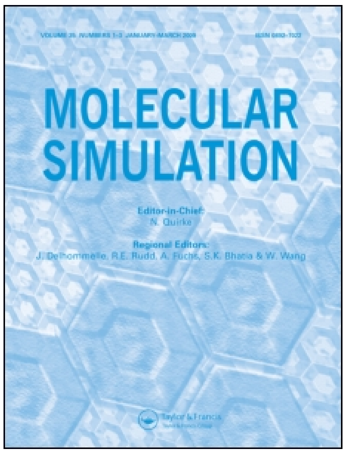

A Non-Iterative Matrix Method for Constraint Molecular Dynamics Simulations

Makoto Yoneya[a]; H. J. C. Berendsen[b]; Kootaro Hirasawa[ac]
[a] Hitachi Research Laboratory, Hitachi, Ltd., Ibaraki, Japan [b] Laboratory of Biophysical Chemistry, Universtiy of Groningen, Groningen, The Netherlands [c] Department of Electrical Engineering, Kyusyu University, Fukuoka, Japan

To cite this Article Yoneya, Makoto , Berendsen, H. J. C. and Hirasawa, Kootaro(1994) 'A Non-Iterative Matrix Method for Constraint Molecular Dynamics Simulations', Molecular Simulation, 13: 6, 395 — 405
To link to this Article: DOI: 10.1080/08927029408022001
URL: http://dx.doi.org/10.1080/08927029408022001

Molecular Simulation, 1994, Vol. 13, pp. 395-405
Reprints available directly from the publisher
Photocopying permitted by license only

A NON-ITERATIVE MATRIX METHOD FOR CONSTRAINT MOLECULAR DYNAMICS SIMULATIONS

MAKOTO YONEYA†, H. J. C. BERENDSEN‡ and KOOTARO HIRASAWA†*

† Hitachi Research Laboratory, Hitachi, Ltd., 7-1-1 Omika, Hitachi, Ibaraki 319-12, Japan
‡Laboratory of Biophysical Chemistry, Universtiy of Groningen, Nijenborgh 4, 9747 AG Groningen, The Netherlands

(Received February 1994, accepted May 1994)

A non-iterative version of the matrix method which can be easily vectorised and parallelized is presented. The original matrix method, which is conceptually identical to the familiar SHAKE algorithm, is shown to be non-iterative when incorporated with the Verlet and leap-frog integration schemes with the same constraint error order as the (local) error order of the accompanying integration schemes. The method is checked by test simulations with n-butane and the liquid crystal moleculae 5CB (4-n-pentyl-4'-cyanobiphenyl) in which its effectiveness is demonstrated.

KEY WORDS: Molecular dynamics simulation, constraint dynamics, SHAKE

1. INTRODUCTION

Constraint dynamics is a way to allow larger integration time steps than those possible in ordinary dynamics by placing constraints on some intramolecular degrees of freedom. Since the introduction of the algorithm SHAKE [1] in the late '70s, most of practical dynamics calculations have been done by applying constraints. In the original SHAKE paper [1], two methods were presented; the first considers all constraints at the same time in a matrix equation and the second considers them individually in succession. The latter approach was designated as "SHAKE" in the paper and the former was referred to as the "matrix method". These two methods are conceptually identical and SHAKE can be said to be a practical variant on the matrix method which allows it to be applied to macromolecules with large numbers of constraints.

Efforts have been made to improve the constraint algorithm [2–6], among which the algorithm proposed by Edberg, Evans and Morris [4] (EEM) is quite attractive. This is due to its non-iterative nature which is very desirable if a vector and /or parallel computer is used. In contrast, the iterative loop of SHAKE has recurrences and these inhibit vectorisation and parallelisation of the algorithm [6]. The most basic difference

* Present address: Department of Electrical Engineering, Kyusyu University, 6-10-1 Hakozaki, Higashi-ku, Fukuoka 812, Japan.

between SHAKE (also the matrix method) and the EEM algorithm is that the former defines constraint forces to fulfill the constraint condition at the next time step, but the latter defines them to fulfill it at the present time. This difference requires that the EEM algorithm has an error accumulation handler which makes programmes quite complex and also demands the use of a self-starting time integration schemes [5]. Although both the EEM algorithm and matrix method are based on similar matrix calculations, no investigations have examined the relationship between them.

