Proteins Wriggle

Michael Cahill,* Sean Cahill,† and Kevin Cahill‡
*School of Medicine, Uniformed Services University, Bethesda, Maryland 20814, †Department of Computer Science, and ‡Department of Physics and Astronomy, University of New Mexico, Albuquerque, New Mexico 87131-1156

ABSTRACT We propose an algorithmic strategy for improving the efficiency of Monte Carlo searches for the low-energy states of proteins. Our strategy is motivated by a model of how proteins alter their shapes. In our model, when proteins fold under physiological conditions, their backbone dihedral angles change synchronously in groups of four or more to avoid steric clashes and respect the kinematic conservation laws. They wriggle; they do not thrash. We describe a simple algorithm that can be used to incorporate wriggling in Monte Carlo simulations of protein folding. We have tested this wriggling algorithm against a code in which the dihedral angles are varied independently (thrashing). Our standard of success is the average root-mean-square distance (rmsd) between the α-carbons of the folding protein and those of its native structure. After 100,000 Monte Carlo sweeps, the relative decrease in the mean rmsd, as one switches from thrashing to wriggling, rises from 11% for the protein 3LZM with 164 amino acids (aa) to 40% for the protein 1A1S with 313 aa and 47% for the protein 16PK with 415 aa. These results suggest that wriggling is useful and that its utility increases with the size of the protein. One may implement wriggling on a parallel computer or a computer farm.

WHY PROTEINS WRIGGLE

We propose an algorithmic strategy for improving the efficiency of Monte Carlo searches for the low-energy states of proteins. Our strategy is motivated by a model in which proteins alter their shapes by means of local motions that minimize the displacement of distant atoms.

Folding proteins avoid steric clashes and respect the kinematic conservation laws. The system consisting of a protein and the nearby solvent molecules approximately conserves its energy, momentum, and angular momentum. The shape of a protein is mainly defined by the angles of rotation, ϕ_i and ψ_i, about the backbone bonds that link the α-carbons to the adjacent amide planes. A change in one of these dihedral angles would rotate a significant part of the protein molecule, moving each atom by a length proportional to its distance from the axis of rotation. In a protein consisting of hundreds or thousands of amino acids in water, such a rotation would engender steric clashes and grossly violate the kinematic conservation laws. Instead, when a protein folds or unfolds in our model, its backbone dihedral angles conspire in groups of four or more to change in ways that limit their displacement of distant atoms. Proteins wriggle; they do not thrash.

Localized motions of the protein backbone involve at least four bonds, but simpler local motions are possible in simpler systems. In lattice models, all local motions, such as corner moves and crankshaft moves (Schatzki 1965), are localized. Polymers also possess simple local motions in the continuum. In polyethylene, for instance, two backbone bonds separated by a *trans* bond are parallel, and so equal and opposite rotations about these parallel bonds constitute a motion that is localized. Such crankshaft moves in polymers have been seen in simulations guided by brownian and molecular dynamics (Helfand, 1971, 1984; Skolnick and Helfand, 1980; Helfand et al., 1980, 1981a,b; Weber et al., 1983).

Continuum Monte Carlo searches strike a balance between the temporal detail of molecular dynamics and the rigidity of the lattice. They are defined by their "kinematics" (how the proteins move) and their "dynamics" (why they move). By kinematics, we mean the variables that define the state of the protein and the kinds of Monte Carlo moves that are permitted. The dynamics is determined by the energy function. Monte Carlo simulations do not incorporate wriggling in their kinematics; this paper is about how they could and whether they should.

Our wriggling algorithm is based upon the linear dependence of every quartet of three-dimensional (3D) vectors. In the next section, we use this linear dependence to show that one may choose the four angles of rotation about any four backbone bonds so that the combined motion of the protein is localized. A simple computer algorithm that may be used to incorporate wriggling in Monte Carlo searches is outlined in the section, A Wriggling Algorithm.

