Sequential Regularization Methods for simulating mechanical systems with many closed loops

Uri Ascher*       Ping Lin†

April, 1998

Abstract

The numerical simulation problem of large multibody systems has often been treated in two separate stages: (i) the forward dynamics problem for computing system accelerations from given force functions and constraints, and (ii) the numerical integration problem for advancing the state in time. For the forward dynamics problem, algorithms have been given with optimal, linear complexity in the number of bodies, in case that the system topology does not contain many closed loops.

But the interaction between these two stages can be important. Using explicit time integration schemes, we propose a sequential regularization method that has a linear complexity in the number of bodies per time step, even in the presence of many closed loops. The method also handles certain types of constraint singularity.

Keywords. Differential-algebraic equations, regularization, stabilization, higher index, multibody systems, robot simulation, constraint singularities.

AMS(MOS) subject classifications. 65L10, 65L20

1 Introduction

There has been a growing interest in the development of more efficient algorithms for multibody dynamics simulations. The increase in size and complexity of spacecraft and robotic systems is one motivation for this development; another is physically-based modeling in computer graphics. The numerical simulation process has been typically treated as two separate stages. The first stage consists of the forward dynamics problem for computing system accelerations, given the various constraints, torque and force functions. For tree-structured multibody systems, algorithms have been proposed with optimal $O(n)$ complexity, where $n$ is the total number of rigid bodies in the system (see e.g. [10, 13, 18, 6, 21]). They have also been extended to cope with systems with a small number of closed loops compared with the total number $n$ of links [10, 18]. For a system with $m$ closed loops ($m < n$), a typical complexity of $O(n + m^3)$ is obtained using a cut-loop technique. But to our knowledge,

*Institute of Applied Mathematics and Department of Computer Science, University of British Columbia, Vancouver, B.C., Canada V6T 1Z4 (ascher@cs.ubc.ca). The work of this author was partially supported under NSERC Canada Grant OGP0004306.
†Division of Mechanics and Computation, Durand 266, Stanford University, Stanford CA94305-4040 (plin@scm.Stanford.edu)
no $O(n)$ algorithm has been reported in the literature for cases where the topology of the multibody system contains a large number (e.g. $m \geq \gamma n$ for some $0 < \gamma < 1$) of closed loops.

The second stage of the simulation algorithm design addresses the numerical integration problem for advancing the state in time, obtaining generalized body positions and velocities from the computed accelerations. Explicit or implicit time discretization schemes can generally be used, although in many applications in robotics and graphics explicit schemes seem to be preferred.

While these two stages are usually treated separately, there are situations in which the specific treatment of one affects the other, so a global, unified view is beneficial (e.g. [6, 1]). In this paper we use such a global view of the simulation process and devise a method which requires $O(n)$ operations per time step even in the presence of many closed loops. Specifically, we propose using a sequential regularization method (SRM) [4, 5] for this purpose, combined with an explicit time integration scheme. The method produces iterates which get arbitrarily close to the solution of the discretized differential system. It also handles certain types of constraint singularity; however, the latter aspect is not proved in this article.

The mathematical modeling of constrained multibody systems yields differential-algebraic equations (DAEs) of index 2 or 3 [9, 11]. For tree-structured systems (i.e. no closed loops), one can formulate the model in terms of a minimal set of relative solution coordinates, obtaining a system of ordinary differential equations (ODEs), see e.g. [14, 10]. Some existing commercial software packages (e.g. SD/FAST\(^1\)) utilize this approach. Forward dynamics algorithms of complexity $O(n)$ can be interpreted then as embedding the ODE in a DAE, at a given time, and eliminating some of the unknowns locally in the larger but sparser system [6].

In this work, however, we consider the equations of motion in descriptor form (see, e.g., [9, 14]). These are formulations in non-minimal (redundant) sets of coordinates which yield an often simpler, albeit larger, DAE, even in the tree-structured case. This DAE is typically treated by differentiating the constraints to the acceleration level and using one of the well-known stabilization techniques (see, e.g. [2, 3]). An $O(n)$ forward dynamics algorithm for this formulation is recalled in §2, following [18].

A system with closed loops may now be considered as being composed of a tree-structured system plus a set of loop-closing constraints. The latter are treated using an SRM technique combined with an explicit time discretization. The method is described in §3, where we prove that the number of operations needed per time step when using an explicit time discretization scheme remains $O(n)$, for any $m \leq n$.

Finally, in §4 we demonstrate our algorithm on two closed-loop chain examples. For the first example we also demonstrate that the excellent package MEXX [18] has an execution time per time-step which increases much faster than linearly in the number of bodies $n$, as theory predicts. MEXX is well-known for its robust linear algebra routines, and other general-purpose packages are not expected to exhibit a significantly better complexity. On the other hand, the running time of our algorithm per time step remains $O(n)$ as the number of bodies is increased. While avoiding a full, direct run-time comparison between a method

---

\(^1\) SD/FAST is a trademark of Symbolic Dynamics, Inc., 561 Bush Street, Mountain View, CA 94041 USA.
and a package, we still see that for realistic, large enough values of $n$ our method becomes clearly superior.

2 Algorithms with optimal computational complexity for tree-structured problems

2.1 General multibody systems

Consider an idealized multibody system consisting of rigid bodies and point masses with a kinematic tree-structure. We use redundant, world coordinates $p$ for the positions of the system, i.e. for describing the position and orientation of each individual body. The set of feasible positions, which correspond to physically possible geometric configurations, is given by holonomic constraints,

$$g(p) = 0. \quad (2.1)$$

Differentiations of the constraints with respect to time yield corresponding constraints for the velocities $\dot{p}$,

$$G(p)\dot{p} = 0 \quad (2.2)$$

and the accelerations $\ddot{p}$,

$$G(p)\ddot{p} + \frac{\partial (G(p)\dot{p})}{\partial p} = G(p)\ddot{p} - \gamma(p, \dot{p}) = 0. \quad (2.3)$$

Here $G(p)$ is the constraint Jacobian matrix, $G = \frac{\partial g}{\partial p}$. The Euler-Lagrange equations are

$$M(p)\ddot{p} = f(t, p, \dot{p}) - G(p)^T \lambda. \quad (2.4)$$

In combination with the constraint equations at the acceleration level (2.3) the Euler-Lagrange equations form a DAE of index 1 for the differential variables $p$, $\dot{v}$ (= $\dot{p}$) and the algebraic variables $\lambda$:

$$\begin{pmatrix} \dot{p} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} M(p) & G(p)^T \\ G(p) & 0 \end{pmatrix} \begin{pmatrix} \dot{p} \\ \lambda \end{pmatrix} = \begin{pmatrix} f(t, p, v) \\ \gamma(p, v) \end{pmatrix}. \quad (2.5)$$

The constraints on the position level (2.1) and on the velocity level

$$G(p)v = 0 \quad (2.6)$$

define an invariant manifold for (2.5).