In this study, we examine their relationship and show that the matrix method can be non-iterative when incorporated with the Verlet and leap-frog integration schemes. Furthermore, we show that the non-iterative version of the matrix method (the matrix method without iteration) can be a way to overcome the inconvenience of the EEM algorithm described above. First, we discuss the difference between the matrix method and the EEM algorithm, and then we describe the non-iterative matrix method (NIMM). After that, test simulation results with *n*-butane and a liquid crystal molecule 5CB (4-*n*-pentyl-4'-cyanobiphenyl) are described.

2. MATRIX METHOD AND EEM ALGORITHM

First, formulation of the problem must be clarified. Usually bond lengths and sometimes bond angles are frozen in constraint dynamics. In this article, we discussed only bond length constraints because the bond angle constraint is formally the same as that of bond length. The discussion here could also easily be generalized to constraints of higher order in the coordinates.

Let us assume there are N bonds in a molecule. $\mathbf{x}_i$, $\mathbf{x}_j$ are the coordinates of two atoms which form the n-th bond of this molecule and $\mathbf{d}_n$ is its fixed bond length. The total constraint condition of a molecule can be expressed as follows:

$$\chi_n(t) \equiv \mathbf{r}_n^2(t) - \mathbf{d}_n^2 = 0 \quad (n = 1 - N) \tag{1}$$

where

$$\mathbf{r}_n(t) \equiv \mathbf{x}_i(t) - \mathbf{x}_j(t).$$

We can define the first and second time derivatives of this constraint condition as follows.

$$\dot{\chi}_n(t) \equiv 2\mathbf{r}_n(t)\cdot\dot{\mathbf{r}}_n(t) = 0 \quad (n = 1 - N) \tag{2}$$

$$\ddot{\chi}_n(t) \equiv 2(\mathbf{r}_n(t)\cdot\ddot{\mathbf{r}}_n^2(t) + \dot{\mathbf{r}}_n^2(t)) = 0 \quad (n = 1 - N) \tag{3}$$

Constraint dynamics algorithms define constraint forces to fulfill these conditions.

A constraint algorithm must have two functions: a function to calculate constraint forces and a function to handle numerical error accumulation. The first function is trivial. The second function is necessary because numerical time integration schemes have only limited accuracy. Pioneer efforts in constraint dynamics encountered a severe error problem by ignoring correction of the error accumulation [7]. So, this second function is very important. In the following, we discuss the difference between the matrix method and the EEM algorithm from the viewpoint of these two functions. The most of the discussion below can also be applied to SHAKE and the EEM algorithm because SHAKE and the matrix method are equivalent regarding these two functions.

2.1. *Constraint force calculation*

As we stated in the first section, the most remarkable difference between the matrix method (also SHAKE) and the EEM algorithm is that the former defines constraint forces to fulfill equation (1) at the next time step $t + \Delta t$, but the latter defines them to fulfill equation (3) at the present time. In both the matrix method and the EEM algorithm, constraint forces are actually given in terms of Lagrange multipliers, and equation (1) is second order (nonlinear) whereas equation (3) is first order (linear) regarding these multipliers. This means that the matrix method (also SHAKE) needs an iterative procedure to get constraint forces, but the EEM algorithm can get them without iterations.

In SHAKE, the major loop is this iterative loop and its vectorisation is possible by splitting the constraint lists into mutually independent subsets [6]. But this splitting also shorten the list (vector loop) length and that sometimes makes it difficult to achieve the highest vector efficiency. On the other hand, the EEM algorithm can be easily vectorised (e.g. simply apply system supplied vectorised matrix solvers) and for this reason, it is better suited to a vector machine than SHAKE and the matrix method. If we calculate constraint forces in parallel, a difference in iteration number between molecules in the matrix method (and SHAKE) would cause load imbalance and parallel performance reduction [8]. On parallel computer architectures with distributed memory, the iterative procedure of SHAKE causes severe communication overhead which can in practice be prohibitive [9] unless the similar splitting technique above is utilized [10]. So, the EEM algorithm would also be preferable in a parallel machine.

These iterative and non-iterative features characterize the main difference between the matrix method (also SHAKE) and the EEM algorithm in the calculation of constraint forces.

2.2. *Error handling*

In the matrix method, error accumulation handling is built into the calculation procedure of constraint forces. By fulfilling constraint equation (1) not at the present time, but at the next time step, the matrix method effectively suppresses error accumulation. In contrast, the corresponding procedure in the EEM algorithm has no such error handling function itself. So, it needs an independent error handler. The original EEM algorithm used an adaptive correction for penalty functions of constraint errors. This error handler invokes a nonlinear minimizer whenever the penalty functions exceed some tolerance and minimizes constraint errors. But it has some drawbacks. The nonlinear minimizer complicates the program structure. Moreover, sometimes the minimizer needs a self-start scheme (e.g. Runge-Kutta) in time integrations [5].

