METHODS AND APPLICATIONS OF ANALYSIS.
Vol. 7, No. 3, pp. xxx–xxx, September 2000

ANALYSIS OF THE SHAKE-SOR ALGORITHM FOR CONSTRAINED MOLECULAR DYNAMICS SIMULATIONS*

DEXUAN XIE†, L. RIDGWAY SCOTT‡, AND TAMAR SCHLICK§

Abstract. Molecular dynamics integration with bonds constrained to equilibrium values is a common approach used to increase the feasible timestep and hence reduce the overall simulation time. Here we analyze the widely used numerical iterative scheme for constrained molecular dynamics simulations, SHAKE, in a general algorithmic framework, from which SHAKE's relationship to nonlinear solvers can be established. Using the nonlinear SOR-Newton iterative method, we define an accelerated variant of SHAKE, called SHAKE-SOR, and prove a fundamental relationship between SHAKE-SOR and SOR-Newton. Based on this relationship, the convergence of SHAKE-SOR is proved in the framework of nonlinear SOR theory. Numerical results show that SHAKE-SOR can significantly improve the performance of standard SHAKE by reducing the number of iterations per timestep through an optimal parameter choice.

1. Introduction. Molecular dynamics (MD) simulations constitute an important tool for exploring molecular motions of chemical and biological systems [1, 16]. In the underlying models, macromolecules are represented as chains of atoms linked by covalent bonds. Such bonds vibrate at a very high frequency, so that a typical timestep for the MD simulation is very small – around one femtosecond (10^{-15} seconds) – thereby limiting severely the total simulation time. Many effective techniques have been proposed to increase the timestep (see recent reviews in [16, 17, 18]). One of the simplest techniques is to freeze the high-frequency vibrational motions by imposing algebraic constraints that fix the bond lengths. The algebraic constraints have little influence on the underlying dynamics in most cases [7] and can increase the timestep by a factor of two or three, with minimal added cost per step. The cost involved is solution of an auxiliary set of nonlinear equations at each timestep.

Ryckaert *et al.* [13] described a scheme termed SHAKE for MD simulations subject to a set of bond-constraint equations based on the Verlet discretization [21]. Since then, SHAKE has become a widely used algorithm in biomolecular simulations. To improve the stability of the Verlet discretization, SHAKE has been adapted to the velocity Verlet scheme, leading to the so called RATTLE algorithm [2]. SHAKE has also been extended to handle general constraints (such as of other internal variables) [14]. An accelerated version of SHAKE based on the nonlinear successive over relaxation (SOR) method [12] has been proposed in [3, 22].

Different approaches to analyze the convergence of SHAKE can be found in [3, 15, 22]. A recent attractive approach for analyzing SHAKE within the framework of nonlinear SOR theory [12] was described in [3]. That work also led to a practical improvement of SHAKE. However, the theoretical relationship between SHAKE and the Gauss-Seidel-Newton (GSN) method [12] was incomplete due to over simplifications in eqns. (23) through (26) of [3] (see details in footnote [24] here). Here we use a slightly different approach to couch the SHAKE process in a mathematical nonlinear

*Received November 5, 1999.
†Department of Mathematics, Graduate Program in Scientific Computing, University of Southern Mississippi, Hattiesburg, MS 39406-5054, USA (Dexuan.Xie@usm.edu).
‡Departments of Computer Sciences and Mathematics, Computation Institute, University of Chicago, Chicago, IL 60637, USA (ridg@uchicago.edu).
§Departments of Chemistry, Mathematics, and Computer Science, Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012, USA (schlick@nyu.edu).

iteration framework, completing the proof in [3] by adding the conditions under which the equivalence of a SHAKE variant and nonlinear SOR holds.

We first formulate the SHAKE algorithm according to standard routines used in the packages CHARMM [4, 9] and GROMOS [6]; these are two widely-used molecular mechanics and dynamics programs. We then show that the SHAKE iterates $\{\mathbf{r}^{(m)}\}$ generated by the SHAKE algorithm can be formulated in the general form

$$\mathbf{r}^{(m+1)} = \mathbf{r}^{(m)} + B\Lambda, \quad m = 0, 1, 2, \ldots, \tag{1.1}$$

where $\mathbf{r}^{(m)}$ approximates the collective position vector of the molecular system at each dynamic step, B is a matrix independent of $\mathbf{r}^{(m)}$, and the vector Λ (a function of $\mathbf{r}^{(m)}$) is defined by using GSN as an approximate solution of the nonlinear constrained equations of motion. The detailed definitions of B and Λ will be given in the next section.

Expression (1.1) invites definitions of variants that can improve the performance of SHAKE (in terms of the convergence at each dynamic step). In fact, if the vector Λ in (1.1) is defined by a more effective nonlinear iterative method than GSN, the resulting scheme can perform better than SHAKE of [13]. In this paper, we define Λ by the SOR-Newton method [12], leading to the same accelerated variant of SHAKE proposed in [3, 22]. For convenience, we refer to this variant as SHAKE-SOR.

With the general framework of (1.1), we also easily overcome the difficulty that exists in the standard SHAKE algorithm. Namely, SHAKE [13] is thought to fail if the inner product between a reference bond vector and a corresponding updated bond vector is zero. In [4, 6], this case is interpreted as too large a deviation in a SHAKE iteration. This simple treatment to this case seriously affects the robustness of SHAKE because this situation happens occasionally during the SHAKE execution. In this paper, we show that the setting of Λ in SHAKE is essentially an approximate solution of the nonlinear constrained equations of motion; thus, it is natural to select another definition for Λ in the case of the above zero inner product. We propose a modification of Λ that ensures the convergence of SHAKE-SOR.

Noting that SHAKE-SOR includes SHAKE as a special case, we analyze SHAKE-SOR in this paper. We prove the basic relationship between SHAKE-SOR and SOR-Newton, from which the convergence of SHAKE-SOR follows under additional conditions. We derive these conditions, with which the convergence of SHAKE-SOR is equivalent to that of SOR-Newton. These conditions correct the work in [3].

Finally, numerical results using CHARMM [4, 9] are presented showing that SHAKE-SOR can significantly improve the performance of SHAKE. Examples are shown for three biomolecules: the protein BPTI (bovine pancreatic trypsin inhibitor), the protein lysozyme, and a DNA dodecamer (i.e., 12 base pairs). These systems have 582, 2050, and 11510 bond constraints, respectively. Other numerical results that demonstrate the good performance of SHAKE-SOR can also be found in [3, 22].

The remainder of the paper is organized as follows. Section 2 describes the constrained Verlet discretization, and Section 3 presents a practical SHAKE algorithm. Section 4 formulates the SHAKE iterates in the form of (1.1), and Section 5 defines SHAKE-SOR. Section 6 presents and proves the fundamental relationship between SHAKE-SOR and SOR-Newton. Section 7 presents numerical results for SHAKE-SOR, with conclusions following in Section 8. For completeness, the description of the SOR-Newton method for solving the constrained equations of motion is given in the Appendix.

2. The constrained Verlet method. We consider a molecular system of N atoms interacting via a force field $\mathbf{F}(\mathbf{r}(\mathbf{t}))$, where $\mathbf{r}(\mathbf{t})$ denotes the collective position vector of the molecular system at time t, i.e., $\mathbf{r}(t) = (\mathbf{r}_1(t), \mathbf{r}_2(t), \ldots, \mathbf{r}_N(t))^T$ with $\mathbf{r}_i(t)$ being the position vector of atom i in the real space R^3. We often simply write $\mathbf{r}(\mathbf{t})$ and $\mathbf{r}_i(t)$ as $\mathbf{r}$ and $\mathbf{r}_i$ in this paper. The superscript T denotes a vector or matrix transpose.

