3 LINCS: a linear constraint solver for molecular simulations

B. Hess, H. Bekker, H. J. C. Berendsen and J. G. E. M. Fraaije

J. Comp. Chem. 18: 1463-1472 (1997)

In this chapter we present a new LINear Constraint Solver (LINCS) for molecular simulations with bond constraints. The algorithm is inherently stable, as the constraints themselves are reset instead of derivatives of the constraints, thereby eliminating drift. Although the derivation of the algorithm is presented in terms of matrices, no matrix matrix multiplications are needed and only the nonzero matrix elements have to be stored, making the method useful for very large molecules. At the same accuracy, the LINCS algorithm is 3 to 4 times faster than the SHAKE algorithm. Parallelization of the algorithm is straightforward.

3.1 Introduction

In classical molecular simulation methods, such as Molecular Dynamics (MD) or Langevin Dynamics (LD), the time step is limited by bond oscillations. These oscillations have a relatively high frequency and low amplitude. By replacing the bond vibrations by holonomic constraints the time step in molecular simulations can be increased by a factor of 4. Constraints are often considered a more faithful representation of the physical behavior of bond vibrations which are almost exclusively in their vibrational ground state.

Because bonds in molecules are coupled, resetting coupled constraints after an unconstrained update is a nonlinear problem. Many algorithms have been proposed for solving this problem. The most widely used algorithm for large molecules is SHAKE [9]. For small molecules SETTLE is a faster algorithm [26]. Both these methods reset bonds (and angles) to prescribed values by moving the bonded particles parallel to the old bond directions. SETTLE solves this problem analytically, but for larger molecules this is too complicated. SHAKE is an iterative method. Sequentially all the bonds are set to the correct length. Because the bonds are coupled, this procedure has to be repeated until the desired accuracy is reached. The number of iterations can be decreased using over relaxation [27]. SHAKE is simple and numerically stable since it resets all constraints within a prescribed tolerance. It has been shown that, when the iteration is carried to convergence, SHAKE in combination with Verlet is simplectic and time reversible [28]. SHAKE has the drawback that no solutions may be found when displacements are large: because the coupled bonds are handled one by one, correcting one bond may tilt a coupled bond so far that the method does not converge. Due to its iterative nature it is difficult to parallelize SHAKE [29]. Therefore there is need for a fast, reliable constraint algorithm in molecular simulations.

The constraint problem reduces to a linear matrix equation if the second derivatives of the constraint equations are set to zero. However, in a finite discretization scheme corrections are necessary to achieve accuracy and stability. Several algorithms have been proposed, including the EEM method of Edberg et al. [30] who applied a penalty function, a modification of EEM by Baranyai and Evans [31] applying damping corrections and the 'non-iterative SHAKE' method of Yoneya et al. [32, 33], approximating the constraint equations rather than their second derivatives.

In this chapter we present a new LINear Constraint Solver (LINCS) for molecular simulations with bond constraints. LINCS is similar to EEM with a few practical differences. We have implemented an efficient solver for the matrix equation, a velocity correction that prevents rotational lengthening and a length correction that improves accuracy and stability. The LINCS algorithm is worked out in detail for application to a Leap-Frog or Verlet-type algorithm for molecular dynamics. The application to position Langevin Dynamics is briefly discussed. The sparse constraint coupling matrix is inverted using a power series, which converges rapidly because the constraining influence is relatively local; therefore parallelization of the algorithm is straightforward. The resulting method is 3 to 4 times faster than SHAKE at the same accuracy.

3.2 The projection method

We consider a system of N particles, with positions given by a $3N$ vector $\mathbf{r}(t)$. The equations of motion are given by Newton's law

$$\frac{\mathrm{d}^2\mathbf{r}}{\mathrm{d}t^2} = \mathbf{M}^{-1}\mathbf{f} \tag{3.1}$$

where $\mathbf{f}$ is the $3N$ force vector and $\mathbf{M}$ is a $3N \times 3N$ diagonal matrix, containing the masses of the particles. In general a system is constrained by K time-independent constraint equations

$$g_i(\mathbf{r}) = 0 \qquad i = 1, \dots, K \tag{3.2}$$

As was shown by H. Bekker [34] the constrained system can still be described by $3N$ second order differential equations in Cartesian coordinates. The proof runs as follows.

The system is constrained according to the principal of least action [8]. In this approach the constraints are added as a zero term to the potential $\mathbf{V}(\mathbf{r})$, multiplied by Lagrange multipliers $\lambda_i(t)$

$$-\mathbf{M}\frac{\mathrm{d}^2\mathbf{r}}{\mathrm{d}t^2} = \frac{\partial}{\partial \mathbf{r}}(\mathbf{V} - \boldsymbol{\lambda} \cdot \mathbf{g}) \tag{3.3}$$

A new notation is introduced for the gradient matrix of the constraint equations which appears on the right hand side of the equation

$$B_{hi} = \frac{\partial g_h}{\partial r_i} \tag{3.4}$$

Notice that $\mathbf{B}$ is a $K \times 3N$ matrix, it contains the directions of the constraints. Equation (3.3) can now be simplified to give

$$-\mathbf{M}\frac{\mathrm{d}^2\mathbf{r}}{\mathrm{d}t^2} + \mathbf{B}^T\boldsymbol{\lambda} + \mathbf{f} = 0 \tag{3.5}$$