We have run three simple tests to determine whether wriggling actually improves the efficiency of a Monte Carlo search; we describe these tests and their results in the section, Does Wriggling Work. In each test, we compared simulations guided by the wriggling algorithm to ones guided by a standard thrashing algorithm in which the dihedral angles are varied independently. To separate the kinematics of folding from the dynamics of folding, we used a nearly perfect but artificial energy function that is proportional to the root-mean-square distance (rmsd) of the folding α-carbons from the α-carbons of the native structure. Because there is no simple relation between this rmsd and an energy, and because the use of wriggling alters the effective temperature, we performed our Monte Carlo runs at absolute zero. On each protein, we performed

Submitted August 21, 2001, and accepted for publication January 2, 2002.
Address reprint requests to Kevin Cahill, Dept. of Physics and Astronomy, Univ. of New Mexico, Albuquerque, NM 87131. Tel.: 505-277-5318; Fax: 505-277-1520; E-mail: cahill@unm.edu.

0006-3495/02/05/2665/06 $2.00

two sets of eight or more runs of 100,000 sweeps, one set controlled by the wriggling algorithm and the other by a thrashing algorithm. The runs started from fully denatured random coils. We averaged the final rmsd. The relative decrease in the average final rmsd as one switches from the thrashing code to the wriggling code $(\langle\text{rmsd}\rangle_{\text{th}} - \langle\text{rmsd}\rangle_{\text{wr}})/\langle\text{rmsd}\rangle_{\text{wr}}$ is a measure of the utility of wriggling. This wriggling advantage rose from 11% for the protein 3LZM with 164 amino acids (aa), to 40% for the protein 1A1S with 313 aa, and to 47% for the protein 16PK with 415 aa. The advantage of wriggling seems to grow with the length of the protein.

In the section, Wriggling on a Parallel Computer, we sketch how one might implement wriggling with a realistic energy function on a parallel computer or on a farm of computers. We summarize the present work and mention some of its limitations in the Conclusions and Caveats section.

HOW PROTEINS WRIGGLE

Three-dimensional space is spanned by any three linearly independent vectors. Every quartet of 3D vectors is linearly dependent—that is, for every four 3D vectors $\vec{v}_1$, $\vec{v}_2$, $\vec{v}_3$, $\vec{v}_4$, there exist four numbers x_1, x_2, x_3, x_4, such that the weighted sum of the vectors vanishes,

$$\sum_{\text{i}=1}^{4} x_{\text{i}}\vec{v}_{\text{i}} = 0. \tag{1}$$

In this section, we use these mathematical facts to show that one always may choose the angles of rotation about any four bonds to minimize the net effect of the four rotations upon distant atoms.

The change $\vec{dr}$ in the position $\vec{r}$ of an atom due to a rotation by a small angle ϵ about a bond axis taken to be a unit vector $\hat{b}$ is the cross-product of $\epsilon\hat{b}$ with the vector to the point $\vec{r}$ from any point $\vec{c}$ on the axis,

$$\vec{dr} = \epsilon\hat{b} \times (\vec{r} - \vec{c}). \tag{2}$$

To first order in ϵ, the net displacement $\vec{dr}$ of the position $\vec{r}$ of an atom due to four rotations by the small angles ϵ_{i} about the bonds $\hat{b}_{\text{i}}$ for $i = 1, 2, 3, 4$, is the sum

$$\vec{dr} = \sum_{\text{i}=1}^{4} \vec{dr}_{\text{i}} = \sum_{\text{i}=1}^{4} (\epsilon_{\text{i}}\hat{b}_{\text{i}} \times (\vec{r} - \vec{c}_{\text{i}})). \tag{3}$$

If we use $\vec{a}$ for the average of the four points $\vec{c}_{\text{i}}$, which typically would be the midpoint between two α-carbons, then we may express the net displacement $\vec{dr}$ as

$$\vec{dr} = \sum_{\text{i}=1}^{4} (\epsilon_{\text{i}}\hat{b}_{\text{i}} \times (\vec{r} - \vec{a} + \vec{a} - \vec{c}_{\text{i}}))$$

$$= \left(\sum_{\text{i}=1}^{4} \epsilon_{\text{i}}\hat{b}_{\text{i}}\right) \times (\vec{r} - \vec{a}) + \sum_{\text{i}=1}^{4} (\epsilon_{\text{i}}\hat{b}_{\text{i}} \times (\vec{a} - \vec{c}_{\text{i}})). \tag{4}$$

The displacement $\vec{dr}$ will be independent of the potentially long vector $\vec{r} - \vec{a}$ (for every atom) if the sum of the bond vectors $\hat{b}_{\text{i}}$ weighted by their angles ϵ_{i} vanishes. That is, the net displacement $\vec{dr}$ is merely

$$\vec{dr} = \sum_{\text{i}=1}^{4} (\epsilon_{\text{i}}\hat{b}_{\text{i}} \times (\vec{a} - \vec{c}_{\text{i}})), \tag{5}$$

which is independent of $\vec{r} - \vec{a}$, if the angle-weighted sum of bond vectors vanishes,

$$\sum_{\text{i}=1}^{4} \epsilon_{\text{i}}\hat{b}_{\text{i}} = 0. \tag{6}$$

And, because every quartet of 3D vectors is linearly dependent (Eq. 1), it is always possible to choose the four small angles ϵ_{i} so that this sum vanishes for any four bond vectors $\hat{b}_{\text{i}}$.