In constrained MD, a molecular system satisfies the Newtonian equation of motion

$$M\ddot{\mathbf{r}} = \mathbf{F}(\mathbf{r}) \tag{2.1}$$

subject to the algebraic constraint condition

$$g(\mathbf{r}) = 0. \tag{2.2}$$

Here $\ddot{\mathbf{r}}$ is the second derivative of $\mathbf{r}$ with respect to time t, $g(\mathbf{r})$ is a vector function of $\mathbf{r}$, M is the diagonal mass matrix defined by $M = \mathrm{diag}(m_1, m_1, m_1, m_2, m_2, m_2, \ldots, m_N, m_N, m_N)$, and m_i is the mass of atom i. To define g, we consider a set of l specified nonlinear equations for the l bond constraints[1]. That is, the vector function $g(\mathbf{r}) = (g_1, g_2, \ldots, g_l)^T$ has component k defined as

$$g_k(\mathbf{r}) = \|\mathbf{r}_{i_k} - \mathbf{r}_{j_k}\|^2 - d_k^2, \quad k = 1, 2, \ldots, l, \tag{2.3}$$

where k labels the rigid bond connecting atom i_k and atom j_k of length d_k (see Fig. 1), and the norm $\|\cdot\|$ is the standard Euclidean distance norm in R^3.

The system above can be rewritten as unconstrained by introducing the Lagrange multiplier vector $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_l)^T$. The constraint condition (2.2) can then be regarded as an additional force expressed as $-J(\mathbf{r})^T\lambda$, where $J(\mathbf{r})$ is the Jacobian matrix (i.e., the first derivative) of the vector $g(\mathbf{r})$. Thus, system (2.1) together with (2.2) is equivalent to:

$$M\ddot{\mathbf{r}} = \mathbf{F}(\mathbf{r}) - J(\mathbf{r})^T\lambda \tag{2.4}$$

provided that λ is appropriately chosen so that the solution $\mathbf{r}$ of (2.4) satisfies the constraint condition (2.2). In the remainder of this paper, we simply denote the Jacobian matrix $J(\mathbf{r})$ as J.

We approximate (2.4) by the following second-order finite-difference equation:

$$\mathbf{r}(t + \triangle t) = 2\mathbf{r}(t) - \mathbf{r}(t - \triangle t) + \triangle t^2 M^{-1}\mathbf{F}(\mathbf{r}(t)) - \triangle t^2 M^{-1} J(\mathbf{r}(t))^T\lambda, \tag{2.5}$$

where $\triangle t$ is a timestep, and λ is a solution of the following nonlinear equation of λ:

$$g(\hat{\mathbf{r}} - \triangle t^2 M^{-1} J^T\lambda) = 0, \tag{2.6}$$

where

$$\hat{\mathbf{r}} = 2\mathbf{r}(t) - \mathbf{r}(t - \triangle t) + \triangle t^2 M^{-1}\mathbf{F}(\mathbf{r}(t)). \tag{2.7}$$

Together, Eqns. (2.5) and (2.6) are commonly referred to as the constrained Verlet method [10, 21]. Setting $\lambda = 0$ in (2.5) yields the unconstrained Verlet scheme.

To simplify the notation above, we set $\Lambda = -2\triangle t^2\lambda$ and $B = \frac{1}{2}M^{-1}J^T$. Thus, eq. (2.6) becomes

$$g(\hat{\mathbf{r}} + B\Lambda) = 0. \tag{2.8}$$

[1] The algorithm can be generalized to other internal constraints [14].

3. The SHAKE algorithm. We describe a practical SHAKE algorithm as used by both the CHARMM [4, 9] and GROMOS [6] programs. We then present an important modification to SHAKE.

Let $\{\mathbf{r}^{(m)}\}$ be a sequence of SHAKE iterates that approximate the constrained Verlet update $\mathbf{r}(t+\triangle t)$ and set $\mathbf{r} = \mathbf{r}(t)$. The initial guess $\mathbf{r}^{(0)}$ is obtained from the unconstrained Verlet method, namely, $\mathbf{r}^{(0)} = \hat{\mathbf{r}}(t)$, where $\hat{\mathbf{r}}(t)$ is given in (2.7). In the computer implementation, the update $\mathbf{r}^{(m+1)}$ shares the array for $\mathbf{r}^{(m)}$ for $m = 0, 1, 2, \ldots$; hence, only one array $\bar{\mathbf{r}}$ is required for the sequence $\{\mathbf{r}^{(m)}\}$. Suppose that $\bar{\mathbf{r}}$ holds the m-th SHAKE iterate $\mathbf{r}^{(m)}$, i.e., $\bar{\mathbf{r}} \leftarrow \mathbf{r}^{(m)}$. To define the SHAKE iterate $\mathbf{r}^{(m+1)}$, the array $\bar{\mathbf{r}}$ is updated by considering the constraints successively. That is, for $k = 1, 2, \ldots, l$, the k-th component Λ_k of the vector Λ is computed via

$$\Lambda_k = \frac{d_k^2 - \|\bar{\mathbf{r}}_{(k)}\|^2}{2\,(1/m_{i_k} + 1/m_{j_k})\,\mathbf{r}_{(k)}{}^T\,\bar{\mathbf{r}}_{(k)}}, \tag{3.1}$$

where the bond vectors $\mathbf{r}_{(k)}$ and $\bar{\mathbf{r}}_{(k)}$ are computed by $\mathbf{r}_{(k)} = \mathbf{r}_{i_k} - \mathbf{r}_{j_k}$ and $\bar{\mathbf{r}}_{(k)} = \bar{\mathbf{r}}_{i_k} - \bar{\mathbf{r}}_{j_k}$, and the position vectors $\bar{\mathbf{r}}_{i_k}$ and $\bar{\mathbf{r}}_{j_k}$ of the k-th bond are then updated via

$$\bar{\mathbf{r}}_{i_k} \leftarrow \bar{\mathbf{r}}_{i_k} + \frac{1}{m_{i_k}}\mathbf{r}_{(k)}\Lambda_k \quad \text{and} \quad \bar{\mathbf{r}}_{j_k} \leftarrow \bar{\mathbf{r}}_{j_k} - \frac{1}{m_{j_k}}\mathbf{r}_{(k)}\Lambda_k. \tag{3.2}$$

This completes one iteration of SHAKE, and the updated $\bar{\mathbf{r}}$ holds the SHAKE iterate $\mathbf{r}^{(m+1)}$.

A common convergence (or termination) rule for SHAKE iteration is below

$$\|g(\mathbf{r}^{(m)})\|_\infty = \max_{1\le k\le l} |g_k(\mathbf{r}^{(m)})| < \epsilon, \tag{3.3}$$

where ϵ is a small number such as $\epsilon = 10^{-10}$, the default value in CHARMM [4, 9].

Clearly, if the inner product $\mathbf{r}_{(k)}{}^T\,\bar{\mathbf{r}}_{(k)} = 0$ (or small enough in practice), the value of Λ_k defined by formula (3.1) is undefined, leading to the failure of SHAKE. This case happens occasionally in the implementation of SHAKE. To overcome this difficulty, we first prove the following theorem.