To overcome this problem, Baranyai and Evans [5] proposed another error handler which uses proportional feedback of constraint error. In this case, the error handler is simpler than the original one and a wider choice is possible in the time integration scheme. But, this error handler needs feedback gain values as user input parameters and sometimes it can be difficult to find good parameters to achieve both accuracy and stability.

3. NON-ITERATIVE MATRIX METHOD

In this section, we describe the non-iterative matrix methods (NIMM) which solves the above mentioned problems by combining concepts from the matrix method and the EEM algorithm.

3.1. *Basic concept*

Let us start from the following Taylor expansion of $\chi_n(t+\Delta t)$ about the time step Δt.

$$\chi_n(t+\Delta t) = \chi_n(t) + \dot{\chi}_n(t)\Delta t + (1/2)\ddot{\chi}_n(t)\Delta t^2 + O(\Delta t^3) \tag{4}$$

As we stated in the previous section, the matrix method (also SHAKE) actually solves the equation in the following way:

$$\chi_n(t+\Delta t) + O(\varepsilon) = 0 \tag{5}$$

where ε is the tolerance of the constraint error in the iterative procedure of the matrix method. The EEM algorithm solves the next equation.

$$\ddot{\chi}_n(t) = 0 \tag{6}$$

Here, equation (5) is second order and equation (6) is first order regarding the Lagrange multipliers and the later one is represented by a system of linear equations involving Lagrange multipliers [4] (see Appendix 1).

Then we consider the next equation.

$$\chi_n(t+\Delta t) + O(\Delta t^3) = \chi_n(t) + \dot{\chi}_n(t)\Delta t + (1/2)\ddot{\chi}_n(t)\Delta t^2 = 0 \tag{7}$$

In this equation, the L.H.S. is similar to the equation solved in the matrix method (equation (5)). On the other hand, the R.H.S. of equation (7) is obtained by adding the constraint error and its first time derivative with a weight to the equation solved by the EEM algorithm. This R.H.S. of equation (7) differs only in the R.H.S. vector from the EEM matrix equation and it is also linear regarding the Lagrange multiplier. By considering these extra terms, (7) is equivalent to the matrix method with regard to satisfying constraints in the next step (with extrapolation by Taylor expansion) and the same error suppression function can be expected. It is from this point that we develop the NIMM.

3.2. *Implementations*

There are some practical problems in the basic concept in the former section. Equation (7) can be written in the following form by substituting equations (1)–(3):

$$\mathbf{r}_n^2(t) - \mathbf{d}_n^2 + 2\mathbf{r}_n(t)\cdot\dot{\mathbf{r}}_n(t)\Delta t + (\mathbf{r}_n(t)\cdot\ddot{\mathbf{r}}_n(t) + \dot{\mathbf{r}}_n^2(t))\Delta t^2 = 0 \tag{8}$$

It is clear from this equation that we not only need information on positions but also on velocities. For this reason, if we use schemes like Verlet (which does not use velocity) or leap-frog (which evaluates position and velocity at different times) we can not use our concept directly. The method can be naturally introduced into the velocity-Verlet algorithm which is similar to the constraint algorithm RATTLE [3]. This however, requires the constraint procedure be applied twice per step. Moreover, the constraint error order must be the same as the error of the accompanying integration scheme (see Appendix 1). So, if we use the Verlet scheme we must use one more higher order equation than (7) because the error order of the Verlet scheme is $O(\Delta t^4)$.

We can solve these problems by using not the Taylor expansion above, but the integration scheme itself in the evaluation of $\chi_n(t+\Delta t)$ as in the matrix method.

First, the Verlet scheme can be expressed as follows.

$$\mathbf{x}(t+\Delta t) = 2\mathbf{x}(t) - \mathbf{x}(t-\Delta t) + \ddot{\mathbf{x}}(t)\Delta t^2\ (+O(\Delta t^4)) \tag{9}$$

Here, the last term of the R.H.S. represents an error term. The equation for $\mathbf{r}_n(t+\Delta t)$ takes the same form as above, and we can get the following equation for $\chi_n\ (t+\Delta t)$.

$$\begin{aligned}\chi_n(t+\Delta t) &= (2\mathbf{r}_n(t) - \mathbf{r}_n(t-\Delta t))^2 - \mathbf{d}_n^2 \\ &\quad + 2(2\mathbf{r}_n(t) - \mathbf{r}_n(t-\Delta t))\cdot\ddot{\mathbf{r}}_n(t)\Delta t^2(+O(\Delta t^4)) = 0\end{aligned} \tag{10}$$

The constraint error order of this equation is the same as the error order of the Verlet scheme (i.e. $O(\Delta t^4)$) and the equation does not include velocity terms. This means we can solve the former problems by using this equation instead of (8).

