A Rapid Finite Difference Algorithm, Utilizing Successive Over-Relaxation to Solve the Poisson–Boltzmann Equation

Anthony Nicholls* and Barry Honig

Department of Biochemistry and Molecular Biophysics, Columbia University, New York, New York, 10032

Received 28 May 1990; accepted 7 November 1990

An efficient algorithm is presented for the numerical solution of the Poisson–Boltzmann equation by the finite difference method of successive over-relaxation. Improvements include the rapid estimation of the optimum relaxation parameter, reduction in number of operations per iteration, and vector-oriented array mapping. The algorithm has been incorporated into the electrostatic program DelPhi, reducing the required computing time by between one and two orders of magnitude. As a result the estimation of electrostatic effects such as solvent screening, ion distributions, and solvation energies of small solutes and biological macromolecules in solution, can be performed rapidly, and with minimal computing facilities.

INTRODUCTION

In recent years the linear Poisson–Boltzmann (PB) equation has been applied with some success to the problem of electrostatic interactions of molecules in solution.[1] The PB equation accounts for solvent screening, solvation energies, and salt effects, and through the use of local dielectric constants, can be applied, with some validity, at the microscopic level. However, the equation only admits analytic solution for a few idealized shapes, e.g., spheres and cylinders, so that the complex surfaces formed by most solutes, and biological molecules in particular, require a numerical solution. We are concerned in this article with the rapid achievement of this solution by use of numerical algorithms.

We shall review the finite difference method for solving the PB equation, and three algorithms for its implementation, namely the Jacobian, Gauss–Seidel (GS), and Successive Over-Relaxation (SOR) algorithms. The first and second of these are simple to program but slow to converge to the exact solution. SOR can be rapid if one has knowledge of a critical "relaxation" parameter, ω. We shall then report our advances in both the programming of these algorithms, and in estimating ω, the result of which is a remarkably efficient program, both in terms of required processing power and computer memory. Finally we shall consider other first algorithms, such as that recently proposed by Davis and McCammon.[2]

*To whom all correspondence should be addressed.

THE FINITE DIFFERENCE METHOD

The Poisson–Boltzmann equation is

$$\nabla \cdot [\varepsilon(\mathbf{x})\nabla\phi(\mathbf{x})] - \kappa(\mathbf{x})^2\sinh(\phi(\mathbf{x})) = -4\pi\rho(\mathbf{x}) \tag{1}$$

Here $\phi(\mathbf{x})$ is the electrostatic potential, and is determined by $\varepsilon(\mathbf{x})$, the spatial dielectric function, $\kappa(\mathbf{x})$, a modified Debye–Huckel parameter, $\rho(\mathbf{x})$, the charge distribution function.

This equation linearizes to

$$\nabla \cdot [\varepsilon(\mathbf{x})\nabla\phi(\mathbf{x})] - \kappa(\mathbf{x})^2\phi(\mathbf{x}) = -4\pi\rho(\mathbf{x}) \tag{2}$$

and this is the equation whose solution we shall be concerned with in what follows. Our results have implications for the solution of the full, nonlinear, PB, and we shall mention these at the appropriate juncture.

The spatially varying dielectric function is what distinguishes this approach from, say, Coulomb's law. It allows for an explicit description of the dielectric discontinuity that comes about because of the difference between the electronic polarizability of the macromolecule and that of the solvent. Including the electrostatic effects of salt, ubiquitous in real biological systems, requires the addition of the Debye–Huckel term. In the above equation we have the Debye–Huckel parameter, κ, (modified to be dielectric independent) which is equal to

$$(8\pi e^2 N_A I/1000kT)^{1/2}$$

(I = bulk salt concentration in moles per litre, N_A = Avagadro's number, k = Boltzmann's constant, T = temperature in Kelvin, e the charge on an elec-

CCC 0192-8651/91/040435-11$04.00

tron in SI units). Charges are assigned to atom centers by any of several semiempirical charge "sets."

The finite difference philosophy is to map all physical quantities onto a grid, replacing differential operators by grid value differences, hopefully finite, hence the name. To this end, charge is mapped to grid points composing the corners of the cube wherein the charge lies. There are many ways this mapping might be carried out, but usually a linear interpolation scheme is used to weight the eight corners. The dielectric constant is typically set to one value inside the molecule, for instance $\varepsilon = 2$, to simulate the high frequency electronic response, and to 80 outside to simulate water.[1,3] Inside and outside here is defined not by the van der Waals radii, but by solvent accessibility, i.e., any point which at some time lies within a probe sphere, of radius approximating water, rolled around the surface of the molecule, is considered "outside." As for κ, the above value only holds for grid points accessible to salt. This is determined by the van der Waals radii of the atoms which compose the macromolecule, plus a constant meant to represent the finite radius of the salt atoms (typically that of sodium, i.e., about 2 Ångstroms). Together these define an 'exclusion-zone' wherein $\kappa = 0$. The interested reader is referred to Klapper, et al.[3]

We shall consider only cubic lattices, with L points per side, and N $(=L{*}L{*}L)$ points in all, although there is no difficulty in generalizing to other shapes.

The final reduction to finite difference form gives,[3]

$$\phi_0 = \left[\frac{\left(\sum_{i=1}^{6} \varepsilon_i\phi_i\right) + 4\pi q_0/h}{\left(\sum_{i=1}^{6} \varepsilon_i\right) + (\kappa_0 h)^2}\right] \tag{3}$$

where ϕ_0 is the potential at a particular grid point, ϕ_i the potential at the six nearest neighbors, and ε_i the dielectric constant at the *midpoint* between ϕ_0 and ϕ_i, q_0, the charge assigned to the grid point, and κ_0 nonzero if the grid point is in salt. The parameter h is the grid spacing, i.e., the distance, in Ångstroms, between grid points.

We see that this form of the reduction could be written in matrix form as

$$\Phi = \mathbf{T}\Phi + \mathbf{Q} \tag{4}$$

where $\mathbf{T}$ is a matrix, Φ and $\mathbf{Q}$ are vectors. One perceives that this correspondence is not straightforward in practice, since the vector space is necessarily linear, while the potential values refer to points on a three dimensional grid. The mapping between these is arbitrary, but, as we shall see later, can be important in so far as efficient computer implementation is concerned. For now we leave the mapping general, i.e.,

$$\phi(x, y, z) = \Phi_i$$
$$i = g(x, y, z)$$

where $g(x, y, z)$ is an arbitrary, but unique, mapping function.

