iteratively until convergence
- compute $v^1 = Bu^1 - b^2$
- solve $Sv^2 = v^1$ iteratively until $|r_s| < \varepsilon |b|$
- compute $u^2 = B^t v^2$
- solve $Au^3 = u^2$ iteratively until convergence
- set $s = \left(u^1 + u^3, -v^2\right)$

When solving the system $Sv^2 = v^1$ we have to evaluate products $Sv$. This is done with the following algorithm.

```
Algorithm 4.5
INPUT: matrices $A$ and $B$, vector $v$
OUTPUT: vector $d = Sv$
  • compute $w^1 = B^t v$
  • solve $Aw^2 = w^1$ iteratively until convergence
  • compute $d = -Bw^2$
```

It remains to choose the iterative method. Some natural candidates are GMRES, as previously mentioned, and BiCG (bi-conjugate gradient). In initial tests GMRES did not turn out to be very efficient and the problem with BiCG is its irregular convergence. One method which combines efficiency with regularity is symmetric LQ method Fischer (1996), SymmLQ. This method was used in test cases below.

Let us then analyse the choice of the initial point $z^0$. As before let us denote by $\mathfrak{p}$ the current point, and let us set $\mathsf{p} = \pi_M(a)$ where $a$ is as in the right hand side of (3.1). One possibility is simply to take $z^0 = (\mathfrak{p}, 0)$. With this choice the algorithm will converge, if the step size is sufficiently small. However, the point $z^0 = (a, 0)$ is better for two reasons: first $|\mathsf{p} - \mathfrak{p}| = O(h)$ while $|\mathsf{p} - a| = O(h^2)$. Moreover, computing the first step in the Newton iteration with $z^0 = (\mathfrak{p}, 0)$ shows that $z^1 \approx (a, \mu^1)$. In other words the first iteration is waste of time and we may as well start directly with $z^0 = (a, 0)$. Moreover using this initial point the first step of the (inexact) Newton iteration is much simpler than others, so it is reasonable to initialize the computations in the following way. Here $r_\mu = f(a) - \Big(df_a(df_a)^t\Big)\mu^1$.

```
Algorithm 4.6 Initialization (first Newton step)
INPUT: vector $a$, tolerance $\varepsilon \in (0,1)$
OUTPUT: $z^1 = (\mathsf{p}^1, \mu^1)$
  • solve $\Big(df_a(df_a)^t\Big)\mu^1 = f(a)$ iteratively until $|r_\mu| < \varepsilon |f(a)|$
  • compute $\mathsf{p}^1 = a - (df_a)^t \mu^1$
  • set $z^1 = (\mathsf{p}^1, \mu^1)$
```

Note that here we are solving a positive definite system because we suppose that $df$ is of full rank, so we can use the conjugate gradient method. Now the overall inexact Newton method can be formulated as follows:

```
Algorithm 4.7
  • compute $z^1$ with Algorithm 4.6
  • for $k = 1, 2, \dots$ do until convergence
      – choose $\varepsilon_k \in (0,1)$
      – solve $dF_k s^k = -F_k$ approximately with Algorithm 4.4
      – set $z^{k+1} = z^k + s^k$.
```

## 4.3 *Holonomic case*

In this case $f$ in (4.1) is given by (3.8). It will be convenient to introduce the following notation. Recall that the map $\tilde{g}$ in (3.8) depends on $y$ and $y_1$. The differential of $\tilde{g}$ with respect to $y$ (resp. $y_1$) is denoted by $d_0\tilde{g}$ (resp. $d_1\tilde{g}$). So we can write $d\tilde{g} = \left(d_0\tilde{g}, d_1\tilde{g}\right)$. Similar notation is also used for higher order differentials.

Let us now compute the jacobian of (4.1) when $f$ is given by (3.8). It is useful first to compute $df$; this gives

$$df = \begin{pmatrix} d^2g(y_1,\cdot) & dg \\ dg & 0 \\ d_0\tilde{g} & d_1\tilde{g} \end{pmatrix}.$$

Then putting $a = (a^1, a^2)$ and $\mu = (\alpha, \beta, \gamma) \in \mathbb{R}^{2l+r}$ the map $F$ in (4.1) can be written as[3]

$$F(z) = \begin{pmatrix} y + d^2g^t(y_1,\cdot)\alpha + dg^t\beta + d_0\tilde{g}^t\gamma - a^1 \\ y_1 + dg^t\alpha + d_1\tilde{g}^t\gamma - a^2 \\ dgy_1 \\ g(y) \\ \tilde{g}(y,y_1) \end{pmatrix}. \tag{4.4}$$

Computing further we get

$$dF = \begin{pmatrix} I + \mathscr{A} & (df)^t \\ df & 0 \end{pmatrix} \tag{4.5}$$

where $\mathscr{A}$ is a block matrix given by

$$\mathscr{A} = \begin{pmatrix} d^3g(y_1,\cdot,\cdot)\alpha + d^2g(\cdot,\cdot)\beta + d_0^2\tilde{g}(\cdot,\cdot)\gamma & d^2g(\cdot,\cdot)\alpha + d_0d_1\tilde{g}(\cdot,\cdot)\gamma \\ d^2g(\cdot,\cdot)\alpha + d_0d_1\tilde{g}(\cdot,\cdot)\gamma & d_1^2\tilde{g}(\cdot,\cdot)\gamma \end{pmatrix}.$$

Now denoting

$$\mathscr{A} = \begin{pmatrix} A_1 & A_2 \\ A_2 & A_3 \end{pmatrix}$$

we observe that $A_1$, $A_2$ and $A_3$ are symmetric which implies that $dF$ is symmetric and positive definite for small enough $h$.

Note that in some cases of practical interest $g$ (or a part of $g$) is a second order polynomial which implies that $d^3g = 0$ and $d^2g$ is constant. Also if the only invariant is the energy, then in many cases we have

$$\tilde{g}(y,y_1) = \tfrac{1}{2}\langle y_1, By_1\rangle + U(y)$$

where $B$ is some symmetric positive definite matrix which does not depend on $y$ and $U$ is the potential energy which does not depend on $y_1$. Hence in this case we obtain

$$\mathscr{A} = \begin{pmatrix} d^3g(y_1,\cdot,\cdot)\alpha + d^2g(\cdot,\cdot)\beta + \gamma d^2U & d^2g(\cdot,\cdot)\alpha \\ d^2g(\cdot,\cdot)\alpha & \gamma B \end{pmatrix}.$$

[3]Strictly speaking we do not include the equation for $x$ because the system does not depend explicitly on $x$.

## 5. Distribution

### 5.1 *General case*

Here we have to compute the kernel of the matrix given in (3.4). Now in many cases of practical interest it is known that $D_2$ is nonsingular, and moreover it may well contain a (constant) square submatrix (or even identity matrix) of full rank. In this case the computation of the distribution is easy:

Algorithm 5.1

- compute $b = f_x + D_1\tilde{y}$
- choose the relevant square submatrix $\hat{D}_2$ of $D_2$
- choose the corresponding components $\hat{b}$ of $b$
- solve $\hat{D}_2\hat{v} = -\hat{b}$
- set $v = (1, \tilde{y}, \hat{v})$
- normalise $v = v/|v|$

For example if the relevant $\hat{D}_2$ is constant then obviously this is very efficient.

To compute $\tilde{v}$ in the general case we can use the singular value decomposition. If $D = U\Sigma V^t$, then the last singular value should be very close to zero, and hence the last column of $V$ gives a good approximation of the required direction. This is a reliable way to compute the distribution, but since we need only one vector of the whole decomposition, the computation is unnecessarily costly for large systems.

One possible method to compute only the required vector is as follows. Let us be given a matrix $D \in \mathbb{R}^{k\times(n+1)}$ and let $D = U\Sigma V^t$ be its singular value decomposition. Suppose that $k \geqslant n$ and that all the singular values are distinct (the generic case). To avoid some notational nuisance we make the following convention: when $k = n$ we call $\sigma_{n+1} = 0$ the smallest singular value although strictly speaking there are only $n$ singular values in that case. We order the singular values in the usual way: $\sigma_1 \geqslant \cdots \geqslant \sigma_{n+1} \geqslant 0$ (and $\sigma_{n+1} = 0$ when $k = n$). The corresponding right singular vectors (columns of $V$) are denoted by $v^i$.