Because the constraint equations are zero, the first and second derivatives of the constraints are also zero

$$\frac{\mathrm{d}\mathbf{g}}{\mathrm{d}t} = \mathbf{B}\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}t} = 0 \tag{3.6}$$

$$\frac{\mathrm{d}^2\mathbf{g}}{\mathrm{d}t^2} = \mathbf{B}\frac{\mathrm{d}^2\mathbf{r}}{\mathrm{d}t^2} + \frac{\mathrm{d}\mathbf{B}}{\mathrm{d}t}\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}t} = 0 \tag{3.7}$$

To solve for $\boldsymbol{\lambda}$, left-multiply equation (3.5) with $\mathbf{BM}^{-1}$, and use (3.7) to get

$$\frac{\mathrm{d}\mathbf{B}}{\mathrm{d}t}\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}t} + \mathbf{BM}^{-1}\mathbf{B}^T\boldsymbol{\lambda} + \mathbf{BM}^{-1}\mathbf{f} = 0 \tag{3.8}$$

Thus the constraint forces are

$$\mathbf{B}^T\boldsymbol{\lambda} = -\mathbf{B}^T(\mathbf{BM}^{-1}\mathbf{B}^T)^{-1}\mathbf{BM}^{-1}\mathbf{f} - \mathbf{B}^T(\mathbf{BM}^{-1}\mathbf{B}^T)^{-1}\frac{\mathrm{d}\mathbf{B}}{\mathrm{d}t}\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}t} \tag{3.9}$$

Substituting this into (3.5) gives the constrained equations of motion. Using the abbreviation $\mathbf{T} = \mathbf{M}^{-1}\mathbf{B}^T(\mathbf{BM}^{-1}\mathbf{B}^T)^{-1}$ these are

$$\frac{\mathrm{d}^2\mathbf{r}}{\mathrm{d}t^2} = (\mathbf{I} - \mathbf{TB})\mathbf{M}^{-1}\mathbf{f} - \mathbf{T}\frac{\mathrm{d}\mathbf{B}}{\mathrm{d}t}\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}t} \tag{3.10}$$

$\mathbf{I} - \mathbf{TB}$ is a projection matrix that sets the constrained coordinates to zero, $\mathbf{BM}^{-1}\mathbf{f}$ is a K vector of second derivatives of the bond lengths in the direction of the bonds, $\mathbf{T}$ is a $3N \times K$ matrix that transforms motions in the constrained coordinates into motions in Cartesian coordinates, without changing the equations of motion of the unconstrained coordinates. The last term in (3.10) represents centripetal forces caused by rotating bonds. If the constraints are satisfied in the starting configuration, equation (3.10) will conserve the constraints.

3.3 A constrained leap-frog algorithm

A straightforward discretization in a leap-frog method uses a half time-step difference between the discretization of the first and last derivatives in (3.10)

$$\frac{\mathbf{v}_{n+\frac{1}{2}} - \mathbf{v}_{n-\frac{1}{2}}}{\Delta t} = (\mathbf{I} - \mathbf{T}_n\mathbf{B}_n)\mathbf{M}^{-1}\mathbf{f}_n - \mathbf{T}_n\frac{\mathbf{B}_n - \mathbf{B}_{n-1}}{\Delta t}\mathbf{v}_{n-\frac{1}{2}} \tag{3.11}$$

$$\frac{\mathbf{r}_{n+1} - \mathbf{r}_n}{\Delta t} = \mathbf{v}_{n+\frac{1}{2}} \tag{3.12}$$

The discretization of the last term in (3.11) at $t_{n-\frac{1}{2}}$ is the actual linearization of the problem. Because the right hand side of equation (3.11) does not depend on $t_{n+\frac{1}{2}}$ the new velocities can be calculated directly. However, the half time-step difference removes the correction for centripetal forces. This correction has to be done afterwards. If in equation (3.11) the term $\mathbf{B}_{n-1}\mathbf{v}_{n-\frac{1}{2}}$ is zero, the equation simplifies to

$$\mathbf{v}_{n+\frac{1}{2}} = (\mathbf{I} - \mathbf{T}_n\mathbf{B}_n)\left(\mathbf{v}_{n-\frac{1}{2}} + \Delta t\mathbf{M}^{-1}\mathbf{f}_n\right) \tag{3.13}$$

Since $\mathbf{I} - \mathbf{T}_n\mathbf{B}_n$ is the projection matrix that sets the constrained coordinates to zero, thus $\mathbf{B}_n\mathbf{v}_{n+\frac{1}{2}} = 0$. This proves that the velocities in the bond directions stay zero. Thus if we set $\mathbf{B}_0\mathbf{v}_{\frac{1}{2}} = 0$, equation (3.13) can be used instead of equation (3.11).