The elements of $\mathbf{T}$ are then

$$\begin{aligned} T_{ij} = 0 \text{ if } j \neq\ & g(x + 1, y, z), g(x - 1, y, z), \\ & g(x, y - 1, z), g(x, y + 1, z), \\ & g(x, y, z + 1) \text{ or } g(x, y, z - 1) \end{aligned}$$

If we refer to these six nearest neighbors locations as i_k, $k = 1, 6$ respectively, then

$$T_{ii_1} = \varepsilon(x, x + 1, y, z)^* \, d(x, y, z)$$

$$T_{ii_2} = \varepsilon(x - 1, x, y, z)^* \, d(x, y, z)$$

$$T_{ii_3} = \varepsilon(x, y, z + 1, z)^* \, d(x, y, z) \text{ etc.}$$

where

$$\begin{aligned} & d(x, y, z) \\ & \quad = 1/\ (\varepsilon(x, x + 1, y, z) + \varepsilon(x - 1, x, y, z) \\ & \qquad + \varepsilon(x, y, y + 1, z) + \varepsilon(x, y - 1, y, z) \\ & \qquad + \varepsilon(x, y, z, z + 1) + \varepsilon(x, y, z - 1, z) \\ & \qquad + (\kappa_0 h)^2), \end{aligned} \tag{5}$$

and, for instance, $\varepsilon(x, y, z, z + 1)$ is the dielectric constant at the midpoint between (x, y, z) and $(x, y, z + 1)$. (If N is the number of grid points, then there are $3{*}N$ such quantities.) Similarly, the elements of $\mathbf{Q}$ are

$$Q_i = [4\pi q(x, y, z)/h]^* \, d(x, y, z) \tag{6}$$

Finally, solutions of partial differential equations are generally dependant upon boundary conditions. Since these need to be fixed *before* the system is solved, this represents a problem. However, we know that sufficiently far from the dielectric boundary the potentials will not depend on the details of the shape of the molecule. Hence, at such a distance we can use Coulomb's law, or Debye–Huckel theory, i.e., an exponential damping factor to the potential fall off, if there is salt. A typical 'box fill' is therefore about 50%, i.e., the maximum linear dimension of the molecule is half the box width. There exists a method to test if the boundary conditions have been set correctly for a particular box fill, based upon a back calculational technique we have developed, which will be described in a later article, and which supports the choice of box fill of this order as a conservative starting point.

JACOBIAN RELAXATION

The simplest relaxation algorithm, namely Jacobian, consists of multiple applications of eq. (4), i.e., the

n^{th} approximation to Φ, $\Phi(n)$, is defined by

$$\Phi(n) = \mathbf{T}\Phi(n-1) + \mathbf{Q}. \qquad (7)$$

The zeroth approximation is typically $\Phi(0) = \mathbf{0}$, the null vector. Actually, we do not have to worry what initial state we chose, because of the nature of **T**, a matter we shall return to shortly.

To see why this procedure might work, we consider the vector generated by this process, i.e. in terms of **T**, $\Phi(0)$ and **Q** we have

$$\Phi(n) = (\mathbf{T}^{n-1} + \mathbf{T}^{n-2} + \mathbf{T}^{n-3} + \mathbf{T}^{n-4} + \mathbf{T}^{n-5} \ldots + \mathbf{T} + 1)\mathbf{Q} + \mathbf{T}^{n}\Phi(0). \qquad (8)$$

Let us suppose that we use $\Phi(0) = 0$ as our starting state. In the limit of (n $->$ ∞), we note that the right hand side of eq. (8) is equivalent to the expansion of

$$[\mathbf{I}/(\mathbf{I} - \mathbf{T})]\mathbf{Q} \qquad (9)$$

where **I** is the identity matrix.

Now if we were to take eq. (4) and reorganize it

$$\Phi = \mathbf{T}\Phi + \mathbf{Q} = (4)$$

$$(\mathbf{I} - \mathbf{T})\Phi = \mathbf{Q}$$

$$\Phi = [\mathbf{I}/(\mathbf{I} - \mathbf{T})]\mathbf{Q} = (9)$$

Therefore the iterative solution works because it gradually builds up the inverse of (**I** − **T**), term by term.

However, we have been a little cavalier in our mathematics for two reasons, first by assuming that the infinite series, represented by (**I**/(**I** − **T**)), converges, and secondly, since we stated that we do not have to worry what starting state we use, we had better have,

$$\mathbf{T}^{n}\Phi(0) \rightarrow 0 \text{ as } (n \rightarrow \infty)$$

Both these concerns, (and whether the procedure is numerically stable) have the same resolution, namely, we must require that the *eigenvalues* of the eigenvectors of the matrix **T** lie within the range [−1, 1]. To see this, suppose we expand the **Q** vector in terms of these eigenvectors, i.e.,

$$\mathbf{Q} = \sum_{i=1}^{N} a_i\mathbf{F}_i, \qquad (10)$$

where $\mathbf{F}_i$ is the eigenvector of **T** with eigenvalue λ_i, and a_i is its contribution to the decomposition of **Q**, and N is the dimension of the vector **Q**. Then

$$(\mathbf{I}/(\mathbf{I} - \mathbf{T}))\mathbf{Q} = (\mathbf{I}/(\mathbf{I} - \mathbf{T}))\sum_{i=1}^{N} a_i\mathbf{F}_i \qquad (11)$$

$$= \sum_{i=1}^{N} a_i(\mathbf{I}/(\mathbf{I} - \mathbf{T}))\mathbf{F}_i$$

$$= \sum_{j=1}^{N} a_i(1/(1 - \lambda_i))\mathbf{F}_i \qquad (12)$$

If $(1 > \lambda_i > -1)$, then the right-hand side of the above equation is well defined. Similarly, if we expand $\Phi(0)$ in terms of the same eigenfunctions, then,

