JCTC Journal of Chemical Theory and Computation

Constant Constraint Matrix Approximation: A Robust, Parallelizable Constraint Method for Molecular Simulations

Peter Eastman*,† and Vijay S. Pande‡

Department of Bioengineering and Department of Chemistry, Stanford University, Stanford, California 94305

Received June 16, 2009

Abstract: We introduce a new algorithm, the constant constraint matrix approximation (CCMA), for constraining distances in molecular simulations. It combines the best features of many existing algorithms while avoiding their defects: it is fast and stable, can be applied to arbitrary constraint topologies, and can be efficiently implemented on modern parallel architectures. We test it on a protein with bond length and limited angle constraints and find that it requires less than one-sixth as many iterations as SHAKE to converge.

Introduction

Rigid distance constraints are a popular method of increasing the integration step size in simulations of macromolecules. Using standard molecular force fields with no constraints, one is generally limited to a step size of about 1 fs. By constraining the lengths of bonds involving a hydrogen atom, one can increase the step size to 2 fs, thus doubling the amount of time that can be simulated in a given number of time steps. By constraining all bond lengths, as well as the most rapidly oscillating bond angles, the step size can be further increased to 4 fs.[1] Furthermore, due to the quantization of vibrational motion of bonds, constraints may be a more realistic representation of these stiff degrees of freedom than the harmonic forces conventionally used for them.[2]

Many algorithms have been suggested for implementing these constraints, but all of them have disadvantages that restrict their usefulness. The choice of which to use involves trade-offs between speed, stability, and range of applicability. For example, some algorithms are only useful for small molecules, or for short time steps, or for particular constraint topologies, or on particular computer architectures.

In this paper, we introduce a new constraint algorithm called the constant constraint matrix approximation (CCMA). It combines the best features of many existing algorithms while avoiding their disadvantages: it is fast, has good stability, can be applied to arbitrary sets of constraints, and can be efficiently implemented on a variety of modern computer architectures.

Background

Most constraint algorithms used in molecular simulations are based on (or are equivalent to) the method of Lagrange multipliers. For each interatomic distance that is to be constrained, one defines an error function

$$\sigma_i(\{\mathbf{r}_k\}) = |\mathbf{r}_m - \mathbf{r}_n| - d_i \tag{1}$$

where i is the index of the constraint, $\{\mathbf{r}_k\}$ is the set of all atomic coordinates, $\mathbf{r}_m$ and $\mathbf{r}_n$ are the positions of the two atoms whose distance is constrained, and d_i is the required distance between them. One then applies a constraint force $\lambda_i(t)$ to atoms m and n, which produces a combined displacement δ_i along the constraint direction during each time step. (Atom m is displaced by $(\delta_i/m_m)/(1/m_m + 1/m_n)$, while atom n is displaced by $-(\delta_i/m_n)/(1/m_m + 1/m_n)$, where m_m and m_n are the masses of the two atoms). The challenge at each time step is to calculate the vector of displacements $\boldsymbol{\delta}(t)$ such that $\sigma_i(\{\mathbf{r}_k\}) = 0$ for every constraint at the end of the time step.

This requires solving a system of nonlinear equations, which is typically done with an iterative algorithm such as Newton iteration:

$$\delta^{N+1} = \delta^N - \mathbf{J}^{-1}\boldsymbol{\sigma}^N \tag{2}$$

* Corresponding author e-mail: peastman@stanford.edu.

† Department of Bioengineering.

‡ Department of Chemistry.

where $\boldsymbol{\delta}^N$ is the vector of displacements calculated in the Nth iteration, $\boldsymbol{\sigma}^N$ is the vector of constraint errors in the Nth iteration, and $\mathbf{J}$ is the Jacobian matrix

$$\mathbf{J}_{ij} = \frac{\partial \sigma_i}{\partial \delta_j} \tag{3}$$

where i and j each run over all constraints in the system.