The derived algorithm is correct, but not stable. Numerical errors can accumulate, which leads to drift, because the second derivatives of the constraints were set to zero instead of the constraints themselves. This problem can be overcome by making a small change to the expression for the constraint forces. For holonomic constraints the constraint equations can be chosen as

$$g_i(\mathbf{r}) = |\mathbf{r}^{i_1} - \mathbf{r}^{i_2}| - d_i = 0 \qquad i = 1, \ldots, K \tag{3.14}$$

where d_i is the length of the bond between atoms i_1 and i_2, the vectors with superscript are the positions of atoms. To get a stable and efficient algorithm a term is added to equation (3.13)

$$\mathbf{v}_{n+\frac{1}{2}} = (\mathbf{I} - \mathbf{T}_n\mathbf{B}_n)\left(\mathbf{v}_{n-\frac{1}{2}} + \Delta t \mathbf{M}^{-1}\mathbf{f}_n\right) - \frac{1}{\Delta t}\mathbf{T}_n(\mathbf{B}_n\mathbf{r}_n - \mathbf{d}) \tag{3.15}$$

The added term should physically be zero, as it is the real distance between bonded atoms minus the prescribed bond-lengths. Of course in a simulation this term is almost never exactly zero, so adding this term eliminates accumulation of numerical errors and makes the method stable. Substituting (3.15) into (3.12) gives the constrained new positions

$$\mathbf{r}_{n+1} = (\mathbf{I} - \mathbf{T}_n\mathbf{B}_n)\left(\mathbf{r}_n + \Delta t \mathbf{v}_{n-\frac{1}{2}} + (\Delta t)^2\mathbf{M}^{-1}\mathbf{f}_n\right) + \mathbf{T}_n\mathbf{d} \tag{3.16}$$

This is actually how the method is implemented. It calculates the new positions as in the unconstrained case. Then it sets the components of the distances in the direction stored in $\mathbf{B}_n$ exactly to the prescribed lengths. The new velocity is calculated from the positions with (3.12). The corresponding Verlet algorithm is derived by substituting $\mathbf{v}_{n-\frac{1}{2}}$ from (3.12) into (3.16).

The main drawback of the method presented so far is that it does not set the real bond lengths to the prescribed lengths, but the projection of the new bonds onto the old directions of the bonds. Thus the bond lengths are increased by a factor $1/\cos\theta$, where θ is the angle over which a bond rotates in one time step. In an 'average' MD-simulation of a protein or peptide with a 2 fs time step the relative deviation is 0.0005 (r.m.s.). Notice that the error over n steps is still 0.0005, because accumulation of errors is prevented by adding the correction term. To get a higher accuracy another projection can be applied, again using the old bond directions.

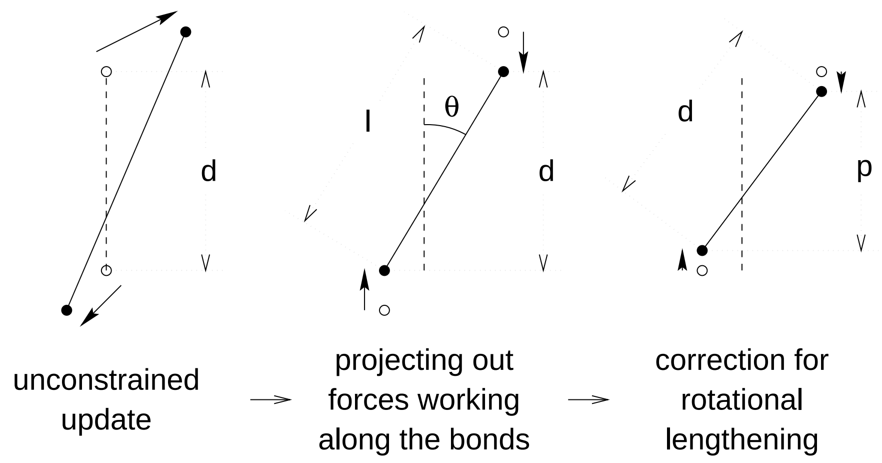
To correct for the rotation of bond i, the projection of the bond on the old direction is set to

$$p_i = \sqrt{2d_i^2 - l_i^2} \tag{3.17}$$

where l_i is the bond length after the first projection. The corrected positions are

$$\mathbf{r}^*_{n+1} = (\mathbf{I} - \mathbf{T}_n\mathbf{B}_n)\mathbf{r}_{n+1} + \mathbf{T}_n\mathbf{p} \tag{3.18}$$

Figure 3.1 shows a schematic picture of how the algorithm works for one bond. For



d
l
θ
d
d
p
unconstrained update
projecting out forces working along the bonds
correction for rotational lengthening

Figure 3.1: Schematic picture showing the three position updates needed for one time step. The dashed line is the old bond of length d, the solid lines are the new bonds. $l = d\cos\theta$ and $p = (2d^2 - l^2)^{\frac{1}{2}}$.

proteins with a 2 fs time step the relative accuracy is 0.000001 (r.m.s.). The error is not zero, because bonds are rotated slightly due to the coupling. So actually this correction procedure should be iterative, but the accuracy after one iteration is high enough for all purposes. The constraints can not be reset by moving atoms in the old bond directions in the extreme case that for a certain bond l_i^2 is larger than $2d_i^2$. However setting p_i to zero results in a 'solution' with bond lengths that are as close as possible to the prescribed lengths.

Just like SHAKE, LINCS calculates the new constrained positions from the old positions and the new unconstrained positions. Thus the same methods can be used to calculate the virial and free energy differences. However the free energy calculation for the constraints can be implemented simply and efficiently in the LINCS algorithm. The formulas are given in the appendix.