$$\Phi(0) = \sum_{j=1}^{N} b_i\mathbf{F}_i \qquad (13)$$

Then

$$\mathbf{T}^n\Phi(0) = (\mathbf{T}^n)\sum_{i=1}^{N} b_i\mathbf{F}_i$$

$$= \sum_{i=1}^{N} b_i(\mathbf{T}^n)\mathbf{F}_i$$

$$= \sum_{i=1}^{N} b_i(\lambda_i^n)\,\mathbf{F}_i \qquad (14)$$

Clearly, if $(1 > \lambda_i > -1)$, then

$$\mathbf{T}^n\Phi(0) \;->\; 0 \text{ as } n \;->\; \infty$$

This eigenvalue range, or the *spectrum*, of **T**, has consequence beyond whether the Jacobian method will converge or not. It also determines the rate at which the method will converge. For the Poisson–Boltzmann equation we can show that indeed $(1 > \lambda_i > -1)$ (we would use the boundary conditions plus the weak row sum criterion[4]). We can also show that if (λ_k) is an eigenvalue, so is $(-\lambda_k)$ (Theorem 8.3.9[4]). Therefore the largest absolute eigenvalue is equal to the most positive eigenvalue. (We shall be much concerned with the maximum absolute eigenvalue, and we shall do so in the spirit of it being the "most positive," since there is no contradiction inherent). Since the ordering of the eigenvalues is arbitrary, let it be by eigenvalue magnitude, so that $\lambda_{\max} = \lambda_N$. This is also referred to as the *spectral radius* of **T**.

Now, consider the quantity

$$\Delta\Phi_n = \{\Phi - \Phi(n)\} \qquad (15)$$

i.e., the difference between the nth approximation and the exact solution. Let us suppose we start with $\Phi(0) = \mathbf{0}$, the zero vector. If we expand in the eigenfunctions of **T**, we have

$$(\Delta\Phi)_n = \sum_{i=1}^{N} a_i\left\{(1/(1-\lambda_i)) - \sum_{j=0}^{n-1}(\lambda_i^j)\right\}\mathbf{F}_i$$

$$= \sum_{i=1}^{N} a_i(\lambda_i^n)/(1 - \lambda_i)\mathbf{F}_i \qquad (16)$$

Since $(1 > |\lambda_i|)$ we know that $(\Delta\Phi)_n$ goes to zero as n becomes large. However, it is clear that in this limit the sum will be dominated by the terms with the largest eigenvalues. Hence the rate of convergence to the exact solution is determined by the spectral radius. In fact, in the limit as $(n -> \infty)$, we have

$$|(\Delta\Phi)_n| \approx (\lambda_N)^n.$$

Suppose we decide the process has converged when $|(\Delta\Phi)_n|$ is less than some number, say 10^{-m}, we have

$$|(\Delta\Phi)_n| \approx (\lambda_N)^n \approx 10^{-m}$$

$$n \approx -\text{mlog}(10)/[\ln(\lambda_N)]$$

If we set $(\lambda_N) = (1 - \varepsilon)$, then, since ε is small, we have

$$n \approx (\text{mlog}(10))/\varepsilon, \tag{17}$$

i.e., that the number of iterations to reach our convergence criteria is roughly dependent upon the reciprocal of ε, the difference between the maximum eigenvalue and unity.

One can show that ε is approximately $(1/L^2)$ (see below), hence for Jacobian relaxation the number of iterations required is proportional to L^2. This makes the penalty against increasing the grid size considerable, i.e., if the linear grid size is doubled, not only are there eight times as many grid points to relax, but it will take four times as long to converge!

The only assumption in the derivation of this convergence rate is that a_N is not zero. Since we expect the potential to change rapidly near charged grid points there must be high frequency eigenvalues present in the spectral decomposition, and so this is not an unreasonable assumption.

It should be noted that the charge distribution does not affect the convergence rate. Neither do the values assigned to box boundary points. To see this we consider the role of these points in the relaxation process. Their only impact is to be a part of the updating of those points in the first "inner shell" of the box. Hence they act just like source terms, i.e., charges, for these points. In fact, they could be set to zero, and replaced by a set of charges, of appropriate magnitude, at those inner points. Since we know a charge distribution does not affect the convergence rate, neither can the boundary values.

GAUSS–SEIDEL RELAXATION

Gauss–Seidel differs from Jacobian relaxation by using updated potential values as soon as they are available. This is different from Jacobian since a glance back at eq. (7) shows that $\Phi(n)$ depends only upon $\Phi(n - 1)$ i.e., quantities from the previous iteration. Note that we are imagining matrix multiplication as a sequential operation, where one entry at a time in $\Phi(n)$ is updated from $\Phi(n - 1)$. Using Gauss–Seidel the values doing the updating may already be part of $\Phi(n)$. The order in which grid points are mapped to the vector Φ will determine if updating values (grid potentials) are "new" or not, hence there is no unique way to implement Gauss–Seidel. However the spectral radius of the method is independent of this mapping (reference 4), hence the *intrinsic* rate of convergence does not depend upon the implementation. This does not mean that some way will not be computationally more efficient, i.e., faster. We will consider a particularly apt ordering shortly.

The particular matrix structure (i.e., where the zeros lie) of **T** allows us to say more about the spectral radius of Gauss–Seidel for the PB problem. **T** is said to be "consistently ordered" (reference 4, pg. 549), and therefore λ_N(Gauss–Seidel) = λ_N(Jacobian).[2] Remembering the relationship between the spectral radius and the rate of convergence, this means that Gauss–Seidel for the Poisson–Boltzmann equation converges twice as fast as Jacobian, i.e., it takes half as many iterations.

This surprising result is easier to understand if we consider a particular mapping of grid-to-vector known as the "checkerboard" ordering. Consider that each grid point can be assigned as odd or even, by the sum of its grid coordinates, e.g., the grid point (34,45,21) would be (34 + 45 + 21 = 100) an even point. Next consider that the six nearest neighbors to any point must be of opposite nature, since their grid coordinates differ by just one. Therefore every odd point is surrounded by even points and vice versa. Since the updating of each point is local, i.e., only by these nearest neighbor points, we see that even is only updated by odd, and odd only be even.