THEOREM 3.1. *The value of Λ_k defined by formula (3.1) is the first iterate of the Newton iteration method for solving the k-th constraint equation*

$$\|\bar{\mathbf{r}}_{i_k}^{new} - \bar{\mathbf{r}}_{j_k}^{new}\|^2 - d_k^2 = 0 \tag{3.4}$$

with an initial guess of zero. Here $\bar{\mathbf{r}}_{i_k}^{new}$ and $\bar{\mathbf{r}}_{j_k}^{new}$ denote the updated $\bar{\mathbf{r}}_{i_k}$ and $\bar{\mathbf{r}}_{j_k}$ in (3.2), respectively. That is, $\bar{\mathbf{r}}_{i_k}^{new} = \bar{\mathbf{r}}_{i_k} + \frac{1}{m_{i_k}}\mathbf{r}_{(k)}\Lambda_k$ and $\bar{\mathbf{r}}_{j_k}^{new} = \bar{\mathbf{r}}_{j_k} - \frac{1}{m_{j_k}}\mathbf{r}_{(k)}\Lambda_k$.

Proof. The k-th constraint equation (3.4) can be written as a quadratic equation of Λ_k:

$$a\Lambda_k^2 + b\Lambda_k + c = 0, \tag{3.5}$$

where $a = (1/m_{i_k} + 1/m_{j_k})^2\,\|\mathbf{r}_{(k)}\|^2$, $b = 2(1/m_{i_k} + 1/m_{j_k})\mathbf{r}_{(k)}^T\bar{\mathbf{r}}_{(k)}$, and $c = \|\bar{\mathbf{r}}_{(k)}\|^2 - d_k^2$. For an initial guess $\rho^{(0)}$, the first iterate $\rho^{(1)}$ of the Newton iteration method for solving (3.5) is defined by:

$$\rho^{(1)} = \rho^{(0)} - \frac{a\rho^{(0)\,2} + b\rho^{(0)} + c}{2a\rho^{(0)} + b}. \tag{3.6}$$

If ${\mathbf{r}_{(k)}}^T \bar{\mathbf{r}}_{(k)} \neq 0$ (i.e., $b \neq 0$), we can simply select $\rho^{(0)} = 0$, leading to $\rho^{(1)} = -c/b$, which coincides with Λ_k as defined by (3.1). This completes the proof of Theorem 1.

According to Theorem 3.1, we now can easily overcome the difficulty that exists in SHAKE. That is, if ${\mathbf{r}_{(k)}}^T \bar{\mathbf{r}}_{(k)} = 0$, we can define Λ_k as the first Newton iterate $\rho^{(1)}$ from (3.6) with a small nonzero value of $\rho^{(0)}$ (such as $\rho^{(0)} = 0.001$). Since ${\mathbf{r}_{(k)}}^T \bar{\mathbf{r}}_{(k)} = 0$, $b = 0$, and the formula of Λ_k defined by (3.6) can be simplified as follows:

$$\Lambda_k = \frac{1}{2}\rho^{(0)} + \frac{d_k^2 - \|\bar{\mathbf{r}}_{(k)}\|^2}{2(1/m_{i_k} + 1/m_{j_k})^2 \|\mathbf{r}_{(k)}\|^2 \rho^{(0)}}. \tag{3.7}$$

The complete SHAKE algorithm can now be described in the following algorithm.

ALGORITHM 1 (An improved SHAKE algorithm). *Let the k-th bond constraint connect atoms i_k and j_k for $k = 1, 2, \ldots, l$.*

1. *Set initial guess* $\bar{\mathbf{r}} = 2\mathbf{r}(t) - \mathbf{r}(t - \triangle t) + \triangle t^2 M^{-1}\mathbf{F}(\mathbf{r}(t))$ *and* $\mathbf{r} = \mathbf{r}(t)$
2. *Set* $m = 1$
3. *One SHAKE iteration: For* $k = 1, 2, \ldots, l$, *compute*
   (a) $\bar{\mathbf{r}}_{(k)} = \bar{\mathbf{r}}_{i_k} - \bar{\mathbf{r}}_{j_k}$ *and* $\mathbf{r}_{(k)} = \mathbf{r}_{i_k} - \mathbf{r}_{j_k}$
   (b) *If* $|{\mathbf{r}_{(k)}}^T \bar{\mathbf{r}}_{(k)}| > \epsilon$ *(e.g.,* $\epsilon = 10^{-10}$*), set* Λ_k *by (3.1); else, compute* Λ_k *via (3.7)*
   (c) $\bar{\mathbf{r}}_{i_k} \leftarrow \bar{\mathbf{r}}_{i_k} + \frac{1}{m_{i_k}}\mathbf{r}_{(k)}\Lambda_k$ *and* $\bar{\mathbf{r}}_{j_k} \leftarrow \bar{\mathbf{r}}_{j_k} - \frac{1}{m_{j_k}}\mathbf{r}_{(k)}\Lambda_k$
4. *Convergence test: If the termination rule (3.3) holds, i.e., all updated bond vectors* $\bar{\mathbf{r}}_{(k)}$ *satisfy*

$$|\ \|\bar{\mathbf{r}}_{(k)}\|^2 - d_k^2\ | < \epsilon, \qquad k = 1, 2, \ldots, l,$$

stop and define the m-th SHAKE iterate as: $\mathbf{r}^{(m)} = \bar{\mathbf{r}}$*; else, go to the next SHAKE iteration by setting* $m \leftarrow m + 1$ *and returning to Step 3.*

As mentioned above, we can use one scalar variable to implement the calculation of Λ_k for $k = 1, 2, \ldots, l$ to save memory space for array Λ.

Our modification to Λ_k given by (3.7) ensures convergence; see next section.

4. The mathematical form of SHAKE iteration. The following theorem assigns to SHAKE a common mathematical iterative expression, and shows that such SHAKE iterates are essentially defined by using GSN.

THEOREM 4.1 (SHAKE iterative expression). *The SHAKE iterate sequence* $\{\mathbf{r}^{(m)}\}$ *generated from Algorithm 1 is equivalent to the iterative process*

$$\mathbf{r}^{(m+1)} = \mathbf{r}^{(m)} + \frac{1}{2}M^{-1}J^T\Lambda, \quad m = 0, 1, 2, \ldots, \tag{4.1}$$

where M is the diagonal mass matrix, J is the Jacobian matrix of the constraint vector function g defined in (2.3), and $\Lambda = (\Lambda_1, \Lambda_2, \ldots, \Lambda_l)^T$ *with* Λ_k *being defined by (3.1) or (3.7). Moreover, the vector* Λ *is the first GSN iterate for solving the constraint equation of* Λ

$$g(\mathbf{r}^{(m)} + \frac{1}{2}M^{-1}J^T\Lambda) = 0 \tag{4.2}$$

with the initial guess $\Lambda^0 = (\Lambda_1^0, \Lambda_2^0, \ldots, \Lambda_l^0)^T$ *defined by* $\Lambda_k^0 = 0$ *if* $|{\mathbf{r}_{(k)}}^T \bar{\mathbf{r}}_{(k)}| > \epsilon$, *and* $\Lambda_k^0 = \rho^0$ *(such as* $\rho^0 = 0.001$*) otherwise.*

Proof. For simplicity, we assume that the inner product ${\mathbf{r}_{(k)}}^T\bar{\mathbf{r}}_{(k)} \neq 0$. Hence, from Algorithm 1 it follows that all Λ_k for $k = 1, 2, \ldots, l$ are defined by formula (3.1).