Let $S^n$ be the $n$ – dimensional unit sphere and define $\mathsf{f} : S^n \to \mathbb{R}$ by $\mathsf{f}(x) = \frac{1}{2}|Dx|^2$. The gradient of $\mathsf{f}$ induces a vector field on $S^n$ by $Y(x) = -\pi_t(\nabla \mathsf{f}(x))$. More explicitly we have

$$Y(x) = |Dx|^2 x - D^t Dx. \tag{5.1}$$

The zeros of $Y$ are clearly the right singular vectors $v^i$ of $D$. Let us then analyse the stability of these zeros. Taking $w \in T_x S^n$ we compute

$$\nabla_w Y = \pi_t\big(dY\, w\big) = |Dx|^2 w - D^t Dw + \langle Dx, Dw\rangle x.$$

The above formula then defines a linear map $T_x S^n \to T_x S^n$ as in (2.2). Let us call this map $L_x$. Then we obtain

$$L_{v^i} w = \sigma_i^2 w - D^t Dw.$$

Now $T_{v^i} S^n$ is spanned by other singular vectors $v^k$, $k \neq i$, which implies that the eigenvalues of $L_{v^i}$ are $\sigma_i^2 - \sigma_k^2$, $k \neq i$. Hence the zero of $g$ corresponding to the largest (resp. smallest) singular value is a source (resp. sink) and other zeros are saddles. This is because eigenvalues of $L_{v^1}$ (resp. $L_{v^{n+1}}$) are all positive (resp. negative) and in other cases $L_{v^i}$ has positive as well as negative eigenvalues.

In other words the global minimum (resp. maximum) of f is attained at $v^{n+1}$ (resp. $v^1$). Now we may use a minimization algorithm given in Smith (1994) to find $v^{n+1}$. Recall that when one uses various optimization algorithms in $\mathbb{R}^n$, there are usually subproblems involving looking for an extremum along given line, i.e. along a geodesic of $\mathbb{R}^n$. Hence one gets an optimization algorithm on a Riemannian manifold by replacing lines by geodesics, and doing parallel translation along geodesics. Now in case of $S^n$ geodesics and parallel translations are easy to compute and we have then the following algorithm.

```
Algorithm 5.2
INPUT: matrix $D$, initial guess $u^0$ (previously computed $\tilde{v}$)
OUTPUT: vector $\tilde{v}$

• set $G^0 = H^0 = Y(u^0)$, $h^0 = H^0/|H^0|$, $i = 0$
• while not converged do
    – set $a = 2\langle Du^i, Dh^i\rangle$, $b = |Du^i|^2 - |Dh^i|^2$, $r = \sqrt{a^2+b^2}$
    – if $b \geqslant 0$ then
          set $s = \sqrt{(1+b/r)/2}$, $c = -a/(2rs)$
      else
          set $c = \sqrt{(1-b/r)/2}$, $s = -a/(2rc)$
      endif
    – $u^{i+1} = cu^i + sh^i$ [4]
    – $\tau H^i = |H^i|(ch^i - su^i)$
    – $\tau G^i = G^i - \langle h^i, G^i\rangle(su^i + (1-c)h^i)$
    – $G^{i+1} = Y(u^{i+1})$
    – $\gamma = \langle G^{i+1}, G^{i+1} - \tau G^i\rangle / \langle H^i, G^i\rangle$
    – $H^{i+1} = G^{i+1} + \gamma\tau H^i$, $h^{i+1} = H^{i+1}/|H^{i+1}|$
    – if $i \equiv n \mod n+1$ then
          set $H^{i+1} = G^{i+1}$
      endif
    – set $i = i+1$
  endwhile
• set $\tilde{v} = u^i$
```

[4]Note that $|u^{i+1}|$ is mathematically exactly one after computing $u^{i+1} = cu^i + sh^i$. However, numerically it is best to normalize by $u^{n+1} := u^{n+1}/|u^{n+1}|$.

In practice the above algorithm converges linearly, though much faster than the method of steepest descent which also converges linearly. We may expect slow convergence if the condition number of $D$ is large; recall that this is defined as $\kappa(D) = \sigma_1/\sigma_{n+1}$. Now our problem evolves on $S^n$, and moreover only the neighbourhood of $v^{n+1}$ is of practical interest. Hence we may expect that a reasonable measure of the numerical difficulty of the problem would be

$$\kappa(L_{v^{n+1}}) = \frac{\sigma_1^2 - \sigma_{n+1}^2}{\sigma_n^2 - \sigma_{n+1}^2}.$$

In the case which interests us there is a significant gap between $\sigma_n$ and $\sigma_{n+1}$, so we have in fact $\kappa(L_{v^{n+1}}) \approx \sigma_1^2/\sigma_n^2$.

## 5.2 *Systems with invariants*

Here we have to solve the system (3.6). The most natural choice is to use LU–decomposition. This was used in example *two phase plug flow* below. In all other examples of this form the relevant matrix $A$ was in fact an identity matrix, so the whole computation is immediate. Of course in general there could be situations where the use of some other method than LU–decomposition could be more efficient. However, the computational problem is in any case conceptually straightforward, and the choice of the method should not be difficult.

## 5.3 *Holonomic case*

To compute the distribution in this case we have to solve the system (3.9). This is of the same form as (4.3). However, here the situation is even better because now $E$ is always positive definite, and in many cases it is even a diagonal matrix. Moreover we have now a good initial guess for the solution, namely the solution computed in the previous step which we denote by $v_p$. Hence we will compute the distribution as follows:

```
Algorithm 5.3
INPUT: matrix $C$, vector $b$ partitioned as in (3.9); previous solution $\mathfrak{s} = (y_2, \lambda)$
OUTPUT: vector $v$

  • set $r = b - C\mathfrak{s}$
  • solve $Cs = r$ with Algorithm 4.3 using conjugate gradient method and iterating all systems
    to convergence
  • set $\mathfrak{s} = \mathfrak{s} + s$
  • set $v = (1, y_1, y_2)$
  • normalize $v = v/|v|$
```

Of course if $E$ is a diagonal matrix, the computation simplifies in an obvious way.

## 6. numerical results

In this section, we first discuss implementation issues, then present the examples, and finally the actual results. The main thing to test is the efficiency of our block-solver with inexact Newton (Algorithm 4.4) compared to exact Newton, as well as the effect of initialization (Algorithm 4.6). We demonstrate these with small and moderate size systems. In all of the runs we tried a few different initial conditions, the tables represent typical results.

### 6.1 *Implementation*

We have seen above that we could use any (explicit) Runge–Kutta scheme in our computations. We chose a well-known scheme by Dormand and Prince which is a 7 stage 5th order method which contains also 4th order method which is used in the step size control Hairer *et al.* (1993). The scheme will be abbreviated as dopri54. For comparison we have also used our variant of Euler's method as in Tuomela & Arponen (2000), denoted Euler in the tables.

In the step size control we used essentially the same strategy as explained in Hairer *et al.* (1993). However, we needed to modify this a little because in our case it may happen that the step size is quite big and so the projection step needs a lot of iterations. In inexact Newton method we used the following rule for $\varepsilon_k$:

$$\varepsilon_{k+1} = 0.8 \cdot \varepsilon_k.$$

The code was written in C++ and the simulations were done with a 1GHz PC using Linux operating system. All matrix/tensor operations were also implemented using the sparse data structure. In small problems this is not essential, but in big problems various differentials tend to be quite sparse, so we can save a substantial amount of computing time by exploiting sparsity.

The stopping criterion for the Newton methods was chosen as a hybrid absolute-relative:

$$|z^k - z^{k-1}| + |F_k| \ < \ \mathsf{tol}_{\mathsf{abs}} + \mathsf{tol}_{\mathsf{rel}}/|F_{k-1}|,$$

where $\mathsf{tol}_{\mathsf{abs}} = 10^{-6}$ and $\mathsf{tol}_{\mathsf{rel}} = 10^{-10}$. The tolerance for solving $A$-systems in Algorithm 4.4 was $10^{-12}$ in all tests. In inexact case we chose $\varepsilon_0 = 0.5$ if not mentioned separately. In the case of exact Newton, $\varepsilon_0$ for the conjugate gradient was chosen as 0.01.

### 6.2 *Descriptions of the examples*

Here only the systems are presented, the actual computational results are in the next subsection.