It is well-known that simply simulating (2.5) numerically may cause severe drift off the constraints manifold, manifesting a mild instability in this formulation. This phenomenon can be prevented by using stabilization techniques (e.g. invariant stabilization, Baumgarte’s stabilization and projection methods [2, 3, 7]) or regularization techniques (e.g. penalty and sequential regularization methods [17, 19, 12, 15, 16, 8, 4, 5, 20]) during the numerical integration. In this paper, we assume that such a stabilization technique has already been applied and is included in (2.5), and we concentrate on aspects of solving this system.
2.2 Algorithms for tree-structured problems

Consider the case of a multibody system with a kinematic tree structure. As in [18], consider a graph whose nodes correspond to the bodies and whose edges represent the joints in the system (see, e.g., Figure 2.1). When there are no closed loops, the graph consists of trees. In each tree one body is singled out as the root. Every other body then has a unique father in the tree, which is its neighboring body on the path to the root. A node “0” is introduced as the father of the roots. This gives one tree, and we label its nodes as follows: bodies (links) are numbered from 1 to \( n \) such that the label of a body is always greater than that of its father. Joints are numbered such that a joint common to a father body and a son body has the number of the son. For example, in Figure 2.1, body 2 is the father of body 7, and these two bodies are connected by joint 7.

We use simple, redundant, absolute generalized coordinates \( \mathbf{p}_j^T = (x_j^T, \mathbf{d}_j^T) \) to describe the \( j \)th body, where \( x_j \) are the center of mass coordinates and \( \mathbf{d}_j \) is an orientation vector along the body, e.g. connecting its joints, see examples in §4. Thus, \( \mathbf{p}_j^T = (p_{j1}^T, \ldots, p_{j6}^T) \).

For this kind of tree-structured systems, the mass matrix \( M \) is symmetric positive definite and block-diagonal with blocks \( M_k, k = 1, 2, \ldots, n \), which are symmetric positive definite. The constraint equations in (2.3) of a joint connecting two bodies \( k \) and \( j = \text{father of } k \) are of the form (see [18]):

\[
G_k \mathbf{v}_k + Q_k \mathbf{v}_j = \gamma_k, \tag{2.7}
\]

where \( \mathbf{v}_k = \dot{\mathbf{p}}_k \), and \( G_k \) is assumed to have full row rank for each \( k \). Then, as analyzed by many articles (see, e.g. [10, 13, 18, 21]), a recursive algorithm can be derived to obtain an \( O(n) \) algorithm to invert the left-hand matrix in (2.5b).
Let us recall a version of this algorithm following [18]. We start at a terminal body \( k \) (i.e. one which has no sons):

\[
M_k u_k + G_k^T \lambda_k = f_k,
\]

\[
G_k u_k = \gamma_k - Q_k u_j.
\]

(2.8a)

(2.8b)

Setting \( \hat{M}_k = M_k \), \( \hat{f}_k = f_k \), we can obtain

\[
\lambda_k = -(G_k \hat{M}_k^{-1} G_k^T)^{-1}(\gamma_k - Q_k u_j - G_k \hat{M}_k^{-1} \hat{f}_k).
\]

(2.9)

Next, consider the equations for a body \( j \) which has terminal sons \( k \):

\[
M_j u_j + G_j^T \lambda_j + \sum_{k=\text{sons of } j} Q_k^T \lambda_k = f_j,
\]

\[
G_j u_j = \gamma_j - Q_j u_i,
\]

(2.10a)

(2.10b)

where \( i = \text{father of } j \). For each son, (2.9) holds. Substitution into (2.10) yields:

\[
\hat{M}_j u_j + G_j^T \lambda_j = \hat{f}_j,
\]

\[
\hat{G}_j u_j = \gamma_j - Q_j u_i,
\]

(2.11a)

(2.11b)

where

\[
\hat{M}_j = M_j + \sum_{k=\text{sons of } j} Q_k^T (G_k \hat{M}_k^{-1} G_k^T)^{-1} Q_k
\]

(2.12a)

\[
\hat{f}_j = f_j + \sum_{k=\text{sons of } j} Q_k^T (G_k \hat{M}_k^{-1} G_k^T)^{-1}(\gamma_k - G_k \hat{M}_k^{-1} \hat{f}_k).
\]

(2.12b)

The system (2.11) has the same form as (2.8), so a recursive algorithm results:

**Algorithm**

**step 1:** Climb down the tree (towards the root) recursively, repeatedly forming \( \hat{M}_j \) and \( \hat{f}_j \) by (2.12).

**step 2:** Starting from the root of the tree, solve each local system (2.11) for \( u_j \) (with \( u_i \) known) by climbing up the tree recursively.

Note that in Step 1, we replace the original problem by \( n \) local problems (2.11). This algorithm has \( O(n) \) complexity because all operations are local for one body and we only climb down and up the tree once. The work for different subtrees can be done in parallel as well.

The result of the forward dynamics algorithm just described is an expression for \( u = \dot{v} \) in terms of \( v = \dot{p} \) and \( p \). This is an ODE system that must be integrated in time. Note that the dimension of each of \( u \), \( v \) and \( p \) in 3D is \( \hat{n} = 6n \). This is contrasted with the treatment using minimal coordinates, where \( u \), \( v \) or \( p \) may have dimension as small as \( \hat{n} = n \). However, the saving in the latter formulation is chiefly in the time advancement, and in a faster forward dynamics algorithms for small \( n \), but not in the \( O(n) \) forward dynamics algorithm (see [6]2).

\[ ^2 \text{Also, the resulting functions for } f, \ M, \text{ etc., are typically more complex and may be more costly to evaluate. We do not address this question in the present article.} \]
3 Algorithms for systems with many closed loops

We consider now multibody systems with loops. The problem can be seen as a tree-structured system plus some extra loop-closing constraints in the form

$$ D_h p_h - D_j p_j = 0, \quad i = 1, \ldots, m $$

(3.1)

where $D_h$ and $D_j$ are constant matrices with full row rank, and $m$ represents the number of loops. We can write (3.1) as

$$ D p = 0, $$

(3.2)

where $D$ is a sparse $\hat{m} \times \hat{n}$ constant matrix (e.g., $\hat{m} = 3m$ is 3D) which has a full row rank and only $O(n)$ nonzero elements. In the closed-loop case, we need to “invert” a matrix $\hat{T} = \begin{pmatrix} T & D^T \\ D & 0 \end{pmatrix}$ instead of the matrix $T = \begin{pmatrix} M & G^T \\ G & 0 \end{pmatrix}$ in (2.5b). Block Gaussian elimination gives $\begin{pmatrix} T & -D T^{-1} D^T \\ 0 & D \end{pmatrix}$, where the Schur complement $D T^{-1} D^T$ can be obtained using the $O(n)$ algorithm described in the previous section. Using usual techniques for inverting $D T^{-1} D^T$, an $O(n + \hat{m}^2)$ algorithm results which is satisfactory when $m$ is small (cf. [18]).