The most straightforward way to implement this is to explicitly construct the Jacobian matrix $\mathbf{J}$ and then invert it using a standard technique such as LU decomposition. This is, in fact, precisely what the M-SHAKE algorithm does.[3] The result is a stable algorithm that converges rapidly. Unfortunately, the time required to build and invert $\mathbf{J}$ increases rapidly with the number of constraints. For this reason, M-SHAKE is only useful for small molecules, not for macromolecules such as proteins and nucleic acids.

The LINCS algorithm takes a slightly different approach to performing the iteration.[4] Instead of explicitly calculating and inverting the Jacobian matrix, it represents $\mathbf{J}^{-1}$ as a power series. The usefulness of this approach depends on how quickly the series converges. For weakly connected sets of constraints, such as when only bond lengths are constrained, it converges quickly. For more strongly connected systems, such as when both bond lengths and angles are constrained, it converges more slowly, and may even fail to converge at all. For this reason, LINCS is generally only useful for bond length constraints.

Instead of accurately calculating $\mathbf{J}^{-1}$, one can instead try to approximate it with a different matrix $\mathbf{K}^{-1}$ that is easier to calculate. It can be shown that this approximation has no effect on the final result: because the displacements are uniquely determined by the requirement $\sigma_i(\{\mathbf{r}_k\}) = 0$, any convergent procedure is guaranteed to produce the same result.[5] On the other hand, the approximation will generally increase the number of iterations required and may also decrease the radius of convergence. How serious these problems are depends on how close $\mathbf{K}^{-1}$ is to $\mathbf{J}^{-1}$. The challenge is to find a matrix that is as close as possible to $\mathbf{J}^{-1}$ while still being easy to calculate.

The simplest approximation one might consider is the identity matrix. This is equivalent to assuming that all constraints are decoupled from each other, so that the force applied along one constraint has no effect on any other constrained distance. For weakly connected sets of constraints, this is actually not too bad an approximation and may produce a useful algorithm. For more strongly connected sets of constraints, however, it produces very poor convergence.

The SHAKE algorithm uses a small variation on this procedure that significantly improves its speed and stability.[5] It still computes δ_i independently for each constraint, but it processes them serially: each constraint force is calculated and the positions of its two atoms are updated before the next δ_i is calculated. As a result, each constraint implicitly sees the effect of all other constraints that were processed before it, but not those processed after it. This is equivalent to approximating $\mathbf{J}^{-1}$ using its true upper triangle, while setting all elements below the diagonal to zero. The result is significantly improved convergence at very little extra cost, which accounts for the popularity of this method.

SHAKE has an important disadvantage, however: it is an inherently serial algorithm. Each constraint must be fully processed and the atom positions updated before the next constraint can be processed. As a result, it is impossible to implement SHAKE efficiently on parallel architectures (multicore processors, graphics processing units, clusters, etc.). As parallel computing has become increasingly prevalent, the need for alternatives to SHAKE has become clear.

Another important class of constraint algorithms is ones that solve the constraint equations analytically rather than using an iterative method. The most important algorithm in this class is SETTLE, which uses an analytical solution for rigid water molecules.[6] It is both fast and extremely stable. As a result, it is clearly the method of choice for simulations involving explicit water molecules. Because it is applicable only to one very specific type of molecule, however, another algorithm must be used along with it to constrain the geometry of solute molecules.

A very different approach to implementing constraints is to work in internal coordinates.[7,8] Instead of representing the molecular conformation by the Cartesian coordinates of each atom, one instead represents it by the bond lengths and angles between atoms. It then becomes trivial to constrain those lengths and angles by keeping the corresponding coordinates fixed. This leads to a description of the system as a set of rigid bodies, each containing multiple atoms, connected by a minimal set of internal coordinates. Because the molecular force field depends on the Cartesian coordinates of atoms, it is necessary to convert positions and forces between Cartesian and internal coordinates as part of each time step. The algorithms for doing this are difficult to implement and add overhead to each time step. They also involve tree-structured computations that are difficult to parallelize efficiently. For these reasons, internal coordinates have been much less widely used than Cartesian coordinates for molecular simulations. They have the interesting property that their computational cost scales with the number of *free* degrees of freedom, in contrast to most other constraint algorithms whose cost scales with the number of *constrained* degrees of freedom. This makes internal coordinates most appropriate for highly constrained systems, such as when entire secondary structure elements or even protein domains are held rigid.