At this point, we should compare equation (10) and the equation which is solved in the matrix method. The matrix method starts from a linearized (in the Lagrange multiplier) equation in the following and then makes an iteration with non-linear terms.

$$\begin{aligned}\chi_n(t+\Delta t) &= \mathbf{q}_n^2(t+\Delta t) - \mathbf{d}_n^2 + 2\mathbf{q}_n(t+\Delta t)\cdot\ddot{\mathbf{r}}_n^c(t)\Delta t^2(+O(\Delta t^4)) \\ &= eq.(10) + 2\ddot{\mathbf{r}}_n^{nc}(t)\cdot\ddot{\mathbf{r}}_n^c(t)\Delta t^4 + \ddot{\mathbf{r}}_n^{nc}(t)^2\,\Delta t^4(+O(\Delta t^4)) = 0\end{aligned} \tag{11}$$

Where,

$$\mathbf{q}_n(t+\Delta t) \equiv 2\mathbf{r}_n(t) - \mathbf{r}_n(t-\Delta t) + \ddot{\mathbf{r}}_n^{nc}(t)\Delta t^2$$

and $\ddot{\mathbf{r}}_n^{nc}(t)$ and $\ddot{\mathbf{r}}_n^c(t)$ are accelerations from non-constraint and constraint forces, respectively ($\ddot{\mathbf{r}}_n(t) \equiv \dot{\mathbf{r}}_n^0(t) + \ddot{\mathbf{r}}_n^{nc}(t)$). From this equation, we can see equation (10) differs from the linearized equation of the matrix method only in $O(\Delta t^4)$ order terms. As a result, the first step equation of the matrix method is equivalent to equation (10) and we can use both to solve constraint forces. From another viewpoint, this means the matrix method can be non-iterative, i.e. linearizable with the constraint error of $O(\Delta t^4)$ when we incorporate it with the Verlet time integration scheme. This linearization with the Verlet scheme is consistent with the discussion in Appendix 1, but it can not always be true. For example, we can not use the linearized equation with the Gear algorithms which have an error order higher than $O(\Delta t^4)$. So, we do need iteration with these higher order schemes to slove the non-linear equation.

Let us consider the following Taylor expansion of $\mathbf{r}_n\ (t-\Delta t)$ about Δt.

$$\mathbf{r}_n(t-\Delta t) = \mathbf{r}_n(t) - \dot{\mathbf{r}}_n(t)\Delta t + (1/2)\ddot{\mathbf{r}}_n(t)\Delta t^2 + O(\Delta t^3) \tag{12}$$

If we use this expression in (10) or (11), we can get (8), i.e. (7). This means that the basic concept is the same as that described at the beginning of this section and moreover, the non-iterative version of the matrix method corresponds to the EEM algorithm with extra feedback terms which was described in section 3.1.

Now, we move to the case of leap-frog. The leap-frog algorithm can be expressed as follows.

$$\dot{\mathbf{x}}(t+\Delta t/2)=\dot{\mathbf{x}}(t-\Delta t/2)+\ddot{\mathbf{x}}(t)\Delta t(+O(\Delta t^3)) \tag{13a}$$

$$\mathbf{x}(t+\Delta t)=\mathbf{x}(t)+\dot{\mathbf{x}}(t+\Delta t/2)\Delta t(+O(\Delta t^3)) \tag{13b}$$

We can get the following equation for $\mathbf{r}_n(t+\Delta t)$ from the above equations.

$$\mathbf{r}_n(t+\Delta t)=\mathbf{r}_n(t)+\dot{\mathbf{r}}_n(t-\Delta t/2)\Delta t+\ddot{\mathbf{r}}_n(t)\Delta t^2(+O(\Delta t^3)) \tag{14}$$

Here, the third-order local error terms in this equation do not cancel. We note the global error order of the leap-frog algorithm is the same as that of the Verlet algorithm (i.e. $O(\Delta t^4)$) because the Verlet algorithm (equation 9) for the positions is recovered with a few manipulations from equations (13a) and (13b) [11]. From this equation and (1), the equation for $\chi_n(t+\Delta t)$ can be expressed as follows.