For problems with a large number of loops, it appears difficult to extend this $O(n)$ algorithm (cf. [10]). In particular, no $O(m)$ algorithm for inverting the Schur complement is available. Unrelated to this, we have found no discussion in the literature about the performance of the method when the constraint Jacobian matrix $\begin{pmatrix} G \\ D \end{pmatrix}$ is rank deficient. (This situation often happens in closed-loop chains, including the first example in §4.) The sequential regularization method provides a possibility to address these two challenges, although we have a proof only for the first.

3.1 The algorithm

We now write down the Euler-Lagrange equations for a multibody system with loops as

$${\dot{p}} = v, \quad (3.3a)$$

$$M \ddot{v} = f(p,v) - G^T(p) \lambda - D^T \nu, \quad (3.3b)$$

$$g(p) = 0 \text{ (constraints corresponding to a tree structure) }, \quad (3.3c)$$

$$D p = 0 \text{ (loop-closing constraints) }, \quad (3.3d)$$

where the number of rows of $D$ may be large, but it has a sparse structure. We assume that $G$ and $D$ have full rank. This assumption is generally true for chain problems (see examples in §4).

If we apply a stabilization method (e.g., Baumgarte’s stabilization [7], or the first stage of the invariant stabilization [2, 3], or projection methods [2]) first for the constraints (3.3c), then the following system must be considered,

$${\dot{p}} = v, \quad (3.4a)$$

$$M \ddot{v} = f(p,v) - G^T(p) \lambda - D^T \nu, \quad (3.4b)$$

$$G(p) \dot{v} = r(p,v), \quad (3.4c)$$

$$D p = 0. \quad (3.4d)$$
For any given function $\nu = \nu(p, v)$, the system (3.4a)-(3.4c) is then tree-structured and in the form (2.5), so the $O(n)$ algorithm of the previous section can be applied to obtain an explicit ODE for $p$ and $v$ which depends on $\nu(p, v)$.

For the loop-closing constraints, one possibility is a penalty method:

$$\begin{align*}
\hat{p} &= \bar{v}, \quad \text{(3.5a)} \\
M\hat{v} &= f(\bar{p}, \bar{v}) - G^T\bar{x} - \epsilon^{-1}D^TD\bar{v}, \quad \text{(3.5b)} \\
G(\bar{p})\hat{v} &= r(\bar{p}, \bar{v}) \quad \text{(3.5c)}
\end{align*}$$

and

$$\bar{v} = \epsilon^{-1}D\bar{v} \quad \text{(3.6)}$$

where $0 < \epsilon \ll 1$ is the penalty parameter. However, it is well known that we have to use implicit integration schemes for a system such as (3.5) since $\epsilon$ has to be very small and then (3.5) is a stiff system. Therefore we have to invert a matrix such as

$$\begin{pmatrix}
M + \frac{1}{\epsilon}D^TD & G^T \\
G & 0
\end{pmatrix}$$

for a discretization step-size $h$ ($h \geq \epsilon$). Because $M + \frac{1}{\epsilon}D^TD$ is not block diagonal any more, the previous algorithm will be difficult to apply.

In contrast, the SRM allows us to use explicit schemes to solve the regularized problem [4, 5]. Hence, SRM combined with explicit time discretization makes it possible to obtain an $O(n)$ algorithm for problems with many loops.

Applying the SRM to problem (3.4) to treat the loop-closing constraints, we obtain a new algorithm: given $\nu_0(t)$, for $s = 1, 2, \ldots$ find $\{p_s, v_s, \lambda_s, \nu_s\}$ such that

$$\begin{align*}
\hat{p}_s &= v_s - \epsilon^{-1}M^{-1}D^TDp_s, \quad \text{(3.7a)} \\
M\hat{v}_s &= f(p_s, v_s) - G^T(p_s)\lambda_s - D^Tv_s, \quad \text{(3.7b)} \\
G(p_s)\hat{v}_s &= r(p_s, v_s), \quad \text{(3.7c)}
\end{align*}$$

where

$$\nu_s = \nu_{s-1} + \epsilon^{-1}Dv_s. \quad \text{(3.7d)}$$

For each step $s$ the algorithm of the previous section is used to obtain an ODE for $p_s$ and $v_s$ which is integrated using an explicit time discretization scheme. The stabilization of the constraints (3.3c) into (3.4c) prevents excess drift of these constraints in (3.7).

An even simpler algorithm was suggested in [5] for general multibody systems without singularities. For (3.3), it has the form:

$$\begin{align*}
\hat{p}_s &= v_s - \epsilon^{-1}M^{-1}G^Tg(p_s) - \epsilon^{-1}M^{-1}D^TDp_s, \quad \text{(3.8a)} \\
M\hat{v}_s &= f(p_s, v_s) - G^T(p_s)\lambda_s - D^Tv_s, \quad \text{(3.8b)}
\end{align*}$$

and

$$\lambda_s = \lambda_{s-1} + \epsilon^{-1}G(p_s)v_s, \quad \nu_s = \nu_{s-1} + \epsilon^{-1}Dv_s. \quad \text{(3.8c)}$$
This form has $O(n)$ computational complexity and it is simpler than the form (3.7) since we only need to invert the block diagonal matrix $M$. However, for problems with singularities (i.e. rank deficiency, or even nonsingular cases close to rank deficiency) the form (3.8) is not recommended [5]. Also, it is indicated in [5] that for the form (3.8) the best choice of $\epsilon$ is generally $ch$ (because of stability of the explicit difference schemes), where $h$ is the step size of the chosen difference scheme and $c$ is a constant dependent on the eigenvalues of the matrix $\begin{pmatrix} G \\ D \end{pmatrix} M^{-1} \begin{pmatrix} G \\ D \end{pmatrix}^T$. Hence, the best choice of $\epsilon$ for this form changes in time $t$, even for a constant $h$. We give an example in §4 which demonstrates that (3.7) is the overall preferred choice. In comparison with the usual stabilization methods, an additional advantage of (3.7) is that we never invert a singular matrix even if $\begin{pmatrix} G \\ D \end{pmatrix}$ is singular, since $G$ is assumed to have full rank.

Next, we discuss the convergence of the SRM and prove that the iteration number of the method is independent of the body count $n$. Hence, per time step our method is an $O(n)$ algorithm even for chains with many closed loops.