3.4 Implementation

Equation (3.16) is of the shape

$$\mathbf{r}_{n+1} = (\mathbf{I} - \mathbf{T}_n\mathbf{B}_n)\mathbf{r}_{n+1}^{unc} + \mathbf{T}_n\mathbf{d} =$$
$$\mathbf{r}_{n+1}^{unc} - \mathbf{M}^{-1}\mathbf{B}_n(\mathbf{B}_n\mathbf{M}^{-1}\mathbf{B}_n^T)^{-1}(\mathbf{B}_n\mathbf{r}_{n+1}^{unc} - \mathbf{d}) \tag{3.19}$$

where $\mathbf{r}_{n+1}^{unc}$ are the new positions after an unconstrained update. Half of the CPU time goes to inverting the constraint coupling matrix $\mathbf{B}_n\mathbf{M}^{-1}\mathbf{B}_n^T$, which has to be done every time step. This $K \times K$ matrix has $1/m_{i_1} + 1/m_{i_2}$ on the diagonal. The off-diagonal elements are only non-zero when two bonds are connected, then the element is $\cos\phi/m_c$, where m_c is the mass of the atom connecting the two bonds and ϕ is the angle between the bonds. To make the inversion easier a $K \times K$ matrix $\mathbf{S}$ is introduced which is the inverse square root of the diagonal of $\mathbf{B}_n\mathbf{M}^{-1}\mathbf{B}_n^T$

$$\mathbf{S} = \text{Diag}\left(\sqrt{\frac{1}{m_{1_1}} + \frac{1}{m_{1_2}}}, \ldots, \sqrt{\frac{1}{m_{K_1}} + \frac{1}{m_{K_2}}}\right) \tag{3.20}$$

This matrix is used to convert the diagonal elements of the coupling matrix to one

$$\begin{aligned}(\mathbf{B}_n\mathbf{M}^{-1}\mathbf{B}_n^T)^{-1} &= \mathbf{S}\mathbf{S}^{-1}(\mathbf{B}_n\mathbf{M}^{-1}\mathbf{B}_n^T)^{-1}\mathbf{S}^{-1}\mathbf{S} \\ &= \mathbf{S}(\mathbf{S}\mathbf{B}_n\mathbf{M}^{-1}\mathbf{B}_n^T\mathbf{S})^{-1}\mathbf{S} = \mathbf{S}(\mathbf{I} - \mathbf{A}_n)^{-1}\mathbf{S}\end{aligned} \tag{3.21}$$

The matrix $\mathbf{A}_n$ is symmetric and sparse and has zeros on the diagonal. Thus a simple trick can be used to calculate the inverse

$$(\mathbf{I} - \mathbf{A}_n)^{-1} = \mathbf{I} + \mathbf{A}_n + \mathbf{A}_n^2 + \mathbf{A}_n^3 + \ldots \tag{3.22}$$

This inversion method is only valid if the absolute values of all the eigenvalues of $\mathbf{A}_n$ are smaller than one. In molecules with only bond constraints the connectivity is so low that this will always be true, even if ring structures are present. Problems can arise in angle-constrained molecules. By constraining angles with additional distance constraints multiple small ring structures are introduced. This gives a high connectivity, leading to large eigenvalues. For example, the largest eigenvalue for angle-constrained butane is 0.8, but for angle-constrained pentane this is 1.2. However, the projection method itself is still applicable, if the constraint coupling matrix is inverted with another method.

The inversion through an expansion is efficient, because the inverse itself is not needed, only the product of the inverse with a vector. This product can be calculated without multiplying matrices. $\mathbf{A}_n^{i+1}\mathbf{a}$ can be written as $\mathbf{A}_n(\mathbf{A}_n^i\mathbf{a})$, showing that this product can be calculated by multiplying vectors with $\mathbf{A}_n$.

The error made by truncating the series expansion in (3.22) is quite clear. The first power of $\mathbf{A}_n$ gives the coupling effects of neighboring bonds. The second power gives the coupling effects over a distance of two bonds, not only between bonds that are separated by one bond, but also the feedback of a bond on itself through neighboring bonds. The third power gives the third order coupling effects and so on. So truncating the series after a specified number of terms means neglecting all higher order coupling effects. In molecules with bond angles closer to 90 degrees than to 0 or 180, like proteins, the sum can be truncated after 4 terms. Little storage is needed, since only the non-zero elements of $\mathbf{A}_n$ have to be stored. $\mathbf{B}_n$ can be stored in a $3K$-array and some extra arrays are needed for temporary storage. Pseudo code for the algorithm is given in the appendix. The optimized FORTRAN code is implemented as a subroutine in the GROMACS software package [1, 3].