For g_k given in (2.3), the entry $\partial g_k/\partial \mathbf{r}_\mu$ of the Jacobian matrix J is defined as:

$$\frac{\partial g_k}{\partial \mathbf{r}_\mu} = \begin{cases} 2\mathbf{r}_{(k)} & \text{for } \mu = i_k, \\ -2\mathbf{r}_{(k)} & \text{for } \mu = j_k, \\ 0 & \text{otherwise,} \end{cases} \tag{4.3}$$

where $k = 1, 2, \ldots, l$, and $\mu = 1, 2, \ldots, N$ (N = number of atoms). Hence, the SHAKE iterate sequence $\{\mathbf{r}^{(m)}\}$ generated from Algorithm 1 can be expressed as

$$\mathbf{r}_i^{(m+1)} = \mathbf{r}_i^{(m)} + \frac{1}{2m_i}\sum_{k=1}^{l}\frac{\partial g_k}{\partial \mathbf{r}_i}\Lambda_k, \quad i = 1, 2, 3, \ldots, N, \tag{4.4}$$

where $m = 0, 1, 2, \ldots$, and $\mathbf{r}_i^{(0)}$ is an initial guess. Writing the above expressions in a matrix form gives (4.1).

To obtain the expression of Λ in terms of $\mathbf{r}_i^{(m)}$, we introduce vector $\mathbf{b}_\mu^{(k)} = \frac{1}{2}\left(\frac{1}{m_{i_k}}\frac{\partial g_\mu}{\partial \mathbf{r}_{i_k}} - \frac{1}{m_{j_k}}\frac{\partial g_\mu}{\partial \mathbf{r}_{j_k}}\right)$. By using (4.3), the vector $\mathbf{b}_\mu^{(k)}$ can be written as

$$\mathbf{b}_\mu^{(k)} = \begin{cases} \left(\frac{1}{m_{i_k}} + \frac{1}{m_{j_k}}\right)\mathbf{r}_{(k)} & \text{if } \mu = k, \\ -\frac{1}{m_{i_k}}\mathbf{r}_{(\mu)} & \text{if } \mu \in \Theta_{k,i_k}, \\ -\frac{1}{m_{j_k}}\mathbf{r}_{(\mu)} & \text{if } \mu \in \Theta_{k,j_k}, \\ 0 & \text{otherwise,} \end{cases} \tag{4.5}$$

where $\Theta_{k,i}$ denotes the set of bond indices besides k which atom i is involved in; see FIG. 4.1 for an illustration. Hence, the vector Λ_k defined in (3.1) can be expressed by

$$\Lambda_k = \frac{d_k^2 - \|\mathbf{r}_{(k)}^m + \sum_{\mu=1}^{k-1}\mathbf{b}_\mu^{(k)}\Lambda_\mu\|^2}{2(\mathbf{r}_{(k)}^m + \sum_{\mu=1}^{k-1}\mathbf{b}_\mu^{(k)}\Lambda_\mu)^T\mathbf{b}_k^{(k)}}, \quad k = 1, 2, \ldots, l, \tag{4.6}$$

where $\mathbf{r}_{(k)}^m = \mathbf{r}_{i_k}^m - \mathbf{r}_{j_k}^m$.

We next show that the vector Λ defined in (4.6) is the first GSN iterate for solving the constraint equation with an initial guess of zero.

With (4.4) and notation $\mathbf{b}_\mu^{(k)}$, we can write $\mathbf{r}_{(k)}^{m+1}$ as

$$\mathbf{r}_{(k)}^{m+1} = \mathbf{r}_{(k)}^m + \sum_{\mu=1}^{l}\mathbf{b}_\mu^{(k)}\Lambda_\mu,$$

where $k = 1, 2, \ldots, l$, so that the constraint equation (4.2) becomes

$$\|\mathbf{r}_{(k)}^m + \sum_{\mu=1}^{l}\mathbf{b}_\mu^{(k)}\Lambda_\mu\|^2 - d_k^2 = 0, \quad k = 1, 2, \ldots, l. \tag{4.7}$$

Let $f_k(\Lambda)$ denote the left hand side of the above equation. Then $\frac{\partial f_k(\Lambda)}{\partial \Lambda_k} = 2(\mathbf{r}_{(k)}^m + \sum_{\mu=1}^{l}\mathbf{b}_\mu^{(k)}\Lambda_\mu)^T\mathbf{b}_k^{(k)}$. Hence, for an initial guess of zero, the first GSN iterate $\Lambda^{(1)}$

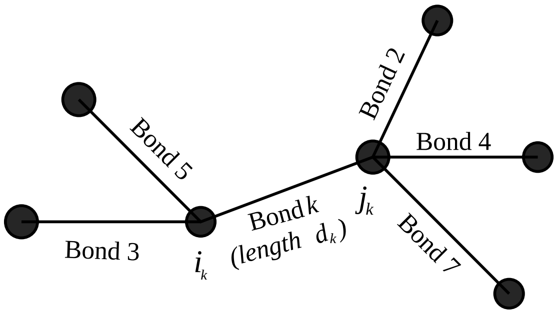

FIG. 4.1. *Illustration of the index set* $\Theta_{k,i}$*:* $\Theta_{k,i_k} = \{3,5\}$ *and* $\Theta_{k,j_k} = \{2,4,7\}$

for solving the nonlinear equations $f_k(\Lambda) = 0$ for $k = 1, 2, \ldots, l$ has the following expression:

$$\Lambda_k^{(1)} = -\frac{\|\mathbf{r}_{(k)}^m + \sum_{\mu=1}^{k-1} \mathbf{b}_\mu^{(k)} \Lambda_\mu^{(1)}\|^2 - d_k^2}{2(\mathbf{r}_{(k)}^m + \sum_{\mu=1}^{k-1} \mathbf{b}_\mu^{(k)} \Lambda_\mu^{(1)})^T \mathbf{b}_k^{(k)}}, \quad k = 1, 2, \ldots, l. \tag{4.8}$$

By comparing (4.6) to (4.8), we see that $\Lambda_k = \Lambda_k^{(1)}$ for $k = 1, 2, \ldots, l$. This completes the proof of Theorem 2.

5. The SHAKE-SOR method. Theorem 2 implies that the vector Λ of the SHAKE iterative expression (4.1) can also be computed by using a more effective nonlinear iterative method. Since SOR-NEWTON is an accelerated variant of GSN, it is natural to define the vector Λ of SHAKE by using SOR-NEWTON. This variant of SHAKE is referred to as the SHAKE-SOR method.

The first iterate of SOR-Newton for solving the nonlinear constraint equations (4.7) is defined by

$$\Lambda_k^{(1)} = \Lambda_k^{(0)} - \omega \frac{\|\mathbf{r}_{(k)}^{(m)} + \sum_{\mu=1}^{k-1} \mathbf{b}_\mu^{(k)} \Lambda_\mu^{(1)} + \sum_{\mu=k}^{l} \mathbf{b}_\mu^{(k)} \Lambda_\mu^{(0)}\|^2 - d_k^2}{2(\mathbf{r}_{(k)}^{(m)} + \sum_{\mu=1}^{k-1} \mathbf{b}_\mu^{(k)} \Lambda_\mu^{(1)} + \sum_{\mu=k}^{l} \mathbf{b}_\mu^{(k)} \Lambda_\mu^{(0)})^T \mathbf{b}_k^{(k)}} \tag{5.1}$$

for $k = 1, 2, \ldots, l$. Hence, for the initial guess $\Lambda^{(0)} = 0$, the value of Λ_k defined by (5.1) has the following formula:

$$\Lambda_k = -\omega \frac{\|\mathbf{r}_{(k)}^m + \sum_{\mu=1}^{k-1} \mathbf{b}_\mu^{(k)} \Lambda_\mu^{(1)}\|^2 - d_k^2}{2(\mathbf{r}_{(k)}^m + \sum_{\mu=1}^{k-1} \mathbf{b}_\mu^{(k)} \Lambda_\mu^{(1)})^T \mathbf{b}_k^{(k)}}. \tag{5.2}$$

Incorporating a relaxation parameter of unity (i.e., $\omega = 1$) into (5.2) yields the formula (4.6) for SHAKE. Hence, the standard SHAKE method can be regarded as a special case of SHAKE-SOR. Its convergence also follows from that of SOR-Newton by the arguments outlined in the next section.