### 3.2 Convergence of the stabilization-SRM form

Here we analyze the convergence of the iterative procedure (3.7). The method of analysis is related to that in [5], but there are important differences: here we only apply the SRM to part of the constraints (i.e. the loop-closing constraints (3.1)), and the problem we consider may have a large number of unknowns (linear in $n$). These differences result in significant complications in the analysis, mainly in deriving careful estimates for $v_s$ and $p_s$. As in [5], we consider only the nonsingular case. We have no complete proof for the singular case, although our method seems to work well for practical cases such as Example 4.1 (see Table 4.2).

The system (3.7) is clearly singularly perturbed for $0 < \epsilon \ll 1$. Starting from an arbitrary $\nu_0(t)$ we may therefore expect an initial layer. For simplicity of the convergence analysis we assume

**Assumption 3.1**

$$
\nu_0(0) = \nu(0), \quad \nu'_0(0) = \nu'(0), \ldots, \nu^{(l)}_0(0) = \nu^{(l)}(0),
$$

where $l = -1$ if $\nu_0(0) \neq \nu(0)$.

For initial value problems it is possible to obtain the exact $\nu^{(j)}(0)$, $j = 0, 1, 2, \ldots$ in advance [4, 5]. For a general semi-explicit DAE, under this assumption there are no initial layers for the solutions of (3.7) up to the $l$th derivatives.

Now let us make assumptions on the boundedness of the solution. For an $n$-body chain, it is reasonable to assume that the generalized positions $p$, velocities $v = \dot{p}$ and their derivatives are bounded linearly in $n$. In other cases the solution may be bounded independently of $n$. We make corresponding assumptions on $p_s$ and $v_s$, since our regularization is a nearby problem to the original system and there will be no initial layers involved in the solution under the condition (3.9) (letting $l \geq 0$).
Assumption 3.2 The solutions $p_s$ and $v_s$ of (3.7) satisfy

\[ \|p_s\|, \|v_s\|, \|\dot{v}_s\| \leq \kappa \]  \hspace{1cm} (3.10a)

\[ \|\ddot{p}_s\|, \|\ddot{v}_s\| \leq \kappa \]  \hspace{1cm} (3.10b)

where $\kappa$ may depend on $n$.

From (3.7b) we can solve for $\lambda_s$ in terms of $p_s$, $v_s$ and $\dot{v}_s$, i.e. we can write\textsuperscript{3}

\[ \lambda_s = \Lambda(p_s, v_s, \dot{v}_s). \]  \hspace{1cm} (3.11)

Below we will also assume:

Assumption 3.3 Further to (3.10), the Jacobian matrix of $\Lambda$ with respect to $p_s$, $v_s$ and $\dot{v}_s$ satisfies

\[ \left\| \frac{\partial \Lambda}{\partial (p_s, v_s, \dot{v}_s)} \right\| \leq K. \]  \hspace{1cm} (3.12)

Hereinafter, $\| \cdot \|$ represents the maximum norm and $K$ is a generic positive constant independent of $n$.

Then (3.10) and (3.12) imply that

\[ \|\tilde{\lambda}_s\| \leq K \kappa. \]  \hspace{1cm} (3.13)

In Remark 3.1 below we give convergence bounds for the case where we only assume that (3.10a) without (3.12) holds, in which case $\lambda_s$ is bounded according to (3.11).

We have noted that the mass matrix $M$ is positive definite and block-diagonal and that the closing-loop constraint matrix $D$ is block-sparse. From the examples in §4, we can see that the Jacobian matrix $G$ for the constraints (2.1) is block-sparse too. In fact, the closing-loop constraints matrix $D$ is not only block-sparse but also block-row-orthogonal. More concretely, there is at most one nonzero element in each column (see, e.g. (4.6) in §4). We thus assume:

Assumption 3.4 The matrices $M$, $D$ and $G$ and their finite multiplications (i.e. the number of multiplications is independent of $n$) are all block-sparse. More precisely, if we multiply such a matrix and a vector with norm $O(n^k)$, then the product norm is still $O(n^k)$, where $k$ is any positive number. In $D$ there is at most one nonzero element in each column.

Combining the properties of $M$ and $D$ we thus obtain that $DM^{-1}D^T$ is positive definite and block-diagonal. Hence the eigenvalues of the matrix $DM^{-1}D^T$ are the union of the eigenvalues of diagonal blocks of the matrix. Thus all these eigenvalues are positive and independent of $n$ since the size and elements of diagonal blocks of $DM^{-1}D^T$ are independent of $n$. We write this as a lemma:

\textsuperscript{3}Let $D$ be an $\tilde{m} \times \tilde{n}$ matrix, $G$ an $\tilde{l} \times \tilde{n}$ matrix, and $\tilde{m} + \tilde{l} < \tilde{n}$. Let $\text{Range}(V)$ be the null space of $D$, where $V$ is an $\tilde{n} \times (\tilde{n} - \tilde{m})$ matrix. Then $DV = 0$. Multiplying (3.7b) by $V^T$ we have

\[ V^T M \dot{v}_s = V^T f - V^T G^T \lambda_s, \]

where $V^T G^T$ is an $(\tilde{n} - \tilde{m}) \times \tilde{l}$ matrix. It has full column rank, because we are assuming that $\begin{bmatrix} G \\ D \end{bmatrix}$ is nonsingular in this section. Hence, we can solve for $\lambda_s$ to obtain (3.11).
Lemma 3.1 If the mass matrix \( M \) and the closing-loop constraint matrix \( D \) have the above properties then the eigenvalues of the matrix \( DM^{-1}D^T \) are all positive and independent of \( n \).

We finally assume the following perturbation bound (cf. the perturbation index [11] and corresponding discussion in [5]).

Assumption 3.5 Let \( \hat{z} = (\hat{p}, \hat{v}, \hat{\lambda}, \hat{\nu})^T \) and \( z = (p, v, \lambda, \nu)^T \). Then there exists a constant \( K \) independent of \( n \) such that:

\[
\| \hat{z} - z \| \leq K (\| \delta \| + \| \delta \| + \| \theta \| + \| \theta \|),
\]

where \( z \) is the solution of (3.4) and \( \hat{z} \) satisfies the perturbed (3.4):

\[
\begin{align*}
\hat{p} &= \hat{v} + \theta(t), \\
M\hat{v} &= f(\hat{p}, \hat{v}) - G^T(\hat{p})\hat{\lambda} - D^T\hat{\nu}, \\
G(p)\hat{v} &= r(\hat{p}, \hat{v}), \\
D\hat{p} &= \delta(t).
\end{align*}
\]

Here, although inverting \( \begin{pmatrix} M & G^T \\ G & 0 \end{pmatrix} \) requires \( O(n) \) operations (see the algorithm in §2), these operations will not involve the perturbations \( \theta(t) \) and \( \delta(t) \). So we believe the bound \( K \) in (3.14) to be independent of \( n \).