$$\begin{aligned}\chi_n(t+\Delta t)&=\mathbf{r}_n^2(t)-\mathbf{d}_n^2+2\mathbf{r}_n(t)\cdot\dot{\mathbf{r}}_n(t-\Delta t/2)\Delta t\\&\quad+(2\mathbf{r}_n(t)\cdot\ddot{\mathbf{r}}_n(t)+\dot{\mathbf{r}}_n^2(t-\Delta t/2))\Delta t^2(+O(\Delta t^3))=0\end{aligned} \tag{15}$$

The constraint error of this equation is $O(\Delta t^3)$ i.e., it is the same as the error order of equation (14). Also in this case, we can get (7) if we substitute the following Taylor expansion of $\mathbf{r}_n(t-\Delta t)$ into the former equation.

$$\dot{\mathbf{r}}_n(t-\Delta t/2)=\dot{\mathbf{r}}_n(t)-\ddot{\mathbf{r}}_n(t)\Delta t/2+O(\Delta t^2) \tag{16}$$

In Appendix 2, we give a detailed description of the algorithm implementation for a simple example.

4. TEST SIMULATION RESULTS

First, we did test simulations of the *n*-butane system to check the NIMM. The *n*-butane model we used is a united atom model in which we constrained the bond lengths only. Force field parameters were taken from these of GROMOS [12]. The calculations were done with a 32 molecule system under periodic boundary conditions and near the boiling point. The initial condition was selected to be accurate enough in constraint error. In the calculations, total energy was conserved with enough accuracy and no sign of numerical instability was observed in a simulation of many tens of thousands of steps. Figure 1 shows results on the relationship between time step and root mean square (R.M.S.) constraint error (the average of all constraints and 100 time steps). The constraint errors are proportional to Δt^4 for the Verlet scheme and Δt^3 for the leap-frog scheme. Absolute constraint error is on the order of 10^{-5}–10^{-6} at the ordinary value of a time step (a few femto seconds). This value is the same order as in ordinary SHAKE runs. Moreover, this accuracy is automatically obtained without any user-defined parameters such as the feedback parameters in the method by Baranyai and Evans.

Then, to compare NIMM with SHAKE in practical applications, we did test simulations of the liquid crystal molecule 5CB (4-*n*-pentyl-4'-cyanobiphenyl) [13]. The test simulations were done in a system of 64 5CB molecules (total 1216 atoms including united atoms) under periodic boundary conditions at isotropic states. And the leap-

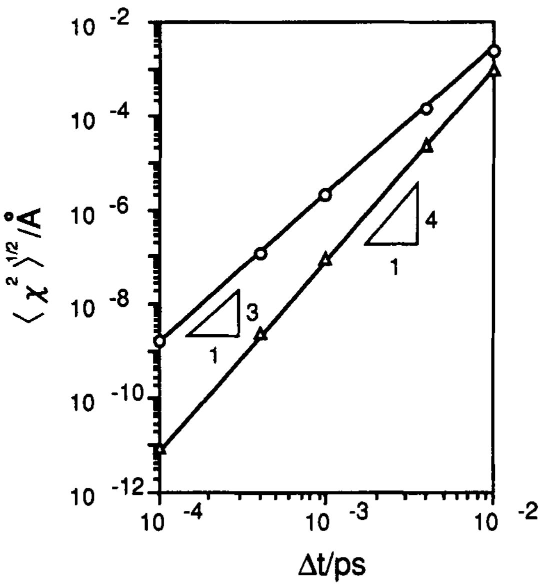

Figure 1 Relationship between time step (Δt) and R.M.S. constraint errors ($\langle \chi^2 \rangle^{1/2}$) in log–log plot (△ for Verlet, ○ for leap-frog).

frog integration time step of $2fs$ was used in a total of 100 steps of constant energy simulations with bond length constraints. The constraint part timing results from this simulation were obtained by a Silicon Graphics IRIS 4D/340 multiple processor (4CPUs) workstation and a Hitachi S-820/60 vector supercomputer and they are summarized in table 1. NIMM is 5.6 and 4.6 items faster than SHAKE by the IRIS 4D/340 and S-820/60, respectively. (Note that the current implementation of NIMM was not extensively tuned to the machines above.) In this test simulation, R.M.S. constraint errors and energy fluctuations (shown in table 2) are similar in the two methods.