6. The relationship between SHAKE-SOR and SOR-Newton. Another approach to update the position vector $\mathbf{r}(t + \triangle t)$ defined in (2.5) is to directly solve the constraint equation (2.8) by SOR-Newton. The sequence $\{\Lambda^{(m)}\}$ of SOR-Newton iterates for solving (2.8) is defined by

$$\Lambda_k^{(m+1)} = \Lambda_k^{(m)} - \omega \frac{\|\hat{\mathbf{r}}_{(k)} + \sum_{\mu=1}^{k-1} \mathbf{b}_\mu^{(k)} \Lambda_\mu^{(m+1)} + \sum_{\mu=k}^{l} \mathbf{b}_\mu^{(k)} \Lambda_\mu^{(m)}\|^2 - d_k^2}{2(\hat{\mathbf{r}}_{(k)} + \sum_{\mu=1}^{k-1} \mathbf{b}_\mu^{(k)} \Lambda_\mu^{(m+1)} + \sum_{\mu=k}^{l} \mathbf{b}_\mu^{(k)} \Lambda_\mu^{(m)})^T \mathbf{b}_k^{(k)}} \tag{6.1}$$

for $k = 1, 2, \ldots, l$, and $m = 0, 1, 2, \ldots$. Here $\hat{\mathbf{r}}_{(k)} = \hat{\mathbf{r}}_{i_k} - \hat{\mathbf{r}}_{j_k}$, and $\hat{\mathbf{r}}$ is defined in (2.7). See Appendix for details of the classic SOR-Newton iteration. The termination rule of SOR-Newton is that $\|g(\hat{\mathbf{r}} + \frac{1}{2}M^{-1}J^T\Lambda^{(m)})\|_\infty < \epsilon$.

Although SHAKE-SOR is a different algorithm from the SOR-Newton iteration defined in (6.1), we show here that their convergence behavior is equivalent under certain conditions.

THEOREM 6.1 (Relationship of SHAKE-SOR to SOR-Newton). *Let $\{\Lambda^{(m)}\}$ be a sequence of SOR-Newton iterates defined in (6.1), and $\{\mathbf{r}^{(m)}\}$ a sequence of SHAKE-SOR iterates defined by (4.1) and (5.2). If $\Lambda^{(0)} = 0$, then SHAKE-SOR has the following relationship to SOR-Newton:*

$$\mathbf{r}^{(m)} = \mathbf{r}^{(0)} + \frac{1}{2}M^{-1}J^T\Lambda^{(m)}, \quad m = 0, 1, 2, \ldots. \tag{6.2}$$

Proof. Clearly, the vector Λ defined in (5.2) depends on m. For clarity, we denote it as $\rho^{(m)}$. We first show, by induction, that

$$\rho^{(m)} = \Lambda^{(m)} - \Lambda^{(m-1)}, \quad m = 1, 2, \ldots. \tag{6.3}$$

By the definition of $\rho^{(1)}$, it immediately follows that (6.3) holds for $m = 1$ because $\rho^{(1)} = \Lambda^{(1)}$, $\Lambda^{(0)} = 0$, and $\mathbf{r}^0 = \hat{\mathbf{r}}$. Suppose that (6.3) holds for $m = j$. We show that (6.3) holds for $m = j + 1$ by induction. The induction assumption follows that (6.3) holds for $m \leq j$, so that we can write the SHAKE-SOR expression (4.1) as (6.2) with $m = j$. Using this $\rho^{(j)}$ expression, we write the term in the numerator of the SOR-Newton eq. (6.1) inside the norm function as follows:

$$\mathbf{r}^0_{(k)} + \sum_{\mu=1}^{k-1} \mathbf{b}^{(k)}_\mu \Lambda^{(j+1)}_\mu + \sum_{\mu=k}^{l} \mathbf{b}^{(k)}_\mu \Lambda^{(j)}_\mu = \mathbf{r}^j_{(k)} + \sum_{\mu=1}^{k-1} \mathbf{b}^{(k)}_\mu (\Lambda^{(j+1)}_\mu - \Lambda^{(j)}_\mu).$$

Consequently, the SOR-Newton expression (6.1) is written as

$$\Lambda^{(j+1)}_k - \Lambda^{(j)}_k = -\omega \frac{\|\mathbf{r}^j_{(k)} + \sum_{\mu=1}^{k-1} \mathbf{b}^{(k)}_\mu (\Lambda^{(j+1)}_\mu - \Lambda^{(j)}_\mu)\|^2 - d_k^2}{2(\mathbf{r}^j_{(k)} + \sum_{\mu=1}^{k-1} \mathbf{b}^{(k)}_\mu (\Lambda^{(j+1)}_\mu - \Lambda^{(j)}_\mu))^T \mathbf{b}^{(k)}_k}.$$

Substituting $\rho^{(j+1)}$ for the above quantity $\Lambda^{(j+1)}_k - \Lambda^{(j)}_k$, the same expression as (5.2) follows. This proves that $\rho^{(j+1)} = \Lambda^{(j+1)} - \Lambda^{(j)}$. Therefore, (6.3) is true for all integers m.

We can now prove the relationship (6.2) by induction. According to the definition of SHAKE-SOR, (6.2) holds for $m = 1$. Suppose that (6.2) holds for $m - 1$. Then, with (6.3),

$$\begin{aligned}
\mathbf{r}^{(m)} &= \mathbf{r}^{(m-1)} + \frac{1}{2}M^{-1}J^T\rho^{(m)} \\
&= \mathbf{r}^{(m-1)} + \frac{1}{2}M^{-1}J^T(\Lambda^{(m)} - \Lambda^{(m-1)}) \\
&= \mathbf{r}^{(0)} + \frac{1}{2}M^{-1}J^T\Lambda^{(m-1)} + \frac{1}{2}M^{-1}J^T(\Lambda^{(m)} - \Lambda^{(m-1)}) \\
&= \mathbf{r}^{(0)} + \frac{1}{2}M^{-1}J^T\Lambda^{(m)}.
\end{aligned}$$

This completes the proof of (6.2) for all $m = 1, 2, \ldots$.

From relation (6.2), the convergence of SHAKE-SOR follows from that of SOR-Newton. In fact, if $\lim_{m\to\infty} \Lambda^{(m)} = \Lambda^{(*)}$, and $\Lambda^{(*)}$ satisfies

$$g(\mathbf{r}^{(0)} + \frac{1}{2}M^{-1}J^T\Lambda^{(*)}) = 0,$$

then, with (6.2),

$$\lim_{m\to\infty} \mathbf{r}^{(m)} = \mathbf{r}^{(0)} + \frac{1}{2}M^{-1}J^T \lim_{m\to\infty} \Lambda^{(m)} = \mathbf{r}^{(0)} + \frac{1}{2}M^{-1}J^T\Lambda^{(*)},$$

and

$$\lim_{m\to\infty} g(\mathbf{r}^{(m)}) = g(\mathbf{r}^{(0)} + \frac{1}{2}M^{-1}J^T\Lambda^{(*)}) = 0.$$

Conversely, the convergence of SOR-Newton does not follow that of SHAKE-SOR unless the linear system $J^T\Lambda = 0$ has a *unique zero solution*. This means that if the rank of Jacobian matrix J^T is larger than or equal to l, and the initial guess $\Lambda^{(0)} = 0$, the convergence is equivalent for SHAKE-SOR and SOR-Newton. In general, however, the convergence of the two iterative sequences $\{\Lambda^{(m)}\}$ and $\{\mathbf{r}^{(m)}\}$ is not equivalent.