Now we can consider the convergence of the stabilization-SRM form (3.7). We choose \( \nu_0 \) such that \( \nu_0 \) and \( \nu_0 \) are of \( O(\kappa) \). Let \( u_s = Dp_s \) and \( w_s = Dv_s \). Then we have drift equations:

\[
\begin{align*}
\dot{u}_s &= D\hat{p}_s = Dv_s - \epsilon^{-1}DM^{-1}D^Tu_s = w_s - \epsilon^{-1}DM^{-1}D^Tu_s, \\
\dot{w}_s &= D\hat{v}_s = DM^{-1}f - DM^{-1}G^T\lambda_s - DM^{-1}D^Tv_{s-1} - \epsilon^{-1}DM^{-1}D^Tw_s
\end{align*}
\]

or

\[
\begin{align*}
\epsilon\dot{u}_s + DM^{-1}D^Tu_s &= \epsilon w_s, \quad (3.17a) \\
\epsilon\dot{w}_s + DM^{-1}D^Tw_s &= \epsilon DM^{-1}(f - G^T\lambda_s - D^Tv_{s-1}), \quad (3.17b)
\end{align*}
\]

with the initial conditions \( u_s(0) = 0, w_s(0) = 0 \). Applying (3.17) for \( s = 1 \) and using the Assumptions and Lemma 3.1 we obtain \( u_1 = O(\kappa^2) \), \( w_1 = O(\kappa) \). Then (3.17a) further yields \( u_1 = O(\kappa^2) \) and \( \dot{u}_1 = O(\kappa) \). Comparing (3.15) with the stabilization-SRM form, we must bound

\[
\delta = u_1, \quad \theta = -\epsilon^{-1}M^{-1}D^Tu_1,
\]

and their derivatives appearing in (3.14). We already have that

\[
\delta = O(\kappa^2), \quad \dot{\delta} = O(\kappa), \quad \theta = O(\kappa).
\]

From (3.17a) we obtain \( \dot{u}_1(0) = 0 \). Using the condition \( \nu_0(0) = \nu(0) \) gives

\[
DM^{-1}[f - G^T\lambda_1 - D^Tv_0]|_{t=0} = 0,
\]

10
where $\lambda_1(0)$ can be obtained from (3.7b) and (3.7d). So, from (3.17b), $\hat{w}_1(0) = 0$. Differentiating (3.17b) we have

$$\epsilon \hat{w}_1 + DM^{-1}D^T \hat{w}_1 = O(\kappa \epsilon), \quad \hat{w}_1(0) = 0,$$

where we use $\nu_1 = O(\kappa)$ obtained from (3.7d) and $w_1 = O(\kappa \epsilon)$ and then $\hat{\lambda}_1 = O(\kappa)$ obtained from (3.13). Hence $\hat{w}_1 = O(\kappa \epsilon)$. Differentiating (3.17a) we next have

$$\epsilon \hat{v}_1 + DM^{-1}D^T \hat{v}_1 = O(\kappa \epsilon^2), \quad \hat{v}_1(0) = 0,$$

so $\hat{v}_1 = O(\kappa \epsilon^2)$. This implies

$$\hat{\theta} = O(\kappa \epsilon), \quad \hat{\delta} = O(\kappa \epsilon).$$

We thus use (3.14) and obtain the desired conclusion for $s = 1$:

$$p_1 = p + O(\kappa \epsilon), \quad \nu_1 = v + O(\kappa \epsilon), \quad \lambda_1 = \lambda + O(\kappa \epsilon), \quad \nu_1 = v + O(\kappa \epsilon). \quad (3.18)$$

Subtracting (3.7a), (3.7b) from (3.4a), (3.4b), respectively, and using (3.18) and (3.12) we have

$$\hat{p}_1 = \hat{p} + O(\kappa \epsilon), \quad \hat{v}_1 = \hat{v} + O(\kappa \epsilon), \quad \hat{\lambda}_1 = \hat{\lambda} + O(\kappa \epsilon). \quad (3.19)$$

Also, by differentiating (3.7d), we conclude that $\hat{v}_1$ is of $O(\kappa)$.

For $s = 2$, we first have as for the $s = 1$ case

$$u_2, \quad \hat{w}_2 = O(\kappa \epsilon^2), \quad w_2, \quad \hat{w}_2, \quad \hat{w}_2 = O(\kappa \epsilon),$$

and hence also

$$p_2 = p + O(\kappa \epsilon), \quad \nu_2 = v + O(\kappa \epsilon), \quad \lambda_2 = \lambda + O(\kappa \epsilon), \quad \nu_2 = v + O(\kappa \epsilon).$$

The estimate (3.19) also holds for $s = 2$. This yields that the right hand side of (3.17b) is $O(\kappa \epsilon^2)$, so

$$w_2 = O(\kappa \epsilon^2), \quad w_2 = O(\kappa \epsilon^3).$$

Now we want to get a better estimate for $\hat{w}_2$. Differentiating (3.17b) we have

$$\epsilon \hat{w}_2 + DM^{-1}D^T \hat{w}_2 = O(\kappa \epsilon^2) + \epsilon \hat{F}(p_2, \nu_2, \lambda_2, \nu_1) \quad (3.20)$$

and $\hat{w}_2(0) = 0$ obtained as for the $s = 1$ case. Here

$$F(p_2, \nu_2, \lambda_2, \nu_1) := DM^{-1}[f - G^T \lambda_2 - D^T \nu_1]. \quad (3.21)$$

We want to show that $\hat{F} = O(\kappa \epsilon)$. For this purpose we must estimate $\hat{w}_2$ first. Using the condition $\hat{v}_0(0) = \hat{v}(0)$, and the fact that $p_1$, $\nu_1$, $\hat{p}_1$, $\hat{v}_1$, $\lambda_1$, $\hat{\lambda}_1$ are also exact at $t = 0$, we can obtain

$$\hat{F}(p_1, \nu_1, \lambda_1, \nu_0)|_{t=0} = 0.$$

Hence $\hat{w}_1(0) = 0$ from (3.20). Differentiating (3.20) again we now obtain precisely as when estimating $\hat{w}_1$ above,

$$\hat{w}_1 = O(\kappa \epsilon).$$

11
Here we need the estimates $\tilde{p}_1, \tilde{v}_1, \tilde{\lambda}_1 = O(\kappa)$ which can be obtained from a differentiation of (3.7), applications of (3.10), (3.12) and (3.13), and the estimates for $\tilde{w}_1$ obtained above. Noting

$$\tilde{w}_1 = F(p_1, v_1, \lambda_1, \nu_1)$$

we thus have

$$F(p_2, v_2, \lambda_2, \nu_1) = F(p_2, v_2, \lambda_2, \nu_1) - F(p_1, v_1, \lambda_1, \nu_1) + \tilde{w}_1.$$  