Suppose we order our points (choose a mapping $g(x,y,z)$) such that the even points occupy the first half of the vector, the odd the second half. Then the matrix **T** will have the following structure

$$\begin{bmatrix} 0 & T_1 \\ T_2 & 0 \end{bmatrix},$$

where T_1 is the submatrix which updates even entries with odd entries, and T_2 the odd with the even. With Gauss–Seidel we want to update with the latest values. In this scheme we see that the even values would be updated as with the Jacobian procedure, but when we come to the odd values of the methods diverge, for Gauss–Seidel wants to use the most recent even values, i.e., those just calculated from the old odd values. Hence the calculation of the odd values skips a step, since, in effect it does exactly what the Jacobian procedure would do in the next iteration cycle. Then, in the second iteration the even values are updated with odd values, which, as we have seen are one iteration ahead of the Jacobian procedure, and hence gain a step, while the odd values will skip one more iteration and become identical to those for the fourth Jacobian iteration.

In general, we have,

Φ(even, Gauss-Seidel, nth iteration)
= Φ(even, Jacobian, ($2n$-1)th iteration),
Φ(odd, Gauss-Seidel, nth iteration)
= Φ(odd, Jacobian, ($2n$)th iteration).

Therefore, we have that Gauss-Seidel converges twice as fast as Jacobian relaxation.

SUCCESSIVE OVER-RELAXATION

SOR can provide a convenient means to speed up both the Jacobian and Gauss-Seidel methods of solving the Poisson–Boltzmann equation. Let us examine the Jacobian case. Equation (2) is altered in the following, seemingly trivial way

$$\begin{aligned} \Phi &= \mathbf{T}\Phi + \mathbf{Q} \qquad (4)\\ \Phi - \alpha\Phi &= \mathbf{T}\Phi - \alpha\Phi + \mathbf{Q}\\ (1-\alpha)\Phi &= (\mathbf{T} - \alpha^*\mathbf{I})\Phi + \mathbf{Q}\\ \Phi &= ([\mathbf{T}/(1-\alpha)] - [\alpha/(1-\alpha)]^*\mathbf{I})\Phi\\ &\quad + \mathbf{Q}/(1-\alpha)\\ \Phi &= \omega\mathbf{T}\Phi + (1-\omega)\Phi + \omega\mathbf{Q} \qquad (18)\end{aligned}$$

where, **I** is the identity matrix and

$$\omega = 1/(1-\alpha)$$

The parameter ω is referred to as the relaxation parameter. Clearly for $\omega = 1$ we restore the original equations. If $\omega < 1$ we talk of under-relaxation, and this can be important for some systems which will not converge under normal Jacobian relaxation. If $\omega > 1$, we have over-relaxation, with which we will be more concerned.

We note that eq. (18) is similar in form to eq. (4), in that we have

$$\begin{aligned}\mathbf{T}' &-> \omega\mathbf{T} + \mathbf{I}^*(1-\omega)\\ \mathbf{Q}' &-> \omega\mathbf{Q}\end{aligned}$$

But we know that only $\mathbf{T}'$ determines the convergence, via its spectral radius, which has now become a function of ω.

For Jacobian relaxation the answer is rather dull. For this set of equations the minimum spectral radius will occur at $\omega = 1$, i.e., no relaxation. Hence introduction of a relaxation parameter does not help us.

However, the method is much more powerful when applied to Gauss–Seidel. In fact, when mention is made of successive over-relaxation, it is usually as an extension of Gauss–Seidel. We will not derive the equations describing the mapping of the spectral radius with respect to ω. The interested reader is directed to Stoer and Bulirsch (reference 4, pages 545 and following). We shall merely quote the results, namely that, the optimal value for ω occurs when,

$$\omega = 2/(1 + \sqrt{1 - \lambda_N}) \qquad (19)$$

where the minimum spectral radius is

$$\lambda_N = \left(\lambda_N^{gs} \Big/ \left[1 + \sqrt{1 - \lambda_N^{gs}}\right]^2\right) \qquad (20)$$

where λ_N^{gs} refers to the spectral radius of Gauss-Seidel. To see why this is much better, let us again make the substitution, $\varepsilon = 1 - \omega_N^{gs}$. Then the spectral radius becomes

$$\lambda_N = (1-\varepsilon)/(1+\sqrt{\varepsilon})^2 \approx 1 + 2\varepsilon - 2\sqrt{\varepsilon} \qquad (21)$$

Now, since ε is a small number, $\sqrt{\varepsilon}$ is substantially bigger. If we were to assume $\varepsilon = 0.01$, then the spectral radius goes from 0.99 to about 0.82. This corresponds to an increase in rate of convergence of about 18-fold over Gauss-Seidel (36-fold over Jacobian)!

This improvement clearly comes about because of the appearance of a square root in the calculation of the spectral radius. Since the ratio of $2\sqrt{\varepsilon}$ to ε roughly determines the speed up at optimum SOR, we see that the improvement is greater the smaller ε, i.e., the slower the original ($\omega = 1$) method. If ε were not small, the gain would be minimal. The reason for the occurrence of the square root can be traced to the fact that Gauss–Seidel splits the relaxation process into two parts, conveniently odd and even in our checkerboard mapping. Finally, we note that since ε is roughly $(1/L^2)$ (as we alluded to previously, and will show below), then $\sqrt{\varepsilon}$ is about $(1/L)$. Therefore the number of iterations necessary for convergence with SOR, at the optimum parameterization, scales linearly with the grid length L, compared to quadratically as with Jacobian and Gauss–Seidel. This means that the larger the grid size, the better the *relative* improvement offered by optimal SOR.

Of course we need to remember that this impressive gain in convergence rate occurs at a particular ω, which is determined by λ_N, which in general we do not know. Unfortunately the spectral radius is fairly sensitive to ω, hence the improvement can be degraded, or even lost, by a poor choice of λ_N. One way to ensure a good choice of ω is to solve the system of equations with several different choices of ω and pick the best! This, then is the catch-22 of SOR. To find a good relaxation parameter one needs to have already solved the problem! In fact there exist easier ways to estimate λ_N and we shall now present one such, which we believe novel and of some utility.