7. Numerical experiments. To show that SHAKE-SOR can improve the performance of SHAKE [13], we experimented in CHARMM [4, 9] with two proteins, BPTI and lysozyme, and a DNA dodecamer. The DNA system is the Protein Data Bank structure 1D98 with sequence d(CGCAAAAAAGCG)·d(GCGTTTTTTCGC). These three systems use 582, 2050, and 11510 bond constraints, respectively. The proteins BPTI (568 atoms) and lysozyme (2030 atoms) are simulated in vacuum; hydrogens have been added to the structures using the HBUILD algorithm of CHARMM [4, 9]. The DNA model, which has 760 atoms (including hydrogens), is placed in a hexagonal prism with 27 crystallographic waters, 22 sodium ions, and 3537 additional water molecules [20]. Each water molecule in the DNA system is modeled with three constraints in CHARMM.

We set the timestep to $\triangle t = 2$ fs (femtosecond) and performed one step of dynamics for all numerical experiments. Default parameters in CHARMM were used for computing the force field $\mathbf{F}(\mathbf{r})$. The numerical experiments were performed in double precision on a single R10000 processor (195 MHZ) of an SGI Power Challenge L computer at New York University. CPU times in FIG. 7.2 in seconds were derived by using the SGI system routine *etime*().

FIG. 7.1 plots the total number of SHAKE-SOR iterations, determined by the convergence rule (3.3) with tolerance $\epsilon = 10^{-12}$, as a function of relaxation parameter ω. It compares the convergence performance of SHAKE-SOR for simulating the three molecular systems with different numbers of bond constraints. The total number of iterations of SHAKE [13] (which can be thought of as SHAKE-SOR using $\omega = 1$) for BPTI, lysozyme, and DNA are 37, 37, and 44, respectively; SHAKE-SOR with $\omega = 1.2$ reduces these numbers to 24, 22, and 25. These values are very similar to the averages obtained over a long trajectiory (see Table 1). FIG. 7.1 also suggests that $\omega = 1.2$ is the *optimal* relaxation parameter for all three molecular systems.

FIG. 7.2 plots the total CPU time of SHAKE-SOR as a function of ω for simulating DNA with 11510 bond constraints. SHAKE-SOR with $\omega = 1.2$ took 0.17 seconds, less than half the CPU time for standard SHAKE (0.36 seconds). This implies about a factor of two saving in CPU time over a dynamics trajectory.

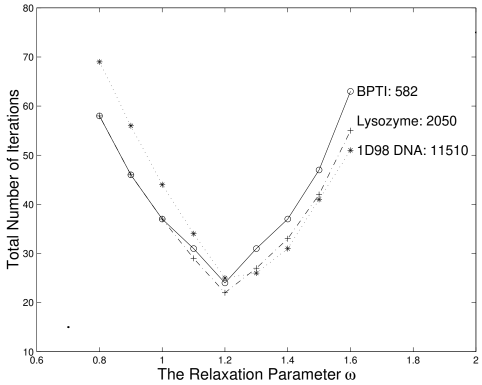

FIG. 7.1. *The total number of SHAKE-SOR iterations as a fuction of relaxation parameter ω. This figure shows that SHAKE-SOR with $\omega = 1.2$ converges much faster than SHAKE [13] (i.e., SHAKE-SOR with $\omega = 1$). Here the timestep $\triangle t = 2$ fs.*

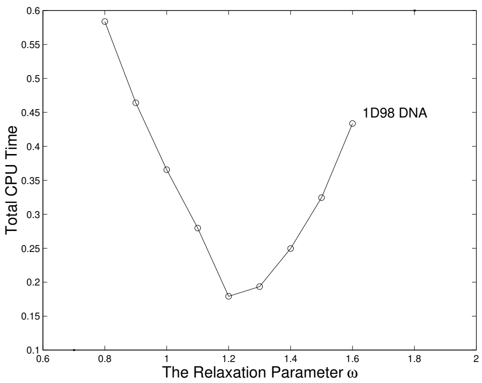

FIG. 7.2. *The total CPU time of SHAKE-SOR as a fuction of relaxation parameter ω for simulating 1D98 DNA with 11508 bond constraints. SHAKE-SOR with $\omega = 1.2$ took less than half the CPU time of standard SHAKE [13].*

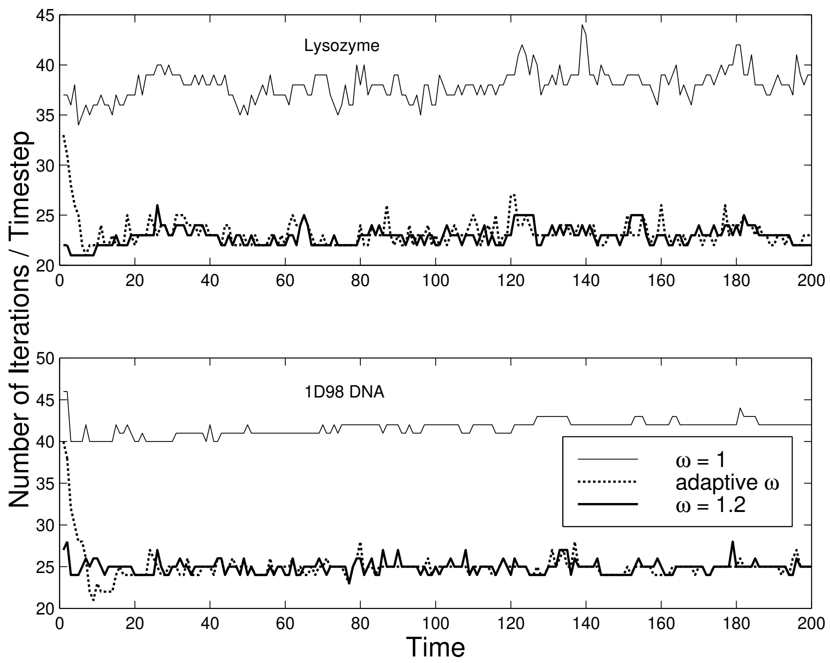

FIG. 7.3. *The total number of SHAKE-SOR iterations per timestep as a fuction of time*

FIG. 7.3 plots the total number of SHAKE-SOR iterations as a function of time for lysozyme and DNA as obtained over 400 fs. Corresponding CPU times and average iteration counts per timestep are listed in Table 7.1. From this figure and table we see that the SHAKE-SOR using the optimal relaxation parameter $\omega = 1.2$ similarly requires about half the number of iterations and also half the CPU time as standard

TABLE 7.1
Performance of SHAKE-SOR for 200 timestep simulations. The number in parentheses is the average value per timestep. CPU time is measured in seconds.

SHAKE-SOR	Total CPU Time		Total Iterations	
	Lysozyme	1D98 DNA	Lysozyme	1D98 DNA
$\omega = 1$	9.59 (0.05)	67.42 (0.34)	7603 (38)	8326 (42)
$\omega = 1.2$	5.98 (0.03)	33.46 (0.17)	4579 (23)	4983 (25)
adaptive ω	6.02 (0.03)	33.53 (0.17)	4651 (23)	4998 (25)

SHAKE (i.e., $\omega = 1$).