3.5 Parallelization

The inversion through a series expansion also makes parallelization easy. In one time step the bonds only influence each other when they are separated by less bonds than the highest order in the expansion; with the correction for the rotation this number is doubled. Because of this local coupling a decomposition method can be applied. This can be illustrated by a simple example. Consider a linear bond constrained molecule of 100 atoms to be simulated on a 2 processor computer, using rotation correction and an expansion to the second order of the matrix $(\mathbf{I}-\mathbf{A}_n)^{-1}$. Since the order of the expansion is two, bonds influence each other over a distance of at most 2*(2+1)=6 bonds. The forces can be calculated as in the unconstrained case, but the update of the positions and call of the LINCS algorithm must be done for atoms 1 to 56 and 44 to 100 on processor 1 and 2, respectively. After the update only the positions of atoms 1 to 50 should be passed from processor 1 to processor 2, as the new positions of atoms 51 to 56 on processor 1 are not accurate. The same holds for atoms 44 to 49 on processor 2. The result of this parallel procedure is identical to the single-processor case. For branched molecules like proteins the same procedure can be used.

3.6 Position Langevin dynamics

Consider a system of N over-damped particles. The equations of motion are given by a position Langevin equation

$$\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}t} = \mathbf{\Gamma}^{-1}\mathbf{f} + \sqrt{2\mathbf{\Gamma}^{-1}k_bT}\boldsymbol{\eta} \tag{3.23}$$

where $\mathbf{\Gamma}$ is the friction matrix and $\boldsymbol{\eta}$ a $3N$ noise vector with no time correlation, Gaussian distributed with $\mu = 0$ and $\sigma = 1$. The algorithm can be derived analogous to the second order case. The result is

$$\mathbf{r}_{n+1} = (\mathbf{I} - \mathbf{T}_n\mathbf{B}_n)(\Delta t\,\mathbf{\Gamma}^{-1}\mathbf{f}_n + \sqrt{2\Delta t\mathbf{\Gamma}^{-1}k_bT}\boldsymbol{\eta}) + \mathbf{T}_n\mathbf{d} \tag{3.24}$$

where $\mathbf{T}_n = \mathbf{\Gamma}^{-1}\mathbf{B}_n^T(\mathbf{B}_n\mathbf{\Gamma}^{-1}\mathbf{B}_n^T)^{-1}$. The nice property of a projection matrix $\mathbf{P} = \mathbf{I} - \mathbf{T}_n\mathbf{B}_n$ is that $\mathbf{P}^2 = \mathbf{P}$, thus the square root of a projection matrix is the projection matrix itself. This is used in (3.24) because the noise must be multiplied by the square root of the operator that works on the force. When the friction matrix $\mathbf{\Gamma}$ is diagonal, the implementation for leap-frog MD can be used, with the masses replaced by friction coefficients. The projection algorithm can also be applied to Monte Carlo simulations, since the distribution of the unconstrained coordinates is not affected by the projection.

algorithm	order	max. dev.	r.m.s. dev.	time (s)
LINCS	2	0.004 9	0.000 59	1.8
no rotation	4	0.004 8	0.000 49	2.0
correction	8	0.004 9	0.000 47	2.4
	2	0.000 32	0.000 082	2.2
LINCS	4	0.000 088	0.000 023	2.5
	8	0.000 016	0.000 003 8	3.3
		0.001 0	0.000 48	5.3
SHAKE		0.000 10	0.000 052	8.4
		0.000 010	0.000 005 5	12.4

Table 3.1: The relative deviation of the constraints in lysozyme (solvated in 4535 water molecules and 8 Cl^- ions) averaged over 500 steps: the maximum and the r.m.s. deviation shown for SHAKE and for LINCS without and with rotation correction. The water was constrained with SETTLE. The second column shows the order after which expansion (22) is truncated. The last column shows the total CPU time for the constraint algorithms. The total run time is 1300 seconds. Vacuum runs give almost identical numbers for the accuracy and CPU time of the constraint algorithms. The total run time of the vacuum system is about 90 seconds.

3.7 Results

A large test system was used to test how the accuracy of the constraints depends on the order of expansion (3.22) and to compare the speed of the LINCS algorithm with the SHAKE algorithm. Results are given for 1 ps MD runs on lysozyme (129 residues, 1345 bond constraints) with the GROMACS package. The protein was solvated in a box of 4535 water molecules and 8 Cl^- ions, with a Lennard-Jones cut-off of 1 nm and a twin-range Coulomb cut-off (1 and 1.8 nm). The water is constrained with the SETTLE algorithm. After energy minimization and 2 ps equilibration, test runs of 500 steps of 2 fs were performed. The test runs took 1300 seconds on one processor of an R8000 PowerChallenge. Table 3.1 and Figure 3.2 show the results for LINCS and SHAKE. Both routines use optimized FORTRAN code. Notice that the difference between the maximum deviation and the r.m.s. deviation is a factor 2 for SHAKE and a factor 4 for LINCS. Some runs were also performed without water, giving almost the same accuracy and CPU time for the constraint algorithms as with water (data not shown), but with a total run time of 90 seconds.