6.2.1 *Rigid body with dissipation* This is a spinning rigid body with dissipative term. The equations for the angular velocity $y = (y^1, y^2, y^3)$ are

$$y_1 - y \times \nabla H - \alpha y \times (y \times \nabla H) = 0 \tag{6.1}$$

where $\times$ is the usual cross-product of $\mathbb{R}^3$ and

$$H = \sum_{i=1}^{3} \frac{(y^i)^2}{2I_i}$$

with $I_i$ the constant moments of inertia. If $\alpha = 0$ this becomes the well-known Euler equations for rigid body with $H$ its hamiltonian function, see e.g. Olver (1986). Now $|y|$ is a constant of motion. Indeed,

$$\tfrac{1}{2}\tfrac{d}{dx}|y|^2 = \langle y, y_1\rangle = \langle y, y \times \nabla H\rangle + \alpha\langle y, y \times (y \times \nabla H)\rangle = 0. \tag{6.2}$$

Hence we can write our system as

$$\begin{cases} y_1 - y \times \nabla H - \alpha y \times (y \times \nabla H) = 0 \\ |y|^2 - a = 0 \end{cases} \tag{6.3}$$

where the constant $a$ is determined by initial conditions. This is of the form (3.5), and the relevant manifold $M$ is simply the two dimensional sphere of radius $\sqrt{a}$. But this implies that the projection step can be done just by scaling which yields very fast computations. Of course the standard (in)exact Newton iteration will also converge very quickly so we did not modify the code to solve this problem optimally.

6.2.2 *Two phase plug flow problem* This flow problem is taken from Hairer *et al.* (1989). The initial system is given by:

$$\begin{cases} u_1^1 + (u^2)^2 = 0 \\ u^2(R-u^3)^2\Big(2.5\ln\big(c_1u^2u^3-5\big)+10.5\Big) - c_2 - \frac{c_3}{u^1} = 0 \\ \big(2.5Ru^3 - 1.25(u^3)^2\big)\ln\big(c_1u^2u^3-5\big) + 3Ru^3 - 2.125(u^3)^2 - \frac{13.6R+c_1c_4}{c_1u^2} = 0. \end{cases} \tag{6.4}$$

Here one is really interested in the variables $u^1$ (pressure) and $u^3$ (annular phase), $u^2$ having no physical significance. The constants $c_i$ are defined in terms of more physical parameters as follows:

$$c_1 = \frac{1}{\mu}\sqrt{\frac{\rho R}{2}} \quad c_2 = \frac{bQ_{co}}{\pi}\sqrt{\frac{2\rho}{R}} \quad c_3 = \frac{P_0Q_{co}(1-b)}{\pi}\sqrt{\frac{2\rho}{R}} \quad c_4 = \frac{Q_a}{2\pi}\sqrt{\frac{2\rho}{R}}$$

This example is interesting for us because for some parameter values the solution has a singularity. Physically this means that the flow chokes and at that instant the pressure approaches zero while its derivative goes to $-\infty$. This kind of singularity is called an *impasse point*, see for example Rabier & Rheinboldt (1994) for some discussion. The traditional numerical methods encounter big difficulties when they approach the singularity and the computed solution may not be very accurate near the singularity. However, in our approach the impasse points are really regular points so our numerical method passes the impasse point without any problems. The reason why this is possible is explained in Tuomela (1997) so we just refer to it for more details.

For example the following parameter values yield a singular solution [5]

$$\begin{array}{llll} R = 45.72 & \rho = 0.814 & \mu = 0.098 & b = 0.345 \\ Q_{co} = 1.7153\cdot 10^6 & Q_a = 3.027\cdot 10^5 & P_0 = 13.78 & \\ c_1 = 44.017 & c_2 = 35545 & c_3 = 929940 & c_4 = 9090.9 \end{array}$$

Now when the solution approaches singularity $u^1 \to 0$, $u^3 \to 0$, $u^2 \to \infty$ while $u^2u^3$ seems to stay bounded. Hence let us introduce new variables: $y^1 = u^1$, $y^2 = u^2u^3$ and $y^3 = u^3$; then the constraint equations can be written as

$$f(y) = \begin{cases} y^1y^2(R-y^3)^2\Big(2.5\ln\big(c_1y^2-5\big)+10.5\Big) - c_2y^1y^3 - c_3y^3 = 0 \\ y^2\big(2.5R-1.25y^3\big)\ln\big(c_1y^2-5\big) + y^2\big(3R-2.125y^3\big) - \frac{13.6R+c_1c_4}{c_1} = 0 \end{cases} \tag{6.5}$$

[5]Note that compared to Hairer *et al.* (1989) we have scaled time variable and $y^1$ by $10^{-7}$.

Our solution is thus a curve on $M = f^{-1}(0)$. Then we have to find the distribution. To this end we differentiate $f$:

$$\begin{aligned}
df &= \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ 0 & f_{22} & f_{23} \end{pmatrix} \quad \text{where} \\
f_{11} &= y^2(R-y^3)^2\Big(2.5\ln\big(c_1y^2-5\big)+10.5\Big) - c_2y^3 \\
f_{12} &= y^1(R-y^3)^2\Big(2.5\ln\big(c_1y^2-5\big)+10.5\Big) + \frac{2.5c_1y^1y^2(R-y^3)^2}{c_1y^2-5} \\
f_{13} &= -2y^1y^2(R-y^3)\Big(2.5\ln\big(c_1y^2-5\big)+10.5\Big) - c_2y^1 - c_3 \\
f_{22} &= (2.5R-1.25y^3)\ln\big(c_1y^2-5\big) + \frac{c_1y^2(2.5R-1.25y^3)}{c_1y^2-5} + 3R - 2.125y^3 \\
f_{23} &= -1.25y^2\ln\big(c_1y^2-5\big) - 2.125y^2
\end{aligned}$$

Note that $df$ is of full rank even at point $y^1 = y^3 = 0$. The initial differential equation gives

$$(y^3)^2y^1_1 + (y^2)^2 = 0$$

Combining this with $df$ we define

$$A = \begin{pmatrix} (y^3)^2 & 0 & 0 \\ f_{11} & f_{12} & f_{13} \\ 0 & f_{22} & f_{23} \end{pmatrix} \quad , \quad \tilde{f} = \begin{pmatrix} (y^2)^2 \\ 0 \\ 0 \end{pmatrix} \tag{6.6}$$

$$\tilde{A} = \Big(\tilde{f}\ ,\ A\Big) \quad , \quad \mathscr{D} = \ker(\tilde{A})$$

Now as explained in Tuomela (1997) the solutions to our problem are integral manifolds of the distribution $\mathscr{D}$. But now we easily see that

$$\dim\ker(\tilde{A}) = 1$$

even at $y^1 = y^3 = 0$. Hence the original impasse point is simply a regular point for this distribution, so we can easily compute the solution through the “singularity”.

Note that $y^1$ (i.e. pressure) will be negative after the singularity. Of course this is absurd from the physical point of view. However, from the numerical point of view it is important that we can accurately compute the solution up to singularity, and then simply stop the computation when the result has become nonphysical.

6.2.3 *Particle in a magnetic field* This is a hamiltonian system with one or two invariants. The equations are, with $q, p : \mathbb{R} \to \mathbb{R}^3$,

$$\begin{cases} \dot{q} = \frac{1}{m}p \\ \dot{p} = -\nabla U(q) + \frac{1}{m}B \times p \end{cases} \tag{6.7}$$

where $m$ is the mass of the particle, $U$ is the electrical potential and $B$ is the magnetic field. Introducing the notation

$$\mathsf{skew}(B) := \begin{pmatrix} 0 & -B^3 & B^2 \\ B^3 & 0 & -B^1 \\ -B^2 & B^1 & 0 \end{pmatrix}, \tag{6.8}$$

the above system can be cast into a Hamiltonian system

$$\begin{pmatrix} \dot{q} \\ \dot{p} \end{pmatrix} = J\nabla H \tag{6.9}$$

where $H$ and the non-canonical structure matrix $J$ are given by

$$\begin{aligned} H &= \frac{|p|^2}{2m} + U(q) \\ J &= \begin{pmatrix} 0 & I \\ -I & \mathsf{skew}(B) \end{pmatrix}. \end{aligned}$$

We note that by a straightforward computation one can see that $J$ is a hamiltonian structure matrix (see (Olver, 1986, p. 384)) if and only if $\nabla \cdot B = 0$ which of course is the case for physically relevant magnetic fields.

Now $H$ is one invariant, but it is easy to check that if $B$ is constant, we have in addition a second invariant:

$$\mu = \langle q \times (p + \tfrac{1}{2}\, q \times B), B\rangle = \text{constant}. \tag{6.10}$$

This invariant $\mu$ is called magnetic momentum of the particle.

We remind the reader of our preference using second order formulation instead of first order one, see (Tuomela & Arponen, 2000, remark 3.3). Hence writing $y$ for $q$ we obtain

$$\begin{cases} my_2 + \nabla U(y) - B \times y_1 = 0 \\ \langle y \times (y_1 + \frac{1}{2m} y \times B), B\rangle - a_1 = 0 \\ \frac{m}{2}|y_1|^2 + U(y) - a_2 = 0 \end{cases} \tag{6.11}$$

where the constants $a_1$ and $a_2$ are determined by initial conditions. This is again of the form (3.5). We used the electrical potential $U(y) = -1/|y|$ and the magnetic field $B = (0,0,1)$ in our computations. All the results were qualitatively the same as in Leimkuhler & Reich (2005).