To determine the optimal relaxation parameter of SHAKE-SOR, a simple adaptive algorithm has been proposed in [3]. Using that formula, we performed expriments for the lysozyme and DNA systems. Very similar performance was obtained as for the SHAKE-SOR using $\omega = 1.2$ (see FIG. 7.3 and Table 7.1). This shows that the adaptive algorithm in [3] also works well for choosing the optimal relaxation parameter.

Finally, we show by a constructed example that SHAKE-SOR with our modification of Λ given in (3.7) works well in the case of $\mathbf{r}_{(k)}{}^T \bar{\mathbf{r}}_{(k)} = 0$. The numerical example was constructed for lysozyme by modifying the current position vector $\bar{\mathbf{r}}_{j_k}$ in such a way that the current bond vector $\bar{\mathbf{r}}_{(k)}$ satisfies $\mathbf{r}_{(k)}{}^T \bar{\mathbf{r}}_{(k)} = 0$. Specifically, for $k = 10$ and $j_k = 23$, we set $\bar{\mathbf{r}}_{j_k} = (1/(4x_1), 1/(4y_1), -1/(2z_1))^T + \bar{\mathbf{r}}_{i_k}$, where (x_1, y_1, z_1) denotes the reference vector $\mathbf{r}_{(k)}$. Thus,

$$\bar{\mathbf{r}}_{(k)} = \bar{\mathbf{r}}_{i_k} - \bar{\mathbf{r}}_{j_k} = (-1/(4x_1), -1/(4y_1), 1/(2z_1))^T,$$

leading to $\mathbf{r}_{(k)}{}^T \bar{\mathbf{r}}_{(k)} = -1/4 - 1/4 + 1/2 = 0$. With $\triangle t = 2$ fs and $\omega = 1.2$, we tested SHAKE-SOR for four different numerical values with our modification of eq. (3.7). With the convergence criterion (3.3) with $\epsilon = 10^{-12}$, 18, 21, 21 and 29 iterations were required for the convergence of SHAKE. In contrast, the standard SHAKE implementation does not work in this case.

8. Conclusions. The SHAKE scheme proposed in [13] is a widely-used numerical iterative scheme for constrained molecular dynamics simulations. For the purpose of analyzing SHAKE, we have presented SHAKE in a well known iterative framework and shown that the SHAKE iterates can be defined by using the Gauss-Seidel-Newton method or other more efficient nonlinear iterative solvers. By relying on the SOR-Newton method, we have derived the same accelerated variant of SHAKE proposed in [3, 22], and called it SHAKE-SOR. The basic relationship between SHAKE and SOR-Newton was then proven. Convergence of SHAKE-SOR follows that of the nonlinear SOR method [12] only when certain conditions hold.

We have also proposed a simple modification to the standard SHAKE process (eq. (3.7)) that ensures convergence even in the case of zero inner product between a reference bond vector and a corresponding updated bond vector.

Our SHAKE description provides many possibilities for defining other efficient variants of SHAKE. SHAKE is a sequential scheme since it is defined by the sequential GSN (or SOR-Newton) method. To efficiently implement SHAKE on parallel computers, efficient parallel versions of SHAKE can be used, mirroring parallel versions of SOR-Newton, such as the SOR-Newton using the Red-Black ordering [8] or the PSOR ordering [23].

Acknowledgments. T. Schlick is indebted to Cathleen Morawetz for her guidance and spirit. We thank Dr. Dan Strahs for providing the 1D98 DNA system for our numerical experiments. This work is supported by the National Science Foundation (ASC-9157582 and BIR 94-23827EQ) [to T.S] and ASC-9217374 [to L.R. Scott], the National Institutes of Health (R01 GM55164) [to T.S.], and the University of Southern Mississippi (#2221709006) [to D.X.]. T. Schlick is an investigator of the Howard Hughes Medical Institute.

Appendix A. The SOR-Newton method for solving nonlinear constrained equations. A direct way to approximate $\mathbf{r}(t + \triangle t)$ defined in (2.5) is to solve the constraint equation (2.8) approximately for Λ by an efficient nonlinear iterative method. Here we consider the SOR-Newton method [12] for solving (2.8) because it is closely related to SHAKE.

We start by defining the nonlinear Gauss-Seidel iterates $\{\Lambda^{(m)}\}$ for solving a system of nonlinear equations:

$$f_k(\Lambda_1, \Lambda_2, \ldots, \Lambda_l) = 0, \quad k = 1, 2, \ldots, l.$$

Suppose that iterate $\Lambda^{(m)}$ is available. For $k = 1, 2, \ldots, l$, the nonlinear Gauss-Seidel method defines the update $\Lambda_k^{(m+1)}$ as a solution of the following nonlinear equation of Λ_k

$$f_k(\Lambda_1^{(m+1)}, \Lambda_2^{(m+1)}, \ldots, \Lambda_{k-1}^{(m+1)}, \Lambda_k, \Lambda_{k+1}^{(m)}, \ldots, \Lambda_l^{(m)}) = 0. \tag{A.1}$$

Based on the nonlinear Gauss-Seidel method, the nonlinear successive over-relaxation (SOR) iterates $\{\Lambda^{(m)}\}$ are then defined by

$$\Lambda^{(m+1)} = (1 - \omega)\Lambda^{(m)} + \omega\tilde{\Lambda}, \quad m = 0, 1, 2, \ldots, \tag{A.2}$$

where $\tilde{\Lambda}$ is the nonlinear Gauss-Seidel iterate, $\Lambda^{(0)}$ is an initial guess, and ω is the relaxation parameter, which is often chosen from the interval $(0, 2)$. Clearly, with $\omega = 1$, the nonlinear SOR method is reduced to the nonlinear Gauss-Seidel method.

Eq. (A.1) can be solved for Λ_k approximately by n steps of Newton iteration:

$$\Lambda_k^{\mu+1} = \Lambda_k^{\mu} - \omega \frac{f_k(\Lambda_1^{(m+1)}, \Lambda_2^{(m+1)}, \ldots, \Lambda_{k-1}^{(m+1)}, \Lambda_k^{\mu}, \Lambda_{k+1}^{(m)}, \ldots, \Lambda_l^{(m)})}{\frac{\partial f_k}{\partial \Lambda_k}(\Lambda_1^{(m+1)}, \Lambda_2^{(m+1)}, \ldots, \Lambda_{k-1}^{(m+1)}, \Lambda_k^{\mu}, \Lambda_{k+1}^{(m)}, \ldots, \Lambda_l^{(m)})}$$

for $\mu = 0, 1, 2, \ldots, n$. Here the initial guess $\Lambda_k^0 = \Lambda_k^{(m)}$. Substituting

$$\Lambda^n = (\Lambda_1^n, \Lambda_2^n, \ldots, \Lambda_l^n)^T$$

to the term $\tilde{\Lambda}$ in (A.2) leads to the n-step SOR-Newton method. Since the asymptotic convergence rate of the n-step SOR-Newton method is independent of n [12], a one-step SOR-Newton method is often used, and is called the SOR-Newton method. That is, the SOR-Newton iterates $\{\Lambda^{(m)}\}$ are defined by

$$\Lambda_k^{(m+1)} = \Lambda_k^{(m)} - \omega \frac{f_k(\Lambda_1^{(m+1)}, \Lambda_2^{(m+1)}, \ldots, \Lambda_{k-1}^{(m+1)}, \Lambda_k^{(m)}, \ldots, \Lambda_l^{(m)})}{\frac{\partial f_k}{\partial \Lambda_k}(\Lambda_1^{(m+1)}, \Lambda_2^{(m+1)}, \ldots, \Lambda_{k-1}^{(m+1)}, \Lambda_k^{(m)}, \ldots, \Lambda_l^{(m)})}$$

for $k = 1, 2, \ldots, l$ and $m = 0, 1, 2, \ldots$. A special case of SOR-Newton with $\omega = 1$ is called the Gauss-Seidel-Newton (GSN) method.