A WRIGGLING ALGORITHM

In this section, we show how to transform the wriggling condition (Eq. 6) into a matrix equation that can be solved by standard linear-algebra software, such as the freely available LAPACK (Anderson et al., 1999a) subroutine DGESV (Anderson et al., 1999b).

The wriggling condition (Eq. 6) can be written in the more explicit form

$$\sum_{\text{n}=1}^{3} \hat{b}_{\text{in}}(-\epsilon_{\text{n}}/\epsilon_4) = \hat{b}_{\text{i4}} \tag{7}$$

for $i = 1, 2, 3$. Let us arrange the first three axes $\hat{b}_1$, $\hat{b}_2$, and $\hat{b}_3$ into the matrix A with elements $A_{\text{in}} = \hat{b}_{\text{in}}$ for $i, n = 1, 2, 3$ and rename the fourth axis $\hat{b}_4$ as the vector B with components $B_{\text{i}} = \hat{b}_{\text{i4}}$. If we now use X for the vector with components $X_{\text{n}} = -\epsilon_{\text{n}}/\epsilon_4$ for $n = 1, 2, 3$, then the wriggling condition (Eq. 7) becomes

$$\sum_{\text{n}=1}^{3} A_{\text{in}}X_{\text{n}} = B_{\text{i}} \quad \text{for} \quad i = 1, 2, 3. \tag{8}$$

The LAPACK subroutine DGESV is designed to solve such linear equations. The call

$$\text{call DGESV(3, 1, A, 3, ipiv, B, 3, info)} \tag{9}$$

returns the three angle ratios $X_{\text{n}} = -\epsilon_{\text{n}}/\epsilon_4$ for $n = 1, 2, 3$ as the three components B_{n} of the vector B. The value info = 0 indicates that the computation is successful, and ipiv contains pivot indices, which may be ignored. We set $\epsilon_4 = -1$ so that $\epsilon_{\text{n}} = -\epsilon_4 B_{\text{n}} = B_{\text{n}}$ for $n = 1, 2, 3$ and then normalize the four angles,

$$\sum_{\text{n}=1}^{4} \epsilon_{\text{n}}^2 = 1. \tag{10}$$

Last, we multiply them by a random number x drawn uniformly from the interval $(-0.0125, 0.0125)$ radians, so that our final angles are $\theta_n = x\epsilon_n$ for $n = 1, 2, 3, 4$.

Although we used the linear equation (Eq. 7) as our wriggling condition, we used the exact and general form (Eq. A8) of the rotation matrix described in the Appendix to implement all rotations.

DOES WRIGGLING WORK?

To test the utility of our wriggling algorithm, we performed Monte Carlo simulations of protein folding on three proteins: lysozyme (3LZM.pdb, 164 aa), ornithine carbamoyltransferase (1A1S.pdb, 313 aa), and phosphoglycerate kinase (16PK.pdb, 415 aa). We used an artificially nearly perfect energy function that is proportional to the rmsd between the α-carbons of the folding protein and those of its native structure. This nearly perfect energy function allowed us to separate the kinematics of folding (the Monte Carlo moves—wriggling or thrashing) from the dynamics of folding (the mechanisms in the energy function—conformational entropy, charge–charge interactions, hydrogen bonds, van der Waals interactions, hydrophobicity (Dill 1990; Chan and Dill 1990)). Because there is no simple relationship between the α-carbon rmsd and an energy, and because wriggling changes the effective temperature, we conducted our simulations at absolute zero rather than at physiological temperatures or at that of liquid nitrogen.