Table 1 Comparison of constraint part CPU time (in seconds) between SHAKE and NIMM.

Machine	SHAKE	NIMM
IRIS 4D/340 (with 4CPUs)	9.14	1.62
Hitachi S-820/60	8.00	1.73

Table 2 Comparison of R.M.S. constraint errors and energy fluctuations between SHAKE and NIMM.

Item	SHAKE	NIMM
R.M.S. constraint errors (Å)	3.46×10^{-5}	2.54×10^{-5}
R.M.S. energy fluctuations (%)	2.06	1.95

5. CONCLUSIONS

We showed the matrix method can be non-iterative when we incorporated it with the Verlet and leap-frog integration schemes with the constraint error of the same order as the (local) error order of these schemes. Moreover, the resultant non-iterative matrix method (NIMM) is also the natural extension of the EEM algorithm to one with the matrix method-(or SHAKE-) like error handling functions. In this NIMM algorithm, the constraint force is calculated by a simple matrix calculation without iteration. This makes it more vectorisable and parallelisable than the iterative procedure of the matrix method and SHAKE, and more suited to vector and/or parallel machines.

We checked the NIMM with simulations of *n*-butane and the liquid crystal molecule 5CB. For 5CB, NIMM was about 5 times faster (in the constraint part CPU time) than SHAKE when either a multiple processor workstation or a vector supercomputer were used. In these test simulations, the same order of absolute constraint accuracy in ordinary SHAKE runs was obtained by NIMM and no user-defined parameters were needed to achieve this accuracy. These timing results are only one aspect in the performance comparison between NIMM and SHAKE, but they showed that at least for a small molecule like 5CB, NIMM would be preferred over SHAKE.

Acknowledgements

We are much indebted to the referees for the critical readings of the manuscript. One of us (M.Y.) would like to thank Professor W.F. van Gunsteren for helpful discussions and the Laboratory of Biophysical Chemistry of University of Groningen for their hospitality during his stay in 1990–1991.

APPENDICES

1. *Order of the Error*

Equation (4) can formally be written in matrix form as follows.

$$A(\mathbf{r}_n(t))\boldsymbol{\lambda}(t) = \mathbf{b}(t) - \boldsymbol{\chi}(t)\Delta t^{-2} - \dot{\boldsymbol{\chi}}(t)\Delta t^{-1} + \cdots \qquad (A1)$$

Where, the Lagrange multiplier $\boldsymbol{\lambda}$ is and an N (total number of constraints in a molecule) dimensional vector; $\boldsymbol{\chi}$, $\dot{\boldsymbol{\chi}}$ is a vector representation of equation (1); and A is a coefficient matrix. Vector $\mathbf{b}$ is defined by the following equation.

$$\mathbf{b}(t) \equiv -\mathbf{r}_n(t).\boldsymbol{\alpha}_n(t) - \dot{\mathbf{r}}_n^2(t) \qquad (A2)$$

where

$$\boldsymbol{\alpha}_n(t) = \mathbf{f}_i^{nc}(t)/m_i - \mathbf{f}_j^{nc}(t)/m_j;$$

$\mathbf{f}^{nc}(t)$ is a non-constraint force; and m is atomic mass. The precise Lagrange multiplier at time $t(\lambda_T)$ is given in equation (6) (the equation solved in the EEM algorithm) and it can be expressed in matrix form as follows.

$$A(\mathbf{r}_n(t))\lambda_T(t) = \mathbf{b}(t) \tag{A3}$$

The λ obtained from $(A1)$ can be written down as follows.

$$\lambda(t) = \lambda_T(t) + \lambda'(t) \tag{A4}$$

Here, λ' is an additional term to suppress constraint error accumulation.

Let us assume all valuables are normalized and the coefficient matrix is $O(1)$, the order of this λ' can be expressed as follows by comparing (A1) with (A3).

$$O(\lambda') = O(\chi\Delta t^{-2}) \tag{A5}$$

On the other hand, λ of (A4) is used in the time integration scheme of the coordinates and the order of the additional term is $O(\lambda'\Delta t^2)$. This additional term must be the same order as the error of the time integration scheme because it was introduced to cancel the error of the integration scheme. So, if we assume the error order of the coordinate integration scheme is $O(\varepsilon)$, the following equation must be fulfilled.

$$O(\varepsilon) = O(\lambda'\Delta t^2) \tag{A6}$$

By substituting (A5) into this equation, we get the following equation.

$$O(\chi) = O(\varepsilon) \tag{A7}$$

As a result, the constraint error order must be the same as the error order of the time integration scheme.