In particular, for $f_k(\Lambda) = g_k(\bar{\mathbf{r}}(t) + B\Lambda)$, where g_k is defined in (2.3), we obtain the iterative expression of SOR-Newton as below:

$$\Lambda_k^{(m+1)} = \Lambda_k^{(m)} - \omega \frac{\|\bar{\mathbf{r}}_{(k)} + \sum_{\mu=1}^{k-1} \mathbf{b}_\mu^{(k)} \Lambda_\mu^{(m+1)} + \sum_{\mu=k}^{l} \mathbf{b}_\mu^{(k)} \Lambda_\mu^{(m)}\|^2 - d_k^2}{2(\bar{\mathbf{r}}_{(k)} + \sum_{\mu=1}^{k-1} \mathbf{b}_\mu^{(k)} \Lambda_\mu^{(m+1)} + \sum_{\mu=k}^{l} \mathbf{b}_\mu^{(k)} \Lambda_\mu^{(m)})^T \mathbf{b}_k^{(k)}}$$

for $k = 1, 2, \ldots, l$, and $m = 0, 1, 2, \ldots$, where $\bar{\mathbf{r}}_{(k)}$ denotes the bond vector $\bar{\mathbf{r}}_{i_k}(t) - \bar{\mathbf{r}}_{j_k}(t)$ of the k-th constraint, and the vector $\mathbf{b}_\mu^{(k)}$ is defined by (4.5).

When an SOR-Newton iterate $\Lambda^{(m)}$ is sufficiently close to an exact solution of eq. (2.8), it can be used to substitute the vector Λ in (2.5) to obtain a satisfactory approximation of $\mathbf{r}(t + \triangle t)$.

REFERENCES

[1] M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids,* Oxford University Press, Oxford, 1987.

[2] H. C. Andersen, *Rattle: a 'velocity' version of the SHAKE algorithm for molecular dynamics calculations,* J. Comp. Phys., 52 (1983), pp. 24–34.

[3] E. Barth, K. Kuczera, B. Leimkuhler, and R. D. Skeel, *Algorithms for constrained molecular dynamics,* J. Comp. Chem., 16 (1995), pp. 1192–1209.

[4] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus, *CHARMM: A program for macromolecular energy, minimization, and dynamics calculations,* J. Comp. Chem., 4 (1983), pp. 187–217.

[5] W. F. van Gunsteren and H. J. C. Berendsen, *Algorithms for macromolecular dynamics and constraint dynamics*, Mol. Phys., 34 (1977), pp. 1311–1327.

[6] W. F. van Gunsteren and H. J. C. Berendsen, *GROMOS: GROningen molecular simulation software,* Technical report, Laboratory of Physical Chemistry, University of Groningen, Nijenborgh, The Netherlands, 1988.

[7] W. F. van Gunsteren and M. Karplus, *Effect of constraints on the dynamics of macromolecules,* Macromolecules, 15 (1982), pp. 1528–1543.

[8] L. A. Hageman and D. M. Young, *Applied Iterative Methods*, Academic press, New York, 1981.

[9] A. D. Jr. MacKerell, etal, *An all-atom empirical potential for molecular modeling and dynamics of proteins,* J. Phys. Chem. B, 102 (1998), pp. 3586–3616.

[10] J. A. McCammon and S. C. Harvey, *Dynamics of Proteins and Nucleic Acids,* Cambridge University Press, Cambridge, 1987.

[11] J. A. McCammon, B. M. Pettitt, and L. R. Scott, *Ordinary differential equations of molecular dynamics,* Comp. Math. Applic., 28 (1994), pp. 319–326.

[12] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic press, New York, 1970.

[13] J. P. Ryckaert, G. Ciccoti, and H. J. C. Berendsen, *Numerical integration of the Cartesian equations of motion of a system with constraints: molecular dynamics of n-Alkanes,* J. Comput. Phys., 23 (1977), pp. 327–341.

[14] J. P. Ryckaert, *Special geometrical constraints in the molecular dynamics of chain molecules*, Mol. Phys., 55 (1985), pp. 549–556.

[15] J. P. Ryckaert, *On the convergence of the SHAKE algorithm,* Comp. Phys. Comm., 62 (1991), p. 336.

[16] T. Schlick, E. Barth, and M. Mandziuk, *Biomolecular dynamics at long timesteps: bridging the timescale gap between simulation and experimentation,* Ann. Rev. Biophys. Biomol. Struc., 26 (1997), pp. 179–220.

[17] T. Schlick, *Some failures and successes of long-timestep approaches for biomolecular simulations,* in Computational Molecular Dynamics: Challenges, Methods, Ideas – Proceedings of the 2nd International Symposium on Algorithms for Macromolecular Modeling, P. Deuflhard, J. Hermans, B. Leimkuhler, A. E. Mark, S. Reich, and. R. D. Skeel, eds., Lecture Notes in Computational Science and Engineering 4, Springer-Verlag, Berlin and New York, 1998, pp. 227–262.

[18] T. Schlick, R. D. Skeel, A. T. Brünger, L. V. Kalé, Jr. J. A. Board, J. Hermans, and K. Schulten, *Algorithmic challenges in computational molecular biophysics,* J. Comp. Phys., 151 (1999), pp. 1–8.

[19] J. Shen and J. A. McCammon, *Molecular dynamics simulation of superoxide interacting with superoxide dismutase,* Chem. Phys., 158 (1991), pp. 191–198.

[20] D. Strahs and T. Schlick, *A-Tract bending: Insights into experimental structures by computational models*, J. Mol. Biol., 301 (2000), pp. 643–666.

[21] L. Verlet, *Computer 'experiments' on classical fluids. I. Thermodynamical Properties of Lennard-Jones Molecules,* Phys. Rev., 159 (1967), pp. 98–103.

[22] D. Xie, *New Parallel Iteration Methods, New Nonlinear Multigrid Analysis, and Application in Computational Chemistry*, Research Report UH/MD - 208, Ph.D. thesis, University of Houston, 1995.

[23] D. Xie and L. Adams, *New parallel SOR method by domain partitioning,* SIAM J. Sci. Comput., 20 (1999), pp. 2261–2281.

[24] Note: The following two technical inaccuracies arise from the analysis in [3]:

(1) The notation used in eq. (23) of [3] (p. 1198) is not appropriate for theoretical analysis of SHAKE because it is a formula for practical implementation in which the position vector is overwritten by the updated vector. See eq. (3.2) in this paper. Thus, the notation in [3] cannot be used to prove the identity between SHAKE and GSN.

(2) In the three equations above (26) of [3] (p. 1198), the scalar Λ_i^{k-1} used in the argument of the function g_i should be replaced by the vector

$$\Lambda^{k,k-1} = (\Lambda_1^k, \Lambda_2^k, \ldots, \Lambda_{i-1}^k, \Lambda_i^{k-1}, \ldots, \Lambda_m^{k-1})^T.$$

Still, even if this simple substitution is made, the equation above (26) does not follow from the preceding one because

$$Q_{i-1}^k \neq \bar{Q} - M^{-1} G^T \Lambda^{k,k-1}.$$

Here m is the number of bond constraints, and G^T is the Jacobian matrix. Therefore, the statement "which is precisely SHAKE iteration of eq. (25)" does not follow the line of argument presented. Recall that here we show the convergence equivalence between SHAKE and GSN only when the initial guess for GSN is zero and the rank of Jacobian matrix is larger than or equal to the number of bond constraints.