6.2.4 *Ideal 3D magnetohydrodynamics* This system comes from modelling an ideal, incompressible magnetohydrodynamics in 3D, see (Goriely, 2001, p. 48) and references therein. The system has 3 second order equations and one invariant:

$$\begin{cases} y^1_2 - \frac{1}{3}y^1y^3 + \frac{\beta}{3}y^1 = 0 \\ y^2_2 - \frac{1}{3}y^2y^3 - \frac{\beta}{3}y^2 + \frac{\gamma}{3}y^1 = 0 \\ y^3_2 - (y^3)^2 - 8y^2_1 + \beta^2 = 0 \\ 4(y^2)^2 + 3y^3y^2_1 + \frac{1}{12}(y^3)^3 - y^2y^3_1 - \frac{1}{8}(y^3_1)^2 - 3\beta y^2_1 - \frac{1}{4}\beta^2y^3 + 3\gamma y^1_1 = 0. \end{cases} \tag{6.12}$$

One can easily check that the system is involutive. This is also of the form (3.5).

6.2.5 *Mechanical system* This is of the form (3.7) and a straightforward generalization of the final example of Tuomela & Arponen (2000), a mechanical system with $n$ particles. Now $y \in \mathbb{R}^{2n}$, $\lambda \in \mathbb{R}^{n-1}$, $E = I_{2n}$, $K = \nabla F$, where $F = \frac{c}{2}\langle y, \tilde{K}y\rangle$, where $c$ is the parameter of rigidity and $\tilde{K}$ is the stiffness matrix.

For instance, in Tuomela & Arponen (2000) we had $n = 6$, whence

$$\tilde{K} = \begin{pmatrix} I_2 & 0 & -I_2 & 0 & 0 & 0 \\ 0 & I_2 & 0 & -I_2 & 0 & 0 \\ -I_2 & 0 & 2I_2 & 0 & -I_2 & 0 \\ 0 & -I_2 & 0 & 2I_2 & 0 & -I_2 \\ 0 & 0 & -I_2 & 0 & I_2 & 0 \\ 0 & 0 & 0 & -I_2 & 0 & I_2 \end{pmatrix},$$

$$g(y) = \frac{1}{2} \begin{pmatrix} (y^1 - y^3)^2 + (y^2 - y^4)^2 - 1 \\ (y^3 - y^5)^2 + (y^4 - y^6)^2 - 1 \\ (y^5 - y^7)^2 + (y^6 - y^8)^2 - 1 \\ (y^7 - y^9)^2 + (y^8 - y^{10})^2 - 1 \\ (y^9 - y^{11})^2 + (y^{10} - y^{12})^2 - 1 \end{pmatrix},$$

$$\mathscr{R}_1 \ : \quad \begin{cases} g(y) = 0 \\ dg y_1 = 0 \\ \mathscr{D} = \operatorname{span}(V) \\ V = \big(1, y_1, -(dg)^t\lambda - c\tilde{K}y\big) \\ dg(dg)^t\lambda + c\, dg\tilde{K}y - d^2g\,(y_1, y_1) = 0. \end{cases}$$

The energy of the system is

$$E = \frac{1}{2}\left|y_1\right|^2 + \frac{c}{2}\langle y, \tilde{K}y\rangle. \tag{6.13}$$

We can use this example to test the behaviour of our method on as big systems as we please.

## 6.3 *Test runs.*

We show that it is advantageous to:

- project the solutions from $J_q(\mathscr{E})$ to $J_{q-1}(\mathscr{E})$ (system with invariants)
- replace exact Newton by inexact Newton
- initialize the projection.

In the relevant tables we therefore denote

| | | |
|---|---|---|
| Gen. Case | = | General case |
| Sys. with Invar. | = | System with invariants |
| Holon. Case | = | Holonomic case. |

The other symbols of the tables are

| | | |
|---|---|---|
| # succ. (rej.) steps | = | number of successful (rejected) steps by the DE solver |
| Newt. steps: av(max) | = | Newton iteration average (maximum number of Newton iterations within a single Newton loop) |
| CPU dist | = | time used to distributions |
| CPU proj | = | time used to orthogonal projections |
| CPU total | = | total DE solver time. |

Units of the CPU are in seconds. We have several properties to compare, these are organized as follows:

- within each table we compare the effect of initialization for both a constant and an adaptive step size. We are mainly interested in dopri54 but compare it also to Euler.
- to compare between exact/inexact Newton methods, and between Gen. Case/Sys. with Invar., four different tables are made. For instance, for the dissipated rigid body, Tables 1 and 2 compare exact/inexact to each other (both are Gen. Case) as well as Tables 3 and 4 (both are Sys. with Invar.).

To avoid excessive size for this publication, we do not present all four tables for every example. Finally note that in publications on DAEs the initial point is given excessively accurately to delay the drift off. Because drift off does not occur in our method there is no need to give the initial point any special attention.

6.3.1 *Rigid body with dissipation* We solved the system (6.3) using parameters $a = 1$, $\alpha = 0.01$, $I_1 = I_2 = 5/8$ and $I_3 = 1/4$. The results are in Tables 1-4. Other parameters used in the computations are as follows:

Adaptive step size tolerance: $10^{-6}$ ($h^0 = 0.20$)
$\mathsf{fac}_{\mathsf{max}} = 5.0$
Constant step size: $h = 0.20$
Distribution method: explicit
Time frame for the solution: $x = [0, 3600]$

Initial value (general case):

$$(x, y, y_1) = (0, 0.1204, 0.9631, 0.2408, 0.5568, -0.0682, -0.0054)$$

Initial value (systems with invariants):

$$(x, y) = (0, 0.1204, 0.9631, 0.2408).$$

From any of the Tables 1 - 4 can be seen, by comparing a “no INIT” column with a “w/INIT” column, the significant effect of initialization: it speeded up the computation 30% – 50%.

For the exact/inexact comparison, from Tables 1 and 2 (or, Tables 3 and 4) one can see, by comparing corresponding columns, that in this small system inexact Newton is typically slightly slower than exact Newton. In any case the adaptive dopri54 is clearly the fastest.

For the Gen. Case/Sys. with Invar. comparison, from the "CPU total" entries in Tables 1 and 3 (or, Tables 2 and 4) one can see the superiority of Sys. with Invar.: it is 65% – 80% faster. Figure 1 represents the solution.

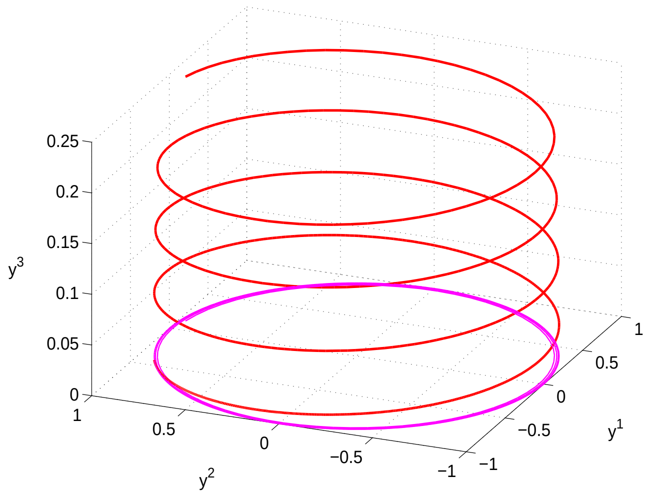

Figure 1. Solution curve of the rigid body and its projection.

6.3.2 *Two phase plug flow problem* The results are in Tables 5-6. Here more interesting than mere efficiency of the code is the fact that with our approach we are able to pass through the impasse point without any problems; i.e. impasse points are regular points for our code. In Figure 2 we show a typical computation using the original model (6.4). When approaching the impasse point one is forced to take smaller and smaller step sizes and it is not possible to pass through the singularity.

In Figure 3 there is a solution computed with formulation given in (6.5) and (6.6). Note that the solutions are smooth curves, in the sense of being smooth one dimensional manifolds, but they are only continuous at the impasse point when parametrised by $x$.