OPTIMAL SUCCESSIVE OVER-RELAXATION

Our contribution here is to provide a simple method to calculate λ_N from just a handful of iterations, rather than thousands. It is based on the observation that finding λ_N is just an eigenvalue problem, for which a vast literature already exists.

To illustrate this, consider first that we know the eigenvalue spectrum of **T** lies completely between plus and minus one. Therefore the spectrum of $(1 - \mathbf{T})$ lies between zero and two. The problem of finding the maximum eigenvalue of **T** is identical to finding the minimum eigenvalue of $(1 - \mathbf{T})$. Minimum eigenvalues are of much interest in quantum mechanics, since they represent ground state energy of the system under study. Hence it is reasonable to expect to be able to borrow from this field.

One approach which is particularly convenient for our investigation is a recent formulation by Cioslowski, namely the Connected-Moments Expansion (CMX).[6] In this approach one takes an approximation to the ground state wavefunction $|\phi>$, and calculates the moments of the Hamiltonian, i.e.,

$$n^{\text{th}} \text{ Moment} = M_n = <\phi|H^n|\phi> \qquad (22)$$

where H is the Hamiltonian matrix. From here one calculates the connected moments, which are defined by

$$I_k = M_k - \sum_{i=0}^{k-2} \left(\frac{(k-1)!}{i!(k-1-i)!} \right) I_{i+1} M_{k-i-1} \qquad (23)$$

Hence the kth connected moment is calculated from M_k, from lower order moments and connected moments. The connected moments then deliver the ground state energy via

$$E_0 = I_1 - \frac{I_2^2}{I_3} - \left(\frac{1}{I_3}\right) \frac{(I_4 I_2 - I_3^2)^2}{(I_5 I_3 - I_4^2)} - \dots \qquad (24)$$

where higher order terms are produced by the rule of replacing each I_k in the previous term by $(I_{k+2} I_k - I_{k+1})$ and dividing the total term by I_3. The series is therefore very easy to generate, and relies merely upon knowing the first $(2n - 1)$ moments for an nth-order CMX. The approach is nonperturbational, and relies only upon the eigenvalues of H being not wholly imaginary.

The application of CMX to our problem is fairly straightforward. We need to know an approximation to the lowest eigenstate of $(1 - \mathbf{T})$, i.e., the largest eigenstate of $\mathbf{T}$. Suppose the problem we were looking at was one for which there was no dielectric boundary, and no salt. The Poisson–Boltzmann equation then becomes the Laplace equation, and the answer to our question is well known to any student of elementary quantum mechanics. The eigenfunctions of the Laplace equation in a box, with boundary values clamped to zero, are the same as for a free particle in a (three-dimensional) box (the Laplace operator here corresponds to the momentum operator).

Normally we are not interested in the eigenvectors and eigenvalues of the differential operator as such, but of the relaxation matrix $\mathbf{T}$, as defined previously. However since in this case since the diagonal elements are all equal, the eigenfunctions for each are identical. For the discretized version of the same problem, the eigenfunctions, for a cube with $L \times L \times L$ grid points are,

$$2\sqrt{2}\sin(n\pi(x-1)/(L-1))\sin(m\pi(y-1) \div (L-1))\sin(l\pi(z-1)/(L-1)) \qquad (25)$$

where n, m, and l determine the particular eigenfunction (and run from 1 to $L - 2$), and x, y, and z are the grid indices. The eigenvalue corresponding to this eigenfunction, of $\mathbf{T}$, is

$$(1/3)[\cos(n\pi/(L-1)) + \cos(m\pi/(L-1)) + \cos(l\pi/(L-1))] \qquad (26)$$

The maximum eigenvalue is therefore, $n = m = l = 1$, i.e.

$$(1/3)[\cos(\pi/(L-1)) + \cos(\pi/(L-1)) + \cos(\pi/(L-1))] \qquad (27)$$

For large L, this is approximately equal to,

$$1 - \pi^2/2(L-1)^2 \qquad (28)$$

The eigenfunction corresponding to this eigenvalue is

$$2\sqrt{2}\sin(\pi(x-1)/(L-1))\sin(\pi(y-1)/(L-1)) \times \sin(\pi(z-1)/(L-1)) \qquad (29)$$

We note that this is the basis for assuming ε is of order $(1/L^2)$, i.e., that Jacobian and Gauss-Seidel need of order L^2 iterations to come to convergence.

Armed with the exact expression for the "ground state wavefunction" in this case, we proceed by assuming that, in the more general case with salt and dielectric boundaries, this will still be a good approximation. One of the nice things about using CMX is that since the method is not perturbational, $|\phi>$ can be quite a crude estimate to the exact wavefunction and the method will still converge.

Given that the above assumption is a good one, we proceed to calculate the required lowest eigenstate by calculating the moments of $\mathbf{T}$. This, of course, could not be easier since, if we set $\mathbf{Q} = 0$ in eq. (8) we get

$$\Phi(n) = \mathbf{T}^n \Phi(0)$$

i.e., our program already calculates moments when it solves the PB via Jacobian or Gauss–Seidel algorithms. Hence the prescription is to set the initial vector equal to the approximation highest eigenvalue eigenstate, set all charges equal to zero, and calculate a few moments.

The CMX method for our problem converges very rapidly, e.g., by the third term. In fact, beyond this double precision is really required so that the delicate cancelling, which goes into ensuring high order terms are small, is not affected by round-off error. However, since including just two terms (or even just the first) gives very good agreement with other, more computational intensive, methods, the calculations may safely be done in single precision.

Hence finding λ_N requires about as much computer time as it takes to do three iterations, or cycles, of Jacobian or Gauss–Seidel. This is a marked improvement over such strategies as a fixed relaxation parameter, calculating λ_N from the convergence properties, or finding the best relaxation parameter

Table I. Number of iterations necessary for convergence.

Box fill/Salt Conc. (%)	0.1 Molar salt	No Salt
80	112	168
40	76	184

by doing multiple runs. In Table I we show the results of using this method to calculate the potentials in and around a lysozyme molecule, i.e., the number of iterations needed to come to complete convergence, for a variety of conditions. The criteria for complete convergence here is for the root mean square difference between potential maps of successive iterations to plateau, which it typically does at around a value of $2*10^{-6}$. (Note that this criteria is needlessly strict for most applications).