Many other constraint algorithms have been proposed, and a complete survey of them is beyond the scope of this paper. The methods described above include the most popular ones and are illustrative of the general approaches taken by many algorithms. Below, we expand on the details of our proposed approach, the constant constraint matrix approximation.

Constant Constraint Matrix Approximation

The CCMA algorithm is based on the observation that the Jacobian matrix changes very little over the course of a simulation. All elements along the diagonal are equal to 1. Each off-diagonal element describes the coupling between two constraints. If the two constraints do not share an atom, the corresponding element is zero. If they do share an atom, it is equal to

$$\mathbf{J}_{ij} = \frac{1/m_1}{1/m_1 + 1/m_2}\cos\theta \tag{4}$$

where m_1 is the mass of the atom that is shared by the two constraints, m_2 is the mass of the other atom affected by constraint i, and θ is the angle between the two constraints. The atomic masses usually do not change with time. If the angle θ is itself constrained, the corresponding element of **J** is constant over the simulation. In fact, if all bond lengths and angles are constrained, then **J** is constant.

If the angle is not constrained, the element will vary with time, but usually not very much. Molecular force fields typically include a harmonic force term for each angle that restricts its motion to a narrow range. This suggests that if we construct and invert **J** once at the start of the simulation, we can reuse it for every time step and it will continue to be a good approximation. (We note in passing that this same observation was made by Weinbach and Elber, but they did not pursue it further or develop an algorithm based on it.[9])

Specifically, we construct and invert a matrix **K** that is an approximation to **J** as follows. For each element in **K**:

(1) If the angle between the two constraints is itself constrained, we calculate the value on the basis of the actual constrained angle.

(2) Otherwise, we calculate it on the basis of the equilibrium angle of the corresponding harmonic force term.

How much **K** deviates from **J** is determined by how far the unconstrained angles vary from their equilibrium values. For typical molecular force fields, these deviations are very small. In the more general case of arbitrary constrained systems, however, there might be situations where angles have more flexibility. For example, some coarse-grained lipid models use a relatively soft force term on angles that allows larger fluctuations.[10] CCMA will still work with these systems, but the number of required iterations is expected to increase as the difference between **J** and **K** increases.

When solving the constraint equations for each time step, we replace $\mathbf{J}^{-1}$ in eq 2 by $\mathbf{K}^{-1}$. This involves a matrix-vector multiplication at each iteration, which will be efficient if and only if $\mathbf{K}^{-1}$ is sufficiently sparse. **K** is very sparse, since a single atom is almost never bonded to more than four other atoms, but it does not automatically follow that $\mathbf{K}^{-1}$ is also sparse. In practice, we find that most of its elements are extremely small and can be neglected. We therefore set all elements of $\mathbf{K}^{-1}$ that fall below a cutoff to zero, yielding a sparse matrix which still is an excellent approximation to $\mathbf{J}^{-1}$.

For highly constrained systems, such as when all bond lengths and angles are constrained, care must be taken to prevent **K** from becoming singular. This happens when a rigid cluster of atoms contains more constraints than are necessary to remove all internal degrees of freedom of the cluster. For example, a methane molecule has nine internal degrees of freedom, but if one naively constrains all bond lengths and angles, this produces ten constraints. Ideally, one should identify such clusters and remove the redundant constraints. Alternatively, one can invert **K** with a method that is robust to singular matrices, such as QR decomposition or singular value decomposition.[11] This approach assumes that the redundant constraints are all consistent with each other; if the constraints are inconsistent, it is impossible to find a solution which satisfies all of them.