Each of our tests consisted of 8 or 10 pairs of Monte Carlo runs, one guided by the wriggling algorithm and the other by a thrashing algorithm. Each run began with a random coil and ran for 100,000 sweeps, each sweep being a sequence of applications of the algorithm successively along the primary structure of the protein. The wriggling code applies the algorithm described in Eqs. 7–10 to successive quartets of dihedral angles. After each wriggle, the code performs a Metropolis step; because the temperature is zero, the wriggle is accepted if and only if it lowers the rmsd. In each sweep, the wriggling code wriggles first the four dihedral angles ϕ_2, ψ_2, ϕ_3, ψ_3 of residues 2 and 3; second, the angles ψ_2, ϕ_3, ψ_3, ϕ_4 of residues 2, 3, and 4; third, the angles ϕ_3, ψ_3, ϕ_4, ψ_4 of residues 3 and 4, and continues in this way down the chain to the penultimate residue. The code does not vary the ϕ angle of any proline residue. To keep these angles fixed, the code performs one of several procedures when one or more proline residues is involved in a wriggle. In each sweep, the thrashing code successively and independently changes, by a random angle $\delta\theta$ drawn uniformly from the interval $(-0.0125, 0.0125)$ radians, every dihedral angle from the first ψ to the last ϕ, except for the ϕ of the prolines; it accepts each change if and only if the change lowers the rmsd. Apart from end effects, the thrashing code makes the same number of Monte Carlo judgments per sweep as does the wriggling code.

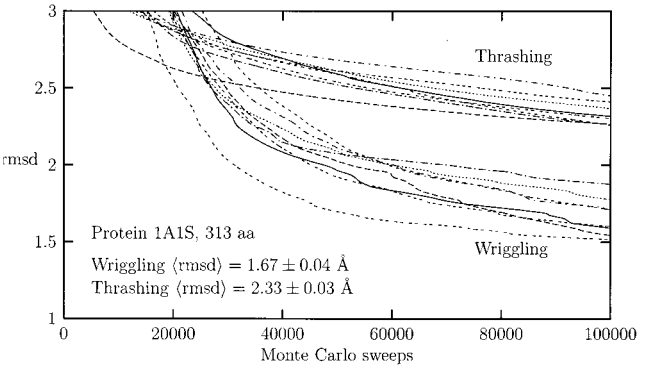

FIGURE 1 For the protein 1A1S, the lines trace the values of the rmsd for eight runs guided by the wriggling algorithm and eight guided by the thrashing algorithm.

In our first test of wriggling, we constructed eight fully denatured random coils of the protein 3LZM, which has 164 residues. The rmsd of these denatured configurations ranged from 18.2 to 128.5 Å. In eight runs of 100,000 sweeps, the average rmsd was 1.46 ± 0.07 Å for the wriggling code and 1.62 ± 0.05 Å for the thrashing code. The mean thrashing rmsd was 11% larger than the mean wriggling rmsd.

We performed our second test of wriggling on the protein 1A1S, which has 313 residues. Our eight denatured coils of 1A1S had rmsd running from 25.9 to 251.5 Å. The rmsd of the 16 wriggling and thrashing runs are plotted in Fig. 1. After ~40,000 sweeps, the thrashing runs separate out into a cluster of lines, labeled as thrashing, that lie distinctly above the wriggling runs, labeled as wriggling. The eight wriggling runs had an average final rmsd of 1.67 ± 0.04 Å, whereas that of the eight thrashing runs was 2.33 ± 0.03 Å or 40% greater.

For our third test of wriggling, we first randomized and stretched the native structure of the protein 16PK, which has 415 (visible) residues, into 10 fully denatured coils with rmsd running from 24.7 to 341.8 Å. We then allowed our wriggling and thrashing codes to reduce the rmsd of these 10 random coils in runs of 100,000 sweeps. The rmsd of the 20 runs are plotted in Fig. 2 as a function of sweep number. Apart from one thrashing run, the rmsd of the wriggling runs drop below those of the thrashing runs after ~30,000 sweeps. The outlying thrashing run, labeled by the letter "t," did slightly better than the two worst wriggling runs. After 100,000 sweeps, the average rmsd of the wriggling runs was 1.66 ± 0.03 Å, whereas that of the thrashing runs was 2.44 ± 0.08 Å. The mean rmsd of the thrashing code was 47% greater than that of the wriggling code.