It should be mentioned that the above method is easily generalizable to orthorhombic shaped boxes. It is also straightforward to adapt it to the case of periodic boundary conditions, which are often used in dealing with extended structures such as DNA.

Finally we should acknowledge that the optimum relaxation parameter is only optimal for the limiting rate of convergence, i.e., the rate of convergence when one is already close to the solution. It is not necessarily the best parameter for the most rapid *initial* convergence. For this reason, some textbooks suggest the use of *Chebyshev acceleration*,[5] in which the relaxation parameter for the first iteration is unity, and, with each proceeding iteration, approaches the value suggested by λ_N in a predefined manner. Although such methods can reduce the number of necessary iterations, we have not found the enhancement to be significant, and we shall leave their condsideration to a later report.

CONTIGUOUS MEMORY MAPPING

The previously mentioned checkerboard mapping is useful for more than just illustrating the convergence of the Gauss–Seidel method. Modern computers can handle matrix multiplications very rapidly because of vector processing. Here a pipeline of numbers is constructed for a certain set of operations. The only caveat is that the operation on a particular number not involve numbers already in the pipeline. Clearly Gauss–Seidel in its most general form as outlined above can fall foul of this one condition, since the grid potential at a particular point depends upon that of its neighbors, which may themselves be in the process of being updated, i.e., be already in the pipeline. However, if the checkerboard ordering is employed, then we can effectively cut up the iteration into two half-cycles, namely calculating the even from odd as the first half, followed by the odd from even, *once* the first half is completed. This way there can never be any pipeline contention.

However, we note that the implementation of the checkerboard ordering as suggested by some texts[5] is hopelessly slow to run, even on scalar machines, but especially on vector processors. The reason for this is that the separation into odd and even cycles is done within the inner loop of the algorithm, usually with an "IF" statement, i.e., a logical operation. In general this will slow down both scalar and vector machine implementations.

The best way to efficiently code the ordering is to map the odd and even points separately into *contiguous* memory, i.e., as suggested earlier in our mathematical *gedanken*. This leads to a more complex coding of the relaxation algorithm, but has the advantage of avoiding any branching in the inner loop, and has the potential for very long vector lengths. The latter can be equally important for vector machines, since there is overhead associated with each "pipeline" set up.

The potential vector length also depends upon how one chooses to treat the boundary points of the box. One wants to include them in the numbering scheme so that points just interior to the box can be updated as any other within the box. However, if they are in the linear numbering scheme, and the inner loop runs over all the box, then these boundary points will "relax," i.e., be (erroneously) updated. The solution of this problem is either to skip these points, or to reset them to their correct values before these incorrect values can propagate. The former path has the drawback of restricting the maximum vector length to half a box side, in our odd/even arrangement. The latter requires additional computing. For vector machines we have found the latter more efficient, restoring the odd boundary points after the odd cycle, but before the even cycle, and vice versa for the even boundary points.

We would also mention that if the algorithm is coded as suggested, adaptation for parallel processing is straightforward, since, within each half cycle, calculated values are all independent, and there is no "write" contention. On a Convex C220, with two processors, 95% simultaneous utilization of each was observed.

STRIPPING

Consider the inner loop of, say, the Jacobian method. By "inner loop" we mean the section of code which corresponds to eq. (3) i.e., the finite difference version of the Poisson-Boltzmann equation. Typical implementations look like,

$$\phi_i = Q_i + \sum_{k=1,6} T_{i,ik}\phi_{i_k} \qquad (30)$$

where the quantities are as described earlier.

If we now analyze the operations involved here, we see that there are six multiplications, six additions, 13 loads, and one store. We can also see that the storage required to contain all quantities in main memory is 8*N. Since these quantities are single precision, each requires four bytes, hence, if the grid size is 65*65*65, we need about 8 megabytes.

One common way to reduce the memory requirements is to note that the T quantities are calculated from 3*N dielectric values and one denominator term (see definitions in the finite difference section). Hence, if instead of using T, we use $d(x, y, z)$ and $\varepsilon(x, y, z)$, then we cut the storage to 6*N. However, we have increased the number of loads by one and added one more multiplication to the inner loop.

Suppose we consider eq. (30) for a point which is not near the dielectric boundary or a charge. A point is near the dielectric boundary if not all of its six nearest neighbor midpoints have been assigned the same dielectric value. Similarly, a point is near a charge if it has been assigned some charge via the linear interpolation scheme refered to earlier. A point which is near neither has a particularly simple relaxation, namely all entries in **T** are equal to (1/6) and Q_i is zero. The potential is merely the average of its six nearest neighbors. This is exactly what we would get if we were solving the Laplace equation instead of the Poisson–Boltzmann equation, hence we shall refer to such points are Laplacian in nature.

The question arises as to how many points on the grid are Laplacian rather than "Poisson–Boltzmannian." This will depend of course upon the particular molecule, box fill, charge set employed and so on. For example, a molecule filling 50% of a 65 cubed box, may have about 6000 points near the dielectric boundary, and a 1000 or so grid points assigned charge. This would mean that 97% of the points are Laplacian!

If we consider the Laplace equation inner loop we see that

$$\phi_i = (1/6)^* \left(\sum_{k=1,6} \phi_{ik} \right) \tag{31}$$

Counting operations, we see that there are only five additions, one multiplication, six loads, and one store. Hence the inner loop of the Laplace equation requires less than half the operations of that of the Poisson Boltzmann equation. We also note that the storage required is only N, not eight or six times N as before.

Hence the concept of stripping is to treat every point as Laplacian, and then correct those for which this is incorrect in a separate algorithm (i.e., they have been *stripped* from the main algorithm). If the number of non-Laplacian points is small enough, relative to the total number of points, then we would expect a significantly faster algorithm, as well are greatly reducing the main memory required. This is an unusual situation since algorithms for solving elliptic equations typically must sacrifice either storage or speed.