Our previous estimates allow the conclusion that

$$(p_2, v_2, \lambda_2, \tilde{p}_2, \tilde{v}_2, \tilde{\lambda}_2)^T = (p_1, v_1, \lambda_1, \tilde{p}_1, \tilde{v}_1, \tilde{\lambda}_1)^T + O(\kappa \epsilon),$$

hence we can conclude that $\tilde{F} = O(\kappa \epsilon)$ and obtain

$$p_2 = p + O(\kappa \epsilon^2), \quad v_2 = v + O(\kappa \epsilon^2), \quad \lambda_2 = \lambda + O(\kappa \epsilon^2), \quad v_2 = v + O(\kappa \epsilon^2).$$  \hspace{1cm} (3.22)

Repeating this procedure, we can finally obtain:

**Theorem 3.1** Let all the assumptions described at the beginning of this section hold. Then, for the solution of the stabilization-SRM form (3.7) with $v_0$ and $\dot{v}_0$ bounded by $O(\kappa)$, we have the following error estimates:

$$p_s = p + O(\kappa \epsilon^s), \quad v_s = v + O(\kappa \epsilon^s), \quad \lambda_s = \lambda + O(\kappa \epsilon^s), \quad v_s = v + O(\kappa \epsilon^s).$$  \hspace{1cm} (3.23)

for $1 \leq s \leq l + 1$ and $0 \leq t \leq t_f$. Here $(p, v, \lambda, \nu)^T$ is the solution of the multibody problem (3.3).

**Remark 3.1** If the bounds in (3.10) are only assumed for $p_s, v_s$ and $\dot{v}_s$, then we may generally obtain $p_s, \dot{v}_s, \ddot{v}_s = O(\kappa^2)$ and the $j$-th derivatives would be $O(\kappa^{j+1})$. This finally yields the following error estimates:

$$p_s = p + O(\kappa^j \epsilon^s), \quad v_s = v + O(\kappa^j \epsilon^s), \quad \lambda_s = \lambda + O(\kappa^j \epsilon^s), \quad v_s = v + O(\kappa^j \epsilon^s).$$  \hspace{1cm} (3.24)

This result is obviously weaker than (3.23), corresponding to weaker boundedness assumptions.

### 3.3 Computational complexity of the stabilization-SRM form

Consider our algorithm for (3.7). At each regularization iteration we can use the $O(n)$ algorithm described in §2. So, to consider the computational complexity of our algorithm we need only study the number of iterations $s$ required.

It is simple to show that $s$ is independent of $n$. Given the worst error estimate (3.24), we must choose $\epsilon$ small enough so that

$$\kappa \epsilon \leq \alpha < 1.$$ 

Then each SRM iteration reduces the error, viz. the difference between the solution of (3.7) and the solution of (3.3), by at least a factor $\alpha$, and so a fixed number of iterations $s$, independent of $n$, is needed to reduce this error below any given tolerance.
Note, though, that $\kappa$ may grow with $n$, depending on the problem being simulated. Hence the range of choices for $\epsilon$ ($\epsilon \leq \alpha/\kappa$) is restricted depending on $n$. Since the time discretization scheme is explicit, the step size $h$ must be restricted by absolute stability requirements to satisfy

$$h \leq \gamma \epsilon$$

for an appropriate constant $\gamma$ of moderate size. The number of time steps required may therefore depend on $n$, too (cf. [21])$^4$. Still, the number of iterations $s$ required to obtain a given accuracy obviously remains independent of $n$, hence operation count per time step remains $O(n)$.

For the worst error estimate (3.24) suppose that

$$\kappa^{1-\beta} \leq 1,$$  \hspace{1cm} (3.25)

where $\beta$ is a given positive constant, $0 < \beta < 1$, and take $\epsilon = h$ for simplicity. Now let us apply an $r$-th order explicit difference scheme to the stabilization-SRM form. At the $s$-th iteration, the worst combined error for our algorithm is $O(\kappa^s \epsilon^s) + O(h^r)$ (see Remark 3.1). Trying to roughly equate the two sources of error, we set

$$(\kappa \epsilon)^s = h^r,$$

hence

$$s = \frac{-\ln h^r}{-\ln(\kappa \epsilon)}.$$  \hspace{1cm} (3.26)

Using (3.25) we then obtain a rough upper bound for the number of iterations $s$:

$$s \leq \frac{-r \ln h}{-\beta \ln h} = \frac{r}{\beta}.  \hspace{1cm} (3.26)$$

**Remark 3.2** For the error estimate (3.23) the condition (3.25) can be weakened significantly to read

$$\kappa^{s_0 - \beta} \leq 1,$$  \hspace{1cm} (3.27)

where $s_0$ is an arbitrary integer independent of $n$. In this case, (3.26) is replaced by

$$s \leq \frac{r}{\beta} s_0.$$  \hspace{1cm} (3.28)

### 4 Numerical experiments

We now present a couple of examples to demonstrate the efficacy of the algorithm that was proposed and analyzed in the previous sections. At first, we build up the system for a special type of $n$-body chains (see Figure 2.1) which include our two examples. We use essentially the method described in [21].

Consider a chain consisting of $n$ bodies. Each body is modeled as a line segment of length $l_j$ and mass $m_j$, with uniform mass distribution. We choose Cartesian coordinates of the bodies' centers of mass, $x_j$, and the vectors connecting the joints along the links, $d_j$, in order

$^4$A requirement such as $h = O(\kappa^{-1})$ is likely to arise also from accuracy, not only stability considerations.
to describe the position $p$ of the chain: $p = (p_1, p_2, \ldots, p_n)$, where $p_j = (x_j, d_j)$, $j = 1, \ldots, n$. For 3-D chains $p_j$ has six components, and for 2-D chains it has four components. The labeling of the chain bodies and joints has been shown in Figure 2.1 and explained in §2. Hence the holonomic constraints ($g(p) = 0$) for a tree-structured system include length conditions

$$||d_j||^2 = l_j^2$$  \hspace{1cm} (4.1)

and connection conditions

$$x_i + \frac{1}{2}d_i = x_j - \frac{1}{2}d_j$$  \hspace{1cm} (4.2)

for each edge $(i, j)$, i.e., for each pair such that $i =$ father of $j$. Therefore

$$g(p) = \left( \begin{array}{c} g_1 \\ \vdots \\ g_n \end{array} \right), \quad g_j = \left( \begin{array}{c} \frac{1}{2}(||d_j||^2 - l_j^2) \\ x_j - \frac{1}{2}d_j - x_i - \frac{1}{2}d_i \end{array} \right).$$  \hspace{1cm} (4.3)