A small test system was used to test the accuracy of the simulation as a function of the accuracy of the constraints. A 32 residue $\beta\alpha\beta$ peptide was chosen, which could be simulated in vacuo without cutoff. Simulations in single precision of 4 ps were per-

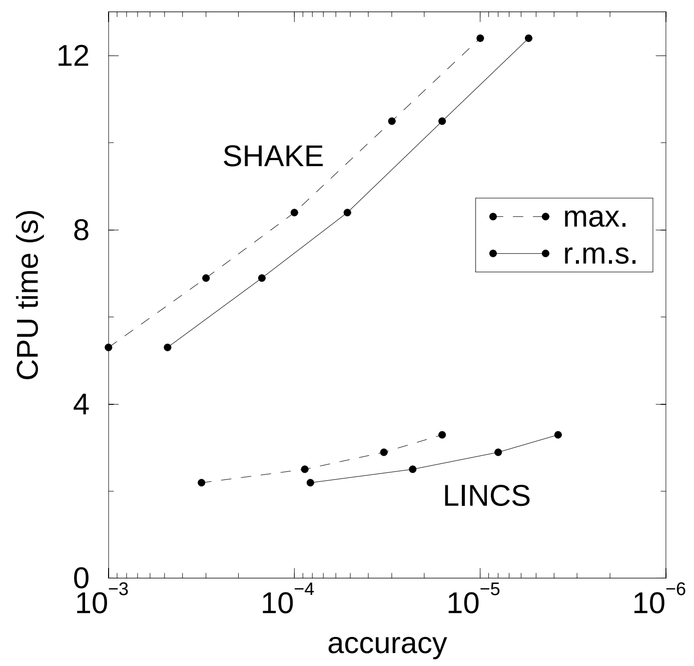


Figure 3.2: The CPU time for the constraint algorithms in lysozyme plotted as a function of the accuracy, i.e. the relative deviation of the constraints averaged over 500 steps. The dashed lines show the maximum deviation and the solid lines show the r.m.s. deviation for SHAKE and for LINCS.

algorithm	order	constraints	E_{total}		E_{kin}
		rms dev.	drift	rmsf	rmsf
			kJ/mol s	kJ/mol	kJ/mol
	2	0.000 088	-10.26	1.21	39
LINCS	4	0.000 027	-1.67	1.24	41
	8	0.000 004 9	0.25	1.23	41
		0.000 089	-6.81	1.23	39
SHAKE		0.000 027	-1.80	1.24	41
		0.000 004 9	0.14	1.27	41

Table 3.2: The accuracy of the constraints and the total energy in a 32 residue $\beta\alpha\beta$ peptide using LINCS and SHAKE. All entries are averages over 8 vacuum runs in single precision of 4 ps (2000 steps of 2 fs) with different starting structures, without temperature and pressure coupling and without cutoff. The second column shows the order after which expansion (22) is truncated. The third column shows the r.m.s. of the relative deviation of the constraints. The fourth column shows the slope of a straight line fitted to the total energy. The last columns show the r.m.s. fluctuations in the total and the kinetic energy.

formed without temperature and pressure coupling to check the conservation of total energy. The tolerance for SHAKE was chosen such that the r.m.s. of the relative deviation of the constraints was equal to LINCS case. Table 3.2 shows the results for different accuracies of the constraints for LINCS and SHAKE; each entry is averaged over 8 runs of 4 ps. The accuracy of the constraints only influences the drift, not the fluctuations around the drift. Except for the second order expansion there is little difference between LINCS and SHAKE.

3.8 Removing degrees of freedom

For a highly constrained degree of freedom with a relatively small mass it might be more convenient to remove the degree of freedom completely from the dynamic equation and replace it by an interaction site. The mass should be transferred to a neighboring particle. The only difference between a particle with a three degrees of freedom constrained and an interaction site is the mass distribution in the system. All thermodynamic properties remain unaffected, only dynamical properties will change. When the particle is light compared to particles it is constrained to, the change in the dynamical will be negligible. A reduction of degrees of freedom from coordinates $\mathbf{r}$ to $\mathbf{r}'$ and interaction sites $\mathbf{r}''$ goes as follows. We will only treat the case where all the degrees of freedom of M particles are removed. The $3N$ position vector $\mathbf{r}$ can then be separated in $3(N-M)$ dynamic degrees of freedom and $3M$ interaction site coordinates:

$$\mathbf{r} = \begin{pmatrix} \mathbf{r}' \\ \mathbf{r}'' \end{pmatrix} \tag{3.25}$$

The interaction site coordinates are a function of the dynamic coordinates:

$$\mathbf{r}'' = \mathbf{h}(\mathbf{r}') \tag{3.26}$$

Using this we can define a new potential $V'(\mathbf{r}')$, which is only a function of the dynamic coordinates:

$$V'(\mathbf{r}') = V\left(\begin{pmatrix} \mathbf{r}' \\ \mathbf{h}(\mathbf{r}') \end{pmatrix}\right) \tag{3.27}$$

We can now write the equations of motion for the reduced system:

$$M\frac{\mathrm{d}^2\mathbf{r}'}{\mathrm{d}t^2} = -\nabla_{\mathbf{r}'}V'(\mathbf{r}') = -\nabla_{\mathbf{r}'_i}V(\mathbf{r}) - \sum_{j=1}^{3M}\frac{\partial V(\mathbf{r})}{\partial r''_j}\nabla_{\mathbf{r}'}h_j(\mathbf{r}') \tag{3.28}$$

The implementation of interaction sites is straightforward. At the beginning of each time step the interaction site coordinates $\mathbf{r}''$ need to be calculated from the dynamic coordinates $\mathbf{r}'$ (equation (3.26)). After that the potential and forces can be calculated just like for the complete system. Before integrating the equations of motion the forces on the interaction sites need to be redistributed to the dynamical coordinates using the second term in equation (3.27). The pressure should be calculated using the redistributed forces.