The following parameters were used in the computations:

Adaptive step size tolerance: $10^{-10}$ ($h^0 = 0.1$)
$\mathsf{fac}_{\mathsf{max}} = 4.0$
Constant step size: $h = 0.1$
Distribution method: SVD
Time frame for the solution: $x = [0, 3.2188]$

Initial value:

$$(x,y) = (0, 13.78, 11.394, 4.8147)$$

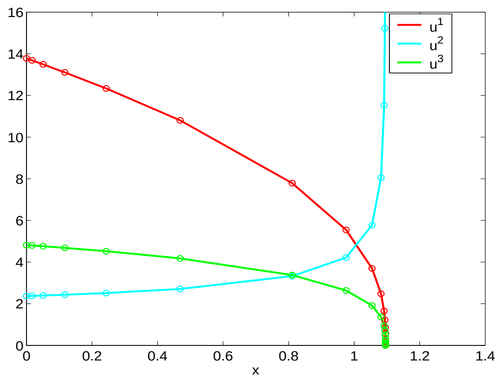

Figure 2. The singularity of the two phase plug flow.

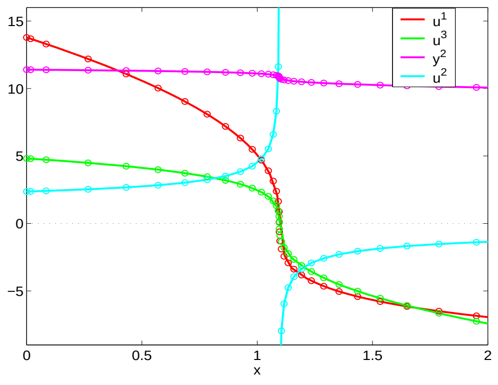

Figure 3. Solution of the two phase plug flow.

6.3.3 *Particle in a magnetic field* The results are in Tables 7-10. The following parameters were used in the computations:

Adaptive step size tolerance: $10^{-5}$ ($h^0 = 0.01$)
$\mathsf{fac}_{\mathsf{max}} = 5.0$
Constant step size: $h = 0.01$
Distribution method (general case): explicit
Time frame for the solution: $x = [0,20]$

Initial value (general case):

$$(x,y,y_1,y_2) = (0,1,1,1,0,0,0,-1/\sqrt{27},-1/\sqrt{27},-1/\sqrt{27}).$$

Initial value (systems with invariants):

$$(x,y,y_1) = (0,1,1,1,0,0,0).$$

The comparison results were very similar to the rigidbody results. Here the effect of initialization is, as in the rigidbody case, a significant speed-up, even 55%. Also for the exact/inexact comparison, the results were similar to the rigidbody case: for this small system, inexact Newton is not faster than exact Newton. But, again we have the superiority of Sys. with Invar.: it is 30% – 80% faster (compare Table 7 to Table 9, or, Table 8 to Table 10 as before). Figure 4 represents the solution.

6.3.4 *Ideal 3D MHD* We computed the solution of (6.12) with $\beta = 1, \gamma = 1$. The results are in Tables 11 and 12. To save space, we present only two tables. The following parameters were used in the computations:

Adaptive step size tolerance: $10^{-7}$ ($h^0 = 0.05$)
$\mathsf{fac}_{\mathsf{max}} = 2.5$
Constant step size: $h = 0.05$

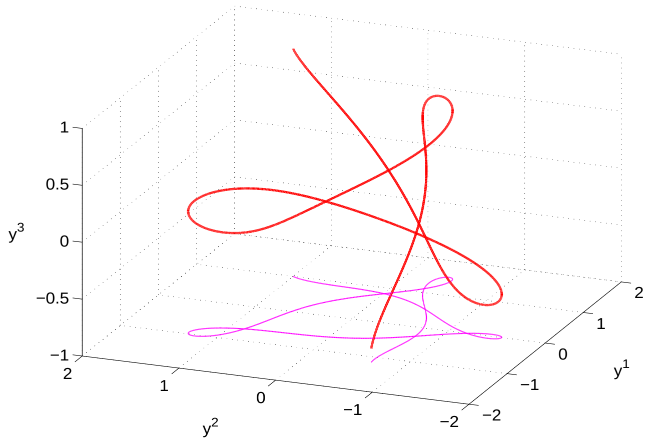

Figure 4. Solution curve of the particle in magnetic field and its projection.

Distribution method: explicit
Time frame for the solution: $x = [0, 1.75]$

Initial value (general case):

$$(x, y, y_1, y_2) = (0, 1, 1, 1, 0.4, 0.5, 0.6, 0, 0.3333, 4).$$

Initial value (systems with invariants):

$$(x, y, y_1) = (0, 1, 1, 1, 0.4, 0.5, 0.6).$$

For this system, the Gen. Case with constant step size dopri54 did not work very well. On the other hand, adaptive dopri54 worked beautifully fast for any combination of Gen. Case/Sys. with Invar./inexact/exact. And, Sys. with Invar. was remarkably $85\% - 98\%$ faster than Gen. Case. The solution is represented in Figure 5.

6.3.5 *Mechanical system* Here the parameter of rigidity was chosen as $c = 10$. The following parameters were used in the computations:

Adaptive step size tolerance: $10^{-6}$
Initial step size $h^0$ :

| | |
|---|---|
| mech6 : | $h^0 = 0.02$ |
| mech24 & mech50 : | $h^0 = 5.0$ |

$\mathsf{fac}_{\mathsf{max}} = 3.0$
Constant step size: $h = 0.02$
Distribution method: conjugate gradient iteration with block solve

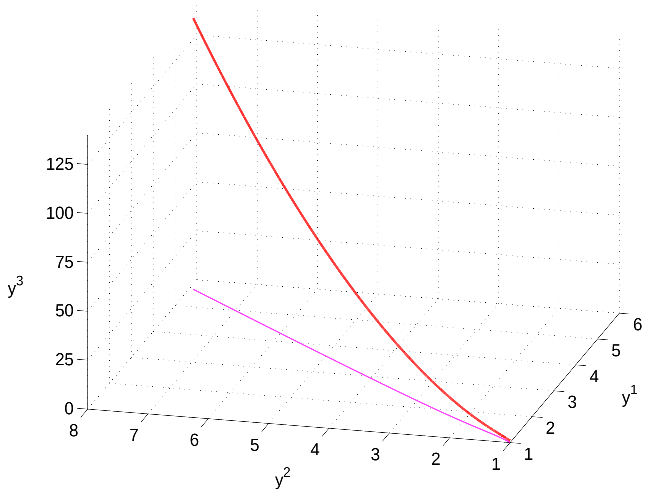

Figure 5. Solution curve of 3D-MHD and its projection.

$E$-system tolerance: (no need to iterate)
$S$-system tolerance: $10^{-12}$
Time frame for the solution:

$$\begin{aligned} \text{mech6 \& mech24 :} &\qquad x = [0,1] \\ \text{mech50 :} &\qquad x = [0,0.73] \end{aligned}$$

We used only dopri54, tested with and without the energy conservation equation. Determining a consistent initial value was based on a similar strategy in all "mechanical" systems. The Figure 7 shows the initial state of 24 particle system after the first orthogonal projection.

**6 particle system**. Constant energy of the system is $E = 59.653$. Solutions with and without conservation of energy are shown in the Table 13. The energy of the 6 particle system in the latter case starts to grow apparently linearly (Figure 6).

**24 particle system**. Constant energy of the system is $E = 3008.6$. Figure 8 shows that energy equation (6.13) was satisfied up to the order $\mathscr{O}(10^{-5})$. The step sizes of dopri54 are shown in Figure 9. The results are in Table 14. The final states with and without conservation of energy are shown in Figures 10-11. Here we needed a longer test run to see the effect of conservation of energy.

Using inexact Newton speeds up the computation, as does the use of initialization. Interestingly, the effect of initialization is much more pronounced in exact Newton case (33%) than in inexact case (6%). In fact, although initialization speeds up both exact and inexact Newton, the former is at the end of the day 15% faster. Also slight slowing down occurs with the constant energy equation. That meets our expectations since we are then handling one extra equation.

**50 particle system**. Constant energy of the system is $E = 26914$. In this case we restricted time frame

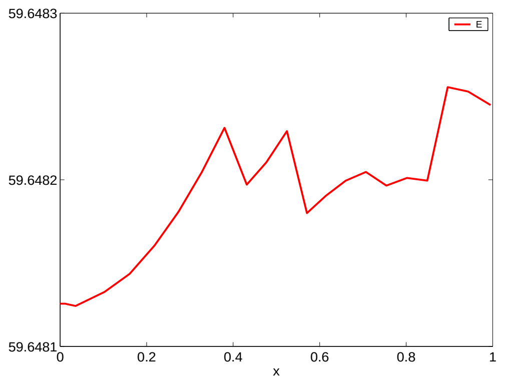

Figure 6. Energy of `mech6`.