The condition that "most" points be Laplacian, will depend upon the machine used. Because the position of non-Laplacian points must be specified explicitly, the loads in the secondary algorithms are necessarily more complicated. This will be especially relevant to vector machines which are designed to work optimally on sequential memory locations. As a rough guide, as long as the number of points near the dielectric boundary is less than 10% of the total, their impact on performance is small. The points near a charge have a much smaller effect (only two loads and one addition, compared to 13 loads, six additions, and six multiplications for a boundary point), and restrictions on both are much looser on scalar machines.

The gain in speed will, of course, also be machine dependent. For example, many machines can chain multiplication and addition operations together doing two for the price of one. Stripping "unbalances" the number of additions and multiplications, i.e., one goes from six of each, to one multiplication and five additions, and hence can negate gains from use of this architecture. However, working on a vector machine which employs such mechanisms, for instance a Convex C1, speed-up factors of between two and three were achieved.

If salt is present the situation is but slightly more complicated. Discounting non-Laplacian points (for which the secondary algorithms are no more complicated), we see that Laplacian points now fall into two classes, those in salt, and those not. Those not in salt will have the same multiplier, i.e., (1/6), where as those in salt will have a different, but constant multiplier, dependent on the salt concentration. While it might be possible to code so that the multiplier is determined via a bit-mask, in general this implies that a new array is required. Hence the memory required increases to 2*N, and the number of loads from six to seven, in the inner loop of our modified Laplace equation. This reduces performance by about 15%.

We have described stripping as applied to the Jacobian method, however it can be transferred transparently to both Gauss–Seidel, SOR and OSOR, the latter leading to Stripped Optimal Successive Over Relaxation, or SOSOR. The code for SOSOR has been incorporated into the program DelPhi, available for distribution from the authors. In Table II we show the execution times for DelPhi/SOSOR for various machines, again for lysozyme, with no salt and a box fill of 65%, and convergence to three decimal places at charged grid points. It is clear that even on inexpensive workstations, such as the personal IRIS, one is now able to perform a calculation in an almost "interactive" time span. In effect this means that anyone with minimal computing facilities can use

Table II. Convergence times on various computers.[a]

Time/Machine	Convex C2	IRIS 4D/220	Peronal Iris
Set up	5.7	9.8	14.1
Iteration	12.6	32.65	162.0
Total	18.3	42.45	178.1

[a]All times in seconds, 131 iterations performed. Processor time for the C2 during the iteration subroutine is twice wall clock, i.e., real time is 6.3 seconds, and reflects the near perfect parallelism of the convex adapted code.

the PB equation to model electrostatics in biological macromolecules.

APPLICATION TO THE NONLINEAR POISSON BOLTZMANN EQUATION

We remind the reader that the full Poisson–Boltzmann equation is

$$\nabla \cdot [\varepsilon(\mathbf{x})\nabla\phi(\mathbf{x})] - \kappa(\mathbf{x})^2\sinh(\phi(\mathbf{x})) = -4\pi\rho(\mathbf{x}) \quad (1)$$

where all symbols are as previously defined.

The $\sinh(\phi)$ term, which makes the equation nonlinear, arises due to the fact that salt atoms will tend to pile up in regions of large potential, and that they will do so with a Boltzmann-like factor, i.e., and exponential factor. If we assume the salt is a (1:1) charge pair, then the two exponentials (one for each charge species) form the hyperbolic sine function. If we were to retain only the first term in the series expansion of the $\sinh(\phi)$, i.e., (ϕ) we would have the linearized Poisson Boltzmann.

The relaxation processes defined previously are only strictly valid for linear equations. However, it is interesting to see if they may be applied to the nonlinear equation, since for highly charged molecules (such as DNA), one might expect to see significant deviations from the linear regime.

One way in which the full equation might be manipulated has been described earlier by Jayaram et al.[7] Essentially the $\kappa^2\sinh(\phi)$ term is rewritten as ϕ $*[\kappa^2 *\sinh(\phi)/\phi]$, i.e., the nonlinear term is treated as a linear term, multiplied by a "dressed" Debye–Huckel term. With this the iteration equation looks like

$$\phi_1 = \left[\frac{\left(\sum_{i=1}^{6} \varepsilon_i\phi_1\right) + 4\pi q_0/h}{\left(\sum_{i=1}^{6} \varepsilon_i\right) + (\kappa_0 h)^2(\sinh(\phi_1))/\phi_0}\right] \quad (32)$$

There is no guarantee this procedure will converge. However, inclusion of an *under-relaxation* parameter (i.e., $\omega = 0.6$), has shown to give stable convergence under most circumstances.

We have developed a slightly different approach which, while not as robust as Jayaram et al.,[7] gives rapid convergence to the same solution, by incorporating the concepts developed for SOSOR.

Suppose we separate out the $\kappa^2\sinh(\phi)$ term into the difference between the linear and non-linear terms, i.e.,

$$\kappa^2\sinh(\phi) = \kappa^2\phi + (\kappa^2\sinh(\phi) - \kappa^2\phi)$$

If we now place the purely non-linear term on the right-hand side of the full Poisson–Boltzmann equation,

$$\nabla \cdot [\varepsilon(\mathbf{x})\nabla\phi(\mathbf{x})] - \kappa(\mathbf{x})^2\phi(\mathbf{x}) = -4\pi\rho(\mathbf{x}) + \kappa(\mathbf{x})^2[\sinh(\phi(\mathbf{x})) - \phi(\mathbf{x})]$$

We see that the nonlinear terms act as an "excess charge." Now we know that the charge assigned to a system does not alter the rate of convergence. Hence, if we fix the nonlinear contribution to the right-hand side charge for a particular ϕ, then we know we can rapidly converge to the solution for this charge distribution. However, the excess charge distribution, if the recalculated, would have changed. This would mean having to solve for the new charge distribution. This procedure might or might not converge, and it would be slow.

A more efficient application of the above ideas is an inexact science. However, we have had some success based upon the following two approaches, applied after having calculated the solution to the linearized PB. Firstly, we update the excess charge every two to four full cycles. To do so less frequently is inefficient in terms of the total number of cycles to convergence. To do so more often is wasteful in terms of the number of operations necessary to recalculate the excess charge. Secondly, we multiply the excess charge term by a strength parameter χ, which we set initially low, e.g., $\chi = 0.05$. Then we slowly increase χ at each recalculation of the excess charge calculation, until it equals unity. The rational behind this is that we want to "ease" the potential distribution into the correct solution. To do so gradually reduces the "overshoot" one sometimes sees, and tends to improve the eventual convergence plateau which is common from this procedure.