In the most simple cases the interactions site coordinates are linear combinations of the other coordinates. Equation (3.26) then reduces to a matrix multiplication. Also the last gradient operation in the force redistribution (equation (3.27)) reduces to a multiplication with the same matrix. This method is computationally more efficient than solving the constraint equations.

The more interesting applications are those where it is impossible to use constraints. An example of this is a proton which is bonded to a heavy atom, which is in turn bonded to two heavy atoms, on either side of the proton to heavy-atom bond (see Figure 3.3). The bond vibrations in the system can be removed by constraining the bond lengths. Then the time step for the integration is limited by the angle vibrations of the protons. Fixing the angle of the proton by constraining the distance to particles A or B is incorrect, since this fixes one angle, but not the other. The proton will move out of the minimum of the sum of the two angle potentials as the angles change. When the mass is transferred to particle A, the proton can be treated as an interaction site. The function h can be chosen such that the interaction site is located at at a fixed distance from particle A, which should correspond to the equilibrium bond length. The direction the vector A-H is determined by its intersection with the line connecting particles B and C. The parameter α, which

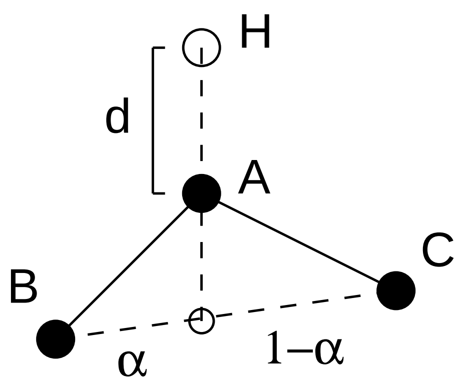


Figure 3.3: Construction of interaction site position H from the positions of particles A, B and C at a fixed distance d from particle A and with the angle determined by α.

determines the position of the intersection along the line B-C, should be chosen such that it reproduced the equilibrium angles. The fixed distance makes the function $h(\mathbf{r}')$ slightly more complicated than for a linear combination.

To construct tetrahedral hydrogens as interaction sites one needs cross products. This increases the computational cost, but poses no conceptual problems, since it just changes the function $h(\mathbf{r}')$. For simulations of condensed phases the computational cost of constructing the interaction sites and redistributing the forces will always be negligible compared to the evaluation of the non-bonded interactions. In MD simulations of proteins with constrained bond lengths the proton angle vibrations limit the time step to 2 fs. Constraining the angle of Hydroxyl protons and converting the other protons to interactions sites enables the use of a time step of 6 to 8 fs.

3.9 Discussion

LINCS and SHAKE both solve the same non linear problem: resetting coupled constraints after an unconstrained update. Just like SHAKE, LINCS is time reversible, because convergence can be obtained by including more matrices in the expansion and using multiple rotation corrections. In single precision one rotation correction is enough to get structures that only differ in machine precision with those obtained by SHAKE. The results show that the LINCS algorithm is 3 to 4 times faster than the SHAKE algorithm. In addition LINCS is has broader convergence conditions than SHAKE, and it can be easily parallelized. The latter is of major importance for simulations of large molecules on parallel computers. For efficient parallelization each molecule must be distributed over several processors, which prevents the straightforward use of the iterative SHAKE routine. Not only will the LINCS algorithm itself decrease the CPU time, but better load balancing of other routines can be achieved as well. In the case of a protein in water on one processor the CPU time for the constraint algorithm is negligible in comparison with the total run time, but LINCS still has the advantage that it has better convergence properties than SHAKE. In vacuum the decrease in total CPU time can be up to 10 percent.

The algorithm has also been tested on peptides and smaller molecules such as benzene, giving the same 3 to 4 fold decrease in CPU time. With position Langevin Dynamics SHAKE will not converge for large time steps. When LINCS is used the time step is no longer limited by the constraint algorithm but by physical conditions.

3.A Appendix: virial and free energy calculation

The virial formulas are given for the case without periodic boundary conditions, for a more detailed description see [35]. The 3×3 virial tensor Ξ is defined as

$$\Xi = -\frac{1}{2}\sum_i^N \mathbf{r}^i \otimes \mathbf{f}_{tot}^i \tag{A.1}$$

where $\mathbf{f}_{tot}^i$ is the total force working on atom i and $\mathbf{r}^i$ is the position of atom i, $\otimes$ denotes a direct product. For a constrained system a time t_n the virial is

$$\Xi_n = -\frac{1}{2}\sum_i^N \mathbf{r}_n^i \otimes \left(\mathbf{f}_n^i + \frac{m_i \Delta\mathbf{r}_n^i}{\Delta t^2}\right) \tag{A.2}$$

where $\mathbf{f}_n^i$ is the unconstrained force working on atom i at time t_n. The $\Delta\mathbf{r}_n$ term in the expression for the constraint force is given by

$$\Delta\mathbf{r}_n = \mathbf{r}_{n+1} - (\mathbf{r}_n + \Delta t \mathbf{v}_{n-\frac{1}{2}} + (\Delta t)^2 \mathbf{M}^{-1}\mathbf{f}_n) \tag{A.3}$$