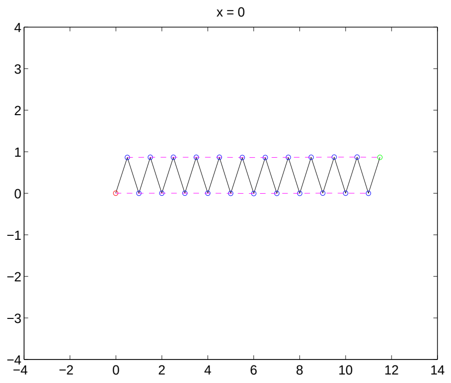

Figure 7. Initial state of `mech24`.

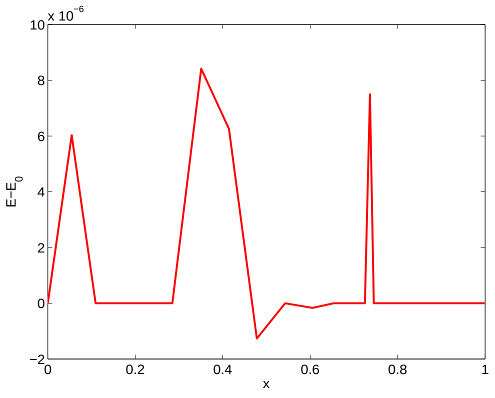

Figure 8. Energy balance of `mech24`.

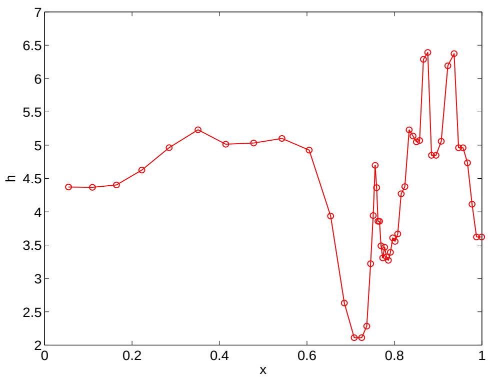

Figure 9. `mech24`: The step sizes of dopri54.

to $x = [0, 0.73]$. The results are in Table 15. With this huge system inexact becomes remarkably faster than exact; comparing the appropriate columns we see that the speed-up is $16\% - 51\%$.

## 7. Conclusion and perspectives

We have enhanced our method proposed in Tuomela & Arponen (2000, 2001) by exploiting the structure of the needed algorithms. First, we introduce an initialization step for the Newton algorithm which typically reduced the computation time by 50%. Second, the matrix needed in the Newton iteration has a particular block structure which makes it possible to use certain reduced size iterative solvers. Third, we showed that sometimes one can reduce the size and computation time of the system in case of invariants. All of these have been demonstrated by several examples. Finally we showed that our method can easily deal with impasse points.

There are several directions where this work can be continued. One problem is that in the Newton iteration we need second (and in the holonomic case even third) order differentials. One possibility to

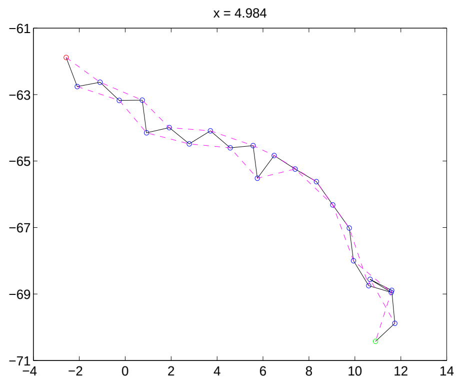

Figure 10. `mech24`: The configuration at $x = 4.984$, without conservation of energy.

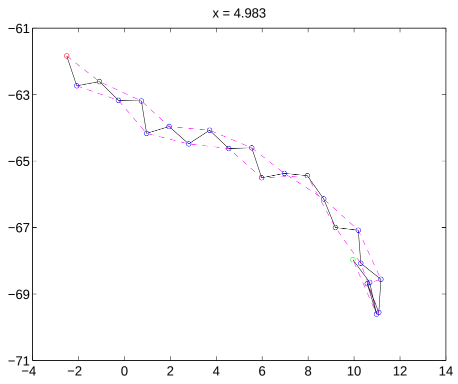

Figure 11. `mech24`: The configuration at $x = 4.983$, with conservation of energy.

avoid forming these symbolically is to use *automatic differentiation* Griewank (2000). We did some preliminary tests, but these did not result in faster code. However, it is quite possible that a more careful implementation could be a competitive alternative. This will be explored in future papers.

We have not discussed stiff problems. Recall that the standard way is to consider DAEs as “infinitely” stiff, which implies that implicit methods should be used. However, as our approach shows DAEs are not intrinsically stiff: indeed in all our examples (standard examples from DAE literature) explicit methods worked very well. Of course *some* overdetermined problems are stiff, and then we should adapt implicit schemes to our case. This will be reported elsewhere.

Finally in the present paper the examples are really quite “academic”. However, we have shown that our method is reasonably fast even for quite large problems. The next step will be to handle some “real life” problems. The work in this direction has already started: we are now collaborating with the *Laboratory of Mechatronics, Lappeenranta University of Technology, Finland* in this direction. The results of this research will appear elsewhere.

## REFERENCES

*Test set for IVP solvers*, http://pitagora.dm.uniba.it/∼testset//.

BHATIA, R. (1997) *Matrix Analysis*, Graduate texts in mathematics, vol. 169, Springer-Verlag.

BRENAN, K., CAMPBELL, S., & PETZOLD, L. (1989) *Numerical solution of initial-value problems in differential-algebraic equations*, North-Holland.

DEMBO, R., EISENSTAT, S., & STEINHAUG, T. (1982) Inexact Newton methods, *SIAM J. Numer. Anal.*, **19** (2), 400–408.

FISCHER, B. (1996) *Polynomial based iteration methods for symmetric linear systems*, Wiley-Teubner Series Advances in Numerical Mathematics, John Wiley & Sons Ltd.

GORIELY, A. (2001) *Integrability and nonintegrability of dynamical systems*, Advanced Series in Nonlinear Dynamics, vol. 19, World Scientific Publishing Co. Inc.

GRIEWANK, A. (2000) *Evaluating derivatives*, SIAM.

HAIRER, E., LUBICH, C., & ROCHE, M. (1989) *The numerical solution of differential-algebraic systems by Runge-Kutta methods*, Lecture notes in mathematics, vol. 1409, Springer-Verlag.

HAIRER, E., NØRSETT, S., & WANNER, G. (1993) *Solving ordinary differential equations I, Nonstiff problems, 2nd ed.*, Springer series in comp. math., vol. 8, Springer-Verlag.

HAIRER, E. & WANNER, G. (1991) *Solving ordinary differential equations II, Stiff and differential-algebraic problems*, Springer series in comp. math., vol. 14, Springer-Verlag.

HORN, R. A. & JOHNSON, C. R. (1994) *Topics in Matrix Analysis*, Cambridge University press.

LEIMKUHLER, B. & REICH, S. (2005) *Simulating Hamiltonian Dynamics*, Cambridge University press.

MIAO, J. (1991) General expressions for the Moore-Penrose inverse of a $2 \times 2$ block matrix, *Lin. Alg. Appl.,* **151**, 1–15.

OLVER, P. J. (1986) *Applications of Lie groups to differential equations*, Graduate texts in mathematics, vol. 107, Springer-Verlag.

RABIER, P. & RHEINBOLDT, W. (1994) On impasse points of quasilinear differential algebraic equations, *J. Math. Anal. Appl.,* **181**, 429–454.

SAUNDERS, D. (1989) *The geometry of jet bundles*, London Math. Soc. Lecture note series, vol. 142, Cambridge university press.

SMITH, S. T. (1994) Optimization techniques on Riemannian manifolds. In: A. BLOCH, ed. *Hamiltonian and gradient flows, algorithms and control.*, Amer. Math. Soc., Providence, RI, 113–136.

SPIVAK, M. (1979) *A comprehensive introduction to differential geometry, vol 1–5, 2nd ed.* Publish or Perish.

TREFETHEN, N. & BAU, D. (1997) *Numerical linear algebra*, SIAM.

TUOMELA, J. (1997) On singular points of quasilinear differential and differential-algebraic equations, *BIT,* **37**, 966–975.

TUOMELA, J. & ARPONEN, T. (2000) On the numerical solution of involutive ordinary differential systems, *IMA J. Num. Anal.,* **20**, 561–599.

———, (2001) On the numerical solution of involutive ordinary differential systems: Higher order methods, *BIT,* **41**, 599–628.