2. *Example of a Practical Algorithm*

In this appendix, we demonstrate the algorithm for the integration step with NIMM for the simple example of n-butane. The united CH_3, CH_2 atoms are numbered 1 to 4 and the constraints are numbered 1 to 3 as follows:

$$\begin{aligned} \mathbf{r}_1(t) &= \mathbf{x}_2(t) - \mathbf{x}_1(t) \\ \mathbf{r}_2(t) &= \mathbf{x}_3(t) - \mathbf{x}_2(t) \\ \mathbf{r}_3(t) &= \mathbf{x}_4(t) - \mathbf{x}_3(t) \end{aligned} \tag{A8}$$

We must solve the matrix equation $A\lambda = \mathbf{b}$ to get the Lagrange multiplier λ. The actual matrix for the Verlet (A^V) and leap-frog (A^{l-f}) algorithms are given by the following (Note whereas A^V is asymmetric, A^{l-f} is a symmetric matrix).

$$A^V = \begin{pmatrix} 2\mu_{21}\mathbf{q}_1(t+\Delta t)\cdot\mathbf{r}_1(t) & -\frac{2}{m_2}\mathbf{q}_1(t+\Delta t)\cdot\mathbf{r}_2(t) & 0 \\ -\frac{2}{m_2}\mathbf{q}_2(t+\Delta t)\cdot\mathbf{r}_1(t) & 2\mu_{32}\mathbf{q}_2(t+\Delta t)\cdot\mathbf{r}_2(t) & -\frac{2}{m_3}\mathbf{q}_2(t+\Delta t)\cdot\mathbf{r}_3(t) \\ 0 & -\frac{2}{m_3}\mathbf{q}_3(t+\Delta t)\cdot\mathbf{r}_2(t) & 2\mu_{43}\mathbf{q}_3(t+\Delta t)\cdot\mathbf{r}_3(t) \end{pmatrix} \tag{A9a}$$

$$A^{l-f} = \begin{pmatrix} \mu_{21}\mathbf{r}_1{}^2(t) & -\frac{1}{m_2}\mathbf{r}_1(t)\cdot\mathbf{r}_2(t) & 0 \\ -\frac{1}{m_2}\mathbf{r}_2(t)\cdot\mathbf{r}_1(t) & \mu_{32}\mathbf{r}_2{}^2(t) & -\frac{1}{m_3}\mathbf{r}_2(t)\cdot\mathbf{r}_3(t) \\ 0 & -\frac{1}{m_3}\mathbf{r}_3(t)\cdot\mathbf{r}_2(t) & \mu_{43}\mathbf{r}_3{}^2(t) \end{pmatrix} \qquad \text{(A9b)}$$

where

$$\mu_{ij} \equiv \frac{1}{m_i} + \frac{1}{m_j}$$

$$\mathbf{q}_n(t+\Delta t) \equiv 2\mathbf{r}_n(t) - \mathbf{r}_n(t-\Delta t) + \ddot{\mathbf{r}}_n^{nc}(t)\Delta t^2$$

The corresponding R.H.S. vectors are given by the following.

$$\mathbf{b}^V = \begin{pmatrix} (\mathbf{q}_1^2(t+\Delta t) - \mathbf{d}_1^2)/\Delta t^2 \\ (\mathbf{q}_2^2(t+\Delta t) - \mathbf{d}_2^2)/\Delta t^2 \\ (\mathbf{q}_3^2(t+\Delta t) - \mathbf{d}_3^2)/\Delta t^2 \end{pmatrix} \qquad \text{(A10a)}$$

$$\mathbf{b}^{l-f} = \begin{pmatrix} -\alpha_1(t)\cdot\mathbf{r}_1(t) - \frac{\mathbf{r}_1^2(t) - \mathbf{d}_1^2}{2\Delta t^2} - \frac{\mathbf{u}_1(t-\Delta t/2)\cdot\mathbf{r}_1}{\Delta t} - \frac{\mathbf{u}_1^2(t-\Delta t/2)}{2} \\ -\alpha_2(t)\cdot\mathbf{r}_2(t) - \frac{\mathbf{r}_2^2(t) - \mathbf{d}_2^2}{2\Delta t^2} - \frac{\mathbf{u}_2(t-\Delta t/2)\cdot\mathbf{r}_2(t)}{\Delta t} - \frac{\mathbf{u}_2^2(t-\Delta t/2)}{2} \\ -\alpha_3(t)\cdot\mathbf{r}_3(t) - \frac{\mathbf{r}_3^2(t) - \mathbf{d}_3^2}{2\Delta t^2} - \frac{\mathbf{u}_3(t-\Delta t/2)\cdot\mathbf{r}_3(t)}{\Delta t} - \frac{\mathbf{u}_3^2(t-\Delta t/2)}{2} \end{pmatrix} \qquad \text{(A10b)}$$

where

$$\mathbf{u}_n(t-\Delta t/2) = \mathbf{v}_i(t-\Delta t/2) - \mathbf{v}_j(t-\Delta t/2)$$