Due to the uniform mass distribution, the center of mass of each body coincides with its geometric center, $x_j$, and the moment of inertia about the center of mass is $\frac{1}{2}m_j(l_j/2)^2$. The total kinetic energy is given by

$$T = \sum_{j=1}^{n} \left[ \frac{1}{2}m_j||\dot{x}_j||^2 + \frac{1}{3}m_j(l_j/2)^2||\dot{d}_j||^2 \right]$$

$$= \sum_{j=1}^{n} \left[ \frac{1}{2}m_j \left( \begin{array}{c} \dot{x}_j \\ \dot{d}_j \end{array} \right)^T \left( \begin{array}{cc} I & 0 \\ 0 & \frac{1}{12}I \end{array} \right) \left( \begin{array}{c} \dot{x}_j \\ \dot{d}_j \end{array} \right) \right] = \frac{1}{2}\hat{p}^T M \hat{p},$$

where

$$M = \text{diag}(M_1, \ldots, M_n), \quad M_j = m_j \left( \begin{array}{cc} I & 0 \\ 0 & \frac{1}{12}I \end{array} \right).$$  \hspace{1cm} (4.4)

The potential energy due to gravity is given by

$$V = \sum_{j=1}^{n} m_j g e_v^T x_j.$$  

Here $e_v$ is the unit vector along the vertical axis. For the 2-D case $e_v = (0, 1)^T$ and for the 3-D case $e_v = (0, 0, 1)^T$. Note that $e_v^T x_j$ is the height of the center of mass of body $j$ above zero level, and $\hat{g} \approx 9.81 \text{ m/s}^2$ is the gravity constant. This gives the force vector

$$f = \left( \begin{array}{c} f_1 \\ \vdots \\ f_n \end{array} \right), \quad f_j = -m_j \hat{g} \left( \begin{array}{c} e_v \\ 0 \end{array} \right).$$  \hspace{1cm} (4.5)

Hence, we know the mass matrix $M$, the force term $f$ and the constraints $g$ and can write down the system (2.5) for this kind of tree-structured chain problems. For chains with loops, we only need to impose some additional geometric constraints onto their corresponding tree-structured formulation (see the discussion at the beginning of §3).

Next we consider two specific examples.
Example 4.1 Consider a 2D square chain with unit length and mass for each link (i.e. $m_j = l_j = 1$). The labeling of a corresponding tree structure is shown in Figure 4.1 (where the point $O$ is fixed and chosen as the origin). Under this labeling,

$$i = \text{father of } j = \begin{cases} \text{root} & \text{if } j = 1 \text{ or } 2 \\ j - 3 & \text{if } \text{mod}(j, 3) = 2 \\ j - 2 & \text{otherwise} \end{cases}.$$ 

Hence we can write down the Jacobian matrix $G(p)$ for the tree-structure constraints $g(p)$. For example, when $n = 7$,

$$G(p) = \begin{pmatrix} G_1 & P & G_2 & P & G_3 & P & G_4 \\ P & G_5 & P & G_6 & P & G_7 & \end{pmatrix}, \text{ where } G_j = \begin{pmatrix} 0 & d_j^T \\ I & -\frac{1}{2} I \end{pmatrix} \text{ and } P = \begin{pmatrix} 0 & 0 \\ -I & -\frac{1}{2} I \end{pmatrix}. $$

Here $I$ is a $2 \times 2$ unit matrix. The extra loop-closing constraints $d(p) = Dp = 0$ are:

$$x_j + \frac{1}{2} d_j - x_{j+1} - \frac{1}{2} d_{j+1} = 0, \text{ if } j = 3k, k = 1, 2, \ldots, m,$$

where $m = (n - 1)/3$ is equal to the number of squares. For example, when $n = 7$ we have $m = 2$ closed loops and the $2m \times 4n$ matrix $D$ is given by

$$D = \begin{pmatrix} 0 & 0 & 0 & 0 & I & \frac{1}{2} I & -I & -\frac{1}{2} I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & \frac{1}{2} I & -I & -\frac{1}{2} I \end{pmatrix}. \quad (4.6)$$

The Jacobian matrix $\left( \begin{pmatrix} G(p) \\ D \end{pmatrix} \right)$ of all constraints has rank deficiency when all four links of a square are on a line. We let the chain fall freely from the position shown in Figure 4.1 where the joint “0” is fixed. For each SRM iteration our algorithm (3.7) solves a tree-structured problem using the $O(n)$ algorithm of §2. Here, we only demonstrate that the number of SRM iterations is independent of $n$. This means that the computational complexity of our algorithm is indeed $O(n)$ per time step. We choose step size $h = .001$ and regularization parameter $\epsilon = 5h$, and apply an explicit second-order Runge-Kutta scheme to the regularized problem at each iteration. We do the computations for $t = 0.4$ at first, because to clearly see a relation between the number of iterations and $n$ we want to avoid singularities which occur around and after $t = 0.5$ and whose error situation is very complicated. We count
Table 4.1: Relation between the number of iterations and $n$ for the square chain

<table>
<thead>
<tr>
<th>$n$</th>
<th>7</th>
<th>19</th>
<th>49</th>
<th>79</th>
<th>100</th>
<th>499</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td># of iterations</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>max drift</td>
<td>6.650e-6</td>
<td>2.906e-6</td>
<td>2.589e-6</td>
<td>1.821e-6</td>
<td>6.541e-6</td>
<td>1.561e-5</td>
<td>1.457e-5</td>
</tr>
<tr>
<td># of iterations</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>max drift</td>
<td>6.650e-6</td>
<td>2.906e-6</td>
<td>2.589e-6</td>
<td>4.280e-6</td>
<td>1.528e-5</td>
<td>1.553e-5</td>
<td>1.792e-5</td>
</tr>
</tbody>
</table>

the number of SRM iterations until the errors in the constraints do not exhibit obvious improvement. Table 4.1 lists iteration counts and constraint errors for various $n$.

Theoretically, we expect that two SRM iterations be sufficient, since the difference scheme is second order and $\epsilon = O(h)$. From the table we see that at the second and the third successive iterations the maximum drifts are almost the same. Additional experiments with $h = .01$ and $n = 1000$ also result with the need of only two iterations for the algorithm (3.7) to achieve the second order discrete accuracy.

Next we compare the performance of the algorithms (3.7) (AlgI) and (3.8) (AlgII). We set $n = 7$, where computational results show that the first singularity occurs after $t = 0.5$, and integrate up to $t = 1.0$, thus crossing the singularity. The simple forward Euler scheme is used with $h = .001$ for both algorithms.