Table 1. Average Number of Iterations Needed for the Constraint Algorithm To Converge with a Relative Tolerance of 10^{-4}

	1 fs	2 fs	3 fs	4 fs
CCMA (0.01 cutoff)	3.09	4.02	4.64	5.03
CCMA (0.1 cutoff)	3.58	4.50	4.79	5.38
SHAKE	20.8	27.7	31.9	34.7

Results

To test the CCMA algorithm, we incorporated it into OpenMM, a library for performing molecular simulations on graphics processing units (GPUs) and other high-performance architectures.[12] The implementation was straightforward since all elements of the algorithm (computing the vector of constraint errors, the sparse matrix−vector multiply, and updating atom positions) are easily parallelized. We also created a serial implementation to facilitate comparison with other algorithms.

We tested it by simulating the D14A variant of the lambda repressor monomer,[13,14] an 80 residue protein, in implicit solvent (Onufriev−Bashford−Case generalized Born model[15]). All bond lengths were constrained, as well as angles of the form H−X−H or H−O−X. This gives a total of 1570 constraints, none of which are redundant. Keeping all elements of $\mathbf{K}^{-1}$ whose absolute value is greater than 0.1 gives 8.1 nonzero elements per constraint, making the matrix−vector multiplies extremely fast. If we instead keep all elements greater than 0.01, there are 19.9 nonzero elements per constraint. The maximum number of nonzero elements in any row of $\mathbf{K}^{-1}$ (that is, the maximum number of other constraints that any constraint is directly affected by) is 22 with a cutoff of 0.1, or 47 with a cutoff of 0.01.

Simulations were run using time steps of 1−4 fs with both CCMA and SHAKE. Iteration was continued until all constraints were satisfied to within a relative tolerance of 10^{-4}. All simulations used a Langevin integrator to couple the protein to a thermal bath at 300 K with a friction coefficient of 91 ps^{-1}.

The results are shown in Table 1. CCMA requires only a small fraction as many iterations as SHAKE. More computation is required for each iteration due to the matrix−vector multiply, but the total number of FLOPS is still much smaller. We profiled a single threaded CPU implementation of each algorithm to precisely measure the computational work required for each one. When using a 4 fs time step, 1.1% of the total CPU time is spent in the SHAKE algorithm, while CCMA with a cutoff of 0.1 takes up 0.8% of the total CPU time.

More importantly, CCMA is easily parallelized. This makes it a far more efficient algorithm than SHAKE on modern parallel architectures. Massively parallel processors such as GPUs typically have hundreds or even thousands of processing units, and all other parts of the simulation can be efficiently implemented on them.[12] SHAKE, being a single threaded algorithm, would then become more expensive than all other parts of the simulation put together. In contrast, CCMA can be efficiently parallelized and remains a small contributor to the computation time on a GPU.

The rate of convergence is only weakly affected by how many elements of $\mathbf{K}^{-1}$ we keep. Decreasing the cutoff from 0.1 to 0.01 decreases the average required iterations by 3−16%, but it also more than doubles the number of elements (and hence the cost of the matrix−vector multiply). Cutoffs much larger than 0.1, on the other hand, do significantly impact the convergence. The optimal value for this cutoff will depend on the detailed performance of a particular implementation. On a cluster, for example, there is a communication overhead for every iteration, so it is probably best to use a small value; on a multicore shared memory computer, a larger value that minimizes the total amount of computation will likely be faster.

We also studied the effect of constraint topology on the rate of convergence. We repeated the above simulations using a time step of 2 fs, but constraining only bond lengths, not any angles. In that case, the average number of iterations for SHAKE drops by a factor of 3 to 9.67, while CCMA drops to 2.56 with a cutoff of 0.01, or to 3.52 with a cutoff of 0.1. We see that CCMA is less sensitive than SHAKE to the constraint topology. This is not surprising, since the accuracy of its approximation to the Jacobian does not change significantly, whereas SHAKE uses a much less accurate approximation when constraints are highly coupled.

Conclusions

We have developed a new constraint algorithm for use in molecular simulations. It produces very rapid convergence and has a lower overall computational cost than many popular algorithms. It can be used for arbitrary constraint topologies and works well for constraining angles as well as bond lengths. It also is easy to parallelize, making it a good choice for use on modern parallel architectures.