One estimate of the utility of wriggling is the relative decrease in the average final rmsd as one switches from the thrashing code to the wriggling code,

$$\frac{\langle \mathrm{rmsd} \rangle_{\mathrm{th}} - \langle \mathrm{rmsd} \rangle_{\mathrm{wr}}}{\langle \mathrm{rmsd} \rangle_{\mathrm{wr}}}. \qquad (11)$$

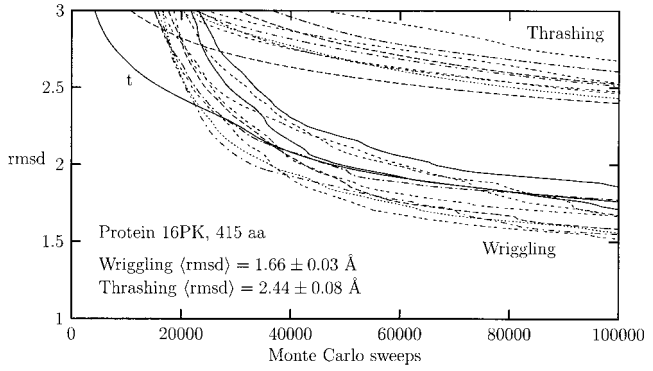

FIGURE 2 For the protein 16PK, the lines trace the values of the rmsd for 10 runs guided by the wriggling algorithm and 10 guided by the thrashing algorithm. The plot labeled by the letter "t" is a successful thrashing outlier.

In these three tests, the utility of wriggling increased with the size of the protein, rising from an 11% advantage at 164 aa to 40% at 313 aa and 47% at 415 aa. The longer the protein, the larger are the motions that occur when the backbone-bond angles are varied one at a time, and so the greater are the need for and the advantage of wriggling.

Each of the proteins 1A1S and 16PK has two domains. Is the advantage of wriggling over thrashing in these two cases due merely to a better twist in the polypeptide strand that connects the two domains? To answer this question, we measured the rmsd of the individual domains of the final configurations of 1A1S and 16PK given by the wriggling and thrashing codes. The average rmsd of the first and second domains of 1A1S, respectively, were 1.78 and 1.56 Å with wriggling, and 2.05 and 2.57 Å with thrashing. Those of 16PK were 1.52 and 1.79 Å with wriggling, and 2.34 and 2.55 Å with thrashing. These results suggest that wriggling gives better domains, not just better connecting strands.

The wriggling code differs from the thrashing code in two respects: its basic moves are four rotations rather than a single rotation and it suppresses large motions of remote atoms. To evaluate the two effects separately, we wrote a code in which the elemental moves are groups of four rotations but in which no wriggling condition is imposed. We let this coordinated-thrashing code fold our 10 denatured starting configurations of the protein 16PK and found, after 100,000 sweeps, that the average rmsd was 1.88 $\pm$ 0.04 Å, which is to be compared with 1.66 $\pm$ 0.03 Å for the wriggling code and 2.44 $\pm$ 0.08 Å for the thrashing code. So wriggling is better than coordinated thrashing and much better than thrashing, but part of the success of wriggling arises from the coordination of its compound elemental moves.

Because of our use of the rmsd as an artificially nearly perfect energy function, the proteins of our simulations are phantoms; they can move through each other. A real but approximate energy function would reject all moves into excluded volume; it therefore would reject many thrashing moves because of their large-scale motions. The use of the rmsd in our three tests deprives wriggling of one of its key advantages over thrashing and over coordinated thrashing, namely that its localized motions are less likely to involve steric clashes. Thus, the utility of wriggling in simulations with real energy functions may be greater than is indicated by these tests.

The wriggling code runs somewhat more slowly than the thrashing code. But a realistic energy function would slow down both codes by so much that the speed advantage of thrashing would be negligible.

In the first 10,000 sweeps of our tests of the wriggling algorithm, the thrashing code reduced its rmsd more quickly than the wriggling code. It might therefore be worthwhile to experiment with codes that relax the wriggling condition for the first 10,000 sweeps or that mix coordinated thrashing with wriggling.

The action of a wriggle is much less than that of a thrash; the action of a small-angle wriggle may be as little as 10 $\hbar$. So quantum-mechanical effects are more important with wriggling than with thrashing, but even so, they probably would be obscured by decoherence (Habib et al., 1998).