It is possible to see the method completely fail, i.e., the convergence suddenly become much worse, rather than better. Examination of these cases suggests that trouble starts when the potential at a point in salt passes a certain threshold, typically of 7 or 8 kt. There are more parameters involved than this alone, such as the grid spacing for example. We also found that damping the exponential term by, for instance, using only the first two terms of the nonlinear contribution, stabilizes the convergence considerably. The reason for this behavior seems to be that the differences in potential between grid points in the high potential regions do not settle down, because the hyperbolic term varies so rapidly with just slight variation in potential between sites.

However, despite these occasional failures, the method has the advantages of being quite rapid. Typically it wil take only about twice as much computing to converge once the linear equation is solved. This compares favorably with previous methods, which, should it fail, one has to fall back upon.

OTHER METHODS

Although we have detailed three commonly used algorithms for solving the PB equation, there are, of course, many more, as a visit to any textbook on numerical methods will confirm. We will limit ourselves to a consideration of two such, one newly proposed in the literature, and the other under active research in our lab.

Recently Davis and McCammon have described implementing the Incomplete Cholesky Conjugate Gradient method (ICCG). This is still a grid based method, but reaches the solution by recasting the matrix equation into a minimization problem, to which steepest descent is applied. They obtained fast convergence, claiming an improvement over OSOR of a factor of two (the relaxation parameter being precalculated). However, it is hard to do exact comparisons since the basis of comparison was computational, not theoretical, and, as we have seen, the coding of relaxational techniques can be critical. Perhaps the best parameter for comparison is an estimate of the number of floating point operations required to reach convergence. For a 65-cubed grid, SOSOR typically takes (2-3)*10^8 such operations.

While it is clear that stripping could be applied to their method, it is unlikely to be as effective, since sparse matrix multiplication is only one part of the iteration cycle. Furthermore, it should be noted that their method requires an increase in memory over relaxational techniques, namely 9^*N values over 6^*N. This is then almost an order of magnitude increase over a minimally coded SOSOR. Nevertheless, ICCG is an important attempt to step beyond traditional methods in the field.

Another potentially useful method, which is under development in our lab, is the Alternating Direction Implicit (ADI) algorithm. Here the solution is reached via a mix of explicit matrix multiplication, as in SOR, using part of the **T** matrix, and an implicit step, involving the inversion of rest. The determination of which part of the matrix is which, is typically along physical lines, i.e., explicit relaxation along the y and z directions, and implicit inversion in the x direction. This then constitutes a third of the total iteration, as the second part would have x and z explicit, y implicit, and the third part x and y explicit, z implicit. The reason it is possible to easily invert one third of the matrix is that each part, on its own, is tridiagonal, and hence easily solvable (e.g., reference 5).

Although ADI contain considerably more operations per iteration (at least 30), theory predicts suitably chosen relaxation parameters should lead to a threefold computationally faster solution for a L = 65 grid size. For larger grids the news is better still, as the convergence dependence is a logarithmic function of L, compared with linear for OSOR, and quadratic for Jacobian and GS. Finally, the optimum relaxation paramters for ADI also depend on the spectral radius. Hence we may be able to apply the techniques developed above, although the sensitivity to the parameter used may be less than with SOR.

One further reason for optimism over ADI is that it overcomes one limitation of locally relaxing methods. By this we mean that each cycle in Jacobian, GS, or SOR, potentials are updated by neighboring, or local, values. Therefore a source at one grid point only affects the potential m grid points away after m Jacobian cycles, or $m/2$ (GS/SOR) cycles. However, in ADI, one performs an inversion in one direction every one third cycle. Obtaining this implicit partial solution means that all potentials affect all other potentials in that direction. Hence, over a full cycle, every point on the grid can affect every other point. The updating, in a sense, has nonlocal character.

It would be imprudent to attempt to comment upon all the methods which might yet be brought to bear upon the Poisson–Boltzmann equation, since there are so many. However, as the merit of modeling macromolecule electrostatics by solving the PB increases, the usefulness of faster algorithms grows likewise. In which case a thorough investigation of alternative methods becomes more than worthwhile.

Finally, we should sound a note of caution. Table II clearly demonstrates that the time to set up the arrays necessary to solve the BP is no longer negligible in comparison with the time required to attain the solution once so set. These set up times are the result of no little effort in code optimization in our lab, the details of which will appear elsewhere. As a result, our opinion is that, although algorithms will continue to improve, the description of the molecular surface may become the rate limiting step in the solution of the Poisson–Boltzmann equation.

CONCLUSIONS

We have introduced three major improvements in our code for solving the Poisson–Boltzmann equation, contiguous memory allocation, stripping, and a method to rapidly find the optimum relaxation parameter for SOR. This has led to an efficient algorithm (dubbed SOSOR), both in terms of speed of convergence, and required computer memory, which has been incorporated into the electrostatics program DelPhi. With these improvements, and more likely in the future, the ability to solve the Poisson–Boltzmann equation on a reasonably dense grid, has become possible on relatively inexpensive machines, to which most researchers have access, and in a time-frame which allows the process to be almost interactive. This, in turn, should lead to a more wide-spread appreciation of the uses and limits of this equation in describing electrostatics at the molecular level.

This work was supported, in part, by NIH grant GM 30518.

References

1. K. Sharp, B. Honig, *Ann. Rev. Biophys. Biophys. Chem.*, **19,** 301, (1990).
2. M.E. Davis and J.A. McCammon, *J. Comp. Chem.*, **10,** 387, (1989).
3. I. Klapper, R. Hagstrom, R. Fine, K. Sharp, and B. Honig, *Proteins,* **1,** 47 (1986).
4. J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, Springer-Verlag, New York, 1980.
5. Press, Flannery, Teukolsky, and Vetterling, *Numerical Recipies*, Cambridge University Press, 1986.
6. J. Cioslowski, *Phys. Rev. Letters.*, **58,** (2) (1986).
7. B. Jayaram, K.A. Sharp, B. Honig, *Biopolymers*, **28,** 975, (1989).