Availability

The implementation reported in this paper will be made available at Simtk.org as part of the OpenMM API (http://simtk.org/home/openmm). OpenMM is designed for incorporation into molecular dynamics codes to enable execution on GPUs and other high-performance architectures.

Acknowledgment. The authors would like to thank Michael Sherman for useful conversations related to the design of this algorithm. This work was supported by Simbios via the NIH Roadmap for Medical Research Grant U54 GM072970.

References

(1) Feenstra, K. A.; Hess, B.; Berendsen, H. J. C. Improving Efficiency of Large Time-Scale Molecular Dynamics Simulations of Hydrogen-Rich Systems. *J. Comput. Chem.* **1999**, *20*, 786–798.

(2) Tironi, I. G.; Brunne, R. M.; van Gunsteren, W. F. On the Relative Merits of Flexible Versus Rigid Models for Use in Computer Simulations of Molecular Liquids. *Chem. Phys. Let.* **1996**, *250*, 19–24.

(3) Kräutler, V.; van Gunsteren, W. F.; Hünenberger, P. H. A Fast SHAKE Algorithm to Solve Distance Constraint Equations for Small Molecules in Molecular Dynamics Simulations. *J. Comput. Chem.* **2001**, *22*, 501–508.

(4) Hess, B.; Bekker, H.; Berendsen, H. J. C.; Fraaije, J. G. E. M. LINCS: A Linear Constraint Solver for Molecular Simulations. *J. Comput. Chem.* **1997**, *18*, 1463–1472.

(5) Ryckaert, J.-P.; Ciccotti, G.; Berendsen, H. J. C. Numerical Integration of the Cartesian Equations of Motion of a System with Constraints: Molecular Dynamics of n-Alkanes. *J. Comp. Phys.* **1977**, *23*, 327–341.

(6) Miyamoto, S.; Kollman, P. A. SETTLE: An Analytical Version of the SHAKE and RATTLE Algorithm for Rigid Water Models. *J. Comput. Chem.* **1992**, *13*, 952–962.

(7) Vaidehi, N.; Jain, A.; Goddard, W. A., III. Constant Temperature Constrained Molecular Dynamics: The Newton−Euler Inverse Mass Operator Method. *J. Phys. Chem.* **1996**, *100*, 10508–10517.

(8) Schwieters, C. D.; Clore, G. M. Internal Coordinates for Molecular Dynamics and Minimization in Structure Determination and Refinement. *J. Magn. Reson.* **2001**, *152*, 288–302.

(9) Weinbach, Y.; Elber, R. Revisiting and Parallelizing SHAKE. *J. Comp. Phys.* **2005**, *209*, 193–206.

(10) Marrink, S. J.; de Vries, A. H.; Mark, A. E. Coarse Grained Model for Semiquantitative Lipid Simulations. *J. Phys. Chem. B* **2004**, *108*, 750–760.

(11) Press, W. H.; Teukolsky, S. A.; Vetterling, W. T.; Flannery, B. P., *Numerical Recipes in C++*, 2nd ed.; Cambridge University Press: Cambridge, U.K., 2003.

(12) Friedrichs, M. S.; Eastman, P.; Vaidyanathan, V.; Houston, M.; LeGrand, S.; Beberg, A. L.; Ensign, D. L.; Bruns, C. M.; Pande, V. S. Accelerating Molecular Dynamic Simulation on Graphics Processing Units. *J. Comput. Chem.* **2009**, *30*, 864–872.

(13) Yang, W. Y.; Gruebele, M. Rate−Temperature Relationships in λ-Repressor Fragment λ_{6-85} Folding. *Biochemistry* **2004**, *43*, 13018–13025.

(14) Yang, W. Y.; Gruebele, M. Folding λ-Repressor at Its Speed Limit. *Biophys. J.* **2004**, *87*, 596–608.

(15) Onufriev, A.; Bashford, D.; Case, D. A. Exploring Protein Native States and Large-Scale Conformational Changes with a Modified Generalized Born Model. *Proteins* **2004**, *55*, 383–394.

CT900463W