The matrix equations are solved for examples by the LU decomposition method, yielding a vector $\lambda(t)$, which determines the constraint forces $\mathbf{f}^c$ (t) as follows.

$$\begin{aligned} \mathbf{f}_1^c(t) &= -\lambda_1(t)\mathbf{r}_1(t) \\ \mathbf{f}_2^c(t) &= \lambda_1(t)\mathbf{r}_1(t) - \lambda_2\mathbf{r}_2(t) \\ \mathbf{f}_3^c(t) &= \lambda_2(t)\mathbf{r}_2(t) - \lambda_3\mathbf{r}_3(t) \\ \mathbf{f}_4^c(t) &= \lambda_2(t)\mathbf{r}_3(t) \end{aligned} \qquad \text{(A11)}$$

Then, the usual updates follow by applying equations (9) and (13), and replacing $\mathbf{f}(t)$ by $\mathbf{f}^{nc}(t) + \mathbf{f}^c(t)$ (here $\mathbf{f}^{nc}(t)$ stands for non-constraint forces).

References

[1] J. P. Ryckaert, G. Ciccotti and H. J. C. Berendsen, "Numerical Integration of the Cartesian Equation of Motion of a System with Constraints: Molecular Dynamics of *n*-Alkanes", *J. Comp. Phys.*, **23**, 327 (1977).

[2] M. K. Memon, R. W. Hockney and S. K. Mitra, "Molecular Dynamics with Constraints", *J. Comp. Phys.*, **43**, 345 (1981).

[3] H. C. Andersen, "Rattle: A "Velocity" version of the Shake Algorithm for Molecular Dynamics Calculations", *J. Comp. Phys.*, **52**, 24 (1983).

[4] R. Edberg, D. J. Evans and G. P. Morriss, "Constrained Molecular Dynamics: Simulations of Liquid Alkanes with a New Algorithm ", *J. Chem. Phys.*, **84**, 6933 (1986).

[5] A. Baranyai and D. J. Evans, "New Algorithm for Constrained Dynamics simulation of Liquid Benzene and Naphthalene" *Molec. Phys.*, **70**, 53 (1990).

[6] F. Muller-Plathe and D. Brown, "Multi-colour algorithms in molecular simulation: vectorisation and parallelisation of internal forces and constraints", *Comput. Phys. Commun.*, **64**, 7 (1991).

[7] J. Orban and J. P. Ryckaert, "Methods in Molecular Dynamics", *Report on CECAM Workshop (unpublished)* (1974).

[8] J. E. Mertz,D. J. Tobias, C. L. Brooks III, and U. C. Singh, "Vector and Parallel Algorithms for the Molecular Dynamics Simulation of Macromolecules on Shared-Memory Computers", *J. Comp. Chem.*, **12**, 1270 (1991).

[9] A. R. C. Raine, "Systolic Loop Methods for Molecular Dynamics Simulation, Generalized for Macromolecules", *Mol. Sim.*, **7**, 59 (1991).

[10] S. L. Lin, J. Mellor-Crummey, B. M. Pettitt and G. N. Phillips, Jr., "Molecular Dynamics on a Distributed-Memory Multiprocessor", *J. Comp. Chem.*, **13**, 1022 (1992).

[11] W. F. van Gunsteren and H. J. C. Berendsen, "A Leap-frog Algorithm for Stochastic Dynamics", *Mol. Sim.*, **1**, 173 (1988).

[12] W. F. van Gunsteren and H. J. C. Berendsen, *GROMOS manual*, BIOMOS BV (1987).

[13] S. J. Picken, W. F. van Gunsteren, P.TH. van Duijnen and W.H. de Jew, "A Molecular Dynamics Study of the Nematic Phase of 4-*n*-pentyl-4'-cyanobiphenyl", *Liq. Cryst.*, **6**, 357 (1989).