WRIGGLING ON A PARALLEL COMPUTER

Most of the residues of a folded protein lie in alpha helices and beta sheets. The secondary structure of a protein is the assignment of residues to helices, sheets, turns, and coils. One may list the possible secondary structures of a protein and assign one secondary structure to each processor of a parallel computer or computer farm. Each processor would perform Monte Carlo moves on the dihedral angles of the residues in the turns and coils of its secondary structure, but would leave invariant the dihedral angles of its helices and sheets. Wriggling should be used in the coils and in turns longer than 4 or 5 aa, but probably not in turns of 3 or 4 aa, where it might overly constrain the folding of the protein. Because each processor would vary the dihedral angles (and possibly the principal side-chain angles) only of the residues in the coils and turns, the simulation would run quickly enough to be guided by a realistic energy function (Weiner and Kollman 1981; Brooks et al., 1983; Lazarides and Karplus 1998, 1999) with solvation and excluded volume. At the end of a run of perhaps 100,000 sweeps, the final energies of the different secondary structures would be compared and their folds stored. Many runs would be required to test all the plausible secondary structures. This implementation of wriggling would make optimum use of a parallel computer or of a computer farm; no time would be lost to interprocessor communication or to waiting.

CONCLUSIONS AND CAVEATS

We have described and tested an algorithmic strategy for improving the efficiency of Monte Carlo searches for the

low-energy states of proteins. Our strategy is motivated by a model in which the laws of physics constrain the incremental motions of proteins to be essentially local. Localized motions can be incorporated in Monte Carlo searches by a simple algorithm that rotates the dihedral angles in groups of four. To test this wriggling algorithm, we performed 52 zero-temperature, 100,000-sweep, Monte Carlo searches for the low-energy states of the proteins 3LZM, 1A1S, and 16PK using the rmsd as an artificially nearly perfect energy function. The searches guided by the wriggling algorithm reached lower rmsd than those guided by the usual thrashing algorithm by a margin that increased with the length of the protein. But it remains to be seen whether and how this wriggling algorithm might improve the efficiency of Monte Carlo searches performed at finite temperature and guided by an approximate, realistic energy function with solvation and excluded volume.

APPENDIX: ROTATION MATRICES

For the sake of completeness, we derive in this appendix an exact formula for the general rotation matrix from the expression (Eq. 2) for an infinitesimal rotation. To simplify the notation, we shall consider an axis that runs through the origin, and choose $\vec{c} = 0$. In this case by Eq. 2, a right-handed rotation about a bond $\hat{b}$ by an infinitesimal angle ϵ changes a vector $\vec{r}$ by the small amount $\vec{dr} = \epsilon\hat{b} \times \vec{r}$ where the cross-product $\hat{b} \times \vec{r}$ has the components

$$(\hat{b} \times \vec{r})_i = \sum_{j=1}^{3} \sum_{k=1}^{3} \varepsilon_{ikj} \hat{b}_k r_j \qquad (A1)$$

in which the totally antisymmetric tensor ε_{ikj} has elements $\varepsilon_{123} = \varepsilon_{231} = \varepsilon_{312} = 1$, $\varepsilon_{213} = \varepsilon_{132} = \varepsilon_{321} = -1$, with all other elements zero, e.g., $\varepsilon_{113} = 0$, etc. If we use the definition $(L_k)_{ij} = \varepsilon_{ikj}$ of the rotation generators $\vec{L}$, then we may write the change dr_i as

$$dr_i = \epsilon \sum_{j=1}^{3} \sum_{k=1}^{3} \hat{b}_k (L_k)_{ij} r_j = \epsilon \sum_{j=1}^{3} (\hat{b} \times \vec{L})_{ij} r_j, \qquad (A2)$$

or, in matrix notation as

$$dr = \epsilon \hat{b} \times \vec{L} r. \qquad (A3)$$

Let us use $\vec{r}(\theta)$ for the vector $\vec{r}$ after a right-handed rotation by the angle θ about the axis $\hat{b}$. Then, by Eq. A3, the vector $\vec{r}(\theta)$ satisfies the differential equation

$$\frac{\mathrm{d}r(\theta)}{\mathrm{d}\theta} = \hat{b} \times \vec{L} r(\theta). \qquad (A4)$$