TABLE 1 `rigidbody`, *Gen. Case: Run characteristics with exact Newton.*

| | Euler (constant) | Euler (adaptive) | dopri54 (constant) | dopri54 (adaptive) |
|---|---|---|---|---|
| no INIT: | | | | |
| # succ. (rej.) steps | 18049 | 4835(491) | 18019 | 115(6) |
| Newt. steps: av(max) | 0.135(3) | 1.65(3) | 0.0302(2) | 0.819(3) |
| CPU dist | 1.03 | 0.60 | 7.03 | 0.05 |
| CPU proj | 1.32 | 7.02 | 3.12 | 0.42 |
| CPU total | 2.52 | 7.78 | 12.34 | 0.48 |
| w/INIT: | | | | |
| # succ. (rej.) steps | 18049 | 4835(491) | 18019 | 115(6) |
| Newt. steps: av(max) | 0.135(3) | 1.65(3) | 0.0302(2) | 0.819(3) |
| CPU dist | 0.95 | 0.51 | 6.77 | 0.04 |
| CPU proj | 0.95 | 4.05 | 1.95 | 0.27 |
| CPU total | 2.00 | 4.73 | 10.83 | 0.32 |

TABLE 2 `rigidbody`, *Gen. Case: Run characteristics with inexact Newton.*

| | Euler (constant) | Euler (adaptive) | dopri54 (constant) | dopri54 (adaptive) |
|---|---|---|---|---|
| no INIT: | | | | |
| # succ. (rej.) steps | 18049 | 4835(491) | 18019 | 115(6) |
| Newt. steps: av(max) | 0.135(3) | 1.65(3) | 0.0302(2) | 0.819(3) |
| CPU dist | 1.04 | 0.63 | 6.70 | 0.02 |
| CPU proj | 1.64 | 8.39 | 2.99 | 0.62 |
| CPU total | 2.86 | 9.29 | 11.80 | 0.64 |
| w/INIT: | | | | |
| # succ. (rej.) steps | 18049 | 4835(491) | 18019 | 115(6) |
| Newt. steps: av(max) | 0.135(3) | 1.65(3) | 0.0302(2) | 0.819(3) |
| CPU dist | 1.06 | 0.61 | 6.75 | 0.02 |
| CPU proj | 1.34 | 5.89 | 2.25 | 0.45 |
| CPU total | 2.53 | 6.67 | 10.99 | 0.50 |

TABLE 3 `rigidbody`, *Sys. with Invar.: Run characteristics with exact Newton.*

| | Euler (constant) | Euler (adaptive) | dopri54 (constant) | dopri54 (adaptive) |
|---|---|---|---|---|
| no INIT: | | | | |
| # succ. (rej.) steps | 18010 | 4902(800) | 18016 | 134(11) |
| Newt. steps: av(max) | 0.0805(3) | 1.5(3) | 0.0239(2) | 0.793(2) |
| CPU dist | 0.16 | 0.11 | 0.95 | 0.02 |
| CPU proj | 0.33 | 1.29 | 1.61 | 0.06 |
| CPU total | 0.56 | 1.56 | 4.08 | 0.12 |
| w/INIT: | | | | |
| # succ. (rej.) steps | 18010 | 4902(800) | 18016 | 134(11) |
| Newt. steps: av(max) | 0.0805(3) | 1.5(3) | 0.0239(2) | 0.793(2) |
| CPU dist | 0.15 | 0.17 | 1.03 | 0.03 |
| CPU proj | 0.15 | 0.74 | 0.86 | 0.05 |
| CPU total | 0.44 | 1.09 | 3.37 | 0.09 |

TABLE 4 `rigidbody`, *Sys. with Invar.: Run characteristics with inexact Newton.*

| | Euler (constant) | Euler (adaptive) | dopri54 (constant) | dopri54 (adaptive) |
|---|---|---|---|---|
| no INIT: | | | | |
| # succ. (rej.) steps | 18010 | 4902(800) | 18016 | 134(11) |
| Newt. steps: av(max) | 0.0805(3) | 1.5(3) | 0.0239(2) | 0.793(2) |
| CPU dist | 0.13 | 0.20 | 1.07 | 0.01 |
| CPU proj | 0.33 | 1.69 | 1.27 | 0.13 |
| CPU total | 0.58 | 2.06 | 3.87 | 0.16 |
| w/INIT: | | | | |
| # succ. (rej.) steps | 18010 | 4902(800) | 18016 | 134(11) |
| Newt. steps: av(max) | 0.0805(3) | 1.5(3) | 0.0239(2) | 0.793(2) |
| CPU dist | 0.19 | 0.16 | 1.16 | 0.01 |
| CPU proj | 0.21 | 1.07 | 0.66 | 0.08 |
| CPU total | 0.47 | 1.32 | 3.34 | 0.11 |

TABLE 5 `plugflow`, *Sys. with Invar.: Run characteristics with exact Newton.*

| | Euler (constant) | Euler (adaptive) | dopri54 (constant) | dopri54 (adaptive) |
|---|---|---|---|---|
| no INIT: | | | | |
| # succ. (rej.) steps | 289 | 101537(4) | 288 | 37(0) |
| Newt. steps: av(max) | 2(2) | 1(2) | 1(2) | 0.868(2) |
| CPU dist | 0.03 | 11.15 | 0.09 | 0.01 |
| CPU proj | 0.05 | 37.88 | 0.25 | 0.05 |
| CPU total | 0.08 | 51.01 | 0.34 | 0.06 |
| w/INIT: | | | | |
| # succ. (rej.) steps | 289 | 101558(4) | 288 | 37(0) |
| Newt. steps: av(max) | 2.99(3) | 1.01(3) | 1.3(3) | 1.13(3) |
| CPU dist | 0.01 | 10.42 | 0.12 | 0.01 |
| CPU proj | 0.08 | 17.90 | 0.17 | 0.03 |
| CPU total | 0.09 | 30.40 | 0.29 | 0.06 |

TABLE 6 `plugflow`, *Sys. with Invar.: Run characteristics with inexact Newton.*

| | Euler (constant) | Euler (adaptive) | dopri54 (constant) | dopri54 (adaptive) |
|---|---|---|---|---|
| no INIT: | | | | |
| # succ. (rej.) steps | 289 | 102705(4) | 288 | 37(0) |
| Newt. steps: av(max) | 23.8(26) | 1.03(21) | 2.45(11) | 5.71(30) |
| CPU dist | 0.01 | 12.10 | 0.13 | 0.01 |
| CPU proj | 0.51 | 45.56 | 0.50 | 0.18 |
| CPU total | 0.52 | 59.82 | 0.66 | 0.19 |
| w/INIT: | | | | |
| # succ. (rej.) steps | 289 | 102705(4) | 288 | 37(0) |
| Newt. steps: av(max) | 23.8(26) | 1.03(21) | 2.45(11) | 5.71(30) |
| CPU dist | 0.01 | 10.72 | 0.10 | 0.02 |
| CPU proj | 0.51 | 16.54 | 0.41 | 0.15 |
| CPU total | 0.52 | 29.41 | 0.54 | 0.18 |

TABLE 7 `partmagnfield`, *Gen. Case: Run characteristics with exact Newton.*

| | Euler (constant) | Euler (adaptive) | dopri54 (constant) | dopri54 (adaptive) |
|---|---|---|---|---|
| no INIT: | | | | |
| # succ. (rej.) steps | 3875 | 2106(0) | 3825 | 55(11) |
| Newt. steps: av(max) | 1.81(2) | 2(2) | 0(0) | 1.75(4) |
| CPU dist | 0.30 | 0.29 | 1.96 | 0.02 |
| CPU proj | 6.42 | 11.15 | 0.78 | 1.08 |
| CPU total | 6.76 | 11.56 | 3.26 | 1.13 |
| w/INIT: | | | | |
| # succ. (rej.) steps | 3875 | 2106(0) | 3825 | 55(11) |
| Newt. steps: av(max) | 1.81(2) | 2(2) | 0(0) | 1.75(4) |
| CPU dist | 0.24 | 0.30 | 2.05 | 0.03 |
| CPU proj | 3.90 | 6.75 | 0.34 | 0.74 |
| CPU total | 4.20 | 7.10 | 2.86 | 0.79 |

TABLE 8 `partmagnfield`, *Gen. Case: Run characteristics with inexact Newton.*

| | Euler (constant) | Euler (adaptive) | dopri54 (constant) | dopri54 (adaptive) |
|---|---|---|---|---|
| no INIT: | | | | |
| # succ. (rej.) steps | 3875 | 2106(0) | 3825 | 55(11) |
| Newt. steps: av(max) | 1.81(2) | 2(2) | 0(0) | 1.75(4) |
| CPU dist | 0.32 | 0.22 | 1.91 | 0.04 |
| CPU proj | 7.44 | 12.87 | 0.50 | 1.72 |
| CPU total | 7.77 | 13.25 | 3.01 | 1.77 |
| w/INIT: | | | | |
| # succ. (rej.) steps | 3875 | 2106(0) | 3825 | 55(11) |
| Newt. steps: av(max) | 1.81(2) | 2(2) | 0(0) | 1.75(4) |
| CPU dist | 0.28 | 0.30 | 1.90 | 0.02 |
| CPU proj | 5.48 | 9.57 | 0.37 | 1.46 |
| CPU total | 5.84 | 9.97 | 2.87 | 1.50 |