One way to calculate the free energy difference between two systems or two states of a system is to make the Hamiltonian an analytical function of a coupling parameter α, where $\alpha = 0$ corresponds to system A and $\alpha = 1$ to system B [36]. The constraint distances may be a function of this coupling parameter

$$\mathbf{d}(\alpha) = (1-\alpha)\mathbf{d}_\mathrm{A} + \alpha\mathbf{d}_\mathrm{B} \tag{A.4}$$

where $\mathbf{d}_\mathrm{A}$ and $\mathbf{d}_\mathrm{B}$ are the lengths of the constraints in system A and B, respectively. For the contribution of the constraints to the free energy F^{con} the derivative with respect to α of the added term to the potential in (3.3) is needed

$$\frac{\mathrm{d}F^{con}}{\mathrm{d}\alpha} = -\frac{\partial(\boldsymbol{\lambda}\cdot\mathbf{g})}{\partial\alpha} = -\boldsymbol{\lambda}\cdot\frac{\partial\mathbf{g}}{\partial\alpha} = -\boldsymbol{\lambda}\cdot(\mathbf{d}_\mathrm{B} - \mathbf{d}_\mathrm{A}) \tag{A.5}$$

The discretized Lagrange multipliers are already calculated by the LINCS algorithm.

3.B Appendix: pseudo code

```
LINCS(x,xp,invmass,K,nrec,atom1,atom2,length,ncc,cmax,con,Sdiag,coef)
# x[N,3]       old positions of the atoms, N is the number of atoms
# xp[N,3]      input: new unconstrained positions,
#              output: the constrained positions
# invmass[N]   inverse masses of the atoms
# K            number of constraints
# nrec         order after which the power series is truncated
# atom1[K]     first atom of the constraint
# atom2[K]     second atom of the constraint
# length[K]    length of the bond
# ncc[K]       number of constraints connected to a constraint
# cmax         maximum number of constraints coupled to one constraint
# con[K,cmax]  index of the constraints connected to a constraint
# Sdiag[K]     1/sqrt(invmass[atom1]+invmass[atom2])
# coef[K,cmax] coef[i,j]=sign*invmass[c]*Sdiag[i]*Sdiag[con[i,j]],
#              c is the atom coupling constraints i and con[i,j],
#              if atom1[i]=atom1[con[i,j]] or atom2[i]=atom2[con[i,j]]
#              sign=-1 else sign=1
{ real B[K,3]    # directions of the constraints
  real A[K,cmax] # normalized constraint coupling matrix
  real rhs[2,K]  # right hand side of the matrix equation,
                 # two arrays needed to iterate
  real sol[K]    # solution array of the matrix equation
  int i,j,k,n,a1,a2
  real len,p

  for i=1 to K
  { for j=1 to 3
    { B[i,j] = x[atom1[i],j] - x[atom2[i],j] }
    len = sqrt(sqr(B[i,1]) + sqr(B[i,2]) + sqr(B[i,3]))
    for j=1 to 3
    { B[i,j] = B[i,j]/len }
  }
  for i=1 to K
  { for n=1 to ncc[i]
    { k = con[i,n]
      A[i,n] = coef[i,n] *
                 (B[i,1]*B[k,1] + B[i,2]*B[k,2] + B[i,3]*B[k,3]) }
    a1 = atom1[i]
    a2 = atom2[i]
    rhs[1,i] = Sdiag[i] * (B[i,1]*(xp[a1,1] - xp[a2,1]) +
                           B[i,2]*(xp[a1,2] - xp[a2,2]) +
                           B[i,3]*(xp[a1,3] - xp[a2,3]) - length[i])
    sol[i]=rhs[1,i]
  }
  call SOLVE(xp,invmass,K,nrec,atom1,atom2,ncc,con,Sdiag,B,A,rhs,sol)
```

```
     # CORRECTION FOR ROTATIONAL LENGTHENING
  for i=1 to K
  { a1 = atom1[i]
    a2 = atom2[i]
    p = sqrt(2*sqr(length[i]) - sqr(xp[a1,1] - xp[a2,1]) -
                                sqr(xp[a1,2] - xp[a2,2]) -
                                sqr(xp[a1,3] - xp[a2,3]))
    rhs[1,i] = Sdiag[i]*(length[i] - p)
    sol[i] = rhs[1,i]
  }
  call SOLVE(xp,invmass,K,nrec,atom1,atom2,ncc,con,Sdiag,B,A,rhs,sol)
}


SOLVE(xp,invmass,K,nrec,atom1,atom2,ncc,con,Sdiag,B,A,rhs,sol)
{ int i,j,n,rec,w,a1,a2

  w = 2
  for rec=1 to nrec
  { for i=1 to K
    { rhs[w,i] = 0
      for n=1 to ncc[i]
      { rhs[w,i] = rhs[w,i] + A[i,n]*rhs[3-w,con[i,n]] }
      sol[i] = sol[i] + rhs[w,i]
    }
    w = 3 - w
  }
  for i=1 to K
  { a1 = atom1[i]
    a2 = atom2[i]
    for j=1 to 3
    { xp[a1,j] = xp[a1,j] - invmass[a1]*B[i,j]*Sdiag[i]*sol[i]
      xp[a2,j] = xp[a2,j] + invmass[a2]*B[i,j]*Sdiag[i]*sol[i] }
  }
}
```