The solution that satisfies the boundary condition $\vec{r}(0) = \vec{r}$ is

$$r(\theta) = \exp(\theta \hat{b} \times \vec{L}) r(0), \qquad (A5)$$

and so the matrix that represents a finite rotation by the angle θ about the axis $\hat{b}$ is

$$R(\theta \hat{b}) = \exp(\theta \hat{b} \times \vec{L}). \qquad (A6)$$

The exact form of Eq. 2 when the axis $\hat{b}$ does not go through the origin but through another point $\vec{c}$ is

$$r_i(\theta) - c_i = \sum_{j=1}^{3} R_{ij}(\theta \hat{b})(r_j - c_j)$$

$$= \sum_{j=1}^{3} [\exp(\theta \hat{b} \times \vec{L})]_{ij}(r_j - c_j). \qquad (A7)$$

In our codes, we used this formula with the matrix R given by (Cahill et al., 2000)

$$R_{ij}(\theta \hat{b}) = \cos\theta \delta_{ij} - \sin\theta \left(\sum_{k=1}^{3} \epsilon_{ijk} \hat{b}_k \right) + (1 - \cos\theta) \hat{b}_i \hat{b}_j, \qquad (A8)$$

which is convenient for computation.

We have benefited from conversations with Susan Atlas, David Baker, Ken Dill, Norman van Gulick, Gary Herling, and Charlie Strauss. One of us (S.C.) would like to thank Ken Dill for the hospitality extended to him at the University of California–San Francisco. Most of our computations were performed on the computers of the Albuquerque High-Performance Computing Center.

REFERENCES

Anderson, E., Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. 1999a. LAPACK Users' Guide. 3d ed. SIAM, Philadelphia, PA. Available on line at http://www.netlib.org/lapack/lug/lapack_lug.html.

Anderson, E., Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. 1999b. LAPACK Users' Guide. 3d ed. SIAM, Philadelphia, PA. 237–238. Available on line at http://www.netlib.org/lapack/.

Brooks, B. R., R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus. 1983. Charmm: a program for macromolecular energy, minimization, and dynamics calculations. *J. Comput. Chem.* 4:187–217.

Cahill, M., M. Fleharty, and K. Cahill. 2000. Simulations of protein folding. *Nucl. Phys. B (Proc. Suppl.).* 83:929–931.

Chan, H. S., and K. A. Dill. 1990. Origins of structure in globular proteins. *Proc. Natl. Acad. Sci. U.S.A.* 87:6388.

Dill, K. A. 1990. Dominant forces in protein folding. *Biochemistry.* 29: 7133–7155.

Habib, S., K. Shizume, and W. H. Zurek. 1998. Decoherence, chaos, and the correspondence principle. *Phys. Rev. Lett.* 80:4361–4365.

Helfand, E. 1971. Theory of the kinetics of conformational transitions in polymers. *J. Chem. Phys.* 54:4651–4661.

Helfand, E. 1984. Dynamics of conformational transitions in polymers. *Science.* 226:647–650.

Helfand, E., Z. R. Wasserman, and T. A. Weber. 1980. Brownian-dynamics study of polymer conformational transitions. *Macromolecules.* 13:526–533.

Helfand, E., Z. R. Wasserman, and T. A. Weber. 1981a. Kinetics of conformational transitions in bulk polymers. *Polymer Preprints (American Chemical Society, Division of Polymer Chemistry).* 22: 279–280.

Helfand, E., Z. R. Wasserman, T. A. Weber, J. Skolnick, and R. J. H. 1981b. The kinetics of conformational transitions: effect of variation of bond angle bending and bond stretching force constants. *J. Chem. Phys.* 75:4441–4445.

Lazarides, T., and M. Karplus. 1998. Discrimination of the native from misfolded protein models with an energy function including implicit solvation. *J. Mol. Biol.* 288:477–487.

Lazarides, T., and M. Karplus. 1999. Effective energy function for proteins in solution. *Proteins Struct. Funct. Genet.* 35:133–152.

Schatzki, T. F. 1965. Molecular interpretation of the γ-transition in polyethylene and related compounds. *Polymer Preprints (American Chemical Society, Division of Polymer Chemistry).* 6:646–651.

Skolnick, J., and E. Helfand. 1980. Theory of the kinetics of conformational transitions in polymers. *J. Chem. Phys.* 72:5489–5500.

Weber, T. A., E. Helfand, and Z. R. Wasserman. 1983. Simulation of polyethylene, molecular-based study of fluids 20. *In* ACS Advances in Chemistry Series. No. 204. American Chemical Society, Washington, DC. 487–500.

Weiner, P. K., and P. A. Kollman. 1981. Amber: assisted model building with energy refinement. A general program for modeling molecules and their interactions. *J. Comput. Chem.* 2:287.