TABLE 9 `partmagnfield`, *Sys. with Invar.: Run characteristics with exact Newton.*

| | Euler (constant) | Euler (adaptive) | dopri54 (constant) | dopri54 (adaptive) |
|---|---|---|---|---|
| no INIT: | | | | |
| # succ. (rej.) steps | 2860 | 1596(10) | 2860 | 44(11) |
| Newt. steps: av(max) | 1.43(2) | 1.93(2) | 0(0) | 1.47(3) |
| CPU dist | 0.03 | 0.05 | 0.25 | 0.01 |
| CPU proj | 1.11 | 2.04 | 0.33 | 0.22 |
| CPU total | 1.15 | 2.13 | 0.88 | 0.25 |
| w/INIT: | | | | |
| # succ. (rej.) steps | 2860 | 1596(10) | 2860 | 44(11) |
| Newt. steps: av(max) | 1.43(2) | 1.93(2) | 0(0) | 1.47(3) |
| CPU dist | 0.01 | 0.04 | 0.10 | 0.01 |
| CPU proj | 0.70 | 1.29 | 0.22 | 0.16 |
| CPU total | 0.73 | 1.35 | 0.71 | 0.18 |

TABLE 10 `partmagnfield`, *Sys. with Invar.: Run characteristics with inexact Newton.*

| | Euler (constant) | Euler (adaptive) | dopri54 (constant) | dopri54 (adaptive) |
|---|---|---|---|---|
| no INIT: | | | | |
| # succ. (rej.) steps | 2860 | 1596(10) | 2860 | 44(11) |
| Newt. steps: av(max) | 1.43(2) | 1.93(2) | 0(0) | 1.47(3) |
| CPU dist | 0.01 | 0.03 | 0.27 | 0.02 |
| CPU proj | 1.44 | 2.72 | 0.27 | 0.39 |
| CPU total | 1.50 | 2.80 | 0.78 | 0.43 |
| w/INIT: | | | | |
| # succ. (rej.) steps | 2860 | 1596(10) | 2860 | 44(11) |
| Newt. steps: av(max) | 1.43(2) | 1.93(2) | 0(0) | 1.47(3) |
| CPU dist | 0.03 | 0.03 | 0.24 | 0.00 |
| CPU proj | 1.11 | 2.02 | 0.21 | 0.36 |
| CPU total | 1.15 | 2.13 | 0.70 | 0.38 |

TABLE 11 `3Dmhd`, *Gen. Case: Run characteristics with exact Newton.*

| | Euler (constant) | Euler (adaptive) | dopri54 (constant) | dopri54 (adaptive) |
|---|---|---|---|---|
| no INIT: | | | | |
| # succ. (rej.) steps | 362730 | 9958(278) | 390587 | 53(0) |
| Newt. steps: av(max) | 0.732(2) | 1.76(2) | 0.000197(2) | 0.981(3) |
| CPU dist | 22.34 | 1.22 | 167.30 | 0.02 |
| CPU proj | 206.99 | 35.08 | 68.91 | 0.49 |
| CPU total | 232.87 | 36.69 | 295.97 | 0.53 |
| w/INIT: | | | | |
| # succ. (rej.) steps | 362730 | 9958(278) | 390587 | 53(0) |
| Newt. steps: av(max) | 0.732(2) | 1.86(3) | 0.000197(2) | 1.57(4) |
| CPU dist | 22.67 | 1.25 | 166.80 | 0.03 |
| CPU proj | 43.08 | 18.89 | 28.71 | 0.38 |
| CPU total | 69.40 | 20.51 | 256.31 | 0.43 |

TABLE 12 `3Dmhd`, *Sys. with Invar.: Run characteristics with exact Newton.*

| | Euler (constant) | Euler (adaptive) | dopri54 (constant) | dopri54 (adaptive) |
|---|---|---|---|---|
| no INIT: | | | | |
| # succ. (rej.) steps | 24546 | 5545(1) | 27094 | 35(0) |
| Newt. steps: av(max) | 1.05(2) | 1.73(2) | 1(2) | 0.944(3) |
| CPU dist | 0.16 | 0.13 | 1.77 | 0.01 |
| CPU proj | 4.90 | 4.31 | 7.27 | 0.07 |
| CPU total | 5.21 | 4.63 | 12.22 | 0.09 |
| w/INIT: | | | | |
| # succ. (rej.) steps | 24546 | 5545(1) | 27094 | 35(0) |
| Newt. steps: av(max) | 1.05(2) | 1.73(2) | 1(2) | 1.06(4) |
| CPU dist | 0.26 | 0.10 | 1.74 | 0.01 |
| CPU proj | 1.38 | 2.26 | 2.45 | 0.04 |
| CPU total | 1.78 | 2.46 | 7.10 | 0.05 |

TABLE 13 `mech6`, *Holon. Case: Run characteristics (dopri54, adaptive).*

| energy / Newton method | yes / exact | yes / inexact | no / exact | no / inexact |
|---|---|---|---|---|
| no INIT: | | | | |
| # succ. (rej.) steps | 22(1) | 22(1) | 23(1) | 23(1) |
| Newt. steps: av(max) | 0.913(3) | 0.913(3) | 0.958(3) | 0.958(3) |
| CPU dist | 0.01 | 0.04 | 0.05 | 0.06 |
| CPU proj | 1.91 | 1.55 | 1.52 | 0.99 |
| CPU total | 2.01 | 1.69 | 1.65 | 1.10 |
| w/INIT: | | | | |
| # succ. (rej.) steps | 22(1) | 22(1) | 23(1) | 23(1) |
| Newt. steps: av(max) | 0.913(3) | 0.913(3) | 0.958(3) | 0.958(3) |
| CPU dist | 0.03 | 0.04 | 0.05 | 0.05 |
| CPU proj | 1.20 | 1.42 | 0.89 | 0.88 |
| CPU total | 1.30 | 1.53 | 0.99 | 0.97 |

TABLE 14 *`mech24`, Holon. Case: Run characteristics (`dopri54`, adaptive).*

| energy / Newton method | yes / exact | yes / inexact | no / exact | no / inexact |
|---|---|---|---|---|
| no INIT: | | | | |
| # succ. (rej.) steps | 49(10) | 49(10) | 49(11) | 49(11) |
| Newt. steps: av(max) | 1.7(3) | 2.14(5) | 1.44(3) | 1.64(4) |
| CPU dist | 1.98 | 1.80 | 1.96 | 1.99 |
| CPU proj | 353.08 | 292.66 | 308.35 | 234.99 |
| CPU total | 364.70 | 303.78 | 315.23 | 240.98 |
| w/INIT: | | | | |
| # succ. (rej.) steps | 49(10) | 49(10) | 49(11) | 49(11) |
| Newt. steps: av(max) | 1.8(4) | 2.02(5) | 1.44(3) | 1.58(4) |
| CPU dist | 1.92 | 1.90 | 1.98 | 1.91 |
| CPU proj | 234.36 | 274.33 | 191.06 | 217.25 |
| CPU total | 242.96 | 285.22 | 196.19 | 222.62 |

TABLE 15 *`mech50`, Holon. Case: Run characteristics (`dopri54`, adaptive).*

| energy / Newton method | yes / exact | yes / inexact | no / exact | no / inexact |
|---|---|---|---|---|
| no INIT: | | | | |
| # succ. (rej.) steps | 14(4) | 14(4) | 14(4) | 14(4) |
| Newt. steps: av(max) | 1.53(3) | 2.33(5) | 1.47(3) | 1.6(4) |
| CPU dist | 2.31 | 2.37 | 2.36 | 2.40 |
| CPU proj | 1003.40 | 483.38 | 821.63 | 352.23 |
| CPU total | 1071.50 | 586.61 | 889.01 | 432.84 |
| w/INIT: | | | | |
| # succ. (rej.) steps | 14(4) | 14(4) | 14(4) | 14(4) |
| Newt. steps: av(max) | 1.53(4) | 2.33(5) | 1.47(3) | 1.53(4) |
| CPU dist | 2.31 | 2.34 | 2.39 | 2.36 |
| CPU proj | 621.41 | 455.66 | 485.03 | 324.82 |
| CPU total | 665.83 | 557.81 | 529.45 | 399.21 |

## List of Figures