Computational results indicate that for the algorithm (3.7) we can still take $\epsilon = 5h$; however, for the algorithm (3.8) a larger $\epsilon$ must be used: the algorithm is unstable immediately after the first time step when we take $\epsilon = .005$. The algorithm blows up around $t = 0.4$ for $\epsilon = .01$, and it becomes stable only for $\epsilon = .05$. This agrees with our observation in §3.1, namely, that for the sake of stability of the difference scheme the smallest $\epsilon$ we can choose using (3.8) depends on $t$ and it is often larger than that needed for the algorithm (3.7). Hence, more iterations are needed for the simpler algorithm (3.8).

Table 4.2: Maximum drifts of two algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th># of iterations</th>
<th>$\epsilon$</th>
<th>$t = .4$</th>
<th>$t = .5$</th>
<th>$t = .8$</th>
<th>$t = 1.0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlgI 1</td>
<td>1</td>
<td>.005</td>
<td>7.763e-3</td>
<td>1.289e-2</td>
<td>1.744e-2</td>
<td>2.959e-2</td>
</tr>
<tr>
<td>AlgI 2</td>
<td>2</td>
<td>.05</td>
<td>7.763e-3</td>
<td>1.289e-2</td>
<td>1.744e-2</td>
<td>3.107e-2</td>
</tr>
<tr>
<td>AlgII 2</td>
<td>2</td>
<td>.05</td>
<td>1.008e-1</td>
<td>1.640e-1</td>
<td>2.270e-1</td>
<td>4.794e-1</td>
</tr>
<tr>
<td>AlgII 4</td>
<td>4</td>
<td>.05</td>
<td>5.869e-2</td>
<td>8.514e-2</td>
<td>1.129e-1</td>
<td>1.155e-1</td>
</tr>
</tbody>
</table>

We list in Table 4.2 the maximum drifts produced by these two algorithms at various times. From the table we see that the overall performance of AlgI is better than that of AlgII. Also, for this example the error improvement of AlgII per iteration is much slower. The motion of this square chain with $n = 7$ from $t = 0$ to $t = 2.2$ is described in Figure 4.2.

The run time required for our method in this example is obviously $O(n)$ per time step. We next perform some computations using the package MEXX [18], to measure its running
Figure 4.2: Motion of the square chain. Time increases in increments $\Delta t = 0.2$ from left to right, top to bottom.

time per time-step for this example as a function of $n$. Note that a head-to-head comparison of run-times between a method, as we have proposed here, and a complete package such as MEXX, is not very meaningful. Moreover, the major contribution of our method is to provide a fast algorithm for special problems (i.e. large with many closed loops), so specialized rather than general-purpose implementations may be more suitable. Our purpose here is to investigate the complexity of a typical package such as MEXX per time step for this example and to contrast this with that of the SRM.

We have used the most appropriate linear algebra version that this code offers (block storage for both $M$ and $G$, utilizing the efficient algorithm for tree-structures). We have also employed various techniques to ensure that what we measure is not simply noise such as increased time-sharing or paging activity\footnote{We have utilized runs on different machines, including an SGI Origin 2000 with 0.5 GB RAM and smaller SUN and SGI workstations. We also averaged times over a few time steps and made just one-step runs to ignore some error control activities. The results in all cases were qualitatively similar.}. A typical result is displayed in Figure 4.3. The increase in run time per step is obviously nonlinear for MEXX. Employing two SRM iterations of algorithm (3.7) with an explicit second-order (two-stage) Runge-Kutta scheme, on the other hand, gives a linear growth in run time. The actual timing of one step of this simple scheme is roughly $\frac{1}{4}$ that of MEXX for 16 bodies, and it becomes roughly $\frac{1}{10}$ that of MEXX for 196 bodies.

\hfill $\Box$

\textbf{Example 4.2} In this example we consider the motion of a square net (veil) started by a breeze. The corresponding tree structure of the net and its labeling are shown in Figure 4.4, where $0_i = (i - 1, 0)$, $i = 1, 2, \ldots, w$, are fixed joints. Under this labeling, we can determine the father of any given body and then the Jacobian matrix $G(p)$ for the constraints $g(p) = 0$. 

\vfill

17
Figure 4.3: Running times per time step as a function of number of bodies for the square chain example when using (i) MEXX and (ii) two SRM iterations with a two-stage Runge-Kutta scheme. Note that the time increases nonlinearly for MEXX and linearly for the SRM.

\[ l = (w-1)l \]

\[ l = (w-2)l \]

\[ l = (w-1)l \]

\[ l = (w-2)l \]

Figure 4.4: Tree structure and its labeling of the square net
The extra loop-closing constraints are

\[ d_i(p) = \begin{cases} 
    x_{2i} + \frac{1}{2}d_{2i} - x_{2i+2l-1} - \frac{1}{2}d_{2i+2l-1}, & i = 1, \ldots, l(w - 2), \\
    x_{2i} + \frac{1}{2}d_{2i} - x_{i+w} - \frac{1}{2}d_{i+w}, & i = l(w - 2) + 1, \ldots, l(w - 1).
\]

For this square net the number of bodies is \( n = 2l(w-1) + 1 \). Again we set the step size \( h = .001 \) and the regularization parameter \( \epsilon = 5h \), and apply at each time step an explicit second-order Runge-Kutta scheme for the regularized problem of each SRM iteration (3.7). We consider the motion of a net whose initial position forms an angle \( \frac{\pi}{8} \) with the vertical line. The number of iterations for various \( n \) at \( t = 0.5 \) is listed in Table 4.3.

<table>
<thead>
<tr>
<th>( n ) (l,w)</th>
<th>10</th>
<th>36</th>
<th>65</th>
<th>91</th>
<th>207</th>
<th>495</th>
<th>975</th>
</tr>
</thead>
<tbody>
<tr>
<td># of iterations</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>max drift</td>
<td>1.790e-7</td>
<td>8.677e-8</td>
<td>7.040e-8</td>
<td>6.060e-8</td>
<td>6.662e-6</td>
<td>2.221e-5</td>
<td>2.272e-5</td>
</tr>
<tr>
<td># of iterations</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>max drift</td>
<td>1.790e-7</td>
<td>8.677e-8</td>
<td>1.125e-7</td>
<td>1.031e-7</td>
<td>6.658e-5</td>
<td>2.389e-4</td>
<td>3.320e-4</td>
</tr>
</tbody>
</table>

Table 4.3: Relation between the number of iterations and \( n \) for the square net

We observe no drift error improvement after the second iteration. So only two iterations are again needed for the algorithm (3.7), independent of \( n \).

The motion of this square chain with \( n = 91 \) (\( l = w = 7 \)) from \( t = 0 \) to \( t = 2.2 \) is depicted in Figure 4.5.

References


Figure 4.5: Motion of the square net with \( l = w = 7 \). Time increases in increments \( \Delta t = .2 \) from left to right, top to bottom.


